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"*

Виконала: Кузьменко О.А.
Група: КВ-31
Номер залікової книжки: -

Допущена до захисту

_____

2 семестр 2023/2024

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Узгоджено

Керівник роботи

_____/Марченко О.І./

ЗАХИЩЕНА "__"_____2023р.

з оцінкою_____

_____ /Марченко О.І./

# ***Дослідження ефективності методів сортування а*лгоритму сортування №3 методу прямого вибору*, алгоритму сортування №5 методу прямого вибору, а*лгоритму сортування №1 методу прямого обміну без модифікацій *на багатовимірних масивах***

Виконавиця роботи:
Кузьменко Олександра Андріївна

_____2024 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 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 ґрунтовного 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 поводять 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 № 86

## Задача

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

## Досліджувані методи та алгоритми

- Алгоритм сортування №3 методу прямого вибору
- Алгоритм сортування №5 методу прямого вибору
- Алгоритм сортування №1 методу прямого обміну без модифікацій

## Спосіб обходу

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

## Випадки дослідження

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

Рекомендовані розміри масива для досліджень:
P = const = 3, M = var, N = var, M*N = const = V.

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

Рекомендовані розміри масива для досліджень:
P = var, M = const = 2000, N = const = 2000.

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

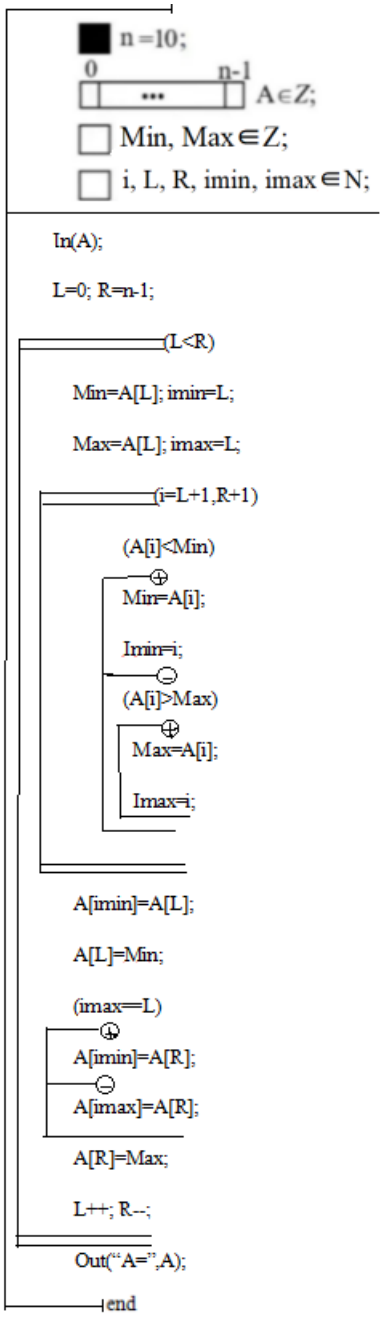
Рекомендовані розміри масива для досліджень:
P = const = 3, M = N.

# 1. Опис теоретичних положень

## (опис алгоритмів відповідно до сортування за незменшенням)

### *1) Алгоритм сортування №3 методу прямого вибору*

Схема алгоритму:



***Принцип роботи алгоритму:***

Цей алгоритм сортує масив шляхом постійного вибору мінімальних і максимальних елементів і розміщення їх у відповідних позиціях. Його складність: $O(n^2)$, тому даний алгоритм сортування не є ефективним для масштабних масивів.

***Цей алгоритм складається з кількох пунктів:***

① ініціалізація змінних Min, Max, L, R, n, imin, imax та i;

② перегляд масиву циклом поки L<R;

③ пошук мінімуму та максимуму, перебираючи елементи;

④ заміна елементів, а саме після знаходження мінімуму та максимуму у поточному діапазоні, алгоритм змінює мін. елемент з індексом L на макс. і макс. на мін. елемент з індексам R;

⑤ оновлення індексів, а саме алгоритм збільшує L і зменшує R, скорочуючи діапазон, що відсортовуватиметься під часнаступних ітерацій;

