

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування спеціалізованих комп'ютерних систем

КУРСОВА РОБОТА

з дисципліни "Структури даних і алгоритми"

Виконав: Хлибов О.Р.

Група: КВ-31

Номер залікової книжки: КВ-3116

Допущений до захисту

2 семестр 2013/2014

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Узгоджено

ЗАХИЩЕНА " __ " _____ 2013р.

Керівник роботи

зоцінкою _____

_____/Марченко О.І./

_____/Марченко О.І./

Дослідження ефективності методів бінарного пошуку на багатовимірних масивах

Виконавець роботи
Хлибов Олександр Романович

_____ 2014р.

Технічне завдання на курсову роботу

I.

Описати принцип та схему роботи досліджуваного методу сортування для одновимірного масиву.

II. Скласти алгоритм сортування багатовимірного масиву заданим методом, згідно з варіантом, та написати відповідну програму на мові програмування.

Програма повинна задовольняти наступні вимоги:

1. Всі алгоритми повинні бути реалізовані в рамках ОДНІЄЇ програми з діалоговим інтерфейсом для вибору варіантів тестування та виміру часу кожного алгоритму.

2. Одним з варіантів запуску програм має бути режим запуску виміру часу всіх алгоритмів у пакетному режимі.

тобто запуск всіх алгоритмів для всіх випадків і побудову результуючої таблиці з наведеним нижче зразком для масиву заданими геометричними розмірами.

3. При реалізації програми повинні бути використані модулі (unit).

4. Програма повинна мати коментарі для всіх структур даних, процедур та функцій, а також до основних смислових фрагментів алгоритмів.

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів.

V. За результатами досліджень скласти порівняльну таблицю з різними ознаками.

VI.

Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами:

1. Для одновимірного масиву відносно загальної відомої теорії.

2. Для багатовимірних масивів відносно результатів для одновимірного масиву.

3. Для заданих алгоритмів на багатовимірних масивах між собою.

4. Дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою.

5. Для всіх вищезазначених пунктів порівняльного аналізу пояснити,
ЧОМУ алгоритми в розглянутих ситуаціях поводяться саме так, а не інакше.

VII. Зробити висновки за виконаним порівняльним аналізом.

Варіант №96

Задача

Визначити знаходження та місце положення заданого елемента X серед елементів кожної діагоналі окрему у всіх перерізах тривимірного масиву $A[p, n, n]$. Елементи кожного перерізу окремо в порядку ваніна скрізно по стовпчиках за незменшенням.

Досліджувані методи та алгоритми

1. Двійковий пошук, що знаходить випадковий елемент з тих, що співпадають з шуканим елементом.
2. Двійковий пошук, що знаходить найлівіший елемент з тих, що співпадають з шуканим елементом.

Випадки дослідження

1. Потрібний елемент знаходиться після $\frac{1}{4}$ максимально можливого числа порівнянь в області пошуку.
2. Потрібний елемент знаходиться після $\frac{1}{2}$ максимально можливого числа порівнянь в області пошуку.
3. Потрібний елемент знаходиться після $\frac{3}{4}$ максимально можливого числа порівнянь в області пошуку.
4. Потрібний елемент знаходиться при останньому порівнянні в області пошуку.
5. Потрібного елемента немає в області пошуку.

Опис теоретичних положень

У роботі використовується два алгоритми бінарного пошуку:

Бінарний пошук №1: Вважається, що масив впорядковано за не зменшенням. На початку ліва і права межі встановлюються на початок та кінець відповідно.

1. Береться елемент, індекс якого дорівнює півсумі лівої та правої меж і порівнюються з шуканим елементом.
2. Якщо вони рівні, пошук закінчується.
3. Якщо середній елемент виявляється меншим за шуканий, то ліва границя зміщується на позицію, що на одиницю *більша* від позиції середнього елемента.
4. Якщо ж навпаки – середній елемент більший за шуканий – зсуваємо праву границю на позицію, що на одиницю *менша* за позицію середнього елемента.
5. Повторюємо доки не знайдемо елемент, границі не перетнуться або стануть рівними.

Схема сортування алгоритму:

Нехай шуканий елемент $X=14$

Задано впорядкований за незменшенням масив:

1	2	3	4	5	6	7	8	9	$n=10$
2	3	5	7	9	10	14	15	16	18

$L=1$

$R=n$

$$i = \left\lceil \frac{L+R}{2} \right\rceil = 5$$

1	2	3	4	5	6	7	8	9	$n=10$
2	3	5	7	9	10	14	15	16	18

$L=1$

$R=n$

$$A[5]=9 < X \Rightarrow L := i+1 = 6;$$

$$i = \left\lfloor \frac{6+10}{2} \right\rfloor = 8$$

1	2	3	4	5	6	7	8	9	n=10
2	3	5	7	9	10	14	15	16	18

L=6

R=n

$$A[8] = 15 > X \Rightarrow R := i - 1 = 7;$$

$$i = \left\lfloor \frac{6+7}{2} \right\rfloor = 6$$

1	2	3	4	5	6	7	8	9	n=10
2	3	5	7	9	10	14	15	16	18

L=6 R=7

$$A[6] = 10 < X \Rightarrow L := i + 1 = 7;$$

$$i = \left\lfloor \frac{7+7}{2} \right\rfloor = 7$$

1	2	3	4	5	6	7	8	9	n=10
2	3	5	7	9	10	14	15	16	18

L=R=7

$$A[7] = X$$

Пошук закінчено

Опис алгоритму мовою Pascal:

L:=1; R:=n;

```

while L<=R do
begin
i:=(L+R) div 2;
if A[i]=X then Break
else
if A[i]<X then
L:=i+1
else
R:=i-1;
end;

if L<=R then writeln('Element has been found at position', i)
elsewriteln('Element has not been found');

end.

```

Бінарний пошук №2: Як і перший алгоритм, алгоритм №2 працює лише для впорядкованих масивів. Відмінність його у тому, що він знаходить не випадковий елемент із рівних заданому, а найлівіший з них. Алгоритм для масиву відсортованого за не зменшенням:

1. Береться елемент , індекс якого дорівнює півсумі лівої та правої меж і порівнюються з шуканим елементом.
2. Якщо середній елемент виявляється меншим за шуканий, то ліва границя зміщується на позицію, що на одиницю *більша* від позиції середнього елемента.
3. Якщо ж навпаки – середній елемент більший за шуканий – зсуваємо праву границю на позицію, що на одиницю *менша* за позицію середнього елемента.
4. Повторюємо доки границі не перетнуться.

