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: Селіверстов Нікіта

#### Група: КB-72

Номер залікової книжки: КВ-7214

#### Допущений до захисту

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

2 семестр 2017/2018

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 методів сортування (назви конкретних методів сортування)на багатовимірних масивах***

#### Виконавець роботи:

Селіверстов Нікіта Сергійович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018р.

***ТЕХНІЧНЕ ЗАВДАННЯ***

***на курсову роботу з дисципліни***

***“Структури даних і алгоритми”***

**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 94**

**Задача №2**

Впорядкувати окремо кожен переріз тривимірного масивy А[p,m,n] наскрізно по стовпчиках за незменшенням.

**Алгоритм №7**

Алгоритм сортування №3 методу прямого вибору.

**Способи обходу №: 1, 2, 3**

1. Переписати елементи заданого двовимірного масиву у додатковий одновимірний масив. Виконати сортування. Повернути результат у початковий масив.
2. Не використовуючи додаткового масиву, виконати сортування перетворюючи один індекс елементів „уявного” вектора у відповідні індекси елементів заданого двовимірного масиву
3. Виконати сортування, здійснюучи обхід безпосередньо по елементах заданого двовимірного масиву, не використвовуючи додаткових масивів і перетворень індексів

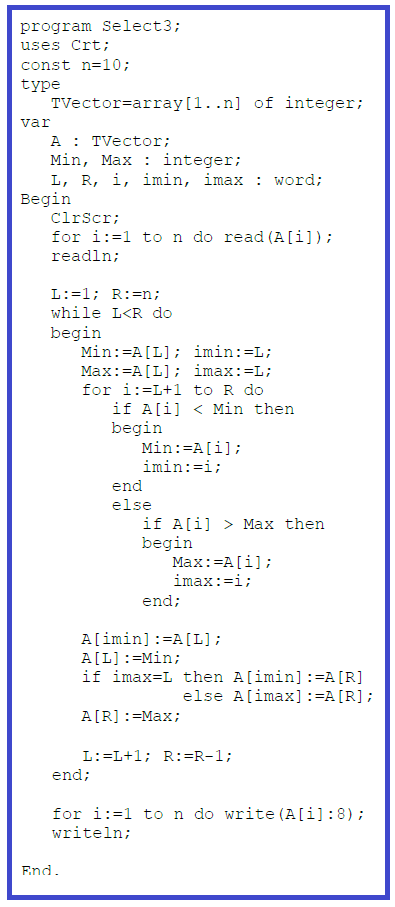
**Випадки дослідження**

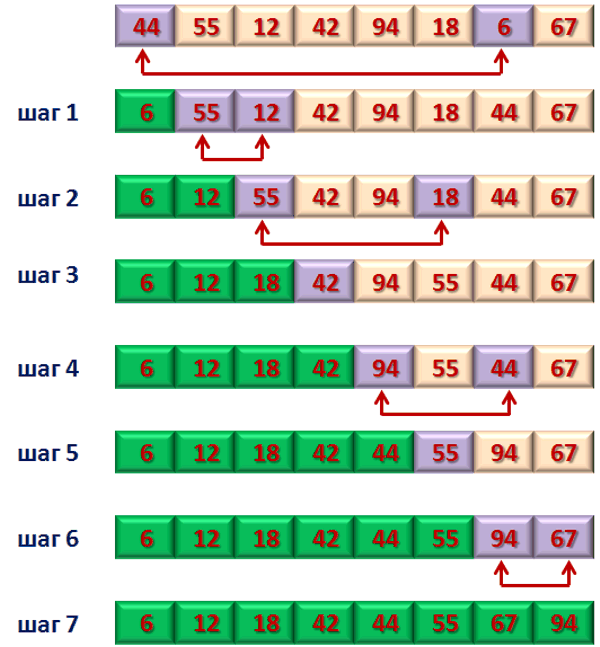
1. Елементи початкового масиву впорядковані відповідно до заданої ознаки.
2. Елементи початкового масиву невпорядковані.
3. Елементи початкового масиву впорядковані за протилежно заданою ознакою.

