

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: Лашков Денис

Група: КВ-33

Номер залікової книжки:

Допущений до захисту

2 семестр 2023/2024

ТЕХНІЧНЕ ЗАВДАННЯ
на курсову роботу з дисципліни
“Структури даних і алгоритми”

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ

I. Описати теоретичні положення, від яких відштовхується дослідження, тобто принцип та схему роботи кожного із досліджуваних алгоритмів сортування для одновимірного масива, навести загальновідомі властивості цих алгоритмів та оцінки кількості операцій порівняння та присвоєння для них.

II. Скласти алгоритми рішення задачі сортування в багато-вимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів, тобто виміри часу роботи цих алгоритмів для різних випадків та геометричних розмірів багатовимірних масивів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

Одна таблиця результатів (вимірів часу сортування впоряд-кованого, випадкового і обернено-впорядкованого масива) для масива з заданими геометричними розмірами повинна бути такою:

Таблиця № для масива $A[P,M,N]$, де $P=$; $M=$; $N=$;

	Впорядко- ваний	Невпорядко- ваний	Обернено впорядкований
Назва алгоритму 1			
Назва алгоритму 2			
Назва алгоритму 3			

- Для варіантів курсової роботи, де крім алгоритмів порівнюються також способи обходу, в назвах рядків таблиць потрібно вказати як назви алгоритмів, так і номери способів обходу.
- Для виконання ґрунтового аналізу алгоритмів потрібно зробити виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами.
- Зробити виміри часу для стандартного випадку одномірного масива, довжина якого вибирається такою, щоб можна було виконати коректний порівняльний аналіз з рішенням цієї ж задачі для багатовимірного масива.
- Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибираються так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів.
- Рекомендації випадків дослідження з різними геометричними розмірами масивів наведені у розділі «Випадки дослідження».

VI. Для наочності подання інформації за отриманими результатами рекомендується також будувати стовпчикові діаграми та графіки.

VII. Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами (вимірами часу):

- для одномірного масива відносно загальновідомої теорії;
- для багатовимірних масивів відносно результатів для одномірного масива;
- 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 поведуть себе саме так, а не інакше.

VIII. Зробити висновки за зробленим порівняльним аналізом.

IX. Програму курсової роботи під час її захисту ОБОВ'ЯЗКОВО мати при собі на електронному носії інформації.

Варіант № 151

151	1	6, 17, 11	3
-----	---	-----------	---

Задача

1. Впорядкувати окремо кожен переріз тривимірного масива $Arr3D [P,M,N]$ наскрізно по стовпчиках за незменшенням.

Досліджувані методи та алгоритми

6. Алгоритм сортування №2 методу прямого вибору
17. Гібридний алгоритм "вставка – обмін"
11. Алгоритм сортування №7 методу прямого вибору

Способи обходу

3. Виконати сортування, здійснюючи обхід безпосередньо по елементах заданого двовимірного масива, не використовуючи додаткових масивів і перетворень індексів.

Випадки дослідження для задачі №1

Випадок дослідження I. Залежність часу роботи алгоритмів від форми перерізу масива.

Рекомендовані розміри масива для досліджень:

$P = \text{const} = 3$, $M = \text{var}$, $N = \text{var}$, $M*N = \text{const} = 100000000$.

1) $M = 10$; $N = 10000000$;

1) $M = 100$; $N = 1000000$;

2) $M = 1000$; $N = 100000$;

3) $M = 10000$; $N = 10000$;

4) $M = 100000$; $N = 1000$;

5) $M = 1000000$; $N = 100$;

6) $M = 10000000$; $N = 10$;

Вектор довжиною $NV = M*N = 100000000$.

Для порівняння час сортування вектора довжиною $NV = M*N$ помножити на P .

Випадок дослідження II. Залежність часу роботи алгоритмів від кількості перерізів масива

Рекомендовані розміри масива для досліджень:

$P = \text{var}$, $M = \text{const} = 2000$, $N = \text{const} = 2000$. (кількість елементів у перерізі $M*N = 4000000$)

1) $P = 2$;

2) $P = 4$;

3) $P = 8$;

4) $P = 16$;

5) $P = 32$;

6) $P = 64$;

Час сортування вектора довжиною $NV = 4000000$ (дорівнює кількості елементів у перерізі) помножити на P . Вимір і порівняння для

вектора достатньо зробити тільки для найбільшого Р.

Випадок дослідження III. Залежність часу роботи алгоритмів від розміру перерізів масива

Порівняння з вектором для цього випадку не обов'язкове.

Для бажаних порівняти з вектором: тривимірний масив кожного розміру $P \times M \times N$ (де $M=N$) потрібно окремо порівнювати з вектором відповідної довжини $M \times N$, і час сортування такого вектора помножити на кількість перерізів Р.

Рекомендовані розміри масива для досліджень:

$P = \text{const} = 3$

- 1) $M = N = 4$ ($M \times N = 16$)
- 2) $M = N = 8$ ($M \times N = 64$)
- 3) $M = N = 16$ ($M \times N = 256$)
- 4) $M = N = 32$ ($M \times N = 1024$)
- 5) $M = N = 64$ ($M \times N = 4096$)
- 6) $M = N = 128$ ($M \times N = 16384$)
- 7) $M = N = 256$ ($M \times N = 65536$)
- 8) $M = N = 512$ ($M \times N = 262144$)
- 9) $M = N = 1024$ ($M \times N = 1048576$)
- 10) $M = N = 2048$ ($M \times N = 4194304$)
- 11) $M = N = 4096$ ($M \times N = 16777216$)
- 12) $M = N = 8192$ ($M \times N = 67108864$)

Теоретичні положення

1. Алгоритм сортування №2 методу прямого вибору

Загальний принцип:

Масив вважається в кожен момент часу поділеним на 2 частини: відсортовану та невідсортовану. Спочатку відсортована частина дорівнює нулю, а невідсортованою увесь масив.

Порядок дій:

(Сортується за не спаданням)

1) Проходимо масив зліва направо, порівнюючи елементи з невідсортованої частини з поточним, та шукаємо індекс мінімального елемента в масиві.

2) Міняємо місцями поточний і мінімальний.

3) Границя відсортованої частини зсувається на 1 вперед.

4) Повторюємо пункти 1, 2, 3 поки не відсортується весь масив.

Схема проходів по масиву:

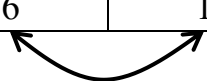
Початковий масив

0	1	2	3	4	5
6	1	2	5	4	4

Перший прохід

S=0 imin=1

0	1	2	3	4	5
6	1	4	5	2	4

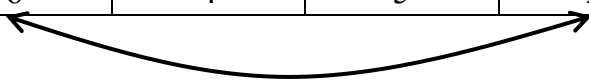


Другий прохід

S=1

imin=4


0	1	2	3	4	5
1	6	4	5	2	4



Третій прохід

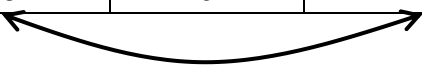
S=2=imin

0	1	2	3	4	5
1	2	4	5	6	4




Четвертий прохід

			S=3		imin=5
0	1	2	3	4	5
1	2	4	5	6	4



П'ятий прохід

				S=4	imin=5
0	1	2	3	4	5
1	2	4	4	6	5



Кінцевий результат

0	1	2	3	4	5
1	2	4	4	5	6

Код мовою C:

```
int imin, tmp;
for(int s=0; s<N-1; s++)
{
    imin=s;
    for(int i=s+1; i<N; i++)
        if (A[i]<A[imin]) imin=i;
    tmp=A[imin];
    A[imin]=A[s];
    A[s]=tmp;
}
```

2. Гібридний алгоритм "вставка – обмін"

Загальний принцип:

Масив вважається в кожен момент часу поділеним на 2 частини: відсортовану та невідсортовану. Спочатку відсортована частина дорівнює одному елементу, а невідсортована решта масиву.

Порядок дій:

(Сортується за не спаданням)

1) Проходимо масив від границі вліво, порівнюючи сусідні елементи, якщо елемент лівіше більший, то змінюємо місцями.

2) Пункт 1 повторюється доки не буде досягнуто початку масиву або буде перша не відповідність умові, тобто лівіший менший.

3) Границя зсувається на 1 вперед.

4) Повторюємо пункти 1, 2, 3 поки не відсортується весь масив.

Схема проходів по масиву:

Початковий масив

0	1	2	3	4	5
6	1	2	5	4	4

Перший прохід

$i=1$

0	1	2	3	4	5
6	1	4	5	2	4

Другий прохід

$i=2$

0	1	2	3	4	5
1	6	4	5	2	4

Третій прохід

$i=3$

0	1	2	3	4	5
1	4	6	5	2	4

Четвертий прохід

$i=4$

0	1	2	3	4	5
1	4	5	6	2	4

П'ятий прохід

$i=5$

0	1	2	3	4	5
1	2	4	5	6	4

Кінцевий результат

0	1	2	3	4	5
1	2	4	4	5	6

Код мовою C:

```
int j, tmp;
for(int i=1; i<N; i++)
{
    j=i;
    while (j>0 && A[j]<A[j-1])
    {
        tmp=A[j];
        A[j]=A[j-1];
        A[j-1]=tmp;
        j=j-1;
    }
}
```

3. Алгоритм сортування №7 методу прямого вибору

Загальний принцип:

Масив вважається в кожен момент часу поділеним на 2 частини: невідсортовану та відсортовану в свою чергу відсортована поділяється на ще дві частини праву і ліву. Спочатку відсортована частина дорівнює нулю, а невідсортована всьому масиву.

Порядок дій:

(Сортується за не спаданням)

- 1) Проходимо масив зліва направо, порівнюючи елементи з невідсортованої частини з елементами на позиціях лівої та правої границі, та шукаємо мінімальний та максимальний елементи в масиві та їх індекси.
- 2) Якщо мінімальний не дорівнює елементу на позиції лівої границі то змінюємо їх місцями, а якщо максимальний не дорівнює позиції елемента на правій границі елемента то міняємо їх місцями(заміни цих елементів не залежні одне від одного), якщо максимальний співпадає з елементом на позиції лівої границі то елемент з правої границі записується в на позицію мінімального елементу.
- 3)Ліва та права границі зміщуються на плюс один і мінус один відповідно.
- 4)Повторюємо пункти 1, 2, 3 поки не перетнуться границі.