Схема сортування алгоритму:

Нехай шуканий елемент $X=14$

Задано впорядкований за не зменшенням масив:

1	2	3	4	5	6	7	8	9	$n=10$
2	2	2	7	9	10	14	15	16	18

$L=1$

$R=n$

$$i = \left\lceil \frac{1+10}{2} \right\rceil = 5$$

1	2	3	4	5	6	7	8	9	<i>n=10</i>
2	2	2	7	9	10	14	15	16	18

L=1

R=n

$A[i]=9 > x \Rightarrow R:=i;$

$$i = \left\lfloor \frac{1+5}{2} \right\rfloor = 3$$

1	2	3	4	5	6	7	8	9	<i>n=10</i>
2	2	2	7	9	10	14	15	16	18

L=1

R=5

$A[i]=2 = X \Rightarrow R:=i;$

$$i = \left\lfloor \frac{1+3}{2} \right\rfloor = 2$$

1	2	3	4	5	6	7	8	9	<i>n=10</i>
2	2	2	7	9	10	14	15	16	18

L=1

R=3

$A[i]=2 = X \Rightarrow R:=i;$

$$i = \left\lfloor \frac{1+2}{2} \right\rfloor = 1$$

1	2	3	4	5	6	7	8	9	<i>n=10</i>
2	2	2	7	9	10	14	15	16	18

L=1 R=2

$A[i]=X \Rightarrow R:=i;$
 $L=R \Rightarrow \text{Пошук закінчено}$

Опис алгоритму мовою Pascal:

```
L:=1; R:=n;  
while L<R do  
begin  
  i:=(L+R) div 2;  
  if A[i]<X then  
    L:=i+1  
  else  
    R:=i;  
end;  
  
if A[R]=X then writeln('Element has been found at position', i)  
else writeln('Element has not been found');  
  
end.
```

Швидкодія обох алгоритмів однакова - $T=O(\log n)$

Схема імпорту/експорту модулів та структурна схема взаємовикликів процедур і функцій