### **Метод прямого вибору**

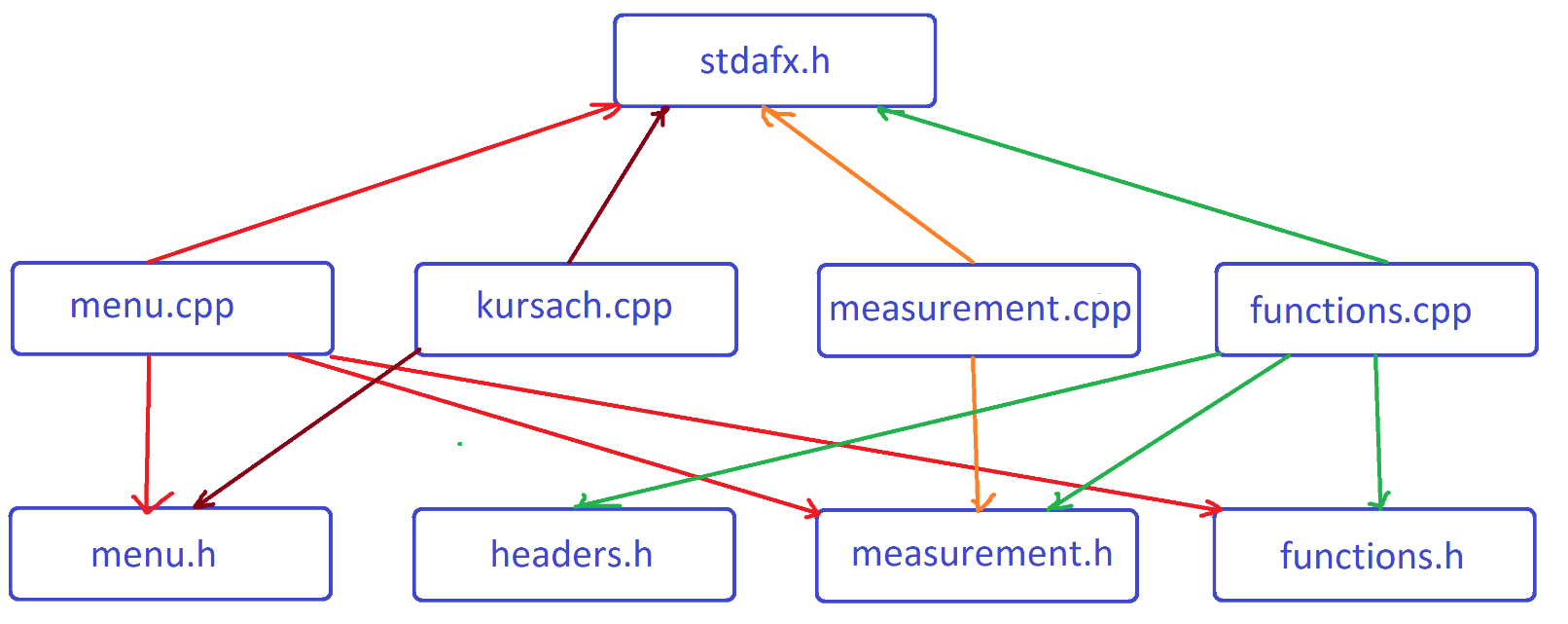
Один з його різновидів полягає в тому, що вибирається максимальний елемент масиву, а потім виконується його обмін з останнім елементом таблиці. Після цього останній елемент вважається впорядкованим і процес повторюється для підмасиву, що містить на один елемент менше за початковий, тобто елементи з 1-го до N-1. Процес повторюється кожен раз для масиву, зменшеного на один елемент. Закінчується він тоді, коли невпорядкований підмасив стає довжиною один елемент. Таким чином, загальна кількість повторень дорівнює N-1 (N - кількість елементів масиву).

**Програмна реалізація (на мові Pascal) запропонованого методу наведена нижче:**





**СХЕМА ІМПОРТУ/ЕКСПОРТУ МОДУЛІВ:**

****

**КОД ПРОГРАМИ:**

**Stdafx.h**

#pragma once

#include "targetver.h"

#include <stdio.h>

#include <tchar.h>

#include "conio.h"

#include <iostream>

#include <ctime>

#include <iomanip>

using namespace std;

**Menu.h**

#pragma once

void menu();

void custom\_test();

void general\_test();

void vector\_general\_test();

float execution(int dimension, int matrix\_type, int sort\_type);

**Headers.h**

#pragma once

const int p=5;

const int m=4;

const int n=10;

int A [p][m][n];

int Av [n]

**Measurement.h**

#pragma once

#ifndef \_\_Measurement\_H\_\_

#define \_\_Measurement\_H\_\_

#define measurements\_number 28

extern clock\_t Res[measurements\_number];

void MeasurementProcessing();

#endif

**Functions.h**

#pragma once;

void show\_matrix();

void show\_vector();

void fill\_vec\_1();

void fill\_vec\_2();

void fill\_vec\_3();

void fill\_1();

void fill\_2();

void fill\_3();

clock\_t sort\_vec();

clock\_t sort\_1();

clock\_t sort\_2(int \*A);

clock\_t sort\_3();

clock\_t time\_calc\_vec();

clock\_t time\_calc\_1();

clock\_t time\_calc\_2();

clock\_t time\_calc\_3();

**Menu.cpp**

#pragma once

#include "stdafx.h"

#include "menu.h"

#include "functions.h"

