ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ
ӘЛ-ФАРАБИ АТЫНДАҒЫ ҚАЗАҚ ҰЛТТЫҚ УНИВЕРСИТЕТІ

Е. П. Мақашев

# АЛГОРИТМДЕР ЖӘНЕ ПРОГРАММАЛАУ

Оқу құралы

Алматы, 2014

ӘОЖ
КБЖ
М

*Механика-математика факультетінің Ғалымдар кеңесімен ұсынылды*

*Пікір беруші*:
***Иманғалиев Е.И.**, физика-математика ғылымдарының кандидаты, Дифференциалдық теңдеулер және басқару теориясы кафедрасының доценті, әл-Фараби атындағы ҚазҰУ*

Е

Мақашев Е.П. Алгоритмдер және программалау: оқу құралы. – Алматы: Қазақ университеті, 2014. – 123 с.

ISBN

Оқу құралында алгоритмдер мен деректердің құрылымы, сұрыптау, іздеу және рекурсивті алгоритмдер, Пост және Тьюринг машинасы, алгоритмдерді талдау негіздері, қасиеттерін, түрлері мен берілу тәсілдерін үйрету арқылы логикалық ойлауын, алгоритмдік білімін дамыту C/C++ программалау тілінде программалар арқылы келтірілген және кітаптың соңында қарастырылған тақырыптарға байланысты тест берілген.

Оқу құралы IT бағытындағы мамандарға, оқытушыларға, «Информатика», «Есептеу техникасы және программалық жабдық», «Ақпараттық технологиялар», «Математикалық және компьютерлік модельдеу» мамандығының студенттеріне, магистранттарына ұсынылады.

ӘОЖ
КБЖ
М

**МАЗМҰНЫ**

# КІРІСПЕ

Қазіргі таңда алгоритм ұғымы тек математикалық есептерге ғана емес, сонымен қатар басқада бағыттарда қолданылады. Әрбір компьютер алдын-ала құрылған алгоритм бойынша, яғни программа бойынша жұмыс істейді.

Алгоритм ұғымының анықтамасы өте көп. Берілген есепті шешу үшін қандай да бір программалау тілінде программа жазғыңыз келсе, онда алдымен есепті шешудің алгоритмін құруыңыз керек.

Алгоритм - берілген есептің шығару жолын реттелген амалдар тізбегі түріне келтіру [1-7]. Кез келген есептің шешу кезеңін қарапайым амалдар тізбегіне бөліктеуге болады. Алгоритмді компьютерде орындау үшін оны программа түрінде жазамыз.

Программа - машина тілі түсінетіндей, нұсқаулар тізбегі түрінде жазылған алгоритм. Программа командалар тізбегінен тұрады. Командалар тізбегі орындалған кезде есептің нәтижесі шығады. Әрбір компьютер алдын-ала құрылған программа бойынша жұмыс жасайды, яғни, программа белгілі бір нәтиже алу үшін процессор түсінікті операциялар тізбегін орындайды. Команда бір қарапайым операцияны орындауға бұйыратын бұйрық түрінде болады. Командалар арифметикалық, логикалық, басқаруды беру, сандарды салыстыру, экранға шығару, принтерге шығару және т.б. болып бөлінеді.

Ғылым мен техниканың ғарыштап дамуына байланысты программалау тілдері үнемі жаңарып, өзгеріп отырады. Ал соның ішінде Си программалау тілі - оқып үйренушіге өте жеңіл, әрі түсінікті. Си тілі программалауды қолданумен қатар, басқа программалау тілдеріне кіріспе болып та табылады. Программа құру оқушылардың білімін, іскерлігі мен дағдысын қалыптастырумен бірге, олардың шығармашылық қабілеті мен ақыл-ойын дамытудың маңызды құралы. Си программалау тілі қазіргі кезде жоғары оқу орындары мен мектептерде кеңінен оқытылады. Мұнда, есептерді шешуде алгоритмдік құрылымдарды қосымша құрал ретінде қолдану, есептің қойылымына сәйкес математикалық моделін құру, алгоритм жазу, нәтижені талдау кезеңдері қарастырады.

Кітаптың мақсаты – сызықтық, тармақталған және циклді алгоритмді программалау негіздері тарауында оқырманға алгоритм ұғымын, қасиеттерін, түрлері мен берілу тәсілдерін үйрету арқылы логикалық ойлауын, алгоритмдік білімін дамыту; Оқырманды алгоритмдік тілдің ережелерін білуге, өз бетінше алгоритм құрып, оған программа жазуға және компьютерде теріп, орындау іскерлігін қалыптастыру; Компьютерде есептер шығару кезеңдерін, программалау тілдерінің құрылымын, командалары мен операторларының қызметін түсініп, оларды тиімді пайдалана білу мәселелерін қалыптастыру. Оқу құралының соңындағы қосымшада баяндалған тақырыптарға байланысты тест сұрақтары берілген.

# 1. АЛГОРИТМДЕРДІҢ НЕГІЗГІ ҚҰРЫЛЫМДАРЫ

Алгоритм атауы атақты шығыс математигі абу Жафар Мұхаммед ибн Мұса әл-Хорезми (763-850 ж.) есімінің латынша Algorithmi (Алгоритми) болып жазылуынан шыққан. Алға қойған мақсатқа жету немесе берілген есепті шешу бағытында атқарушыға біртіндеп қандай әрекеттер жасау қажеттігін әрі түсінікті, әрі дәл етіп көрсететін нұсқаулар тобын агоритм деп атайды.

Алгоритмді компьютерде орындау үшін оны программа түрінде жазып шығу керек [1-3]. Программа машинаға түсінікті бұйрықтардан тұрады. Осы бұйрықтар тізбегі орындалу барысында есептің нәтижесін шығарады. Әрбір компьютер алдын ала жазылған программаны орындайды. Программа дегеніміз – белгілі бір нәтиже алу үшін орындалатын арнайы мәтін арқылы компьютерге тапсырманың ретті кезегін хабарлайтын бұйрықтардың айқындалған тізбегі. Процессор программаның құрамындағы бұйрықтарды кезекпен орындап отырады. Бұйрықтар: арифметикалық немесе логикалық амал; ақпаратты тасымалдау бұйрығы; берілген сандарды салыстыру бұйрығы; нәтижені экранға, қағазға басып шығару бұйрығы; келесі бұйрықтарға көшу тәртібін орындау, т.с.с.

Программалау тілінде қандай да бір программа құру үшін, ең алдымен есепті шешу алгоритмін құру керек. Реттілігін анықтауға байланысты алгоритм бірнеше қадамдардан құралған.

Алгоритм қасиеттері:

- Дискреттік – алгоритм қадам бойынша орнылалады және әрбір қадам алдыңғы қадамдар орындалғаннан кейін басталады.
- Детерминдік (анықтылық) – алгоритм нәтижелі болу үшін оған нақты енгізілетін мәндер айқын болуы керек.
- Нәтижелілік – алгоритмнің орындалуы нақты нәтиже береді.
- Ақырлылық – алгоритм белгілі қадамдарды орындап аяқталынады.
- Көпшілдік – берілген деректерімен ерекшеленетіп, белгілі есептер класына қолданылатын, есепті шешуге арналған жалпы түрдегі алгоритм.

Алгоритм берілген болуы үшін оның келесідей элементтері белгілі болу керек:

- деректерді құрайтын объектілер жиынтығы: алғашқы, ортаңғы және соңғы;
- бастапқы ереже;
- ақпаратты өңдеу ережесі;
- аяқталу ережесі;
- нәтижелерді алу ережесі.

Алгоритм әр уақытта нақты орындаушыға арналған. Егер орындаушы компьютер болса, онда алгоритм программалау тілінде жазылған болу керек.

Алгоритм ұсыныстарының әдістері:

- Сөздік-формулалық
- Алгоритмдік тілдер
- Графикалық тәсілдер
- Программалық (программалау тіліндегі мәтіндер)
- Жалған кодтар (шартты алгоритм тіліндегі жартылай формальды алгоритм сипаттамасы, программалау тілінің элементін өзіне қосып отырады, табиғи тіл фразасы және математикалық мағынасы).

Сөздік әдісте алгоритм жазу, деректерді өңдеу кезеңінде дәйекті сипаттамасын бейнелейді. Алгоритм негізсіз баяндауды табиғи тілге міндеттейді.

*Мысалы*: екі натурал сандардың ЕҮОБ табу алгоритмін жазыңыз.

Алгоритм келесідей болуы мүмкін:

1. Екі сан енгіземіз;
2. Егер сандар тең болса, онда біреуін жауабы ретінде алып тоқтайды, кері жағдайда алгоритм жұмысын жалғастырады.
3. Үлкен санды анықтаймыз.
4. Кіші және үлкен сандар айырымын үлкеніне ауыстыру.
5. 2-ші қадамнан бастап алгоритмді қайталау.

Графикалық түрдегі өзара тізбектелген күйін бейнелейтін алгоритм - функционалды блоктар. Олардың әр қайсысы бір

немесе бірнеше іс-әрекеттің орындалуына сәйкес келеді. Мұндай графикалық ұсынысты алгоритм схемасы немесе блок-схема дейді.

Жалған кодтар ережелері, жүйенің мазмұнын бейнелейтін алгоритм, жазбаларының біркелкілігіне арналған. Ол формальды және табиғи тілдер арасында орын алады. Бір жағынан қарапайым табиғи тілге жақын, сол себепті алгоритмдер қарапайым текст түрінде және жазба ретінде жазылады. Ал басқа қырынан қарасақ, жалған кодтар математикалық символдармен формальды конструкторлардың кейбіреулерін қолданады да, математикалық жазбалардың жалпы қабылдауы алгоритм жазбаларына бір табан жақындайды. Жалған кодта командалық жазба үшін ешқандай қатал синтаксисті ережелер қабылданбайды. Есептелген абстракты қолдануы, кең ауқымды командалар жиынын, мүмкіндігі бойынша қолдануы және жазба алгоритмінің жиынын жобалауда жеңілдетуі, формальды тілдерге тән.

Алайда, жалған кодтарда кейбір конструкциялар болады. Формальды тілдердің алгоритм жазбалары үшін орындалуын жеңілдететін жалған кодтар жазбасы формальды тілге жарасымды.

Жекеше алғанда жалған кодтарда да, формальды тілдер сияқты мағынасы нақты анықталған қызмет сөздері бар. Жалған кодтар үшін біркелкі және формальды анықтамалар жоқ, сондықтан әр түрлі жалған кодтарға ерекшеленетін қызмет сөздерінің жиыны және басты конструкторы болуы мүмкін.

Алгоритм құру үшін көбінде графикалық тәсілдер – блок-схема қолданылады.

Блок-схемада алгоритм әрбір қадамын арнайы мағынасы бар геометриялық фигурамен сызылады және ішінде қарапайым операциялар жазылады. Алгоритмді орындау үшін нұсқауларды көрсеткіш бағыт білдіреді.

Алгоритмі графикалық түрде беріліп, бір – бірімен байланысқан функционалдық блоктардан тұрса, сонымен қатар әрқайсысы белгілі бір әрекетке жауап берсе, ол блок-схема деп аталады. Блок-схемада әрбір әрекетке (кіріс мәліметтер, өрнек мағыналарын есептеу, шартты тексеру, қайталанатын әрекеттерді басқару, талдауды аяқтау және т.б.) геометриялық

фигура тиісті, ол блоктық символ ретінде көрсетіледі. Блоктық символдар бағыт (өту) сызықтарымен байланысады. Ол әрекеттің кезектігін білдіреді.

Блок «процесс» әрекеттің кезестесіп немесе әрекетті белгілеу, мағынасын өзгерту, берілген форманы немесе мәліметтердің орналасуы үшін қолданылады. Схеманың көрнекілігін арттыру мақсатында жеке блоктармен жұмыс жасау арқылы оларды бір блокқа біріктіруге болды. Жеке операциялардың көрсетілімі айтарлықтай бос орын алады.

Алгоритм әр уақытта нақты орындаушыға арналған. Егер орындаушы компьютер болса, онда алгоритм программалау тілінде жазылған болу керек. Келесі мысалда C/C++ программалау тілінде құрылған программа келтірілген.

*Мысалы.* a=3, b=5, c=7 үш сан берілген. Нәтижесінде

5, 5, 8 сандары алынады.

```
#include <iostream.h>
int main () {
int a=3, b=5, c=7;
a=b; b=a; c=c+1;
cout<<”a=”<<a;
cout<<”\tb=”<<b;
cout<<”\tc=<<c;
return 0;
}
```

Төмендегі кестелерде C/C++ программалау тілінде қолданылатын негізгі операциялар келтірілген.

C++ программалау тілінің негізгі операциялары

| белгіленуі | мағынасы |
|---|---|
| ++ | 1-ге өсіру |
| -- | 1-ге азайту |
| Size of | Өлшемі |
| ~ | Разрядтық жалған |
| ! | Логикалық жалған |
| – | Унарлық минус |
| + | Унарлық плюс |

| | |
|---|---|
| & | Адресті алу |
| New | Жадыны бөлу |
| delete | Жадыны босату |

Бинарлық және тернарлық операциялар

| | |
|---|---|
| * | Көбейту |
| / | Бөлу |
| % | Бөлуден қалдық алу |
| + | Қосу |
| - | азайту |
| << | Солға жылжыту |
| >> | Оңға жылжыту |
| < | Кіші |
| <= | Кіші немесе тең |
| > | Үлкен |
| >= | Үлкен немесссе тең |
| == | Тең |
| != | Тең емес |
| & | Разрядты конъюнкция (ЖӘНЕ) |
| ^ | Разрядты алып тастау (исключающие) НЕМЕСЕ |
| \| | Разрядты дизьюкция (НЕМЕСЕ) |
| & | Логикалық ЖӘНЕ |
| \|\| | Логикалық НЕМЕСЕ |

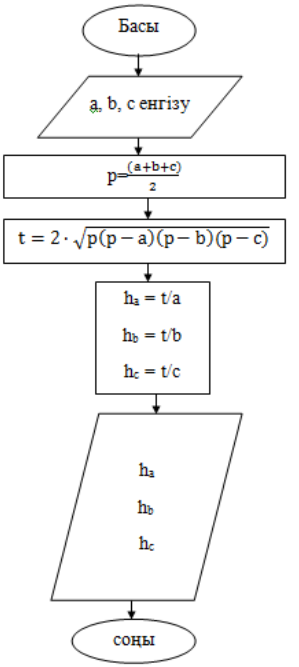
Блок «шешімі» басқарудан шартқа өтуді белгілуе үшін қолданылады. Әр блоктың «шешімі» сұрақ немесе салыстыру болуын анықтайды.

| | |
|---|---|
| ?: | Шартты операция (тернарлық) |
| == | Меншіктеу |
| *= | Көбейтіп меншіктеу |
| /= | Бөліп меншіктеу |
| %= | Қалдығын алып меншіктеу |
| += | Қосып меншіктеу |
| -= | Алып меншіктеу |
| <<= | Солға жылжытып меншіктеу |
| >>= | Оңға жылжытып меншіктеу |

Блок «модификация» циклдық конструкцияны ұйымдастыру үшін қолданылады. (Модификация сөзі түр өзгерту, өзгеріс дегенді білдіреді.) Блок ішінде циклдың параметрлері жазылады, сонымен оның бастапқы мағынасы көрсетіледі, шекаралық шарт және әрбір қайталануға өзгеру қадамдары параметрімен беріледі.

Блок «алдын ала анықталған процесс» программа кітапханасын оқу, автономды өмір сүретін түрлі жеке модульге көмекші алгоритмдерді қолдануға осындай бұйрықтар қолданылады.

Мысалы, берілген үшбұрыштың қабырғаларының ұзындықтарын a, b, c болғанда есептеу, егер




1 - сурет.

$$h_a = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)}$$

$$h_b = \frac{2}{b}\sqrt{p(p-a)(p-b)(p-c)}$$

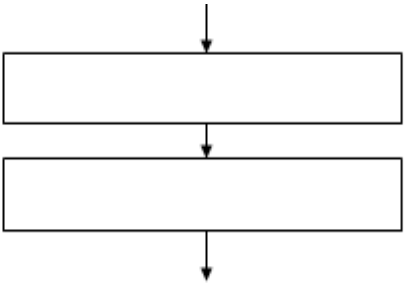
$$h_c = \frac{2}{c}\sqrt{p(p-a)(p-b)(p-c)}$$

Мұндағы, $p = \frac{a+b+c}{2}$

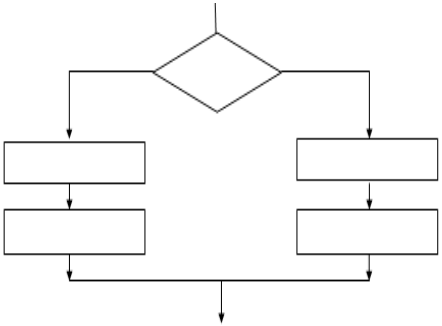
Шешімі:

$$t = 2\sqrt{p(p-a)(p-b)(p-c)}$$

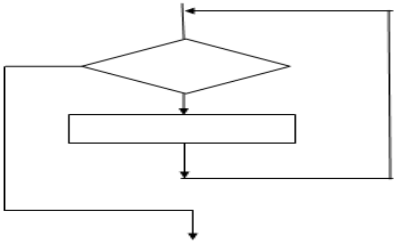
Олай болса енгіземіз, $h_a = t/a$, $h_b = t/b$, $h_c = t/c$. Блок-схеманың басы болуы қажет, a, b, c енгізіп, p, t, $h_a$, $h_b$, $h_c$ есептеп шығарады. Есептің блок-схемасы 1 - суретте көрсетілген.



**Оператор** – тілдің қарапайым сөйлемі, ол белгілі бір әрекет немесе амал орындап, (;) таңбасымен аяқталады. Сызықтық құрылым бірінен кейін бірі орындалып тізбектеле орналасқан бірнеше операторлардан тұрады.



**Тармақты** – шартқа байланысты екі оператордың бірінің орындалуы



**Цикл** – операторлар бөлігінің бірнеше рет қайталана орындалуы. Төменде алгоритмдердің бірыңғай құрылымдарының схемалық бейнеленуі көрсетілген.

Алгоритмдерді бірнеше базалық элементтерден тұратын кейбір құрылым ретінде қарастыруға болады. Алгоритмнің негізгі принциптерін түсіну үшін олардың базалық элементтерімен танысудан бастау керек.

Басқаруды беру операторлары:
С++ тілінде төрт оператор бар:

1) go to – шартсыз өту операторы
2) break – циклдан шығу операторы
3) continue – циклдің келесі итерациясына өту операторы
4) return – функциядан қайтару операторы

*Go to операторы* көрсетілген таңбасы бар қатарға басқаруды беру. Форматы: Go to таңба;

*Break операторы* циклдің немесе switch операторларының ішінде қолданылады. Break операторы басқаруды циклден кейінгі тұрған операторға береді.

*Мысалы:* x аргументті гиперболалық синустың мәнін eps дәлдікпен есептеу керек.

$Sh(x)=1+x^3/3!+x^5/5!+x^7/7!+\ldots$

```
#include <iostream.h>
#include <math.h>
int main () {
const  int MaxIter=500;   //итерацияның шектелген саны
duble X, eps;
cout<<”\n ” ;
cin>>X>>eps;      // аргументті және дәлдікті енгіз
bool flag=true ;                //сәтті есептеу белгісі
doule y=X, ch=X;          // бірінші қатар мүшесінің суммасы
for (int n=0; fabs(ch)>eps; n++) {
ch*=X*X/(2*n+2)/(2*n+3);            //қатардың мүшесі
y+=ch;
if (n>MaxIter) {Cout<<”\n қатарлар бөлінеді!”;
flag=false;
break: }
}
```

if (flag) cout<<”\n функцияның мәні”<<y;
return ø;
}

*Continue операторы* циклдың соңына дейінгі операторларды өткізіп жіберіп, келесі итерацияны бастайды.

*Return операторы* функцияның орындалуын аяқтайды. Форматы: Return [өрнек]; Өрнек скалярлық тип болады. Егер функцияның қайтару типі void болса, онда return операторы болмау керек.

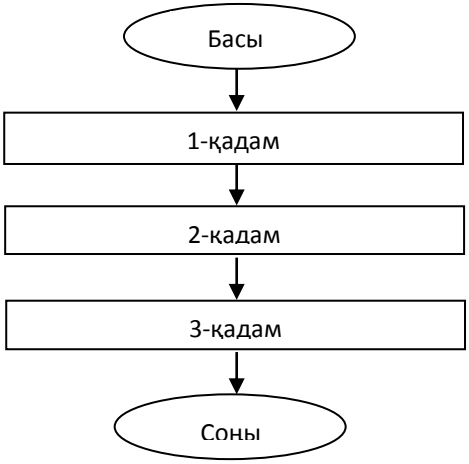
Кез келген алгоритмнің логикалық структурасы 3 базалық құрылымның комбинациясынан тұрады: сызықты, тармақталу, цикл.

Базалық құрылымдардың ерекшеліктері, оларда бір ғана шығыс және кіріс болады.

1. *Сызықты алгоритм.*

Сызықты алгоритм деп N қадамнан тұратын және оның барлық қадамы басынан соңына дейін бірінен соң бірі ретпен орындалуын айтамыз.

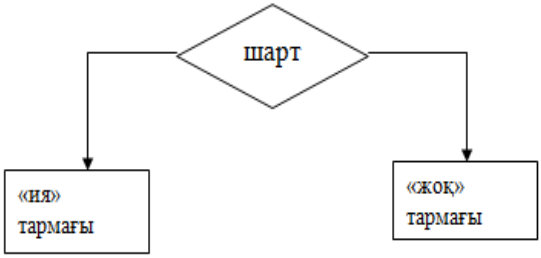
Сызықтық алгоритмнің әрекеттері бірінен кейін бірі орындалады (2 - сурет).




2 - сурет.

2. *Тармақталу алгоритмі.*

Шартты тексерудің нәтижесіне (ақиқат (ия) немесе жалған (жоқ)) байланысты альтернативті жолдардың бірін орындау тармақталу алгоритмі деп аталады. Әрбір жолдың қандай да бір жолын таңдасақ та алгоритм жұмысын нәтижеге әкеледі. Шарт – бұл логикалық тип. Шарттың нәтижесі тек екі мәнді қабылдайды: «иә» егер шарт ақиқат болса және «жоқ» - жалған болса (3 - сурет).

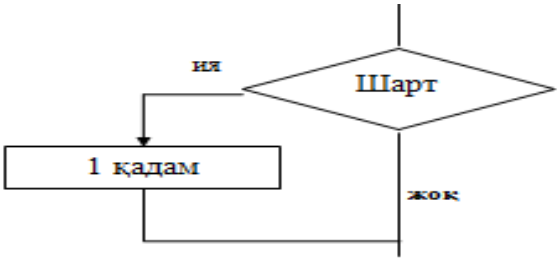



3 - сурет.

Құрылымның тармақталуының 4 түрі бар:

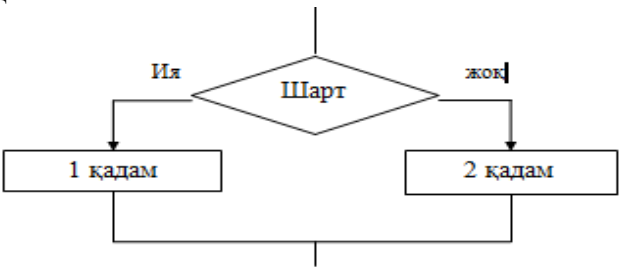
1. Егер;
2. Егер - әйтпесе;
3. Таңдау;
4. Таңдау - әйтпесе;

1) *Егер* (блок-схемасы 4 - суретте көрсетілген)
*егер* (шарт) 1 қадам
егер соңы




4 - сурет.

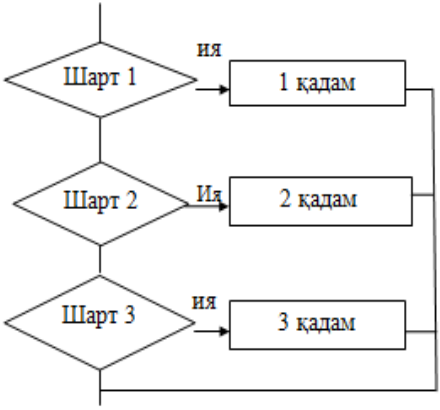
2) *Егер – әйтпесе* (блок-схемасы 5 - суретте көрсетілген)
*егер* (шарт) 1 қадам
*әйтпесе* 2 қадам
егер соңы




5 - сурет.

3) *Таңдау* (блок-схемасы 6 - суретте көрсетілген)

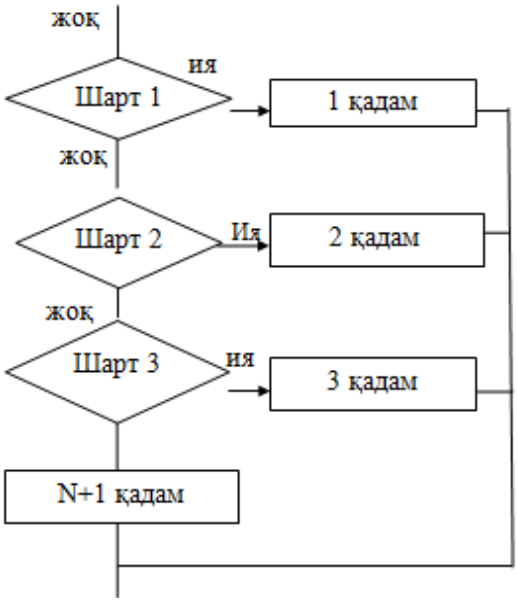
*шарт 1*: 1 қадам
*шарт 2*: 2 қадам
*шарт 3*: 3 қадам
.....
*шарт N*: N қадам
таңдау соңы




6 - сурет.

