

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: Овчінніков Д.С.

Група: КВ-14

Номер залікової книжки: КВ-13455360

Допущений до захисту

---

2 семестр 2021/2022

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних  
систем

Узгоджено

ЗАХИЩЕНА "\_\_\_" \_\_\_\_\_ 2022р.

Керівник роботи

з оцінкою \_\_\_\_\_

\_\_\_\_\_/Марченко О.І./

\_\_\_\_\_/Марченко О.І./

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

Виконавець роботи:  
Овчінніков Дмитро Станіславович

\_\_\_\_\_ 2022 р.

# ***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 впорядкованого, випадкового і обернено-впорядкованого масива) для масива з заданими геометричними розмірами повинна бути такою:

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

	Впорядкований	Невпорядкований	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 ґрунтовного аналізу алгоритмів потрібно зробити виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами.

Зробити виміри часу для стандартного випадку одновимірного 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 одновимірного масива відносно загальновідомої теорії;
- для багатовимірних масивів відносно результатів для одновимірного 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 № 135

### Задача

Впорядкувати окремо кожен переріз тривимірного масива  $\text{Arr3D}[P,M,N]$  таким чином: переставити стовпчики перерізу за незменшенням значень елементів його першого рядка.

### Досліджувані методи та алгоритми

1. Алгоритм сортування №2 методу прямого вибору.
2. Алгоритм сортування №2 методу прямого обміну з використанням прапорця.
3. Алгоритм №2 методу сортування Шелла (спосіб реалізації на основі гібридного алгоритму «вставка-обмін»). Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються.

### Способи обходу

Використовуючи елементи першого рядка кожного перерізу як ключі сортування, переставляти відповідні стовпчики кожен раз, коли треба переставляти ключі. При перестановці стовпчиків потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови C/C++.

### Випадки дослідження

1. Елементи початкового масиву впорядковані відповідно до заданої ознаки.
2. Елементи початкового масиву неупорядковані.
3. Елементи початкового масиву впорядковані протилежно заданої ознаки.

## Схема роботи заданих алгоритмів

### 1. Алгоритм сортування №2 методу прямого вибору.

Принцип. Масив в кожний момент часу поділений на вже відсортовану та ще не відсортовану частини.

Перед початком сортування відсортована частина пуста..

Процес сортування:

1) Беремо черговий елемент і присвоюємо його індекс індексу мінімального з невідсортованої частини і порівнюємо його з наступними елементами, якщо черговий елемент менший за мінімальний то значенню індексу мінімального присвоюється індекс чергового елемента. Йдемо далі по невідсортованій частині поки не закінчиться масив.

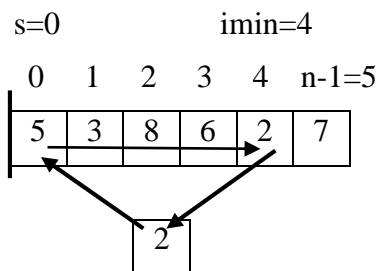
2)Міняємо містами мінімальний елемент масиву з першим елементом невідсортованої частини.Границя відсортованої частини збільшується на 1.

## Загальна схема

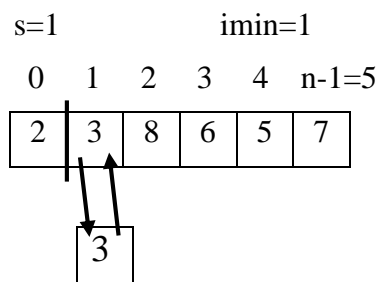
Початковий масив

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

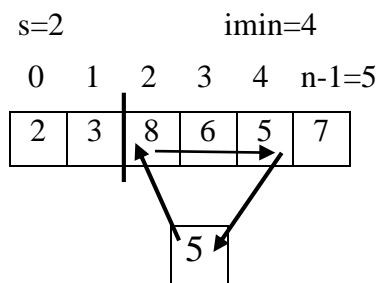
### Перший прохід



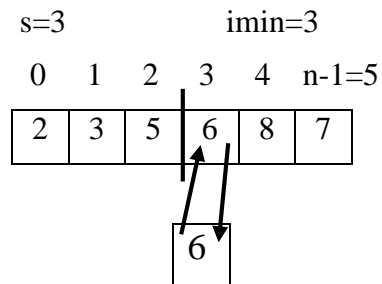
### Другий прохід



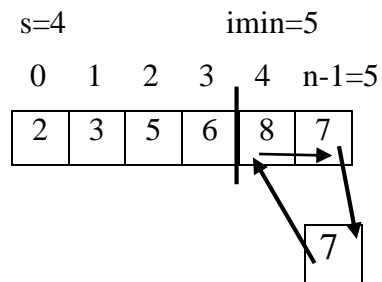
### Третій прохід



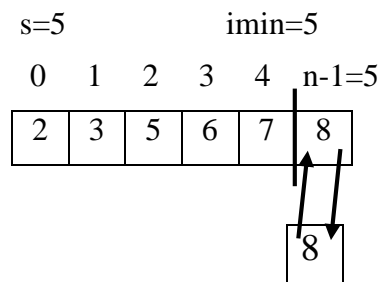
### Четвертий прохід



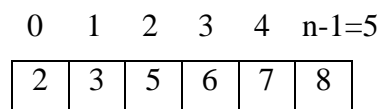
### П'ятий прохід



### Шостий прохід



### Результат



### Алгоритм на мові C

```
void SelectSort(int *A, int N)
{
    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;
    }
}
```

```

        A[s]=tmp;
    }
}

```

## 2. Алгоритм сортування №2 методу прямого обміну (з використанням прапорця).

Принцип. Цей алгоритм це модифікований алгоритм прямого обміну. Він модифікований флажком, ціль якого зменшити час виконання завдяки його перевірці на те чи змінювались місцями елементи масиву. Суть цього алгоритму полягає у зміні елементів місцями якщо наступний елемент менший за попередній.

Процес сортування:

- 1) Присвоюємо R значення N-1 а флажку значення True.
- 2) Перевіряємо чи флажок==True, якщо так то переходимо у тіло циклу і відразу обнуляємо флажок
- 3) Перевіряєм сусідні значення масиву з 0 елементу до R не включно, щоб не вийти за межі масиву. Якщо наступний елемент менший за попередній то вони міняються місцями і флажок переходить в значення True.
- 4) Після проходу зменшуємо R на 1, бо справа вже стоїть максимальний елемент і він більше переставляється не буде.

## Загальна схема

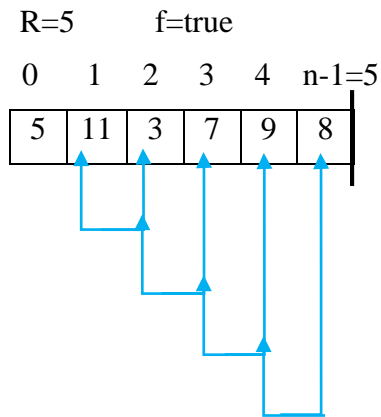
Початковий масив

0    1    2    3    4    n-1=5

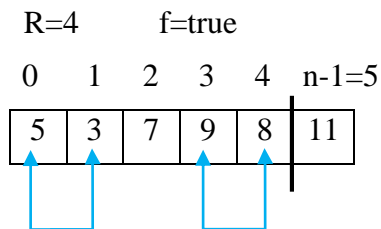
5	11	3	7	9	8
---	----	---	---	---	---



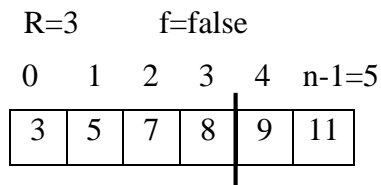
## Перший прохід



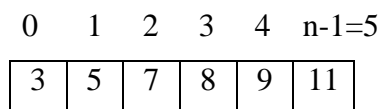
## Другий прохід



## Третій прохід



## Результат



## Алгоритм на мові С

```
void ExchangeSort(int *A, int N)
{
    int R, flag, tmp;

    R=N-1; flag=1;
    while(flag == 1)
    {
        flag=0;
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1])
            {
                tmp=A[i];
```

```

        A[i]=A[i+1];
        A[i+1]=tmp;
        flag=1;
    }
    R--;
}
}

```

### 3. Алгоритм №2 методу сортування Шелла

Принцип. Цей алгоритм це покращений алгоритм вставки, покращений тим що він сортує не весь масив відразу а по частинкам , що економить час виконання бо алгоритму не потрібно багато часу на непотрібне переміщення.

Процес сортування:

- 1) Знаходимо кількість етапів (Залежить від розмірів масиву)
- 2) Присвоюємо масиву етапів значення того на скільки частин поділиться масив при сортуванні у кожному етапі.
- 3) Відсортовуємо окрему кожену групу, після відсортування йдемо на наступний етап, зменшуючи кількість груп поки повністю не відсортуємо.

## Загальна схема

Початковий масив

0	1	2	3	4	5	6	n-1=7
9	3	5	6	1	4	10	2

t=2

Отже буде вирішуватись в два етапи.

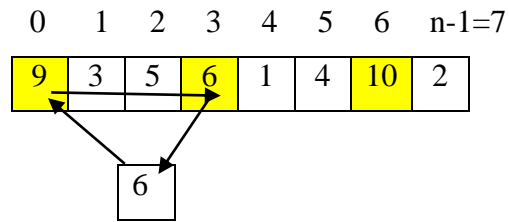
Stages[1]=1; Stages[0]=3;

### Перший прохід 1 етапу

k=3;

i=3;

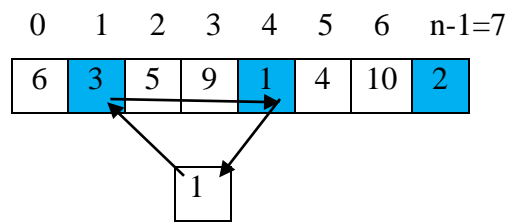
j=3;



### Другий прохід 1 етапу

i=4

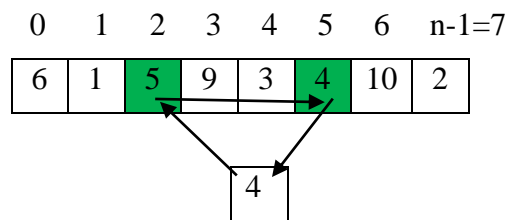
j=4



### Третій прохід 1 етапу

i=5

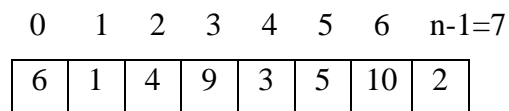
j=5



### Четвертий прохід 1 етапу

i=6

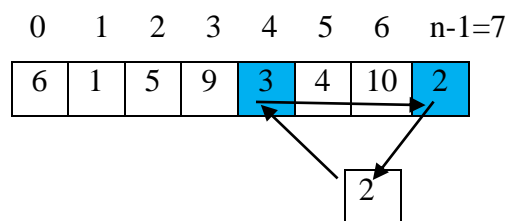
j=6



### П'ятий прохід 1 етапу

$i=7$

$j=7$

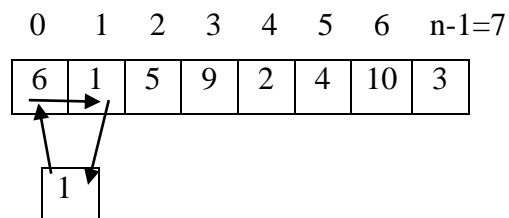


### Перший прохід 2 етапу

$k=1$

$i=1$

$j=1$

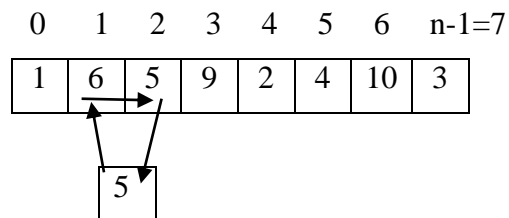


### Другий прохід 2 етапу

$k=1$

$i=2$

$j=2$



### Третій прохід 2 етапу

k=1

i=3

j=3

0	1	2	3	4	5	6	n-1=7
1	5	6	9	2	4	10	3

0
---


### Четвертий прохід 2 етапу

k=1

i=4

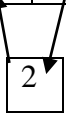
j=4

0	1	2	3	4	5	6	n-1=7
1	5	6	9	2	4	10	3




j=3

0	1	2	3	4	5	6	n-1=7
1	5	6	2	9	4	10	3



j=2

0	1	2	3	4	5	6	n-1=7
1	5	2	6	9	4	10	3



j=1

0	1	2	3	4	5	6	n-1=7
1	2	5	6	9	4	10	3

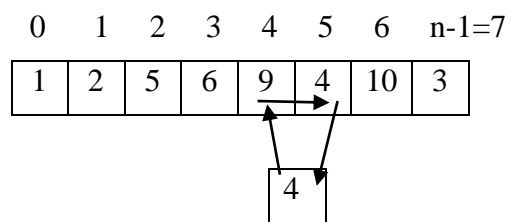


### П'ятий прохід 2 етапу

k=1

i=5

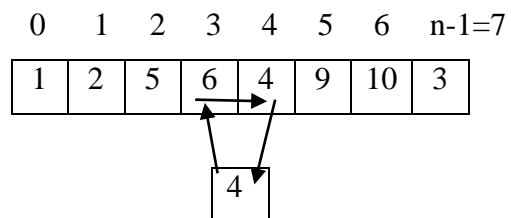
j=5



k=1

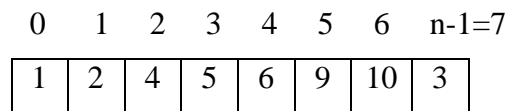
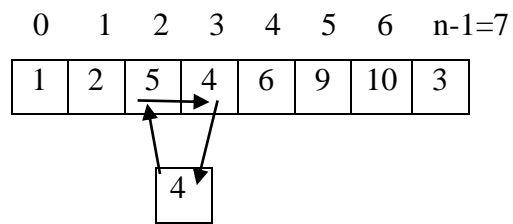
i=5