#include "Measurement.h"

void menu() //interface

{

int react;

system("cls");

cout<<"-MENU"<<endl<<endl<<endl;

cout<<"--Show Matrix (1)"<<endl<<endl;

cout<<"--Show Vector (2)"<<endl<<endl;

cout<<"--Start Custom Test (3)"<<endl<<endl;

cout<<"--Print Matrix Result Table (4)"<<endl<<endl;

cout<<"--Print Vector Result Table (5)"<<endl<<endl;

cout<<"-> ";

cin>>react;

switch (react)

{

case 1:

system("cls");

show\_matrix();

system("pause");

menu();

break;

case 2:

system("cls");

show\_vector();

system("pause");

menu();

break;

case 3:

system("cls");

custom\_test();

system("pause");

menu();

break;

case 4:

system("cls");

general\_test();

system("pause");

menu();

break;

case 5:

system("cls");

vector\_general\_test();

system("pause");

menu();

break;

}

system("cls");

cout<<"-ERROR";

getch();

menu();

}

void custom\_test() // vyborochniy test

{

int dimmension;

cout<<"VECTOR / MATRIX (1/3)"<<endl<<endl;

cin>>dimmension;

if (dimmension != 1 && dimmension != 3)

{

system("cls");

cout<<"-ERROR";

getch();

menu();

}

switch(dimmension)

{

case 1:

system("cls");

cout<<"-CUSTOM VECTOR TEST"<<endl<<endl<<endl;

int ask\_vec;

cout<<"--Vector type"<<endl<<endl;

cout<<"---Random (1)"<<endl<<endl;

cout<<"---Sorted (2)"<<endl<<endl;

cout<<"---Inverted (3)"<<endl<<endl;

cout<<"-> ";

cin>>ask\_vec;

if (ask\_vec != 1 && ask\_vec != 2 && ask\_vec != 3)

{

system("cls");

cout<<"-ERROR";

getch();

menu();

}

system("cls");

cout<<endl<<endl<<"----------------SORTED! Execution time: "<<execution(dimmension, ask\_vec, 0)<<" ----------------"<<endl<<endl;

show\_vector();

break;

case 3:

system("cls");

cout<<"-CUSTOM MATRIX TEST"<<endl<<endl<<endl;

int ask\_1, ask\_2;

cout<<"--Matrix type"<<endl<<endl;

cout<<"---Random (1)"<<endl<<endl;

cout<<"---Sorted (2)"<<endl<<endl;

cout<<"---Inverted (3)"<<endl<<endl;

cout<<"-> ";

cin>>ask\_1;

cout<<endl<<endl<<"--Sorting"<<endl<<endl;

cout<<"---Additional Vector (1)"<<endl<<endl;

cout<<"---Imagine Vector (2)"<<endl<<endl;

cout<<"---Without Additional Structures (3)"<<endl<<endl;

cout<<"-> ";

cin>>ask\_2;

if ((ask\_1 != 1 && ask\_1 != 2 && ask\_1 != 3) || (ask\_2 != 1 && ask\_2 != 2 && ask\_2 != 3))

{

system("cls");

cout<<"-ERROR";

getch();

menu();

}

system("cls");

cout<<endl<<endl<<"----------------SORTED! Execution time: "<<execution(dimmension, ask\_1, ask\_2)<<" ----------------"<<endl<<endl;

show\_matrix();

break;

}

}

void general\_test() // tablitsa rezultatatov matrix

{

const int c = 3;

float Res\_Table[c][c];

cout<<setw(43)<<"Random"<<setw(15)<<"Sorted"<<setw(15)<<"Inverted"<<endl<<endl<<endl;

for(int i=1; i<=c; i++)

{

switch(i)

{

case 1:

cout<<"Additional Vector :"<<setw(23);

break;

case 2:

cout<<"Imagine Vector :"<<setw(26);

break;

case 3:

cout<<"Without Additional Structures :"<<setw(11);

break;

}

for(int j=1; j<=c; j++)

{

cout<<execution(3, j, i)<<setw(15);

}

cout<<endl<<endl;

}

}

void vector\_general\_test()

{

cout<<setw(43)<<"Random"<<setw(15)<<"Sorted"<<setw(15)<<"Inverted"<<endl<<endl<<endl;

cout<<"Sorting for Vector"<<setw(23);

for(int i=1; i<=3; i++)

{

cout<<execution(1, i, 0)<<setw(15);

}

cout<<endl<<endl;

}

float execution(int dimension, int matrix\_type, int sort\_type) //logika podscheta vremeni

{

switch(dimension)

{

case 1: // vector

switch(matrix\_type)

{

case 1:

fill\_vec\_1();

//show\_vector();

return time\_calc\_vec();

break;

case 2:

fill\_vec\_2();

return time\_calc\_vec();

break;

case 3:

fill\_vec\_3();