4) *Таңдау – әйтпесе* (блок-схемасы 7 - суретте көрсетілген)

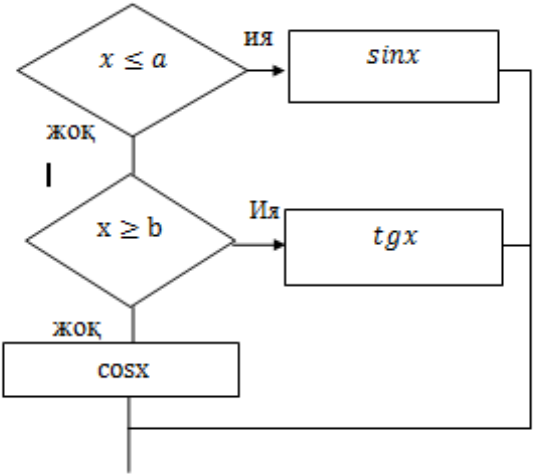
*шарт 1*: 1 қадам
*шарт 2*: 2 қадам
*шарт 3*: 3 қадам
.....
*шарт N*: N қадам
*әйтпесе* N+1 қадам
таңдау соңы




7 - сурет.

*Мысалы*. Функцияны есептеу алгоритміне блок-схема құру

$$x = \begin{cases} \sin x, & \text{егер} \quad x \le a \\ \cos x, & \text{егер} \quad a < x < b \\ tgx, & \text{егер} \quad x \ge b \end{cases}$$




8 - сурет.

Берілген мысалдың блок-схемасы 8 - суретте көрсетілген.

Тармақталу алгоритмінің Си программалау тіліндегі операторы if. Оның Си тіліндегі жалпы түрі мынадай:

if (шарт) оператор-1; [else оператор-2;]

Си программалау тіліндегі бірнеше мысалдар:

1) If (a<0) b=1;
2) If (a<b && (a<d||a==0)) b++; else { b*=a; a=0; }
3) If (a<b) { if (a<c) m=a; else m=c; } else { if (b<c) m=b; else m=c; }
4) If (a++) b++;
5) If (b>a) max=b; else m=a;
6) (max=(b>a)? b:a;)

*Мысалы*. Нысанаға атылған оқтың көрсеткіштерінің ұпай санын есептейтін С/С++ программалау тіліндегі программасы

```
#include <iostream.h>
#include <conio.h>
int main () {
float x,y;
int kol;
cout<<”атылған оқтың координатын енгіз\n”
cin>>x>>y;
if (x*x+y*y<1) kol=2;
else if (x*x+y*y<4) kol=1;
else kol=0;
cout<<”\n ұпай :”<<kol;
return ø;
getch();
}
```

```
#include <iostream.h>
#include <conio.h>
int main ()
float x,y, temp;
int kol;
cout<<”атылған оқтың координатын енгіз\n”
cin>>x>>y;
temp= x*x+y*y;
kol=0;
if (temp<4) kol=1;
if (temp<1) kol=2;
cout<<”\n ұпай :”<,kol;
return ø;
getch();
}
```

*Switch операторы.*
Switch операторы – қосқыш немесе таңдау операторына жатады.

Форматы:
Switch {
case тұрақты_өрнек_1: [1_оператор_тізімі]
case тұрақты_өрнек_2: [2_оператор_тізімі]
.....
case тұрақты_өрнек_n: [n_оператор_тізімі]
[default: операторлар]
}

*Мысалы:*

```
#include <iostream.h>
#include <conio.h>
int main () {
int a, b, res;
char op;
cout<<"\n 1-ші операцияны енгіз :";
cin>>a;
cout<<"\n белгі операциясын енгіз:";
cin>>op;
cout<<"\n 2-ші операцияны енгіз:";
cin>>b;
fool f=true;
swtch (op) {
case "+" res=a+b; break;
case "-" res=a-b; break;
case "*" res=a*b; break;
case "/" res=a/b; break;
defult: cout<<"\n белгісіз операция"; f=false;
}
if (f) cout<<"\n Нәтижесі:"res;
getch();
return ø;
}
```

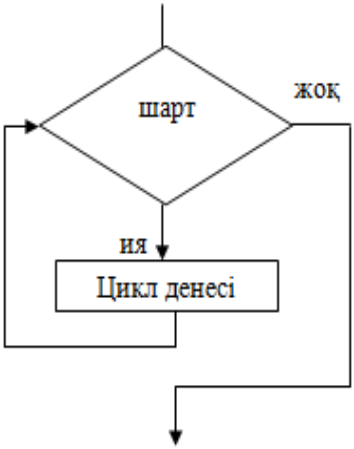
3. *Цикл (Қайталау құрылымы)*.

Цикл (қайталау операторы) деп шарттың ақиқат болуына байланысты цикл денесінің бірнеше рет орындалуын айтамыз. Цикл құрылымының 3 түрі бар:

- «Әзір» циклы (while – Си тіліндегі оператор)
- «Дейін» циклы (do-while – Си тіліндегі оператор)
- «Параметрлі» цикл (for – Си тіліндегі оператор)

*«Әзір» циклы* (while – Си программалау тіліндегі оператор)

«Әзір» циклының блок-схемасы:




9 - сурет.

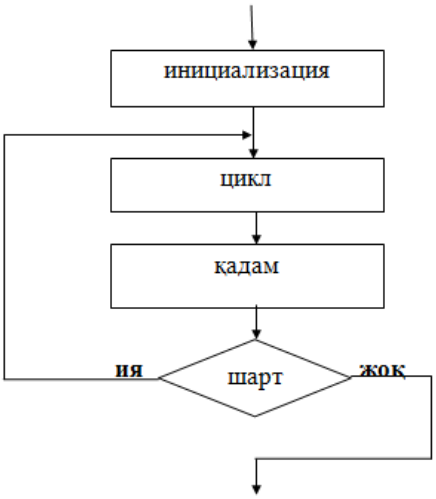
Циклге дейін инициализация орындалады, яғни циклдің алғашқы мәні меншіктеледі. Содан кейін шарт тексеріледі. Әзір шарт ақиқат болса, онда цикл денесі және қадам орындалады. Қадам белгілі бір мәнге өсіп немесе кеміп отырады. Шарт жалған болған кезде цикл қайталауын тоқтатып, циклден кейін орналасқан командаға өтеді. «Әзір» циклының блок-схемасы 9 - суретте көрсетілген

*Мысалы*. Программа берілген аралықтағы $y=x^2+1$ функцияның мәндерін кесте түрінде жазады.

```
#include <stdio.h>
int main(){
float Xn, Xk, Dx;
printf (“аргументтің өзгеру қадамын және аралығын енгізу”);
scanf (“%f%f%f”, &Xn, &Xk, &Dx);
printf (“|x|Y|\n”);            //кесте тақырыбы
float x=xk;                    //цикл параметрін орнату
while(x<=xk) {                 //шарттың жалғасын тексеру
printf(“|%5.2f.|%502f|\n”x, x*x+1);    //цикл денесі
X+=Dx;                                 //модификация параметрі
}
return ø;
}
```

*«Дейін» циклы* (do-while – Си программалау тіліндегі оператор)

«Дейін» циклының блок-схемасы:




10 - сурет.

*Инициализация* (алғашқы меншіктелуі)
*Цикл денесі* (тізбектелген іс-әрекеті, қадам)
*Дейін* (шарт)
цикл соңы

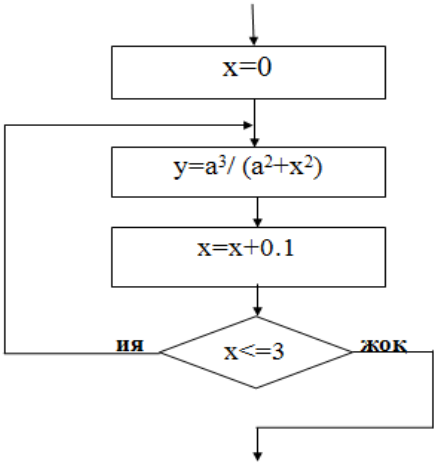
«Дейін» циклында цикл денесіне дейін *Инициализация* (алғашқы меншіктелу) орындалады. *Цикл денесі* (тізбектелген іс-әрекет, қадам) орындалғаннан кейін шарт тексеріледі. Егер шарт ақиқат болса, онда цикл денесі орындалады. Программа шарт жалған болғанда цикл денесінен шығады (10 - сурет).

*Мысалы:*
Программа мәнді енгізуді тексереді.

```
#include <iostream.h>
int main()
{
char answer;
do {
cout<<”\n ! компьютер сатып ал”;
cin>>answer;
}
while (answer!=”y”);
return ø;
}
```

Берілген есепте циклдың қайталау саны белгісіз болса, ондай есептерді, жоғарыда айтылған «Әзір» немесе «Дейін» циклдарының бірін қолданған ыңғайлы. Ондай циклдарды итерациялы цикл деп атайды. Итерациялы цикл берілген шарт орындалғанда жүзеге асады. Әрбір қадамда дәлдікке немесе нәтижеге шартты тексеру арқылы жақындайды.

*Мысалы.* $y=a^3/(a^2+x^2)$ функциясын есептеуге блок-схема құрыңыз. x айнымалысы x=0-ден x=3-ке дейін Dx=1.0 қадамымен өзгереді (11 - сурет).

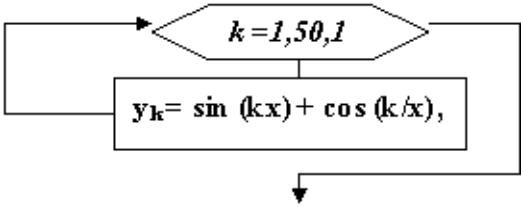



11 - сурет.

*«Параметрлі» цикл* (for – Си программалау тіліндегі оператор). «Параметрлі» цикл құрылымы 12 – суретте кескінделген:


12 - сурет.

*Мысалы.* $y_k=\sin(kx)+\cos(k/x)$, $(k=1,2,\ldots 50)$ функцияның есептеу алгоритміне блок-схема (13 - сурет).




13 - сурет.

C/C++ программалау тіліндегі оператор For (параметрлі цикл). Форматы:

For (инициализация; шарт; модификация) оператор;

*Мысалы*. 1-ден 100-ге дейінгі сандарды қосуды орындайтын параметрлі цикл

for(int i=1; s=0; i<=100; i++) s+=i;

*Мысалы*. Программа берілген диапазондағы $y=x^2+1$ функциясының мәндерін кесте түрінде кескіндеу [8]:

```
#include <stdio.h>
int main () {
float Xn, Xk, Dx, X;
printf(“аргументтің өзгеру қадамы мен диапазонын енгіз”);
scanf(“%f%f%f”, &Xn, &Xk, &Dx,);
printf (“|x|Y|\n”);
for (X’’=Xn; X<Xk; X+=Dx)
printf(“|%5.2f|%5.2f|\n”, X, X*X+1);
return ø;
}
```

For параметрлі циклын

for(b1; b2; b3) оператор
b1;

кез-келген While циклы түрінде жазуға болады, және керісінше:

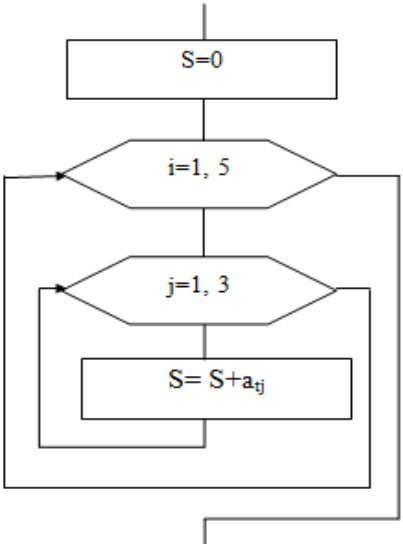
```
while(b2) {
оператор; b3;
}
```

*Цикл ішіндегі циклдар.*

Цикл денесінің ішінде белгілі бір операторлар тізбегін (ішкі циклды ұйымдастыру) бірнеше рет қайталап орындауды цикл ішіндегі цикл құрылымы деп атайды. Осындай құрылымды қолданған кезде, машиналық уақытты үнемдеу үшін ішкі циклға қатысы жоқ операторларды ішкі циклдан сыртқа шығару керек.

Цикл ішіндегі цикл құрылымы екі немесе көп өлшемді массивтерді есептеуде қолданылады. Екі өлшемді массивті матрица дейді. Екі өлшемді массив n жолмен m бағаннан тұратын жиымдар.

*Мысалы.* A(5,3) берілген матрицасының элементтерінің қосындысын есептеу (14 - сурет).




14 - сурет.

C/C++ программалау тілінде екі өлшемді жиымды - матрицаны пайдалану үшін тік жақшалар ішінде олардың екі өлшемінің де мәнін көрсету керек. Мысалы: int a[4][3] - алғашқы сан жолдар санын, ал екінші сан бағаналар санын көрсетеді, а жиымы 12 элменттен тұрады:

```
int a[4][3]={ {0,1,2}, {3,4,5}, {6,7,8}, {9,10,11} };
```

Оларға бастапқы мәнді былай береміз: ішкі жүйелі жақшаларды қоймаса да болады:

```
int a[4][3]={0,1,2,3,4,5,6,7, 8,9,10,11};
```

Келесі түрде сипаттау, жолдардың тек бірінші элементтерін ғана анықтайды, қалған элементтер 0-ге тең болып саналады:

```
int a[4][3]={ {0},{3},{6},{9} };
```

Егер ішкі жүйелі жақшалар алынып тасталса, онда мағынасы өзгереді:

```
int a[4][3]={0,3,6,9};
```

мұнда бірінші жолдың 3 элементі мен екінші жолдың бірінші элементі анықталады да, қалғандары 0 болып саналады.

Екі өлшемді массив элементтерін енгізу. Екі өлшемді массивті меншіктеу (инициалдау) қабаттасқан циклдер арқылы орындалады:

```
main() {
const int row=3, col=4; int a[row][col];
clrscr();
randomize();
for (int i=0; i<row; i++)
for (int j=0; j<col; j++)
a[i][j]= random(100)-50;
printf("\n a[3][4] массив элементтері:");
for (i=0; i<row; i++)
for (j=0; j<col; j++)
```

```
printf(" %i",a[i][j]);
getch();
}
```

Массив элементтері типі ескеріліп, олардың көлеміне жеткілікті етіп массив үшін компьютер жадында қажетті орын бөлініп беріледі немесе массивтің аты осы аймақтың басына сілтейтін нұсқауыш типті константа болып табылады. Берілген екі өлшемді массив мынадай түрде меншіктелген болсын:

```
static int a[3][4]={ {5,3,4,2}, {3,3,4,5}, {2,3,3,4} };
```

Осы массивтің әрбір жолдағы элементтер қосындыларын және сол қосындылардың орташа мәнін анықтайтын программа:

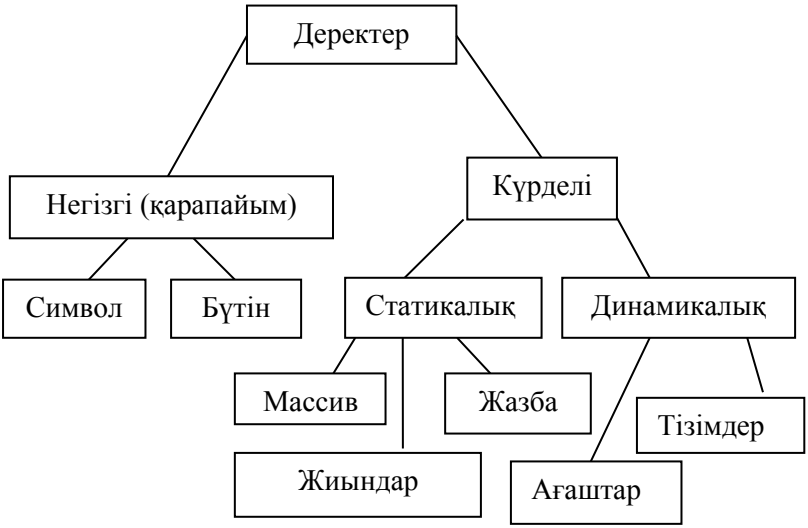
```
/* әр жол қосындысы және солардың арифм. ортасы */
#include <conio.h>
#include <stdio.h>
main () {
static int a[3][4]={ {5,3,4,2}, {3,3,4,5}, {2,3,3,4} }; int i,j,s=0;
float c=0;
clrscr();
for (i=0; i<3; i++) {
for (j=0; j<4; j ++)
s+=a[i][j];
printf("%i-жол косындысы: %i\n",i+1,s);c+=s; }
printf("--------------------");
printf("\n арифм.ортасы %5.2f",c/4);
}
```

## 2. ТИПТЕР ЖӘНЕ ДЕРЕКТЕР ҚҰРЫЛЫМЫ

ЭЕМ жадында сақталатын деректер нөл мен бір (биттер) жиынтығынан тұрады. Биттер тізбектерге бірігеді: байттар, сөздер және т.с.с.. Оперативті жадының әрбір бөлігіне (бір байт немесе сөз сыйғызуға болатын) реттік нөмір (адрес) меншіктеледі.

Практикалық есептерді шешуде барлық деректер бірнеше типтерге бөлінеді. Тип сөзінің ұғымы, тек адрестік кеңістікте деректерді ұсынумен емес, оларды өңдеуменде байланысты. Кез келген деректер екі типке бөлінеді: негізгі (қарапайым) - ЭЕМ архитектурасы анықталуы арқылы көрсетілетін форма; күрделі - нақты есептерді шешудегі тұтынушының құруы.

Негізгі (қарапайым) деректер типтері – символдар, сандар және т.с.с. элементтер, әрі қарай бөлшектеуді қажет етпейді. Қарапайым деректерден деректер құрылымы (күрделі типтер) құрастырылады (15 – сурет).




15 - сурет.

Деректердің кейбір құрылымдары:

- *Массив* (ақырлы облыста анықталатын функция) – бір типтегі деректердің қарапайым элементтер жиынтығы. Массивтің жеке элементі индекспен беріледі. Массивтер бір өлшемді, екі өлшемді және т.с.с. болуы мүмкін. Бір өлшемді массивтің айнымалы ұзындығының әр түрлілігіне *сақина, стек, кезек* және *екі жақты кезек* типті құрылымдар жатады.
- *Жазу* (декартты шығарма) – әр түрлі типтегі деректер элементтерінің жиынтығы. Жазба тұрақты элементтер санынан тұрады, олар *өрістер* деп аталады. Құрылымдары бірдей жазбалардың жиынтығын *файл* деп атайды. (Сол сияқты, сыртқы жадыдағы деректер жиынтығын да файл деп атайды. Мысалы, магниттік дискіде файлдар). Файлдан жеке жазбаларды алу үшін әрбір жазбаға атау немесе нөмір беріледі, сонымен қатар олардың идентификаторы жеке өрісте орналасады және қызмет көрсетеді. Ол идентификаторды *кілт* деп атайды.

Массив немесе жазбалар тәрізді деректер құрылымдары ЭЕМ жадынан үнемі орын алады, сондықтан оларды статикалық құрылымдар дейді. Статикалық құрылымдарға *жиындар* да жатады [1,2].

Ұзындығын өзгерте алатын құрылымдар да бар, оларды *динамикалық құрылым* деп атайды. Оған ағаштар, тізімдер жатады.

Элементтерді орналастыруға қажетті сызықты емес адресті кеңістікті талап ететін маңызды құрылым – *ағаштар*. Ағаш түріндегі деректер құрылымы көп кездеседі. Мысалы, классификациялық, иерархиялық, рекурсивті және т.б. құрылымдар.

## 2.1 Негізгі (қарапайым) деректер типтері

Программада айнымалыларды жариялаған кезде олардың типі де анықталады. Деректер типтері нақты деректерді және оларға қолданылатын амалдарды анықтайды. Деректер типіне жатады: Integer [бүтін], Float, Double [нақты], Character [символдық] және Boolean [логикалық].

Int (Integer) типі бүтін сандар болып табылатын сандық деректерді анықтайды. Жадыда олар әдетте қосымша екілік код ретінде сақталады. Осы Integer типі арқылы қарапайым арифметикалық операциялар мен салыстыру операцияларын орындай аламыз.

Float, Double типі бүтін емес болып табылатын сандық деректерді анықтайды. Жадыда олар әдетте жылжымалы нүктелер түріндегі екілік код ретінде сақталады. Real типті деректерге қолданылатын амалдар Integer типтегі деректерге қолданылатын амалдармен бірдей.

Char (Character) типі жадыда ASCII және UNICODE кодтар түрінде сақталған символдардан тұратын деректерге қолданылады. Осы типтегі деректерді бір-бірімен салыстыруға болады (екі символдың қайсысы алфавит бойынша бірінші кездеседі); екі қатарды бір қатарға біріктіру, бір қатарды екінші қатардың соңына тіркеп жазу арқылы (конкетанация амалы).

Bool (Boolean) типі True [ақиқат] және False [жалған] деген мағынаны ғана қабылдай алатын деректерге жатады. Мысал ретінде екі санды салыстыру нәтижесінде алынған мәліметтерді жатқызуға болады. Boolean типті деректер айнымалы мәні True немесе False екендігін анықтайды.

Машинаның негізгі жадысы адресі тізбектелген өспелі адресті жеке ұяшықтар түрінде ұйымдастырылған. Бірақ, жиі бұл ұяшықтар, деректерді басқаша орналастыруды қолданады. Мысалы, мәтін әдетте символдардан құрылған ұзын жол ретінде қарастырылады, ал сатылым жайлы ақпарат болса, сандық мағынадан тұратын тікбұрышты кесте ретінде көрсетілген.

## 2.2 Күрделі деректер типтері

Компьютерді дұрыс қолдану үшін ондағы деректер арасындағы құрылымдардың байланысын, сонымен қатар олармен жұмыс істеу әдістерін жақсы білу керек. Компьютердегі деректер арасындағы байланыстар үшін келесі ақпараттық құрылымдар қолданылады: массив, жазба, тізім, ағаш, стек, кезек.

*Массивтер*

Массив – бұл құрылым, бір типті элементтерден тұратын. Элементтерді реттеу үшін индекстер қолданылады. Индекстер массив атауынан кейін жақша ішінде жазылады. Бір индекстен тұратын массив - бірөлшемді, ал екеуден тұрса - екі өлшемді және т.с.с. аталады.

C/C++ программалау тілінде массив өлшемін size of (массив атауы) операциясы арқылы анықтауға болды.

*Бір өлшемді массивтер*

Бір өлшемді массивтер сызықты (кей уақытта вектор) деп аталады. Сызықтық массивті программада оператор атауы: айнымалы типінен, массив атауынан және квадраттық жақшаға алынған элементтер санынан тұрады.

*Мысалы:* float Var [2700];

2700 элементтен тұратын Var массивінің типі float (бөлшек сан). Массив элементтерінің санын бүтін сандармен ғана емес, сонымен қатар атауы бар тұрақтымен немесе бүтін типті тұрақты өрнекпен беруге болады.

*Мысалы:* const int Amount = 50;
unsigned short Num [2* Amount];

Массив айнымалыларына int, char, long, double және т.б. типтерді қолдануға болады (void-тан басқа). Массив айнымалысының индексі 0-ден басталады, онан кейін натурал

сандармен жалғасады - 0,1,2.... Массивтің соңғы элементінің индексі n-1-ге тең.

*Мысалы:*

```
Num[5] =785
Var [0] = Num[5]*Num[5]
```

*Мысалы:*

```
#include <iostream.h>
#include <conio.h>
double Numbers [5], Sum, Average;
int main() {
int i;
cout<<5 санының ариф. Орташа мәнін есептеу:» <<endl << endl;
cout<<« 5 санын енгіз:»<<endl;
for(i=0; i<5; ++i)                // Numbers массивінің санын енгіз
{
cout <<(i+1)<<«:»;
cin>>Numbers [i];
}
for (sum=0.0; i=0;i<5; ++i) //сандардың қосындысын есептеу
sum==Numbers [i];
Average = sum/ 5.0;          // орт. мәнін есептеу
cout<<endl<<«орташа мәні «Average «endl;
getch();
return 0;
}
```

*Екі өлшемді массив*

```
float Matrix [5][7];
```

5*7=35 элементтен тұратын Matrix массиві. Элементтер саны 0-ден 4-ке дейін =5;  0-ден 6-ға дейін =7.

*Мысалы.* Екі өлшемді массивті кез келген сандармен толтырып, оның ішінен тақ және жұп сандардың санын анықтайтын программа құрыңыз.

```
#include <stdio.h>
#inlude <conio.h>
int Mat[5][6];
int Even, bdd;
int main () {
int i,j;
//екі өлшемді массивін толтыру 1-100 дейінгі кездей соқ диапазон аралығында толтыру.
randomize();
for(i=0;i<5;i++)
for(j=0;j<6;j++)
Mat [i][j]=1+random(99);
// Mat массивін толықтай экранға басып шығару
for(i=0;i<5;i++)
{
for(j=0;j<6;j++)
printf("\n%5d", Mat[i][j]);
printf("\n");
}
// Mat массивінің тақ, жұп мәндері
Even =0dd= 0;
for(i=0;i<5;i++)
for(j=0;j<6;j++)
if(Mat[i][j]%2)  ++ Even;
else                ++ 0dd;
printf("\n%5d-amount of even numbers"
"\n%5d-amount of even numbers"
"\n\n%5d-amount of even numbers", Even, 0dd, Even+0dd);
getch()
return 0;
}
```

*Көп өлшемді массивтер*

```
class MultiArrayDemo
{
int [ ] [ ] static void main (string[] args)
{
int [ ][ ] ia=new int [3][3];
for (int i=0; i<3; i++)
{
for (int j=0; j<3; j++)
{
Ia[i][j]=10*i+j+100;
System.out.println();
}
}
}
```

Көп өлшемді массивтерді құруда сол жағындағы бірінші өлшемді (қатар санын) міндетті түрде көрсету керек. Қалған өлшемдерін, кейін new операторы арқылы көрсетуге болады. Сөйтіп, осындай әдіс арқылы иррегулярлы массив құруға болады.