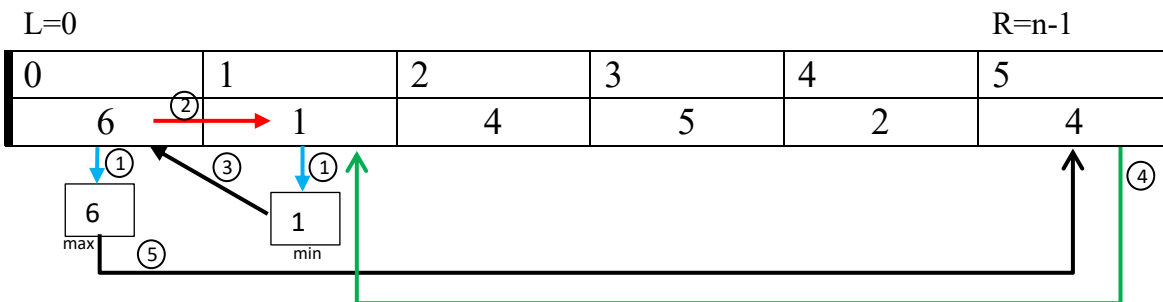
Схема проходу по масиву:

Порядок дій ①

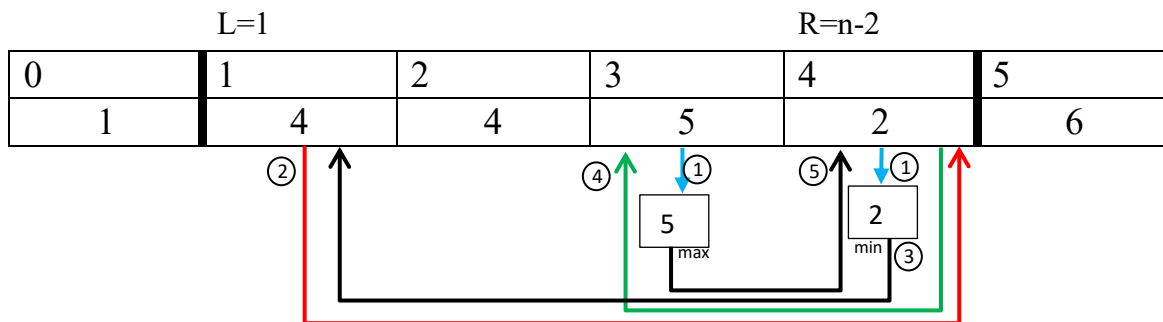
Початковий масив

0	1	2	3	4	5
6	1	2	5	4	4

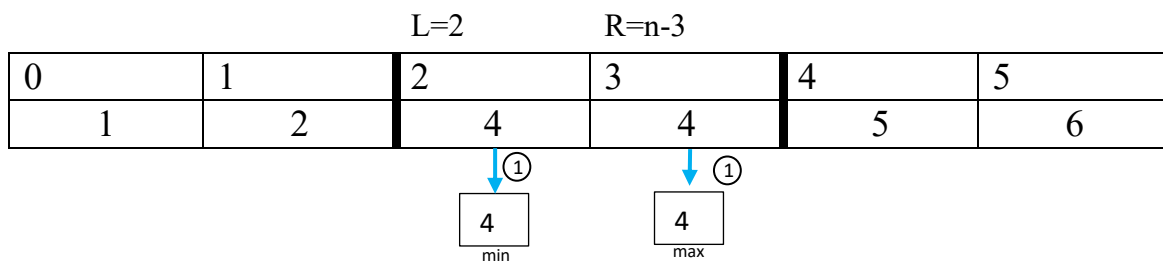
Перший прохід



Другий прохід



Другий прохід



Кінцевий результат

0	1	2	3	4	5
1	2	4	4	5	6

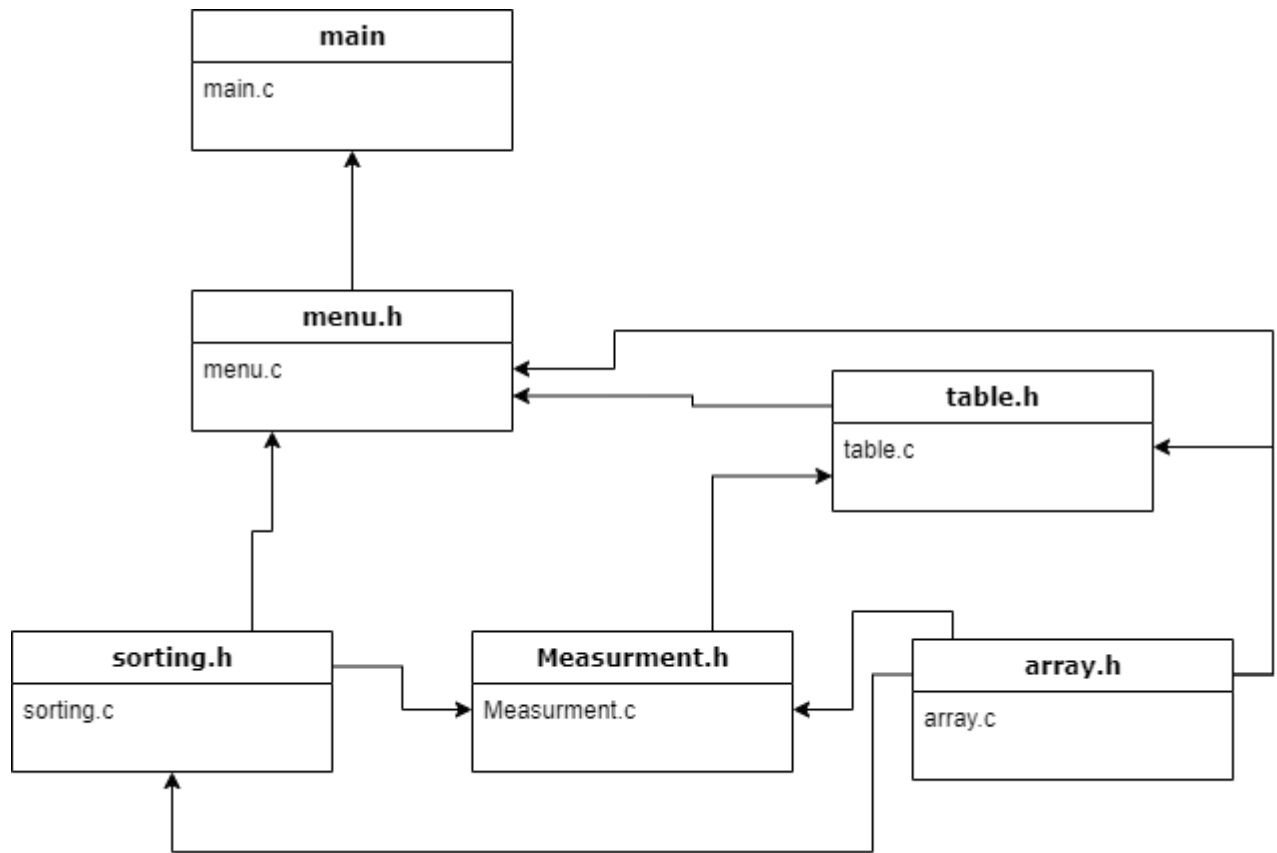
Код мовою C:

```

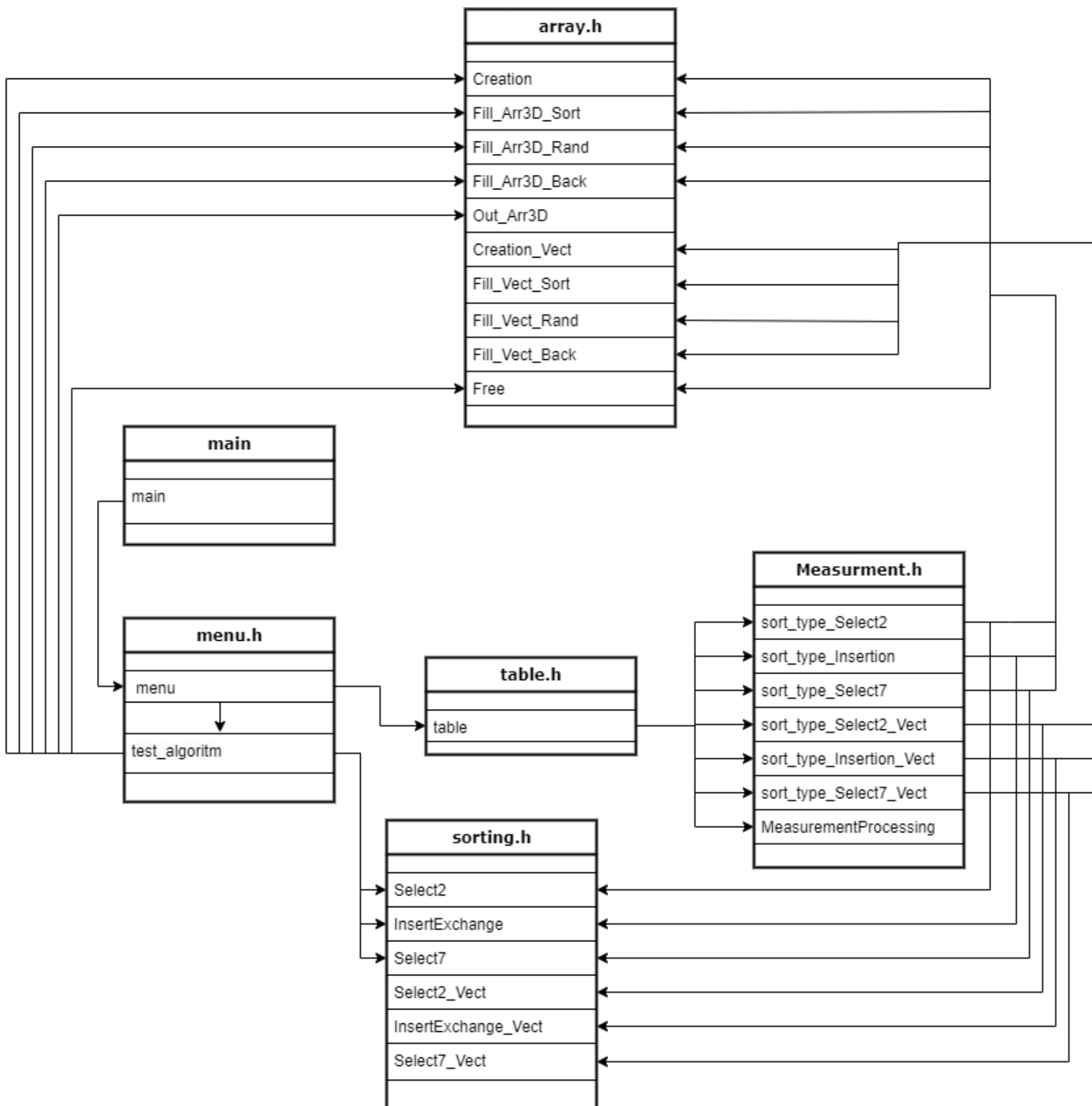
int Min, Max;
int L, R, imin, imax;
L=0;
R=N-1;
while (L<R)
{
    Min=A[L];
    imin=L;
    Max=A[L];
    imax=L;
    for(int i=L+1; i<R+1; i++)
    {
        if (A[i] < Min)
        {
            Min=A[i];
            imin=i;
        }
        else if (A[i] > Max)
        {
            Max=A[i];
            imax=i;
        }
    }
    if (imin!=L)
    {
        A[imin]=A[L];
        A[L]=Min;
    }
    if (imax!=R)
    {
        if (imax==L) A[imin]=A[R];
        else A[imax]=A[R];
        A[R]=Max;
    }
    L=L+1;
    R=R-1;
}

```

Схема імпорту/експорту модулів програми курсової роботи.



Структурна схема взаємо викликів процедур та функцій програми курсової роботи.



Опис призначення всіх функцій і процедур та їх параметрів.

1. Модуль menu

menu - функція головного меню для навігації по програмі

test_algoritm - функція для перевірки сортування, як параметр приймає тип сортування.

2. Модуль array

Creation - функція для створення динамічного тривимірного масиву, як параметри приймає розміри масиву.

Fill_Arr3D_Sort - функція для сортованого заповнення динамічного тривимірного масиву, як параметри приймає розміри масиву.

Fill_Arr3D_Rand - функція для випадкового заповнення динамічного тривимірного масиву, як параметри приймає розміри масиву.

Fill_Arr3D_Rand - функція для оберненого заповнення динамічного тривимірного масиву, як параметри приймає розміри масиву.

Out_Arr3D - функція для виведення динамічного тривимірного масиву, як параметри приймає розміри масиву.

Creation_Vect - функція для створення динамічного вектора, як параметр приймає розмір вектора.

Fill_Vect_Sort - функція для сортованого заповнення динамічного вектора, як параметр приймає розмір вектора.

Fill_Vect_Rand - функція для випадкового заповнення динамічного вектора, як параметр приймає розмір вектора.

Fill_Vect_Back - функція для оберненого заповнення динамічного вектора, як параметр приймає розмір вектора.

Free - функція для створення динамічного тривимірного масиву, як параметри приймає розміри масиву.

3. Модуль sorting

Select2 – функція типу clock_t, повертає час сортування тривимірного масиву методом вибору номер два. Як параметри приймає масив та його розміри.