//show\_vector();

return time\_calc\_vec();

break;

}

break;

case 3: // matrix

switch(matrix\_type)

{

case 1:

fill\_1();

//show\_matrix();

switch(sort\_type)

{

case 1:

return time\_calc\_1();

break;

case 2:

return time\_calc\_2();

break;

case 3:

return time\_calc\_3();

break;

}

break;

case 2:

fill\_2();

switch(sort\_type)

{

case 1:

return time\_calc\_1();

break;

case 2:

return time\_calc\_2();

break;

case 3:

return time\_calc\_3();

break;

}

break;

case 3:

fill\_3();

//show\_matrix();

switch(sort\_type)

{

case 1:

return time\_calc\_1();

break;

case 2:

return time\_calc\_2();

break;

case 3:

return time\_calc\_3();

break;

}

break;

break;

}

}

}

**Kursach.cpp**

#pragma once

#include "stdafx.h"

#include "menu.h"

int main()

{

cout<<"Kursova robota SDA"<<endl<<endl;

cout<<"KV-72"<<endl<<endl;

cout<<"Seliverstov N.S."<<endl<<endl<<endl<<endl;

system("pause");

menu();

return 0;

}

**Mesurement.cpp**

#pragma once

#include "stdafx.h"

#include "Measurement.h"

clock\_t Res[measurements\_number];

void MeasurementProcessing()

{

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 (int i = 3; i < measurements\_number; 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 (int i = 3; i < measurements\_number; 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 (int i = 3; i < measurements\_number; i++)

{

if (Res[i] > Max3) { Max3 = Res[i]; imax3 = i; }

else if (Res[i]<Min3 && Res[i]!=-1) { Min3 = Res[i]; imin3 = i; }

}

Res[0] = Sum - Min1 - Min2 - Min3 - Max1 - Max2 - Max3;

}

**Functions.cpp**

#pragma once

#include "stdafx.h"

#include "headers.h"

#include "functions.h"

#include "Measurement.h"

long int B[m\*n];

int tmp;

clock\_t start\_time;

clock\_t end\_time;

void show\_matrix() //Функция вывода матрицы

{

cout<<"Matrix ["<<p<<"]"<<" ["<<m<<"]"<<" ["<<n<<"]"<<endl<<endl;

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

{

cout<<"P = "<<i+1<<endl<<endl;

for(int j=0; j<m; j++)

{

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

{

cout<<setw(5)<<A[i][j][k];

}

cout<<endl;

}

cout<<endl<<endl;

}

}

void show\_vector()

{

cout<<"Vector ["<<n<<"]"<<endl<<endl;

for(int i=0; i<n; i++)

{

cout<<setw(5)<<Av[i];

}

cout<<endl<<endl;

}

void fill\_vec\_1()

{

for(int i=0; i<n; i++)

{

Av[i]=rand()%1000;

}

}

void fill\_vec\_2()

{

for(int i=0; i<n; i++)

{

Av[i]=-1000+i;

}

}

void fill\_vec\_3()

{

for(int i=0; i<n; i++)

{

Av[i]=1000-i;

}

}

void fill\_1() //функция наполнения random

{

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

{

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

{

for(int k=0; k<m; k++)

{

A[i][k][j] = rand()%100;

}

}

}

}

void fill\_2() //функция наполнения sorted

{

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

{

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

{

for(int k=0; k<m; k++)

{

A[i][k][j] = -100 + j \* m \* p + i \* m + k;

}

}

}

}

void fill\_3() //функция наполнения inverted

{

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

{

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

{

for(int k=0; k<m; k++)

{

A[i][k][j] = 100 - j \* m \* p - i \* m - k;

}

}

}

}

clock\_t time\_calc\_vec()

{

for (int i = 0; i < measurements\_number; i++)

{

Res[i]=sort\_vec();

}

MeasurementProcessing();

return Res[0] / 20;

}

clock\_t time\_calc\_1()

{

for (int i = 0; i < measurements\_number; i++)

{

Res[i]=sort\_1();

}

MeasurementProcessing();

return Res[0] / 20;

}

clock\_t time\_calc\_2()

{

for (int i = 0; i < measurements\_number; i++)

{

Res[i]=sort\_2(\*\*A);

}

MeasurementProcessing();

return Res[0] / 20;

}

clock\_t time\_calc\_3()

{

for (int i = 0; i < measurements\_number; i++)

{

Res[i]=sort\_3();

}

MeasurementProcessing();

return Res[0] / 20;

}

clock\_t sort\_vec()

{

start\_time = clock();

int L,R,imin,imax,min,max;

L=0; R=n;

while (L<R)