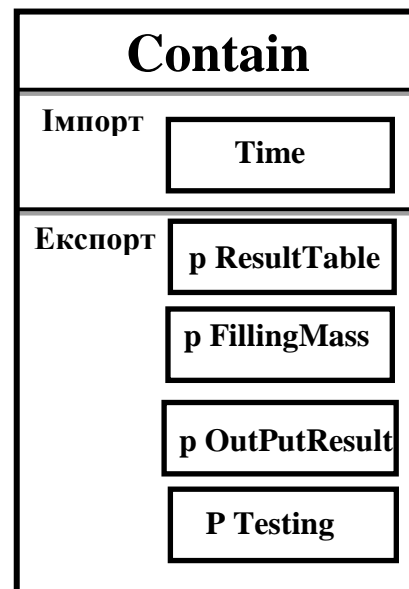
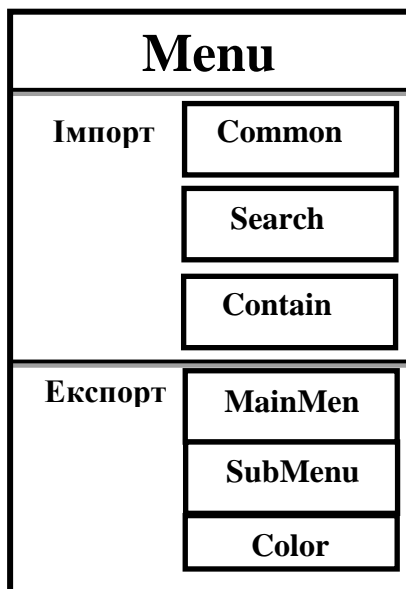
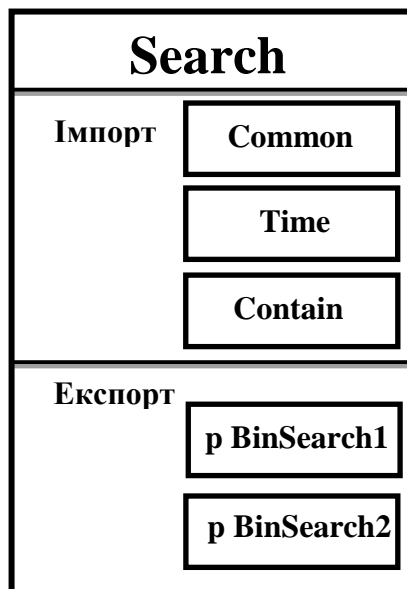
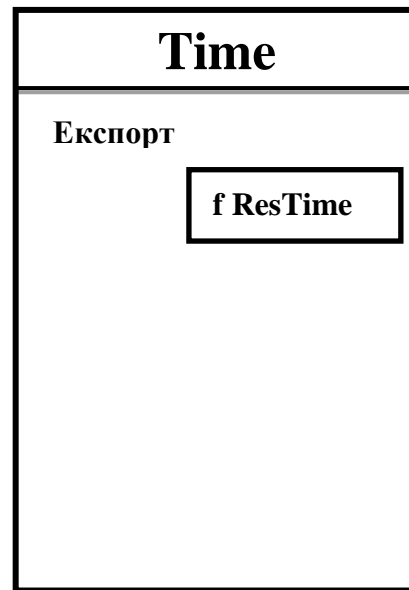
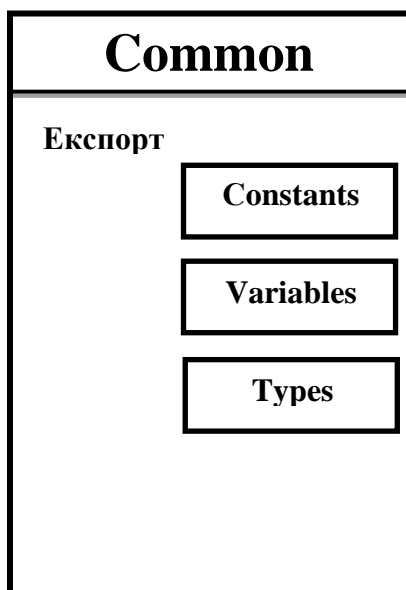
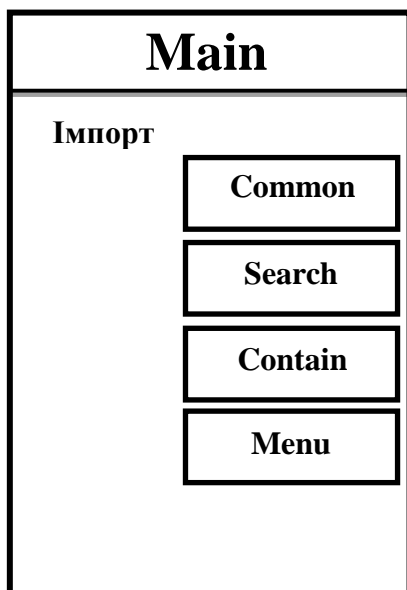
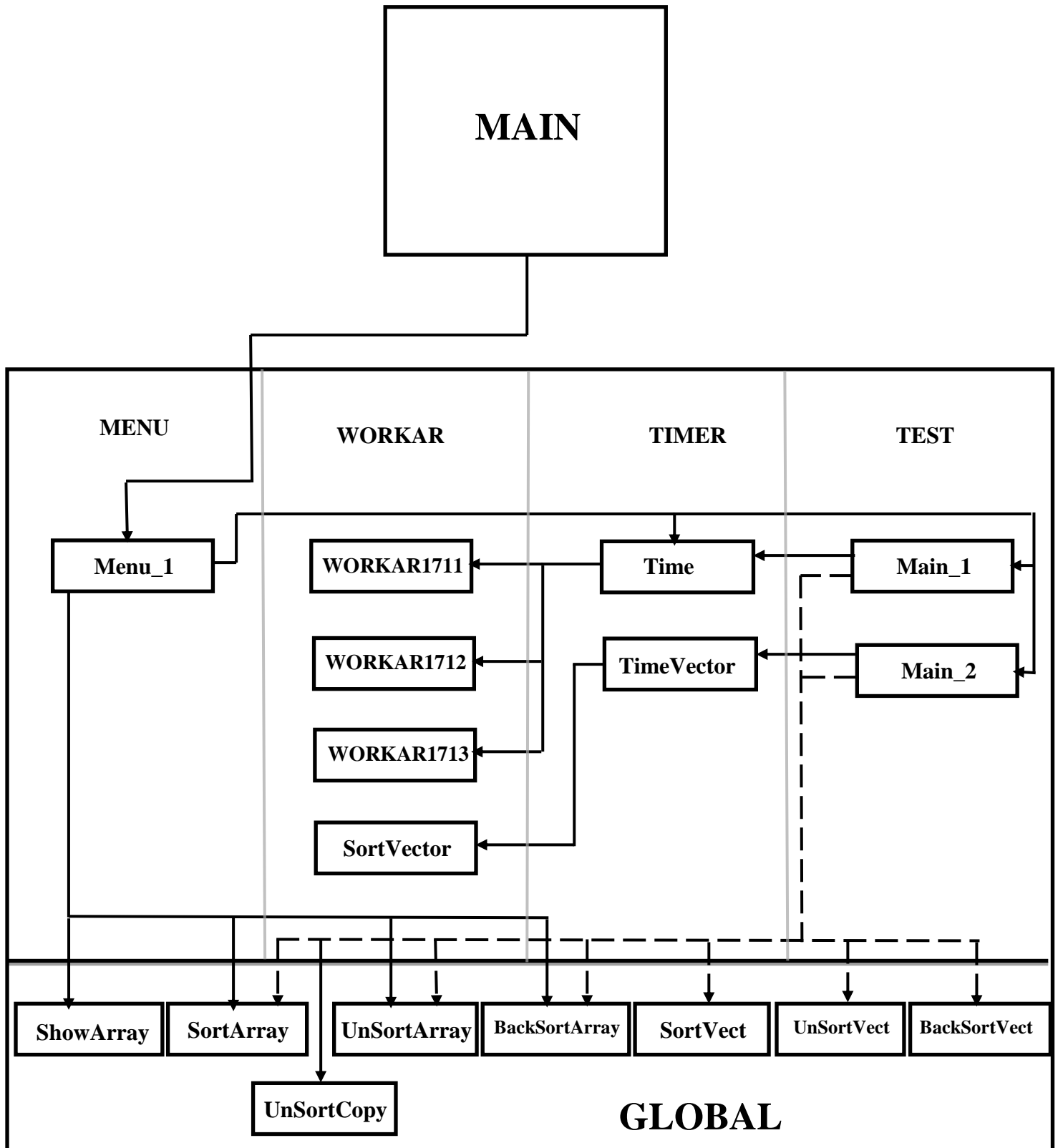


Схема викликів процедур та функцій



Опис призначення процедур і функцій

1.) **Common** – модуль, що слугує для збереження різних структур даних (тривимірний масив, константи, змінні тощо).

2.) **Time** – окремий модуль для процедури та різних структур даних, що використовуються для виміру часу.

2.1 *ResTime* – функція, що визначає різницю між часом закінчення і часом старту алгоритму. Результат функції – час в сотих секунди.

3.) **Search** – модуль в якому зберігаються два алгоритми бінарного пошуку у тривимірному масиві.

3.1 *BinSearch1* – процедура, що виконує бінарний пошук по масиву номер 1.

3.2 *BinSearch2* – процедура, що виконує бінарний пошук по масиву номер 2, який знаходить найлівіший елемент.

4.) **Menu** – модуль, який містить процедури, що організовують діалог з користувачем.

4.1 *Color* – допоміжна процедура, яка зафарбовує вибраний пункт меню.

4.2 *MainMenu* – процедура, яка надає користувачу вибір між методами сортування або дозволяє надрукувати остаточну таблицю чи зовсім завершити програму.

4.3 *SubMenu* – процедура, що дозволяє регулювати «ключі» пошуку так, щоб виконувалась певна кількість порівнянь.

5.) **Contain** – модуль, що містить загальні процедури, які використовуються програмою.

5.1 *ResultTable* – процедура виводу остаточної таблиці з часом для двох алгоритмів і декількох ключових випадків знаходження елементів.

5.2 *FillingMass* – процедура, що впорядковано заповнює масив наскрізно по стовпчиках.

5.2 *OutPutResult* – процедура для виводу результатів виміру часу та координат знайдених елементів.

5.3 *Testing* – процедура, що виводить масив невеликого розміру та проводить пошук заданого користувачем елемента для демонстрації коректності роботи алгоритму.