InsertExchange – функція типу clock_t, повертає час сортування тривимірного масиву методом “обмін-вставка”. Як параметри приймає масив та його розміри.

Select7 – функція типу clock_t, повертає час сортування тривимірного масиву методом вибору номер сім. Як параметри приймає масив та його розміри.

Select2_Vect – функція типу clock_t, повертає час сортування вектора методом вибору номер два. Як параметри приймає вектор та його розмір.

InsertExchange_Vect – функція типу clock_t, повертає час сортування вектора методом “обмін-вставка”. Як параметри приймає вектор та його розмір.

Select7_Vect – функція типу clock_t, повертає час сортування вектора методом вибору номер сім. Як параметри приймає вектор та його розмір.

4. Модуль Measurement

sort_type_Select2 – функція для заповнення вектора, значеннями часу за який сортується динамічний тривимірний масив методом вибору номер два, як параметр приймає тип заповнення масиву.

sort_type_Insertion – функція для заповнення вектора, значеннями часу за який сортується динамічний тривимірний масив методом “вставка-обмін”, як параметр приймає тип заповнення масиву.

sort_type_Select7 – функція для заповнення вектора, значеннями часу за який сортується динамічний тривимірний масив методом вибору номер сім, як параметр приймає тип заповнення масиву.

sort_type_Select2_Vect – функція для заповнення вектора, значеннями часу за який сортується динамічний вектор методом вибору номер два, як параметр приймає тип заповнення динамічного вектора.

sort_type_Insertion_Vect – функція для заповнення вектора, значеннями часу за який сортується динамічний тривимірний масив методом “вставка-обмін”, як параметр приймає тип заповнення динамічного вектора.

sort_type_Select7_Vect – функція для заповнення вектора, значеннями часу за який сортується динамічний вектор методом вибору номер сім, як параметр приймає тип заповнення динамічного вектора.

MeasurementProcessing – функція для обрахунку середнього значення часу роботи різних методів сортування, шляхом обрахунку масиву значень часу відкидаючи: перші, найбільші та найменші значення.

5. Модуль table

table - функція яка виводить остаточний результат (в таблиці) вимірів середнього часу роботи алгоритмів сортування для динамічного тривимірного масиву і(або) динамічного вектора, як параметр приймає значення для виводу одної або одразу обидвох таблиць.

Текст програми

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "menu.h"
```

```
int main()
{
    srand(time(0));
    menu();
    return 0;
}
```

menu.h

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

void menu(); //навігація по програмі
int test_algorithm(int type); //тестування роботи сортування

#endif // MENU_H_INCLUDED
```

menu.c

```
#include <stdio.h>
#include <stdlib.h>
#include "array.h"
#include "menu.h"
#include "table.h"
#include "sorting.h"
```

```
void menu() //навігація по програмі
{
    int choice, type, flag=1, flag1=1;
    char str[1];
    while(flag==1)
```



```

{
    system("cls");
    printf("Press number to choice");
    printf("\n1.Table\n2.Test Select2\n3.Test InsertExchange\n4.Test
Select7\n");
    str[0]=getch();
    choice=atoi(str);
    switch(choice) //Вибір: тест сортування або таблиця результатів
    {
        case 1:
            system("cls");
            printf("Press number to choice\n");
            printf("1. 3D array\n2. Vector\n3. Both\n");
            str[0]=getch();
            type=atoi(str);
            system("cls");
            flag1=table(type);
            break;
        case 2:
            flag1=test_algoritm(1); // flag1 відповідає за коректне введення в меню
і ні на що не впливає
            break;
        case 3:
            flag1=test_algoritm(2);
            break;
        case 4:
            flag1=test_algoritm(3);
            break;
        default:
            printf("Incorrect data, try again");
            sleep(1);
            system ("cls");
            flag1=0;
            break;
    }
    if(flag1==1)
    {
        flag1=1;
        printf("\nPress number to choice\n");
        printf("1.Return to menu\n2.Exit");
        str[0]=getch();
        choice=atoi(str);
        printf("%d", choice);
        if(choice!=1)
        {
            fflush(stdin);
            flag=0;
            system("cls");
        }
    }
}

int test_algoritm(int type) //тестування роботи сортування
{
    system("cls");
    int choice;
    char str[1];
    printf("Press number to choice");
    printf("\n1.Sorted Array\n2.Random Sorted Array\n3.Reverse Sorted Array\n");
    str[0]=getch();
    choice=atoi(str);

```

```

system("cls");
Creation(P_T, NM_Test, NM_Test);
switch(choice)//Вибір методу заповнення масиву
{
case 1:
    Fill_Arr3D_Sort(P_T,NM_Test,NM_Test);
    break;
case 2:
    Fill_Arr3D_Rand(P_T,NM_Test,NM_Test);
    break;
case 3:
    Fill_Arr3D_Back(P_T,NM_Test,NM_Test);
    break;
default:
    printf("Incorrect data, try again");
    sleep(1);
    return 0;
    break;
}
printf("\tP=%d, M=%d, N=%d\n\n",P_T,NM_Test,NM_Test);
printf("\tNot sorted\n");
Out_Arr3D(P_T,NM_Test,NM_Test);
switch(type)//Вибір методу сортування масиву
{
case 1:
    Select2(Arr3D, P_T,NM_Test,NM_Test);
    break;
case 2:
    InsertExchange(Arr3D, P_T,NM_Test,NM_Test);
    break;
case 3:
    Select7(Arr3D, P_T,NM_Test,NM_Test);
    break;
default:
    printf("Incorrect data, try again");
    sleep(1);
    return 0;
    break;
}
printf("\n\t Sorted\n");
Out_Arr3D(P_T, NM_Test, NM_Test);
Free(P_T, NM_Test);
return 1;
}

```

array.h

```

#ifndef ARRAY_H_INCLUDED
#define ARRAY_H_INCLUDED
#define P 3 // Кількість перерізів
#define M 10 // Кількість рядків
#define N 10// Кількість стовпчиків
#define P_T 3// Кількість перерізів для тесту сортування
#define NM_Test 6 //Кількість рядків і стовпчиків для тесту сортування
extern int ***Arr3D;
extern int *Vector;

void Creation(int p, int m, int n);//Створення динамічного багатовимірного масива
void Fill_Arr3D_Sort(int p, int m, int n);// Заповнення багатовимірного масиву для випадку "відсортовано"
void Fill_Arr3D_Rand(int p, int m, int n);// Заповнення багатовимірного масиву для випадку "не впорядкований"

```

```

void Fill_Arr3D_Back (int p, int m, int n); // Заповнення багатовимірного масиву
для випадку "обернено-відсортовано"
void Out_Arr3D(int p, int m, int n); // Виведення багатовимірного масива
void Creation_Vect(int n); // Створення динамічного вектора
void Fill_Vect_Sort(int n); // Заповнення вектора для випадку "відсортовано"
void Fill_Vect_Rand(int n); // Заповнення вектора для випадку "не впорядкований"
void Fill_Vect_Back(int n); // Заповнення вектора для випадку "обернено-
відсортовано"
void Free(int p, int m); // Очищення багатовимірного динамічного масива

#endif // ARRAY_H_INCLUDED

```

array.c

```

#include <stdio.h>
#include <stdlib.h>
#include "array.h"

int ***Arr3D;
int *Vector;

void Creation(int p, int m, int n) // Створення динамічного багатовимірного масива
{
    Arr3D = (int***) malloc(p*sizeof(int**));
    for (int k=0; k<p; k++)
    {
        Arr3D[k] = (int**) malloc(m*sizeof(int*));
        for (int i=0; i<m; i++)
            Arr3D[k][i] = (int*) malloc(n*sizeof(int));
    }
}

void Fill_Arr3D_Sort(int p, int m, int n) // Заповнення багатовимірного масиву для
випадку "відсортовано"
{
    int number=0;
    for (int k=0; k<p; k++)
        for (int j=0; j<n; j++)
            for (int i=0; i<m; i++)
                Arr3D[k][i][j] = number++;
}

void Fill_Arr3D_Rand(int p, int m, int n) // Заповнення багатовимірного масиву для
випадку "не впорядкований"
{
    for (int k=0; k<p; k++)
        for (int j=0; j<n; j++)
            for (int i=0; i<m; i++)
                Arr3D[k][i][j] = rand() % (p*m*n);
}

void Fill_Arr3D_Back(int p, int m, int n) // Заповнення багатовимірного масиву для
випадку "обернено-відсортовано"
{
    int number = p*m*n;
    for (int k=0; k<p; k++)
        for (int j=0; j<n; j++)
            for (int i=0; i<m; i++)
                Arr3D[k][i][j]=number--;
}

void Out_Arr3D(int p, int m, int n) // Виведення багатовимірного масива

```

```

{
    for (int k=0; k<p; k++)
    {
        for (int i=0; i<m; i++)
        {
            for (int j=0; j<n; j++)
            {
                printf("%4d", Arr3D[k][i][j]);
            }
            printf("\n");
        }
        printf("\n");
    }
}

void Creation_Vect(int n) //Створення динамічного вектора
{
    Vector=(int*) malloc(n*sizeof(int));
}

void Fill_Vect_Sort(int n) // Заповнення вектора для випадку "відсортовано"
{
    int number = 0;
    for (int i=0; i<n; i++)
        Vector[i] = number++;
}

void Fill_Vect_Rand(int n) // Заповнення вектора для випадку "не впорядкований"
{
    for (int i=0; i<n; i++)
        Vector[i] = rand() % n;
}

void Fill_Vect_Back(int n) // Заповнення вектора для випадку "обернено-відсортовано"
{
    int number=n;
    for (int i=0; i<n; i++)
        Vector[i]=number--;
}

void Free(int p, int m) //Очищення багатовимірною динамічного масива
{
    for (int k=0; k<p; k++)
    {
        for (int i=0; i<m; i++)
            free(Arr3D[k][i]);
        free(Arr3D[k]);
    }
    free(Arr3D);
}

```

sorting.h

```

#ifndef SORTING_H_INCLUDED
#define SORTING_H_INCLUDED
#include "Measurement.h"
clock_t Select2(int ***A, int p, int m, int n); //Алгоритм сортування №2 методу
прямого вибору.
clock_t InsertExchange(int ***A, int p, int m, int n); //Гібридний алгоритм
"вставка - обмін".
clock_t Select7(int ***A, int p, int m, int n); //Алгоритм сортування №7 методу
прямого вибору.

```

```

////////////////////////////////Сортування Векторів////////////////////////////////
clock_t Select2_Vect(int *A, int n) ;//Алгоритм сортування №2 методу прямого вибору
для вектора.
clock_t InsertExchange_Vect(int *A, int n); //Гібридний алгоритм "вставка - обмін"
для вектора.
clock_t Select7_Vect(int *A, int n); //Алгоритм сортування №7 методу прямого вибору
для вектора.