j=4



i=5

j=3

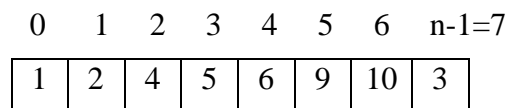


### Шостий прохід 2 етапу

k=1

i=6

j=6

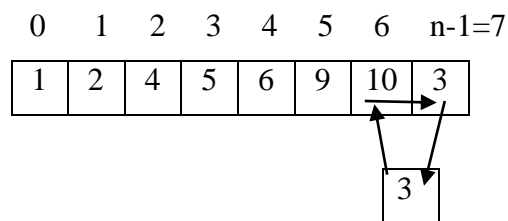


### Сьомий прохід 2 етапу

k=1

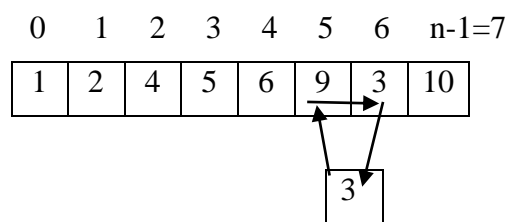
i=7

j=7



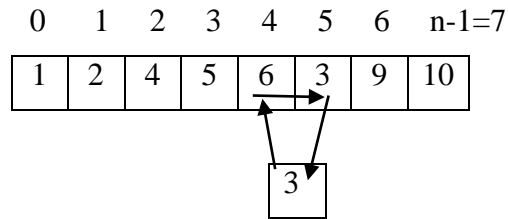
i=7

j=6



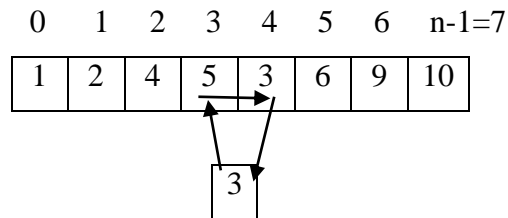
i=7

j=5



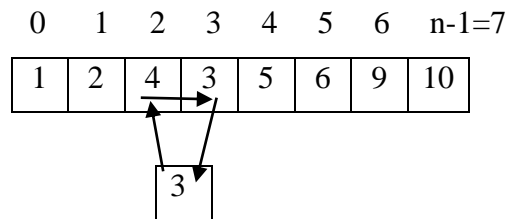
i=7

j=4

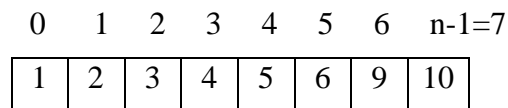


i=7

j=3



## Результат



## Алгоритм на мові С

```
void ShellSort(int *A, int N)
{
    int tmp, t, j, k;

    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;
    int Stages[t];
    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
        Stages[i]=2*Stages[i+1]+1;
    for (int p=0; p<t; p++)
    {
        k=Stages[p];
        for (int i=k; i<N; i++)
        {
            j=i;
```

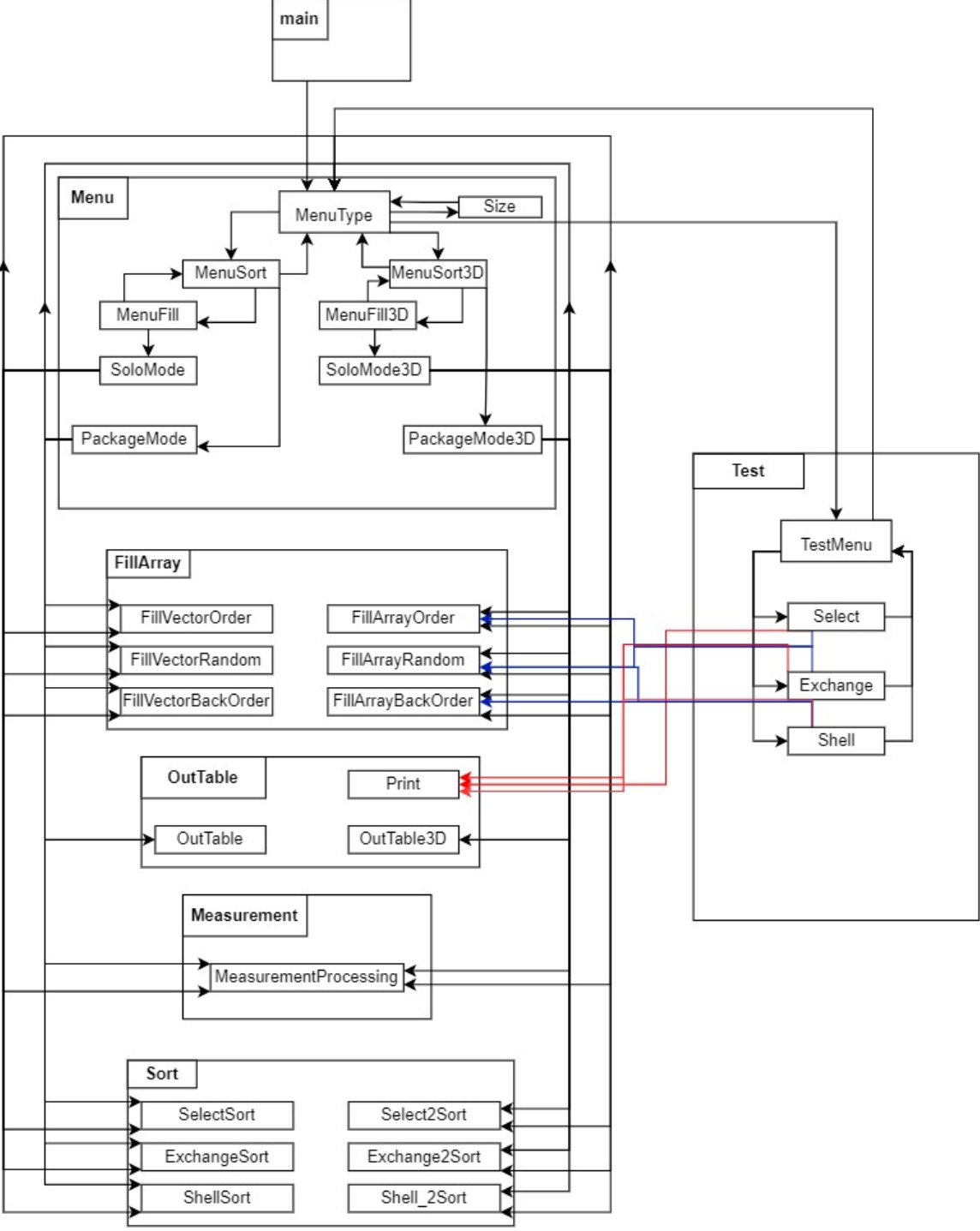


```

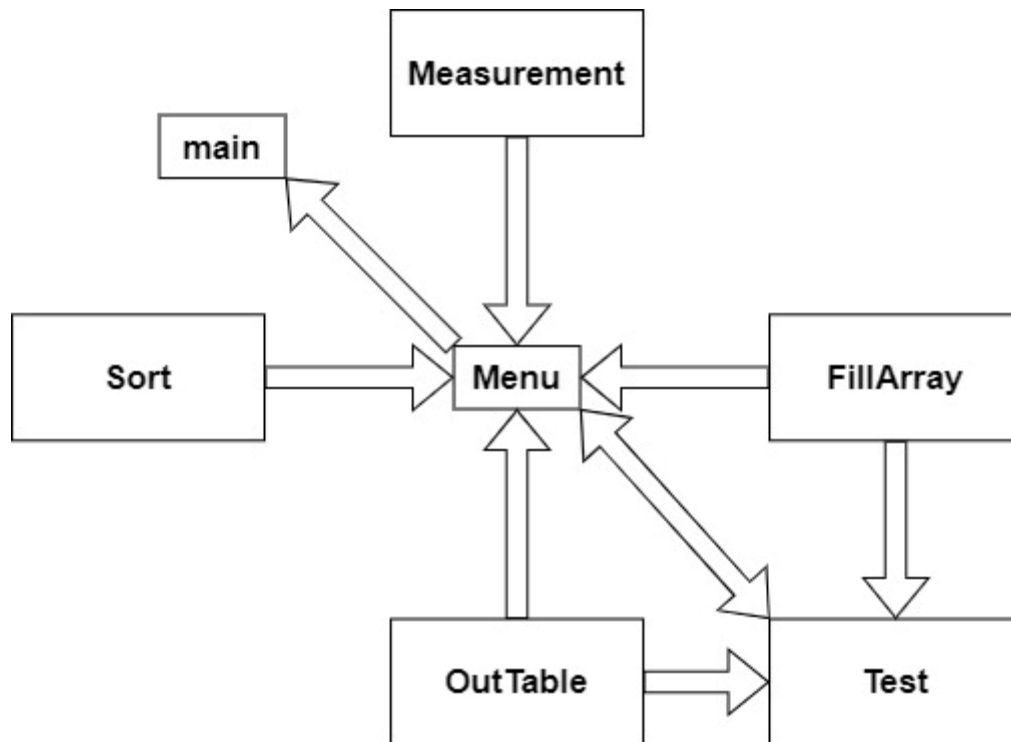
while (j>=k && A[j]<A[j-k])
{
    tmp=A[j];
    A[j]=A[j-k];
    A[j-k]=tmp;
    j=j-k;
}
}
}
}

```

## Структурна схема взаємовикликів процедур та функцій



## Схема імпорту/експорту модулів



## Опис призначення процедур та функцій

### 1) Модуль **Measurement** :

**MeasurementProcessing** – функція для вираховування часу роботи конкретного алгоритму шляхом вираховування середнього арифметичного відкидануючи значення трьох найбільших значень та трьох найменших значень з масива часу Res, та запису цього часу в AverageValue , значення якого вона і повертає.

### 2) Модуль **FillArray** :

**FillArrayOrder** – процедура впорядкованого заповнення трьохвимірної масиву.

**FillArray Random** – процедура невпорядкованого заповнення трьохвимірної масиву.

**FillArray BackOrder** – процедура обернено впорядкованого заповнення трьохвимірної масиви.

**FillVectorOrder** – процедура впорядкованого заповнення вектора.

**FillVectorRandom** – процедура неупорядкованого заповнення вектора.

**FillVectorBackOrder** – процедура обернено впорядкованого заповнення вектора.

### 3) Модуль **OutTable** :

**OutTable** – процедура виводу таблиці значень виміру масиви в пакетному режимі для вектора.

**OutTable3D** – процедура виводу таблиці значень виміру масиви в пакетному режимі для трьохвимірної масиви.

**Print** – процедура виводу трьохвимірної масиви.

### 4) Модуль **Sort** :

**Select2Sort** – функція що відсортовує перший рядок трьохвимірної масиви за неспаданням зі зміною місцями стовпчиків при зміні місцями ключів сортування методом 2 прямого вибору і повертає значення часу.

**SelectSort** – функція що відсортовує вектор за неспаданням методом 2 прямого вибору і повертає значення часу.

**Exchange2Sort** – функція що відсортовує перший рядок трьохвимірної масиви за неспаданням зі зміною місцями стовпчиків при зміні місцями ключів сортування методом 2 прямого обміну (з використанням прапорця) і повертає значення часу.

**Shell\_2Sort** – функція що відсортовує перший рядок трьохвимірної масиви за неспаданням зі зміною місцями стовпчиків при зміні місцями ключів сортування методом 2 алгоритмом сортування Шелла і повертає значення часу.

**ExchangeSort** – функція що відсортовує вектор за неспаданням методом 2 прямого обміну (з використанням прапорця) і повертає значення часу.

**ShellSort** — функція що відсортовує вектор за неспаданням методом 2 алгоритмом сортування Шелла і повертає значення часу.

