НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**КУРСОВА РОБОТА**

***з дисципліни "Структури даних і алгоритми"***

Виконав: Штефанович Г.М.

Група: КB-62

Номер залікової книжки: КВ-6227

Допущений до захисту

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 семестр 2016/2017

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Узгоджено ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_2017р.

Керівник роботи з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_/Марченко О.І./ \_\_\_\_\_\_\_\_\_ /Марченко О.І./

***Дослідження ефективності алгоритмів двійкового пошуку №1,2 на багатовимірних масивах***

Виконавець роботи:

Штефанович Георгій Миколайович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017р.

**ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

**I.** Описати принцип та схему роботи кожного із досліджуваних методів сортування або пошуку для одновимірного масиву.

**II.** Скласти алгоритми сортування або пошуку в багатовимірному масиві заданими методами, згідно до варіанту, та написати відповідну програму на мові програмування.

Програма повинна задовольняти наступним вимогам:

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 програми має бути режим запуску виміру часу всіх алгоритмів у пакетному режимі, тобто запуск всіх алгоритмів для всіх випадків і побудова результуючої таблиці за наведеним нижче зразком для масиву з 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 порівняльні таблиці за різними ознаками.

Для виконання ґрунтовного аналізу алгоритмів потрібно виконати виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами, а також для стандартного випадку одномірного масиву. Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибирається самостійно студентом так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів. За отриманими результатами можуть бути також побудовані графіки для наочності подання інформації.

**VI.** Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами:

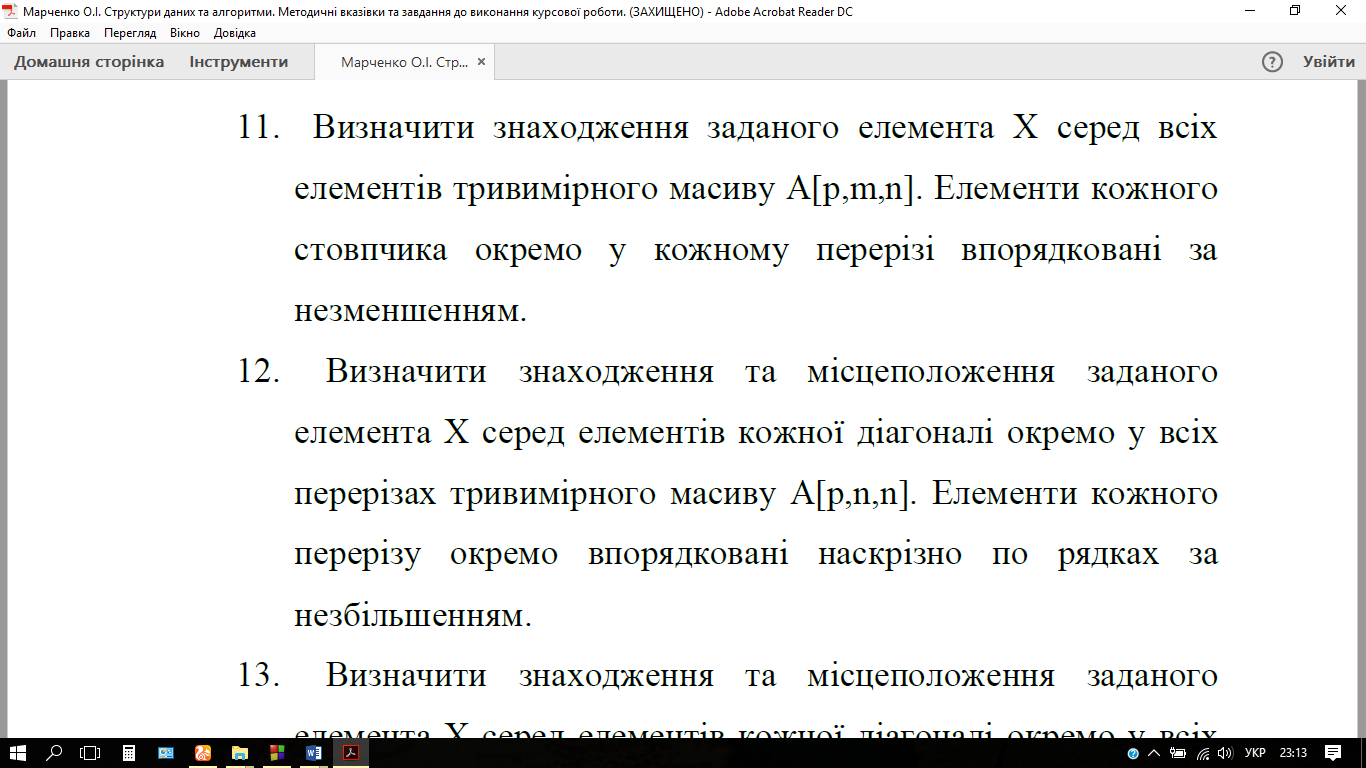
* для одномірного масиву відносно загальновідомої теорії;
* для багатовимірних масивів відносно результатів для одномірного масиву;
* для заданих алгоритмів на багатовимірних масивах між собою;
* дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою;
* для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше.

**VII.** Зробити висновки за виконаним порівняльним аналізом.

**VIII.** Програму курсової роботи під час її захисту ОБОВ’ЯЗКОВО мати при собі на електронному носії інформації.

**Варіант №80**

**Задача**



**Досліджувані методи та алгоритми**

1. Алгоритм двійкового пошуку №1, що знаходить випадковий елемент, з тих що співпадають з шуканим елементом.
2. Алгоритм двійкового пошуку №2 , що знаходить найлівіший елемент з тих , що співпадають з шуканим елементом.

**Випадки дослідження**

1. Потрібний елемент знаходиться після ¼ максимально можливого числа порівнянь в області пошуку.
2. Потрібний елемент знаходиться після ½ максимально можливого числа порівнянь в області пошуку.
3. Потрібний елемент знаходиться після ¾ максимально можливого числа порівнянь в області пошуку.
4. Потрібний елемент знаходиться при останньому порівнянні в області пошуку.
5. Потрібного елемента немає в області пошуку.

**Теоретичні відомості**

*Алгоритм двійкового пошуку №1*

Алгоритми двійкового пошуку використовуються в тому разі коли елементи множини / вектора впорядковані за незменшенням або незбільшенням

*Принцип роботи алгоритму :*

1. Беремо середній елемент масиву і порівнюємо із заданим Х.
2. Якщо вони дорівнюють, то пошук закінчується.
3. Якщо шуканий Х<A[i], всі елементи правіше більше від Х , отже немає сенсу шукати справа від А[i] і елементи виключають з пошуку , якщо шуканий Х>A[i] виключають з пошуку елементи менше A[i] , включно з ним.
4. Повторюємо пункти 1,3 для тієї частини , де є сенс продовжувати пошук , доки не буде знайдений Х , або діапазон пошуку не зменшиться до 0 елементів . В останньому випадку Х не знайдемо.

А 0 n-1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 | 4 | 6 | 6 | 7 | 10 | 11 | 11 | 12 | 14 | 15 |

L R

X=4 i = [L+R]/2 = 6; A[i]>X => R=i-1=5;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 | 4 | 6 | 6 |

L R

i=[L+R]/2 = 2; A[i]<X => L=i+1=3;

|  |  |  |
| --- | --- | --- |
| 4 | 6 | 6 |

|  |
| --- |
| 4 |

L R

i=[L+R]/2 = 4; A[i]>X => R=i-1=3; L=R

i=3 A[i] = 4;

***Код алгоритму:***

int main ()

{

int i, L, R, x, n;

int A[n];

printf(“Enter x : “);

scanf(“%d”, &x);

L=0; R=n-1;

while (L<=R)

{

i=(L+R)/2;

if (A[i]==x) break;

else if (A[i]>x) L=i+1;

else R=i-1;

}

if (A[i]==x) printf("Element found on position : %d",i);

else printf("Element did not found\n");

return 0;

}

*Алгоритм двійкового пошуку №2*

Загальний принцип пошуку залишається таким же , як і у алгоритму двійкового пошуку номер 1, але конкретні дії виконуються трохи інакше. На відміну від алгоритму двійкового пошуку номер 1, цей алгоритм знаходить найлівіший елемент, що співпадає з шуканим. В разі відсутності шуканого елемента в масиві індекси L та R будуть вказувати на останній елемент в масиві.

*Схема алгоритму :*

*A 0 1 2 3 4 5 6 7 8 n-1=9*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 5 | 8 | 10 | 12 | 13 | 13 | 14 |

L R

X= 5; i= [L+R]/2=4; A[i]>X=>R=i=4;

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 5 | 8 |

L R

i=[L+R]/2=2; A[i]=X => R=i=2;

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 5 |

L R

i=[L+R]/2 = 1; A[i] < X => L=i+1=2

|  |
| --- |
| 5 |

L=R

i=2; A[i] = 5;

***Код алгоритму:***

int main()

{

int i, L, R, n, x;

int A[n];

printf(“Enter x : “);

scanf(“%d”, &x);

L=0; R=n-1;

while (L<R)

{

i=(L+R)/2;

if (A[i]<x) L=i+1;

else R=i;