{

min=Av[L]; imin=L;

max=Av[L]; imax=L;

for(int i=L; i<=R; i++)

{

if (Av[i]<min)

{

min=Av[i];

imin=i;

}

else

{

if (Av[i]>max)

{

max=Av[i];

imax=i;

}

}

}

Av[imin]=Av[L];

Av[L]=min;

if (imax==L)

Av[imin]=Av[R];

else

Av[imax]=Av[R];

Av[R]=max;

L=L+1;

R=R-1;

}

end\_time = clock();

return end\_time - start\_time;

}

clock\_t sort\_1() //1 обход (дополнительный вектор)

{

start\_time = clock();

for (int k=0; k<p; k++)

{

tmp=1;

for(int l=0; l<n; l++)

{

for(int j=0; j<m; j++)

{

B[tmp]=A[k][j][l];

tmp++;

}

}

int L,R,imin,imax,min,max;

L=0; R=m\*n;

while (L<R)

{

min=B[L]; imin=L;

max=B[L]; imax=L;

for(int i=L; i<=R; i++)

{

if (B[i]<min)

{

min=B[i];

imin=i;

}

else

{

if (B[i]>max)

{

max=B[i];

imax=i;

}

}

}

B[imin]=B[L];

B[L]=min;

if (imax==L)

B[imin]=B[R];

else

B[imax]=B[R];

B[R]=max;

L=L+1;

R=R-1;

}

tmp=1;

for(int l=0; l<n; l++)

{

for(int j=0; j<m; j++)

{

A[k][j][l]=B[tmp];

tmp++;

}

}

}

end\_time = clock();

return end\_time - start\_time;

}

clock\_t sort\_2(int \*A) //2 обход ("воображаемый" вектор)

{

start\_time = clock();

for (int k=0; k<p; k++)

{

int L,R,imin,imax,min,max;

L=k\*n\*m; R=(k+1)\*n\*m;

while(L<R)

{

min = A[L]; imin=L;

max = A[L]; imax=L;

for (int i=L; i<R; i++)

{

if (A[i]<min)

{

min = A[i];

imin=i;

}

else

{

if (A[i]>max)

{

max = A[i];

imax=i;

}

}

}

A[imin] = A[L];

A[L] = min;

if (imax==L)

A[imin] = A[R];

else

A[imax] = A[R];

A[R] = max;

L=L+1;

R=R-1;

}

}

end\_time = clock();

return end\_time - start\_time;

}

clock\_t sort\_3() //3 обход (непосредственно по елементам матрицы)

{

start\_time = clock();

for(int k=0; k<p; k++)

{

int L,R,imin,imax,jmin,jmax,min,max;

L=0; R=m\*n;

while (L<R)

{

min=A[k][L%m][L/m]; imin=L%m, jmin=L/m;

max=A[k][L%m][L/m]; imax=L%m, jmax=L/m;

for(int i=L; i<R; i++)

{

if (A[k][i%m][i/m]<min)

{

min=A[k][i%m][i/m];

imin=i%m; jmin=i/m;

}

else

{

if (A[k][i%m][i/m]>max)

{

max=A[k][i%m][i/m];

imax=i%m;jmax=i/m;

}

}

}

A[k][imin][jmin]=A[k][L%m][L/m];

A[k][L%m][L/m]=min;

if (imax==L%m && jmax==L/m)

A[k][imin][jmin]=A[k][R%m][R/m];

else

A[k][imax][jmax]=A[k][R%m][R/m];

A[k][R%m][R/m]=max;

L=L+1;

R=R-1;

}

}

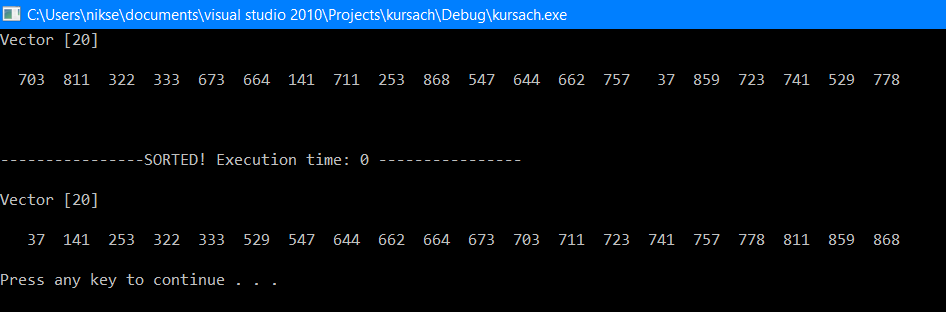
end\_time = clock();

return end\_time - start\_time;