```

```

#endif // SORTING_H_INCLUDED

```

sorting.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "array.h"
#include "sorting.h"

```

```

clock_t Select2(int ***A, int p, int m, int n) //Алгоритм сортування №2 методу
прямого вибору.

```

```

{
    int imin, jmin, tmp, si, sj, flag;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int k=0; k<p; k++)
    {
        si=0;
        sj=0;
        while(sj<=n-1 || si<m-2)
        {
            imin=si;
            jmin=sj;
            flag=1;
            for(int j=sj; j<n; j++)
            {
                int i=si;
                if(flag==1)
                    while(i<m)
                    {
                        if (A[k][i][j]<A[k][imin][jmin])
                        {
                            imin=i;
                            jmin=j;
                        }
                        i++;
                    }

                if(sj<=n-1&&flag==0)
                    for(int i=0; i<m; i++)
                        if (A[k][i][j]<A[k][imin][jmin])
                        {
                            imin=i;
                            jmin=j;
                        }
            }
            flag=0;
            tmp=A[k][imin][jmin];
            A[k][imin][jmin]=A[k][si][sj];
            A[k][si][sj]=tmp;
            si++;
            if(si==m)

```

```

        {
            sj++;
            si=0;
        }
    }
}
time_stop = clock();
return (time_stop - time_start);
}

clock_t InsertExchange(int ***A, int p, int m, int n) //Гібридний алгоритм "вставка
- обмін".
{
    int j, tmp, i, q, r, flag;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int k=0; k<p; k++)
    {
        i=1;

        for(int j=0; j<n; j++)
        {
            if(j>0)
                i=0;
            while(i<m)
            {
                q=j;
                r=i;
                i++;
                flag=1;
                tmp=A[k][r][q];
                if(r==0)
                {
                    q--;
                    flag=0;
                    r=m;
                }
                while (q>=0 && r>0 && tmp<A[k][r-1][q])
                {
                    if(flag==1)
                    {
                        A[k][r][q] = A[k][r-1][q];
                        A[k][r-1][q]=tmp;
                        r--;
                    }
                    else
                    {
                        A[k][0][q+1]= A[k][r-1][q] ;
                        A[k][r-1][q]=tmp;
                        flag=1;
                        r--;
                    }
                }
                if(r==0)
                {
                    tmp=A[k][0][q];
                    q--;
                    r=m;
                    flag=0;
                }
            }
        }
    }
}

```

```

    time_stop = clock();
    return time_stop - time_start;
}

clock_t Select7(int ***A, int p, int m, int n) //Алгоритм сортування №7 методу
прямого вибору.
{
    int Min, Max;
    int i, L_i, L_j, R_i, R_j, imin, imax, jmin, jmax, m1;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int k=0; k<p; k++)
    {
        L_j=0;
        R_j=n-1;
        L_i=0;
        R_i=m-1;
        while (L_j<=R_j || (L_i<R_i&&L_j<R_j))
        {

            Min=A[k][L_i][L_j];
            imin=L_i;
            jmin=L_j;
            Max=A[k][L_i][L_j];
            imax=L_i;
            jmax=L_j;
            m1=m;
            for(int j=L_j; j<R_j+1; j++)
            {
                i=L_i;
                if(L_j!=j)
                    i=0;
                while(i<m1)
                {
                    if (A[k][i][j] < Min)
                    {
                        Min=A[k][i][j];
                        imin=i;
                        jmin=j;
                    }
                    else if (A[k][i][j] > Max)
                    {
                        Max=A[k][i][j];
                        imax=i;
                        jmax=j;
                    }
                    i++;
                    if(j==R_j)
                        m1=R_i+1;
                }
            }
            if (imin!=L_i || jmin!=L_j)
            {
                A[k][imin][jmin]=A[k][L_i][L_j];
                A[k][L_i][L_j]=Min;
            }
            if (imax!=R_i || jmax!=R_j)
            {
                if (imax==L_i&&jmax==L_j) A[k][imin][jmin]=A[k][R_i][R_j];
                else A[k][imax][jmax]=A[k][R_i][R_j];
                A[k][R_i][R_j]=Max;
            }
            L_i++;
        }
    }
}

```

```

        R_i--;
        if (L_i==m)
        {
            L_i=0;
            R_i=m-1;
            L_j++;
            R_j--;
        }
        if (Max<=Min)
            break;
    }
}
time_stop = clock();
return time_stop - time_start;
}

////////////////////////////////Сортування Векторів////////////////////////////////

clock_t Select2_Vect(int *A, int n) //Алгоритм сортування №2 методу прямого вибору
для вектора.
{
    int imin, tmp;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int s=0; s<n-1; s++)
    {
        imin=s;
        for(int i=s+1; i<n; i++)
            if (A[i]<A[imin]) imin=i;
        tmp=A[imin];
        A[imin]=A[s];
        A[s]=tmp;
    }
    time_stop = clock();
    return time_stop - time_start;
}
clock_t InsertExchange_Vect(int *A, int n) //Гібридний алгоритм "вставка - обмін".
{
    int j, tmp;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int i=1; i<n; i++)
    {
        j=i;
        while (j>0 && A[j]<A[j-1])
        {
            tmp=A[j];
            A[j]=A[j-1];
            A[j-1]=tmp;
            j=j-1;
        }
    }
    time_stop = clock();
    return time_stop - time_start;
}

clock_t Select7_Vect(int *A, int n) //Алгоритм сортування №7 методу прямого вибору.
{
    int Min, Max;
    int L, R, imin, imax;
    clock_t time_start, time_stop;

```



```

time_start = clock();
L=0;
R=n-1;
while (L<R)
{
    Min=A[L];
    imin=L;
    Max=A[L];
    imax=L;
    for(int i=L+1; i<R+1; i++)
    {
        if (A[i] < Min)
        {
            Min=A[i];
            imin=i;
        }
        else if (A[i] > Max)
        {
            Max=A[i];
            imax=i;
        }
    }
    if (imin!=L)
    {
        A[imin]=A[L];
        A[L]=Min;
    }
    if (imax!=R)
    {
        if (imax==L) A[imin]=A[R];
        else A[imax]=A[R];
        A[R]=Max;
    }
    L=L+1;
    R=R-1;
}
time_stop = clock();
return time_stop - time_start;
}

```

Measurement.h

```

#ifndef MEASUREMENT_H_INCLUDED
#define MEASUREMENT_H_INCLUDED

#include <time.h>

// Загальна кількість вимірів часу роботи алгоритма
#define measurements_number 28
// Кількість відкинутих початкових вимірів
#define rejected_number 2
#define min_max_number 3// Кількість відкинутих вимірів з мінімальними значеннями
та з максимальними значеннями

extern clock_t Res[measurements_number]; // Масив значень часу роботи алгоритма

void sort_type_Select2(int type); // заповнення масиву значень часу для сортування
вибором №2
void sort_type_Insertion(int type); // заповнення масиву значень часу для сортування
"обмін-вставка"
void sort_type_Select7(int type); // заповнення масиву значень часу для сортування
вибором №7
void sort_type_Select2_Vect(int type); // заповнення вектора значень часу для
сортування вибором №2

```

```

void sort_type_Insertion_Vect(int type); //заповнення вектора значень часу для
сортування "обмін-вставка"
void sort_type_Select7_Vect(int type); //заповнення вектора значень часу для
сортування вибором №7
float MeasurementProcessing(); // Функція обробки і усереднення значень вимірів

#endif // MEASUREMENT_H_INCLUDED

```

Measurement.c

```

#include "Measurement.h"
#include "sorting.h"
#include "array.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>

clock_t Res[measurements_number]; // Масив значень часу роботи алгоритма

void sort_type_Select2(int type) //заповнення масиву значень часу для сортування
вибором №2
{
    for(int i=0; i<measurements_number; i++)
    {
        Creation(P, M, N);
        switch(type)
        {
            case 1:
                Fill_Arr3D_Sort(P, M, N);
                break;
            case 2:
                Fill_Arr3D_Rand(P, M, N);
                break;
            case 3:
                Fill_Arr3D_Back(P, M, N);
                break;
            default:
                break;
        }
        Res[i] = Select2(Arr3D, P, M, N);
        Free(P,M);
    }
}

void sort_type_Insertion(int type) //заповнення масиву значень часу для сортування
"обмін-вставка"
{
    for(int i=0; i<measurements_number; i++)
    {
        Creation(P, M, N);
        switch(type)
        {
            case 1:
                Fill_Arr3D_Sort(P, M, N);
                break;
            case 2:
                Fill_Arr3D_Rand(P, M, N);
                break;
            case 3:
                Fill_Arr3D_Back(P, M, N);
                break;
            default:
                break;
        }
    }
}

```

```

        break;
    }

    Res[i] = InsertExchange (Arr3D, P, M, N);
    Free (P,M);
}

void sort_type_Select7(int type) //заповнення масиву значень часу для сортування
вибором №7
{
    for(int i=0; i<measurements_number; i++)
    {
        Creation(P, M, N);
        switch(type)
        {
            case 1:
                Fill_Arr3D_Sort (P, M, N);
                break;
            case 2:
                Fill_Arr3D_Rand (P, M, N);
                break;
            case 3:
                Fill_Arr3D_Back (P, M, N);
                break;
            default:
                break;
        }
        Res[i] = Select7 (Arr3D, P, M, N);
        Free (P,M);
    }
}

void sort_type_Select2_Vect(int type) //заповнення вектора значень часу для
сортування вибором №2
{
    for(int i=0; i<measurements_number; i++)
    {
        Creation_Vect (M*N);
        switch(type)
        {
            case 1:
                Fill_Vect_Sort (M*N);
                break;
            case 2:
                Fill_Vect_Rand (M*N);
                break;
            case 3:
                Fill_Vect_Back (M*N);
                break;
            default:
                break;
        }
        Res[i] = Select2_Vect (Vector,M*N);
        free (Vector);
    }
}

void sort_type_Insertion_Vect(int type) //заповнення вектора значень часу для
сортування "обмін-вставка"
{
    for(int i=0; i<measurements_number; i++)
    {
        Creation_Vect (M*N);

```

```

        switch(type)
        {
            case 1:
                Fill_Vect_Sort(M*N);
                break;
            case 2:
                Fill_Vect_Rand(M*N);
                break;
            case 3:
                Fill_Vect_Back(M*N);
                break;
            default:
                break;
        }
        Res[i] = InsertExchange_Vect(Vector,M*N);
        free(Vector);
    }
}

void sort_type_Select7_Vect(int type) //заповнення вектора значень часу для
//сортування вибором №7
{
    for(int i=0; i<measurements_number; i++)
    {
        Creation_Vect(M*N);
        switch(type)
        {
            case 1:
                Fill_Vect_Sort(M*N);
                break;
            case 2:
                Fill_Vect_Rand(M*N);
                break;
            case 3:
                Fill_Vect_Back(M*N);
                break;
            default:
                break;
        }
        Res[i] = Select7_Vect(Vector,M*N);
        free(Vector);
    }
}

float MeasurementProcessing() // Функція обробки і усереднення значень вимірів
{
    long int Sum;
    float AverageValue;

    clock_t buf;
    int L = rejected_number, R = measurements_number - 1;
    int k = rejected_number;
    for (int j=0; j < min_max_number; j++)
    {
        for (int i = L; i < R; i++)
        {
            if (Res[i] > Res[i + 1])
            {
                buf = Res[i];
                Res[i] = Res[i + 1];
                Res[i + 1] = buf;
                k = i;
            }
        }
    }
}

```

```

    R = k;
    for (int i = R - 1; i >= L; i--)
    {
        if (Res[i] > Res[i + 1])
        {
            buf = Res[i];
            Res[i] = Res[i + 1];
            Res[i + 1] = buf;
            k = i;
        }
    }
    L = k + 1;
}

Sum=0;
for (int i = rejected_number + min_max_number; i < measurements_number -
min_max_number; i++)
    Sum = Sum + Res[i];

AverageValue = (float)Sum / (float)(measurements_number - 2*min_max_number -
rejected_number);

return AverageValue;
}

```

table.h

```

#ifndef TABLE_H_INCLUDED
#define TABLE_H_INCLUDED

int table(int type); // функція виводу часу роботи різних методів сортування

#endif // TABLE_H_INCLUDED

```

table.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "array.h"
#include "Measurement.h"

int table(int type) // функція виводу часу роботи різних методів сортування
{
    switch(type) // вибір виведення (масив, вектор, чи одночасно)
    {
        case 1:
            printf("\t\tTable for array:\tP=%d, M=%d, N=%d\n", P, M, N);
            printf("\n\t\t\tSorted\t\tRandom\t\tReverse Sorted");

            printf("\nSelect2");
            sort_type_Select2(1);
            printf("\t\t\t%.2f", MeasurementProcessing());
            sort_type_Select2(2);
            printf("\t\t\t%.2f", MeasurementProcessing());
            sort_type_Select2(3);
            printf("\t\t\t%.2f\n", MeasurementProcessing());

            printf("\nInsertionExchange");
            sort_type_Insertion(1);