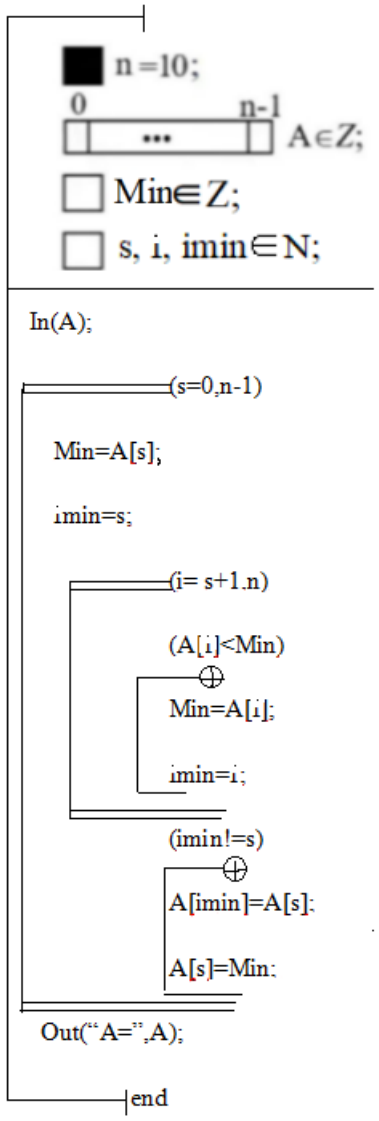
⑥ Повторення попередніх 3 пунктів, поки не буде так, що L>R або L=R, що буде індекатором повністю відсортованого масиву.

***Особливість роботи даного алгоритму у порівнянні з класичним алгоритмом вибору:***

Алгоритм методу вибору №3 виконує значно більше обмінів. А його класична версія є простішою, зрозумілішою і для більших масивів можна вважати оптимізованою (кількість операція менша).

### *2) Алгоритм сортування №5 методу прямого вибору*

Схема алгоритму:




***Принцип роботи алгоритму:***

Цей алгоритм сортує масив шляхом постійного пошуку мінімуму серед невідсортованих елементів і становлення його в правильне місце, також його складність $O(n^2)$, це робить даний алгоритм сортування менш ефективним для масштабних масивів.

***Цей алгоритм складається з кількох пунктів:***

① ініціалізація змінних Min, imin, s, і та n;

②пошук мінімуму, а саме внутрішній цикл виконує ітерацію по несортованій частині від s+1 до n-1, порівнюючи елементи з наявним Min, коли трапляється менший елемент, змінна Min та imin оновлюються;

③заміна елементів ,а саме, коли мінімум знаходиться в діапазоні від s до n-1, йде перевірка чи imin!=s, якщо істина, то менший елемент знайдено і A[s]=A[imin];

④працює зовнішній цикл, зменшуючи кількість невідсортованих елементів шляхом збільшення змінної s, кожна ітерація сприяє тому, що найменший елемент стає на своє місце;

⑤алгоритм завершується, бо завдяки зовнішньому циклу нульовий елемент мінімальний і останній найбільший.

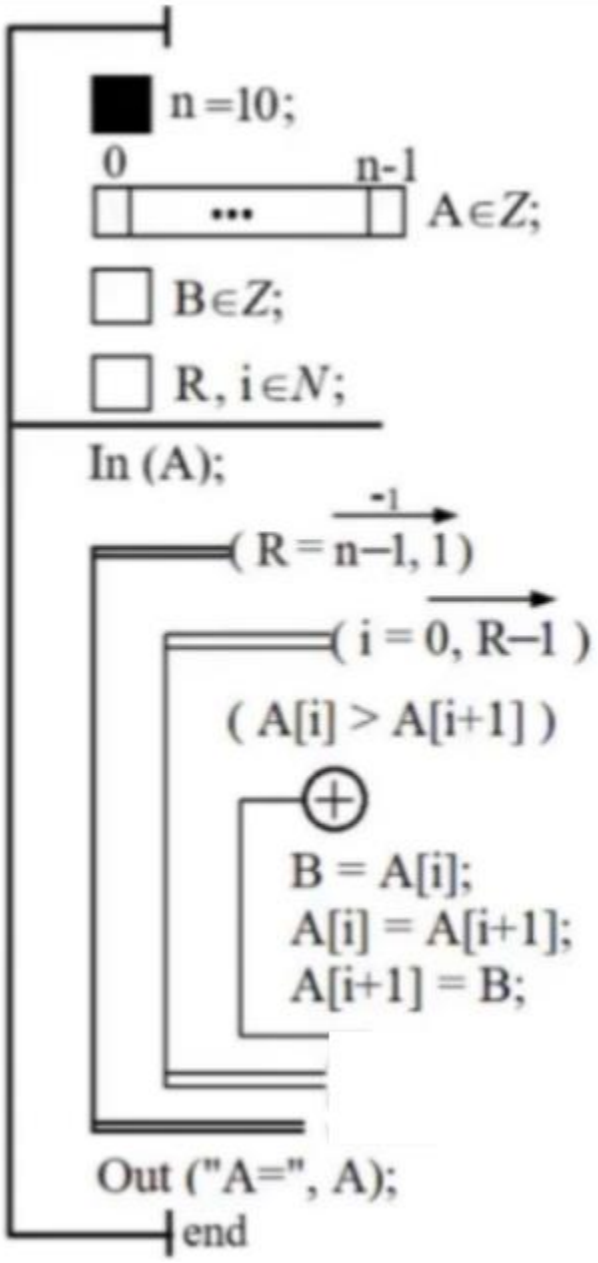
***Особливість роботи даного алгоритму у порівнянні з класичним алгоритмом вибору:***