}

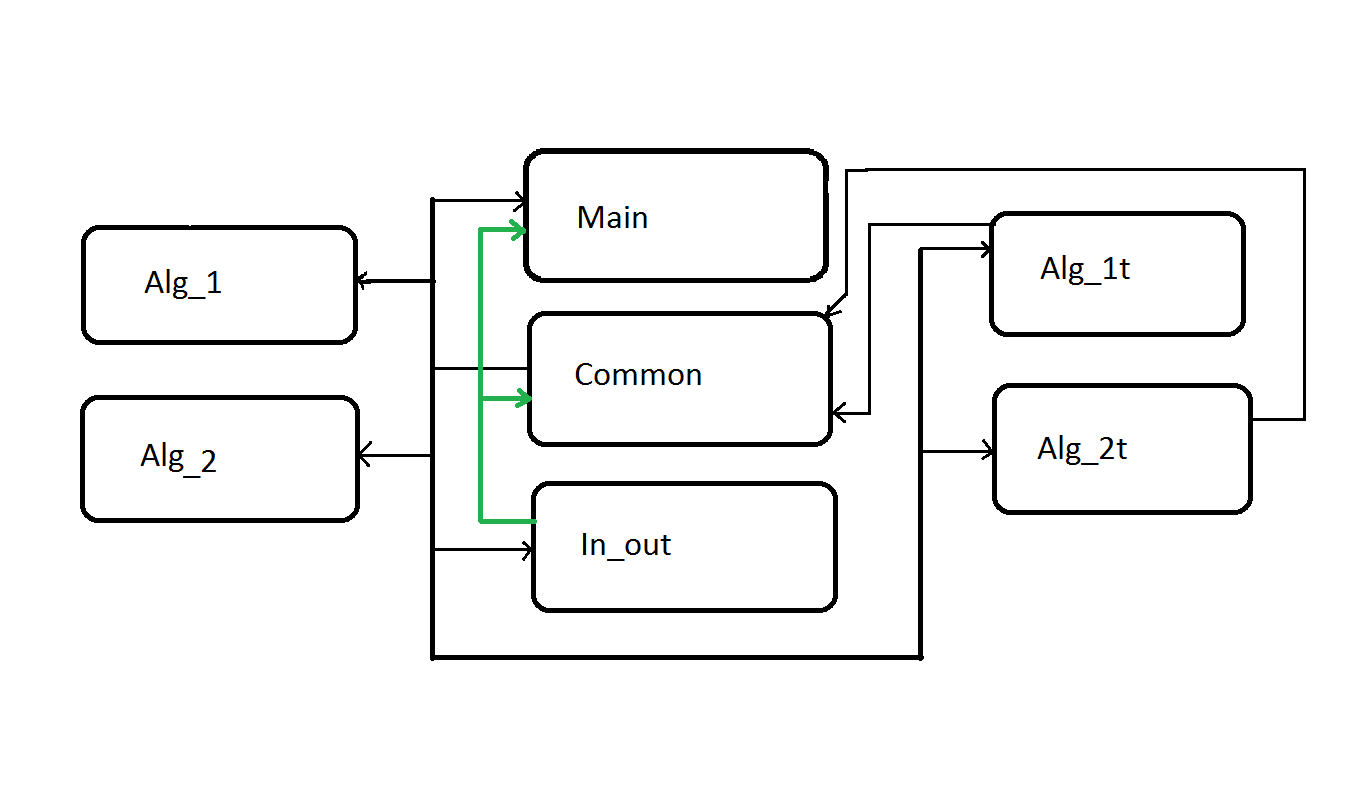
if (A[R]==x)

printf("Element found on position : %d", R);

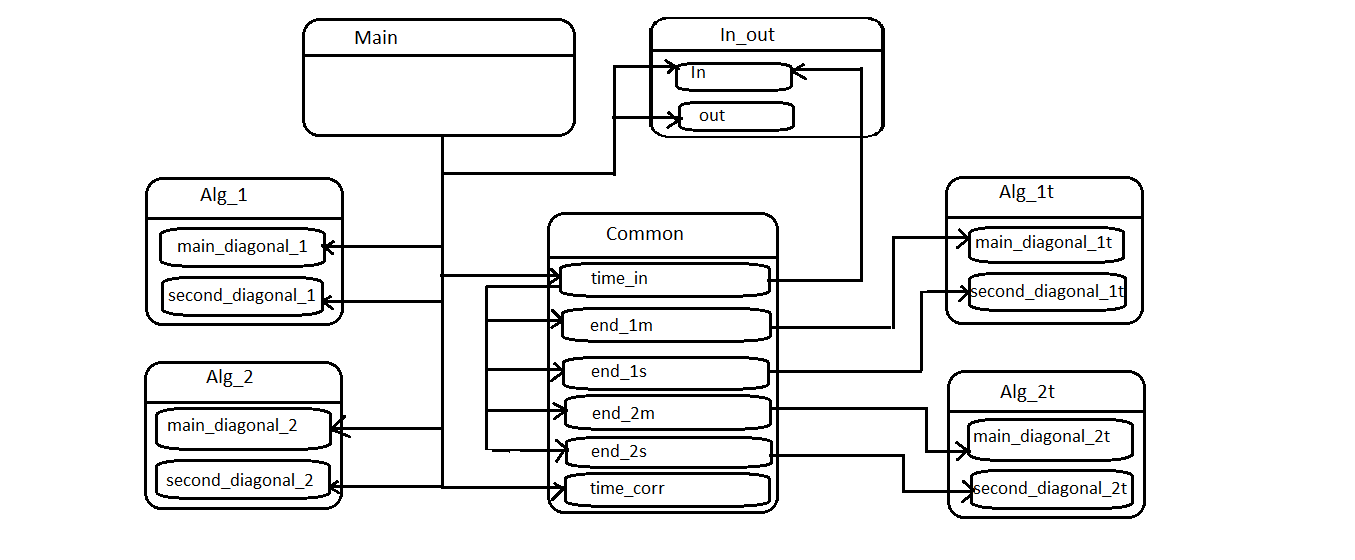
else printf("Element did not found\n");

return 0;

}

**Схема імпорту/експорту модулів**

**Структурна схема взаємовикликів процедур та функцій**

****

**Опис призначення процедур та функцій**

1. Модуль Common
   1. **time\_in -** функція, яка передає в змінні значення функцій з виміру часу для конкретних алгоритмів
   2. **end\_1m -** ф-ція, що визначає час на вказаному етапі роботи програми
   3. **end\_1s –** аналогічно **end\_1m**
   4. **end\_2m -** аналогічно **end\_1m**
   5. **end\_2s -** аналогічно **end\_1m**
2. Модуль Alg\_1
   1. **main\_diagonal\_1 -** алгоритм двійкового пошуку №1 для головної діагоналі
   2. **second\_diagonal\_1 -** алгоритм двійкового пошуку №1 для побічної діагоналі
3. Модуль Alg\_2
   1. **main\_diagonal\_2 -** алгоритм двійкового пошуку №2 для головної діагоналі
   2. **second\_diagonal\_2 -** алгоритм двійкового пошуку №2 для побічної діагоналі
4. Модуль Alg\_1t
   1. **main\_diagonal\_1t -** ф-ція що визначає елемент на певній кількості порівнянь і передає його в ф-цію для знаходження часу роботу цього ж алгоритму
   2. **second\_diagonal\_1t –** аналогічно функції **main\_diagonal\_1t**
5. Модуль Alg\_2t
   1. **main\_diagonal\_2t –** аналогічно функції **main\_diagonal\_1t**
   2. **second\_diagonal\_2t –** аналогічно функції **main\_diagonal\_1t**
6. Модуль in\_out
   1. **In –** функція, що вводить елементи масиву
   2. **Out –** функція, що виводить елементи масиву

**Текст головної програми та модулів**

**Main.c**

#include "Common.h"

#include "in\_out.h"

int main()

{

int a, b;

printf ("Please choose yours mode:\n");

printf ("Press 1 if you want to test algorithmes\n");

printf ("Press 2 if you want to see time of work algorithmes\n");

printf ("And your choise is... "); scanf("%d", &a);

switch (a)

{

case 1:

{

system("cls");

printf("Please choose algorithm you want:\n");

printf("Press 1 if you want to find element whith algorithm 1\n");

printf("Press 2 if you want to find element whith algorithm 2\n");

printf ("And your choise is... "); scanf("%d", &b);

switch (b)

{

case 1:

{

system("cls");

printf("Input element X:");

scanf("%d", &X);

in(); //вводимо масив

//out();

printf("\nFounded elements at main diagonal: \n");

main\_diagonal\_1(); // визначаємо місцезнаходження елемента на головній діагоналі

printf("\n\nFounded elements at secondary diagonal: \n");

second\_diagonal\_1(); // визначаємо місцезнаходження елемента на побічній діагоналі

break;

}

case 2:

{

system("cls");

printf("Input element X:");

scanf("%d", &X);

in(); //вводимо масив

//out();

printf("\nFounded elements at main diagonal: \n");

main\_diagonal\_2(); // визначаємо місцезнаходження елемента на головній діагоналі

printf("\n\nFounded elements at secondary diagonal: \n");

second\_diagonal\_2(); // визначаємо місцезнаходження елемента на побічній діагоналі

break;

}

}

break;

}

case 2:

{

system("cls");

in();

res1 = (clock\_t\*)malloc(t\*sizeof(clock\_t)); //виділення пам"яті для динамічних масивів, які ми

res2 = (clock\_t\*)malloc(t\*sizeof(clock\_t)); //об"являли в common.c

res3 = (clock\_t\*)malloc(t\*sizeof(clock\_t));

res4 = (clock\_t\*)malloc(t\*sizeof(clock\_t));

printf("\n Wait! Program is compiling!");

printf("\n\n|---------------------Element is on main diagonal---------------------------|\n");

printf("|---------------------------------------------------------------------------|\n");

printf("|Comparisoning | 1/4 | 1/2 | 3/4 | Last |No element|\n");

printf("|Searching #1 |");

X=p\*n\*n+1; //задається значення елемента, який вподальшому буде знаходитись в алгоритмах з

//виміру часу. він спеціально виходить за межі масива, для того, щоб ф-ція

//виконала точну кількість порівнянь і елемент не знайшовся раніше ніж потрібно.