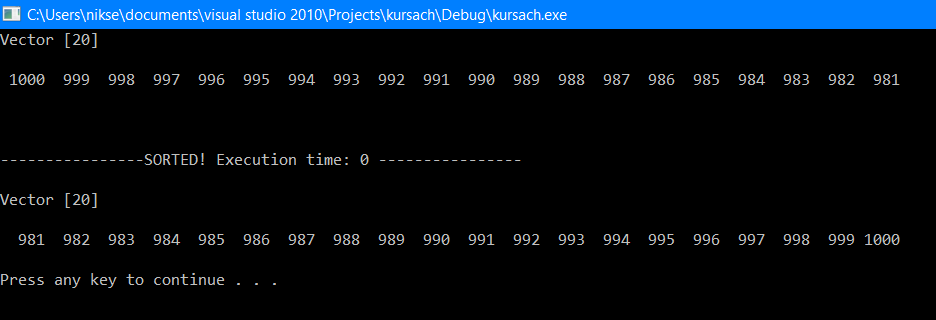
}

**ТЕСТУВАННЯ ПРОГРАММИ**

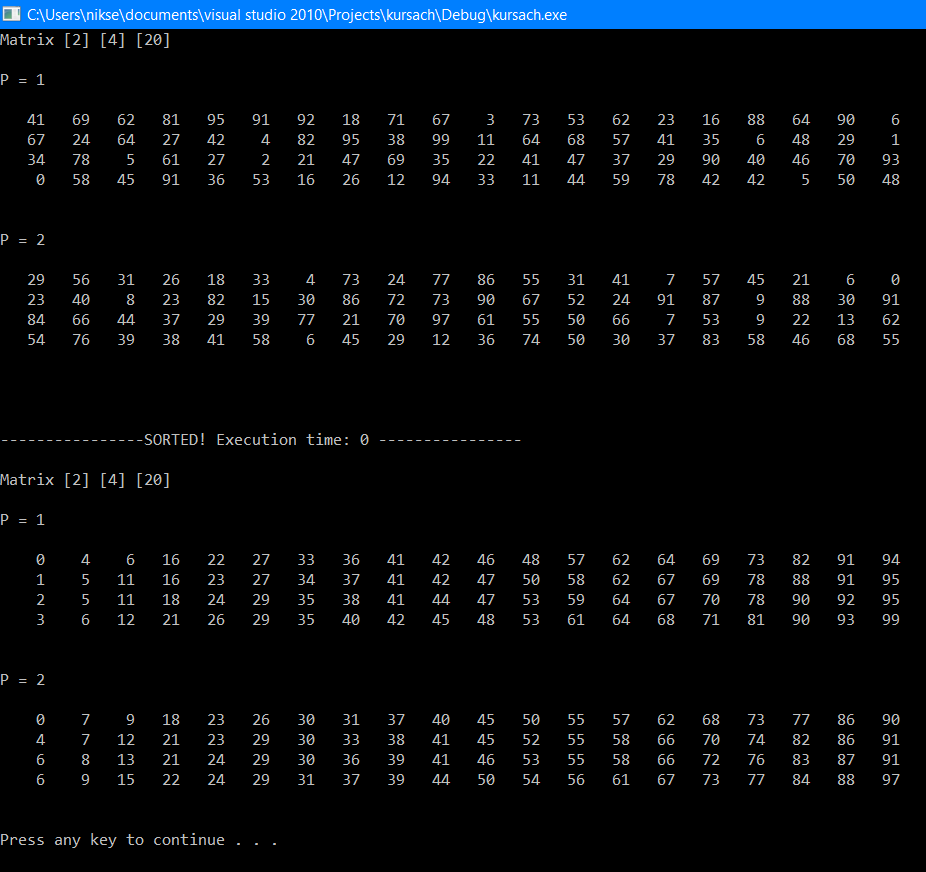
Сортування для вектора (RANDOM)