5) Модуль **Test** :

**TestMenu** – процедура що виводить меню для вибору режиму тестування.

**Select** – процедура що відсортовує масив методом вибору №2 і виводить результат.

**Exchange** – процедура що відсортовує масив методом обміну №2(з флажком) і виводить результат.

**Shell** – процедура що відсортовує масив методом Шелла №2 і виводить результат.

6) Модуль **Menu** :

**Size** – процедура що змінює розмір 3-хвимірного масива.

**MenuType** – процедура вибору того що користувач хоче зробити.

**MenuSort** – процедура вибору користувачем алгоритму сортування вектору.

**MenuSort3D** – процедура вибору користувачем алгоритму сортування трьохвимірного масива.

**MenuFill** – процедура вибору варіанту заповнення вектора.

**MenuFill3D** – процедура вибору варіанту заповнення трьохвимірного масива.

**PackageMode** – процедура що викликає всі методи сортування при всіх варіаціях заповненості вектора.

**PackageMode3D** – процедура що викликає всі методи сортування при всіх варіаціях заповненості трьохвимірного масива.

**SoloMode** – процедура що викликає певний вид сортування при певних заповненості вектора.

**SoloMode3D** – процедура що викликає певний вид сортування при певних заповненості трьохвимірного масива.

# Текст головної програми та модулів

## main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "Menu.h"
int main()
{
    MenuType(); //процедура меню вибору типу сортування
    return 0;
}
```

## FillArray.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "FillArray.h"

void FillArrayOrder(int ***A, int P, int N, int M) // процедура
заповнення масиву впорядковано
{
    int number = 0;
    for (int k = 0; k < P; k++)
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < M; j++)
            {
                A[k][i][j] = number++;
            }
        }
    }
}

void FillArrayRandom(int ***A, int P, int N, int M) // процедура
заповнення масиву рандомно
{
    srand(time(NULL));
    for (int k = 0; k < P; k++)
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < M; j++)
            {
```

```

        A[k][i][j] = rand() % (P*N*M);
    }
}

}

void FillArrayBackOrder(int ***A, int P, int N, int M) // процедура
заповнення масиву обернено впорядковано
{
    int number = P*M*N;
    for (int k = 0; k < P; k++)
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < M; j++)
            {
                A[k][i][j] = number--;
            }
        }
    }
}

void FillVectorOrder(int *B, int N) // процедура заповнення вектору
впорядковано
{
    int number=0;
    for (int i = 0; i < N; i++)
    {
        B[i] = number++;
    }
}

void FillVectorRandom(int *B, int N) // процедура заповнення вектору
рандомно
{
    srand(time(NULL));
    for (int i = 0; i < N; i++)
    {
        B[i] = rand() % N;
    }
}

void FillVectorBackOrder(int *B, int N) // процедура заповнення вектору
обернено впорядковано
{
    int number=N;
    for (int i = 0; i < N; i++)
    {
        B[i] = number--;
    }
}

```

## FillArray.h

```

#ifndef __FillArray_H__
#define __FillArray_H__

void FillArrayOrder(int ***A,int P, int N, int M);
void FillArrayRandom(int ***A,int P, int N, int M);

```

```

void FillArrayBackOrder(int ***A,int P, int N, int M);

void FillVectorOrder(int *B, int N);
void FillVectorRandom(int *B, int N);
void FillVectorBackOrder(int *B, int N);

#endif // __FillArray_H__

```

## Measurement.c

```

#include "Measurement.h"
#include <stdio.h>

clock_t Res[measurements_number];

float MeasurementProcessing()
{
    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;

}

```

## Measurement.h

```

#ifndef __Measurement_H__
#define __Measurement_H__

#include <time.h>

// Загальна кількість вимірів часу роботи алгоритма
#define measurements_number 28
// #define measurements_number 1

// Кількість відкинутих початкових вимірів
#define rejected_number 2

// Кількість відкинутих вимірів з мінімальними значеннями.
// Вона ж дорівнює кількості відкинутих вимірів
// з максимальними значеннями.
#define min_max_number 3

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

// Функція обробки і усереднення значень вимірів
// часу роботи алгоритма.
// Повертає усереднене значення часу роботи алгоритма.
float MeasurementProcessing();

#endif

```

## OutTable.c

```

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

#include "OutTable.h"

void OutTable3D(int P, int N, int M, float SelectOrder, float
SelectRandom, float SelectBackOrder, float ExchangeOrder, float
ExchangeRandom, float ExchangeBackOrder, float ShellOrder, float
ShellRandom, float ShellBackOrder)
{ //процедура створення таблиці вимірів часу роботи алгоритмів
сортування
    printf("A[%d][%d][%d]\n", P, N, M);
    printf("%7s \t %7s \t %7s \t %7s \n", " ", "Order", "Random",
"Backorder");
    printf("%7s \t %7.2f \t %7.2f \t %7.2f \n", "SelectSort",
SelectOrder, SelectRandom, SelectBackOrder);
}

```

```

        printf("%7s \t %7.2f \t %7.2f \t %7.2f \n", "ExchangeSort",
ExchangeOrder, ExchangeRandom, ExchangeBackOrder);
        printf("%7s \t %7.2f \t %7.2f \t %7.2f \n", "ShellSort",
ShellOrder, ShellRandom, ShellBackOrder);
        printf("\n\n");
    }

    void OutTable(int M, float SelectOrder, float SelectRandom, float
SelectBackOrder, float ExchangeOrder, float ExchangeRandom, float
ExchangeBackOrder, float ShellOrder, float ShellRandom, float
ShellBackOrder)
    { //процедура створення таблиці вимірів часу роботи алгоритмів
    сортування
        printf("A[%d]\n", M);
        printf("%7s \t %7s \t %7s \t %7s \n", " ", "Order", "Random",
"Backorder");
        printf("%7s \t %7.2f \t %7.2f \t %7.2f \n", "SelectSort",
SelectOrder, SelectRandom, SelectBackOrder);
        printf("%7s \t %7.2f \t %7.2f \t %7.2f \n", "ExchangeSort",
ExchangeOrder, ExchangeRandom, ExchangeBackOrder);
        printf("%7s \t %7.2f \t %7.2f \t %7.2f \n", "ShellSort",
ShellOrder, ShellRandom, ShellBackOrder);
        printf("\n\n");
    }

    void Print(int ***A, int P, int N, int M) //процедура виводу 3 вимірного
масиву
    {
        for(int k=0; k<P; k++)
        {
            for(int i=0; i<N; i++)
            {
                for(int j=0; j<M; j++)
                {
                    printf(" %4d", A[k][i][j]);
                }
                printf("\n");
            }
            printf("\n\n\n");
        }
    }
}

```

## OutTable.h

```

#ifndef __OutTable_H__
#define __OutTable_H__
void OutTable3D(int P, int N, int M, float SelectOrder, float
SelectRandom, float SelectBackOrder, float ExchangeOrder, float
ExchangeRandom, float ExchangeBackOrder, float ShellOrder, float
ShellRandom, float ShellBackOrder);
void OutTable(int M, float SelectOrder, float SelectRandom, float
SelectBackOrder, float ExchangeOrder, float ExchangeRandom, float
ExchangeBackOrder, float ShellOrder, float ShellRandom, float
ShellBackOrder);
void Print(int ***A, int P, int N, int M);
#endif // __OutTable_H__

```

## Sort.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "Sort.h"

clock_t Select2Sort(int ***A, int P, int N, int M) // функція
сортювання 3-х вимірнього масиву методом прямого вибору
{
    int imin, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int k=0; k<P; k++)
    {
        for(int s=0; s<M-1; s++)
        {
            imin=s;
            for(int j=s+1; j<M; j++)
                if (A[k][0][j]<A[k][0][imin])
                {
                    imin=j;
                }
            for (int i=0; i<N; i++)
            {
                tmp=A[k][i][imin];
                A[k][i][imin]=A[k][i][s];
                A[k][i][s]=tmp;
            }
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}

clock_t SelectSort(int *A, int N) // функція сортювання вектору методом
прямого вибору
{
    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 Exchange2Sort(int \*\*\*A, int P, int N, int M) // функція сортування 3-х вимірної масиви методом прямого обміну з використанням прапорця

```

{
    int R, flag, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int k=0; k<P; k++)
    {
        R=M-1; flag=1;
        while(flag == 1)
        {
            flag=0;
            for(int j=0; j<R; j++)
            {
                if (A[k][0][j]>A[k][0][j+1])
                {
                    for (int i=0; i<N; i++)
                    {
                        tmp=A[k][i][j];
                        A[k][i][j]=A[k][i][j+1];
                        A[k][i][j+1]=tmp;
                    }
                    flag=1;
                }
            }
            R--;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}

```

clock\_t Shell\_2Sort(int \*\*\*A, int P, int N, int M) // функція сортування 3-х вимірної масиви методом сортування Шелла (спосіб реалізації на основі гібридного алгоритму «вставка-обмін»)

```

{
    int tmp, t, j, k;
    clock_t time_start, time_stop;
    if (M<4) t=1;
    else
        t=(int)log2f((float)M)-1;
    int Stages[t];

```

```

time_start = clock();

if (M<4) t=1;
else
    t=(int)log2f((float)M)-1;
Stages[t-1]=1;
for (int i=t-2; i>=0; i--)
    Stages[i]=2*Stages[i+1]+1;
for(int K=0;K<P;K++)
{
    for (int p=0; p<t; p++)
    {
        k=Stages[p];
        for (int i=k; i<M; i++)
        {
            j=i;
            while (j>=k && A[K][0][j]<A[K][0][j-k])
            {
                for (int z=0; z<N; z++)
                {
                    tmp=A[K][z][j];
                    A[K][z][j]=A[K][z][j-k];
                    A[K][z][j-k]=tmp;
                }
                j=j-k;
            }
        }
    }
}

time_stop = clock();

return time_stop - time_start;
}

```

clock\_t ExchangeSort(int \*A, int N)// функція сортування вектору методом прямого обміну з використанням прапорця

```

{
    int R, flag, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    R=N-1; flag=1;
    while(flag == 1)
    {
        flag=0;
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1])
            {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                flag=1;
            }
        R--;
    }
}

```

```

    time_stop = clock();

    return time_stop - time_start;
}

clock_t ShellSort(int *A, int N) // функція сортування вектору методом
сортування Шелла (спосіб реалізації на основі гібридного алгоритму
«вставка-обмін»)
{
    int tmp, t, j, k;
    clock_t time_start, time_stop;
    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;
    int Stages[t];

    time_start = clock();

    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;
    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
        Stages[i]=2*Stages[i+1]+1;
    for (int p=0; p<t; p++)
    {
        k=Stages[p];
        for (int i=k; i<N; i++)
        {
            j=i;
            while (j>=k && A[j]<A[j-k])
            {
                tmp=A[j];
                A[j]=A[j-k];
                A[j-k]=tmp;
                j=j-k;
            }
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}

```

## Sort.h

```

#ifndef __Sort_H__
#define __Sort_H__

clock_t Select2Sort(int ***A, int P, int N, int M);
clock_t Exchange2Sort(int ***A, int P, int N, int M);
clock_t Shell_2Sort(int ***A, int P, int N, int M);

clock_t SelectSort(int *B, int N);

```

```

clock_t ExchangeSort(int *B, int N);
clock_t ShellSort(int *B, int N);

#endif // __Sort_H__

```

## Test.c

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "Test.h"
#include "FillArray.h"
#include "OutTable.h"
#include "Menu.h"

int testfill;
int testsort;
int p;
int n;
int m;
void TestMenu()//меню вибору варіанту тестування
{
    srand(time(NULL));
    system("cls");
    printf("\nEnter size of the mas\n");
    printf("\nP = ");
    scanf("%d", &p);
    printf("\nN = ");
    scanf("%d", &n);
    printf("\nM = ");
    scanf("%d", &m);
    system("cls");
    int ***A; // оголошення динамічного тривимірного масиву
    A = (int***) malloc(p*sizeof(int**));
    for (int k = 0; k < p; k++)
    {
        A[k] = (int**) malloc(n*sizeof(int*));
        for (int i = 0; i < n; i++)
        {
            A[k][i] = (int*) malloc(m*sizeof(int));
        }
    }
    printf("Select a menu item\n");
    printf("1. Select Sort\n");
    printf("2. Exchange Sort\n");
    printf("3. Shell Sort\n");
    printf("4. Exit\n");
    while(1)
    {
        printf("\nEnter a number: ");
        scanf("%d", &testsort);
        switch(testsort)
        {
            case 1: Select(A,p,n,m);return; break;

```

```

        case 2: Exchange(A,p,n,m);return; break;
        case 3: Shell(A,p,n,m); break;
        case 4: MenuType();return; break;
        default: printf ("Wrong number.Try again\n");
    }
}

}

void Select(int ***A,int p, int n, int m)//Тестування алгоритму вибору
2
{
    system ("cls");
    printf("\n\nSelect a menu item\n");
    printf("1. Order\n");
    printf("2. Random\n");
    printf("3. BackOrder\n");
    printf("4. Exit\n");
    while(1)//Вибір заповненності масиву
    {
        printf("\nEnter a number: ");
        scanf("%d", &testfill);
        if(testfill==1)
        {
            FillArrayOrder(A,p,n,m);
            break;
        }
        else if(testfill==2)
        {
            FillArrayRandom(A,p,n,m);
            break;
        }
        else if(testfill==3)
        {
            FillArrayBackOrder(A,p,n,m);
            break;
        }
        else if (testfill==4)
            TestMenu();
        else
            printf ("Wrong number.Try again\n");
    }
    printf("\nBEFORE\n");
    Print(A,p,n,m);
    //Сортування алгоритмом вибору №2
    int imin,tmp;
    for(int k=0;k<p;k++)
    {
        for(int s=0; s<m-1; s++)
        {
            imin=s;
            for(int j=s+1; j<m; j++)
                if (A[k][0][j]<A[k][0][imin])
                {
                    imin=j;
                }
            for (int i=0;i<n;i++)
            {
                tmp=A[k][i][imin];
                A[k][i][imin]=A[k][i][s];

```