//в подальшому значення Х заміниться значенням потрібного елемента масиву

//визначаємо час роботи алгоритму для головної діагоналі алгоритму двійкового пошуку №1 на всіх частинах порівнянь

time\_in(4); //викликається ф-ція для визначення часу на 1/4 від загальної кількості порівнянь

\*(res1+0) = time\_corr(&res1[0]);//запускається ф-ція для точного визначення часу роботи алгоритму,

printf(" %.3f |", (float)\*(res1+0)/20);// присвоюється значення нульвому елементу динамічного

time\_in(2); //визначення часу на 1/2 порівнянь //масиву та виводимо це значення

\*(res1+0) = time\_corr(&res1[0]);

printf(" %.3f |", (float)\*(res1+0)/20);

time\_in(4/3); //визначення часу на 3/4 порівнянь

\*(res1+0) = time\_corr(&res1[0]);

printf(" %.3f |", (float)\*(res1+0)/20);

time\_in(1); //визначення часу при останньому порівнянні

\*(res1+0) = time\_corr(&res1[0]);

printf(" %.3f |", (float)\*(res1+0)/20);

time\_in(-1); //визначення часу коли елемент не знайдено

\*(res1+0) = time\_corr(&res1[0]);

printf(" %.3f |", (float)\*(res1+0)/20);

free(res1); //звільнюється пам"ять для одного з динамічних масивів

//визначаємо час роботи алгоритму для головної діагоналі алгоритму двійкового пошуку №2 на всіх частинах порівнянь

printf("\n|Searching #2 |"); // принцип аналогічний діям, зазначених вище

time\_in(4);

\*(res3+0) = time\_corr(&res3[0]);

printf(" %.3f |", (float)\*(res3+0)/20);

time\_in(2);

\*(res3+0) = time\_corr(&res3[0]);

printf(" %.3f |", (float)\*(res3+0)/20);

time\_in(4/3);

\*(res3+0) = time\_corr(&res3[0]);

printf(" %.3f |", (float)\*(res3+0)/20);

time\_in(1);

\*(res3+0) = time\_corr(&res3[0]);

printf(" %.3f |", (float)\*(res3+0)/20);

time\_in(-1);

\*(res3+0) = time\_corr(&res3[0]);

printf(" %.3f |", (float)\*(res3+0)/20);

free(res3);

printf("\n|---------------------------------------------------------------------------|\n");

printf("\n\n|--------------------Element is on second diagonal--------------------------|\n");

printf("|---------------------------------------------------------------------------|\n");

printf("|Comparisoning | 1/4 | 1/2 | 3/4 | Last |No element|\n");

printf("|Searching #1 |");

//визначаємо час роботи алгоритму для побічної діагоналі алгоритму двійкового пошуку №1 на всіх частинах порівнянь

time\_in(4);

\*(res2+0) = time\_corr(&res2[0]);

printf(" %.3f |", (float)\*(res2+0)/20);

time\_in(2);

\*(res2+0) = time\_corr(&res2[0]);

printf(" %.3f |", (float)\*(res2+0)/20);

time\_in(4/3);

\*(res2+0) = time\_corr(&res2[0]);

printf(" %.3f |", (float)\*(res2+0)/20);

time\_in(1);

\*(res2+0) = time\_corr(&res2[0]);

printf(" %.3f |", (float)\*(res2+0)/20);

time\_in(-1);

\*(res2+0) = time\_corr(&res2[0]);

printf(" %.3f |", (float)\*(res2+0)/20);

free(res2);

//визначаємо час роботи алгоритму для побічної діагоналі алгоритму двійкового пошуку №2 на всіх частинах порівнянь

printf("\n|Searching #2 |");

time\_in(4);

\*(res4+0) = time\_corr(&res4[0]);

printf(" %.3f |", (float)\*(res4+0)/20);

time\_in(2);

\*(res4+0) = time\_corr(&res4[0]);

printf(" %.3f |", (float)\*(res4+0)/20);

time\_in(4/3);

\*(res4+0) = time\_corr(&res4[0]);

printf(" %.3f |", (float)\*(res4+0)/20);

time\_in(1);

\*(res4+0) = time\_corr(&res4[0]);

printf(" %.3f |", (float)\*(res4+0)/20);

time\_in(-1);

\*(res4+0) = time\_corr(&res4[0]);

printf(" %.3f |", (float)\*(res4+0)/20);

free(res4);

printf("\n|---------------------------------------------------------------------------|\n");

break;

}

}

printf("\n");

system("pause");

return 0;}

**Common.h**

#ifndef \_\_Common\_H\_\_

#define \_\_Common\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#define p 2

#define n 14000

#define t 28

int arr[p][n][n];

extern clock\_t \*res1, \*res2, \*res3, \*res4;

int X, L, R, I;

int i, j, k;

void time\_in (float a);

clock\_t time\_corr (clock\_t \*res);

clock\_t end\_1m(float a);

clock\_t end\_1s(float a);

clock\_t end\_2m(float a);

clock\_t end\_2s(float a);

#endif

**Common.c**

#include "Common.h"

#include "in\_out.h"

#include "Alg\_1t.h"

#include "Alg\_2t.h"

clock\_t \*res1, \*res2, \*res3, \*res4; //створення динамічних масивів

void time\_in (float a) //функція, яка передає в змінні значення функцій з виміру часу для конкретних алгоритмів

{

int i;

for (i=0; i<t; i++)

{

in();

\*(res1+i) = end\_1m(a);

\*(res2+i) = end\_1s(a);

\*(res3+i) = end\_2m(a);

\*(res4+i) = end\_2s(a);

}

}

//визначення часу на головній діагоналі першого алгоритму

clock\_t end\_1m(float a) //ф-ція, що визначає час на вказаному етапі роботи програми.

{ //параметр а використовується для того , щоб передати значення в функцію, в якій визначається

clock\_t time\_start, time\_stop; //елемент, знайдений на певній кількості порівнянь, який буде знайдений в даній ф-ції

//визначаєм час знаходження цього елементу на діагоналі та передаємо це значення

X = main\_diagonal\_1t(a); //змінним з ф-ції time\_in

for (i=0; i<p; i++)

{

L=0; R=n-1;

time\_start = clock();

while (L<=R)

{

I=(L+R)/2;

if (arr[i][I][I]==X) break;

else

{

if (arr[i][I][I]>X) L=I+1;

else R=I-1;

}

}

time\_stop = clock();

}

return time\_stop - time\_start;

}

//визначення часу на побічній діагоналі першого алгоритму

clock\_t end\_1s(float a) // принцип роботи даної ф-ції такий самий як і end\_1m

{

clock\_t time\_start, time\_stop;

X = second\_diagonal\_1t(a);

for (i=0; i<p; i++)

{

L=0; R=n-1;

time\_start = clock();

while (L<=R)

{

I=(L+R)/2;

if (arr[i][I][I]==X) break;

else

{

if (arr[i][I][I]>X) L=I+1;

else R=I-1;

}

}

time\_stop = clock();

}

return time\_stop - time\_start;

}

//визначення часу на головній діагоналі другого алгоритму

clock\_t end\_2m(float a) // принцип роботи даної ф-ції такий самий як і end\_1m

{

clock\_t time\_start, time\_stop;

X = main\_diagonal\_2t(a);

for (i=0; i<p; i++)

{

L=0; R=n-1;

time\_start = clock();

while (L<R)

{

I=(L+R)/2;

if (arr[i][I][n-1-I]>X) L=I+1;

else R=I;

}

time\_stop = clock();

}

return time\_stop - time\_start;

}

//визначення часу на побічній діагоналі другого алгоритму

clock\_t end\_2s(float a) // принцип роботи даної ф-ції такий самий як і end\_1m

{

clock\_t time\_start, time\_stop;

X = second\_diagonal\_2t(a);

for (i=0; i<p; i++)

{

L=0; R=n-1;

time\_start = clock();

while (L<R)

{

I=(L+R)/2;

if (arr[i][I][I]>X) L=I+1;

else R=I;

}

time\_stop = clock();

}

return time\_stop - time\_start;

}

//функія для точного визначення часу

clock\_t time\_corr(clock\_t \*res) //в дану ф-цію передаються масиви з ф-ції time\_in над якими будуть

{ //виконуватись подальші дії і повертатиметься остаточний результат

long int Sum,Min1,Min2,Min3,Max1,Max2,Max3;

int imin1,imin2,imin3,imax1,imax2,imax3;

Sum = \*(res+2); Min1 = \*(res+2); Max1 = \*(res+2); imin1 = 2; imax1 = 2;

for (i = 3; i < t; i++)

{

Sum = Sum + \*(res+i);

if (\*(res+i) > Max1) { Max1 = \*(res+i); imax1 = i; }

else if (\*(res+i) < Min1) { Min1 = \*(res+i); imin1 = i; }

}

res[imin1]=-1; res[imax1]=-1;

if (\*(res+2) == -1) {Min2 = \*(res+3); imin2 = 3;}

else {Min2 = \*(res+2); imin2 = 2;}

Max2 = \*(res+2); imax2 = 2;

for (i = 3; i < t; i++)

{

if (\*(res+i) > Max2) { Max2 = \*(res+i); imax2 = i; }

else if (\*(res+i)<Min2 && \*(res+i)!=-1) { Min2 = \*(res+i); imin2 = i; }

}

res[imin2]=-1; res[imax2]=-1;

if (\*(res+2) == -1)

if (\*(res+3) == -1) {Min3 = \*(res+4); imin3 = 4;}

else {Min3 = \*(res+3); imin3 = 3;}

else {Min3 = \*(res+2); imin3 = 2;}

Max3 = \*(res+2); imax3 = 2;

for (i = 3; i < t; i++)

{

if (\*(res+i) > Max3) { Max3 = \*(res+i); imax3 = i; }

else if (\*(res+i)<Min3 && \*(res+i)!=-1) { Min3 = \*(res+i); imin3 = i; }}

return Sum - Min1 - Min2 - Min3 - Max1 - Max2 - Max3;}

**In\_out.h**

#ifndef \_\_in\_out\_h\_\_

#define \_\_in\_out\_h\_\_

void in();

void out();

#endif

**In\_out.c**

#include "Common.h"

#include "in\_out.h"

//модуль з ф-ціями для вводу і виводу тривимірного масиву

void in()

{

int y=0;

for (i=0; i<p; i++)

for (j=0; j<n; j++)

for (k=0; k<n; k++)

{

arr[i][j][k]=p\*n\*n-y;

y++;

}

}

void out()

{

for (i=0; i<p; i++){

printf("No %d", i+1);

printf("\n");

for (j=0; j<n; j++)

{

for (k=0; k<n; k++)

printf("%3d ", arr[i][j][k]);

printf("\n");

}

printf("\n");}

}

**Alg\_1.h**

#ifndef \_\_Alg\_1\_h\_\_

#define \_\_Alg\_1\_h\_\_

void in\_1();

void out\_1();

void main\_diagonal\_1();

void second\_diagonal\_1();

#endif

**Alg\_1.c**

#include "Common.h"

#include "Alg\_1.h"

void main\_diagonal\_1() // алгоритм двійкового пошуку №1 для головної діагоналі

{

for (i=0; i<p; i++)

{

L=0; R=n-1;

while (L<=R)

{

I=(L+R)/2;

if (arr[i][I][I]==X) break;

else

{

if (arr[i][I][I]>X) L=I+1;

else R=I-1;

}

}

if (L<=R) printf("\nElement is found at position: %d %d %d", i+1, I+1, I+1);

else printf("\nElement is not found!");

}

}

void second\_diagonal\_1() //алгоритм двійкового пошуку №1 для побічної діагоналі

{

for (i=0; i<p; i++)

{

L=0; R=n-1;

while (L<=R)

{

I=(L+R)/2;

if (arr[i][I][n-1-I]==X) break;

else

{

if (arr[i][I][n-1-I]>X) L=I+1;

else R=I-1;

}

}

if (L<=R) printf("\nElement is found at position: %d %d %d", i+1, I+1, n-I);

else printf("\nElement is not found!");