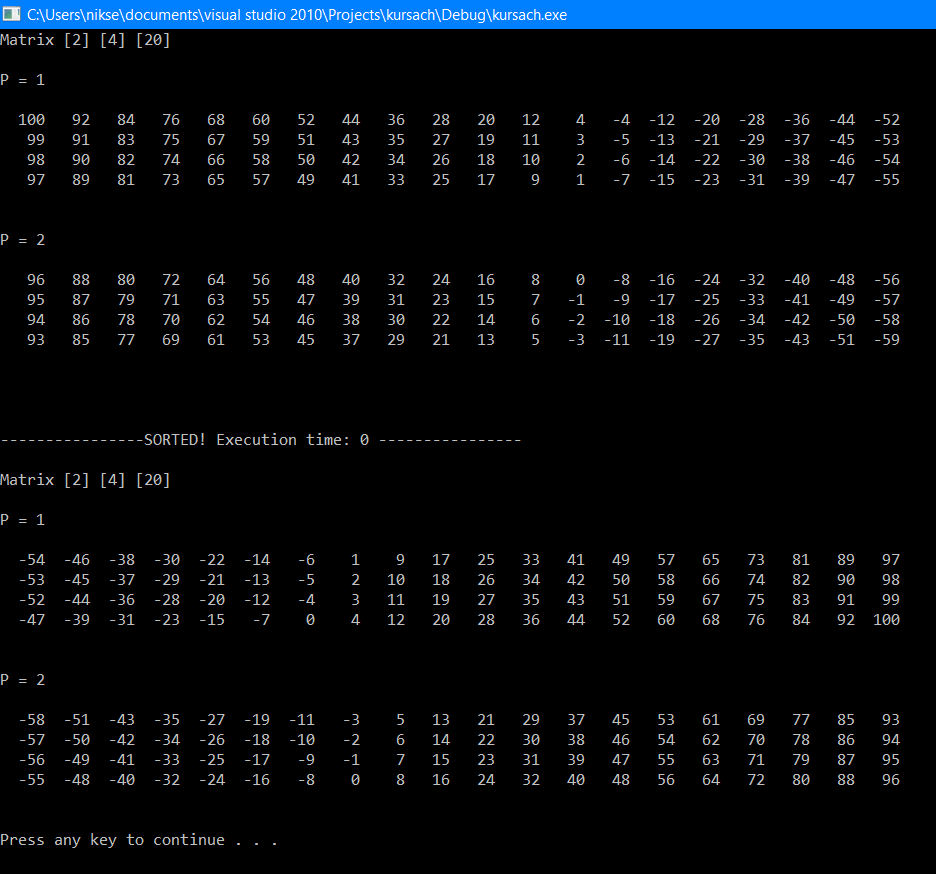
Сортування для вектора (INVERTED)



Сортування для тривимірного масиву (RANDOM)



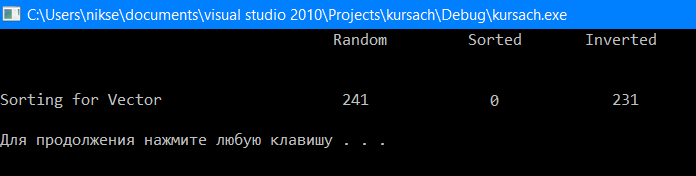
Сортування для тривимірного масиву (INVERTED)