Алгоритм методу вибору №5 є ефективнішим, бо має менше обмінів.

У даному алгоритмі елементи обмінюються один раз, коли змінна Min оновиться, а в класичному варіанті обмін відбувається після кожного пошуку мінімума серед невідсортованих елементів.

### *3) Алгоритм сортування № 1 методу прямого обміну без модифікацій*

Схема алгоритму:




***Принцип роботи алгоритму:***

***Цей алгоритм складається з кількох пунктів:***

①ініціалізація змінних B, R, i та n;

②внутрішній цикл від нульового по R-1 елемент перевіряє чи поточний більший за наступний елемент, якщо це так, то вони стоять неправильно і їх алгоритм має поміняти місцями;

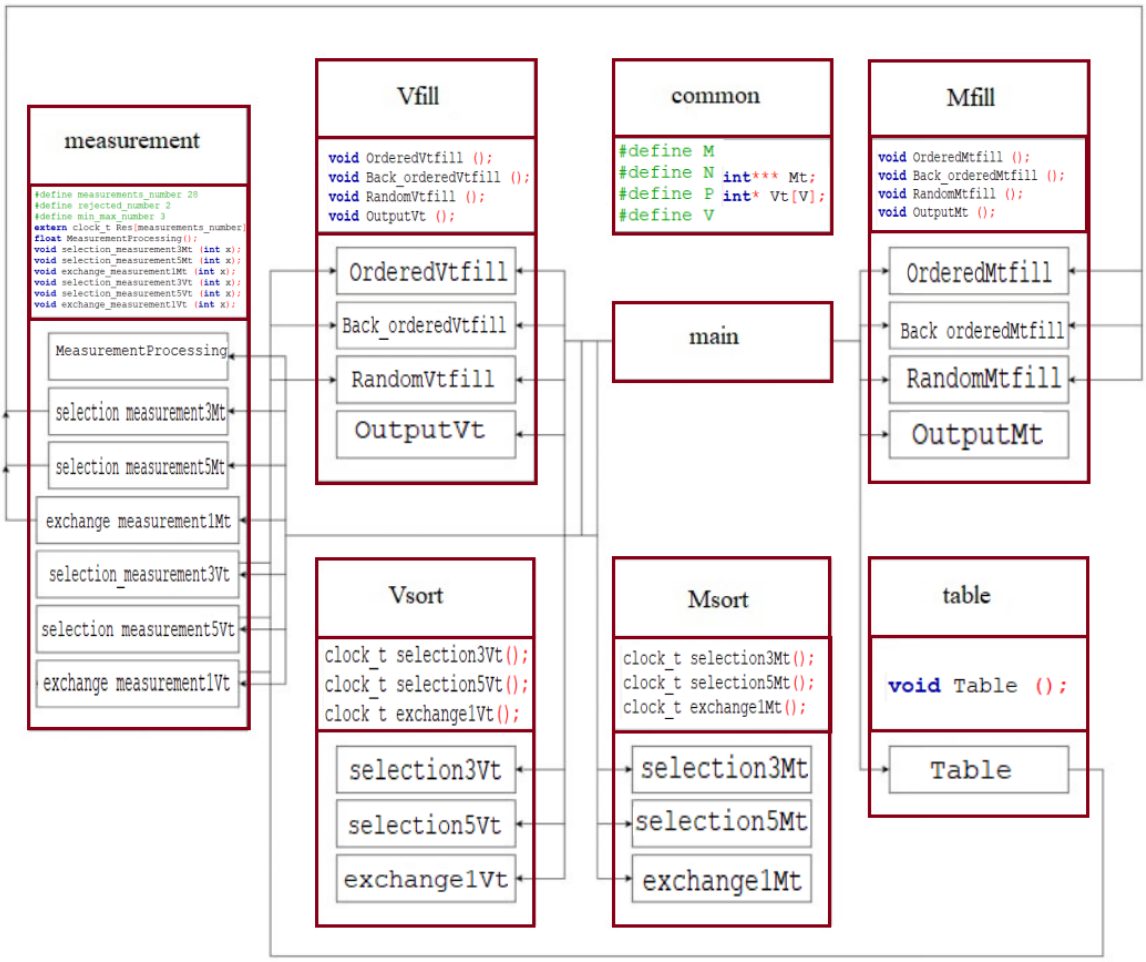
③зміна елементів, що згадувалися в попередньому кроці, якщо дійсно поточний більший за наступний елемент;