Мысал:

```
class  IrregularArrayDemo
{
public static void main (string[ ] args)
{
int [ ] [ ] ia=new int [3] [ ];
for (int j=0; j<ia[i]; j++)
{
ia[i][j]=i*10+j+100;
system.out.print(ia[i][j]+” “);
}
System.out.println();
}
}
}
```

*Функциялар*

Функция - белгілі бір тапсырмаларды орындауға арналған программаның жеке дара бөлігі.

Функция не үшін қажет?

1) Қайталап программалаудан құтқарады. Белгілі бір есепті бірнеше рет орындау үшін функцияны бір рет құруға болады. Сонан кейін негізгі программаның денесін орындау барысында функцияны шақырамыз.
2) Функция программаны модульдық структура түріне келтіреді де бірақ рет орындайды, яғни оны оқуға және теңдеулер енгізуге ыңғайлы.

*Мысалы:* төмендегі амалдарды орындайтын программа құру керек:

Сандар тізімін оқу - readlist();
Сандарды реттеу - sort();
Олардың орта мәнін табу - average();
Гистограмнан печаттау - bargraph();

Бұл тапсырмаларды орындау үшін төмендегідей программа жазылады:

```
#include <stdio.h>
#define SIZE 50
Int main () {
float list [SIZE];
readlist (list, SIZE);
Sort (list, SIZE);
average(list, SIZE);
bargraph(list, SIZE);
return 0;
}
```

Қарапайы функцияларды құру және қолдану үшін төмендегі программаны қарастырайық.

*Мысалы:* қатарға 65 жұлдызшаны басатын программа құру

```
#include <stdio.h>
#define NAME “KazNU”
#define ADDRESS “Timiryazev, 71 ”
#define PLACE  “Almaty”
#define LIMIT 65
void starbar (void);    /* функция прототипі*/
int main()  {
starbar ();
printf(“%5\n”, NAME);
printf(“%5\n”, ADDRESS);
printf(“%5\n”, PLACE);
starbar();        /* функцияны қолдану*/
return0;
}
void starbar (void)     /* функцияны анықтау*/
{
int count;
for(count=1; count<=LIMIT; count++)
putchor(“*”);
putchor(“/n”);
}
```

Экранда шығады

```
**************************
KazNU
Timiryazev, 71
Almaty
**************************
```

Программа іске қосылғанда бірінші main () функциясы орындалады. main функциясы орындалу барысында басқа функцияларды шақырады.

Функцияны анықтау:

Функция тақырыптан және операторлар блогынан тұрады.

1) Қайтатын мәтіннің типі
2) Функция атауы
3) () жакшаға алынған параметрлер тізімі

*Мысалы:* үш санның максимумын есептейтін функция

```
double m=x;
if(m<y) m=y;
if(m<z) m=z;
return m;
}
```

Функцияны шақыру

```
max=Fmax(1.0, arg, arg*arg);
double Fmax (double x, double y, double z);
```

Функцияның ақпаратты беретін ерекшіліктері:

1) Функция оперативтік жадыда өзінің айнымалы-параметрлерін орналастырады.
2) Функцияны шақырғанда аргументтер орындалады және олардың мәндері сәйкесті айнымалы-параметрлерге меншіктеледі.

Return операторы орындалғаннан кейін, функция өз қызметін аяқтайды, сосын:

1) Функцияның тақырыбындағы типке сәйкес уақытша айнымалы құрылады.
2) Return операторында орналасқан мәні есептеледі және оның мәні уақытша айнымалыға сақталады.

*Мысалы:*

```
max=Fmax(1.0, arg, arg*arg);
x=1.0
y=arg
z=arg*arg
```

*Файлдармен жұмыс*

C/C$^{++}$ тіліндегі программа сыртқы құрылғымен өзара байланыс жасау үшін деректер ағыны ұғымын қолданылады. Ағын, әр түрлі байттар тізбегінің жұмысын модельдейтін оперативтік жадыдан физикалық құрылғыға немесе құрылғыдан оперативтік жадыға берілетін әмбебап логикалық құрылғыны құрайды. Ағындар қасиетіне байланысты текстік және екілік (бинарлық) деп бөлінеді.

*Ағынды құру және жабу*

Құрылғыға байланысты ағынды құру (өшу) fopen() функциясы атқарады:

FILE*fopen(const char*filename, const char*mode);

Бірінші параметр filename шақыру барысында дискідегі файлдың атауын немесе енгізу/шығару портындағы символдық қатардың адресін қабылдайды. Дискідегі деректер файлмен ауыстырылған кезде бағыты көрсетілген файлдың атауы қысқа немесе ұзақ болуы мүмкін. Егер қысқа атау көрсетілсе, онда жүйе оны әуелі жұмыс каталогында іздейді, сонан кейін path айнымалылар тізбегінде. Деректерді портқа жібергенде символдық қатар оның атауын білу керек. IBM PC стандартты компьютерлерде енгіру/шығарудың төмендегідей атаулары бар: CON, PRN, AUX, COM 1- COM4, LPT 1- LPT9.

```
Const char FileName 1[]=”c:\\Out\\t1.txt”;
        pF1=fopen(FileName1, “rt”)
```

fopen функциясының екінші параметрі, шақыру барысында, директиваларды құрайтын файлдың құруын және ашуын басқаратын қатардың адресін қабылдайды. Сонымен қатар, төмендегідей директивалар ағын типін анықтайды – текстік және екілік:

“r” – бар файлды оқу;

“w” – жаңа файл құру және оны жазу. Егер файл бар болса, онда оны ашу алдында ішіндегі ақпаратты тазалау (өшіру);

“a” – жаңа файл құру және оны жазу. Егер файл бар болса, онда оны файлдың соңына тіркеу.

“r+” – бар файлды ашу, яғни оны оқу және жазу операциясын атқару;

“w+” – жаңа файл құру, оқу және жазу. Егер файл бар болса, онда ашу алдында оның ішіндегі ақпаратты тазалау.

“a+” – жаңа файл құру, оқу және жазу. Егер файл бар болса, онда көрсеткіш оны файлдың соңына орналастырады.

Ағынның типін көрсету үшін аталған әр директивадан кейін “t” (символ) немесе “b” (бинарлық) символын тіркеу керек. Егер жарияланбаса, онда үнсіз “t” типі қабылданады.

Файлдық операцияны болғаннан кейін ағынды fclose() функциясы арқылы жабу керек.

int fclose(FILE*stream);

Ағын дұрыс жабылса, онда fclose 0-ді береді, ал дұрыс жабылмаса – EOF (END of File). EOF макросы stdio.h файлында анықталған және (-1) мәнді құрайды.

Си программалау тілінде автоматты түрде үш анықталған ағын ашылады: stdin, stderr, stdout. Олар консольді енгізу/шығарумен байланысты. stdin, stdout - стандартты printf(), scanf(), gets(), puts() функцияларын қолданады.

Stderr ағыны шығару болып табылады: оған программа орындалу барысында тоқтау болған кездегі оперативтік жүйе қатерлердің немесе себептен болғаны жөніндегі ақпараттарды енгізеді:

Fclose(pF1).

C/C++ программалау тілдеріндегі деректердің ағындармен алмасу операциясының орындалуы үшін төмендегідей функциялар қолданылады:

Fprintf(), fscanf(), fputs(), fgets(), fwrite(), fread(), fputs(), fgets() және макростар putc(), gets().

*Жазба*

Жазба – бір типті емес элементтерден тұратын құрылым. Жазбаның жеке элементтері алаң деп аталады. Алаң өз кезегінде жазба да бола алады.

*Мысалы:*

record Student (FirstName, LastName, Group)

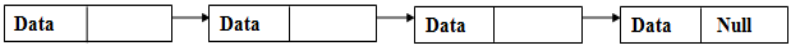
*Тізім*

Тізім – әрқайсысы арнайы нұсқау [Pointer] алаңынан тұратын көптеген жазбалар. Нұсқаушы бір жазбаны басқа кез келген жазбамен байланыстырып немесе Null мағынасын қамтиды да, бұл нұсқаушы мағынасының анықтылмағандығын білдіреді.

Тізім – жазба тізбегі. Тізіммен орындалатын басты операциялар: жазбаны қарау, жаңа жазба қосып және тізімнен жазбаны өшіру. Құрылымы өзгермелі күрделі обьектілерді құруға тізім мүмкіндік береді.

*Бірбайланысты жазбалар*

Бірбайланысты жазбада, әр тізімде, бір-бір нұсқаушыдан болады және олар тізбектей байланысады (16 – сурет):



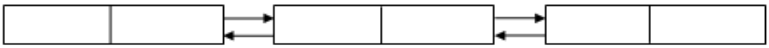

16 - сурет.

16 - суреттегі сызықтар, нұсқауыштар туралы хабар береді, ал Data сөзі мәліметтер сақталған алаң жинағын білдіреді. Тізімді екі өлшемді массив көмегімен құруға болады, бірінші индекстері нөлге тең, элементтер мәліметтерді сақтауға арналған, ал индекстері бірге теңдері нұсқағыштар. Берілген тізімде жазбалар ағылшын алфавитіндегі әріптен құралған және алфавит бойынша орналасқан. Тізімдегі бірінші жазба A символы , екіншісі B сиволы т.с.с.. Тізіммен жұмыс жасағанда үш басты операцияны орындай алалмыз:

Pass() – тексеріс немесе тізім бойына ауысу;
Add() – тізімге жаңа жазба қосу;
Delete() – жазбаны тізімнен жою;

*Екібайланысты тізімдер*

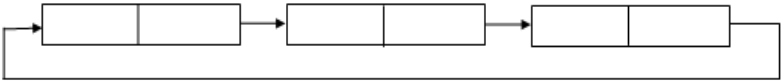
Екібайланысты тізімде жазбалар бір бірімен тізбектей байланысады, бірақ екі алаң және екі нұсқаушысы болады. Оның біреуі тізімдегі алдыңғы элементті, ал екіншісі келесі элементті нұсқайды. Мұндай құрылым екі бағытта ауысуға мүмкіндік береді: алдыға және артқа (17 – сурет).



17 - сурет.

*Сақиналық тізім*

Сақиналық тізім деп соңғы жазба бірінші жазбаны нұсқағанды айтамыз. Бұл тізімдегі жазбада бос нұсқағыш болмайды (18 – сурет).

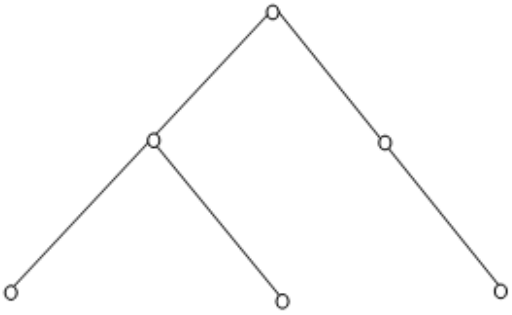


18 - сурет.

*Ағаштар*

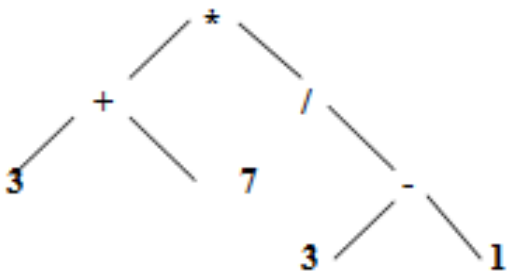
Ағаштар – әр жазба бірнеше нұсқағыштардан тұратын тарамдалған тізім. Ағашқа кіретін тізімдеуші *түйін* деп аталады. Тек бос нұсқағыштардан тұратын тізімді *жапырақтар* деп атаймыз. Ағаштың бастапқы жоғарғы түйіні түбірі деп аталады. Көптеген есептерде екіден жоғары емес нұсқағышы бар түйіндер, яғни екілік (бинарлық) ағаштар қолданылады (19–

сурет). Көптеген қолданбалы есептерді шешуде обьектілік жиынын ағаштар күйінде шешу ыңғайлы.



19 - сурет.

*Мысалы:* (3+7)*(2/(3-1)) математикалық өрнекті есептеу керек. Осыны ағаш түрінде көрсетейік (20 - сурет).




20 - сурет.

Ағаштың әрбір түйіні келесі түрдегі жазбада көрсетіледі.

Record Node(Operation Number LeftPointer RightPointer)

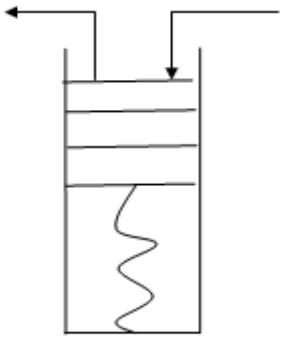
Ағаш жапырақтары сандарды қамтиды, түйіндер операция символарына жатады.

Ағаштың мәнін есептеу үшін оның оң және сол жақ ішкі ағаштардың мәнін есептеп, сонан кейін нәтижелік амалдарды орындаймыз. Осы алгоритмнің псевдокоды мына түрде болады:

```
function Calculate (Current) {
    if (M[2][Current]=Null) then
    Result:= M[1][Current];
    else {
        R1:=Calculate(M[2][Current]);
        R2:=Calculate(M[3][Current]);
        Result:=R1(M[0][Current])R2;
    }
    return (Result);
}
```

*Стек*

Стек – «Соңғы келді – бірінші кетті» принципіне құрылған мәліметтер құрылымы [Last In – First Out, LIFO]. Стек – деректер құрылымының «бірінші болып кіріп, бірінші шығу» принципіне құрылған (21 – сурет). Деректер стекке тізбектей орналасады. Мысалы: кітаптар бумасы, автомат магазині. Сол себепті де жады магазині деп атайды.



21 - сурет.

Стекте сақталған мәліметтерге кіру мүмкіндігі жоғарыдан бастадады. Мәліметтер стекке кезекпен түседі де бірінші түскен элемент ең астында қалып қалады. Оны шығару үшін кейін келген элементтердің барлығын жою керек. Стекпен жұмыс кезінде 2 жағдай орын алуы мүмкін:

1. бос стектен мәліметерді оқу мүмкіндігі;
2. стектегі элемент саны барынша көп деңгейге жеткенде стекке элемент кіргізу мүмкіндігі.

*Кезек*

Кезек – «бірінші келді – бірінші кетті» принципіне құрылған мәліметтер құрылымы [First In – First Out, FIFO] (22 – сурет). FIFO айнымалылардағы мәліметтер саны кезекте тұрады. Кезекке қосылу соңынан жүреді, басынан жойылады. Элементті өңдеу бірінен соң бірі тізбектелген кезегімен жүреді. Қосылған жаңа элемент кезектің соңынан енгізіледі. Кезек элементінің басты операциялары: оқу, өңдеу, кезекке жазу, кезектен өшіру.


22 - сурет.

# 3. СҰРЫПТАУ АЛГОРИТМДЕРІ

Алаңдардың біреуі кілт (кейін оны сұрыпталу кілті деп атаймыз) ретінде таңдап алынған бір типті тізбектеліп орналасқан тізбек берілген. Кілттің деректер типі келесі салыстыру операцияларынан тұруы керек: («=», « >», «<», «<= », «>=»). Сұрыпталудың мақсаты, ағындағы тізбекті өсу (немесе кему) кілтіне қарай сол жазбалардан тұратын басқа тізбекке түрлендіру. Егер сұрыптауды қолданғаннан кейін кілтке қатысты жазбаның орны өзгеріссіз қалса, сұрыпталудың бұл әдісі тұрақты деп аталады [1,5].

Негізгі жадыда орналасқан массив элементтерін сұрыптауды - *ішкі сұрыптау*, ал негізгі жадыға толығымен сыймайтын сыртқы жадыға сақталған файлдық деректерді *сыртқы сұрыпту* деп ажыратады. Әрине ішкі және сыртқы сұрыптауға әртүрлі әдістер қолданылады.

Сұрыптау алгоритмдерінде кілтке байланысты элементтерді салыстырулар және элементтердің орын ауыстырулары орындалады. Сұрыптау әдітерін C/C++ программалау тілінде жазылған толықтырамыз.

*Мысалы:*

```
#include <iostream>
#include <string>
#include <regex>
int main (){
  std::string s ("this subject has a submarine as a subsequence");
  std::regex e ("\\b(sub)([^ ]*)");
  std::regex_iterator<std::string::iterator> rit ( s.begin(), s.end(), e );
  std::regex_iterator<std::string::iterator> rend;
  while (rit!=rend) {
    std::cout << rit->str() << std::endl;
    ++rit;
  }
  return 0;
}
```

## 3.1 Ішкі сұрыптау әдістері

*Қосып сұрыптау*

Ішкі сұрыптаудың қарапайым әдістерінің бірі – қосып сұрыптау алгоритмі. Алгоритмнің мағынасы қарапайым. Берілгені a[1], a[2], ..., a[n] массиві. Массивтің әр элементіне екіншісінен бастап, индекстері кіші элементтерімен салыстыру жүргізіледі (a[i] элементі келесі a[i-1],a[i-2] элементтерімен ретретімен салыстырылады), әзірге a[j] элементіне a[j]> a[i] шарт кездеспейінше. Егер осы шарт орындалса, онда a[i] және a[j] элементтері орын ауыстырады. Егер a[j]<=a[i] шартын қанағаттындыратын a[j] элементін кездестірсек және массивтің төменгі шекарасына жетсек, онда a[i+1] элементін өңдеуге өтеміз (массивтің жоғарғы шекарасына жеткенше).

Көріп отырғанымыздай, n элементтен тұратын массивті сұрыптау (егер массив элементтері реттелген болса) үшін n-1 салыстыру және 0 жіберу орындалады (жақсы жағдайда). Жаман жағдайда (егер массив элементтері кері реттелген болса), n*(n-1)/2 салыстыру және соншама жіберу орындалады. Сонымен, қосып сұрыптау әдісінің күрделілігін $O(n^2)$ арқалы бағалауға болады.

Қарапайым қосып сұрыптау алгоритміндегі салыстыру сандарын кішірейтуге болады; егер массивтің a[i] элементін өңдеу барысында a[1], a[2],...,a[i-1] элементтері алдын-ала реттелген болса. Бұл жағдайда ондай массивке екілік бөлу әдісін қолдануға болады және оның керек салыиырулардың санын бағалау O(n/log n) тең. Орын ауыстыру үшін бір элементке жылжу қажет болғандықтын, жіберу бағасы O(n2) болып қала бермек.

Қосып сұрыптау алгоритміне мысал 1–кестеде келтірілген.

1 - кесте

| *Қарапайым қосып сұрыптау алгоритмі* | |
|---|---|
| **Массивтің бастапқы күйі** | 8 23 5 65 44 33 1 6 |

| | |
|---|---|
| **1-қадам** | 8 23 5 65 44 33 1 6 |
| **2-қадам** | 8 5 23 65 44 33 1 6<br>5 8 23 65 44 33 1 6 |
| **3-қадам** | 5 8 23 65 44 33 1 6 |
| **4-қадам** | 5 8 23 44 65 33 1 6 |
| **5-қадам** | 5 8 23 44 33 65 1 6<br>5 8 23 33 44 65 1 6 |
| **6-қадам** | 5 8 23 33 44 1 65 6<br>5 8 23 33 1 44 65 6<br>5 8 23 1 33 44 65 6<br>5 8 1 23 33 44 65 6<br>5 1 8 23 33 44 65 6<br>1 5 8 23 33 44 65 6 |
| **7-қадам** | 1 5 8 23 33 44 6 65<br>1 5 8 23 33 6 44 65<br>1 5 8 23 6 33 44 65<br>1 5 8 6 23 33 44 65<br>1 5 6 8 23 33 44 65 |

Шелл әдісі арқылы сұрыптау қосып сұрыптаудың әрі қарай дамуы немесе арақашықтығы азаятын қосып сұрыптау деп аталады. Шелл әдісін жалпы жағдайға қарастырмай, тек сұрыпталатын массив элеметтерінің саны 2 дәрежелі (массив элеметтері саны жұп) жағдайын қарастырайық. 2n дәрежелі

массив элементтеріне Шелл әдісі мынадай алгоритммен орындалады.

Бірінші фазада, элементтердің ара қашықтығы 2(n-1) болатын массивтің барлық екі элементіне қосып сұрыптау әдісін қолданамыз. Екінші фазада, элементтердің ара қашықтықтары 2(n-2) болатын, алынған массив элементтеріне қосып сұрыптау әдісін қолданамыз, т.с.с.. Элементтердің ара қашықтықтары 1-ге тең фазаға жеткенше, қосып сұрыптау әдісі аяқталғанша жоғарыдағы процесті жалғастыра береміз. Жоғарыда қарастырылған массивке Шелл әдісін қолданайық (2-кесте).

2-кесте. Шелл әдісімен сұрыптауға мысал.

| | |
|---|---|
| **Массивтің бастапқы күйі** | 8 23 5 65 44 33 1 6 |
| **1-Фаза (аралығы төрт болатын элементтер сұрыпталады)** | 8 23 5 65 44 33 1 6<br>8 23 5 65 44 33 1 6<br>8 23 1 65 44 33 5 6<br>8 23 1 6 44 33 5 65 |
| **2-Фаза (аралығы екі болатын элементтер сұрыпталады)** | 1 23 8 6 44 33 5 65<br>1 23 8 6 44 33 5 65<br>1 23 8 6 5 33 44 65<br>1 23 5 6 8 33 44 65<br>1 6 5 23 8 33 44 65<br>1 6 5 23 8 33 44 65<br>1 6 5 23 8 33 44 65 |
| **3-Фаза аралығы бір болатын элементтер сұрыпталады)** | 1 6 5 23 8 33 44 65<br>1 5 6 23 8 33 44 65<br>1 5 6 23 8 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65 |

Жалпы жағдайда, $h_1$, $h_2$,...,$h_t$ элементтердің ара қашықтығы t болатын берілген тізбекке $h_1=1$ және $h_{i+1}<h_i$ шарты орындалған кезде Шелл алгоритмін қолдануға болады. Дональд Кнут көрсеткендей, егер t және h дұрыс таңдалса, онда Шелл алгоритмінің күрделілігі қарапайым сұрыптау алгоритмдерінің күрделілігінен кіші және $O(n^{(1,2)})$ болады.

*Мысал:* Шелл әдісін қолданып, C++ тілінде программа құру

```
#include <iostream.h >
#include <conio.h>
using namespace std;
#define maxn 100
int a[maxn];
int n;
void merge(int l, int r) {
 if (r == l)
 return;
    if (r - l == 1) {
        if (a[r] < a[l])  swap(a[r], a[l]);
        return;}
    int m = (r + l) / 2;
    merge(l, m);
    merge(m + 1, r);
    int buf[maxn];
    int xl = l;
    int xr = m + 1;
    int cur = 0;
    while (r - l + 1 != cur) {
        if (xl > m)  buf[cur++] = a[xr++];
        else if (xr > r) buf[cur++] = a[xl++];
        else if (a[xl] > a[xr]) buf[cur++] = a[xr++];
        else buf[cur++] = a[xl++];}
    for (int i = 0; i < cur; i++)
        a[i + l] = buf[i];}
int main() {
    cin>> n;
```

```
    for (int i = 0; i < n; i++)
        cin>> a[i];
    merge(0, n - 1);
    for (int i = 0; i < n; i++)
        cout<< a[i] << " ";
    getch();
    return 0;
}
```

*Алмастырып сұрыптау*

a[1], a[2], ...,a[n] берілген массивке алмастырып сұрыптау әдісі («көпіршік әдісі» деп те аталады) төмендегідей орындалады. Массивтің соңынан бастап екі көрші элементтер (a[n] және a[n-1]) салыстырылады. Егер a[n]<a[n-1] шарты орындалса, онда бұл элементтердің мәндері орын алмасады. Процесс a[n-1] және a[n-2] элементтері үшін жалғасады және т.с.с., яғни a[2] және a[1] элементтері өзара салыстырылғанша жалғасады. Соңында a[1]-ші орында массивтің ең кіші мәнді элементі орналасатындығы түсінікті. Екінші қадамда процесс қайталанады, бірақ соңғы болып a[3]-ші және a[2]-ші салыстырылады т.с.с.. Соңғы қадамда ағымдағы a[n] және a[n-1] элементтер салыстырылады. «Көпіршік» ұғымының берілуі түсінікті, себебі ең кіші элементтер (ең «жеңіл») біртіндеп массивтің жоғарғы шекарасына «қалқып» шығады. Көпіршікті сұрыпталу мысалы 3-кестеде көрсетілген.