Код програми

Основна програма **Main**

```
program Main;
{Основна програма}
uses
  Menu, Dos, Crt, Search, Common,
  Contain;
{Підключаємо стандартні та користувацькі модулі,
необхідні для роботи програми}

begin
repeat
  clrscr; {Очищуємо екран}
  FillingMass(cube, p, n); {Викликаємо процедуру для заповнення масиву}
  MainMenu; {Виклик процедури для налагодження діалогу з користувачем}
  c := readkey; {Зчитуємо клавішу}
until c = char(27); {Продовжуємо доки не натиснуто Esc}
end.
```

Модуль **Menu**

```
unit Menu;

interface

{Оголошуємо описані в модулі процедури}
procedure Color(consts: string);
procedure MainMenu;
procedure SubMenu;

implementation

uses
  crt, common, search, Contain;

procedure Color(consts: string);
{Допоміжна процедура для зафарбовування вибраного пункту меню}
begin
  TextColor(10); {Змінюємо колір тексту}
  writeln(s);
  TextColor(15); {І повертаємо його значення до стандартного,
щоб наступні пункти меню не зафарбовувались}
end;

procedure MainMenu;
{Головне меню. Дозволяє вибирати метод сортування або виводити
результуючу таблицю}
var
  ch: char; {Змінна для считування нажатої клавіші}
  s1, s2, s3, s4, s5: string; {Змінні типу string, в яких записані
```

```
пунктименю}
```

```
begin
```

```
{Присвоємоцимзміннимназвипунктівменю}
```

```
s1 := 'Testing';
```

```
s2 := 'The resulting table';
```

```
s3 := 'Binary search N1';
```

```
s4 := 'Binary search N2';
```

```
s5 := 'Exit';
```

```
{"Бігаємо" поциклу, змінючिलічильник, щовідповідаєзаномерпунктуменю,  
докикористувач не натисне "Enter"}
```

```
item1 := 1;
```

```
repeat
```

```
clrscr;
```

```
if item1 = 1 then
```

```
begin
```

```
gotoxy(30, 10);
```

```
Color(s1)
```

```
end
```

```
else
```

```
begin
```

```
gotoxy(30, 10);
```

```
writeln(s1);
```

```
end;
```

```
if item1 = 2 then
```

```
begin
```

```
gotoxy(30, 11);
```

```
Color(s2)
```

```
end
```

```
else
```

```
begin
```

```
gotoxy(30, 11);
```

```
writeln(s2);
```

```
end;
```

```
if item1 = 3 then
```

```
begin
```

```
gotoxy(30, 12);
```

```
Color(s3)
```

```
end
```

```
else
```

```
begin
```

```
gotoxy(30, 12);
```

```
writeln(s3);
```

```
end;
```

```
if item1 = 4 then
```

```
begin
```

```
gotoxy(30, 13);
```

```
Color(s4)
```

```
end
```

```
else
```

```
begin
```

```
gotoxy(30, 13);
```

```
writeln(s4);
```

```
end;
```

```
if item1 = 5 then
```

```
begin
```

```
gotoxy(30, 14);
```

```
Color(s5)
```

```
end
```

```
else
```

```
begin
```

```
gotoxy(30, 14);
```

```

writeln(s5);
end;

ch := readkey; {Зчитуємо натиснуту клавішу}
if ch = 'w' then dec(item1); {І відповідно зменшуємо}
if ch = 's' then inc(item1); {чи збільшуємо лічильник}

if item1 < 1 then item1 := 5; {"Перестрибуємо" на кінець}
if item1 > 5 then item1 := 1; {або початок, якщо дійшли до крайнього пункту меню}

until ch = chr(13);

{В залежності від вибраного пункту меню запускаємо процедуру ResultTable
або виходимо з програми}
if item1 = 1 then
begin
ResultTable; {Виводимо остаточну таблицю}
Exit; {і повертаємось у меню або завершуємо}
end; {програму, якщо натиснути Esc}

if item1 = 5 then
begin
clrscr;
Halt(0); {Завершуємо програму без попередження}
end;

SubMenu;
end;

{Якщо ми не запустили ResultTable і не вийшли з програми
переходимо до підменю}
procedure SubMenu;
{Процедура другого "рівня" меню, яка дозволяє вибирати
яку кількість порівнянь повинна зробити програма, до того
як знайде елемент}
var
ch: char; {Змінна для считування нажатої клавіші}
s1, s2, s3, s4, s5: string; {Змінні типу string, в яких записані
пункти меню}

begin

s1 := 'After 1/4 comparisons';
s2 := 'After 2/4 comparisons';
s3 := 'After 3/4 comparisons';
s4 := 'At the last comparison';
s5 := 'Element is absent';

{Принцип роботи підменю аналогічний до головного меню}
item2 := 1;
repeat
clrscr;
if item2 = 1 then
begin
gotoxy(30, 10);
Color(s1)
end
else
begin
gotoxy(30, 10);
writeln(s1);
end;

if item2 = 2 then
begin

```

```

gotoxy(30, 11);
Color(s2)
end
else
begin
gotoxy(30, 11);
writeln(s2);
end;

if item2 = 3 then
begin
gotoxy(30, 12);
Color(s3)
end
else
begin
gotoxy(30, 12);
writeln(s3);
end;

if item2 = 4 then
begin
gotoxy(30, 13);
Color(s4)
end
else
begin
gotoxy(30, 13);
writeln(s4);
end;

if item2 = 5 then
begin
gotoxy(30, 14);
Color(s5)
end
else
begin
gotoxy(30, 14);
writeln(s5);
end;

ch := readkey;
if ch = 'w' then dec(item2);
if ch = 's' then inc(item2);

if (item2 < 1) then item2 := 5;
if (item2 > 5) then item2 := 1;

until ch = chr(13);
{В залежності від вибраних пунктів меню та підменю
запускаємо відповідні алгоритми пошуку з вибраною користувачем
кількістю порівнянь}
if item1 = 3 then
begin
case item2 of
1: BinSearch1(Cube, n, p, X1[1], X2[1]);
2: BinSearch1(Cube, n, p, X1[2], X2[2]);
3: BinSearch1(Cube, n, p, X1[3], X2[3]);
4: BinSearch1(Cube, n, p, X1[4], X2[4]);
5: BinSearch1(Cube, n, p, X1[5], X2[5]);
end;
end;

if item1 = 4 then
begin
case item2 of

```



```

1: BinSearch2(Cube, n,p,X1[1],X2[1]);
2: BinSearch2(Cube, n,p,X1[2],X2[2]);
3: BinSearch2(Cube, n,p,X1[3],X2[3]);
4: BinSearch2(Cube, n,p,X1[4],X2[4]);
5: BinSearch2(Cube, n,p,X1[5],X2[5]);
end;
end;
{Після того як пошукзавершився
виводиморезультати(час та координати елементів)}
OutPutResult(n,p);
end;
end.