}

}

**Alg\_2.h**

#ifndef \_\_Alg\_2\_h\_\_

#define \_\_Alg\_2\_h\_\_

void in\_2();

void out\_2();

void main\_diagonal\_2();

void second\_diagonal\_2();

#endif

**Alg\_2.c**

#include "Common.h"

#include "Alg\_2.h"

void main\_diagonal\_2() //алгоритм двійкового пошуку №2 для головної діагоналі

{

for (i=0; i<p; i++)

{

L=0; R=n-1;

while (L<R)

{

I=(L+R)/2;

if (arr[i][I][I]>X) L=I+1;

else R=I;

}

if (arr[i][R][R]==X) printf("\nElement is found at position: %d %d %d", i+1, R+1, R+1);

else printf("\nElement is not found!");

}

}

void second\_diagonal\_2() //алгоритм двійкового пошуку №2 для побічної діагоналі

{

for (i=0; i<p; i++)

{

L=0; R=n-1;

while (L<R)

{

I=(L+R)/2;

if (arr[i][I][n-1-I]>X) L=I+1;

else R=I;

}

if (arr[i][R][n-1-R]==X) printf("\nElement is found at position: %d %d %d", i+1, R+1, n-R);

else printf("\nElement is not found!");

}

}

**Alg\_1t.h**

#ifndef \_\_Alg\_1t\_h\_\_

#define \_\_Alg\_1t\_h\_\_

int main\_diagonal\_1t(float a);

int second\_diagonal\_1t(float a);

#endif

**Alg\_1t.c**

#include "Common.h"

#include "Alg\_1t.h"

//ф-ція для головної діагоналі алгоритму двійкового пошуку №1

int main\_diagonal\_1t(float a) //ф-ція що визначає елемент на певній кількості порівнянь і передає його в ф-цію

{ //для знаходження часу роботу цього ж алгоритму

int q; // в параметр а ми передаємо число на яке буде ділитися змінна h для визначення

float h; // кількості порівнянь

h=log2(n);

for (i=0; i<p; i++)

{

L=0; R=n-1;

q=0;

while (L<=R)

{

I=(L+R)/2;

if (arr[i][I][I]==X) break;

else

{

if (arr[i][I][I]>X) L=I+1;

else R=I-1;

}

if (q==h/a) return (arr[i][I][I]);

q++;

}

}

}

//ф-ція для побічної діагоналі алгоритму двійкового пошуку №1

int second\_diagonal\_1t(float a) //принцип роботи подібний ф-ції вище

{

float h;

int q;

h=log2(n);

for (i=0; i<p; i++)

{

L=0; R=n-1;

q=0;

while (L<=R)

{

I=(L+R)/2;

if (arr[i][I][n-1-I]==X) break;

else

{

if (arr[i][I][n-1-I]<X) L=I+1;

else R=I-1;

}

if (q == h/a) return arr[i][I][n-1-I];

q++;

}

}

}

**Alg\_2t.h**

#ifndef \_\_Alg\_2t\_h\_\_

#define \_\_Alg\_2t\_h\_\_

int main\_diagonal\_2t(float a);

int second\_diagonal\_2t(float a);

#endif

**Alg\_2t.c**

#include "Common.h"

#include "Alg\_2t.h"

//ф-ція для головної діагоналі алгоритму двійкового пошуку №1

int main\_diagonal\_2t(float a) //алгоритм аналогічний ф-ції main\_diagonal\_1t

{

int q;

float h;

h=log2(n);

for (i=0; i<p; i++)

{

L=0; R=n-1;

q=0;

while (L<R)

{

I=(L+R)/2;

if (arr[i][I][I]>X) L=I+1;

else R=I;

if (q == h/a) return arr[i][R][R];

q++;

}

}

}

//ф-ція для побічної діагоналі алгоритму двійкового пошуку №1

int second\_diagonal\_2t(float a)//алгоритм аналогічний ф-ції main\_diagonal\_1t

{

int q;

float h;

h=log2(n);

for (i=0; i<p; i++)

{

L=0; R=n-1;

q=0;

while (L<R)

{

I=(L+R)/2;

if (arr[i][I][n-1-I]<X) L=I+1;

else R=I;

if (q == h/a) return arr[i][R][R];

q++;

}

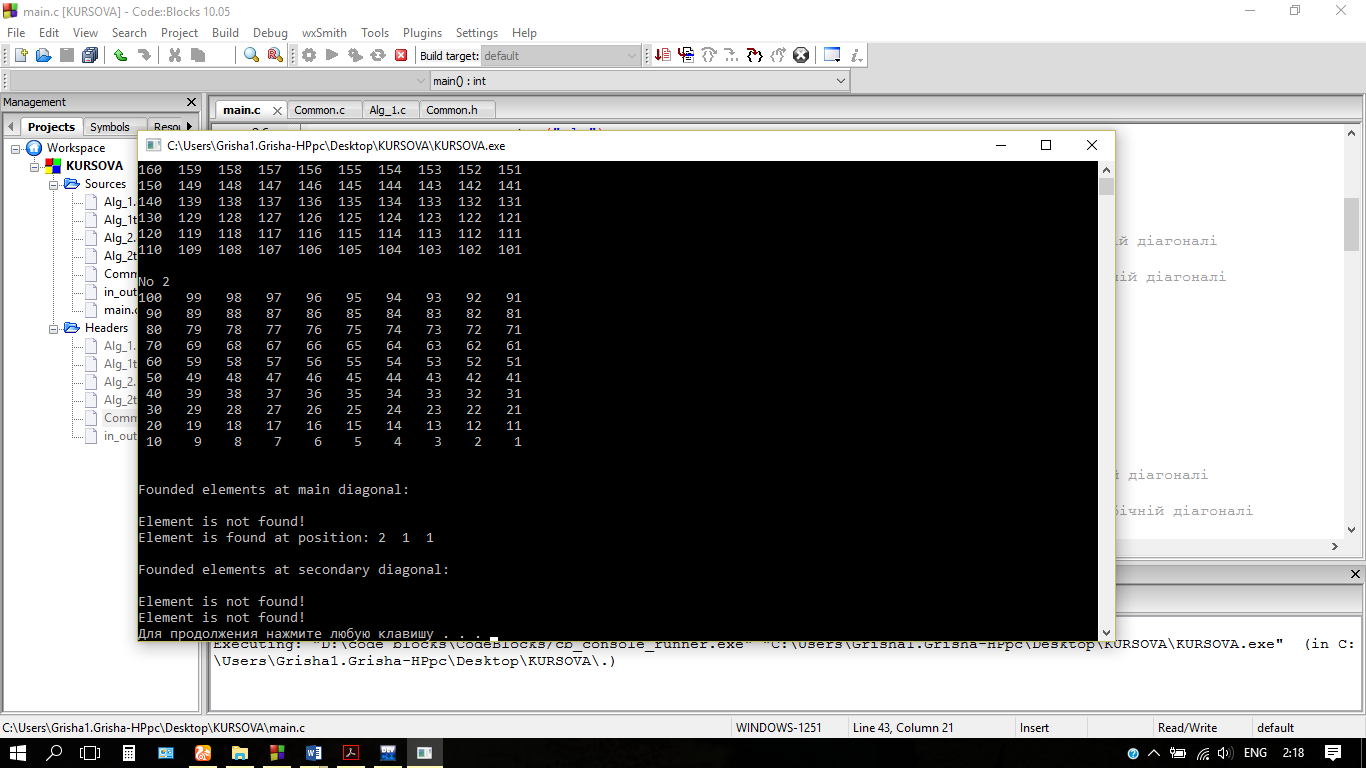
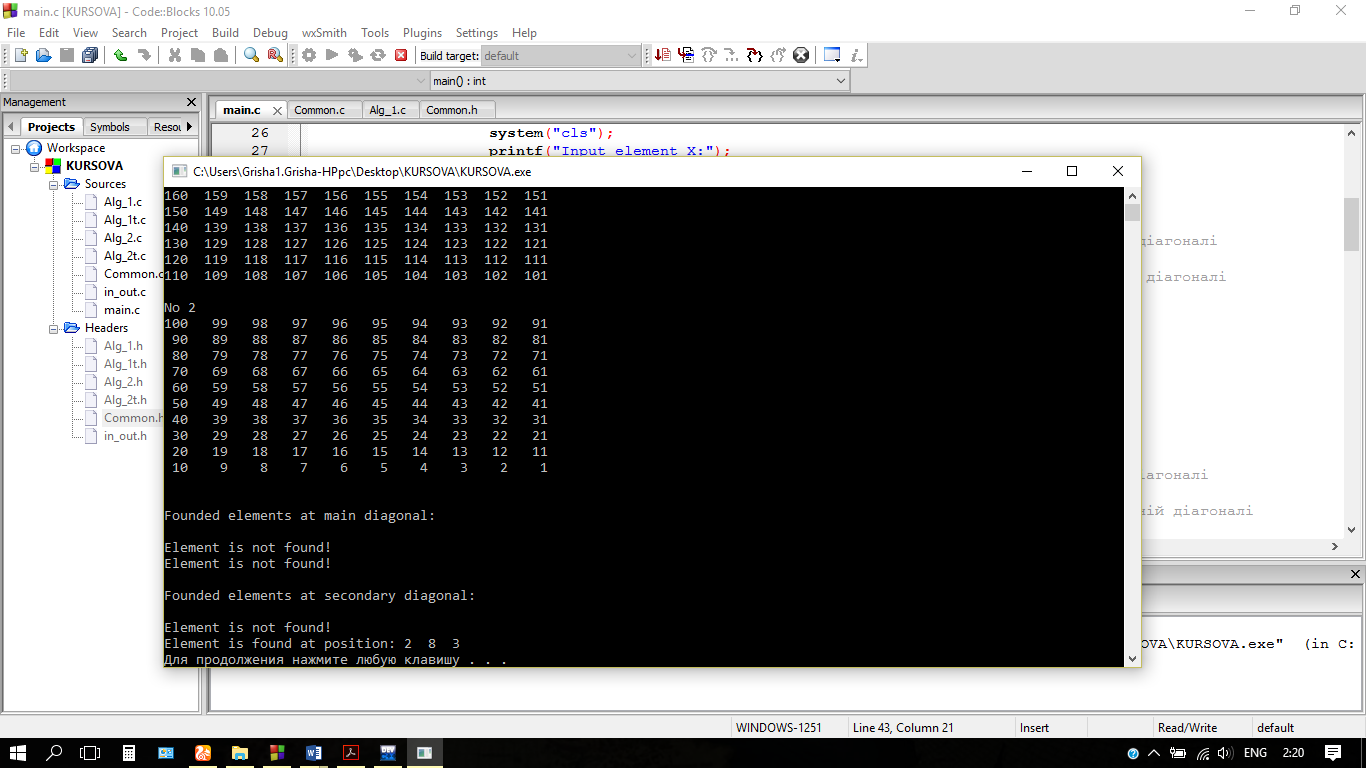
}