④за допомоги внутрішнього циклу найбільший елемент поступово наближається до свого місця в кінці масиву;

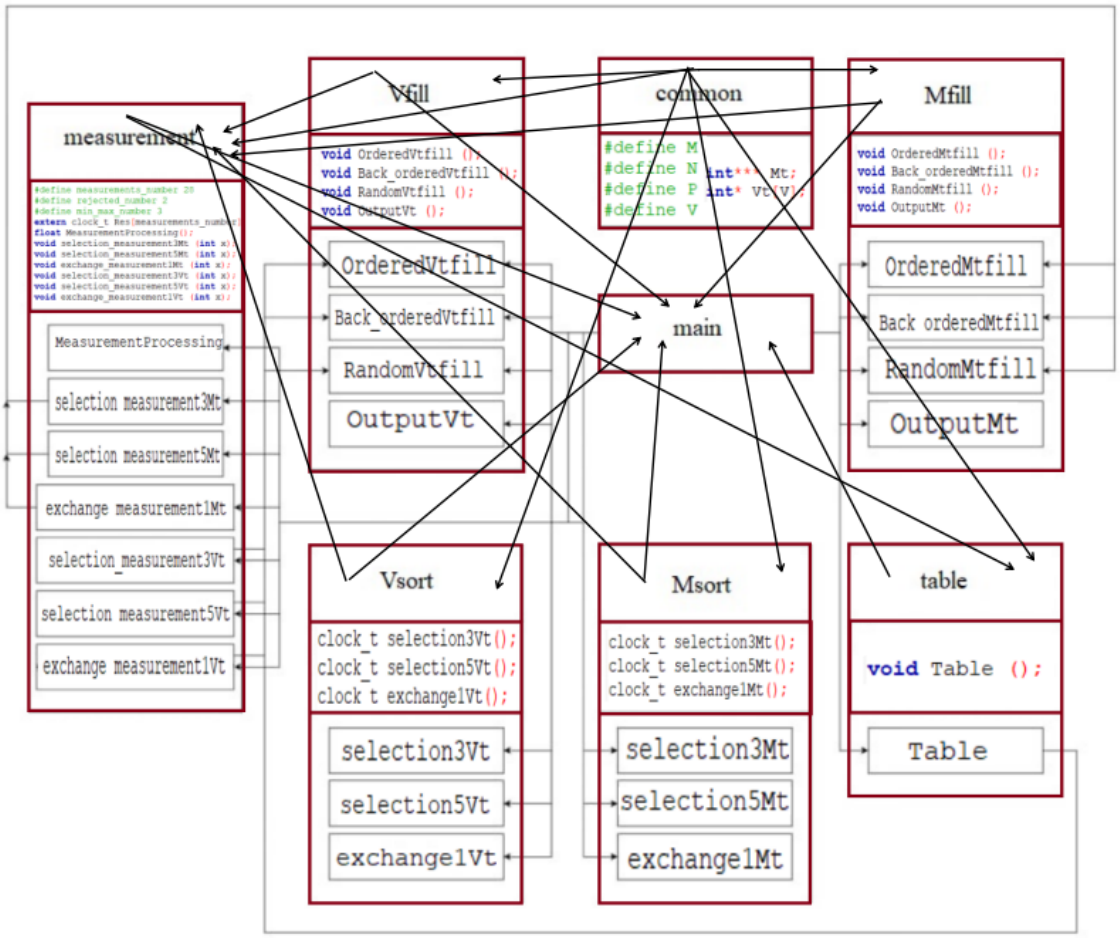
⑤алгоритм завершується, R скорочується у зовнішньому циклі, при цьому кількість невідсортованих елементів зменшується, а цей цикл працюватиме, поки R не буде 1, а якщо R=1, то масив відсортований.