3-кесте. «Көпіршік» әдісімен сұрыптауға мысал.

| **Массивтің бастапқы күйі** | 8 23 5 65 44 33 1 6 |
|---|---|
| **1-қадам** | 8 23 5 65 44 33 1 6<br>8 23 5 65 44 1 33 6<br>8 23 5 65 1 44 33 6<br>8 23 5 1 65 44 33 6<br>8 23 1 5 65 44 33 6<br>8 1 23 5 65 44 33 6 |

| | |
|---|---|
| | 1 8 23 5 65 44 33 6 |
| **2-қадам** | 1 8 23 5 65 44 6 33<br>1 8 23 5 65 6 44 33<br>1 8 23 5 6 65 44 33<br>1 8 23 5 6 65 44 33<br>1 8 5 23 6 65 44 33<br>1 5 8 23 6 65 44 33 |
| **3-қадам** | 1 5 8 23 6 65 33 44<br>1 5 8 23 6 33 65 44<br>1 5 8 23 6 33 65 44<br>1 5 8 6 23 33 65 44<br>1 5 6 8 23 33 65 44 |
| **4-қадам** | 1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65 |
| **5-қадам** | 1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65 |
| **6-қадам** | 1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65 |
| **7-қадам** | 1 5 6 8 23 33 44 65 |

Қарапайым алмастырып сұрыптау әдісіне n*(n-1)/2 салыстыру, минималды 0 сілтеме саны керек, және сілтеменің орташа және жоғарғы саны – $O(n^2)$.

*Мысалы:*

```
#include <iostream.h>
#include <conio.h>
void swap (double*pb);  //swap функциясына қарсы
const int Array Size=5;
Int i, iter, n;
cout<<”Enter”<<Array Size<<”real numbers:”<<endl;
for(i=0; i<Array Size, i++)
{
cout<<endl<<(i+1)<<”:”;
cin>>Array [i];
}
/* көпіршік арқылы сұрыптау
for (iter=0; iter<Array Size-1; ++iter)
for (n=Array Size-1; n>iter; --n)
if(Array [n-1]>Array[n]) swap(&Array[n-1], &*Array[n]);
cout<<”\”the order array:”<<endl;
for(i=0; i<Array Size; i++)
cout<<Array[i]<<endl;
getch()
return()
}
```

Көпіршікті сұрыпталудың үш қарапайым жетілдірілген түрлері бар. Біріншіден, 3-ші кестеде көрсетілгендей, соңғы 4-ші қадамда элемент мәндерінің орналасуы өзгерген жоқ (массив реттелген болды). Осы себепті, егер бір қатарда мүлде алмастыру орындалмаса, онда алгоритмнің орындалуын тоқтатамыз. Екіншіден, ағымдағы қатарда алмастыру орындалған массив индекстерінің кіші мәнін есте сақтауға болады. Осы индекті массив элементінің жоғары жағы сұрыпталған және келесі қадамда осындай индексті мәнге жеткенде көрші элементтерді салыстыруды тоқтатуға болады. Үшіншіден, көпіршікті сұрыптау әдісі «жеңіл» және «ауыр» мәндерге бірдей жұмыс істемейді. Жеңіл мәндер керек орынға бір қадамда барады, ал ауыр мәндер әр қадамда керек орнына бір позицияға жылжиды.

Осы бақылауларға қарап шейкерлік (ShakerSort) сұрыпталу негізделген. Оны қолдану барысында әр қадамда тізбектей қарау бағыты өзгеріп отырады. Нәтижесінде, бір қадамда кезекті жеңіл элемент «қалқып» шығады, ал келесі қадамда ауыр элемент «батады».

4-кесте. Шейкерлік сұрыптау әдісі.

| | |
|---|---|
| **Массивтің бастапқы күйі** | 8 23 5 65 44 33 1 6 |
| **1-қадам** | 8 23 5 65 44 33 1 6<br>8 23 5 65 44 1 33 6<br>8 23 5 65 1 44 33 6<br>8 23 5 1 65 44 33 6<br>8 23 1 5 65 44 33 6<br>8 1 23 5 65 44 33 6<br>1 8 23 5 65 44 33 6 |
| **2-қадам** | 1 8 23 5 65 44 33 6<br>1 8 5 23 65 44 33 6<br>1 8 5 23 65 44 33 6<br>1 8 5 23 44 65 33 6<br>1 8 5 23 44 33 65 6<br>1 8 5 23 44 33 6 65 |
| **3-қадам** | 1 8 5 23 44 6 33 65<br>1 8 5 23 6 44 33 65<br>1 8 5 6 23 44 33 65<br>1 8 5 6 23 44 33 65<br>1 5 8 6 23 44 33 65 |
| **4 қадам** | 1 5 6 8 23 44 33 65<br>1 5 6 8 23 44 33 65<br>1 5 6 8 23 44 33 65 |

| | |
|---|---|
| | 1 5 6 8 23 33 44 65 |
| **5 қадам** | 1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65<br>1 5 6 8 23 33 44 65 |

Шейкерлік сұрыптау әдісі салыстырулар санын азайтады (Кнуттың бағалауы бойынша орташа салыстырулар саны ($n^2$-n*(const+ln n)), сонда да бағалау реті $n^2$ болып қала береді. Жіберулер саны тұрақты.

*Таңдап сұрыптау*

Таңдап сұрыптау әдісінде берілген a[1], a[2], ..., a[n] массивінің ішінен ең кіші a[i] элемент анықталынады, сонан кейін a[1] және a[i] элементтері орындарын алмастырады. Сонан кейін осы процесс a[2], a[3], ...,a[n], ... a[j], a[j+1], ..., a[n] ішкі массиві үшін қолданылады, a[n] ішкі массивіне жеткенше және сол мезетте ең үлкен мәнді меншіктегенше қайталанады. Алгоритмнің жұмысы 5-ші кестеде мысалмен көрсетілген.

5-кесте. Таңдап сұрыптау әдісі.

| **Массивтің бастапқы күйі** | 8 23 5 65 44 33 1 6 |
|---|---|
| **1-қадам** | 1 23 5 65 44 33 8 6 |
| **2-қадам** | 1 5 23 65 44 33 8 6 |
| **3-қадам** | 1 5 6 65 44 33 8 23 |

| | |
|---|---|
| **4-қадам** | 1 5 6 8 44 33 65 23 |
| **5-қадам** | 1 5 6 8 33 44 65 23 |
| **6-қадам** | 1 5 6 8 23 44 65 33 |
| **7-қадам** | 1 5 6 8 23 33 65 44 |
| **8-қадам** | 1 5 6 8 23 33 44 65 |

Қарапайым таңдап сұрыптау әдісіне $n^*(n-1)/2$ салыстырулар саны қажет. Осы әдістің ең нашар жағдайында сілтемелер саны $O(n^2)$ құрайды. Сілтемелердің орташа мәні $O(n^*$ ln n)-ге тең.

*Мысалы. Таңдап сұрыптауды қолданып құрылған программа*

```
#include <stdafx.h>
#include <iostream.h>
using namespace std;
int i, j;
void SelectionSort(int A[], int n)      //таңдап сұрыптау
{
int count, key;
for (i=0; i<n-1; i++)
{
count=A[i]; key=i;
for (j=i+1; j<n; j++)
if (A[j]<A[key]) key=j;
if (key!=i)
{
A[i]=A[key];
A[key]=count;
```

```
}
}
cout<<"Нәтижелік массив: ";
for (i=0; i<n; i++) cout<<A[i]<<" "; //массивті енгізу
}
//басты функция
void main(){
setlocale(LC_ALL, "Rus");
int n, A[1000];
cout<<"элементтер саны > "; cin>>n;
for (i=0; i<n; i++) //массивті енгізу
{ cout<<i+1<<" элемент > "; cin>>A[i];
}SelectionSort(A,n);system("pause>>void");
}
```

*Бөліп сұрыптау (Quicksort)*

Бөліп сұрыптау әдісін 1962 жылы Чарльз Хоар ұсынған болатын (ол өзін Тони деп атағанды ұнататын). Бұл әдіс қарапайым алмастырып сұрыптау әдісінің жетілдірілген түрі және оның тиімділігі соншалықты болғандықтан оны "жылдам сұрыптау әдісі - QuickSort" деп атады.

Алгоритмнің басты мағынасы мынада. Берілген массивтен кездейсоқ бір x элементі таңдалады. Сонан кейін массив сол жағынан бастап a[i]>x шартын қанағаттандыратын a[i] элементі кездескенше қарастырылады. Сосын, массив оң жағынан a[j]<x шартын қанағаттандыратын a[j] элементі кездескенше қарастырылады. Осы екі элементтер орын алмастырады да, қарау, салыстыру және орын алмастыру процесі x элементіне жеткенше жалғастырылады. Нәтижесінде массив екі бөлікке бөлінген болып шығады – сол жағында, x-тен кілті кіші элементтер, және оң жағында, x-тен кілті үлкен элементтер. Әрі қарай процесс, әр жағында бір элемент қалғанша, массивтің сол және оң жақтарына рекурсивті жалғасады. Егер массивтің сәйкесті индекстерін есте сақтап отырсақ, онда рекурсияны итерациямен алмастыруға болады. Біздің стандартты массивімізге осы процестің қалай жүретінін 6-шы кестеден көруге болады.

6-кесте. Бөліп (жылдам) сұрыптау әдісі.

| | |
|---|---|
| **Массивтің бастапқы күйі** | 8 23 5 65 \|44\| 33 1 6 |
| **1-қадам (х ретінде а[5]-ші таңдалынады)** | \|----------\|<br>8 23 5 6 44 33 1 65<br>\|------\|<br>8 23 5 6 1 33 44 65 |
| **2- қадам (а[1], а[5] ішкі массивінде х ретінде а[3] таңдалды)** | 8 23 \|5\| 6 1 33 44 65<br>\|------------\|<br>1 23 5 6 8 33 44 65<br>\|----\|<br>1 5 23 6 8 33 44 65 |
| **3-қадам (а[3], а[5] ішкі массивінде х ретінде а[4] таңдалды)** | 1 5 23 \|6\| 8 33 44 65<br>\|-------\|<br>1 5 8 6 23 33 44 65 |
| **4-қадам (а[3], а[4] ішкі массивінде а[4] таңдалды)** | 1 5 8 \|6\| 23 33 44 65<br>\|-----\|<br>1 5 6 8 23 33 44 65 |

Алгоритмды бекерден жылдам сұрыптау деп атамаған, себебі, ондағы салыстырулар мен алмастырулардың бағалау саны O(n*log n) болып табылады. Шынында, көптеген массивтерді сұрыптауды қолданатын утилиталарда (программалық жабдықтар) осы алгоритм қолданылады.

*Мысалы. Жылдам сұрыптау әдісіне программа құру.*

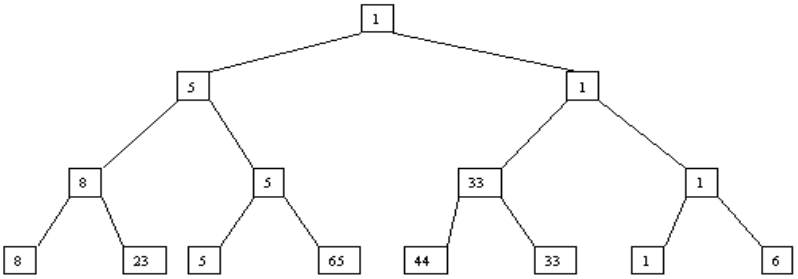
```
#include <iostream.h>
usingnamespace std;
```

```
int a[100];
void quickSort(int l, int r)
{
   int x = a[l + (r - l) / 2];    //эквивалент жазбасы(l+r)/2,
   //үлкен деректерге қосымша шақырулар
   int i = l;
   int j = r;
   // while коды particle процедурасына шақырылады
   while(i <= j)
   {
      while(a[i] < x) i++;
      while(a[j] > x) j--;
      if(i <= j)
      {
         swap(a[i], a[j]);
         i++;
         j--;
      }
   }
   if (i<r)   quickSort(i, r);
      if (l<j)   quickSort(l, j);
}
int main(){
   int n;//массивтегі элементтер саны
   cin >> n;
   for(int i = 0; i < n; i++)
   {       cin>>a[i];    }
   quickSort(0, n-1);
   for(int i = 0; i < n; i++)
   {       cout<<a[i]<<" ";    }
   return 0;
}
```

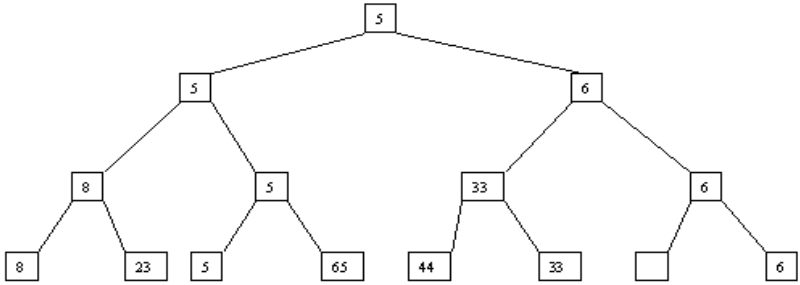
*Ағаш көмегімен сұрыптау (Heapsort)*

Салыстырмалы кілтті екілік ағаштан құрылған қарапайым ағаш көмегімен сұрыптау әдісінен қарастырайық.

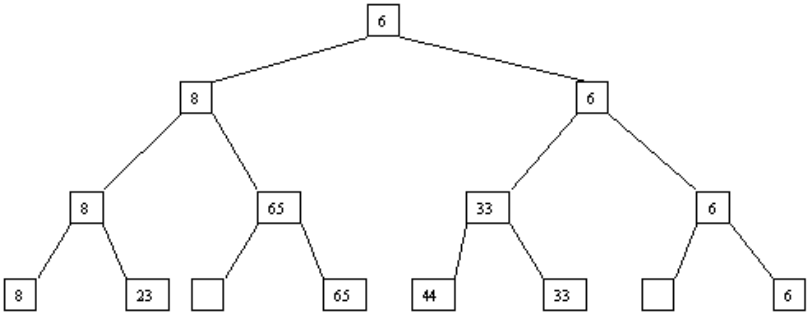
Массивтің барлық элементтері жапырақтарына орналасқан ағашты құрайық. Жапырақта орналасқан әр көрші элементтердің ішінен кішілері таңдап алынып, олар келесі сабақтағы элеметтерді құрайды және т.с.с., сөйтіп n-1 салыстырулардан кейін ағаштың түбіріне массивтің ең кіші кілтті элементі шығады (23,а сурет). Екінші қадамда ең кіші кілтте көрсетілгендей жолмен түсеміз және оны алып тастаймыз, тізбектей оны «тесікке» (шексіз кілт) алмастыру арқылы, немесе осы түйіннің қарама-қарсы сабағында орналасқан элементке ауыстырамыз (23,б - 23,ж суреттер). Ағаштан алынған әр кіші кілтті элементтерді рет-ретімен жаңа массивке жазып отырамыз. Осындай n қадамнан кейін ағаш бос болып қалады (яғни «тесіктерден» тұрады) және сұрыптау процесі аяқталады (23,з сурет). Сонда өсуі бойынша реттелген жаңа массив аламыз.



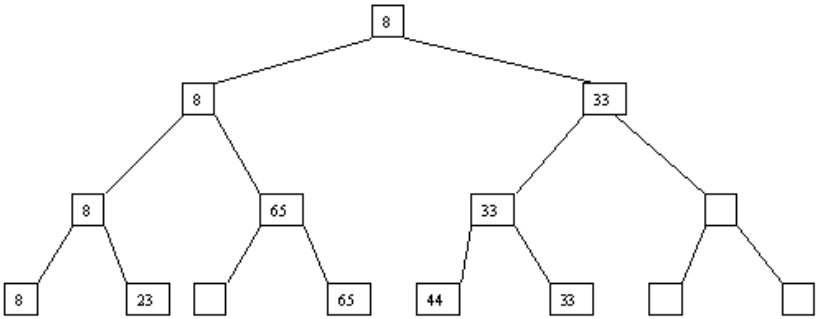

а)



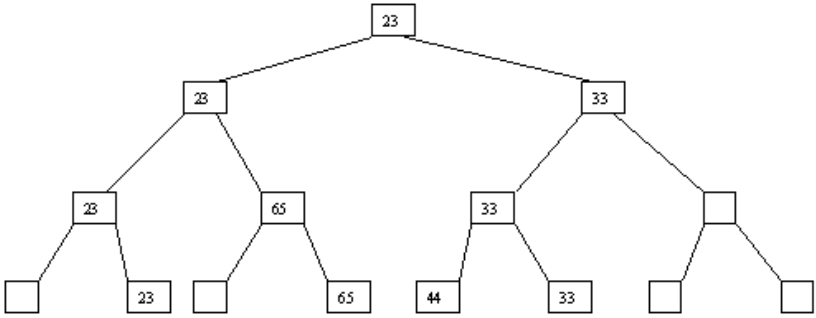

б)


6
8
6
8
65
33
6
8
23
65
44
33
6

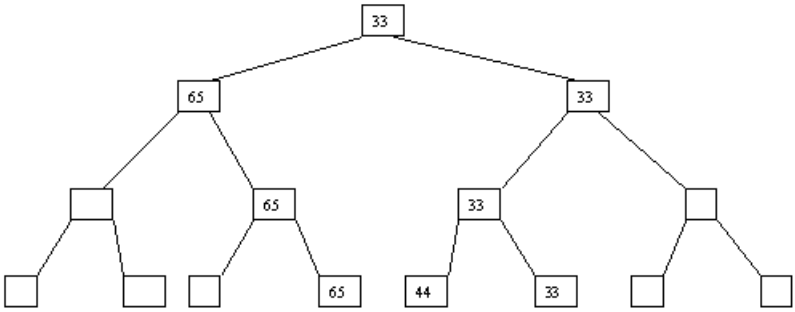
в)


8
8
33
8
65
33
8
23
65
44
33

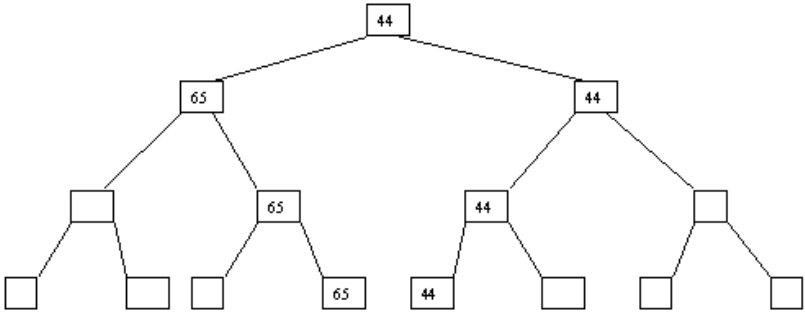
г)


23
23
33
23
65
33
23
65
44
33

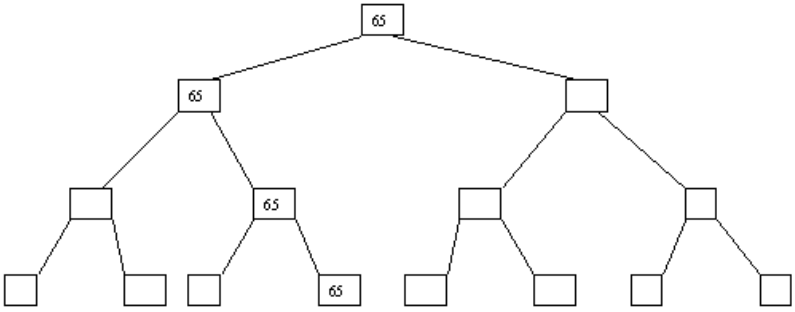
д)




е)




ж)




з)

23-сурет.

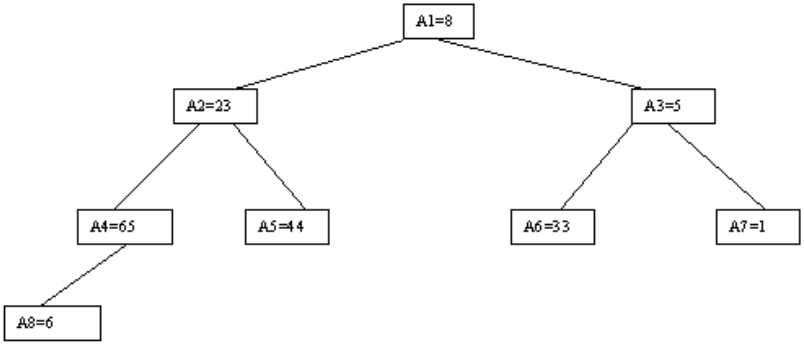
Сонымен, әр n қадам log n салыстыруды қажет етеді. Сондықтан, ағашты тұрғызуға қажетті n қадамды санамағанда, барлық сұрыптау $n*\log_2 n$ элементарлық операцияны қажет етеді. $n^2$ қадамды қолданатын қарапайым әдістен бұл әлде қайда ұтымды.

Әрине, ағаш көмегімен сұрыптауда ақпаратты сақтау күрделілеу болады, сондықтан кейбір қадамдардың күрделілігі артады. Алғашқы жүрісте алынған ұлғайған ақпараттар көлемін сақтау үшін бір ағаштық құрылым құру керек. Біздің келесі мақсатымыз – осы ақпаратты ұйымдастырудың тиімді әдісін қарастыру. Әрине, соңында барлық ағашты толтыратын және көптеген қажет емес салыстыруларға алып келетін «тесіктерден» құтылғанымыз өте қолайлы болар еді. Сонымен қатар, жоғарыда көрсетілгендей, жадыдан 2n-1 бірліктің орнына n бірліктен тұратын, n элементтерден ағашты құрайтын әдіс табу қажет. Ондай нәрсені Дж. Уильямс ойлап тапқан *пирамидалық сұрыптау* әдісі арқылы жасауға болады. Бұл әдіс үйреншікті ағаш көмегімен сұрыптау әдісіне қарағанда тиімді әдіс болып табылады.

*Пирамидалық сұрыптау әдісі (Heapsort)*

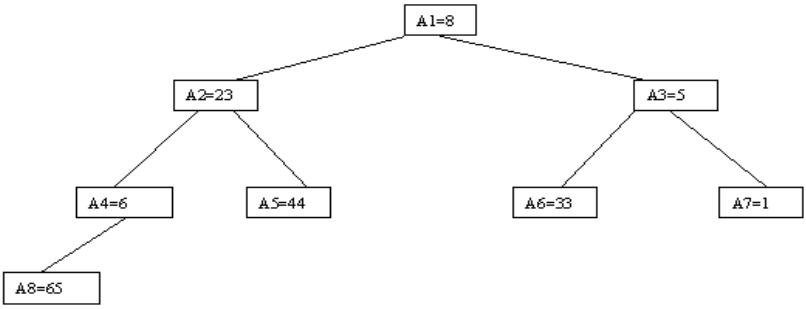
Берілген a[1], a[2], ..., a[n] массиві, толық ағашты салыстырудың орнына, әрбір a[i] элементі үшін a[i]<=a[2i] және a[i]<=a[2i+1] шарттарды қанағаттандыратын қасиеті бар пирамидаға түрлендіріледі. Сонан кейін пирамидаға сұрыпталу қолданылады.

Пирамиданы құру әдісі, 24-суретте көрсетілгендей, массивті ағаш түріне келтіру арқылы көрнекті көрсетіледі. Массив, түбірі массивтің a[1] элементіне сәйкес келетін, екілік ағаш түрінде өрнектеледі. Екінші қабатта a[2] және a[3]-ші элементтер орналасқан. Үшіншіде - a[4], a[5], a[6], a[7] және т.с.с.. 24-суреттен көріп тұрғанымыздай, тақ элементтерден тұратын массив тепе-теңдік ағашты құрайды, ал n элементтер саны жұп болса, онда a[n] жалғыз элементі сол жақ жапырақта орналасады.

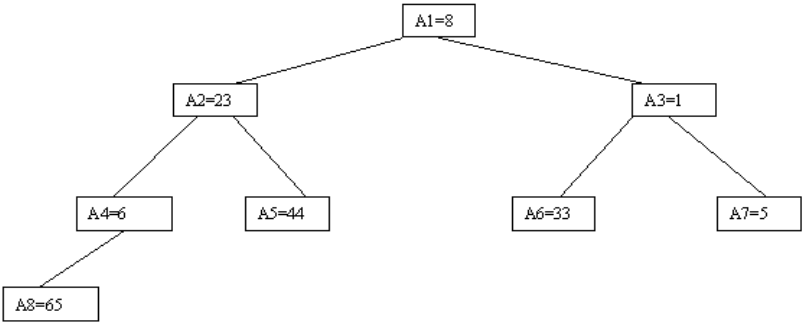



24-сурет.

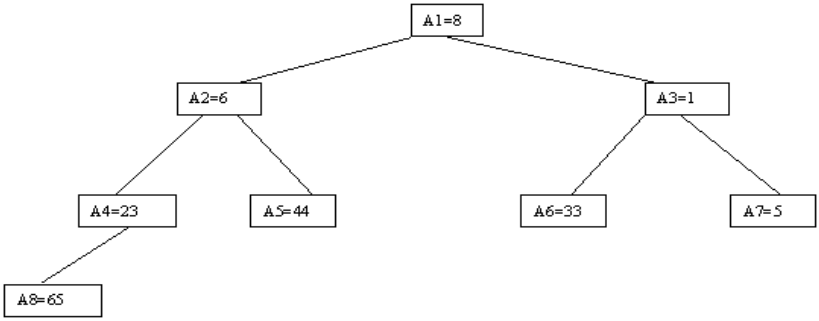
Әрине, пирамида құрған кезде, бізді, жұп элементтерден тұратын массив үшін a[n/2], a[n/2-1], …, a[1] элементтері және тақ элементтерден тұратын массив үшін a[(n-1)/2], a[(n-1)/2-1],…, a[1] элементтері (осындай элементтер үшін ғана пирамида шектеулі) қызықтырады. i – индекстер ішіндегі ең үлкені және де оның пирамиданы шектеуге қатысы бар болсын. Онда, құрылған ағаштан a[i] элементі алынады, min(a[2*i], a[2*i+1]) және a[i] мәні сәйкес элементтің мәнімен алмастырылатын сүзу процедурасы орындалады. Егер бұл элемент ағаштың жапырағы болмаса, онда оған сәйкесті жоғарыда айтылған процедура орындалады және т.с.с.. Осындай әрекеттер a[i], a[i-1], ..., a[1] элементтеріне тізбектей орындалады. Нәтижесінде, 25 суретте қадам бойынша көрсетілген, берілген массивтен пирамида аламыз.