```

Модуль **Contain**

```

unitContain;
{Модуль, щоміститьзагальніпроцедури, яківикористовуютьсяпрограмою}
interface

uses
Common;{Підключаємомодульзглобальнимиструктурамиданих
якінеобхідні нам для опису процедур в розділіinterface}

procedureFillingMass(varCube: Mass; p, n: integer);
procedureResultTable;
procedureOutPutResult(n,p:integer);
procedureTesting;

implementation
usesCrt,Time,Search;
procedureFillingMass(varCube: Mass; p, n: integer);
{Процедура заповнення масиву по стовпчиках за неспаданням}
var
z: longint;
begin
z := 1;
for k := 1 to p do
begin
z:=1;{Присвоюємолічильнику 1, щобвсірозрізибулиідентичні}
for j := 1 to n do
begin
for i := 1 to n do
begin
Cube[k, i, j] := z;
inc(z);
end;
end;
end;
end;

procedureResultTable;
{Процедура створенняостаточноїтаблиці та заповненняїї результатами}

vartime11,time12,time13,time14,time15:integer;{Змінні, необхіднідля}
time21,time22,time23,time24,time25:integer;{зберіганнязначеньчасу}

procedureWriteTime(consttime:real);
{Допоміжнапроцедура, щорегулюєкількістьвідступів
длязбереженняконструкції}

begin
iftime<10 then
write(' ',time:3,' |')
else if time<100 then
write(' ',time:3,' |')
else iftime<1000 then
write(' ',time:3,' |')

```

```

else if time <10000 then
write(' ',time:3,' |')
else
write(' ',time:3,' |');
end;

begin
clrscr;
{Очищуємо екран та запускаємо процедуру пошуку з необхідними параметрами}
BinSearch1(Cube, n,p,X1[1],X2[1]);
time11:=Algorithm_Time;

BinSearch1(Cube, n,p,X1[2],X2[2]);
time12:=Algorithm_Time;

BinSearch1(Cube, n,p,X1[3],X2[3]);
time13:=Algorithm_Time;

BinSearch1(Cube, n,p,X1[4],X2[4]);
time14:=Algorithm_Time;

BinSearch1(Cube, n,p,X1[5],X2[5]);
time15:=Algorithm_Time;

BinSearch2(Cube, n,p,X1[1],X2[1]);
time21:=Algorithm_Time;

BinSearch2(Cube, n,p,X1[2],X2[2]);
time22:=Algorithm_Time;

BinSearch2(Cube, n,p,X1[3],X2[3]);
time23:=Algorithm_Time;

BinSearch2(Cube, n,p,X1[4],X2[4]);
time24:=Algorithm_Time;

BinSearch2(Cube, n,p,X1[5],X2[5]);
time25:=Algorithm_Time;

clrscr;
{Малюємо власне талицю та значення часу виконання алгоритмів}
writeln(' _____ ');
writeln(' | 1/4 | 2/4 | 3/4 | Last | Absent | ');
writeln(' |=====|=====|=====|=====|=====|=====| ');

write(' |BinSearch1 | ');
WriteTime(time11);
WriteTime(time12);
WriteTime(time13);
WriteTime(time14);
WriteTime(time15);
writeln;

writeln(' |=====|=====|=====|=====|=====|=====| ');
write(' |BinSearch2 | ');
WriteTime(time21);
WriteTime(time22);
WriteTime(time23);
WriteTime(time24);
WriteTime(time25);
writeln;

writeln('=====');
end;

procedure OutPutResult(n,p:integer);

```

```
{Процедура виведення значення часу  
для одного випадку сортування, вибраним алгоритмом}
```

```
begin  
clrscr;  
{Виводимо значення часу}  
writeln('Algorithm time ', Algorithm_Time:5);  
writeln('Please, enter any key to output coordinates');  
readln;  
  
{Пробігаємо по масиву і в разі знаходження елемента  
виводимо масив}  
fork := 1 to p do  
begin  
{Масив координат для головної діагоналі}  
if CoordMain[k] <> 0 then  
begin  
write('Element has been found in main diagonal at position ');  
writeln(k, ', ', CoordMain[k], ', ', CoordMain[k]);  
end;  
  
{Масив координат для побічної діагоналі}  
if CoordMinor[k] <> 0 then  
begin  
write('Element has been found in minor diagonal at position ');  
writeln(k, ', ', CoordMinor[k], ', ', n - CoordMinor[k] + 1);  
end;  
end;  
end;  
  
procedure Testing;  
{Процедура демонстрації коректності  
роботи алгоритмів пошуку}  
  
{Задаємо масив невеликих розмірів}  
const  
p = 1; n = 8;  
var  
s1, s2: string;  
A: array[1..p, 1..n, 1..n] of integer;  
z, Element1, Element2, item2, i, j: byte;  
ch: char;  
  
begin  
s1 := 'Binary search N1';  
s2 := 'Binary search N2';  
  
{Заповнюємо масив}  
z := 1;  
fork := 1 to p do  
begin  
z := 1;  
for j := 1 to n do  
begin  
for i := 1 to n do  
begin  
A[k, i, j] := z;  
inc(z);  
end;  
end;  
end;  
  
{Робимо невелике підменю}  
clrscr;  
item2 := 1;  
repeat  
clrscr;
```

```

if item2 = 1 then
begin
gotoxy(30, 10);
Color(s1)
end
else
begin
gotoxy(30, 10);
writeln(s1);
end;

if item2 = 2 then
begin
gotoxy(30, 11);
Color(s2)
end
else
begin
gotoxy(30, 11);
writeln(s2);
end;

ch := readkey;
if ch = 'w' then dec(item2);
if ch = 's' then inc(item2);

if item1 < 1 then item2 := 2;
if item1 > 2 then item2 := 1;

until ch = chr(13);

{Виводимо масив}
clrscr;
for i := 1 to n do
begin
for j := 1 to n do
write(A[1, i, j]:2, ' ');
writeln;
end;

{Згідно вибраного пункту меню запускаємо
пошуковий алгоритм та виводимо результати}

{Start search with BinSearch N1}
if item2 = 1 then
begin
writeln('Element for searching in main diagonal');
write('Element='); readln(Element1);

writeln('Element for searching in minor diagonal');
write('Element='); readln(Element2);

fork := 1 to pdo
begin

{Main diagonal}
L := 1; R := n;
while (L <= R) do
begin
i := (L + R) div 2;
if A[k, i, i] = Element1 then Break
else if A[k, i, i] < Element1 then L := i + 1
else R := i - 1;
end;