```

        A[k][i][s]=tmp;
    }
}

printf("\nAFTER\n");
Print(A,p,n,m);
    for (int k=0; k<p; k++)//звільнення пам'яті від масиву
    {
        for (int i=0; i<n; i++)
        {
            free(A[k][i]);
        }
        free(A[k]);
    }
free(A);
getch();
MenuType();
}

void Exchange(int ***A,int p, int n, int m)//Тестування алгоритму
обміну 2 з прапорцем
{
    system ("cls");
    printf("\n\nSelect a menu item\n");
    printf("1. Order\n");
    printf("2. Random\n");
    printf("3. BackOrder\n");
    printf("4. Exit\n");
    while(1)//Вибір варіанту заповнення
    {
        printf("\nEnter a number: ");
        scanf("%d", &testfill);
        if(testfill==1)
        {
            FillArrayOrder(A,p,n,m);
            break;
        }
        else if(testfill==2)
        {
            FillArrayRandom(A,p,n,m);
            break;
        }
        else if(testfill==3)
        {
            FillArrayBackOrder(A,p,n,m);
            break;
        }
        else if (testfill==4)
            TestMenu();
        else
            printf ("Wrong number.Try again\n");
    }
    printf("\nBEFORE\n");
    Print(A,p,n,m);
    //Сортування методом обміну №2 з флагом
    int R, flag, tmp;
    for(int k=0;k<p;k++)
    {

```

```

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

printf("\nAFTER\n");
Print(A,p,n,m);
for (int k=0; k<p; k++)//звільнення пам'яті від масиву
{
    for (int i=0; i<n; i++)
    {
        free(A[k][i]);
    }
    free(A[k]);
}
free(A);
getch();
MenuType();
}

void Shell(int ***A,int p, int n, int m)//Тестування алгоритму Шелла 2
{
    system("cls");
    printf("\n\nSelect a menu item\n");
    printf("1. Order\n");
    printf("2. Random\n");
    printf("3. BackOrder\n");
    printf("4. Exit\n");
    while(1)//Вибір варіанту заповнення масиву
    {
        printf("\nEnter a number: ");
        scanf("%d", &testfill);
        if(testfill==1)
        {
            FillArrayOrder(A,p,n,m);
            break;
        }
        else if(testfill==2)
        {
            FillArrayRandom(A,p,n,m);
            break;
        }
    }
}

```

```

        else if(testfill==3)
        {
            FillArrayBackOrder(A,p,n,m);
            break;
        }
        else if (testfill==4)
            TestMenu();
        else
            printf ("Wrong number.Try again\n");
    }
    printf("\nBEFORE\n");
    Print(A,p,n,m);
    //Виконання сортування Шелла №2
    int tmp, t, j, k;
    if (m<4) t=1;
    else
        t=(int)log2f((float)m)-1;
    int Stages[t];
    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
        Stages[i]=2*Stages[i+1]+1;
    for(int K=0;K<p;K++)
    {
        for (int x=0; x<t; x++)
        {
            k=Stages[x];
            for (int i=k; i<m; i++)
            {
                j=i;
                while (j>=k && A[K][0][j]<A[K][0][j-k])
                {
                    for (int z=0;z<n;z++)
                    {
                        tmp=A[K][z][j];
                        A[K][z][j]=A[K][z][j-k];
                        A[K][z][j-k]=tmp;
                    }
                    j=j-k;
                }
            }
        }
    }
    printf("\nAFTER\n");
    Print(A,p,n,m);
    for (int k=0; k<p; k++)//звільнення пам'яті від масиву
    {
        for (int i=0; i<n; i++)
        {
            free(A[k][i]);
        }
        free(A[k]);
    }
    free(A);
    getch();
    MenuType();
}

```

## Test.h

```
#ifndef __Test_H__
#define __Test_H__

void TestMenu();

void Select(int ***A, int p, int n, int m);
void Exchange(int ***A, int p, int n, int m);
void Shell(int ***A, int p, int n, int m);

#endif
```

## Menu.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include "OutTable.h"
#include "FillArray.h"
#include "Measurement.h"
#include "Sort.h"
#include "Test.h"
#include "Menu.h"

int sortnum;
int fillnum;

float Select2_Order;
float Select2_Random;
float Select2_BackOrder;
float Exchange2_Order;
float Exchange2_Random;
float Exchange2_BackOrder;
float Shell_2_Order;
float Shell_2_Random;
float Shell_2_BackOrder;

int P = 3;
int N = 12;
int M = 20000;

void Size(int ***A) //процедура зміни розміру масива
{
    system("cls");
    printf("\nCurrent size A[%d][%d][%d] \n\n", P, N, M);
    for (int k=0; k<P; k++) //звільнення пам'яті від масиву
    {
        for (int i=0; i<N; i++)
        {
            free(A[k][i]);
        }
        free(A[k]);
    }
    printf("\nEnter size of the mas\n"); //Задання розмірів нового масива
    printf("\nP = ");
    scanf("%d", &P);
}
```

```

printf("\nN = ");
scanf("%d", &N);
printf("\nM = ");
scanf("%d", &M);
system("cls");
MenuType();
}
void MenuType()//процедура для вибору користувачем типу сортування
{
    int ***Arr3D; // оголошення динамічного тривимірного масиву
    Arr3D = (int**) malloc(P*sizeof(int));
    for (int k = 0; k < P; k++)
    {
        Arr3D[k] = (int**) malloc(N*sizeof(int));
        for (int i = 0; i < N; i++)
        {
            Arr3D[k][i] = (int*) malloc(M*sizeof(int));
        }
    }
    int B[P*N*M];//Оголошення вектору
    int type;
    system ("cls");
    printf("Select a menu item\n");//Вивід пунктів меню
    printf("1. Vector\n");
    printf("2. Paralelepiped\n");
    printf("3. Algorithm check\n");
    printf("4. Size change\n");
    while(1)
    {
        printf("\nEnter a number: ");
        scanf("%d", &type);
        switch(type)//Виклик пунктів меню
        {
            case 1: MenuSort(B,P*N*M); break;
            case 2: MenuSort3D(Arr3D, P, N, M); break;
            case 3: TestMenu(); break;
            case 4: Size(Arr3D); break;
            default: printf ("Wrong number.Try again\n");
        }
    }
}

void MenuSort(int *B, int M)//процедура для вибору користувачем
алгоритму сортування вектора
{
    sortnum = 0;
    system ("cls");
    printf("Select a menu item\n");//Вивід пунктів меню
    printf("1. Select Sort\n");
    printf("2. Exchange Sort\n");
    printf("3. Shell Sort\n");
    printf("4. Packet mode\n");
    printf("5. Exit\n");
    while(1)
    {
        printf("\nEnter a number: ");
        scanf("%d", &sortnum);
        switch(sortnum)//Виклик пунктів меню

```

```

        {
            case 1: MenuFill(B, M); break;
            case 2: MenuFill(B, M); break;
            case 3: MenuFill(B, M); break;
            case 4: PackageMode(B, M); break;
            case 5: MenuType(); break;
            default: printf ("Wrong number.Try again\n");
        }
    }
}

```

**void MenuSort3D(int \*\*\*A, int P, int N, int M)** //процедура для вибору користувачем алгоритму сортування для паралелепіпеда

```

{
    sortnum = 0;
    system ("cls");
    printf("Select a menu item\n");//Вивід пунктів меню
    printf("1. Select Sort\n");
    printf("2. Exchange Sort\n");
    printf("3. Shell Sort\n");
    printf("4. Packet mode\n");
    printf("5. Exit\n");
    while(1)
    {
        printf("\nEnter a number: ");
        scanf("%d", &sortnum);
        switch(sortnum)//Виклик пунктів меню
        {
            case 1: MenuFill3D(A, P, N, M); break;
            case 2: MenuFill3D(A, P, N, M); break;
            case 3: MenuFill3D(A, P, N, M); break;
            case 4: PackageMode3D(A, P, N, M); break;
            case 5: MenuType(); break;
            default: printf ("Wrong number.Try again\n");
        }
    }
}

```

**void MenuFill(int \*B, int M)** //процедура для вибору користувачем того що він хоче зробити

```

{
    system ("cls");
    fillnum = 0;
    printf("Select a menu item\n");//Вивід пунктів меню
    printf("1. Ordered\n");
    printf("2. Random\n");
    printf("3. BackOrdered\n");
    printf("4. Exit\n");
    while(1)
    {
        printf("\nEnter a number: ");
        scanf("%d", &fillnum);
        switch(fillnum)//Виклик пунктів меню
        {
            case 1: SoloMode(B, M); break;
            case 2: SoloMode(B, M); break;
            case 3: SoloMode(B, M); break;
            case 4: MenuSort(B, M); break;

```

```

        default: printf ("Wrong number.Try again\n");
    }
}

}

void MenuFill3D(int ***A, int P, int N, int M) //процедура для вибору
користувачем методу заповнення масиву
{
    system ("cls");
    fillnum = 0;
    printf("Select a menu item\n");//Вивід пунктів меню
    printf("1. Ordered\n");
    printf("2. Random\n");
    printf("3. BackOrdered\n");
    printf("4. Exit\n");
    while(1)
    {
        printf("\nEnter a number: ");
        scanf("%d", &fillnum);
        switch(fillnum) //Виклик пунктів меню
        {
            case 1: SoloMode3D(A, P, N, M); break;
            case 2: SoloMode3D(A, P, N, M); break;
            case 3: SoloMode3D(A, P, N, M); break;
            case 4: MenuSort3D(A, P, N, M); break;
            default: printf ("Wrong number.Try again\n");
        }
    }
}

void PackageMode(int *B, int M) //процедура для запуску пакетного режиму
вимірювання часу роботи алгоритмів сортування векторів
{

    srand(time(NULL));
    char button;

    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування впорядкованого вектора методом вибору
    {
        FillVectorOrder(B, M);
        Res[i] = SelectSort(B, M);
    }
    Select2_Order = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування рандомного вектора методом вибору
    {
        FillVectorRandom(B, M);
        Res[i] = SelectSort(B, M);
    }
    Select2_Random = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування обернено впорядкованого вектора
    методом вибору
    {
        FillVectorBackOrder(B, M);
        Res[i] = SelectSort(B, M);
    }
}

```

```

Select2_BackOrder = MeasurementProcessing();

    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування впорядкованого вектора методом обміну
    {
        FillVectorOrder(B, M);
        Res[i] = ExchangeSort(B, M);
    }
    Exchange2_Order = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування випадкового вектора методом обміну
    {
        FillVectorRandom(B, M);
        Res[i] = ExchangeSort(B, M);
    }
    Exchange2_Random = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування обернено впорядкованого вектора
методом обміну
    {
        FillVectorBackOrder(B, M);
        Res[i] = ExchangeSort(B, M);
    }
    Exchange2_BackOrder = MeasurementProcessing();

    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування впорядкованого вектора методом Шелла
    {
        FillVectorOrder(B, M);
        Res[i] = ShellSort(B, M);
    }
    Shell_2_Order = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування випадкового вектора методом Шелла
    {
        FillVectorRandom(B, M);
        Res[i] = ShellSort(B, M);
    }
    Shell_2_Random = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування обернено впорядкованого вектора
методом Шелла
    {
        FillVectorBackOrder(B, M);
        Res[i] = ShellSort(B, M);
    }
    Shell_2_BackOrder = MeasurementProcessing();

    system ("cls");
    OutTable(M, Select2_Order, Select2_Random, Select2_BackOrder,
Exchange2_Order, Exchange2_Random, Exchange2_BackOrder, Shell_2_Order,
Shell_2_Random, Shell_2_BackOrder); //функція побудови таблиці
результатів вимірювань

```



```

printf("\nPress Space to go to main menu");
while(1) // цикл для виходу користувача у головне меню
{
    char button = getch();
    if(button == ' ')
    {
        MenuType();
    }
}

void PackageMode3D(int ***A, int P, int N, int M) //процедура для
запуску пакетного режиму вимірювання часу роботи алгоритмів сортування
тривимірної масиви
{
    srand(time(NULL));
    char button;

    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування впорядкованого масиву методом вибору
    {
        FillArrayOrder(A, P, N, M);
        Res[i] = Select2Sort(A, P, N, M);
    }
    Select2_Order = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування рандомного масиву методом вибору
    {
        FillArrayRandom(A, P, N, M);
        Res[i] = Select2Sort(A, P, N, M);
    }
    Select2_Random = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування обернено впорядкованого масиву методом
    вибору
    {
        FillArrayBackOrder(A, P, N, M);
        Res[i] = Select2Sort(A, P, N, M);
    }
    Select2_BackOrder = MeasurementProcessing();

    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування впорядкованого масиву методом обміну
    {
        FillArrayOrder(A, P, N, M);
        Res[i] = Exchange2Sort(A, P, N, M);
    }
    Exchange2_Order = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
    часу роботи алгоритму сортування рандомного масиву методом обміну
    {
        FillArrayRandom(A, P, N, M);
        Res[i] = Exchange2Sort(A, P, N, M);
    }
    Exchange2_Random = MeasurementProcessing();
}