```

```

printf("\t%.2f", MeasurementProcessing());
sort_type_Insertion(2);
printf("\t\t%.2f", MeasurementProcessing());
sort_type_Insertion(3);
printf("\t\t%.2f\n", MeasurementProcessing());

printf("\nSelect7");
sort_type_Select7(1);
printf("\t\t\t%.2f", MeasurementProcessing());
sort_type_Select7(2);
printf("\t\t%.2f", MeasurementProcessing());
sort_type_Select7(3);
printf("\t\t%.2f\n", MeasurementProcessing());
break;
case 2:
printf("\n\t\tTable for vector:\tN=%d\n",M*N);
printf("\n\t\t\tSorted\t\tRandom\t\tReverse Sorted");
printf("\nSelect2");
sort_type_Select2_Vect(1);
printf("\t\t\t%.2f",MeasurementProcessing());
sort_type_Select2_Vect(2);
printf("\t\t%.2f", MeasurementProcessing());
sort_type_Select2_Vect(3);
printf("\t\t%.2f\n", MeasurementProcessing());

printf("\nInsertionExchange");
sort_type_Insertion_Vect(1);
printf("\t%.2f", MeasurementProcessing());
sort_type_Insertion_Vect(2);
printf("\t\t%.2f", MeasurementProcessing());
sort_type_Insertion_Vect(3);
printf("\t\t%.2f\n", MeasurementProcessing());

printf("\nSelect7");
sort_type_Select7_Vect(1);
printf("\t\t\t%.2f", MeasurementProcessing());
sort_type_Select7_Vect(2);
printf("\t\t%.2f", MeasurementProcessing());
sort_type_Select7_Vect(3);
printf("\t\t%.2f\n", MeasurementProcessing());
break;
case 3:
table(1);
table(2);
break;
default:
printf("Incorrect data, try again");
sleep(1);
return 0;
break;
}
return 1;
}

```

Тестування коректності роботи методів сортування

P=3, M=N=5-для всіх випадків сотування.

Select2(сортування вибором №2)									
Впорядкований					Невпорядкований				
Початковий		Кінцевий			Початковий		Кінцевий		
P=3, M=5, N=5		Sorted			P=3, M=5, N=5		Sorted		
Not sorted					Not sorted				
0	5	10	15	20	20	20	28	74	16
1	6	11	16	21	73	0	20	22	54
2	7	12	17	22	50	66	42	0	45
3	8	13	18	23	4	60	57	13	17
4	9	14	19	24	71	37	49	73	13
25	30	35	40	45	5	43	31	68	73
26	31	36	41	46	35	20	70	31	9
27	32	37	42	47	44	71	34	57	10
28	33	38	43	48	72	40	45	15	38
29	34	39	44	49	16	11	19	54	57
50	55	60	65	70	47	41	71	36	35
51	56	61	66	71	46	19	18	56	31
52	57	62	67	72	36	1	62	48	48
53	58	63	68	73	53	64	62	28	24
54	59	64	69	74	21	3	16	13	60

InsertExchange (Гібридний алгоритм "вставка – обмін")									
Впорядкований					Невпорядкований				
Початковий		Кінцевий			Початковий		Кінцевий		
P=3, M=5, N=5		Sorted			P=3, M=5, N=5		Sorted		
Not sorted					Not sorted				
0	5	10	15	20	13	32	26	28	3
1	6	11	16	21	59	8	23	59	68
2	7	12	17	22	67	0	61	17	14
3	8	13	18	23	10	25	46	35	69
4	9	14	19	24	18	44	13	53	33
25	30	35	40	45	63	51	70	63	73
26	31	36	41	46	32	21	70	42	49
27	32	37	42	47	19	15	52	55	49
28	33	38	43	48	46	16	60	28	63
29	34	39	44	49	6	13	72	37	45
50	55	60	65	70	43	25	39	42	25
51	56	61	66	71	54	4	21	41	58
52	57	62	67	72	55	50	72	68	9
53	58	63	68	73	22	62	43	54	42
54	59	64	69	74	39	69	57	12	54

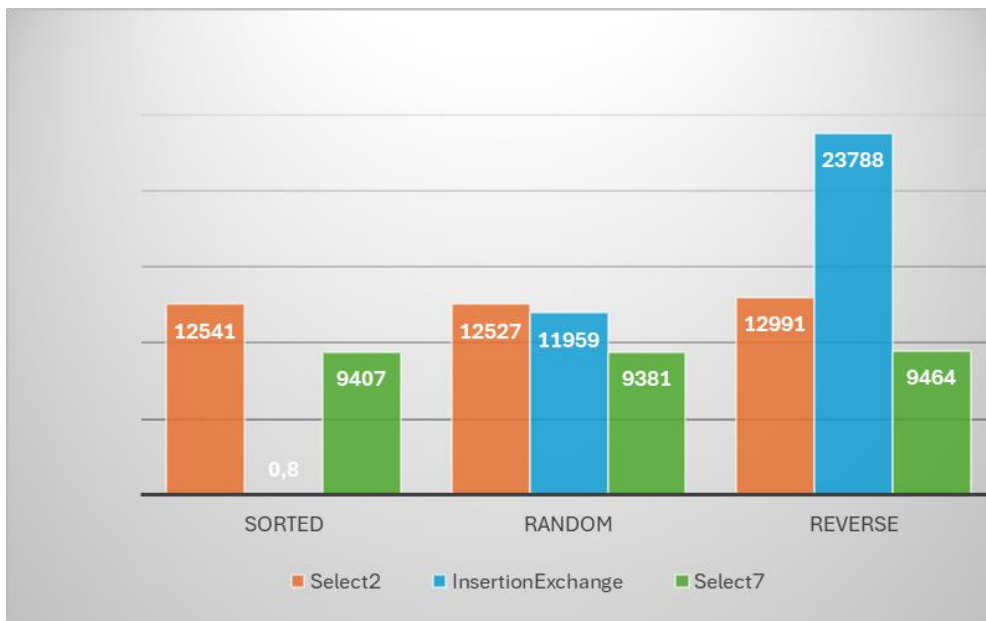
Select7(сортування вибором №7)														
Впорядкований					Невпорядкований					Обернено впорядкований				
Початковий		Кінцевий			Початковий		Кінцевий			Початковий		Кінцевий		
P=3, M=5, N=5					P=3, M=5, N=5					P=3, M=5, N=5				
Not sorted					Not sorted					Not sorted				
0	5	10	15	20	0	5	10	15	20	75	70	65	60	55
1	6	11	16	21	1	6	11	16	21	74	69	64	59	54
2	7	12	17	22	2	7	12	17	22	73	68	63	58	53
3	8	13	18	23	3	8	13	18	23	72	67	62	57	52
4	9	14	19	24	4	9	14	19	24	71	66	61	56	51
25	30	35	40	45	25	30	35	40	45	50	45	40	35	30
26	31	36	41	46	26	31	36	41	46	49	44	39	34	29
27	32	37	42	47	27	32	37	42	47	48	43	38	33	28
28	33	38	43	48	28	33	38	43	48	47	42	37	32	27
29	34	39	44	49	29	34	39	44	49	46	41	36	31	26
50	55	60	65	70	50	55	60	65	70	25	20	15	10	5
51	56	61	66	71	51	56	61	66	71	24	19	14	9	4
52	57	62	67	72	52	57	62	67	72	23	18	13	8	3
53	58	63	68	73	53	58	63	68	73	22	17	12	7	2
54	59	64	69	74	54	59	64	69	74	21	16	11	6	1

Результати (таблиці виміру часу та побудовані за даними таблицями стовпчикові діаграми та графіки).

Випадок дослідження I. Залежність часу роботи алгоритмів від форми перерізу масива.

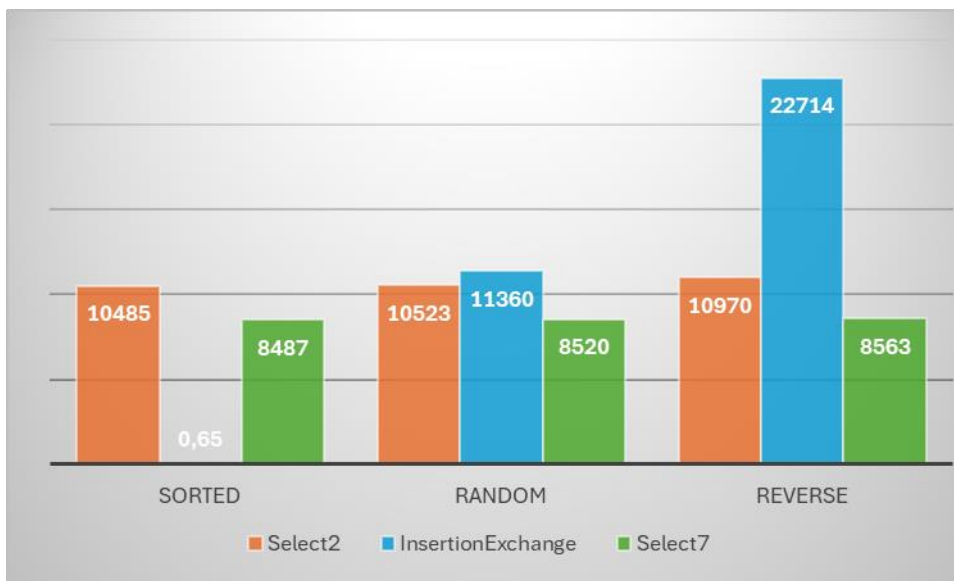
$P = \text{const} = 3, M = 2, N = 25000, M \cdot N = \text{const} = 50000.$

Table for array: P=3, M=2, N=25000			
	Sorted	Random	Reverse Sorted
Select2	12541.4	12527.5	12991.3
InsertionExchange	0.8000	11959.2	23788.1
Select7	9407.5	9381.8	9464.3



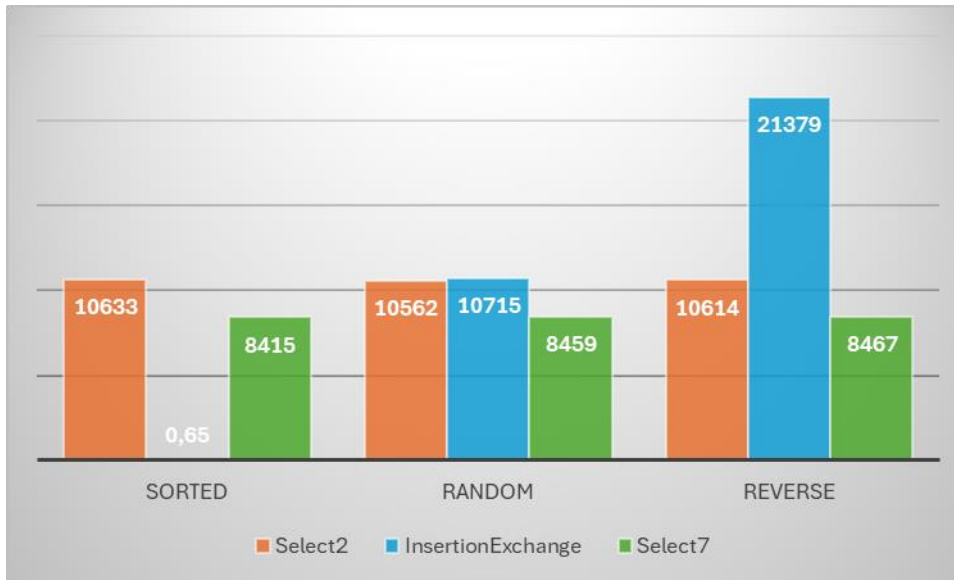
$P = \text{const} = 3, M = 5, N = 10000, M \cdot N = \text{const} = 50000.$