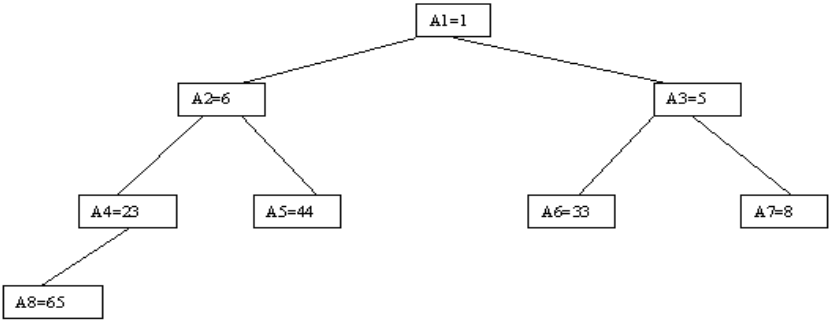

а)


A1=8
A2=23
A3=1
A4=6
A5=44
A6=33
A7=5
A8=65

б)


A1=8
A2=6
A3=1
A4=23
A5=44
A6=33
A7=5
A8=65

в)


A1=1
A2=6
A3=5
A4=23
A5=44
A6=33
A7=8
A8=65

г)

25 сурет.

1964 жылы Флойд пирамиданы құрудың ағашсыз тәсілін ұсынған болатын (әдістің мағынасы жоғарыда айтылғандай). Біздің стандартты массивімізге арналған Флойд әдісі арқылы пирамиданы құру 7-ші кестеде көрсетілген.

7-кесте. Пирамиданы құру

| | |
|---|---|
| **Массивтің бастапқы күйі** | 8 23 5 \|65\| 44 33 1 6 |
| **1-қадам** | 8 23 \|5\| 6 44 33 1 65 |
| **2 қадам** | 8 \|23\| 1 6 44 33 5 65 |
| **3 қадам** | \|8\| 6 1 23 44 33 5 65 |
| **4 қадам** | 1 6 8 23 44 33 5 65<br>1 6 5 23 44 33 8 65 |

8-кестеде құрылған пирамидамен қалай сұрыпталу жүретіні келтірілген. Алгоритм мағынасы мынада. і – пирамиданың шарттары орындалатын массивтің ең үлкен индексі болсын. Онда, а[1]-ден бастап а[і]-ге дейін келесі әрекеттер орындалады. Әр қадамда пирамиданың соңғы элементі таңдалады (біздің жағдайымызда бірінші болып а[8]-ші элемент таңдалады). Оның мәні а[1]-іншінің мәнімен алмастырылады, сонан кейін, а[1] үшін сүзу орындалады. Сол кезде, әрбір қадамда пирамидадағы элемент саны 1-ге кеміп отырады (бірінші қадамнан кейін пирамиданың элементі ретінде а[1], а[2], ..., а[n-1] қарастырылады; екіншіден кейін - а[1], а[2], ..., а[n-2] және т.с.с., пирамидада бір элемент қалғанша орындалады). Нәтижесінде, кему ретімен реттелген массив аламыз (8-кестені қараңыз). Егер пирамида шартын а[і] >=а[2*і] және а[1]>=а[2*i+1] (барлық і индексті мәндер үшін) өзгертсек, онда өсу ретімен реттелген массив аламыз.

8-кесте. Пирамиданы құру

| | |
|---|---|
| **Берілген пирамида** | 1 6 5 23 44 33 8 65 |
| **1-қадам** | 65 6 5 23 44 33 8 1<br>5 6 65 23 44 33 8 1<br>5 6 8 23 44 33 65 1 |
| **2-қадам** | 65 6 8 23 44 33 5 1<br>6 65 8 23 44 33 5 1<br>6 23 8 65 44 33 5 1 |
| **3-қадам** | 33 23 8 65 44 6 5 1<br>8 23 33 65 44 6 5 1 |
| **4-қадам** | 44 23 33 65 8 6 5 1<br>23 44 33 65 8 6 5 1 |
| **5-қадам** | 65 44 33 23 8 6 5 1<br>33 44 65 23 8 6 5 1 |
| **6-қадам** | 65 44 33 23 8 6 5 1<br>44 65 33 23 8 6 5 1 |
| **7-қадам** | 65 44 33 23 8 6 5 1 |

Ең нашар жағдайда, пирамида арқылы сұрыптау процедурасы $n*\log_2 n$ қадамдарды талап етеді. Сондықтан бұл әдісті үлкен массивтерді сұрыптауға қолдану тиімді.

*Мысалы. Пирамида арқылы сұрыптауға программа*

```
#include <iostream.h >
#include <conio.h>
usingnamespace std;
void max_heapify(int*a, int i, int n){
int j, temp;
temp= a[i];   j =2*i;
while(j <= n){
if(j < n && a[j+1]> a[j])   j = j+1;
if(temp > a[j]) break;
else if(temp <= a[j]){
          a[j/2]= a[j];
            j =2*j;}}
a[j/2]= temp;
return;}
void heapsort(int*a, int n){
int i, temp;
for(i = n; i >=2; i--){
temp= a[i]; a[i]= a[1]; a[1]= temp;
        max_heapify(a, 1, i -1);}}
void build_maxheap(int*a, int n){
int i;
for(i = n/2; i >=1; i--){
        max_heapify(a, i, n);}}
int main(){
int n, i, x; int a[20];
cout<<"enter no of elements of array\n";
cin>>n;
for(i =1; i <= n; i++){
cout<<"enter element"<<(i)<<endl;
cin>>a[i];}
     build_maxheap(a,n);
heapsort(a, n);
cout<<"sorted output\n";
for(i =1; i <= n; i++){
cout<<a[i]<<endl;}
getch();
}
```

*Біріктіріп сұрыптау*

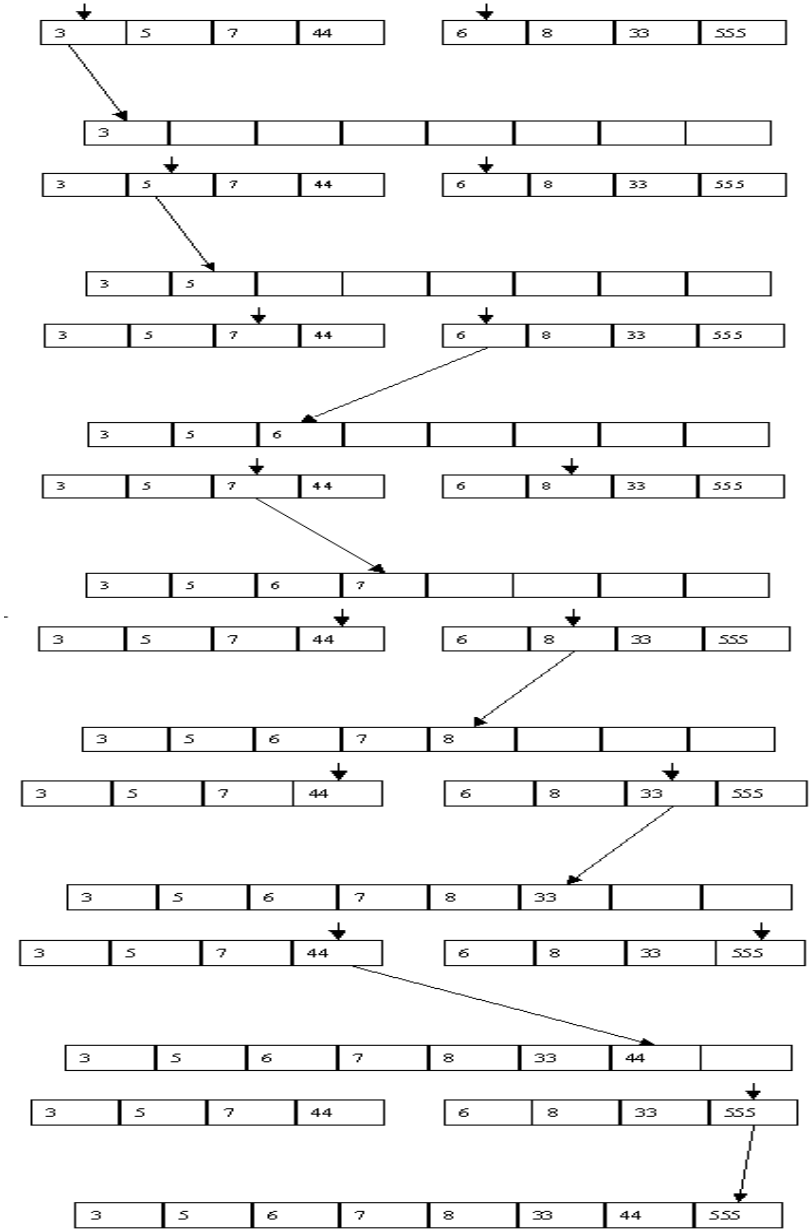
Біріктіріп сұрыптау әдісі, әдетте, негізгі жадыға толығымен сыймай қалған файлдарды сұрыптауға қолданады.

Ішкі сұрыптаудың әйгілі алгоритмдерінің бірі біріктіріп сұрыптау алгоритмі келесі идеяға сүйеніп жасалған (біздің жоғарыдағы мысалымыздағыдай, массивтегі элементтер саны 2 дәрежесіне тең деп қарастырамыз). Біріншіден, біріктіру ұғымына түсінік берейік. Өсу ретімен сұрыпталған p[1], p[2], …,p[n] және q[1], q[2],…,q[n] екі массив, сонымен қатар, r[1], r[2],…r[2*n] бос массив (p және q массивтердің элементтерімен өсу ретімен орналастырып толтыратын массив) берілсін.

Біріктіру үшін келесі әрекеттер орындалады: өзара p[1] және q[1] салыстырылады, және кішісі r[1]-ге жазылады. Мысалы, кішісі p[1] деп ойлайық. Онда, келесі p[2] q[1]-мен салыстырылып, кішісі r[2]-ге жазылады. Мысалы, кішісі q[1] деп есептейік. Онда, келесі қадамда p[2] және q[2] салыстырылады және т.с.с. Бұл процесс, кемінде бір массивтің соңғы шекарасына жеткенше орындалады. Сонда, элементі артық болып қалған массивтің «құйрығы» r массивіне тіркеледі. Біріктіріп сұрыптау алгоритміне мысал 26-суретте көрсетілген.

Берілген a[1], a[2], ..., a[n] массивіне біріктіріп сұрыптауды қолдану үшін қосымша b[1], b[2], ..., b[n] массиві қолданылады. Бірінші қадамда a[1] және a[n] элементтері біріктіріліп, нәтижесі b[1], b[2]-ге жазылады, ал a[2] және a[n-1]-дің бірігу нәтижесі b[3], b[4]-ке, ..., a[n/2] және a[n/2+1]-дің бірігу нәтижесі b[n-1], b[n]-ге орналасады. Екінші қадамда b[1], b[2] және b[n-1], b[n] жұп элементтер бір–бірімен бірігіп нәтижелері a[1], a[2], a[3], a[4]-ке орналастырылады. b[3], b[4] және b[n-3], b[n-2] бірігу нәтижесі a[5], a[6], a[7], a[8]-ге, ..., b[n/2-1], b[n/2] және b[n/2+1], b[n/2+2] жұптық бірігу нәтижесі a[n-3], a[n-2], a[n-1], a[n] т.с.с. орналасады. Соңғы қадамда, мысалы, (n-нің мәніне байланысты) ұзындығы n/2 массивтің элементтер тізбегі біріктіріледі a[1], a[2], ..., a[n/2] және a[n/2+1], a[n/2+2], …, a[n] нәтижелері b[1], b[2], ..., b[n]-ге орналастырылған.

Біздің мысалда қолданылатын массивтің қадамдар реті 9-кестеде көрсетілген.


3 5 7 44
6 8 33 555
3
3 5 7 44
6 8 33 555
3 5
3 5 7 44
6 8 33 555
3 5 6
3 5 7 44
6 8 33 555
3 5 6 7
3 5 7 44
6 8 33 555
3 5 6 7 8
3 5 7 44
6 8 33 555
3 5 6 7 8 33
3 5 7 44
6 8 33 555
3 5 6 7 8 33 44
3 5 7 44
6 8 33 555
3 5 6 7 8 33 44 555

26 сурет.

9-кесте. Біріктіріп сұрыптау алгоритмі

| | |
|---|---|
| **Массивтің бастапқы күйі** | 8 23 5 65 44 33 1 6 |
| **1-қадам** | 6 8 1 23 5 33 44 65 |
| **2 қадам** | 6 8 44 65 1 5 23 33 |
| **3 қадам** | 1 5 6 8 23 33 44 65 |

Біріктіріп сұрыптауды қолданған кезде кілттік салыстырулар саны және сілтемелер саны $O(n*\log_2 n)$-ге бағаланады. n өлшемді массивке сұрыптау алгоритмін қолданған кезде жадының 2*n элементтері керек екенін ескеру керек.

## 3.2 Сыртқы сұрыптау алгоритмдері

Сыртқы жадыда орналасқан және өте үлкен (файлды бүтіндей негізгі жадыға сыйғыза алмай, сонымен қатар, алдыңғы тақырыпта қарастырылған ішкі сұрыптаудың бір әдісін қолданып сұрыптау тиімсіз) файлдарға қолданылатын сұрыптау әдісін «сыртқы» сұрыптау дейді. Көбінесе, сыртқы сұрыптау әдістері деректер қорын басқару жүйелерінде (ДҚБЖ) сұраныстарды орындағанда қолданылады, және қолданылған әдістердің тиімділігінен ДҚБЖ-ның өнімділігі тәуелді болады.

Сыртқы сақтау құрылығылары ретінде магниттік ленталар көп қолданылған кезде сыртқы сұрыптау әдістері пайда болды. Ленталар үшін дәйекті қол жеткізу кәдімгі жағдай еді. Тұтынушылар, сақтау құрылғысы ретінде магниттік дискіге өткенде (ақпараттың кез келген бөлігіне «тікелей» қол жеткізуді

қамтамасыз ету) файлдарға дәйекті қол жеткізу өзекті болмай қалды деп ойлады. Бұл қате көзқарас болды.

Магниттік дискідегі ақпаратты оқу үшін магниттік бастиек (магнитная головка) дискідегі ақпараттың басы орналасқан жолға (дорожка) жылжиды. Магниттік бастиекті қажетті жолға жылжыту үлкен уақытты қажет етеді. Осы кеткен уақыт операцияны орындауға кеткен жалпы уақытты анықтайды. Уақытты үнемдеу үшін магниттік бастиектің оқитын файл бөліктері тізбектей (бірінен соң бірі реттеліп) орнласуы керек. Олай болу үшін сыртқы сұрыптау әдісін қолдану керек. Сұрыптауға қажетті осындай файлдармен заманауи ДҚБЖ–лар жұмыс жасайды.

Сонымен қатар, сыртқы сұрыптаудың орындалу жылдамдығы басты жадыдағы буфердің (немесе буферлердің) көлеміне байланысты.

*Тікелей біріктіру*

Алдыңғы тақырыптың соңында қарастырған қарапайым біріктіріп сұрыптау алгоритмін сыртқы сұрыптауға қолданып көрейік. Тізбектей орналасқан A файлын қарастырайық, a1, a2, ..., an жазуынан тұратын (түсінікті болуы үшін n-ның дәрежесін 2 деп есептейік). Әр жазба бір ғана сұрыптау кілтін көрсететін элеметтен тұрсын. Сұрыптау үшін екі көмекші файл B және C қолданылады (әрқайсысының өлшемі n/2).

Сұрыптау қадамдардың кезектестігінен тұрады және әр қадамда A файлы B және C файлдарына бөлінеді, содан кейін B және C файлдары A файлына біріктіріледі (біріктіру процедурасы 26 суретте көрсетілген). Бірінші қадамда, A файлы оқылып ретімен оқылып, a1, a3, ..., a(n-1) жазбалары B файлына, ал a2, a4, ..., an жазбалары C файлына жазылады (бастапқы үлестірілім). Бастапқы біріктіру (a1, a2), (a3, a4),...,(a(n-1), an) жұптарына жүргізіліп, нәтижесі A файлына жазылады. Екінші қадамда A файлы қайта оқылып, B файлына тақ нөмерлі жұптар жазылып, ал C файлына жұп нөмерлі жұптар жазылады. Біріктіру кезінде бірыңғай төрттік жазбалар A файлына жазылады, т.с.с. Соңғы қадам орындалар алдында A файлы,

әрқайсысы n/2 өлшемнен тұратын, тізбекті құрайды. Оларды таратқанда біреуі B файлына, ал екішшісі C файлына түседі. Қосудан кейін, A файлы толығымен біріңғай реттеліп орналасқан жазулар тізбегінен тұрады. 10-кестеде сыртқы сұрыптаудың тікелей біріктіру әдісіне мысал көрсетілген.

10-кесте. Сыртқы сұрыптаудың тікелей біріктіру әдісі алгоритмі

| | |
|---|---|
| **A файлының бастапқы күйі** | 8 23 5 65 44 33 1 6 |
| **Бірінші қадам**<br>**Үлестіру**<br>**B файлы**<br>**C файлы**<br>**Біріктіру: A файлы** | <br><br>8 5 44 1<br>23 65 33 6<br>8 23 5 65 33 44 1 6 |
| **Екінші қадам**<br>**Үлестіру**<br>**B файлы**<br>**C файлы**<br>**Біріктіру: A файлы** | <br><br>8 23 33 44<br>5 65 1 6<br>5 8 23 65 1 6 33 44 |
| **Үшінші қадам**<br>**Үлестіру**<br>**B файлы**<br>**C файлы**<br>**Біріктіру: A файлы** | <br><br>5 8 23 65<br>1 6 33 44<br>1 5 6 8 23 33 44 65 |

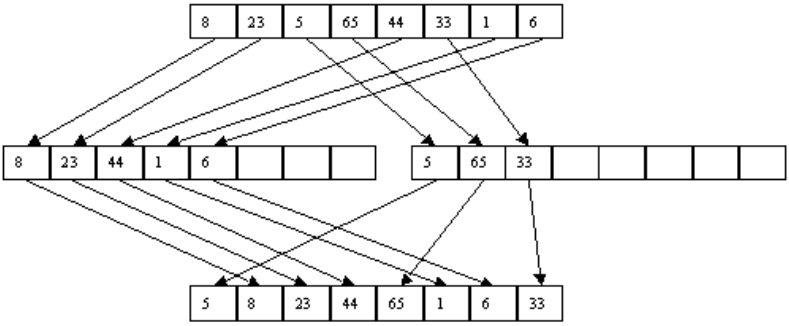
Сыртқы сұрыптауды тікелей біріктіру әдісімен орындағанда, басты жадыда тек екі айнымалы, яғни кезекті B және C

файлдарының жазбаларын орналастыру керек. A, B және C файлдары O(log n) рет оқылады және сонша рет жазылады.

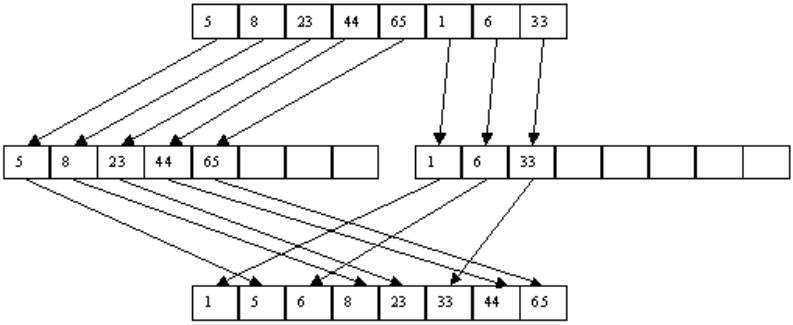
*Табиғи біріктіру*

Тікелей біріктіру әдісін қолданғанда ағымдағы файл ішінара сұрыпталғаны, яғни бірыңғай реттелген жазбалардан тұратыны ескерілмейді. Серия деп ai, a(i+1), ..., aj жазулар тізбегін айтады, ak<=a(k+1) шартты қанағаттандыратын, барлық i<=k<j, ai <a(i-1) және aj>a(j+1) үшін орындалатын. Табиғи сұрыптау әдісі серияны бөлу барысында тану және оларды келесі біріктіру барысында қолдануға сүйенеді.

Тікелей біріктіру әдісіндегі сияқты бұл сұрыптау да бірнеше қадамдардан тұрады және әр қадамда бірінші A файлы B және C файлына бөлінеді, кейін B және C файлдары A файлына бірігеді. Бөлу барысында жазбаның бірінші сериясы анықталып, B файлына жазылады, ал екіншісі - C файлына, т.с.с.. Біріктіруде, B файлының бірінші жазбалар сериясы C файлының бірінші сериясымен бірігеді, B-ның екінші сериясы, C-ның екінші сериясымен, т.с.с. орындалады. Егер бір файл сериясы екіншіге қарағанда бірінші аяқталса (сериялар саны әртүрлі болуға байланысты), қаралмай қалған файлдың бөлігі толығымен A файлының соңына көшіріледі. A файлында тек бір ғана серия қалғанда процесс аяқталады. Файлды сұрыптау мысалы 27-ші және 28-ші суреттерде көрсетілген.




27-сурет. Бірінші қадам.



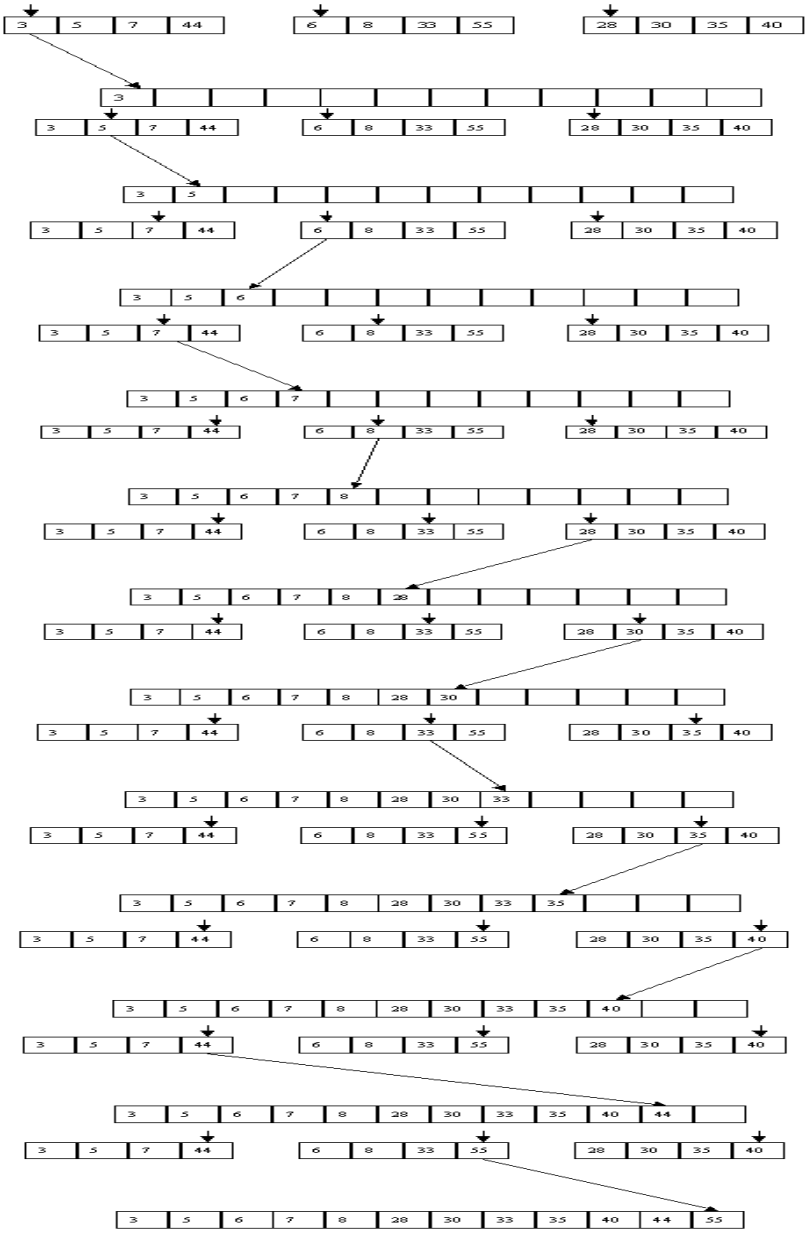

28 сурет. Екінші қадам.

Осы әдісті қолданған кезде файлдағы қайта жазу және оқулар саны тікелей біріктіру әдісін қолдануға қарағанда тиімдірек. Бір жағынан, серияның соңын танудың әсерінен салыстырулар саны көбейеді. Сонымен қатар, серияның ұзындығы кез келген болатындықтан, В және С файлдарының максималды ұзындығы А файлының ұзындығына жақын болуы мүмкін.