***Це і є класичним варіантом методу обміну, тому подібного порівняння не потребує даний алгоритм.***

# Структурна схема взаємовикликів процедур та функцій програми курсової роботи



# Схема імпорту/експорту модулів програми курсової роботи



## Опис призначення всіх функцій і процедур та їх параметрів

1. Vfill:

1) `void OrderedVtfill ();` Ця функція заповнює векторний масив впорядкованими значеннями, використовуючи цикл та змінну, перший елемент=0, а потім з кожною ітерацією значення змінної зростає на 1, і вона присвоюється почергово кожному елементу масива. Функція використовує глобальну змінну V, що є довжиною даного масива та задається в common.c, та сам масив Vt[V].

2) `void Back_orderedVtfill ();` Ця функція заповнює векторний масив обернено впорядкованими значеннями, використовуючи цикл та змінну, перший елемент = V, а потім з кожною ітерацією значення змінної зменшується, і вона присвоюється почергово кожному елементу масива. Функція використовує глобальну змінну V, що є довжиною даного масива та задається в common.c, та сам масив Vt[V].

3) `void RandomVtfill ();` Ця функція заповнює векторний масив випадковими значеннями, використовуючи цикл, генератор випадкових чисел, мінімальне та максимальне значення. Функція використовує глобальну змінну V, що є довжиною даного масива та задається в common.c, та сам масив Vt[V].

4) `void OutputVt ();` Ця функція виводить векторний масив на екран,використовуючи цикл для проходження по кожному елементу, всі елементи є числами, та для візуальної зручності між ними є пробіли.

2. Mfill:

1) `void OrderedMtfill ();`Ця функція заповнює тривимірний масив впорядкованими значеннями, використовуючи 3 цикли та змінну, перший елемент=0, а потім з кожною ітерацією значення змінної зростає на 1, і вона присвоюється почергово кожному елементу масива. Функція використовує глобальні змінні P, M, N, що є розміром даного масива та задається в common.c, та сам масив Mt[P][M][N].

2) `void Back_orderedMtfill ();` Ця функція заповнює тривимірний масив обернено впорядкованими значеннями, використовуючи 3 цикли та змінну, перший елемент=P * M * N, а потім з кожною ітерацією значення змінної зменшується на 1, і вона присвоюється почергово кожному елементу масива. Функція використовує глобальні змінні P,

M, N, що є розміром даного масива та задається в common.c, та сам масив Mt[P][M][N].

3) `void RandomMtfill()` Ця функція заповнює тривимірний масив випадковими значеннями, використовуючи 3 цикли, генератор випадкових чисел, мінімальне та максимальне значення Функція використовує глобальні змінні P, M, N, що є розміром даного масива та задається в common.c, та сам масив Mt[P][M][N].

4) `void OutputMt ();` Ця функція виводить тривимірний масив на екран, використовуючи 3 цикли для проходження по кожному елементу, всі елементи є числами, та для візуальної зручності між ними є пробіли та переходи на наступний рядок.

3. Vsort:

1) `clock_t selection3Vt();` Ця функція сортує векторний масив методом прямого вибору 3.
2) `clock_t selection5Vt();` Ця функція сортує векторний масив методом прямого вибору 5.
3) `clock_t exchange1Vt();` Ця функція сортує векторний масив методом обміну 1.