Table for array:		P=3, M=5, N=10000		
	Sorted	Random	Reverse	Sorted
Select2	10485.85	10523.65	10970.15	
InsertionExchange	0.6500	11360.35	22714.40	
Select7	8487.45	8520.70	8563.15	



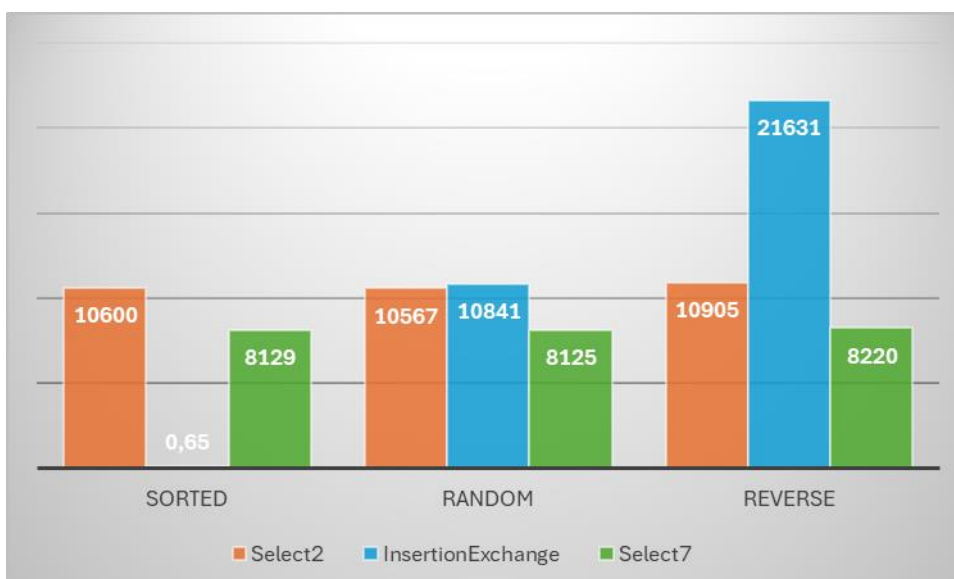
$P = \text{const} = 3, M = 50, N = 1000, M * N = \text{const} = 50000.$

Table for array:		P=3, M=50, N=1000	
	Sorted	Random	Reverse Sorted
Select2	10633.75	10562.00	10614.60
InsertionExchange	0.6500	10715.40	21379.30
Select7	8415.80	8459.60	8487.60



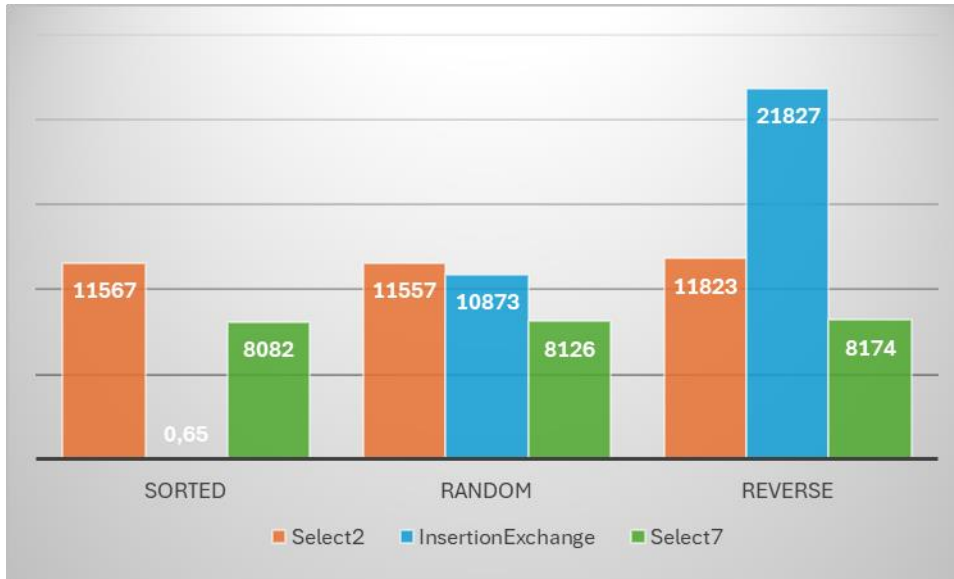
$P = \text{const} = 3, M = 500, N = 100, M * N = \text{const} = 50000.$

Table for array:		P=3, M=500, N=100	
	Sorted	Random	Reverse Sorted
Select2	10600.10	10567.20	10905.55
InsertionExchange	0.6500	10841.50	21631.80
Select7	8129.75	8125.25	8220.95



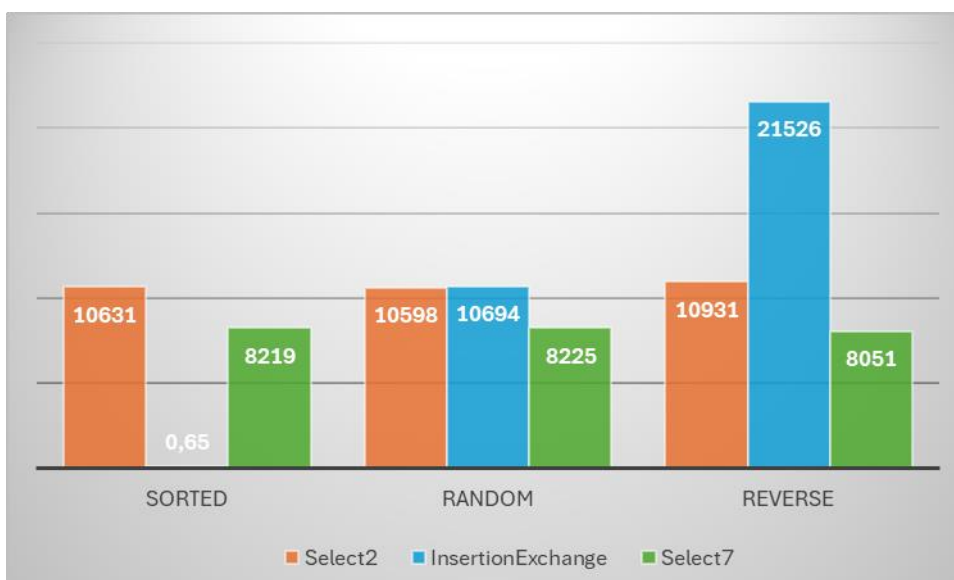
$P = \text{const} = 3, M = 5000, N = 10, M \cdot N = \text{const} = 50000.$

Table for array:		P=3, M=5000, N=10	
	Sorted	Random	Reverse Sorted
Select2	11567.5	11557.8	11823.4
InsertionExchange	0.6500	10873.6	21827.1
Select7	8082.0	8126.4	8174.4



$P = \text{const} = 3, M = 25000, N = 2, M \cdot N = \text{const} = 50000.$

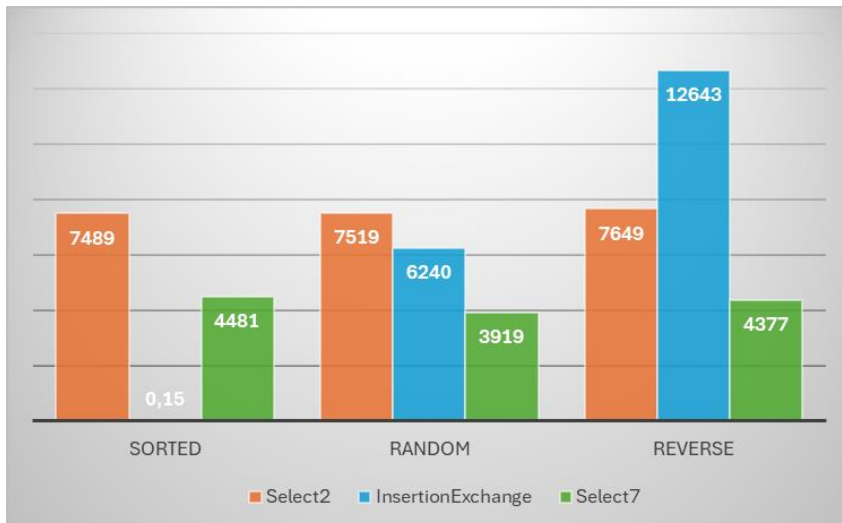
Table for array:		P=3, M=25000, N=2	
	Sorted	Random	Reverse Sorted
Select2	10631.2	10598.5	10931.5
InsertionExchange	0.6500	10694.2	21526.9
Select7	8219.6	8225.0	8051.8



$P = \text{const} = 3, M \cdot N = \text{const} = 50000;$

Vector Length= $N \cdot M = 50000$

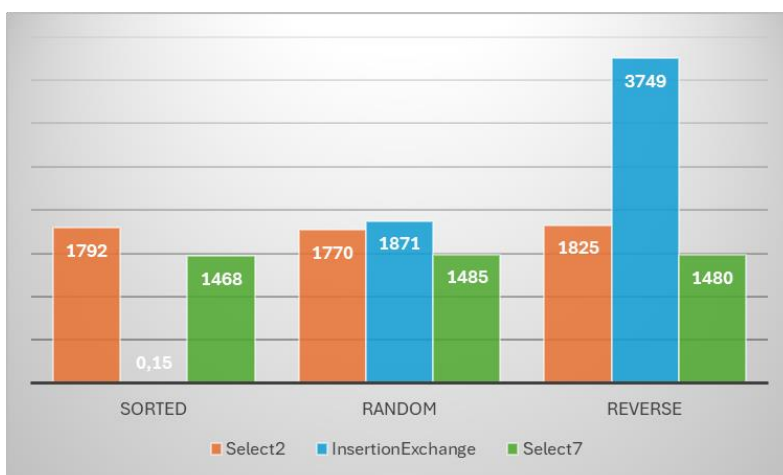
Table for vector:		N=50000	
	Sorted	Random	Reverse Sorted
Select2	7489.7	7519.1	7649.1
InsertionExchange	0.1500	6240.2	12643.6
Select7	4481.4	3919.0	4377.9



Випадок дослідження II. Залежність часу роботи алгоритмів від кількості перерізів масиву

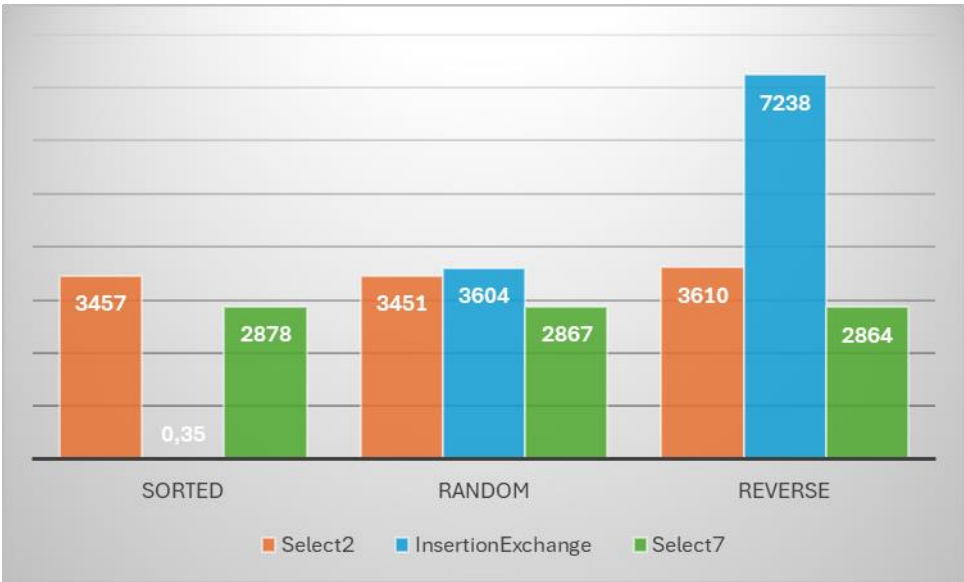
$P = 2, M = N = \text{const} = 160$

Table for array:		P=2, M=160, N=160	
	Sorted	Random	Reverse Sorted
Select2	1792.0	1770.4	1825.1
InsertionExchange	0.1500	1871.1	3749.3
Select7	1468.2	1485.8	1480.9