}

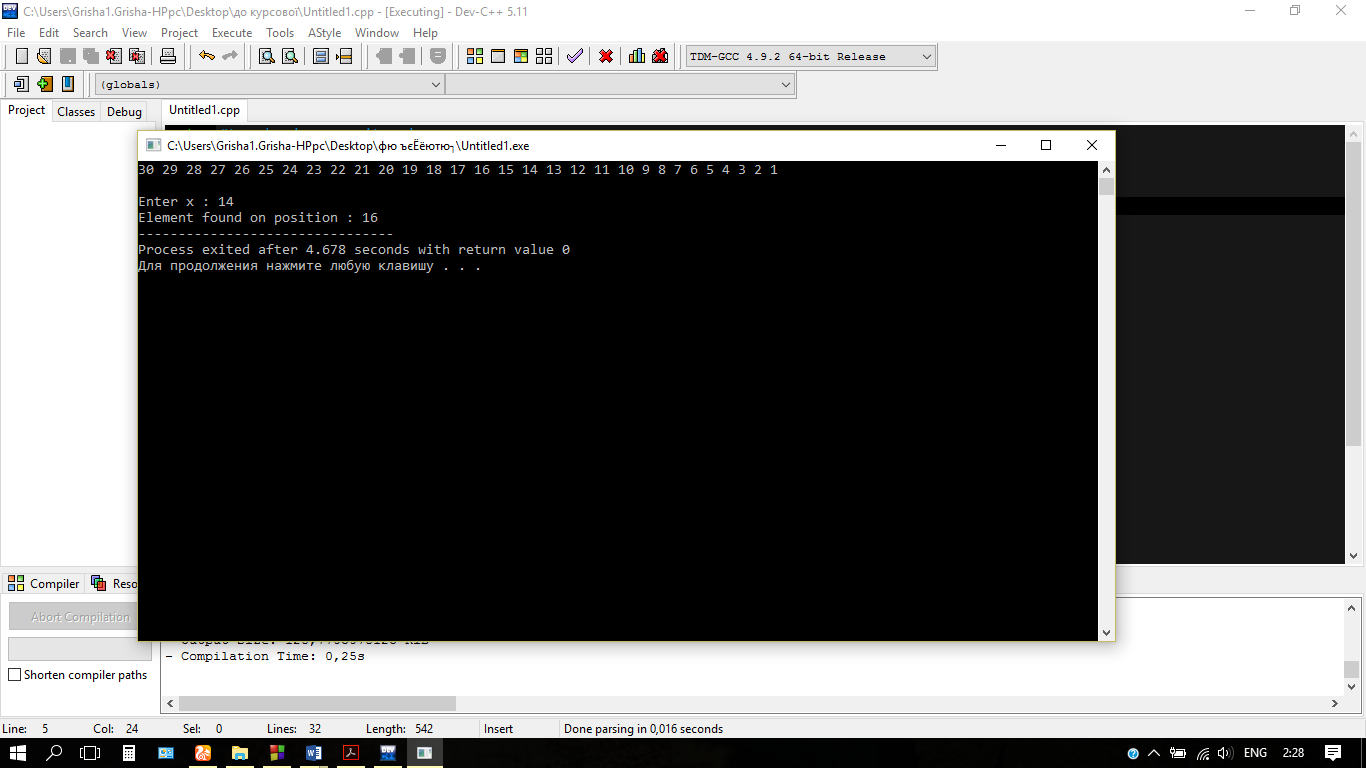
**Тести програми**

**1.Тести алгоритму двійкового пошуку №1**

A) для трьох вимірного масиву . (X=100, 28)

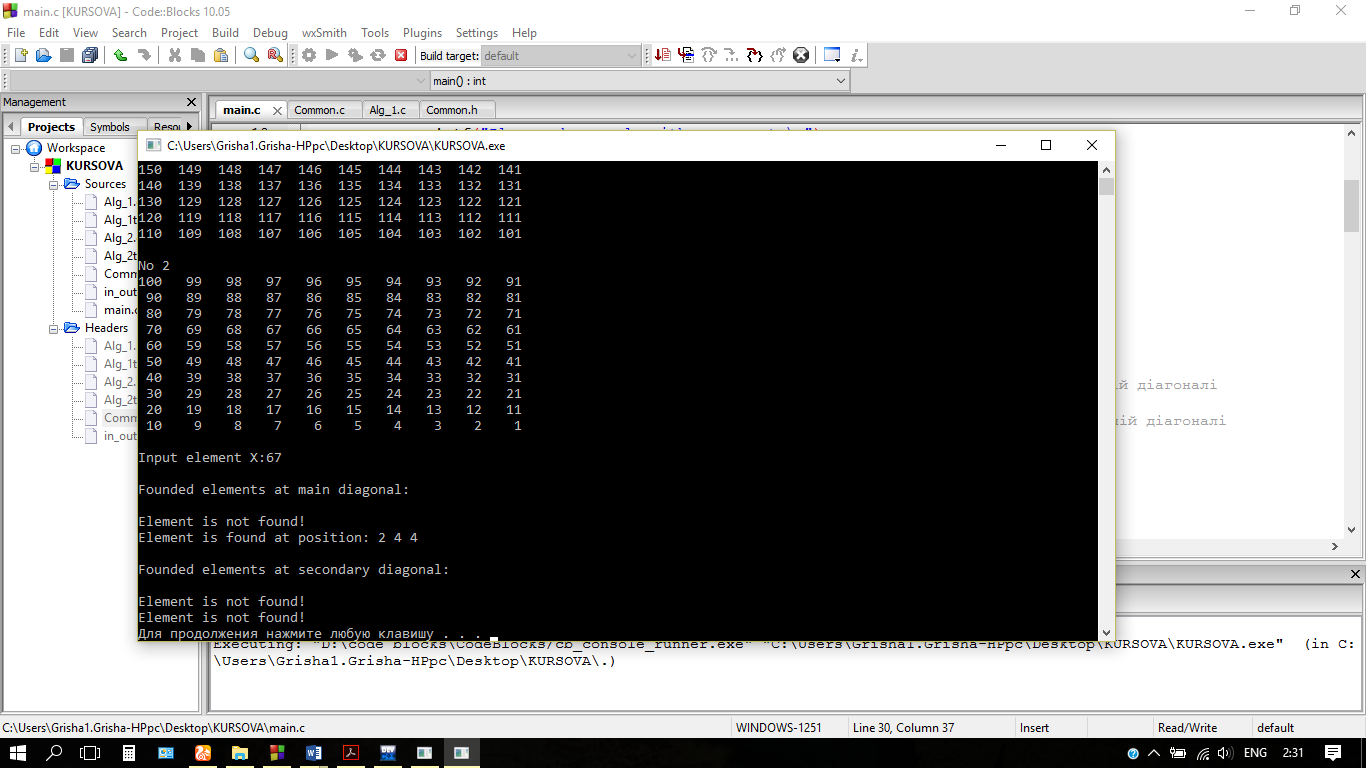
 

Б) для вектору .

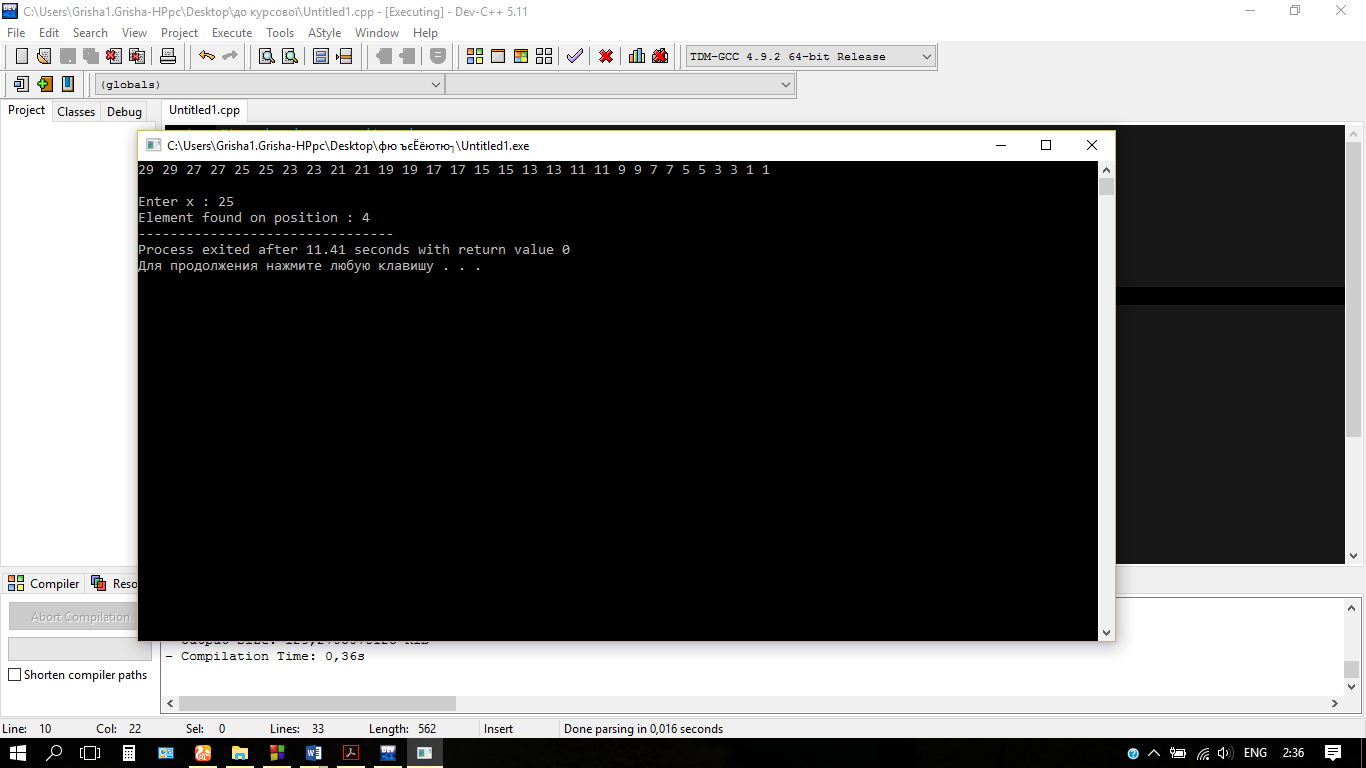


**2.Тести алгоритму двійкового пошуку №2**

А) для трьох вимірного масиву.