```

```

if L <= R then
writeln('Element has been found in main diagonal at position ', k, ', ', i, ', ', i)
elsewriteln('Element has not been found');

{Minor diagonal}
L :=1; R := n;
while(L <= R) do
begin
i := (L + R) div 2;
ifA[k, i, n - i + 1] = Element2 then Break
else if A[k, i, n - i + 1] > Element2 then L := i + 1
elseR := i - 1;
end;

if L <= R then
writeln('Element has been found in minor diagonal at position ', k, ', ', i, ', ', n - i +
1)
elsewriteln('Element has not been found');

end;
end;

{Start search with BinSearch N2}
ifitem2 = 2 then
begin
writeln('Element for searching in main diagonal');
write('Element=');readln(Element1);

writeln('Element for searching in minor diagonal');
write('Element=');readln(Element2);
fork := 1 to pdo
begin

{Main diagonal}
L :=1; R := n;
while(L < R) do
begin
i := (L + R) div 2;
ifA[k, i, i] < Element1 then L := i + 1
elseR := i;
end;

ifA[k, R, R] = Element1 then
writeln('Element has been found at in main position ', k, ', ', R, ', ', R)
elsewriteln('Element has not been found');

{Minor diagonal}
L :=1; R := n;
while(L < R) do
begin
i := (L + R) div 2;
ifA[k, i, n - i + 1 ] > Element2 then L := i + 1
elseR := i;
end;

ifA[k, R, n - R + 1] = Element2 then
writeln('Element has been found in minor diagonal at position ',k, ', ', R, ', ', n - R + 1)
elsewriteln('Element has not been found');
end;
end;

```

```
end;  
end
```

Модуль Search

```
unit Search;  
{Модуль, у якому зберігаються процедури пошуку}  
interface  
  
uses  
Common; {Підключаємо модуль Common  
у розділі interface, тому що він необхідний для опису  
параметрів процедур}  
  
procedure BinSearch1(const Cube: Mass; const n, p, Element1, Element2: integer);  
procedure BinSearch2(const Cube: Mass; const n, p, Element1, Element2: integer);  
  
implementation  
  
uses  
Crt, time, dos, Contain;  
  
{Procedure with binary search N1}  
procedure BinSearch1(const Cube: Mass; const n, p, Element1, Element2: integer);  
{Процедура, що реалізовує бінарний пошук N1}  
var  
    iiii: longint; {Лічильник, що використовується у додатковому циклі,  
    який доданий для збільшення часу виконання пошуку}  
  
begin  
    clrscr;  
    gotoxy(33, 10);  
    writeln('Searching...');  
    Algorithm_Time := 0;  
  
    {Запускаємо цикл поперерізах}  
    fork := 1 to pdo  
    begin  
        {Запускаємо таймер}  
        with StartTimed do  
            GetTime(Hours, Minutes, Seconds, HSeconds);  
  
            {Заголовок додаткового циклу}  
            for iiii := 1 to 1000 do  
                begin  
  
                    {Шукаємо елементу головної діагоналі}  
                    L := 1; R := n;  
                    while (L <= R) do  
                        begin  
                            i := (L + R) div 2;  
                            if Cube[k, i, i] = Element1 then Break  
                            else if Cube[k, i, i] < Element1 then L := i + 1  
                            else R := i - 1;  
                        end;  
  
                    if L <= R then  
                        CoordMain[k] := i  
                    else CoordMain[k] := 0;  
  
                    end;  
  
                    {Зупиняємо таймер і фіксуємо час}  
                    with FinishTimed do  
                        GetTime(Hours, Minutes, Seconds, HSeconds);  
                        Algorithm_Time := Algorithm_Time + restime(starttime, finishtime);
```

```

{Аналогічно міряємо час для побічної діагоналі}
withStartTimedo
GetTime(Hours, Minutes, Seconds, HSeconds);

for iiii := 1 to 1000 do
begin
{Шукаємо у побічній діагоналі}
L := 1; R := n;
while (L <= R) do
begin
i := (L + R) div 2;
if Cube[k, i, n - i + 1] = Element2 then Break
else if Cube[k, i, n - i + 1] > Element2 then L := i + 1
else R := i - 1;
end;

if L <= R then
CoorMinor[k] := i
else CoorMinor[k] := 0;

end;
withFinishTimedo
GetTime(Hours, Minutes, Seconds, HSeconds);

Algorithm_Time := Algorithm_Time + restime(starttime, finishtime);

end;
end;

{Procedure with binary search N2}
procedure BinSearch2(const Cube: Mass; const n, p, Element1, Element2: integer);
{Процедура для пошуку №2}
{Усі дії проводимо аналогічно до процедури BinSearch2}
var
iii: longint;
begin

clrscr;
gotoxy(33, 10);
writeln('Searching...');
Algorithm_Time := 0;

fork := 1 to pdo
begin

withStartTimedo
GetTime(Hours, Minutes, Seconds, HSeconds);

for iiii := 1 to 1000 do
begin
{Main diagonal}
L := 1; R := n;
while (L < R) do
begin
i := (L + R) div 2;
if Cube[k, i, i] < Element1 then L := i + 1
else R := i;
end;

if Cube[k, R, R] = Element1 then
CoorMain[k] := R
else CoorMain[k] := 0;

end;
withFinishTimedo
GetTime(Hours, Minutes, Seconds, HSeconds);