```

```

    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування обернено впорядкованого масиву методом
обміну
    {
        FillArrayBackOrder(A, P, N, M);
        Res[i] = Exchange2Sort(A, P, N, M);
    }
    Exchange2_BackOrder = MeasurementProcessing();

    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування впорядкованого масиву методом Шелла
    {
        FillArrayOrder(A, P, N, M);
        Res[i] = Shell_2Sort(A, P, N, M);
    }
    Shell_2_Order = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++)//Запуск вимірювання
часу роботи алгоритму сортування випадкового масиву методом Шелла
    {
        FillArrayRandom(A, P, N, M);
        Res[i] = Shell_2Sort(A, P, N, M);
    }
    Shell_2_Random = MeasurementProcessing();
    for (int i = 0; i < measurements_number; i++) //Запуск вимірювання
часу роботи алгоритму сортування обернено впорядкованого масиву
методом Шелла
    {
        FillArrayBackOrder(A, P, N, M);
        Res[i] = Shell_2Sort(A, P, N, M);
    }
    Shell_2_BackOrder = MeasurementProcessing();

    system("cls");
    OutTable3D(P,N,M,Select2_Order, Select2_Random, Select2_BackOrder,
Exchange2_Order, Exchange2_Random, Exchange2_BackOrder, Shell_2_Order,
Shell_2_Random, Shell_2_BackOrder); //функція побудови таблиці
результатів вимірювань
    printf("\nPress Space to go to main menu");
    for (int k=0; k<P; k++)//звільнення пам'яті від масиву
    {
        for (int i=0; i<N; i++)
        {
            free(A[k][i]);
        }
        free(A[k]);
    }
    while(1) // цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}

```

```

void SoloMode(int *B, int M)//процедура для запуску вибраного режиму
вимірювання часу роботи алгоритмів для вектору
{

    srand(time(NULL));

    float SoloTime;
    char button;
    if(sortnum == 1)//перевірка який саме алгоритм вибрав користувач(у
данному випадку алгоритм сортування вибору)
    {
        if(fillnum == 1)//перевірка як саме користувач хоче заповнити
масив(у данному випадку впорядковано)
        {
            for (int i = 0; i < measurements_number; i++)
            {
                FillVectorOrder(B, M);
                Res[i] = SelectSort(B, M);
            }
            SoloTime = MeasurementProcessing();
            system("cls");
            printf("Time: %f \n", SoloTime);
            printf("\nPress Space to go to main menu");
            while(1)// цикл для виходу користувача у головне меню
            {
                char button = getch();
                if(button == ' ')
                {
                    MenuType();
                }
            }
        }
        else if(fillnum == 2)//перевірка як саме користувач хоче
заповнити масив(у данному випадку рандомно)
        {
            for (int i = 0; i < measurements_number; i++)
            {
                FillVectorRandom(B, M);
                Res[i] = SelectSort(B, M);
            }
            SoloTime = MeasurementProcessing();
            system("cls");
            printf("Time: %f \n", SoloTime);
            printf("\nPress Space to go to main menu");
            while(1)// цикл для виходу користувача у головне меню
            {
                char button = getch();
                if(button == ' ')
                {
                    MenuType();
                }
            }
        }
        else if(fillnum == 3)//перевірка як саме користувач хоче
заповнити масив(у данному випадку обернено впорядковано)
        {
            for (int i = 0; i < measurements_number; i++)
            {

```

```

        FillVectorBackOrder(B, M);
        Res[i] = SelectSort(B, M);
    }
    SoloTime = MeasurementProcessing();
    system("cls");
    printf("Time: %f \n", SoloTime);
    printf("\nPress Space to go to main menu");
    while(1) // цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}

else if(sortnum == 2) //перевірка який саме алгоритм вибрав
користувач(у данному випадку алгоритм сортування Обміну)
{
    if(fillnum == 1) //перевірка як саме користувач хоче заповнити
масив(у данному випадку впорядковано)
    {
        for (int i = 0; i < measurements_number; i++)
        {
            FillVectorOrder(B, M);
            Res[i] = ExchangeSort(B, M);
        }
        SoloTime = MeasurementProcessing();
        system("cls");
        printf("Time: %f \n", SoloTime);
        printf("\nPress Space to go to main menu");
        while(1) // цикл для виходу користувача у головне меню
        {
            char button = getch();
            if(button == ' ')
            {
                MenuType();
            }
        }
    }

    else if(fillnum == 2) //перевірка як саме користувач хоче
заповнити масив(у данному випадку рандомно)
    {
        for (int i = 0; i < measurements_number; i++)
        {
            FillVectorRandom(B, M);
            Res[i] = ExchangeSort(B, M);
        }
        SoloTime = MeasurementProcessing();
        system("cls");
        printf("Time: %f \n", SoloTime);
        printf("\nPress Space to go to main menu");
        while(1) // цикл для виходу користувача у головне меню
        {
            char button = getch();
            if(button == ' ')
            {
                MenuType();
            }
        }
    }
}

```

```

    }
}
else if(fillnum == 3)//перевірка як саме користувач хоче
заповнити масив(у данному випадку обернено впорядковано)
{
    for (int i = 0; i < measurements_number; i++)
    {
        FillVectorBackOrder(B, M);
        Res[i] = ExchangeSort(B, M);
    }
    SoloTime = MeasurementProcessing();
    system("cls");
    printf("Time: %f \n", SoloTime);
    printf("\nPress Space to go to main menu");
    while(1)// цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}
else if(sortnum == 3)//перевірка який саме алгоритм вибрав
користувач(у данному випадку алгоритм сортування Шелла)
{
    if(fillnum == 1)//перевірка як саме користувач хоче заповнити
масив(у данному випадку впорядковано)
    {
        for (int i = 0; i < measurements_number; i++)
        {
            FillVectorOrder(B, M);
            Res[i] = ShellSort(B, M);
        }
        SoloTime = MeasurementProcessing();
        system("cls");
        printf("Time: %f \n", SoloTime);
        printf("\nPress Space to go to main menu");
        while(1)// цикл для виходу користувача у головне меню
        {
            char button = getch();
            if(button == ' ')
            {
                MenuType();
            }
        }
    }
    else if(fillnum == 2)//перевірка як саме користувач хоче
заповнити масив(у данному випадку рандомно)
    {
        for (int i = 0; i < measurements_number; i++)
        {
            FillVectorRandom(B, M);
            Res[i] = ShellSort(B, M);
        }
        SoloTime = MeasurementProcessing();
        system("cls");
    }
}

```

```

printf("Time: %f \n", SoloTime);
printf("\nPress Space to go to main menu");
while(1) // цикл для виходу користувача у головне меню
{
    char button = getch();
    if(button == ' ')
    {
        MenuType();
    }
}
}
else if(fillnum == 3) //перевірка як саме користувач хоче
заповнити масив (у данному випадку обернено впорядковано)
{
    for (int i = 0; i < measurements_number; i++)
    {
        FillVectorBackOrder(B, M);
        Res[i] = ShellSort(B, M);
    }
    SoloTime = MeasurementProcessing();
    system("cls");
    printf("Time: %f \n", SoloTime);
    printf("\nPress Space to go to main menu");
    while(1) // цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}
}

void SoloMode3D(int ***A, int P, int N, int M) //процедура для запуску
вибраного режиму вимірювання часу роботи алгоритмів
{
    srand(time(NULL));

    float SoloTime;
    char button;
    if(sortnum == 1) //перевірка який саме алгоритм вибрав користувач (у
данному випадку алгоритм сортування вибору)
    {
        if(fillnum == 1) //перевірка як саме користувач хоче заповнити
масив (у данному випадку впорядковано)
        {
            for (int i = 0; i < measurements_number; i++)
            {
                FillArrayOrder(A, P, N, M);
                Res[i] = Select2Sort(A, P, N, M);
            }
            SoloTime = MeasurementProcessing();
            system("cls");
            printf("Time: %f \n", SoloTime);
            printf("\nPress Space to go to main menu");
            while(1) // цикл для виходу користувача у головне меню

```

```

        {
            char button = getch();
            if(button == ' ')
            {
                MenuType();
            }
        }
    }
    else if(fillnum == 2)//перевірка як саме користувач хоче
заповнити масив(у данному випадку рандомно)
    {
        for (int i = 0; i < measurements_number; i++)
        {
            FillArrayRandom(A, P, N, M);
            Res[i] = Select2Sort(A, P, N, M);
        }
        SoloTime = MeasurementProcessing();
        system("cls");
        printf("Time: %f \n", SoloTime);
        printf("\nPress Space to go to main menu");
        while(1)// цикл для виходу користувача у головне меню
        {
            char button = getch();
            if(button == ' ')
            {
                MenuType();
            }
        }
    }
    else if(fillnum == 3)//перевірка як саме користувач хоче
заповнити масив(у данному випадку обернено впорядковано)
    {
        for (int i = 0; i < measurements_number; i++)
        {
            FillArrayBackOrder(A, P, N, M);
            Res[i] = Select2Sort(A, P, N, M);
        }
        SoloTime = MeasurementProcessing();
        system("cls");
        printf("Time: %f \n", SoloTime);
        printf("\nPress Space to go to main menu");
        while(1)// цикл для виходу користувача у головне меню
        {
            char button = getch();
            if(button == ' ')
            {
                MenuType();
            }
        }
    }
}
else if(sortnum == 2)//перевірка який саме алгоритм вибрав
користувач(у данному випадку алгоритм сортування обміну)
{
    if(fillnum == 1)//перевірка як саме користувач хоче заповнити
масив(у данному випадку впорядковано)
    {
        for (int i = 0; i < measurements_number; i++)
        {

```

```

        FillArrayOrder(A, P, N, M);
        Res[i] = Exchange2Sort(A, P, N, M);
    }
    SoloTime = MeasurementProcessing();
    system("cls");
    printf("Time: %f \n", SoloTime);
    printf("\nPress Space to go to main menu");
    while(1) // цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}
else if(fillnum == 2) //перевірка як саме користувач хоче
заповнити масив (у данному випадку рандомно)
{
    for (int i = 0; i < measurements_number; i++)
    {
        FillArrayRandom(A, P, N, M);
        Res[i] = Exchange2Sort(A, P, N, M);
    }
    SoloTime = MeasurementProcessing();
    system("cls");
    printf("Time: %f \n", SoloTime);
    printf("\nPress Space to go to main menu");
    while(1) // цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}
else if(fillnum == 3) //перевірка як саме користувач хоче
заповнити масив (у данному випадку обернено впорядковано)
{
    for (int i = 0; i < measurements_number; i++)
    {
        FillArrayBackOrder(A, P, N, M);
        Res[i] = Exchange2Sort(A, P, N, M);
    }
    SoloTime = MeasurementProcessing();
    system("cls");
    printf("Time: %f \n", SoloTime);
    printf("\nPress Space to go to main menu");
    while(1) // цикл для виходу користувача у головне меню
    {
        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}
}
}

```



```

    else if(sortnum == 3)//перевірка який саме алгоритм вибрав
користувач(у данному випадку алгоритм сортування Шелла)
    {
        if(fillnum == 1)//перевірка як саме користувач хоче заповнити
масив(у данному випадку впорядковано)
        {
            for (int i = 0; i < measurements_number; i++)
            {
                FillArrayOrder(A, P, N, M);
                Res[i] = Shell_2Sort(A, P, N, M);
            }
            SoloTime = MeasurementProcessing();
            system("cls");
            printf("Time: %f \n", SoloTime);
            printf("\nPress Space to go to main menu");
            while(1)// цикл для виходу користувача у головне меню
            {
                char button = getch();
                if(button == ' ')
                {
                    MenuType();
                }
            }
        }
        else if(fillnum == 2)//перевірка як саме користувач хоче
заповнити масив(у данному випадку рандомно)
        {
            for (int i = 0; i < measurements_number; i++)
            {
                FillArrayRandom(A, P, N, M);
                Res[i] = Shell_2Sort(A, P, N, M);
            }
            SoloTime = MeasurementProcessing();
            system("cls");
            printf("Time: %f \n", SoloTime);
            printf("\nPress Space to go to main menu");
            while(1)// цикл для виходу користувача у головне меню
            {
                char button = getch();
                if(button == ' ')
                {
                    MenuType();
                }
            }
        }
        else if(fillnum == 3)//перевірка як саме користувач хоче
заповнити масив(у данному випадку обернено впорядковано)
        {
            for (int i = 0; i < measurements_number; i++)
            {
                FillArrayBackOrder(A, P, N, M);
                Res[i] = Shell_2Sort(A, P, N, M);
            }
            SoloTime = MeasurementProcessing();
            system("cls");
            printf("Time: %f \n", SoloTime);
            printf("\nPress Space to go to main menu");
            while(1)// цикл для виходу користувача у головне меню
            {