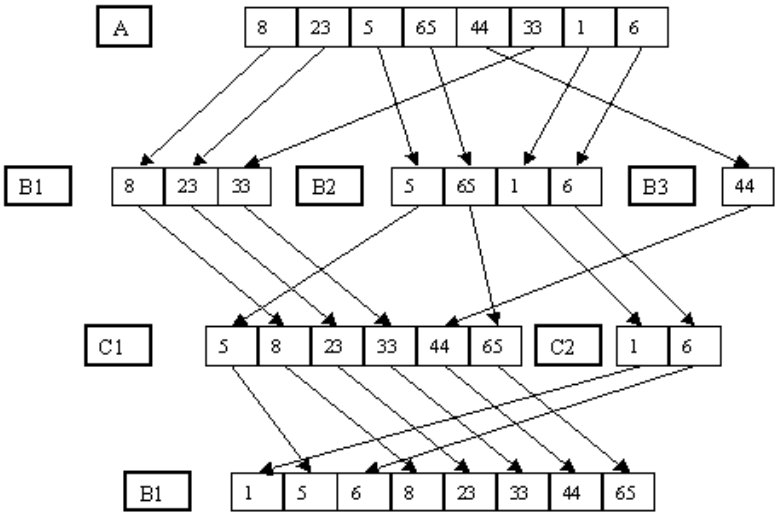
*Теңестірілген көпжолды біріктіру*

Сыртқы сұрыптаудың теңестірілген көпжолды біріктіру әдісінде ағымдағы файл серияларын m көмекші $B_1$, $B_2$, ..., $B_m$ файлдарына бөлу және оларды m көмекші $C_1$, $C_2$ ,...,$C_m$ файлдарына біріктіру болып табылады. Келесі қадамда $C_1$, $C_2$, ..., $C_m$ файлдарын $B_1$, $B_2$,...,$B_m$ файлдарына біріктіру жүргізіледі және т.с.с., $B_1$ және $C_1$-лер бір серия құрағанша.

Көпжолды біріктіру қарапайым (екіжолды) біріктірудің табиғи даму идеясы болып табылады (26-суретте көрсетілген). Үшжолды біріктіруге мысал 29-суретте көрсетілген.




29-сурет.




30-сурет.

30-суретте көпжолды біріктіріп сұрыптауға қарапайым мысал көрсетілген. Серияның ұзындығы ұлғайған сайын үлкен нөмірлі (n нөмерінен бастап) көмекші файлдар қолданылудан қалады, себебі, бір серияда «бұйырмайды». Теңестірілген көпжолды біріктіріп сұрыптаудың артықшылығы сонда, оның өту саны $O(\log_n n)$-ге бағаланады (n файлдағы жазбалар саны ). Әрине, салыстырулар саны қарапайым біріктіру әдісін қолданғаннан кем болмайды.

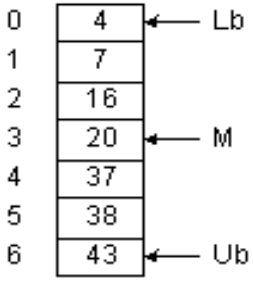
# 4. ІЗДЕУ АЛГОРИТМДЕРІ

Іздеу, жазу, өшіру – деректермен жұмыс жасаудағы негізгі операциялар. Айтылған операциялар массивтер және тізімдер (байланысқан) арқылы қалай жүзеге асатынын қарастырайық. Әрбір элементтің, өзіндік бір ақпаратынан басқа, сәйкес келетін кілті берілген [1].

*Сызықты іздеу алгоритмі*

Берілген реттелген массивтен керек элементті (кілтті) табу үшін сызықты (тізбектей) іздеу алгоритмін қолдануға болады. Ол реттелген немесе реттелмеген массивтермен жұмыс жасайды. Бұл алгритмдерден де тиімділері бар, бірақ сызықты іздеу алгоритмі қарапайымдылығымен және реттелмеген массивтерге қолдануға болатындығымен тартымды.

Сызықты іздеу алгоритмінде кілтті (керек) элемент, массивте орналасқан алғашқы элементтен бастап, рет-ретімен салыстырылып шығады. Егер қандай да бір қадамда кілттері тең элемент кездессе, онда массивтен керек элемент табылды деп есептеледі және нәтижесі ретінде сол элементтің мәні, кілті (орналасқан реті, орны) немесе басқа да ақпараттары алынады. Басқа жағдайда, «массивте ондай элемент жоқ» деген ақпарат алынуы керек




31-суретте көрсетілгендей, 7 элементтен тұратын сандық мәні бар массив берілген. Осы массивтен керек санды табу үшін, оған сызықты іздеу алгоритмін қолдану төмендегі программаның функциясында көрсетілген. Бұл іздеу алгоритмінде максималды салыстыру саны 7-ге тең және ол бізге A[6]-ға жеткенде тоқтайды.

31-сурет. Массив

*Мысалы*. Сызықты іздеу

```
int function SequentialSearch (Array A, int Lb, int Ub, int Key);
  begin
  for i = Lb to Ub do
    if A(i) = Key then
      return i;
    return -1;
```

*Екілік (бинарлы) іздеу алгоритмі (қақ бөліп іздеу, дихотомия)*

Егер деректер сұрыпталған болса, онда екілік іздеу алгоритмін қолдануға болады (төмендегі мысалды қараңыз). *Lb* және *Ub* айнымалылары біздің іздеп отырған элементіміз бар массивтің сол және оң жақ шекаралары. Қарастырылып отырған массивтің орта (31-суретте M) элементін табудан бастаймыз. Егер іздеген элементіміз орта элементтен кіші болса, массивтің жоғарғы бөлігіне, ал үлкен болса – төменгі бөлігіне өтеміз (31-сурет). Басқаша айтқанда, *Ub*-дің мәні (M-1) болады және келесі итерацияда біз массивтің жарты бөлігімен жұмыс жасаймыз. Осылайша, әрбір итерация нәтижесінде іздеу аумағын екі есеге азайтып отырамыз. Біздің мысалымызда, бірінші итерациядан кейін іздеу аумағымызда бар болғаны үш элемент қалады, екінші жағдайда – бір-ақ элемент қалады. Егер массивтің ұзындығы 6-ға тең болса, онда бізге қажет санды табу үшін 3 итерация керек (31-сурет).

Екілік іздеу – өте тиімді әдіс. Мысалы, массивтің ұзындығы 1023 тең болса, онда бірінші салыстырудан кейін 511 элементке дейін қысқарады. Ал екінші іздеуден кейін 255-ке дейін. 1023 элементтен тұратын массивті іздеу үшін 10 салыстыру ғана жеткілікті.

*Мысалы*. Екілік іздеу

```
int function BinarySearch (Array A, int Lb, int Ub, int Key);
  begin
  do forever
```

```
    M = (Lb + Ub)/2;
    if (Key < A[M]) then
      Ub = M - 1;
    else if (Key > A[M]) then
      Lb = M + 1;
    else
      return M;
    if (Lb > Ub) then
    return -1;
  end;
```

*Мысалы:*

```
#include <cstdlib.h>
#include <iostream.h>
using namespace std;
int binary_search(int array[],int first,int last, int value);
int main() {
int list[10];
 for (int k=0; k<11; k++)
 list[k]=2*k+1;
 cout<< "binary search results: "<<
binary_search(list,1,21,11)<<endl;
 return 0;
}//end of main
int binary_search(int array[],int first,int last, int search_key){
 int index;
 if (first > last)
 index = -1;
 else  {  int mid = (first + last)/2;
 if (search_key == array[mid])  index = mid;
 else  if (search_key < array[mid])  index = binary_search(array,first,
mid-1, search_key);
 else  index = binary_search(array, mid+1, last, search_key);
 } // end if
 return index;
 }// end binarySearch
```

Екілік іздеу алгоритмі тиімді, өйткені әрбір қадам сайын интервалы n/2-ден бастап $n^2$-ге дейін қысқарады және орташа $\log_2(\log_2(n))$ қадамды талап етеді, егер деректер алдын ала реттелген болса.

Іздеуден басқа бізге элементтерді қоюға және жоюға тура келеді. Өкінішке орай, ол операцияларды орындауға массив бейімсіз. Мысалы, 31-суретте берілген массивке 18 санын қою үшін A[3]: A[6] элементерін төмен жылжыту керек, содан соң ғана 18 санын A[3]-ші элементке қоюға болады. Осы сияқты мәселелер элементтерді жоюда пайда болады. Қою/Жою операцияларын тиімді пайдалану үшін бір байланысты тізімдерді қолднау ұсынылады.

*Интерполияциялық іздеу алгоритмі*

Интерполяциялық іздеу алгоритмінің негізінде интерполяциялау операциясы жатыр. Интерполяциялау – берілген дискретті белгілі мәндер жиынының аралық мәндерін табу. Интерполяциялық іздеу тек реттелген массивтермен ғана жұмыс жасайды. Ол екілік (бинарлық) іздеуге ұқсас, яғни әр қадамда іздеу аумағы алгоритмнің орындалғанына байланысты қысқарып отырады. Екілік іздеуден айырмашылығы, интерполяциялық іздеу тізбекті бірдей екі бөлікке бөлмейді, ізделінді және ағымдағы элементтердің ара қашықтығына байланысты кілттің (ізделінді элементтің) орналасу орнын жуықтап есептейді. Мысалы ретінде телефон кітапшасынан керек нөмірді іздеуді келтіруге болады. Телефон кітапшасындағы деректер (алфавит әріптері) өсуі бойынша реттелген. Егер тегі «Х» әрпінен басталатын абонент керек болса, онда іздеу үшін кітапшаның соңына өтеміз.

Интерполяциялық іздеуді анықтайтын алгоритм келесі түрдегі формула арқылы сипаталады:

$$mid = left + ((key - A[left]) * (right - left)) / (A[right] - A[left])$$

Бұл формуладағы mid – кілттің мәні салыстырылатын элемент нөмірі, key – кілт (ізделініп отырған элемент), A – реттелген элементтерден тұратын массив, left және right – іздеу

аумағындағы шеткі элементтер нөмірі. Формуладағы бөлу амалының нәтижесі бүтін сан, яғни бөлшек бөлігі, қандай болса да, алынып тасталады.

*Мысалы*. Интерполяциялық іздеу алгоритміне программа

```
#include <stdafx.h>
#include <iostream.h >
using namespace std;
const int N=17;
//интерполяциялық іздеу
int InterpolSearch(int A[], int key) {
int mid, left=0, right=N-1;
while (A[left]<=key && A[right]>=key) {
mid=left+((key-A[left])*(right-left))/(A[right]-A[left]);
if (A[mid]<key) left=mid+1;
else if (A[mid]>key) right=mid-1;
else return mid;}
if (A[left]==key) return left;
else return -1;}
//негізгі функция
void main() {
setlocale(LC_ALL,"Rus");
int i, key;
int A[N]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59};
cout<<"Ізделінді элемент > "; cin>>key;    //кілтті енгізу
cout<<"Ағымдағы массив: ";
for (i=0; i<N; i++) cout<<A[i]<<" "; //массивті шығару
if (InterpolSearch(A, key)==-1) cout<<"\n Элемент табылмады";
else cout<<"\n Элемент нөмірі: "<<InterpolSearch(A, key)+1;
system("pause>>void");
}
```

# 5. РЕКУРСИЯ ЖӘНЕ РЕКУРСИВТІ АЛГОРИТМДЕР

*Рекурсия* – көмекші программаның (функция немесе процедура) операторлары орындалу барысында өзін-өзі шақыруды ұйымдастыратын көмекші алгоритм. Рекурсивті деп өзі арқылы анықталатын объектіні айтады [1,5]. Рекурсивті анықталуда өзін-өзі шақыруды тоқтататын рекурсивті шарт болуы керек. Рекурсияның екі түрі бар: төте рекурсия және жанама рекурсия.

## 5.1 Төте рекурсия

Төте рекурсия деп функция денесінде өзін-өзі шақыруды айтамыз.

*Мысалы:*

```
int a()
{.....a().....}
```

*Мысалы:* n! есептеуге рекурсивті алгоритмді қолданып программа құру.

F(n)=n! есептеуді цикл арқылы орындауға болады. Бұл жағдайда факториал n!=1*2*3*…*n арқылы өрнектеледі, яғни факториалды есептеуде цикл басынан соңына дейін орындалады. Осы есепті рекурсивті алгоритмді қолданып есептеуде рекурсивті формула, рекурсивті шарт болуы керек. Факториалды рекурсивті алгоритм арқылы есептегенде мәндер соңынан басына қарай көбейтіліп орындалады F(n)=n*(n-1)*…*1, 0!=1. Функцияның рекурсивті анықталуы

$$F(n)=\begin{cases}1, n=0\\ n*F(n-1), n>0\end{cases}$$

мұндағы F(n-1)=(n-1)!

6! - алты факториялды есептеу алгоритміне программаны жазайық:

```
#include <stdio.h>
double fact(int n);
int n=6;
void main(){
f=fact(n);
printf('6!=%10.0f\n",f);
return 0;
}
double fact(int n){
double f;
if (n<=1) f=1.; else
f= fact(n-1)*n;
return f;
}
```

*Мысалы.* Сызықты массив элементтерінің қосындысын есептейтін программа

```
#include <stdio.h>
#include <conio.h>
#include <stlib.h>
# include <time.h>
   int summa (int N, int a[100]);
 int i,n, a[100];
void main () {
clrscr ();
printf ("\n массивтегі элементтер саны"); scanf ("%d", &n);
printf ("\n  массивті құру %d сан :\n; n");
randomize ();
for (i=0; i<n; i++)
{ a[i]=-10+random (21);
printf ("%d ", a[i]);
printf ("сумма %d"; summa (n-1, a))
}
int summa (int N, int a[100]) {
If (N==0) return a[0];
```

```
else return a[N]+summa (N-1,a)
}
```

*Мысалы.* Берілген қатар полиндром болатынын, яғни солдан оңға және оңнан солға бірдей оқылатын, анықтайтын программа

```
 #include <stdio.h>
 #include <conio.h>
 #include <string.h>
char s[100];
int pal(char s[100]);
void main()
{ clrscr();
  printf("\n Қатарды енгізіңіз: "); gets(s);
  if (pal(s)) printf("Қатар палиндром болады");
     else  printf("Қатар палиндром болмайды");
}
int pal(char s[100])
{ int l; char s1[100];
  if (strlen(s)<=1) return 1;
  else {l=s[0]==s[strlen(s)-1];
        strncpy(s1, s+1, strlen(s)-2);
        s1[strlen(s)-2]='\0';
        return l && pal(s1);}
}
```

Келесі мысал ретінде Аккерман функциясын есептеуді рекурсивті алгоритмді қолданып қарастырайық.

*Мысалы.* Аккерман функциясын есептеу программасын есептеуін жүзеге асыру нұсқасының бірнеше түрін қарастырамыз:

$$A(N,X,Y) = \begin{cases} X+1, & \text{егер } N=0 \\ X, & \text{егер } N=1,\ Y=0, \\ 0, & \text{егер } N=2,\ Y=0, \\ 1, & \text{егер } N=3,\ Y=0, \\ 2, & \text{егер } N=>4,\ Y=0, \\ A(N-1,A(N,X,Y-1),X), & \text{егер } N\#0,\ Y\#0; \end{cases}$$

Мұндағы N,X,Y – бүтін оң сандар.

Төмендегі программада Аккерман функциясын есептеу үшін ackr рекурсивті функциясы және smacc көмекші функциясы қолданылады:

```
/*     Аккерман фнкциясын рекурсивті есептеу     */
     # include
     main ()                    /*  шақыру      */
     {   int x,y,n,t;           /*  функциясы */
         int ackr(int, int, int);
         scanf("%d %d %d",&n,&x,&y);
         t=ackr(n,x,y);
         printf("%d",t);
     }
    int smacc( int n,int x )      /*  көмекші    */
    {   switch (n)                /*   функция  */
        {  case 0:  return(x+1);
           case 1:  return (x);
           case 2:  return (0);
           case 3:  return (1);
           default: return (2);
         }
    }
    int ackr( int n, int x, int y)    /*  рекурсивті     */
    {  int z;                         /*  функция        */
       int smacc(int,int);
       if(n==0 || y==0)  z=smacc(n,x);
       else { z=ackr(n,x,y-1);        /*  ackr(...) рекурсивті     */
              z=ackr(n-1,z,x);  }     /*  шақыру                   */
       return z;
    }
```

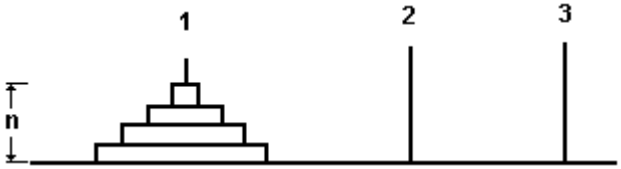
main және ackr функцияларын жетілдіру арқылы жоғарыда айтылған әдіс арқылы келесі программаны аламыз:

```
 /*Аккерман функциясын есептеу үшін эквивалентті рекурсивті
емес программа*/
    #include
    #include
    int main()
    {  typedef struct st
       { int i,j,k,z,lr;
         struct st *pst;
       } ST;
       ST *u, *dl=NULL;
       int l,x,y,n;
       int  smacc(int,int);
       int an,ax,ay,rz,t;
       scanf("%i %i %i",&n,&x,&y);

       an=n;ax=x;ay=y;l=1;       /*  -   шақыруды ауыстыру    -  */
       goto ackr;                /*     t=ackr(n,x,y);                */
   l1: t=rz;                     /*  - - - - - - - - - - - - - - - - */

       printf("\n %d ",t);
       goto jackr;

       /*    ackr функциясын ауыстыратын фрагменттің басы    */
    ackr:
       u=( ST *) malloc( sizeof (ST) );
       u->i=an;
       u->j=ax;
       u->k=ay;
       u->lr=l;
       u->pst=dl;
       dl=u;
       if (an==0||ay==0)
       dl->z=smacc(an,ax);
       else
           {
             an=dl->i;        /*  -   шақыруды ауыстыру    -  */
             ax=dl->j;        /*                                */
             ay=dl->k-1;      /*       z=ackr(n,x,y-1);          */
```

```
        l=2;              /*                              */
        goto ackr;        /*                              */
    l2: dl->z=rz;         /* - - - - - - - - - - - - - - - */

        an=dl->i-1;       /* -  шақыруды ауыстыру  -  */
        ax=rz;            /*                              */
        ay=dl->j;         /*        z=ackr(n-1,z,x);      */
        l=3;              /*                              */
        goto ackr;        /*                              */
    l3: dl->z=rz;         /* - - - - - - - - - - - - - - - */
      }
  rz=dl->z;               /* - - - - - - - - - - - - - - - */
  an=dl->i;               /*                              */
  ax=dl->j;               /*            return z          */
  ay=dl->k;               /*                              */
  l=dl->lr;               /*          операторын          */
  u=dl;                   /*                              */
  dl=u->pst;              /*          ауыстыру;           */
  free(u);                /*                              */
  switch(l)               /*                              */
      {  case 1: goto l1;  /*                              */
         case 2: goto l2;  /*                              */
         case 3: goto l3;  /*                              */
      }                   /* - - - - - - - - - - - - - - - */
jackr:
}
int smacc( int n,int x )      /*        көмекші функция      */
{  switch (n)
      { case 0:  return(x+1);
        case 1:  return (x);
        case 2:  return (0);
        case 3:  return (1);
        default: return (2);
      }
}
```

*«Ханой мұнарасы» есебі*

Нөмірленген 1, 2, 3 үш қазық берілген. 1-ші қазықта әр түрлі диаметрлі (пирамида түрінде) n сақина орналасқан (32-сурет).



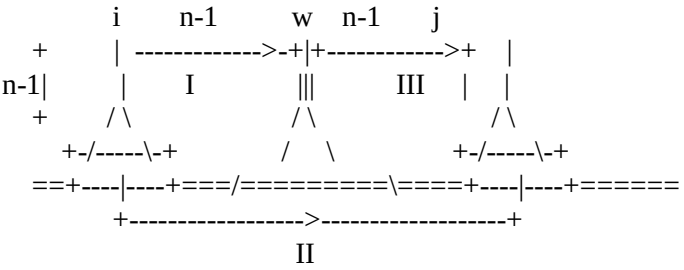

32-сурет. Ханой мұнарасының есебі.

Есептің шарты бойынша, 1-ші қазықтағы пирамида түрінде орналасқан n сақинаны, 2-ші қазықты пайдаланып, 3-ші қазыққа сондай пирамида түрінде жинау керек. Сақиналарды бір қазықтан екіншісіне ауыстыру барысында кіші диаметрлі сақинаның үстіне үлкен диаметрлі сақинаны қоюға болмайды. Бір ауыстырғанда тек бір сақина жылжиды. n сақина үшін $2^{n-1}$ алмастыру орындалатындығы дәлелденген. Осы есептің алгоритмін талдап, программасын С программалау тілінде құрайық.

Қарапайым жағдайда есепті шешетін болсақ, онда пирамида тек бір дискіден тұрып, бір ғана іс-әрекет орындайды, яғни і қазықтан j-ға 1 сақинаны көшіреміз (ол ауыстыру і->j деп белгіленеді). Есептің жалпы жағдайы 33-суретте көрсетілген. N сақинаны і қазықтан j қазыққа жинау үшін w қосалқы қазықты пайдаланамыз. Алдымен, n-1 сақинаны і қазықтан w қазығына j қосалқы қазықты пайдаланып көшіреміз. Содан кейін, і қазықтан бір сақинаны j-ға, және n-1 сақинаны w қазығынан j-ға, қосалқы і қазығын қолдана отырып, көшіреміз. Сонымен, n сақинаны ауыстыру есебіекі есептен тұрады: n-1 сақинаны ауыстыру және бір қарапайым есеп. Схема түрінде оны былай жазуға болады: T(n,i,j,w) = T(n-1,i,w,j), T(1,i,j,w), T(n-1,w,j,i).

Төмендегі мысалда программа келтірілген. Онда, n сақиналар санынан тұратын Ханой мұнарасы есебінің шешу программасында, n сақиналар саны енгізіліп, сол сақиналарды ауыстыру жолы көрсетілген ақпарат экранға шығады.

Ауыстыруды баспаға шығаратын tn(n,i,j,w) ішкі рекурсивті процедурасы, {i,j,w}={1,3,2} болғанда, n сақиналарды і қазықтан j қазыққа w қосалқы қазықты пайдаланып көшіруде қолданылады.

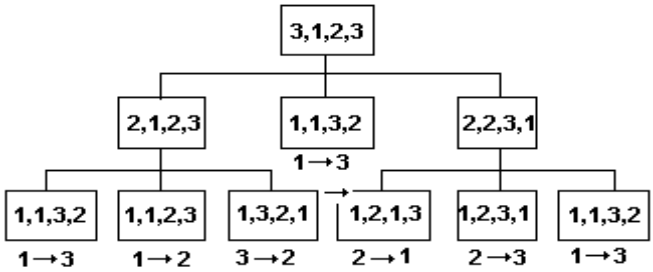



33-сурет. Ханой мұнарасы есебін шешу схемасы.

*Мысалы.* Ханой мұнарасы есебінің программасы

```
/*                Ханой мұнарасы                */
#include
main()                              /*  шақыру  */
{   void tn(int, int, int, int);      /*  функция     */
    int n;
    scanf(" %d",&n);
    tn(n,1,2,3);
 }
 void tn(int n, int i, int j, int w)   /*  рекурсивті  */
 {  if (n>1)                          /*  функция     */
     {  tn (n-1,i,w,j);
        tn (1,i,j,w);
        tn (n-1,w,j,i);
      }
     else printf(" \n %d -> %d",i,j);
     return ;
 }
```

n=3 болған жағдайдағы tn функциясын ретімен шақыру ағаш түрлі құрылымы 34-суретте көрсетілген.




34-сурет. tn функциясын ретімен шақыру.

tn функциясының әрбір шақырылуында n, i, j параметрлеріне жадыдан орын бөлінеді және қайтару орыны есте сақталады. tn функциясынан қайтқанда n, i, j, w параметрлеріне бөлінген жады босайды, және алдыңғы шақырудан n, i, j, w параметріне бөлінген жады қол жетімді болады, ал басқару қайтару орнына беріледі.

## 5.2 Жанама рекурсия

Жанама рекурсия деп функция денесінде тізбектей басқа функцияларды шақыруды айтамыз.

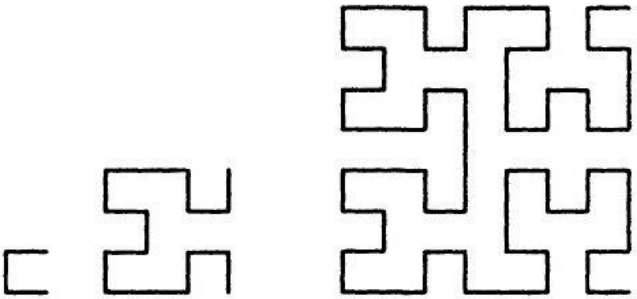
Кезекке тұрған барлық функцияларды да рекурсивті деп есептейміз.

*Мысалы:*

```
a(){.....b().....}
b(){.....c().....}
c(){.....a().....} .
```

Барлық a,b,c функциялары рекурсивті, өйткені олардың біреуін шақырғанда басқалары да шақырылады немесе өзін-өзі шақыру жүзеге асады.

Жанама рекурсияға мысал ретінде 35-суретте көрсетілген Гильберт қисығын сызатын программаны келтірейік.



35-сурет. Гильберт қисығы.

Екінші ретті қисық төрт бірінші ретті қисықтардың (екеуі сағат тіліне бағыттас 90 градусқа, ал қалған екеуі - сағат тіліне қарсы) құралған. Осы сияқты үшінші ретті қисық алынады, бірақ «кірпіштер» ретінде екінші ретті қисықтар қолданылады. Қисықты сызу реті программа орындалу барысында енгізіледі.