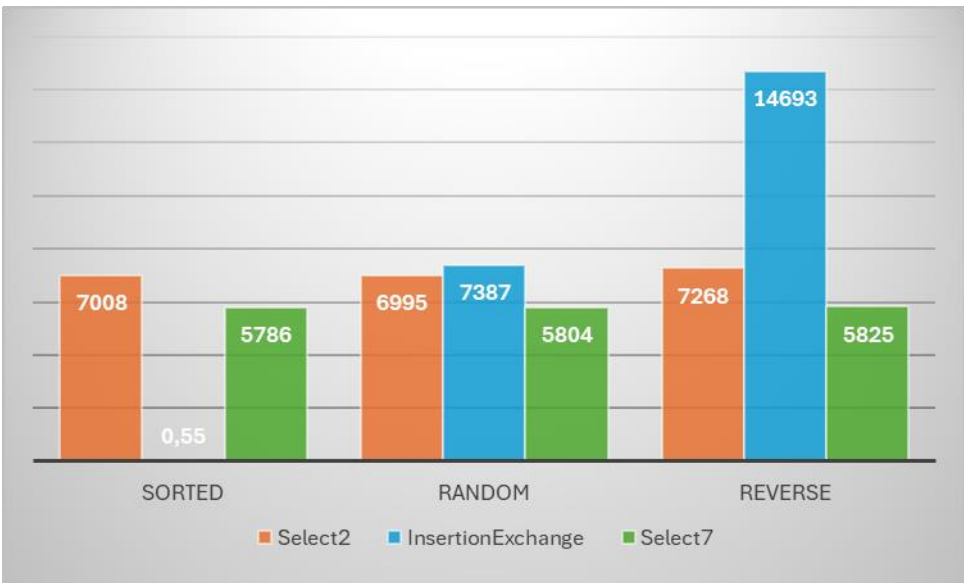
P = 4, M = N =const=160

Table for array:		P=4, M=160, N=160	
	Sorted	Random	Reverse Sorted
Select2	3457.1	3451.9	3610.1
InsertionExchange	0.3500	3604.4	7238.0
Select7	2878.9	2867.3	2864.1



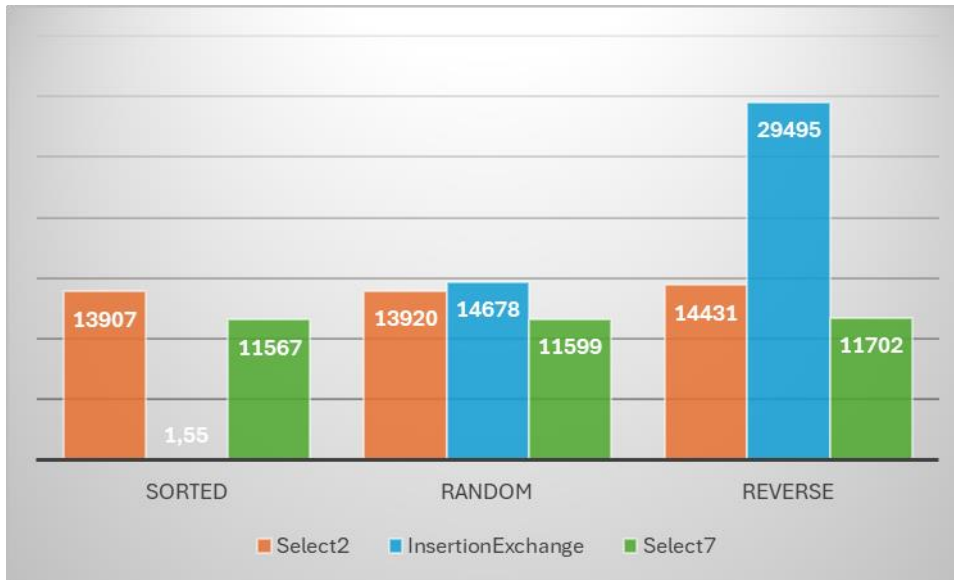
P = 8, M = N =const=160

Table for array:		P=8, M=160, N=160	
	Sorted	Random	Reverse Sorted
Select2	7008.0	6995.1	7268.5
InsertionExchange	0.5500	7387.0	14693.3
Select7	5786.9	5804.7	5825.4



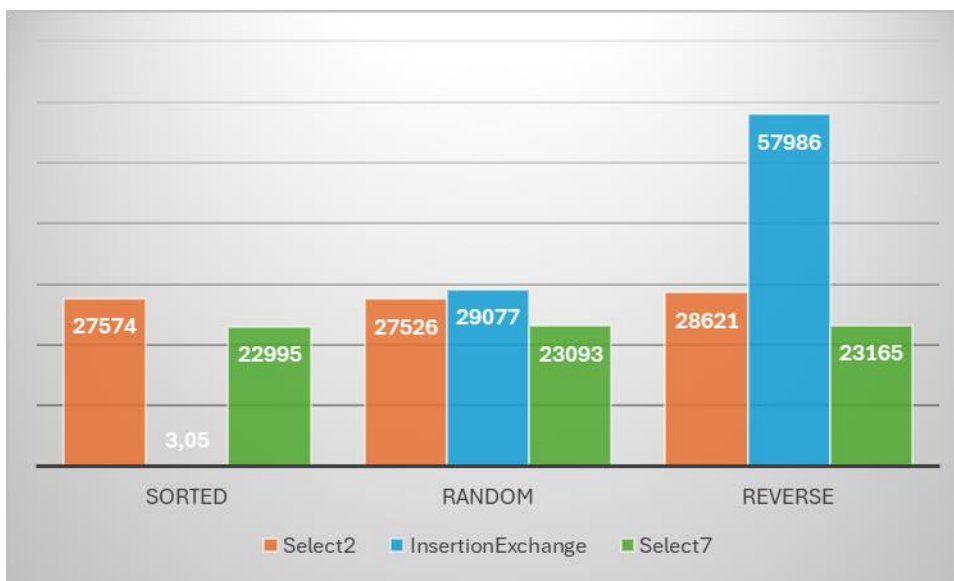
P = 16, M = N = const = 160

Table for array:		P=16, M=160, N=160	
	Sorted	Random	Reverse Sorted
Select2	13907.3	13920.7	14431.1
InsertionExchange	1.5500	14678.7	29495.1
Select7	11567.8	11599.5	11702.2



P = 32, M = N = const = 160

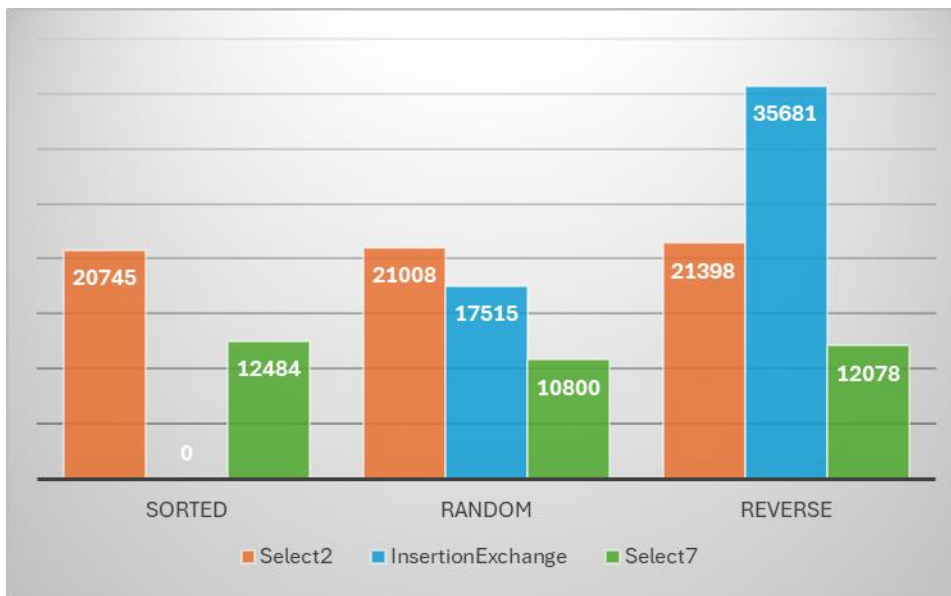
Table for array:		P=32, M=160, N=160	
	Sorted	Random	Reverse Sorted
Select2	27574.1	27526.6	28621.7
InsertionExchange	3.0500	29077.1	57986.1
Select7	22995.9	23093.2	23165.3



$P = 32, M = N = \text{const} = 160;$

Vector Length = $M * N = 160 * 160 = 25600$

Table for vector:		N=25600	
	Sorted	Random	Reverse Sorted
Select2	20745.6	21008.0	21398.4
InsertionExchange	0.0000	17515.2	35681.6
Select7	12484.8	10800.0	12078.4



Випадок дослідження III. Залежність часу роботи алгоритмів від розміру перерізів масиву

$P = \text{const} = 3, M = N = 4$

Table for array:		P=3, M=4, N=4	
	Sorted	Random	Reverse Sorted
Select2	0.0000	0.0000	0.0000
InsertionExchange	0.0000	0.0000	0.0000
Select7	0.0000	0.0000	0.0000

P =const=3, M = N =8

Table for array:		P=3, M=8, N=8	
	Sorted	Random	Reverse Sorted
Select2	0.0000	0.0000	0.0000
InsertionExchange	0.0000	0.0000	0.0000
Select7	0.0000	0.0000	0.0000

P =const=3, M = N =16

Table for array:		P=3, M=16, N=16	
	Sorted	Random	Reverse Sorted
Select2	0.2500	0.2500	0.2500
InsertionExchange	0.0000	0.2000	0.6000
Select7	0.1500	0.2000	0.1500



P =const=3, M = N =32

Table for array:		P=3, M=32, N=32	
	Sorted	Random	Reverse Sorted
Select2	4.7000	4.3000	4.7500
InsertionExchange	0.0000	5.0000	8.7500
Select7	3.9000	3.9000	3.7500



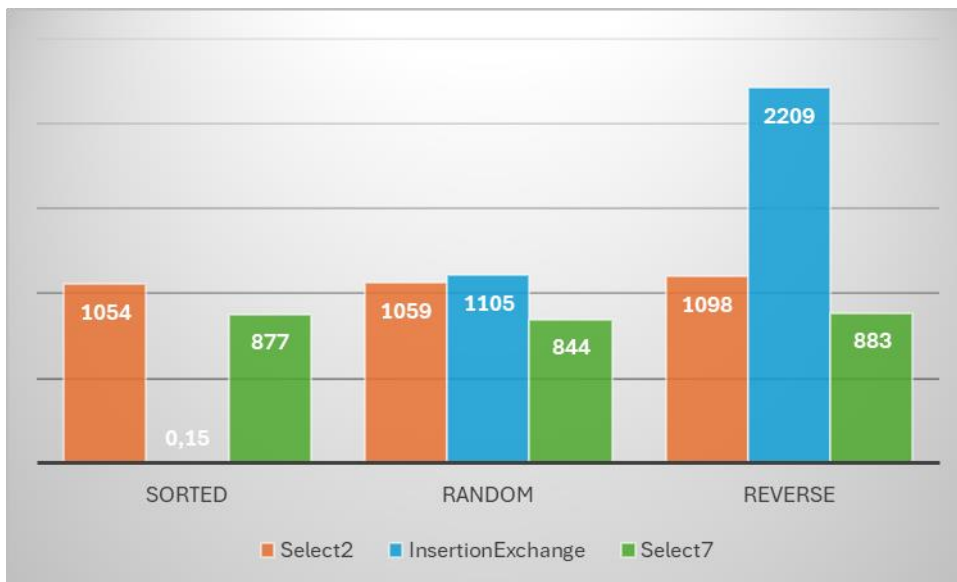
$P = \text{const} = 3, M = N = 64$

Table for array:		P=3, M=64, N=64	
	Sorted	Random	Reverse Sorted
Select2	68.9	70.6	70.4
InsertionExchange	0.0000	73.5	154.1
Select7	58.9	59.8	56.8