4. Msort:

1) `clock_t selection3Mt();` Ця функція сортує тривимірний масив методом прямого вибору 3.
2) `clock_t selection5Mt();` Ця функція сортує тривимірний масив методом прямого вибору 5.
3) `clock_t exchange1Mt();` Ця функція сортує тривимірний масив методом обміну 1.

5. Table:

`void Table ();` Ця функція виконує вимірювання часу сортування алгоритмами прямого вибору 3, прямого вибору 5, обміну 1 тривимірного та векторного масивів, які заповнюються послідовними, обернено послідовними та випадковими числами та виводить на консоль таблицю з результатами та підписом колонок.

6. Measurement:

1) `float MeasurementProcessing();` Ця функція обробляє вимірювання шляхом відкидання випадкових викидів та визначення середнього значення залишкових даних.

2) `void selection_measurement3Mt (int x);` Ця функція використовується для заповнення тривимірного масиву різними типами значень та виклику прямого вибору 5 алгоритму для кожного з них. Параметр x вказує на тип заповнення масиву: якщо x = 1, то масив заповнюється послідовними числами, якщо x = 2, то обернено послідовними, а якщо x = 3, то випадковими. Функція зберігає результат в глобальному масиві Res.

3) `void selection_measurement5Mt (int x);` Ця функція використовується для заповнення тривимірного масиву різними типами значень та виклику прямого вибору 3 алгоритму для кожного з них. Параметр x вказує на тип заповнення масиву: якщо x = 1, то масив заповнюється послідовними числами, якщо x = 2, то обернено послідовними, а якщо x = 3, то випадковми. Функція зберігає результат в глобальному масиві Res.

3) `void exchange_measurement1Mt (int x);` Ця функція використовується для заповнення тривимірного масиву різними типами значень та виклику обміну 1 алгоритму для кожного з них. Параметр x вказує на тип заповнення масиву: якщо x = 1, то масив заповнюється послідовними числами, якщо x = 2, то обернено послідовними, а якщо x = 3, то випадковими. Функція зберігає результат в глобальному масиві Res.

4) `void selection_measurement3Vt (int x);` Ця функція використовується для заповнення векторного масиву різними типами значень та виклику прямого вибору 3 алгоритму для кожного з них. Параметр x вказує на тип заповнення масиву: якщо x = 1, то масив заповнюється послідовними числами, якщо x = 2, то обернено послідовними, а якщо x = 3, то рандомними. Функція зберігає результат в глобальному масиві Res.

5) `void selection_measurement5Vt (int x);` Ця функція використовується для заповнення векторного масиву різними типами значень та виклику прямого вибору 5 алгоритму для кожного з них. Параметр x вказує на тип заповнення масиву: якщо x = 1, то масив заповнюється послідовними числами, якщо x = 2, то обернено послідовними, а якщо x = 3, то рандомними. Функція зберігає результат в глобальному масиві Res.

6) `void exchange_measurement1Vt (int x);` Ця функція використовується для заповнення векторного масиву різними типами значень та виклику обміну 1 алгоритму для кожного з них. Параметр x

вказує на тип заповнення масиву: якщо x = 1, то масив заповнюється послідовними числами, якщо x = 2, то обернено послідовними, а якщо x = 3, то рандомними. Функція зберігає результат в глобальному масиві Res.

7. main:

1) `void GotoXY(int X, int Y)` Функція для постановки курсора в консольному вікні за координатами (X, Y).

2) `void colour(int i)` Функція для задання кольору виведення наступних символів.

3) `void ConsoleCursorVisible(int show, short size)` Функція, що функція керує видимістю курсора в консольному вікні та його розміром.

4) `void submenu( int q)`Підменю для вибору виду заповнення масиву, використовується параметр для передання методу сортування.

5) `void menu()` Головне меню програми.