*Мысалы.* Гильберт қысығын сызатын программа

```
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define pi M_PI
void Draw (double x, double y, double r, double a)
{
int i;
double xx[6], yy[6];
for (i=0; i<6; i++)
{
xx[i] = r*cos(a+i*pi*2/5);
yy[i] = r*sin(a+i*pi*2/5);
}
for (i=0; i<5; i++)
```

```
line (x+xx[i], y+yy[i], x+xx[i+1], y+yy[i+1]);
}
void ProvRis (double x, double y, double r, double a, int d)
{
int i;
double h;
h = 2*r*cos(pi/5);
for (i=0; i<5; i++)
{
Draw (x - h*cos(a+i*pi*2/5), y - h*sin(a+i*pi*2/5), r, a+pi);
if ( d > 0 )
ProvRis (x - h*cos(a+i*pi*2/5), y - h*sin(a+i*pi*2/5),
r/(2*cos(pi/5)+1), a, d-1);
}
//Draw (x, y, r, a);
if ( d > 0 )
ProvRis (x, y, r/(2*cos(pi/5)+1), a+pi, d-1);
}
int main ()
{
int gm, gd=DETECT;
initgraph (&gd, &gm, "egavga.bgi");
ProvRis (320, 260, 95, pi/2, 3);
getch ();
closegraph ();
return 0;
}
```

# 6. ПОСТ ЖӘНЕ ТЬЮРИНГ МАШИНАСЫ

*Пост машинасы*

1936 жылдың қыркүйегіндегі «Журнал символической логики» нөмірінде жарық көрген фундаменталды мақалалардың бірі, Эмиль Посттың (Emil Post) «Финитные комбинаторные процессы, формулировка 1» мақаласының нәтижелері заманауи алгоритмдер теориясының негізін құрайды [4].

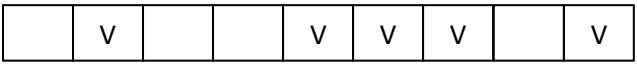
Пост көптеген нақты мәселелерден тұратын жалпы мәселені қарастырады, яғни жалпы мәселенің шешімі әр нақты мәселелердің шешімі болып табылады.

Мысалы, 3*x+9=0 теңдеуін шешу, бұл нақты мәселердің бірі, ал a*x+b=0 теңдеуін шешу бұл жалпы мәселе, яғни алгоритм («алгоритм» терминін Пост қолданбайды) жалпы мәселелерді шешетін әмбебап болуы керек.

Посттың алгоритм формализмінің жалпы түсінігі – бұл символдар кеңістігі (*L* тілі), онда нақты мәселелер беріледі және жауабы алынады, және ережелер жинағы, яғни, операциялардың өзін, сонымен қатар ережелердің орындалу ретін беретін символдар кеңістігіндегі операциялар.

Посттың символдар кеңістігі – бұл шексіз ленталар ұяшығы (жәшіктер).

Әрбір жәшік немесе ұяшық белгіленуі немесе белгіленбеуі мүмкін (36-сурет).




36-сурет

Нақты мәселе «сыртқы күшпен» (Пост термині) беріледі, санаулы ұяшықтың соңы белгіленген, яғни, кез келген конфигурация белгіленген жәшікпен басталады және аяқталады.

Нақты мәселеге белгілі бір ережелер жинағын қолданғаннан кейін жауабы сыртқы күшпен танылатын белгіленген және белгіленбеген жәшіктер жиынтығынан тұрады.

Пост «жұмысшы» орындайтын ережелер (элементарлы операциялар) жинағын ұсынды. 1936 жылы бірде-бір

электронды есептеу машинасы болған жоқ. Бұл ережелер жинағы биттік операциялардың минималды жинағы болып табылады:

1. егер жәшік бос болса, оны белгілейміз;
2. егер жәшікте белгі болса, онда белгіні өшіреміз;
3. сол жаққа 1 жәшікке жылжыту;
4. оң жаққа 1 жәшікке жылжыту;
5. жәшіктің белгіленген немесе белгіленбегенін анықтау, және нәтижесіне байланысты берілген 2 ереженің біріне өту;
6. тоқтау;

1 және 2 нұсқаулар құрылымы қате жағдайлардан қорғаудан тұрады.

Программа нөмірлеген нұсқаулар тізбегінен тұрады, 5-ші нұсқаудағы көрсетілген өтулер нөмірлері көрсетілген басқа нұсқауларға сілтейді.

Программа (Пост терминіндегі нұсқаулар жинағы) барлық нақты мәселелерге бірдей, сондықтан Пост әмбебаптық талапты тұжырымдайды.

Әрі қарай, Пост келесі түсініктерді еңгізеді:

- ережелер жиынтығы жалпы мәселелерге *қолданылады*, егер әрбір нақты мәселеде 1, 2 нұсқаумен қайшылық болмаса, яғни программа ешқашан бос жәшіктегі белгіні өшірмейді және белгіленген жәшікке белгі салмайды;
- нұсқау жинағы *аяқталады* (нұсқау саны шектеулі), егер (6) нұсқау орындалса;
- нұсқаулар жинағы *финитті 1 – процесті* береді, егер жинақ қолданымды, және әр нақты мәселеге арналып аяқталса;
- жалпы мәселе үшін финитті 1 – процесс 1-шешім болады, егер әрбір нақты мәселенің жауабы дұрыс болса (бұл сыртқы күшпен анықталады).

Пост бойынша мәселе лентаның шектеулі жәшіктер саны белгілеу арқылы сыртқы күшпен беріледі. Пост машинасы бірлік есептеу жүйесінде (0=V; 1-VV; 2=VVV; 3=VVVV) жұмыс

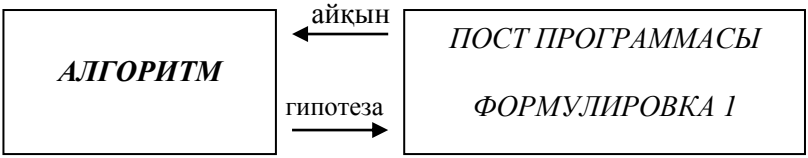
жасайды, яғни нөл бір беліленген жәшік түрінде, ал оң бүтін сан – оның мәнінен бірге артық белгіленген жәшікпен.

Жалпы мәселені құрайтын нақты мәселелер жиыны жұп, онда $N$ оң бүтін сандар жиыны және нақты мәселелер жиыны арасында өзара біркелкі қатынас (биективті бейне) орнатуға болады.

Жалпы мәселе Пост бойынша *1-берілді* деп аталады, егер $n \in N$-ге берілген жәшіктер конфигурациясы ретінде қолданылатын сондай финитті 1–процесс бар болса. Ол $n$-ші нақты мәселені белгіленген жәшіктер жиынтығы ретінде беріледі.

Егер жалпы мәселе 1-берілген және 1-шешілетін болса, онда берілген мәселелерге байланысты нұсқаулар жиынын оның шешуімен байланыстырсақ, біз мәселенің нөмірі бойынша жауап аламыз. Осы Посттың мақаласындағы терминдер *формулировка 1* деп аталады.

Эмиль Пост өзінің мақаласын келесі сөйлеммен аяқтайды: «Автор өзінің тұжырымдамасын Гедель-Черчтің тұжырымдамасының рекурсивтігіне логикалық эквивалентті болады деп түсінеді. Тұжырымдаманың мақсаты мынада: тек белгілі логикалық күшті ұсыну ғана емес, сонымен қатар психологиялық ақиқаттық. Соңғы ой кең ауқымды тұжырымдаманы қажет етеді. Басқаша айтқанда, біздің мақсатымыз олардың барлығы да логикалық формулировка 1-ге келетінін көрсету. Осы мезетте, біз ойымыздың қорытындысын жұмыс болжамы ретінде қарастырамыз. ... Жоғарыда көрсетілген программаның жетістігі, біз үшін осы болжамның анықтамаға немесе аксиомаға айналуында емес, табиғат заңдылығы екенін білдіреді».




37-сурет.

Егер гипотеза ақиқат болса, онда кейбір алгоритмдер класын беретін басқа барлық формалды анықтамалар, Эмиль Посттың формулировка 1-імен берілген алгоритмдер класына эквивалентті (37-сурет).

*Тьюринг машинасы*

1936 жылы Алан Тьюринг (Turing) Лондондағы математикалық қауым еңбектерінде «О вычислимых числах в приложении к проблеме разрешения» атты мақаласын жариялады [4]. Бұл мақала заманауи алгоритмдер теориясының негізін құрайтын Пост пен Черч еңбектерімен бір деңгейде тұр.

Тьюринг мақаласында Гильберттің екінші мәселесінің шешімі табылмайтындығына жауап берді. Тьюринг мақаласының мағынасы қарастырылған мәселелер шекарасынан әлде қайда кең жатыр еді.

Джон Хопкрофттың айтылған жұмысқа берген мінездемесі: «Гильберт мәселесіне орай жұмыс жасай отырып, Тьюрингке әдіс ұғымына нақты анықтама беруге тура келді. Әдіс ұғымын, алгоритм ретінде, яғни механикалық орындалатын процедура түрінде, интуитивті сезіне отырып, ол осы идеяны есептеу процесінің реттелген моделі ретінде қалай қолдану керек екендігін көрсетті. Алынған есептеу моделі, онда әр алгоритм қарапайым тізбектерге, элементар қадамдаға бөлінген, логикалық конструкция болды. Осыны кейін Тьюринг машинасы деп атады».

Тьюринг машинасы, осы мезетте қарастырылып отырған ұяшықтан сол немесе оң көршісіне өтуге (жылжуға) болатын шексіз жадыны қамтитын, шекті автомат моделінің кеңейтілуі болып табылады.

Жалпы түрде Тьюринг машинасын былай сипаттауға болады:

Берілсін:

1. шекті күйлер жиыны – $Q$, онда Тьюринг машинасы орналасуы мүмкін;
2. лентаның шекті символдар жиыны – $Г$;
3. δ функциясы (өту функциясы немесе программа), $Q \times Г$ декарттық көбейтудің жұп бейнесі (машина $q_i$ күйінде

болады және $\gamma_i$ символын көреді) үштік $Q \times \Gamma \times \{L,R\}$ декарттық көбейтудің бейнесімен беріледі (машина $q_j$ күйіне өтеді, $\gamma_i$ символын $\gamma_j$ символына алмастырады және лентаның сол немесе оң бір символына жылжиды) - $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ ;

4. бір символ $\Gamma \rightarrow$ е (бос);
5. ішкіжиын $\Sigma \in F \rightarrow$ лентаның кіріс символадрының ішкіжиыны ретінде анықталады $e \in (\Gamma - \Sigma)$;
6. күйлердің бірі – $q_0 \in Q$ машинаның бастапқы күйі болып табылады;

Шешілетін мәселе $\Sigma \in F$ жиынындағы шекті символдар санын $S_i \in \Sigma$ лентаға жазу жолымен беріледі (38-сурет),

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *e* | *S1* | *S2* | *S3* | *S4* | ..... | *Sn* | *e* |

38-сурет.

содан кейін машина бастапқы күйге өтеді және бастиек ең сол жақтағы бос емес символға орнатылады – ($q_0 \uparrow \omega$), сосын ($q_i$,$S_i$)→($q_j$, $S_k$, *L немесе R*) өту функциясы арқылы машина, өту функциясының нұсқауымен, бастиекті солға немесе оңға жылжыта, және басқа күйге өткізу көмегімен көрініп тұрған символдарды алмастыра бастайды.

Машинаның тоқтауы жүп ($q_i$, $S_i$) өту функциясы анықталмаған кезде орындалады.

Алан Тьюринг мынадай тұжырымдама жасады: кез-келген алгоритм, интуитивті түрде, эквивалентті Тьюринг машинасымен берілуі мүмкін. Бұл тұжырым Черч-Тьюрингтің тезисі ретінде белгілі. Әрбір компьютер Тьюринг машинасын модельдей алады (ұяшықтарды қайта жазу операциясы, машина күйінің өзгеруіне байланысты ұяшықтарды салыстырып, басқа көрші ұяшыққа өту).

# 7. АЛГОРИТМДЕРДІ ТАЛДАУ

*Алгоритмдерді салыстырмалы бағалау*

Практикалық есептерді шешу үшін алгоритмдерді қолдану барысында біз тиімді алгоритмді таңдау мәселесіне кезігеміз. Таңдау мәселесін шешу жүйені салыстыру бағасына келіп тіреледі, ол өз жағынан алгоритмнің формалды моделіне сүйенеді [3-6].

Әрі қарай, Пост анықтамасына сүйеніп, жалпы мәселені шешіде 1-*шешім* беретін дұрыс және финитті алгоритмдерді қарастырамыз. Формалды жүйе ретінде, фон-Нейман архитектурасын қамтитын процессоры, адрестік жадысы және жоғарғы деңгейлі тілдерге байланысты «элементар» операциялары бар, абстракты машинаны қарастырамыз,

Әрі қарай талдау жасау үшін мынадай ойды қабылдайық:

- әр команда белгіленген уақытта орындалуы керек;
- алгоритмнің берілген деректері әр қайсысы $\beta$ биттен тұратын машиналық сөздермен беріледі.

Нақты мәселе жадының $N$ сөзімен беріледі, яғни алгоритмнің кірісінде - $N_\beta=N*\beta$ бит ақпарат. Кейбір жағдайда, әсіресе матрицалық есептерді қарастырғанда $N$, сызықтық өлшемді өрнектейтін, алгоритмге кіру ұзындығының өлшемі болып табылады.

Программа, жалпы мәселені шешетін алгоритмді орындайтын, $\beta_M$ битті $M$ машиналық нұсқаудан тұрады - $M_\beta=M*\beta_M$ ақпарат биті.

Сонымен қатар, алгоритм абстрактылы машинаның келесі қосымша ресурстарын қажет етуі мумкін:

- $S_d$ – аралық нәтижені сақтауға арналған жады;
- $S_r$ – есептеу процессін ұйымдастыратын жады (рекурсивті шақыру және қайтаруларды ұйымдатыруға керекті жады).

*Анықтама.* Алгоритмнің еңбексыйымдылығы.

Берілген $F_a(N)$ нақты кірістің *алгоритмнің еңбексыйымдылығы* деп нақты мәселені шешу үшін берілген формалды жүйеде алгоритмнің орындайтын «элементар» операциялар санын айтамыз.

Әр түрлі облысқа ресурстар салмағы әр түрлі болады, ол алгоритмді комплексті бағалауға алып келеді:

$$\Psi_A = c_1 * F_a(N) + c_2 * M + c_3 * S_d + c_4 * S_r,$$

мұнда $c_i$ – ресурс салмағы.

*Алгоритмдерді талдаудағы белгілеу жүйесі*

$D_A$ – формалды жүйеде берілген есептің нақты мәселелер жиыны. D $\epsilon$ $D_A$ – нақты мәселенің берілуі және $|D|=N$.

Жалпы жағдайда $D_A$ жиынының жеке ішкіжиыны болады, $N$ қуатты барлық нақты мәселелерден тұратын:

бұл ішкіжиынды $D_N$ арқылы белгілейік: $D_N=\{D \in D_N, : |D|=N\}$;

$D_N$ ішкіжиынның қуаттылығын $M_{DN} \rightarrow M_{DN} = |D_N|$ арқылы белгілейік.

Келесі белгілеулерді еңгізейік:

1. $F_a\hat{}(N)$ – *ең нашар жағдай* – ең көп операциялар саны, $A$ алгоритмінің $N$ өлшемді нақты мәселені шешуде орынадатын:

$F_a\hat{}(N)=max\{F_a(D)\}$ - $D_N$ – нің жаман жағдайы

2. $F_a\check{}(N)$ – *ең жақсы жағдай* – ең кем операциялар саны, $A$ алгоритмінің $N$ өлшемді нақты мәселені шешуде орынадатын:

$F_a\check{}(N)=min\{F_a(D)\}$ - $D_N$ – нің жақсы жағдайы

3. $F_a(N)$ – *орташа жағдай* – операциялардың орташа саны, $A$ алгоритмінің $N$ өлшемді нақты мәселені шешуде орынадатын:

$$F_a(N)=(1/M_{DN}) * \sum_{D \in D_N} \{F_a(D)\}$$ - $D_N$ – нің орташа жағдайы

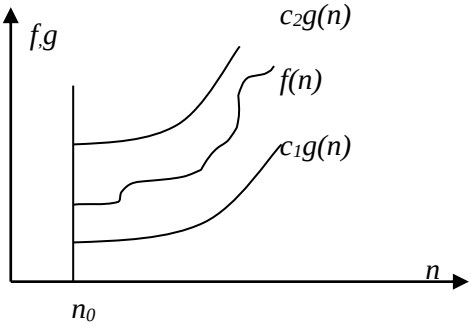
*Функцияны асимптотикалық талдау*

Алгоритмнің еңбексыйымдылық функциясын бағалауды *алгоритм күрделілігі* деп атайды.

Асимптотикалық талдауда келесі белгілеулер қолданылады:

*Θ (тетта) бағалауы*

*f(n)* және *g(n)* – оң аргументтің оң функциясы, *n≥1* кірістегі объектілер және операциялар саны – оң сандар), онда (38-сурет)




$f(n)=\Theta(g(n))$, егер $c_1$, $c_2$, $n_0$ оң болса және $c_1*g(n)\leq f(n)\leq c_2*g(n)$, $n>n_0$

38-сурет.

Әдетте, *g(n)* функциясы *f(n)* функциясының асимптотикалық бағалауы болады, себебі анықтама бойынша *f(n)* функциясы *g(n)* функциясынан айырмашылығы жоқ.

*f(n)=Θ(g(n))-дан g(n)=Θ(f(n)) алынады*

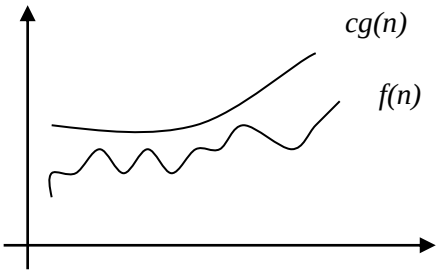
*Мысалдар:*

1) $f(n)=4n^2+nlnN+174$-$f(n)=\Theta(n^2)$;

2) $f(n)=\Theta(1)$ – *f(n)* нөлге тең емес тұрақтыға тең, немесе *f(n)* - $\infty$ тұрақтымен шектелген: $f(n)=7+1/n=\Theta(1)$.

*O (О үлкен)бағалауы*

*Θ* бағалаудан айырмашылығы, *O* бағалауда *f(n)* функциясы *g(n)* функциясынан аспауын, $n>n_0$ бастап тұрақты көбейткешке, талап етеді (39-сурет):




$\exists c > 0, n0 > 0$ :

$0 \le f(n) \le c * g(n),$

$\forall n > n0$

39-сурет.

*O(g(n))* жазуы функция класын белгілейді, яғни олардың бәрі *g(n)* функциясына қарағанда тұрақты көбейткішке тез өседі. Сондықтан, кей кезде, *g(n)* функциясы *f(n)* функцисын мажорлайды дейді.

Мысалы, барлық

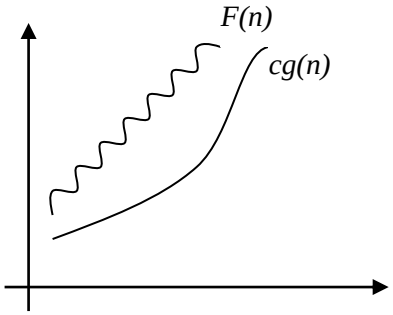
$f(n)=1/n$, $f(n)=12$, $f(n)=3*n+17$, $f(n)=n*Ln(n)$, $f(n)=6*n^2+24*n+77$

функциялары үшін $O(n^2)$ бағалау орынды.

*O* бағалауды айту арқылы мажорлайтын ең «жақын» функцияны көрсетуге болады, себебі, мысалы, $f(n)=n^2$ үшін $O(2^n)$ бағалау орынды, бірақ оның практикалық мағынасы жоқ.

*Ω (омега) бағалауы*

*O* бағалаудан айырмашылығы, *Ω* бағалау төменнен бағалау болып табылады – функция класын анықтайды, *g(n)* –ге қарағанда тұрақты көбейткішке жәй емес өсетін (40-сурет):




$$\exists\, c > 0,\ n0 > 0 :$$
$$0 \le c * g(n) \le f(n)$$

40-сурет.

Мысалы, *Ω(n*Ln(n))* жазу функция класын белгілейді, *g(n)=n*Ln(n)* қарағанда жәй емес өсетін, бұл класқа дәрежесі бірден үлкен барлық полиномдар және негізі бірден үлкен барлық дәрежелі функциялар жатады.

*O* асимпотикалық белгілеу Бахманның жәй сандар теориясы (Bachman, 1892) кітабынан, ал *Θ, Ω* белгілеулері Д.Кнуттан (Donald Knuth) енген [1].

## СҰРАҚТАР

1. Алгоритм ұғымы.
2. Алгоритмнің қасиеттері.
3. Алгоритмнің блок-схемалық жазылуы және оның ерекшеліктері.
4. Программалау тілі алгоритмді жазудың формасы.
5. Тармақталу алгоритмі.
6. Циклдық алгоритмнің түрлері.
7. Массив ұғымы, түрлері және олармен жұмыс жасау.
8. Стек ұғымы және оның ерекшеліктері.
9. Кезек ұғымы және оның ерекшеліктері.
10. Сұрыптау алгоритмінің ұғымы, түрлері және оның ерекшеліктері.
11. Іздеу алгоритмінің ұғымы, түрлері және оның ерекшеліктері.
12. Рекурсивті алгоритмнің түрлері және оның ерекшеліктері.
13. Тьюринг және Пост машинасы, оның ерекшеліктері.
14. Алгоритмдерді талдаудағы белгілеу жүйесі.
15. Функцияны асимптотикалық талдау.

## ҚОЛДАНЫЛҒАН ӘДЕБИЕТТЕР


1. Кнут Д. Искусство программирования. Тома 1, 2, 3. 3-е изд. Пер. с англ. : Уч. пос. – М.: Изд. дом "Вильямс", 2001 г.
2. Ахо А. В. Структуры данных и алгоритмы / А. В. Ахо, Д. Хопкрофт, Д. Д. Ульман. — М.: Издательский дом «Вильямс», 2000. — 384 с.
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2001 г. – 960 с., 263 ил.
4. Ульянов М.В., Шептунов М.В. Математическая логика и теория алгоритмов, часть 2: Теория алгоритмов. – М.: МГАПИ, 2003г.
5. Бентли Д. Жемчужины программирования / Д. Бентли. — СПб: Питер, 2002. — 272 с.
6. Карпов Ю.Г. Теория автоматов – СПб.: Питер, 2002 г. – 224с., ил.
7. Шень А.: Программирование теоремы и задачи – Москва, 2004 г.
8. С. А. Абрамов, Г. Г. Гнездилова, Е. Н. Капустина, М. И. Селюн Задачи по программированию. – М.: Вологда, 2000 г.