```

```
Algorithm_Time := Algorithm_Time+restime(starttime, finishtime);
```

```
withStartTimedo
```

```
GetTime(Hours, Minutes, Seconds, HSeconds);
```

```
foriii := 1 to 1000 do
```

```
begin
```

```
{Minor diagonal}
```

```
L := 1; R := n;
```

```
while (L < R) do
```

```
begin
```

```
i := (L + R) div 2;
```

```
if Cube[k, i, n - i + 1 ] > Element2 then L := i + 1
```

```
else R := i;
```

```
end;
```

```
if Cube[k, R, n - R + 1 ] = Element2 then
```

```
CoorMinor[k] := R
```

```
elseCoorMinor[k] := 0;
```

```
end;
```

```
withFinishTimedo
```

```
GetTime(Hours, Minutes, Seconds, HSeconds);
```

```
Algorithm_Time := Algorithm_Time+restime(starttime, finishtime);
```

```
end;
```

```
end;
```

```
End.
```

Модуль Common

```
unitCommon;
```

```
{Модуль, уякомуістіятьсяглобальніструктуриданих}
```

```
interface
```

```
const
```

```
{Розміримасиву}
```

```
p = 112; n = 16;
```

```
{Спеціальнопідібраніключідляпошуку для забезпечення певної кількості порівнянь}
```

```
X1: array[1..5] of integer = (120, 52, 18, 1, 0);
```

```
X2: array[1..5] of integer = (136, 196, 226, 241, 0);
```

```
X3: array[1..5] of integer = (2016, 976, 66, 1, 0);
```

```
X4: array[1..5] of integer = (2080, 3088, 3970, 4033, 0);
```

```
X5: array[1..5] of integer = (8128, 1936, 517, 1, 0);
```

```
X6: array[1..5] of integer = (8256, 14352, 15749, 16257, 0);
```

```
type
```

```
{Масив в якому ми виконуємопошук}
```

```
mass = array[1..p, 1..n, 1..n] of integer;
```

```
var
```

```
{Масивикординат}
```

```
CoorMain: array[1..p] of integer;
```



```

CoordMinor: array[1..p] of integer;
Cube: mass;

{Змінні для лічильників, границь пошуку тощо }
i, j, k, L, R: integer;
item1, item2: byte;

```

implementation

end.

Модуль **Time**

```

unit Time;
{Модуль для збереження типів, констант та функцій, необхідних для
виміру часу}
interface

```

uses

crt;

```

{Створюємо запис змінних для запису в них значень
годин, хвилин, секунд, сотих секунд}

```

type

TTime = **record**

Hours, Minutes, Seconds, HSeconds: **Word**

end;

```

{Змінні для збереження часу на початок відліку часу, кінець відліку
та різниці}

```

var

StartTime, FinishTime: TTime;

Algorithm_Time: **integer**;

function ResTime(**const** STime, FTime: TTime): **longint**;

implementation

uses Dos; {Підключаємо модуль DOS, оскільки в ньому міститься процедура GetTime}

function ResTime(**const** STime, FTime: TTime): **longint**;

{Переводимо різницю часу в соті долі секунди}

begin

```

ResTime := 36000 * LongInt(FTime.Hours) +
6000 * LongInt(FTime.Minutes) +
100 * LongInt(FTime.Seconds) +
LongInt(FTime.HSeconds) -
36000 * LongInt(STime.Hours) -
6000 * LongInt(STime.Minutes) -
100 * LongInt(STime.Seconds) -
LongInt(STime.HSeconds);

```

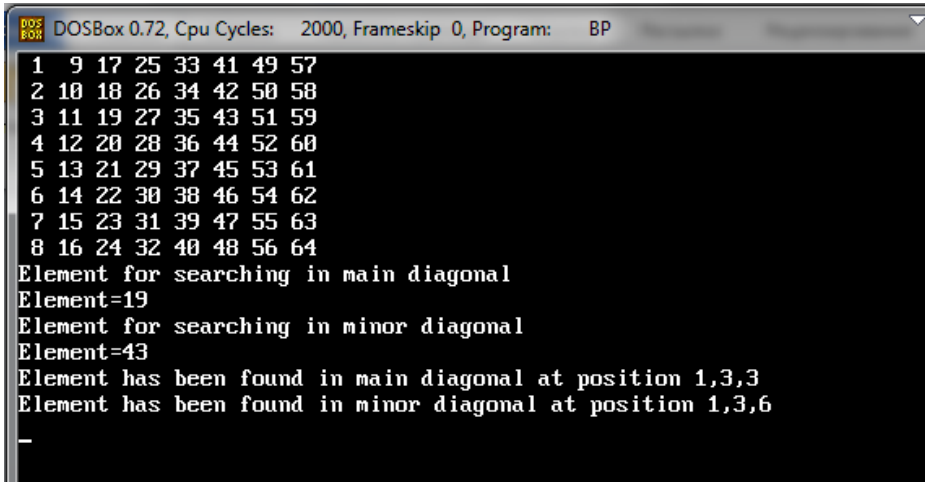
end;

end.

Тестування програми

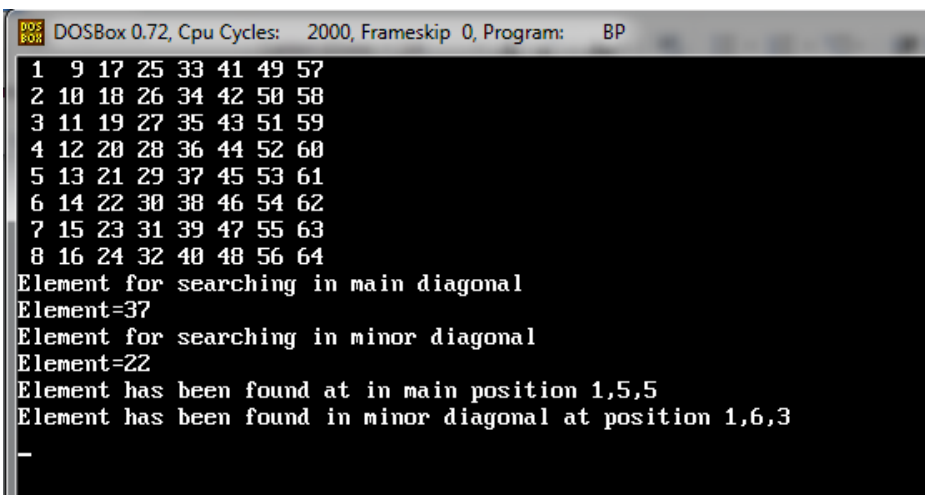
Перевіримо коректність роботи програми на невеликому двомірному масиві