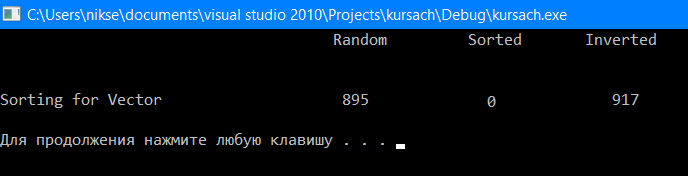
**РЕЗУЛЬТАТИ ЧАСУ СОРТУВАННЯ**

***Вектора***

**Av[20000]**

****

**Av[40000]**

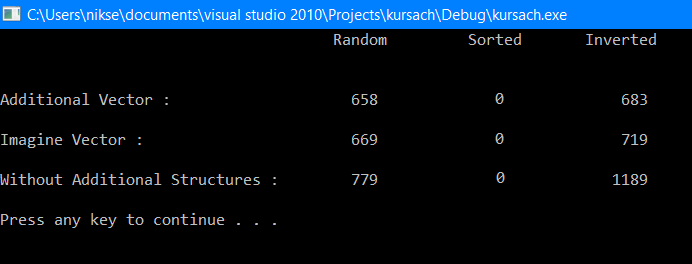
****

**РЕЗУЛЬТАТИ ЧАСУ СОРТУВАННЯ**

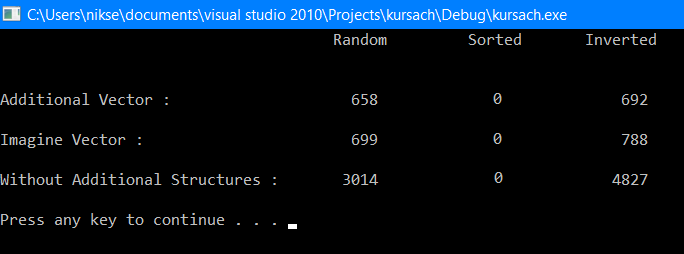
***тривимірного масиву***

Показана залежність часу роботи алгоритмів від форми перерізу масиву

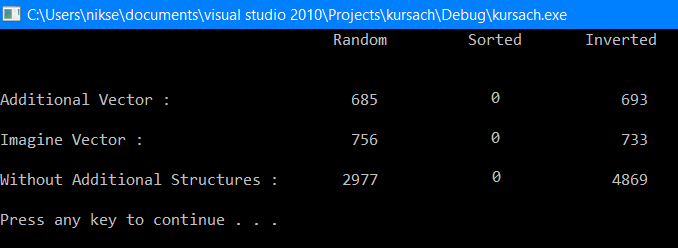
**A[3][4][5000]**



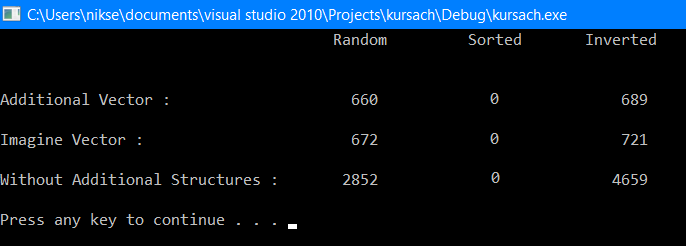
**A[3][5000][4]**



**A[3][40][500]**

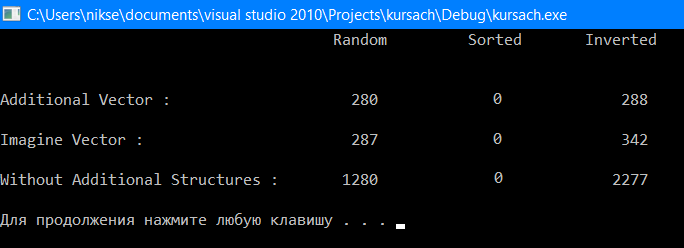


**A[3][500][40]**

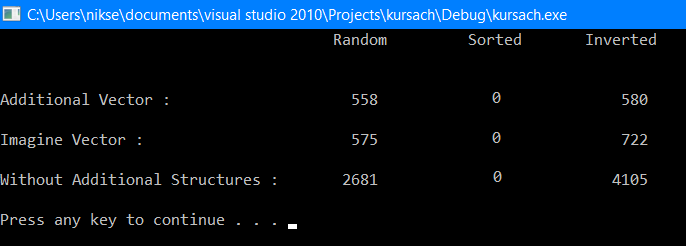


Залежність часу роботи алгоритмів від кількості перерізів масиву

**A[5][100][100]**



**A[10][100][100]**



**ВИСНОВОКИ:**

**Для вектора**

Алгоритм сортування №3 методу прямого вибору для різних типів заповнення вектора, окрім відсортованого працює майже з однією швидкістю. По швидкодії трохи переважає сортування інвертованого вектору. На другому місці йде випадковий. А відсортований сортується найшвидше. Це пов’язано з наявністю в алгоритмі способу перевірки на відсортованість.

**Висновок**

Для будь-якого виду вектора алгоритм сортування №3 методу прямого вибору працює приблизно з однією швидкістю, окрім відсортованого, є достатньо ефективним для відсортованого масиву.

**Для тривимірного масиву**

Було розглянуто алгоритм сортування №3 методу прямого вибору в трьох випадках:

1. Перенос масива у додатковий вектор, сорування вектора, повернення вектора у масив.
2. Розглядаючи масив як вектор без додаткового вектору.
3. Сортування безпосередньо елементів масиву без додаткового вектора або перетворення індексів.

Найкращим методом сортування виявився спосіб з використанням додаткового вектора, найгіршим - метод сортування безпосередньо елементів масиву без додаткового вектора. Пов’язано це з тим, що масив потрібно відсортувати по стовпцях.

1. Для першого способу сортування найшвидше сортується масив випадкових чисел, найгірше інвертований масив. Відсортований масив перевіряється, і програма не продовжує проходити по масиву, тому цей випадок найбільш швидкий. Для першого способу проходу не важлива форма перерізу.
2. Другий спосіб сортування за типом швидкодії майже не відрізняється від першого, але виконується все одно повільніше в обох випадках, бо доступ до елементів масиву виконується через посилання на уявний вектор. Відсортований масив перевіряється, і програма не продовжує проходити по масиву, тому цей випадок найбільш швидкий. Для другого способу проходу важлива форма перерізу.
3. Для третього способу сортування найшвидше сортується масив випадкових чисел, найгірше інвертований масив. Третій спосіб сортування працює найшвидше при p <m < n і є найчутливішим до зміни форми масива та кількості перерізів масива. Час сортування майже в 6 разів перевищуе початковий якщо збільшити дані матриці у 2 рази. Це повязано з тим що доступ до елементів прямий і чим складніша форма матриці тим більше часу потрібно для сортування.

Другий та третій способи працюють швидше при зменшенні кількості рядків.

**Висновок**

Якщо потрібно відсортувати тривимірний масив по

стовпчиках краще всього використовувати додатковий вектор.

**СПИСОК ЛІТЕРАТУРИ**

Марченко А.И., Марченко Л.А. «Программирование в среде Turbo Pascal 7.0»