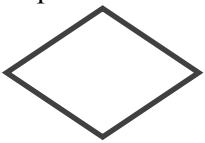
**ҚОСЫМША**

1. Алгоритмдер жазбасының негізгі әдістері:
A) белгілі
B) сандық
C) сөздік-формулалы
D) символдық
E) логикалық
F) графикалық
G) алгоритмдік тілде
H) арифметикалық
{Дұрыс жауабы} = C, F, G

2. Базалық алгоритмдік құрылымдардың түрлері:
A) сызықты
B) арифметикалық
C) файлдық
D) циклдық
E) тұйық
F) логикалық
G) көпше
H) тармақталған
{Дұрыс жауабы} = A, D, H

3. Алгоритмдердің міндетті қасиеттері:
A) шексіздік
B) анықталмағандылық
C) дискреттілік
D) үздіксіздік
E) түсініктілік
F) анықталғандылық
G) аяқталғандылық
H) бірегейлігі
{Дұрыс жауабы} = C, F, G

4. Блок сұлба символының оның атауына дұрыс сәйкестігі:

A)  Логикалық блок

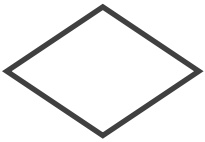
B)  Есептеулер блогы

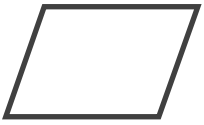
C)  «Үрдіс» блогы

D)  Модификация блогы

E)  Енгізу-шығару блогы

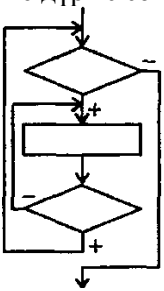
F)  Есептеулер блогы

G)  Модификация блогы

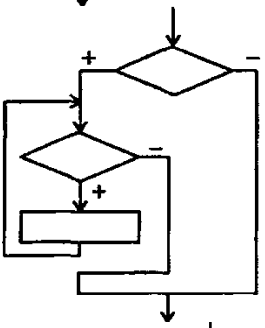
H)  Логикалық блок

{Дұрыс жауабы} = A, E, F

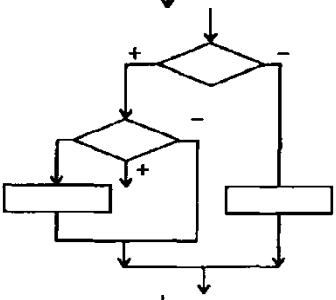
5. Алгоритм сұлбаларының фрагменттерінің осы алгоритм түрлеріне дұрыс сәйкестігі:

A) 

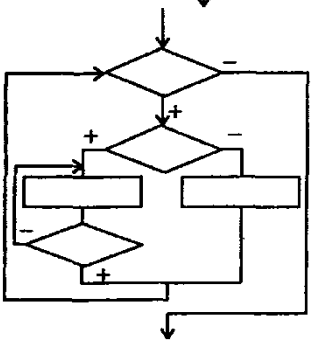

«Циклдағы цикл» түріндегі алгоритм

B) 

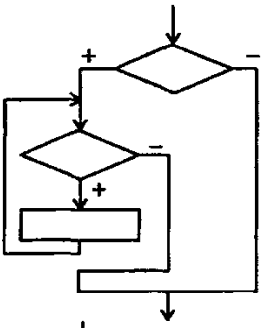

«Толық торапқа кірістірілген цикл» түріндегі алгоритм

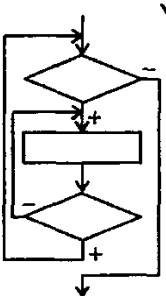
C) 

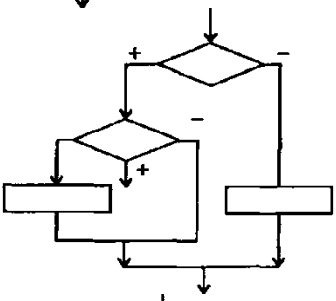

«Тораптағы торап» түріндегі алгоритм

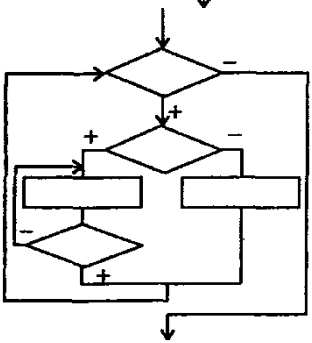
D) 


«Циклдағы торап» түріндегі алгоритм

E)  «Толық емес торапқа кірістірілген цикл» түріндегі алгоритм

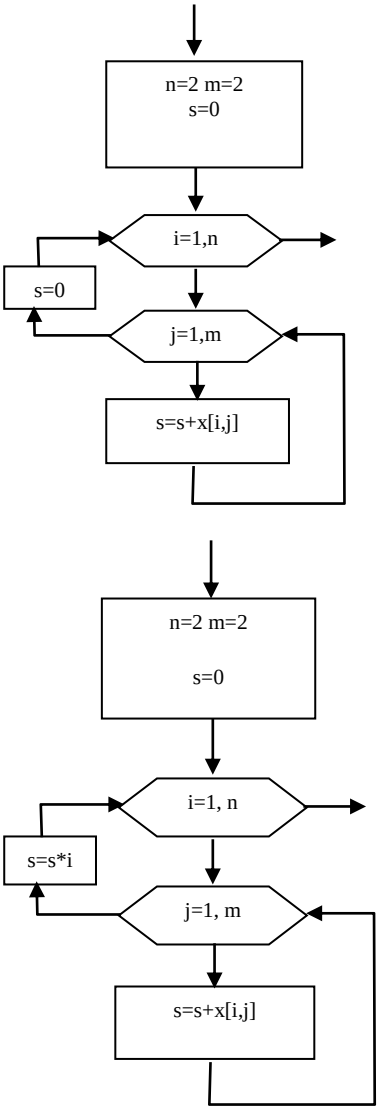
F)  «Тораптағы цикл» түріндегі алгоритм

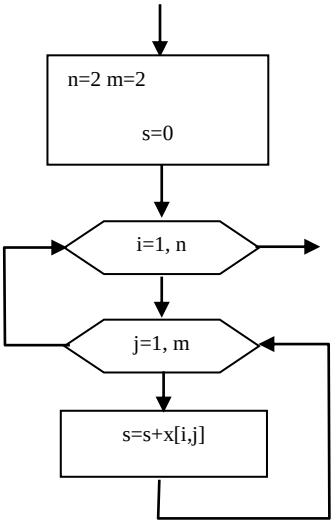
G)  «Циклдегі торап» түріндегі алгоритм

H)  «Тораптағы торап» түріндегі алгоритм

{Дұрыс жауабы} = A, C, E

6. Өлшемі 2 х 2: ((1, 2,), (-1, 2)) болатын X массиві берілген. S айнымалысының мәні келесі блок-сұлбалар орындалғаннан кейін тең болады






A) 3
B) 0
C) -6
D) 2
E) 8
F) 4
G) 1
H) -4
{Дұрыс жауабы} = B, E, F

7. Алгоритмдердің асимптотикалық уақыттық күрделілік белгілеулеріне берілетін корректілі түсініктемелері:
A) O(1) – тапсырма өлшемін еселеу, әрі керекті уақытты еселеу
B) O(1) – жұмыс уақыты тұрақты, ол тапсырма өлшеміне тәуелді емес
C) O(n) - тапсырма өлшемін екі еселеу, әрі керекті уақытты екі еселеу
D) O(n) – тапсырма өлшемін екі еселеу, бірақ керекті уақытты өсірмеу
E) $O(n^2)$ –тапсырма өлшемін екі еселеу, керекті уақытты екі есе өсіреді

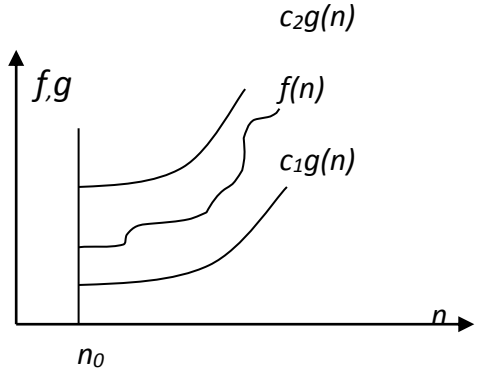
F) $O(n^2)$ - тапсырма өлшемін екі еселеу, керекті уақытты екі бірлікке өсіреді
G) $O(n^3)$ - тапсырма өлшемін екі еселеу, керекті уақытты сегіз есе өсіреді
H) $O(n^3)$ - тапсырма өлшемін екі еселеу, керекті уақытты үш есе өсіреді
{Дұрыс жауабы} = B, C, G

8. Алгоритмдердің күрделілігінің асимптотикалық талдауында грек әріптері келесіні білдіреді:
A) Θ –күрделіліктің жоғарғы бағасы
B) Ω – күрделіліктің нақты бағасы
C) O – күрделіліктің жоғарғы бағасы
D) Ω – күрделіліктің жоғарғы бағасы
E) Θ – күрделіліктің нақты бағасы
F) O – күрделіліктің төменгі бағасы
G) Θ – күрделіліктің төменгі бағасы
H) Ω – күрделіліктің төменгі бағасы
{Дұрыс жауабы} = C, E, H

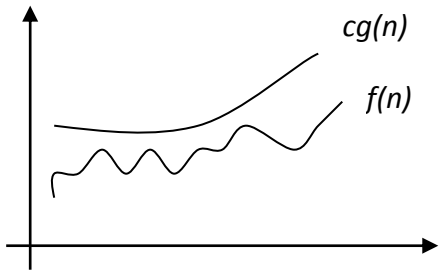
9. Алгоритмдер күрделілігінің асимптотикалық бағасы Ω, Θ, O мен a және b сандарының арасындағы қатынастар үшін келесі парелелдерді жүргізуге болады:
A) $f(n) = \Omega (g(n)) \approx a \leq b$
B) $f(n) = O (g(n)) \approx a \leq b$
C) $f(n) = \Omega (g(n)) \approx a = b$
D) $f(n) = \Theta (g(n)) \approx a = b$
E) $f(n) = O (g(n)) \approx a \geq b$
F) $f(n) = \Theta (g(n)) \approx a \leq b$
G) $f(n) = \Theta (g(n)) \approx a \geq b$
H) $f(n) = \Omega (g(n)) \approx a \geq b$
{Дұрыс жауабы} = B, D, H

10. Асимптотикалық бағалау графиктерінің дұрыс сәйкестіктері:

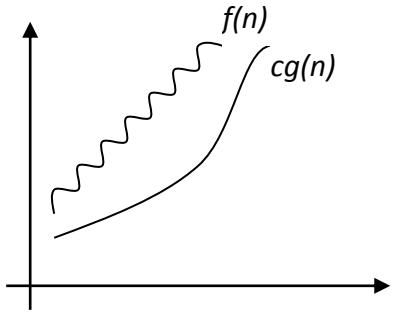
A) $f(n) = \Theta(g(n))$



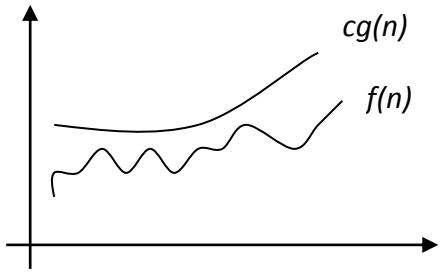

B) $f(n) = \Theta(g(n))$



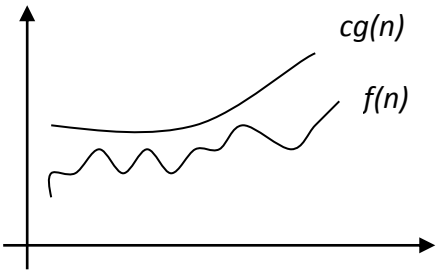

C) $f(n) = \Theta(g(n))$




D) $f(n) = O(g(n))$

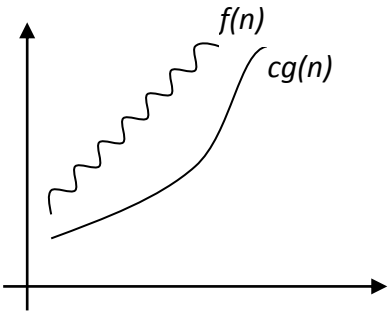


E)



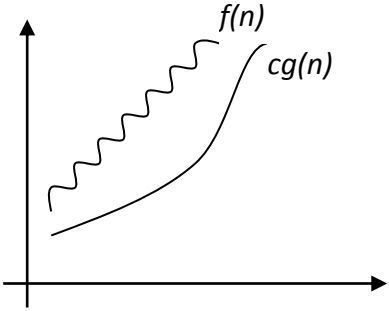

$f(n) = \Omega(g(n))$

F)



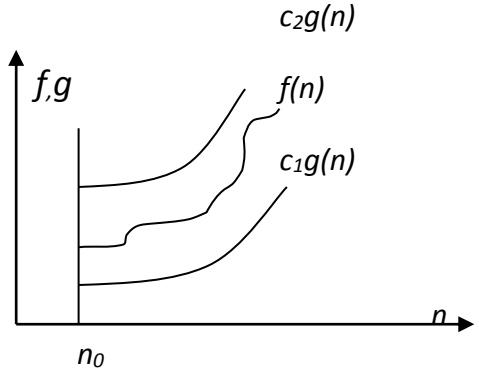

$f(n) = \Omega(g(n))$

G)




$f(n) = \Theta(g(n))$

H) $f(n) = \Theta(g(n))$




{Дұрыс жауабы} = A, D, F

11. Алгоритм күрделілігінің кластары еңбексыйымдылығының өсу реті тәртібімен орналасқан варианттары:
A) O(N), O(1), O(NlogN)
B) O(1), O(N), O($N^3$)
C) O($N^2$), O(N), O(logN)
D) O($2^N$), O(N), O(1)
E) O(N), O(NlogN), O($2^N$)
F) O($N^2$), O(logN), O($N^3$)
G) O(logN), O($N^3$), O($2^N$)
H) O($N^3$), O($2^N$), O(1)
{Дұрыс жауабы} = B, E, G

12. Құрылымдалған деректер түрі:
A) логикалық
B) бүтін
C) тиянақты нүктелі нақты
D) көрсеткіш
E) массивтер
F) жазбалар
G) жылжымалы нүктелі нақты
H) көпмүше
{Дұрыс жауабы} = E, F, H

13. Сызықты құрама құрылым деректеріне:
A) массив
B) граф
C) көпмүше
D) сызықты тізім
E) ағаш
F) кесте
G) жазба
H) мультитізім
{Дұрыс жауабы} = B, E, H

14. Сызықты емес құрылымдар:
A) графтар
B) кестелер
C) сызықты тізімдер
D) массивтер
E) ағаштар
F) көпмүшелер
G) мультитізімдер
H) циклдық тізімдер
{Дұрыс жауабы} = A, E, G

15. Стек, дек және кезектің анықтамалары:
A) Стек – бұл деректер құрылымы, мұнда бәрінен көп болған элемент бірінші болып жойылады
B) Кезек – бұл деректер құрылымы, мұнда қатынау кез-келген элементке ұйымдастырылуы мүмкін
C) Дек – бұл деректер құрылымы, мұнда деректерге қатынау берілген есептің шартына байланысты болады
D) Стек – бұл деректер құрылымы, мұнда элементтерге қатынау LIFO принципі бойынша ұйымдастырылған
E) Дек – бұл деректер құрылымы, мұнда элементтерді басына да, соңына да қосуға және басынан да, соңынан да элементтерді жоюға болады
F) Кезек – бұл деректер құрылымы, мұнда элементтерге қатынау FIFO принципі бойынша ұйымдастырылған
G) Стек – бұл деректер құрылымы, мұнда деректерге қатынау берілген есептің шартына байланысты болады

H) Кезек – бұл деректер құрылымы, мұнда ең соғынан жазылған элемент бірінші болып жойылады
{Дұрыс жауабы} = D, E, F

16. 44 55 12 42 94 18 67 тізбегі үшін тікелей біріктіріп сұрыптаудың үш этапы:
A) b=44 55 94 18 c=12 42 6 67 a= 44 12 55 42 94 6 18 67
B) b=55 44 94 18 c=42 12 67 a= 42 55 12 42 94 18 67
C) b=12 6 44 94 c= 42 18 55 67 a=12 42 18 6 44 55 67 94
D) b=44 55 12 42 c=94 18 6 67 a= 44 94 18 55 6 12 42 67
E) b=94 44 18 55 c=12 6 42 67 a=12 94 44 6 18 42 55 67
F) b=44 94 18 55 c=6 12 42 67 a=6 12 44 94 18 42 55 67
G) b=6 12 44 94 c= 18 42 55 67 a=6 12 18 42 55449467
H) b=6 12 44 94 c= 18 42 55 67 a=6 12 18 42 44 55 67 94
{Дұрыс жауабы} = D, F, H

17. Рекурсияға байланысты дұрыс емес тұжырымдамалар:
A) Рекурсия – бұл өзіне сілтеме бере отырып объектіні анықтау
B) Рекурсия – бұл факториалды табу үшін есептеу үрдісін ұйымдастыру әдісі
C) Рекурсия базасы – бұл тривиальды жағдай, бұл кезде есептің шешімі сірә бар болады, яғни фугкцияның өзіне қатынауын талап етпейді
D) Рекурсивті алгоритм –тікелей немесе жанама шақырудан тұратын алгоритм анықтамасы
E) Рекурсия тереңдігі – бұл бастапқы мәндердің саны
F) Екілік рекурсивті функция, бұл оның рекурсиясының тереңдігі екіге тең екендігі
G) Рекурсия түрі, бұл процедурада тікелей деп аталатын өзіне өзі айқын қатынау
H) Рекурсия түрі, бұл кезде A процедурасы B процедурасына қатынаудан тұрады, ал B A –ға жанама қатынай алатындығын көрсетеді
{Дұрыс жауабы} = B, E, F

18. f(n)=n, f(n)=n!, f(n)=$2^n$ функцияларының дұрыс берілуі:
A) f(0)=1, f(n)=f(n - 1)+1
B) f(0)=2, f(n)=f(n - 1)+2

C) f(0)=0, f(n)=f(n - 1)+2
D) f(0)=0, f(n)=f(n - 1)+1
E) f(0)=1, f(n)=f(n - 1)*3
F) f(0)=1, f(n)=n*f(n - 1)
G) f(0)=0, f(n)=f(n - 1)*n
H) f(0)=1, f(n)=2*f(n - 1)
{Дұрыс жауабы} = D, F, H

19. Рекуррентті арақатынастар арқылы өрнектелген тізбектер:
а) 2, 4, 16, 256, б) 2, 0.5, 2, 0.5, 2, в) 2, 5, 8, 11, 14,
A) $a_{n+1} = a_n + (-1)^{n+1} * 1.5$, $a_1 = 2$
B) $a_{n+1} = (a_n)^2$, $a_1 = 2$
C) $a_{n+1} = a_n * 3$, $a_1 = 2$
D) $a_{n+1} = (a_n)^3$, $a_1 = 2$
E) $a_{n+1} = a_n + (-1)^n * 1.5$, $a_1 = 2$
F) $a_{n+1} = a_n + 6$, $a_1 = 2$
G) $a_{n+1} = a_n + 3$, $a_1 = 2$
H) $a_{n+1} = a_n + 1$, $a_1 = 2$
{Дұрыс жауабы} = B, E, G

20. Берілген қатарлардың ортақ мүшелерін есептеу үшін керекті рекуррентті арақатынастар:
$1 - x^3 + x^6 - x^9 + \ldots$
$1 - x^2 + x^4 - x^6 + \ldots$
$1 - x + x^2 - x^3 + \ldots$
A) $y_{i+1} = y_i * (-x^3)$, $y_0 = 1$
B) $y_{i+1} = y_i * x^3$, $y_0 = 0$
C) $y_{i+1} = y_i * (-x^4)$, $y_0 = 1$
D) $y_{i+1} = y_i * (-x^2)$, $y_0 = 1$
E) $y_{i+1} = y_i * x$, $y_0 = 1$
F) $y_{i+1} = y_i * x^2$, $y_0 = 0$
G) $y_{i+1} = y_i * (-x)$, $y_0 = 1$
H) $y_{i+1} = y_i - x^3$, $y_0 = 1$
{Дұрыс жауабы} = A, D, G

21. Рекурсивті объектілерге жататындар:
A) натурал сандар
B) қарапайым сандар

C) логикалық функциялар
D) бинарлы ағаштар
E) нақты сандар
F) символдар
G) факториал функциясы
H) предикаттар
{Дұрыс жауабы} = A, D, G

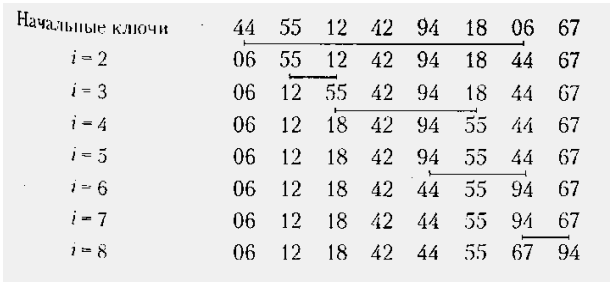
22. Келесі сұрыптау алгоритмдері үшін максимальды және орташа уақыттық күрделілік $O(n^2)$ сәйкес:
A) жедел Хоара сұрыптауы
B) біріктіру сұрыптауы
C) қарапайым енгізу сұрыптауы
D) Шелл сұрыптауы
E) ежелгі сұрыптау
F) көпіршік әдісімен сұрыптау
G) таңдап сұрыптау
H) разряд бойынша сұрыптау
{Дұрыс жауабы} = C, F, G

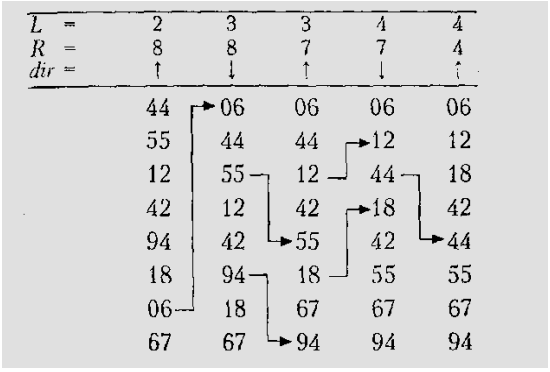
23. ( 6, 3, 2, 8, 1, 7, 4, 5) (i=1) бастапқы тізбек үшін тікелей қосып (i=2, i=3, i=4) сұрыптау алгоритімінің үш тізбектей қадамының нәтижесінде алынатын тізбектерді көрсетіңіз:
A) (1, 3, 2, 8, 6, 7, 4, 5)
B) (3, 6, 2, 8, 1, 7, 4, 5)
C) (1, 6, 3, 2, 8, 4, 7, 5)
D) (1, 2, 6, 3, 4, 8, 5, 7)
E) (2, 3, 6, 8, 1, 7, 4, 5)
F) (1, 2, 3, 8, 6, 7, 4, 5)
G) (1, 2, 3, 8, 6, 7, 4, 5)
H) (2, 3, 6, 8, 1, 7, 4, 5)
{Дұрыс жауабы} = B, E, H

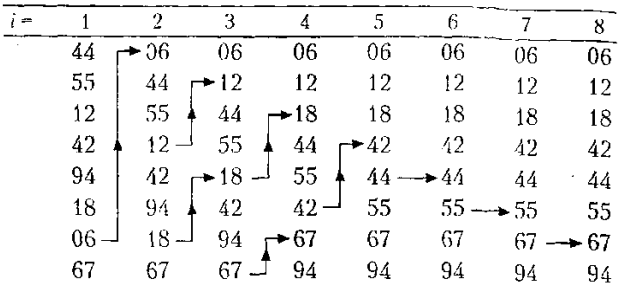
24. (6, 3, 2, 8, 1, 7, 4, 5) бастапқы тізбек үшін тікелей таңдап сұрыптау алгоритімінің үш бірінші қадамының нәтижесінде алынатын тізбектерді көрсетіңіз:
A) (1, 3, 2, 8, 6, 7, 4, 5)
B) (3, 6, 2, 8, 1, 7, 4, 5)

C) (1, 6, 3, 2, 8, 4, 7, 5)
D) (1, 3, 6, 2, 8, 4, 7, 5)
E) (1, 2, 6, 3, 4, 8, 5, 7)
F) (1, 2, 3, 8, 6, 7, 4, 5)
G) (1, 2, 3, 8, 6, 7, 4, 5)
H) (1, 2, 3, 6, 4, 5, 8, 7)
{Дұрыс жауабы} = A, F, G

25. (44, 55, 12, 42, 94, 18, 06, 67) бастапқы тізбектер үшін келесі сұрыптау әдістері қолданылады:






A) шейкерлік
B) шелл
C) жылдам
D) тікелей қосып
E) біріктіріп
F) пирамидалы
G) каскадты
H) көпіршікті
{Дұрыс жауабы} = A, D, H

26. n элементтен тұратын массивтегі сызықты іздеу кезіндегі ең жақсы, ең жаман және орташа жағдайлардағы салыстырулар саны:
A) log n
B) $n^2$
C) n/3
D) 1
E) n/2
F) n -1
G) n+1
H) n
{Дұрыс жауабы} = D, E, H

27. 10 элементтен (3 5 6 8 12 15 17 18 20 25) тұратын массивтен 6 санын бинарлы іздеу кезінде нөлдік іздеу жүргізілетін бірінші және екенші итерациялар кезіндегі массив бөлігі:
A) 15 17 18 20 25
B) 3 5 6 8 12 15 17 18 20 25
C) 3 5 6 8 12

D) 3 5 6 8
E) 6 8 12 15
F) 3 5
G) 6 8
H) 8 12 15 17
{Дұрыс жауабы} = B, D, G

28. Кестедегі нөлге тең емес элементтердің санын есептейтін, кестедегі нөлдерді бірлерге ауыстыратын, кестедегі барлық элементтерді екі есе өсіретін алгоритмдер:
A)алг AL (арг цел n, рез вещ таб a[1:n])
нач цел i, x
| x:=0
| нц для i от 1 до n
| | a[i]:=x
| кц
кон
B) алг AL (арг цел n, арг рез вещ таб a[1:n])
нач цел i, x
| x:=1
| нц для i от 1 до n
| | a[i]:=a[i]+x
| кц
кон
C) алг AL (арг цел n, арг рез цел таб a[1:n])
нач цел i
| нц для i от 1 до n
| | a[i]:=-a[i]
| кц
кон
D) алг AL (арг цел n, арг рез вещ таб a[1:n])
нач цел i, x
| x:=1
| нц для i от 1 до n
| | если a[i]=0
| | |то
| | | a[i]:=x
| | все

```
| кц
кон
```
E) алг AL (арг цел n, арг рез вещ таб a[1:n])
```
нач цел i,x
| x:=2
| нц для i от 1 до n
| | a[i]:=a[i]*x
| кц
кон
```
F) алг цел AL (арг цел n, арг вещ таб a[1:n])
```
нач цел i
| знач:=0
| нц для i от 1 до n
| | если a[i]<>0
| | |то знач:=знач+1
| | все
| кц
кон
```
G) алг AL (арг цел n, арг рез вещ таб a[1:n])
```
нач цел i
| нц для i от 1 до n
| | если a[i]<0
| | |то
| | | a[i]:=a[i]**2
| | все
| кц
кон
```
H) алг цел AL (арг цел n, арг вещ таб a[1:n])
```
нач цел i
| знач:=0
| нц для i от 1 до n
| | если a[i]>0
| | |то
| | | знач:=знач+a[i]
| | все
| кц
кон
```
{Дұрыс жауабы}= D, E, F

29. Ішкі жолдарды іздеу алгоритмдері:
A) Шелл алгоритмі
B) Рабин - Карп алгоритмі
C) Дейкстра алгоритмі
D) Евклид алгоритмі
E) Кнут – Моррис - Пратт алгоритмі
F) Форд алгоритмі
G) Бойер - Мура алгоритмі
H) Краскал алгоритмі
{Дұрыс жауабы}= B, E, G

30. Программадағы қателер түрі:
A) маңызды
B) синтаксистік
C) арифметикалық
D) маңызды емес
E) семантикалық
F) логикалық
G) грамматикалық
H) өте қиындар
{Дұрыс жауабы} = B, E, F