```

```

        char button = getch();
        if(button == ' ')
        {
            MenuType();
        }
    }
}

for (int k=0; k<P; k++)//звільнення пам'яті від масиву
{
    for (int i=0; i<N; i++)
    {
        free(A[k][i]);
    }
    free(A[k]);
}
}

```

## Menu.h

```

#ifndef __Menu_H__
#define __Menu_H__

void Size(int ***A);
void MenuType();
void MenuSort3D(int ***A, int P, int N, int M);
void MenuSort(int *B, int M);
void MenuFill(int *B, int M);
void MenuFill3D(int ***A, int P, int N, int M);
void PackageMode(int *B, int M);
void PackageMode3D(int ***A, int P, int N, int M);
void SoloMode(int *B, int M);
void SoloMode3D(int ***A, int P, int N, int M);
#endif

```

# Тести програми

## 1.Тести сортування масиву

### 1.1 Для трьохвимірного масиву

#### А)Сортування методом вибору №2

Впорядкований масив :

BEFORE					
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
AFTER					
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Невпорядкований масив :

BEFORE					
14	11	0	38	56	11
30	56	48	44	33	48
6	15	47	19	17	18
59	37	41	35	16	36
56	44	54	36	48	48
47	42	30	20	3	14
46	24	39	6	41	27
43	0	42	42	54	39
14	8	50	41	26	49
36	30	18	20	33	40
AFTER					
0	11	11	14	38	56
48	56	48	30	44	33
47	15	18	6	19	17
41	37	36	59	35	16
54	44	48	56	36	48
3	14	20	30	42	47
41	27	6	39	24	46
54	39	42	42	0	43
26	49	41	50	8	14
33	40	20	18	30	36

Обернено впорядкований

BEFORE					
60	59	58	57	56	55
54	53	52	51	50	49
48	47	46	45	44	43
42	41	40	39	38	37
36	35	34	33	32	31
30	29	28	27	26	25
24	23	22	21	20	19
18	17	16	15	14	13
12	11	10	9	8	7
6	5	4	3	2	1
AFTER					
55	56	57	58	59	60
49	50	51	52	53	54
43	44	45	46	47	48
37	38	39	40	41	42
31	32	33	34	35	36
25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6

**Б)Сортування методом обміну №2(з використанням флажка)**

Впорядкований масив :

BEFORE					
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
AFTER					
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Невпорядкований масив :

BEFORE					
4	26	10	28	43	34
46	7	57	7	26	6
31	6	58	39	16	21
27	17	46	29	24	20
35	26	10	7	40	8
36	26	57	48	0	38
25	2	7	23	40	43
26	34	22	58	30	52
49	47	13	21	1	45
35	51	19	53	40	33
AFTER					
4	10	26	28	34	43
46	57	7	7	6	26
31	58	6	39	21	16
27	46	17	29	20	24
35	10	26	7	8	40
0	26	36	38	48	57
40	2	25	43	23	7
30	34	26	52	58	22
1	47	49	45	21	13
40	51	35	33	53	19

# Обернено впорядкований

BEFORE					
60	59	58	57	56	55
54	53	52	51	50	49
48	47	46	45	44	43
42	41	40	39	38	37
36	35	34	33	32	31
30	29	28	27	26	25
24	23	22	21	20	19
18	17	16	15	14	13
12	11	10	9	8	7
6	5	4	3	2	1
AFTER					
55	56	57	58	59	60
49	50	51	52	53	54
43	44	45	46	47	48
37	38	39	40	41	42
31	32	33	34	35	36
25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6

## В)Сортування методом сортування Шелла №2

Впорядкований масив :

BEFORE					
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
AFTER					
0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

Невпорядкований масив :

BEFORE					
2	20	39	8	35	28
45	1	6	58	51	16
39	15	36	52	11	8
4	41	54	49	5	47
58	16	39	48	15	48
15	37	44	48	44	6
45	6	21	42	5	42
58	10	30	51	37	0
3	55	32	23	41	10
40	11	5	51	7	55
AFTER					
2	8	20	28	35	39
45	58	1	16	51	6
39	52	15	8	11	36
4	49	41	47	5	54
58	48	16	48	15	39
6	15	37	44	44	48
42	45	6	21	5	42
0	58	10	30	37	51
10	3	55	32	41	23
55	40	11	5	7	51

## Обернено впорядкований

BEFORE					
60	59	58	57	56	55
54	53	52	51	50	49
48	47	46	45	44	43
42	41	40	39	38	37
36	35	34	33	32	31
30	29	28	27	26	25
24	23	22	21	20	19
18	17	16	15	14	13
12	11	10	9	8	7
6	5	4	3	2	1
AFTER					
55	56	57	58	59	60
49	50	51	52	53	54
43	44	45	46	47	48
37	38	39	40	41	42
31	32	33	34	35	36
25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6



## 2.Таблиці часу сортування

### 2.1 Залежність часу роботи алгоритмів від довжини стовпчиків масива

A[3][1][12000]			
	Order	Random	Backorder
SelectSort	695.40	723.65	809.05
ExchangeSort	0.05	2365.85	2604.85
ShellSort	2.10	13.30	4.30

A[3][2][12000]			
	Order	Random	Backorder
SelectSort	737.10	738.90	746.10
ExchangeSort	0.00	3340.25	4034.90
ShellSort	1.80	18.25	6.15

A[3][4][12000]			
	Order	Random	Backorder
SelectSort	699.90	765.00	809.55
ExchangeSort	0.05	5321.85	8825.75
ShellSort	2.40	27.10	9.15

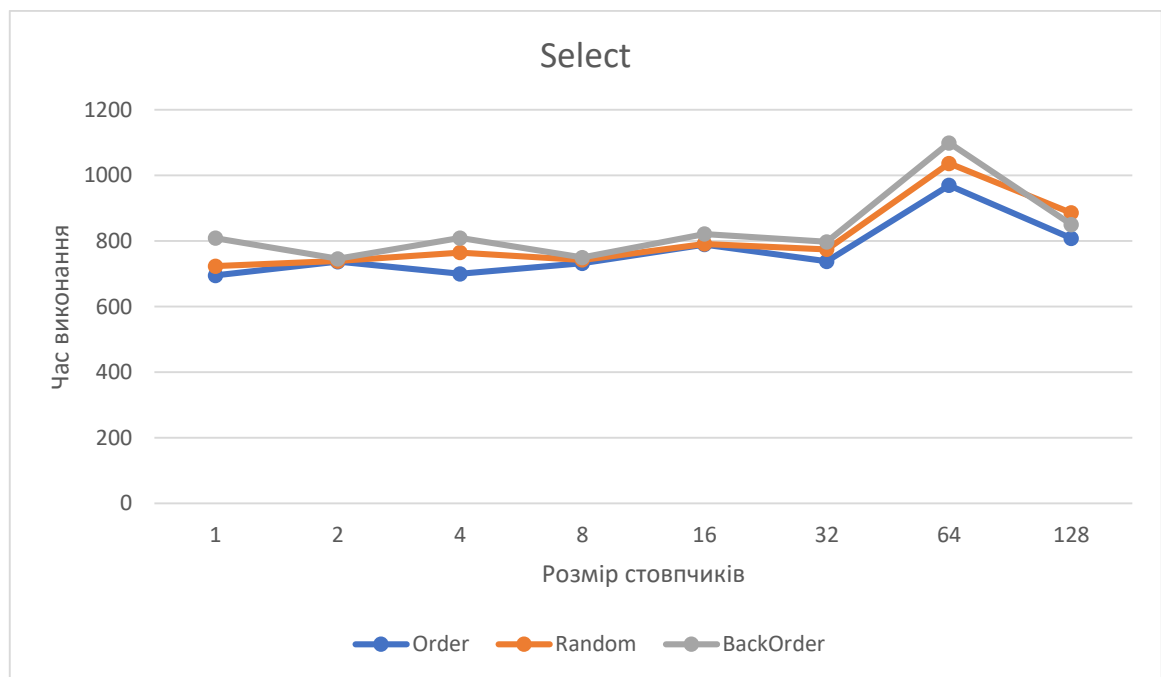
A[3][8][12000]			
	Order	Random	Backorder
SelectSort	732.10	742.45	749.95
ExchangeSort	0.10	8595.65	14108.35
ShellSort	2.00	37.60	13.00

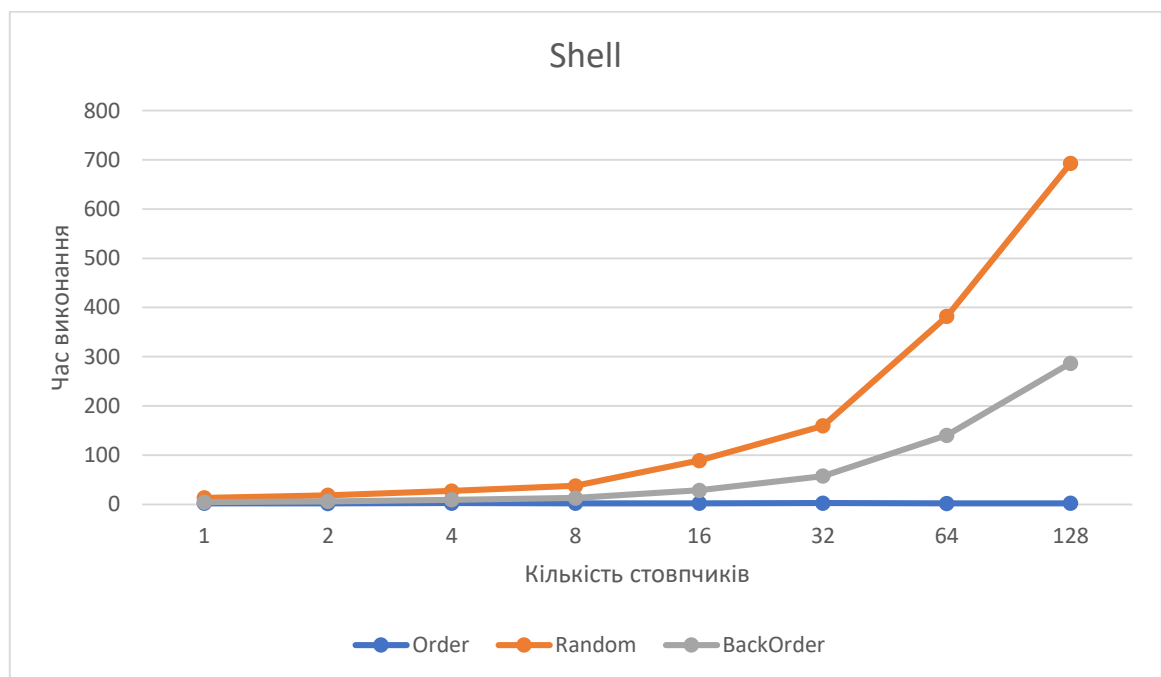
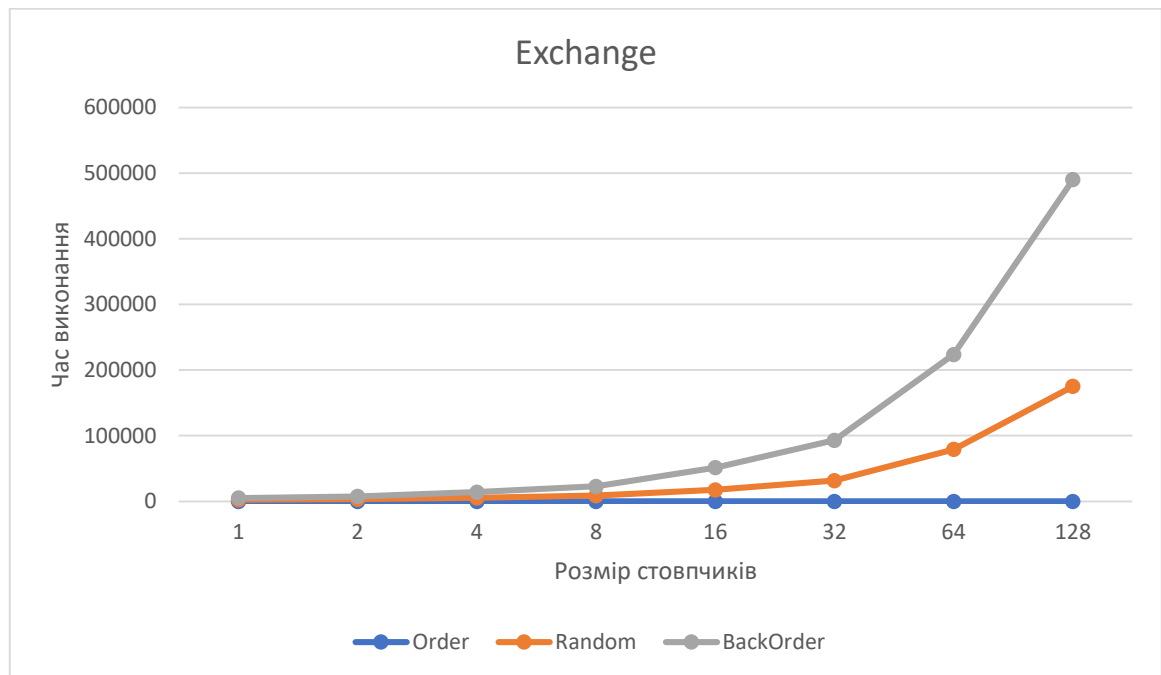
A[3][16][12000]			
	Order	Random	Backorder
SelectSort	789.40	791.30	821.55
ExchangeSort	0.10	17370.00	34004.90
ShellSort	2.00	88.85	28.35