Перевірка бінарного пошуку №1



```
DOSBox 0.72, Cpu Cycles: 2000, Frameskip 0, Program: BP
1 9 17 25 33 41 49 57
2 10 18 26 34 42 50 58
3 11 19 27 35 43 51 59
4 12 20 28 36 44 52 60
5 13 21 29 37 45 53 61
6 14 22 30 38 46 54 62
7 15 23 31 39 47 55 63
8 16 24 32 40 48 56 64
Element for searching in main diagonal
Element=19
Element for searching in minor diagonal
Element=43
Element has been found in main diagonal at position 1,3,3
Element has been found in minor diagonal at position 1,3,6
-
```

Перевірка бінарного пошуку №2



```
DOSBox 0.72, Cpu Cycles: 2000, Frameskip 0, Program: BP
1 9 17 25 33 41 49 57
2 10 18 26 34 42 50 58
3 11 19 27 35 43 51 59
4 12 20 28 36 44 52 60
5 13 21 29 37 45 53 61
6 14 22 30 38 46 54 62
7 15 23 31 39 47 55 63
8 16 24 32 40 48 56 64
Element for searching in main diagonal
Element=37
Element for searching in minor diagonal
Element=22
Element has been found at in main position 1,5,5
Element has been found in minor diagonal at position 1,6,3
-
```

Перевірка знаходження алгоритмом №1 першого елемента, що дорівнює шуканому

```
DOSBox 0.72, Cpu Cycles: 2000, Frameskip 0, Program: BP
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
Element for searching in main diagonal
Element=6
Element for searching in minor diagonal
Element=6
Element has been found in main diagonal at position 1,4,4
Element has been found in minor diagonal at position 1,5,4
-
```

Перевірка знаходження алгоритмом №2 першого елемента, що дорівнює шуканому

```
DOSBox 0.72, Cpu Cycles: 2000, Frameskip 0, Program: BP
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
6 6 6 6 6 6 6 6
Element for searching in main diagonal
Element=6
Element for searching in minor diagonal
Element=6
Element has been found at in main position 1,1,1
Element has been found in minor diagonal at position 1,8,1
```

Результати вимірів часу

CPU Cycles = 2000

Тест для масиву на **64** елементи($p=1$, $n=64$)

	¼ порівнянь	½ порівнянь	¾ порівнянь	Останне порівняння	Елемент відсутній
Пошук №1	11	27,5	72	82,1	99
Пошук №2	60,4	55	55	54,7	54

Тест для масиву на **28672** елементи($p=112$, $n=16$)

	¼ порівнянь	½ порівнянь	¾ порівнянь	Останне порівняння	Елемент відсутній
Пошук №1	1245	2862,8	4487,8	6114,2	7779,6
Пошук №2	4720,2	4684,2	4650,4	4620,4	4671

Тест для масиву на **28672** елементи($p=7$, $n=64$)

	¼ порівнянь	½ порівнянь	¾ порівнянь	Останне порівняння	Елемент відсутній
Пошук №1	77	179,3	485,3	582,3	692
Пошук №2	406,3	404,6	399,3	399	403

Тест для масиву на **16384** елементи($p=1$, $n=128$)

	¼ порівнянь	½ порівнянь	¾ порівнянь	Останне порівняння	Елемент відсутній
Пошук №1	11	40,3	99	98,7	113,7
Пошук №2	65,7	66	66	62,3	66

Порівняльний аналіз алгоритмів

1.Аналізуючи отримані дані ми підтвердили на практиці логарифмічну характеристику швидкодії бінарного пошуку $T=O(\log n)$, який при використанні на впорядкованих масивах виявляється значно швидше від лінійного пошуку у якого лінійна характеристика $T=O(n)$. Саме тому при збільшенні елементів у діагоналі навіть удвічі приріст часу незначний.

2.Результати для вектора є прогнозованими. Час пошуку як для вектору, так і для двовимірного масиву є майже ідентичний, адже пошук відбувається по діагоналі, яка і являє собою вектор.

3.Найбільше час залежить від кількості перерізів, що є досить передбачуваним, адже кількість перерізів слугує коефіцієнтом, який збільшує час виконання у певну кількість разів. Залежність часу від перерізів – лінійна. $T=p*O(\log n)$.

Висновки по отриманих результатах

В процесі виконання даної курсової роботи була досліджена швидкодія різних методів пошуку на тривимірному масиві. На основі даних про час роботи кожного з них були побудовані відповідні таблиці, на аналізуючи які отримані такі висновки:

Як вже вказано у теорії досліджували ми бінарний пошук №1 та бінарний пошук №2, який знаходить най лівіший з елементів, що рівні шуканому. Вони успішно використовуються на впорядкованих масивах будь-якого розміру, що є їх сильною стороною, адже залежність швидкодії від кількості елементів логарифмічна, а не лінійна, як у інших видів пошуку. Проте на багатовимірних масивах час зростає більш помітно, тому що залежність від кількості перерізів лінійна, тобто при збільшенні кількості перерізів у п час теж зростає приблизно у n разів. Це можна помітити якщо порівняти таблиці для **64** елементів ($p=1, n=64$) та **28672** елементів ($p=7, n=64$).

У приведених таблицях ми порівнювали випадки коли елемент було знайдено: одразу, після половини від кількості порівнянь, $\frac{3}{4}$ від цієї кількості, при останньому порівнянні та якщо елемент зовсім відсутній у зоні пошуку. Пошук номер один продемонстрував лінійну залежність від кількості порівнянь. На відміну від бінарного пошуку №2, у якого час для всіх випадків майже однаковий. Пояснюється це тим, що пошук закінчується не при знайденні елемента, а при перетині лівої та правої границь, для чого потребується максимальна кількість порівнянь $\log n$. Загалом час, що показує пошук №2, як правило, трохи більший за середній час пошуку №1 для всіх випадків. Це можна пояснити тим, що в алгоритмі №2 на одне порівняння менше. Тому, якщо елемент знаходиться у першій половині масиву, він показує себе краще.

Слід також враховувати, що другий пошук знаходить найлівіший елемент з усіх, що рівні шуканому. Тому при повторенні деякого елемента в масиві, перший пошук завершиться швидше. І його час буде тим більше, чим ближче ці елементи до центру масиву. Тому, якщо мати вичерпну інформацію про область пошуку можна використовувати то один, то інший способи, виграючи при цьому на часі.

Список використаної літератури

1. Методичні матеріали для виконання курсової роботи.
2. Конспект лекцій з курсу «Структури даних та алгоритми».
3. Підручник «Программирование в среде TurboPascal 7.0»