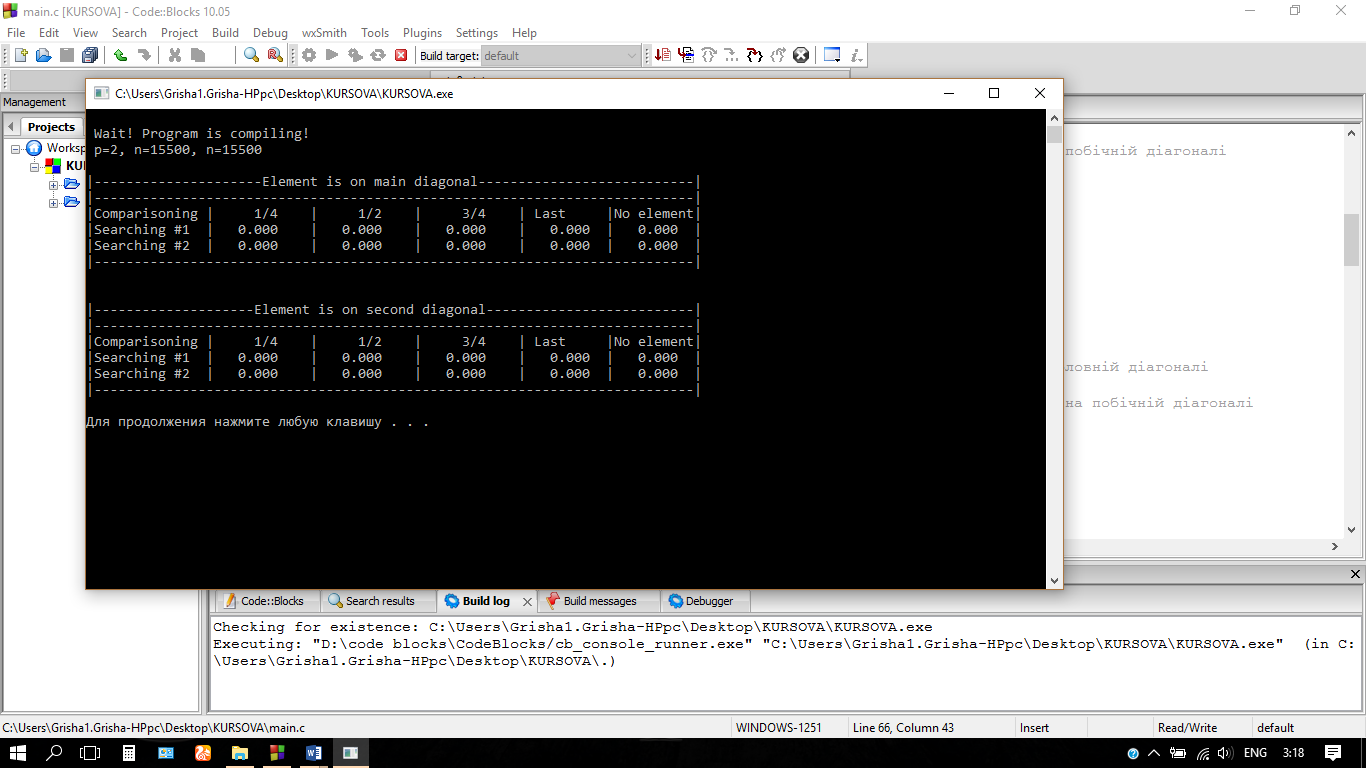


Б) для вектору .

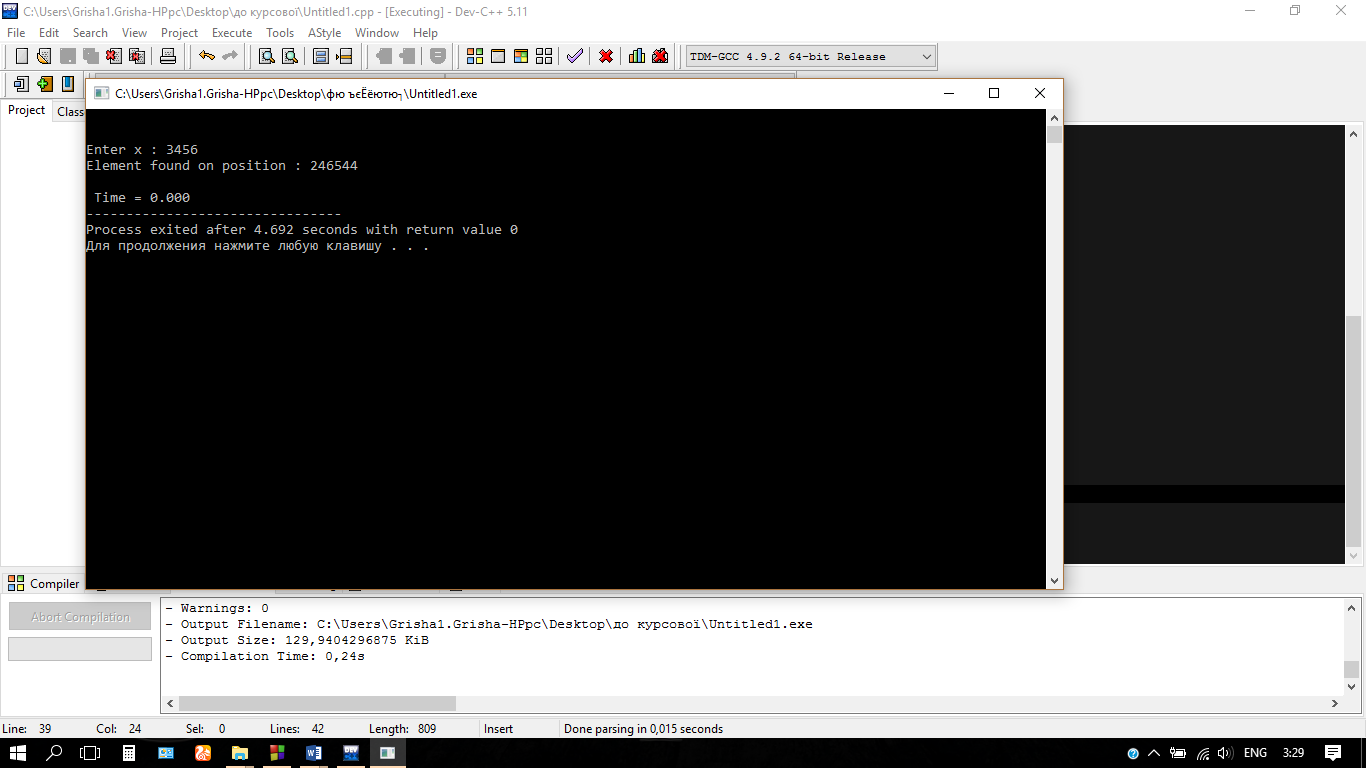


**3. Таблиці часу сортування.**

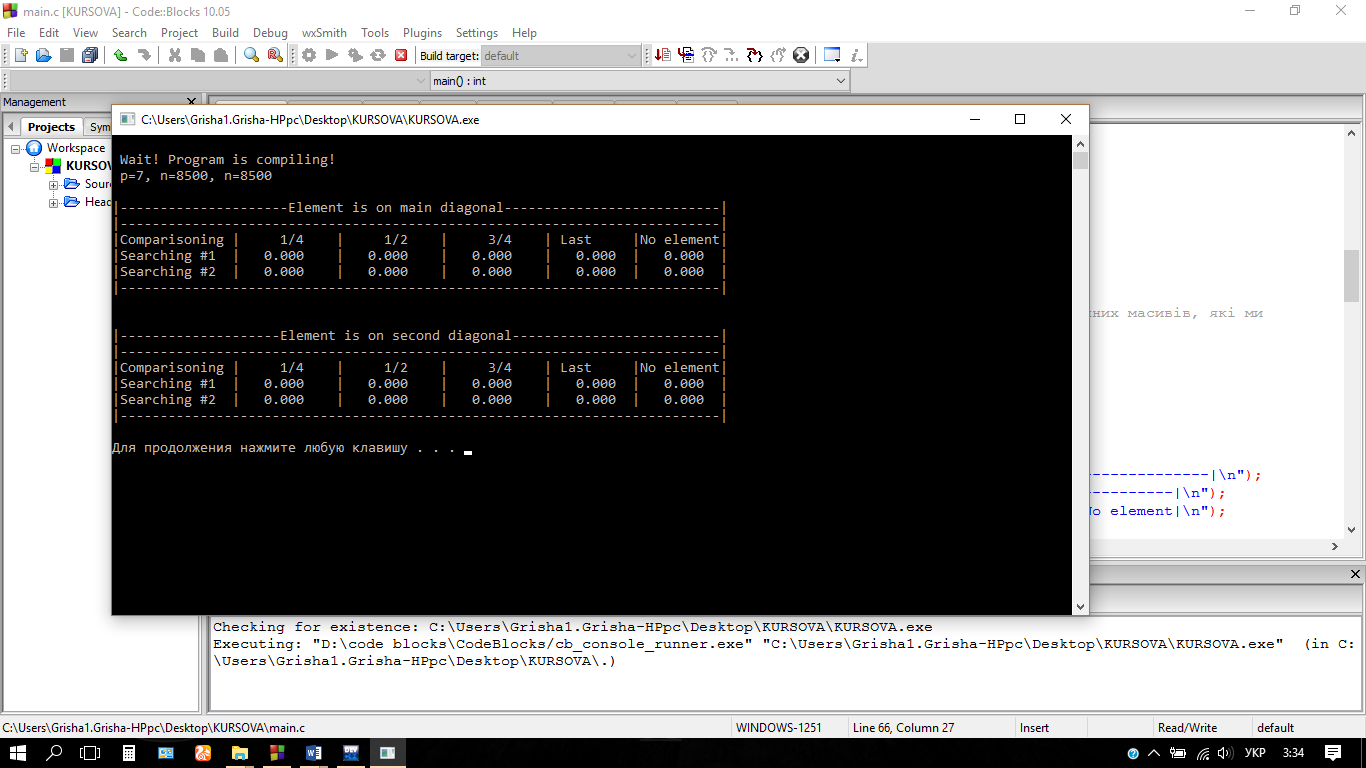
А) Таблиці часу сортування для трьох вимірного масиву.