A[3][32][12000]			
	Order	Random	Backorder
SelectSort	738.35	774.75	797.85
ExchangeSort	0.05	31600.70	61391.65
ShellSort	2.35	159.50	57.30

A[3][64][12000]			
	Order	Random	Backorder
SelectSort	969.60	1036.60	1099.25
ExchangeSort	0.00	79052.30	144763.41
ShellSort	1.80	381.80	140.45

A[3][128][12000]			
	Order	Random	Backorder
SelectSort	808.55	885.80	850.40
ExchangeSort	0.15	175281.66	314958.81
ShellSort	2.05	692.65	286.70





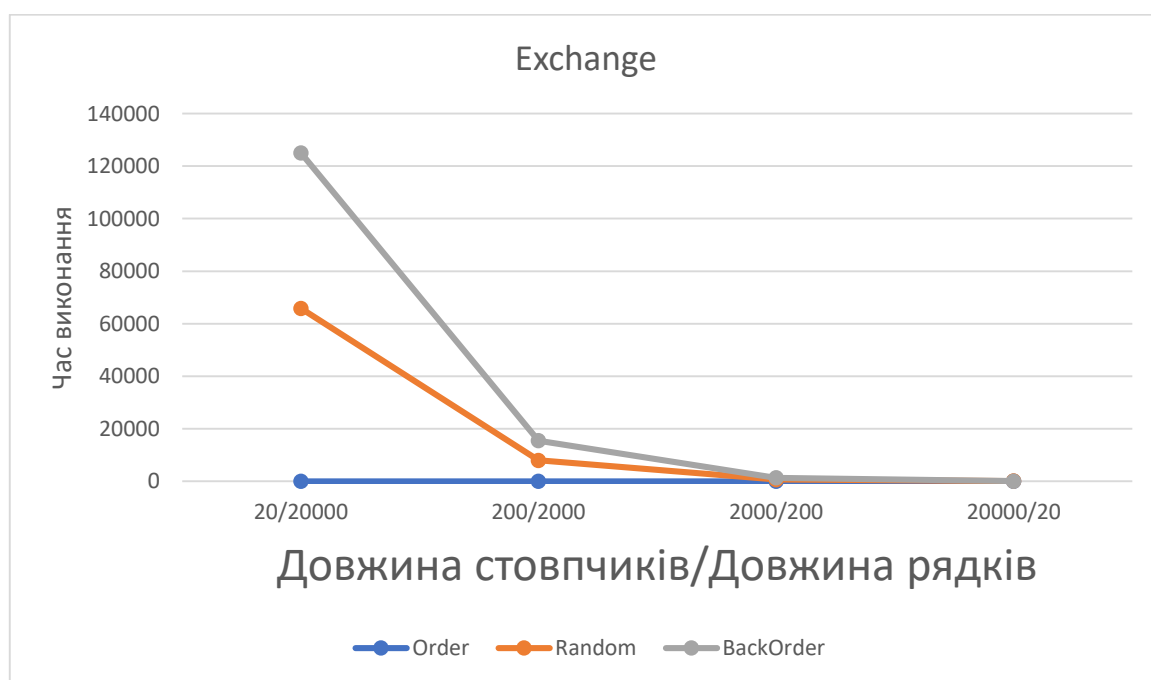
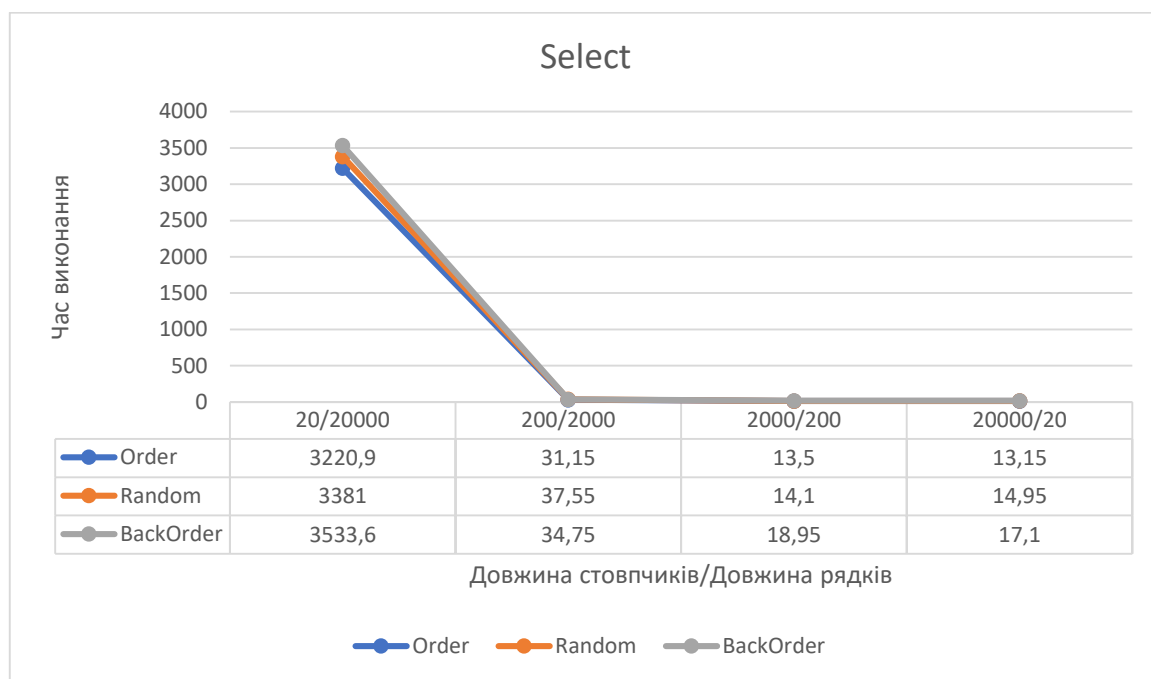
## 2.2 Залежність часу роботи алгоритмів від форми перерізів масива

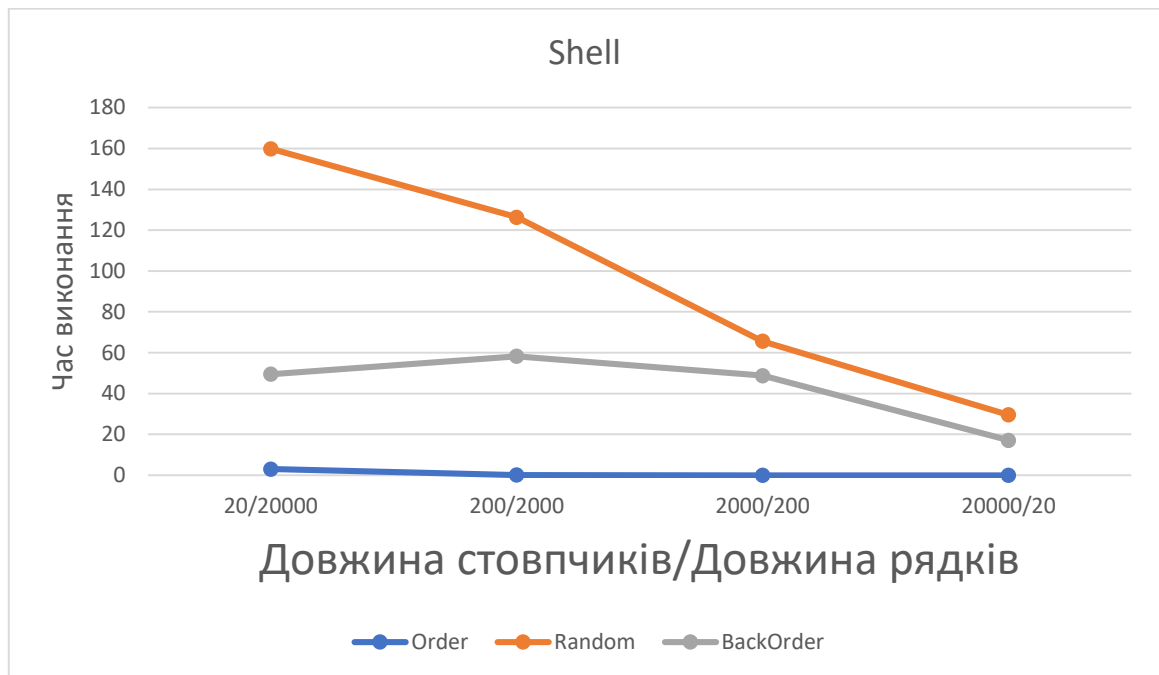
A[3][20][20000]			
	Order	Random	Backorder
SelectSort	3220.90	3381.00	3533.60
ExchangeSort	0.35	65865.20	125088.75
ShellSort	3.00	159.85	49.45

A[3][200][2000]			
	Order	Random	Backorder
SelectSort	31.15	37.55	34.75
ExchangeSort	0.00	7939.90	15486.80
ShellSort	0.15	126.35	58.25

A[3][2000][200]			
	Order	Random	Backorder
SelectSort	13.50	14.10	18.95
ExchangeSort	0.00	665.95	1335.00
ShellSort	0.00	65.60	48.75

A[3][20000][20]			
	Order	Random	Backorder
SelectSort	13.15	14.95	17.10
ExchangeSort	0.00	79.40	142.40
ShellSort	0.00	29.60	32.05





## 2.3 Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масива при однаковій загальній кількості ключів у всьому масиві

A[10][20][10000]

	Order	Random	Backorder
SelectSort	1982.70	2012.35	2026.50
ExchangeSort	0.10	51657.25	95744.20
ShellSort	6.05	272.00	83.75

A[100][20][1000]

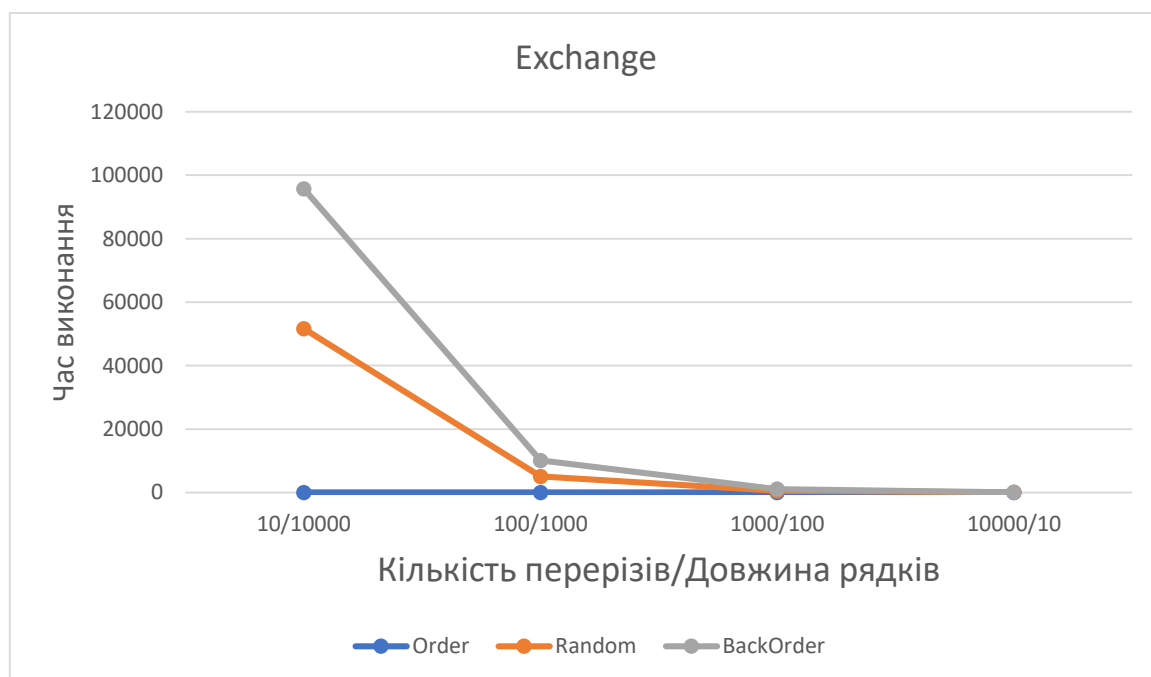
	Order	Random	Backorder
SelectSort	212.10	226.70	233.35
ExchangeSort	0.50	5061.30	10080.90
ShellSort	4.90	155.10	82.15