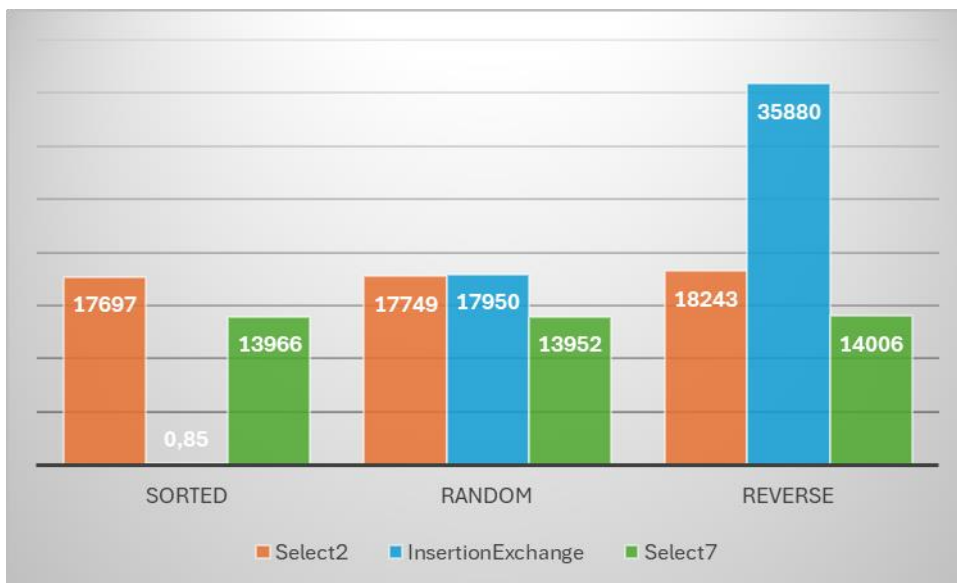
$P = \text{const} = 3, M = N = 128$

Table for array:		P=3, M=128, N=128	
	Sorted	Random	Reverse Sorted
Select2	1054.8	1059.3	1098.7
InsertionExchange	0.1500	1105.6	2209.1
Select7	877.2	884.8	883.6



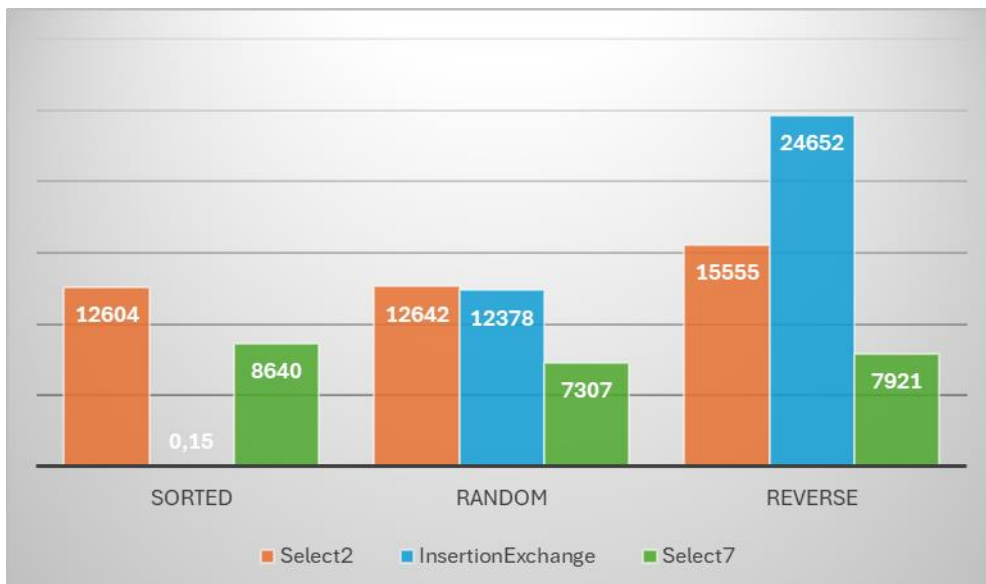
$P = \text{const} = 3, M = N = 256$

Table for array:		P=3, M=256, N=256	
	Sorted	Random	Reverse Sorted
Select2	17697.2	17749.8	18243.7
InsertionExchange	0.8500	17950.2	35880.9
Select7	13966.5	13952.0	14006.0



$P = \text{const} = 3, M = N = 256; \text{Vector Length} = M * N = 256 * 256 = 65536$

Table for vector:		N=65536	
	Sorted	Random	Reverse Sorted
Select2	12604.4	12642.6	15555.0
InsertionExchange	0.1500	12378.9	24652.2
Select7	8640.8	7307.1	7921.7



Порівняльний аналіз алгоритмів

У цій курсовій роботі визначив час сортування 3 алгоритмів, а зараз досліджу їх на ефективність і порівняю, а саме такі алгоритми: №2 методу прямого вибору, №7 методу прямого вибору та гібридний алгоритм “вставка-обмін”.

Розберемо особливості кожного з цих алгоритмів і проведемо порівняльний аналіз.

Розглянемо аспекти, які притаманні усім алгоритмам, і впливають із досліджень.

Випадок дослідження 1

Усі алгоритми залежні від форми перерізів, і коли перерізи набувають більш пропорційної форми час сортування зменшується для всіх випадків початкової відсортованості, також зменшення часу сортування спостерігається при дуже маленькій кількості стовпців, тому що - відбувається менше ітерацій переходу між стовпцями відповідно, це збільшує швидкодію.

Випадок дослідження 2

Для усіх алгоритмів час сортування прямо пропорційний кількості перерізів.

Випадок дослідження 3

В середньому, час сортування дорівнює квадрату зміни розміру масиву, як для трьох вимірних, так і для одно вимірних масивів, тобто при зміні $n \rightarrow 2n$, $m \rightarrow 2m$ час сортування зміниться в 4^2 разів, що видно з вимірів часу і графіків.

Алгоритм №2 методу прямого вибору(Select2)

Цей алгоритм найповільніше сортує обернено впорядкований масив, при чому, в усіх випадках дослідження, це пов'язано з тим, що при кожному проходженні по масиву змінювався індекс мінімального, а це плюс дві операції присвоєння у вкладеному масиві. Щодо впорядкованого масиву і не впорядкованого, то

тут час сортування майже однаковий і залежить від форми перерізу, розміру масиву і системи. Для вектора ситуація майже ідентична найшвидше впорядкований, далі - не впорядкований, і найповільніше- обернено впорядкований.

Гібридний алгоритм "вставка – обмін"(InsertExchange)

Він показує найкращий результат для сортованого масиву, що обумовлено його перевіркою сусідніх елементів, тобто при сортованому масиві він не виконує ніяких замін і зайвих перевірок, він просто йде по масиву і “підтверджує” його сортованість. При не впорядкованому масиві, цей алгоритм досягає свого результату за рахунок того, що він може як заходити в гілку обміну так і не заходити і в середньому виходить результат схожий на попередній алгоритм. При обернено впорядкованому цей алгоритм показує найгірший результат, за рахунок того що йому завжди доводиться переміщувати мінімальний елемент, який дорівнює значенню границі, на початок, при цьому змінює він тільки сусідні елементи і тому робить занадто багато ітерацій обміну, що й призводить до поганого результату.

Для вектора все аналогічно, окрім того, що він навіть трішки ефективніший для не впорядкованого заповнення порівняно з попереднім алгоритмом.

Алгоритм сортування №7 методу прямого вибору(Select7)

Цей алгоритм показує себе найкраще для не впорядкованого масиву, тому що тут є розгалуження для пошуку мінімального і максимального і він може як заходити та і не заходити в ці гілки і є можливість ще не дійшовши до кінця впорядкувати весь масив і тоді не буде потреби робити якісь обміни. На противагу цьому, впорядкований і обернено впорядкований сортуються довше тому що завжди заходять в гілку пошуку, бо так як ми йдемо з ліва на право, в обернено впорядкованого є перевага, тому що елемент границі одразу є максимальним, а так як спочатку йде перевірка на мінімальний, то він далі тільки змінює мінімальний і не виконує ще одну перевірку на максимальність і зміну індексу, але це нівелюється кількістю операцій для змін елементів місцями для обернено впорядкованого, і тому впорядкований масив сортується швидше. Але для вектора, ще одна перевірка

переважає зміну елементів місцями, і обернено впорядкований сортується швидше, та все одно, найшвидше сортується не впорядкований вектор.

Тепер розглянувши всі слабкі і сильні сторони алгоритмів можна переходити до їх порівняння. Однозначно, що в середньому, для всіх випадків відсортованості трьох вимірних масивів найшвидше і найстабільніше працює **Select7**, випереджаючи своїх конкурентів в середньому на 20-25%, а для векторів ця різниця досягає майже двократного значення, і обумовлено це тим, що в цьому масиві кількість ітерацій вдвічі менша за рахунок двох границь, які зміщуються одночасно з двох сторін, якщо брати кількість ітерацій **Select2=n**, то **Select7=n/2**, та для цього він задіює більшу кількість операцій всередині циклів, для трьох вимірних масиву ця кількість ітерацій не відіграє настільки важливу роль, так як, ще більше збільшується кількість операцій всередині циклів. Але попри це, найкращим алгоритмом для впорядкованого масиву, є **InsertExchange**, за рахунок того, що описано вище в особливостях алгоритмів. І порівнюючи його з алгоритмом **Select2**, який є раціональнішим, через свою відносну стабільність в різних випадках початкової відсортованості, хоч для випадків не відсортованості **InsertExchange** показує не набагато гірший результат, і якщо порівнювати сортування векторів то результат навіть кращий, для оберненого сортування **InsertExchange** програє дуже сильно **Select2**, і цим нівелює всю перевагу. Тому, є лише один випадок в якому він сильно перемагає, а в інших двох – програє, в одному трішки, але в іншому-сильно.

Висновки по отриманих результатах

У цій курсовій роботі я дослідив 3 алгоритми, а саме такі алгоритми: №2 методу прямого вибору, №7 методу прямого вибору та гібридний алгоритм “вставка-обмін”. Порівнюючи їх швидкодію та ефективність я дійшов таких висновків:

Найкращим серед заданих алгоритмів - алгоритм сортування №7 методу прямого вибору(**Select7**), він показує себе однаково добре для всіх випадків відсортованості і як сказано вище обганяє своїх конкурентів в середньому на 20-25%.

Другим стає - алгоритм №2 методу прямого вибору(**Select2**), він показує, не погані, стабільні результати, але ці результати помітно гірші за попередні.

Останнім стає - гібридний алгоритм "вставка – обмін" (InsertExchange), зважаючи на його нестабільність в значеннях швидкодії для різного типу відсортованості початкового масиву, він найчастіше програє, (в двох з трьох випадків) алгоритмам на попередніх місцях.

Список літератури

- 1. Методичні матеріали для виконання курсової роботи.*
- 2. Лекції з курсу “Структури даних і алгоритми”*
- 3. Вирт Н. Алгоритмы и структуры данных. М.: Мир, 1989.*