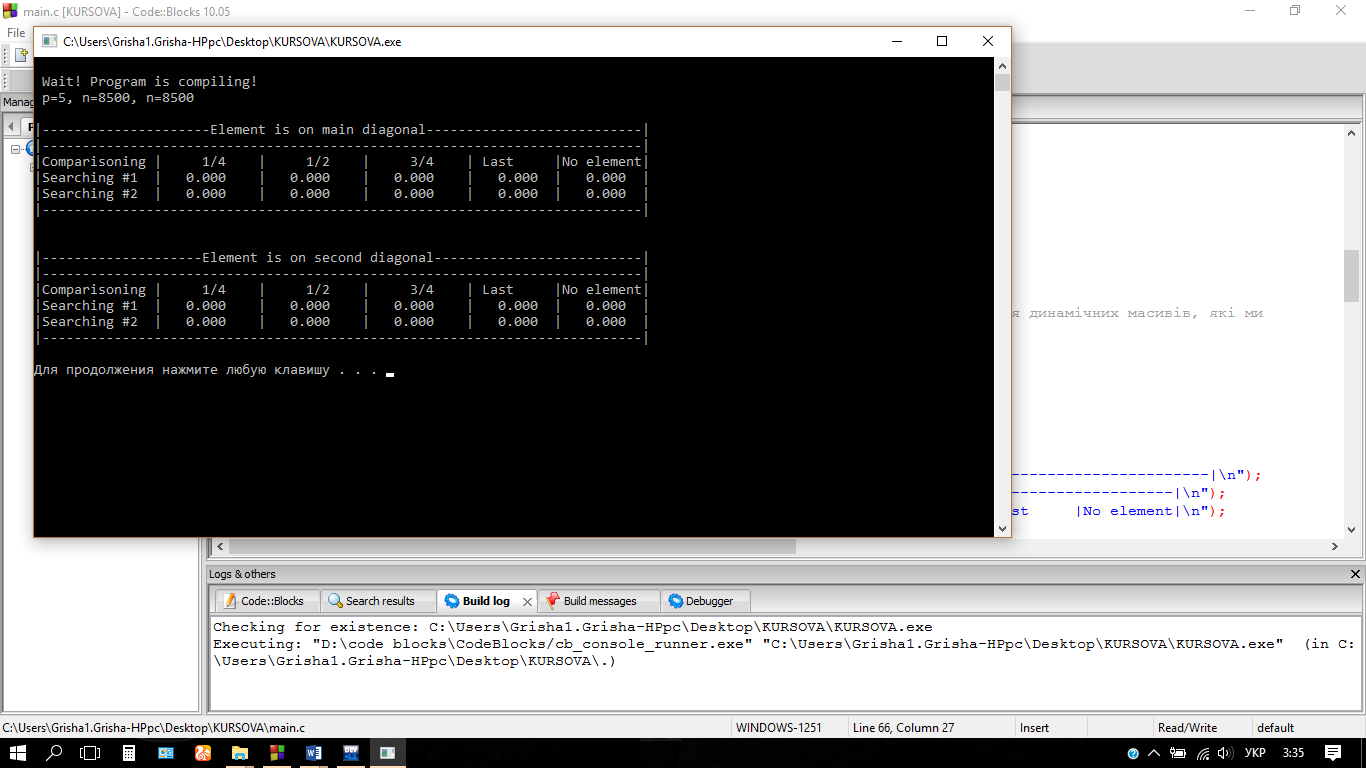
Б) Таблиці часу сортування для вектору.

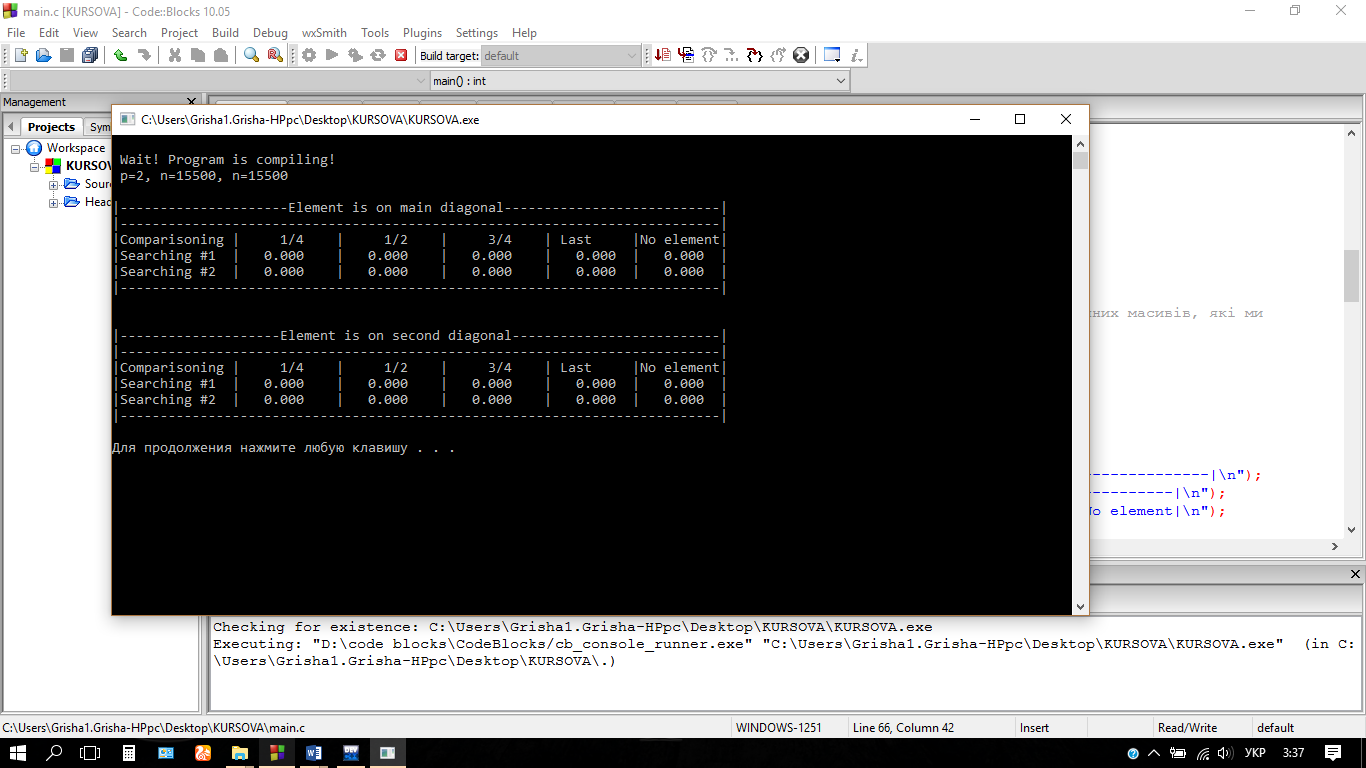
**Результати вимірювань**

Час роботи алгоритмів, від кількості заданих перерізів та від кількості заданих рядків



2)



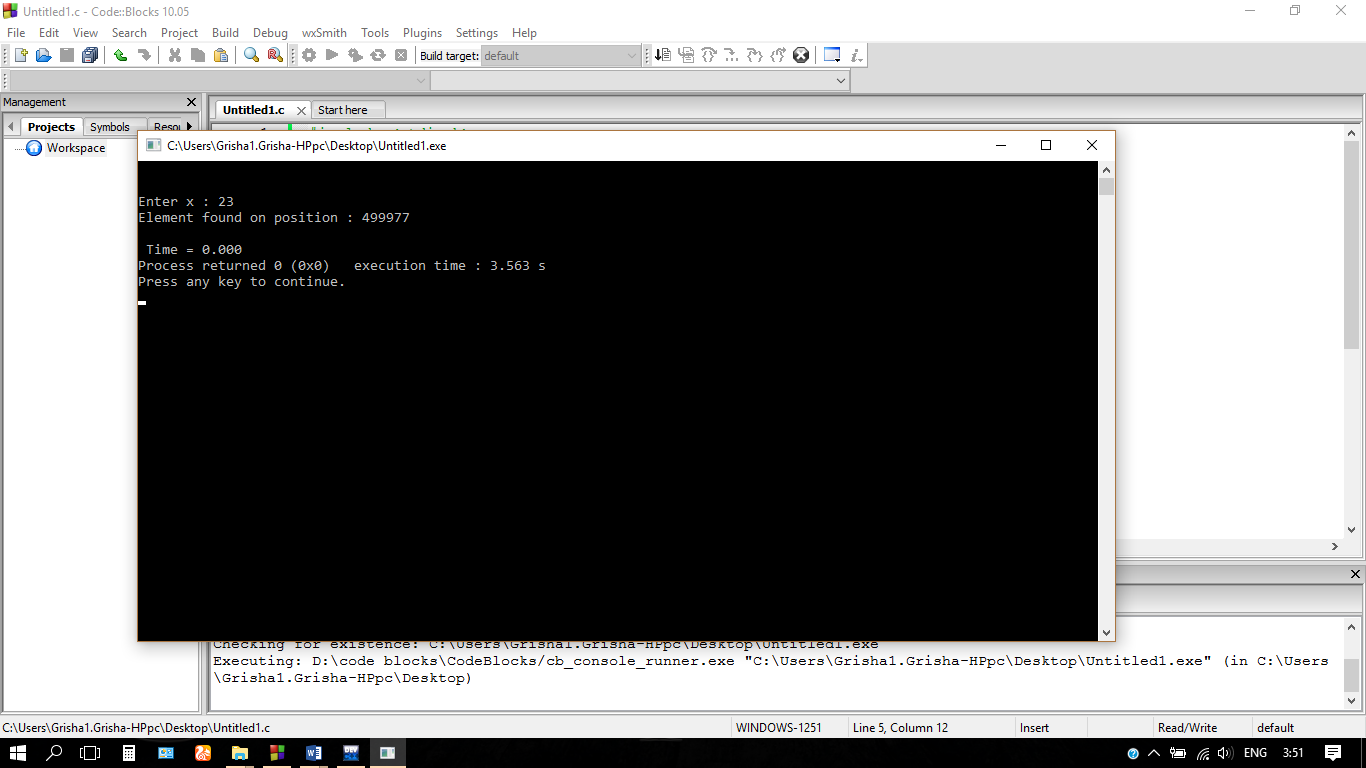
3)