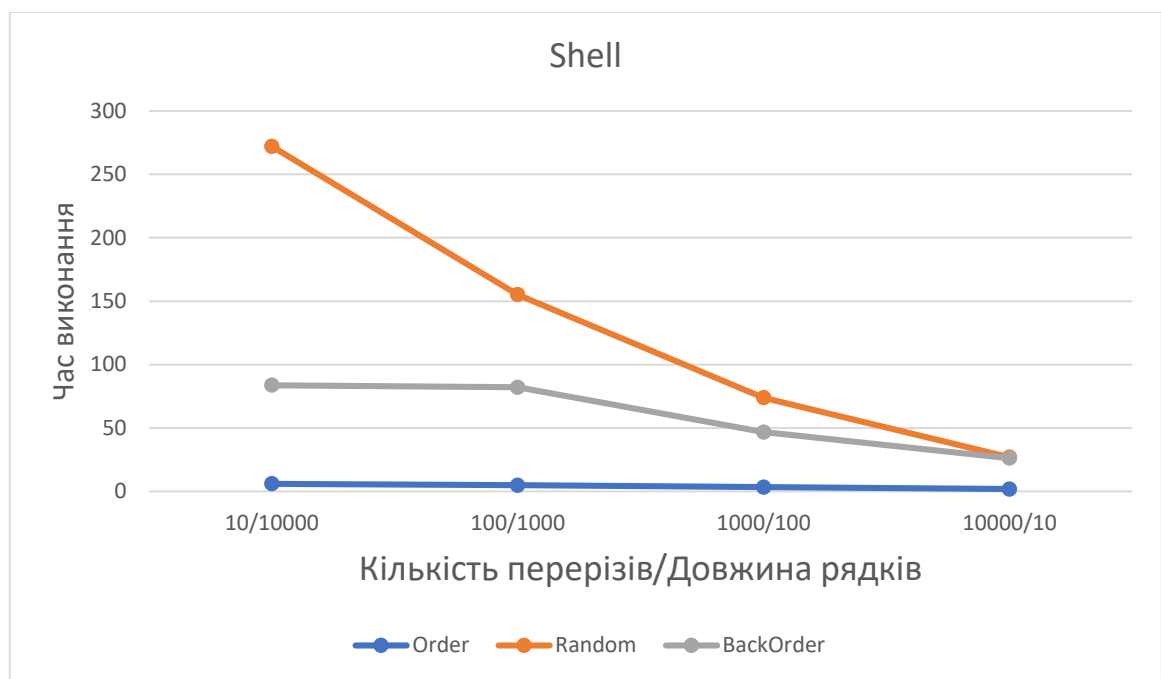
A[1000][20][100]

	Order	Random	Backorder
SelectSort	36.50	44.10	47.65
ExchangeSort	0.55	507.90	1066.45
ShellSort	2.30	73.95	46.75

A[10000][20][10]

	Order	Random	Backorder
SelectSort	17.60	21.00	21.90
ExchangeSort	1.80	44.45	89.90
ShellSort	1.80	28.80	31.55







## 2.4Для вектора

A[150000]

	Order	Random	Backorder
SelectSort	29850.05	29873.30	28827.05
ExchangeSort	0.50	75558.80	48539.45
ShellSort	6.95	51.50	11.10





## Порівняльний аналіз отриманих результатів

Метод прямого вибору №2 : Даний метод показав дивні результати як в трьохвимірному масиві так в одновимірному, але у середньому можна сказати що прямий вибір №2 сортує всі випадки відсортованості приблизно однаково, це зумовлене тим що різниця між відсортованим і невідсортованим масивом у швидкодії лише у присвоєнні одного значення, бо цей алгоритм переміщує свої елементи перед новою ітерацією навіть якщо мінімальне було знайдене з самого початку, через таку структуру алгоритму сортування робить його майже незалежним від варіанту заповнення масиву, бо при всіх варіантах заповненості будуть переставлятися значення, тому швидкість буде схожа. Також можна порівняти мої результати з результатами Ніклауса Вірта, викладеними на лекції. При порівнянні на векторі мої результати відрізняються від його бо обернено впорядкований масив виявився швидшим за інші, хоча в теорії (бо саме у випадку обернено відсортованого масива з кожним новим елементом буде знаходитися нове мінімальне та переприсвоюватись мініимальному індексу новий індекс, що збільшує час виконання) і у Вірта вишло інакше, також у трьохвимірному масиві та векторі інколи Обернено впорядкований сортувався швидше за випадковий, я думаю це зумовлено незначною відмінністю у алгоритмі виконання при знаходженні мініимального

числа та ,можливо, особливостями компілятора або втручання у процес.

Метод прямого обміну №2(з використанням прапорця) : Найкращим випадком для цього методу є впорядкований масив, сортування якого закінчується після першого проходу завдяки прапорцю що показує що перестановок невідбулось, отже масив відсортований. Є найшвидшим алгоритмом при впорядкованому заповненні із всіх що порівнюється у моєму варіанті. У випадку випадкового або обернено впорядкованості масива видає найповільніші значення(Обернено впорядкований набагато повільніше сортується ніж випадковий), така низька швидкість зумовлена тим , якщо наступний елемент менший за поточний елемент то вони міняються місцями, що нераціонально використовує ресурси комп'ютера бо , наприклад , для обернено впорядкованого варіанту заповненості потрібно буде перенести перший елемент на місце останнього, при цьому з постійною зміною елементів на шляху до кінця, а не як у методі вибору №2 відразу, нечіпаючи інших елементів. І чим більше масив тим більшу кількість елементів він буде змінювати місцями, а у моєму варіанті завдання при кожній зміні елементів місцями переставляються і всі стовпчики, що ще тільки збільшує час виконання багатократно. Щодо дивних результатів тесту для вектора це пояснюється особливостями компілятора C, який вирішив сам покращити, в іншому компіляторі значення обернено впорядкованого будуть набагато більше за випадковий,

Метод Шелла №2 : Найшвидший алгоритм із тих що я досліджував. Цей алгоритм є так званим покращеним алгоритмом, принцип його дії заключається на зменшенні відстані сортування через розподілення масиву на групи , що сортуються кожна окремо, поступово поєднуючись. Метод Шелла 2 це покращений метод вставки. Цей алгоритм другий по швидкості при впорядкованості та найшвидший в усіх інших випадках. Швидкість виконання при впорядкованості цього алгоритму лежить у тому що при перевірці елементів в групі виконується лише присвоєння та перевірка для кожної групи, що для мого варіанту задачі при перестанові цілих стовпчиків ставить його на голову швидше за інші мною розглянуті алгоритми тому він набагато швидше за той же вибір що переприсвоював сам собі значення, але ж

ненастільки швидкий як при обміні з прапорцем який відразу розуміє що масив відсортований. Тепер давайте розглянемо обернено впорядкований та випадковий варіанти заповненості, мої тести показали що обернено впорядкований при великих розмірах показував багатократно більшу швидкість виконання ніж випадковий, і це є так насправді, це зумовлено тим що для обернено впорядкового масиву легше відсортовувати об'єднані групи, бо значення, що потрібно замінити щоб відсортувати новостворену групу знаходяться близько, що негарнатується випадковою заповненістю через що алгоритм вимушений довше шукати місце вставки що витрачає більше часу на сортування.

## **Порівняльний аналіз випадків дослідження**

### **Випадок дослідження 1. Залежність часу роботи алгоритмів від довжини стовпчиків масива**

Зміна довжини стовпчиків масиву збільшило час виконання сортування методом вибору №2 при будь-якому заповненні, це пов'язано з тим що незважаючи на поточну заповненість алгоритм один раз переставить елементи, а разом з ним і стовпчики (тут зрозуміло що чим довше стовпчик тим довше буде процес перестановки).

Зміна довжини стовпчиків найбільше впливає на алгоритм обміну №2 з використання прапорця через його постійні зміни місцями сусідніх елементів, що при збільшенні кількості стовпчиків займає ще більше часу. Але збільшення кількості стовпчиків ніяк не впливає на цей алгоритм при впорядкованому заповненні через те що у такому випадку алгоритм взагалі непереставляє жодного елемента.

Зміна довжини стовпчиків вплинула і на метод сортування Шелла №2, але не при впорядкованому варіанті заповненості, бо при такому заповненні алгоритм не переставляє значення через що розмір стовпчиків не має для нього значення. А ось для випадкового та обернено впорядкованого розмір стовпчиків звісно ж має значення бо

від цього залежить як довго буде відбуватися перестановка елементів стовпчиків при вставці елемента першого рядка на потрібне місце.

## **Випадок дослідження 2.Залежність часу роботи алгоритмів від форми перерізів масива**

Для алгоритму вибору№2 з кожним зменшенням довжини рядка час виконання зменшувався незважаючи на те що кількість стовпчиків збільшувалась,це пов'язано з тим що цей алгоритм переставляє елементи лише один раз за ітерацію,кількість яких дорівнює довжині рядка,тож зі зменшенням довжини рядка алгоритм менше витрачав часу на прохід по всьому рядку хоч йому і доводилось довше переставляти стовпчики,що ми і бачимо з результатів моїх тестів.

Для алгоритму обміну№2 з прапорцем з кожним зменшенням довжини рядка час виконання зменшувався незважаючи на те що кількість стовпчиків збільшувалась,це пов'язано з тим що цей алгоритм часто переставляє сусідні елементи,тож зі зменшенням довжини рядка алгоритм менше витрачав часу на прохід по всьому рядку ,переставляючи елементи, хоч йому і доводилось довше робити перестановки через довгі стовпчики.Також є виключення із цього,а саме впорядкований варіант заповнення,який тільки вигравав від зменшення довжини рядків ,бо він швидше проходив рядок і розумів що він вже впорядкований,отже і не переставляв елементи,через що в цьому випадку алгоритму байдуже на довжину стовпчика.Це можна побачити з результатів моїх тестів.

Для алгоритму сортування Шелла№2 з кожним зменшенням довжини рядка час виконання зменшувався незважаючи на те що кількість стовпчиків збільшувалась,це пов'язано з тим що цьому алгоритму потрібно шукати місце вставки нового елемента для кожного етапу сортування,тож зі зменшенням довжини рядка алгоритм менше витрачав часу на пошук місця вставки,хоч йому і доводилось довше робити перестановки через довгі стовпчики.Також є виключення із цього,а саме впорядкований варіант заповнення,який тільки вигравав від зменшення довжини рядків ,бо він швидше проходив рядок, і не

переставляв елементи, через що в цьому випадку алгоритму байдуже на довжину стовпчика. Це можна побачити з результатів моїх тестів.

### **Випадок дослідження 3. Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масива при однаковій загальній кількості ключів у всьому масиві**

Для алгоритму вибору №2 з кожним зменшенням довжини рядка час виконання зменшувався незважаючи на те що кількість перерізів збільшувалась, це пов'язано з тим що зі зменшенням довжини рядка алгоритм менше витрачав часу на прохід по рядку хоч йому і доводилось проходити по більшій кількості перерізів, що ми і бачимо з результатів моїх тестів.

Для алгоритму обміну №2 з прапорцем з кожним зменшенням довжини рядка час виконання зменшувався незважаючи на те що кількість перерізів збільшувалась, це пов'язано з тим що цей алгоритм часто переставляє сусідні елементи, тож зі зменшенням довжини рядка алгоритм менше витрачав часу на прохід по всьому рядку, переставляючи елементи, хоч йому і доводилось обходити більше перерізів.

Для алгоритму сортування Шелла №2 з кожним зменшенням довжини рядка час виконання зменшувався незважаючи на те що кількість перерізів збільшувалась, це пов'язано з тим що цьому алгоритму потрібно шукати місце вставки нового елемента для кожного етапу сортування, тож зі зменшенням довжини рядка алгоритм менше витрачав часу на пошук місця вставки, хоч йому і доводилось довше сортувати через збільшену кількість перерізів.

## **Висновки**

Мною було досліджено метод прямого вибору №2, метод прямого обміну №2 (з прапорцем) та метод сортування Шелла №2.

1. Метод вибору №2 виявився середнім за швидкістю сортування, але має свою особливість яка полягає у тому що

швидкість відсортовування для впорядкованого, обернено впорядкованого та випадкового майже однакова.

2. Метод прямого обміну №2 виявився найповільнішим з усіх досліджених мною алгоритмів через його неоптимізованість у зміні місцями сусідів, проте у випадку впорядкованості він найшвидший завдяки перевірці на відсортованість.
3. З усіх досліджених мною алгоритмів сортування, найшвидшим був алгоритм Шелла №2 через те що це покращений завдяки зменшенню відстані алгоритм сортування. У швидкості він програє тільки прямому обміну №2 у випадку впорядкованості.
4. Час сортування збільшується при збільшенні довжини стовпчиків, бо при кожній зміні місцями елементів необхідно більше переставляти, найбільше це впливає на прямий обмін №2, але є і виключення, при сортування прямим обміном №2 та Шеллом №2 при впорядкованій заповненості швидкість виконання не залежить від довжини стовпчиків бо алгоритми не переставляють елементи місцями.
5. Час роботи сортування більше залежить від довжини рядків ніж від кількості перерізів чи довжини стовпчиків, це зумовлено тим що при меншій довжині рядків алгоритм менше витрачає часу на переміщення по масиву, що виграє достатньо часу щоб компенсувати збільшення довжини стовпчиків чи кількості перерізів.

## **Список використаної літератури**

- 1) Конспект з СДА.
- 2) Відео з ютуб каналу “Олександр Іванович Марченко”  
<https://www.youtube.com/channel/UCerCwtA87I0yuq3tTQck7vw>
- 3) Н. Вірт “Алгоритмы и структуры данных” М. Мир 1989