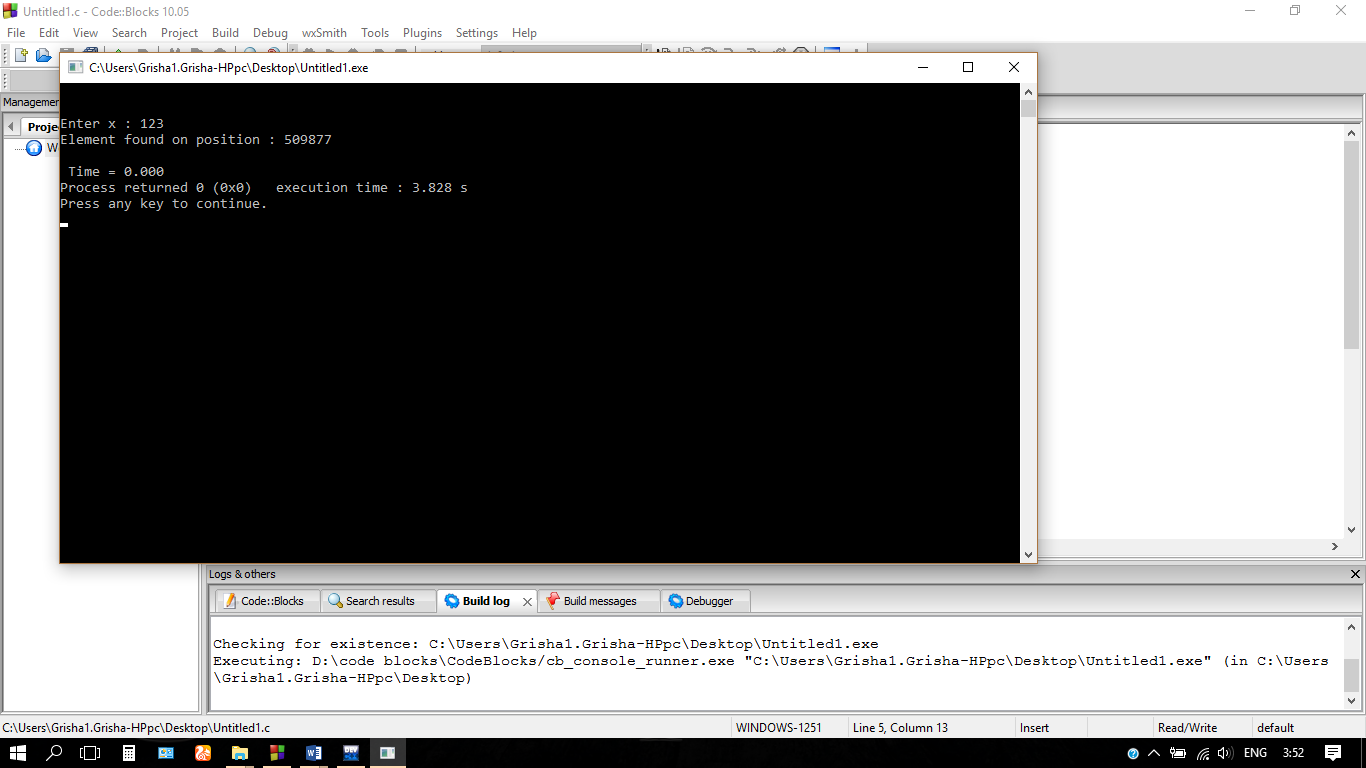
**Графіків залежностей часу роботи алгоритму від розмірів масиву не має, так як вийшли нульові результати.**

**Тестування для вектору**

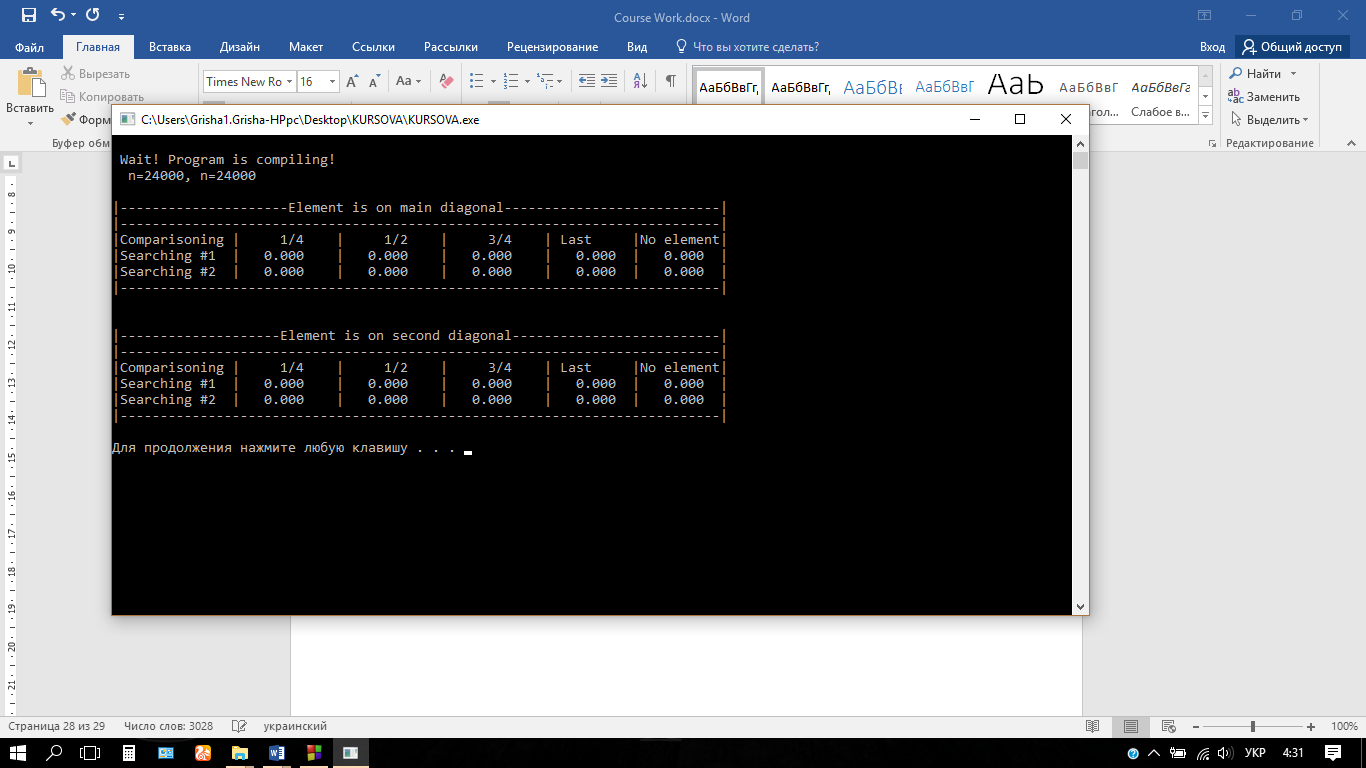
Оскільки алгоритми двійкового пошуку працюють для упорядкованих масивів і за своїм принципом роботи вони постійно відсікають половину масиву, то вони знаходять елемент дуже швидко. На даних прикладах вектор на 500000 елементів відсортувався за 0 секунд тому, що швидкість роботи дуже велика і час роботи алгоритмів прирівнюється до нуля.

( A[500000] )





**Тестування для матриці**



Для матриці результати також нульові, тому що у моєму завданні потрібно шукати елемент на діагоналі матриці. Кількість елементів діагоналі дорівнює кількості елементів в рядку або в стовпці. Оскільки максимальне число яке мені може видати мій комп’ютер (близько 24000\*24000 типу int) буде малим для діагоналі матриці, то двійковий пошук на діагоналях буде виконуватись також дуже швидко і час роботи алгоритмів буде прирівнюватись до 0.

**Аналіз вимірювань**

З отриманих мною результатів можна побачити, що обидва алгоритми двійкового пошуку працюють однаково швидко, хоча в теорії алгоритм двійкового пошуку №2 повинен працювати швидше ніж №1. Оскільки в алгоритмі №1 відбувається додаткова перевірка чергового елемента з шуканим, якої немає в другому алгоритмі. Вона уповільнює роботу алгоритму, тому що в циклі виконується постійно додаткова дія.

Нажаль, зміна розмірів трьохвимірного масиву не дала ніяких результатів і виміряти час який був би більшим за 0 не вдалося. Проте це доказує, що алгоритми двійкового пошуку для упорядкованих масивів працюють миттєво і їх ефективне практичне використання в алгоритмах.

При дослідженні двійкового пошуку на векторі та на матриці також не було ніякого результату, що підтверджує правильність виконання мною роботи.

Теоретично, зі збільшенням розмірів масиву час роботи алгоритму має змінюватись лінійно і він має зростати. Проте для того щоб добитись результату з точним виміром часу потрібно брати перерізи дуже великі за розміром, як мінімум 40\*40 – 50\*50 тис елементів. Для цього потрібно мати потужне обладнання. Тому, що переріз розміром 20000\*20000 елементів буде займати 1,6 Гб оперативної пам’яті. А даний масив до того ж іще тривимірний.

**Висновок**

На практиці я виявив , що алгоритми двійкового пошуку дуже швидкодіючі. Їх дуже доцільно використовувати коли масиви впорядковані за незбільшенням або незменшенням.

Алгоритми №1 та №2 є найефективнішими при пошуку елемента в векторі, в даному випадку їх швидкодія практично однакова. В матрицях же (теоретично, так як на практиці не виявлено) різниця має відчуватись, тому що в них виконується на багато більше дій ніж в векторах.

На результати роботи мають дуже значний вплив характеристики комп’ютера. На результати виконують вплив фонові процеси операційної системи та інші програми які можуть бути ввімкненими, потужність процессора, кількість оперативної пам’яті. Для того щоб покращити роботу алгоритмів і точніше дослідити час виконання роботи алгоритмів двійкового пошуку потрібно по максимуму відключити робочі програми і не заваджувати роботі алгоритмів ніякими діями.

**Джерела**

1. Конспект з курсу лекцій «Структури даних та алгоритми»
2. Марченко А.И «Программирование в среде Turbo Pascal 7.0»,8-е издание Киев «ВЕК+», Санкт-Петербург «КОРОНА-Век», 2004.
3. Методичні матеріали для виконання курсової роботи.