## Текст програми з коментарями

## *Common.h*

```
#ifndef COMMON_H
#define COMMON_H

#define M 3
#define N 4
#define P 2
#define V 10
int*** Mt;
int* Vt[V];
#endif Common_H
```

## *Vfill.h*

```
#ifndef VFILL_H
#define VFILL_H

void OrderedVtfill ();
void Back_orderedVtfill ();
void RandomVtfill ();
void OutputVt ();

#endif VFILL_H
```

## *Vfill.c*

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

#include "common.h"
#include "Vfill.h"
// Функція для заповнення масиву Vt послідовними числами
void OrderedVtfill()
{
    long int num = 0;
    for (long int i = 0; i < V; i++)//цикл для проходу по всіх
елементах масиву Vt
        Vt[i] = num++;//присв. кожному елементу змінну num, яка з
кожною ітерацією збільшується на 1
}
// Функція для заповнення масиву Vt послідовними числами у зворотньому
порядку
void Back_orderedVtfill()
{
    for (long int i = 0; i < V; i++)//цикл для проходу по всіх
елементах масиву Vt
        Vt[i] = V - i;//присв. кожному елементу змінну(спочатку =
конст. V), що з кожною ітерацією зменшується на 1
}
// Функція для заповнення масиву Vt випадковими числами
void RandomVtfill()
{
    srand(time(NULL));//ініціалізація генератора випадкових чисел
    long int max = M*N*P, min = 0;//визначення максимуму та мінімуму
    for(long int j=0; j<V; j++)//цикл для проходу по всіх елементах
масиву Vt
```

```
        Vt[j] = rand() % (P * M * N);//заповнення кожного елемента
масиву випадковими числами
}
// Функція для виведення масиву Vt
void OutputVt()
{
    for (long int i = 0; i < V; i++)//цикл для проходу по всіх
елементах масиву Vt
        printf("%d ", Vt[i]);//виведення кожного елементу масиву
    printf("\n\n");//відступ на екрані для наступних виведень
}
```

## Mfill.h

```
#ifndef MFILL_H
#define MFILL_H

void OrderedMtfill ();
void Back_orderedMtfill ();
void RandomMtfill ();
void OutputMt ();

#endif Mfill_H
```

## Mfill.c

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

#include "common.h"
#include "Mfill.h"
// Функція для заповнення масиву Mt послідовними числами
void OrderedMtfill()
{
    int num=1;
    for (long int k=0; k<P; k++)//цикл для проходу по всіх перерізах
масиву Mt
        for (long int j=0; j<N; j++)//цикл для проходу по всіх стовпцях
поточного перерізу масиву Mt
            for (long int i=0; i<M; i++)//цикл для проходу по всіх
рядках поточного стовпця масиву Mt
                Mt[k][i][j] = num++;//присв. кожному елементу змінну
num, яка з кожною ітерацією збільшується на 1
}
// Функція для заповнення масиву Mt послідовними числами у зворотньому
порядку
void Back_orderedMtfill()
{
    long int num = P * M * N;
    for (long int k = 0; k < P; k++)//цикл для проходу по всіх
перерізах масиву Mt
        for (long int j = 0; j < N; j++)//цикл для проходу по всіх
стовпцях поточного перерізу масиву Mt
            for (long int i = 0; i < M; i++)//цикл для проходу по всіх
рядках поточного стовпця масиву Mt
                Mt[k][i][j] = num--;//присв. кожному елементу
змінну(спочатку = конст. P*M*N), що з кожною ітерацією зменшується на 1
```

```
}
// Функція для заповнення масиву Mt випадковими числами
void RandomMtfill()
{
    for (long int k = 0; k < P; k++)//цикл для проходу по всіх
перерізах масиву Mt
        for (long int j = 0; j < N; j++)//цикл для проходу по всіх
стовпцях поточного перерізу масиву Mt
            for (long int i = 0; i < M; i++)//цикл для проходу по всіх
рядках поточного стовпця масиву Mt
                Mt[k][i][j] = rand() % (P * M * N);//заповнення кожного
елемента масиву випадковими числами
}
// Функція для виведення масиву Mt
void OutputMt()
{
    for (long int k = 0; k < P; k++)//цикл для проходу по всіх
перерізах масиву Mt
    {
        printf("\nP = %d\n", k);
        for (long int i = 0; i < M; i++)//цикл для проходу по всіх
рядках поточного перерізу масиву Mt
        {
            for (long int j = 0; j < N; j++)//цикл для проходу по всіх
стовпцях поточного рядка масиву Mt
                printf("%d ", Mt[k][i][j]);//виведення кожного елементу
масиву
            printf("\n");//перехід на наступний рядок
        }
    }
}
```

## *Vsort.h*

```
#ifndef VSORT_H
#define VSORT_H
#include <time.h>

clock_t selection3Vt();
clock_t selection5Vt();
clock_t exchange1Vt();

#endif VSORT_H
```

## *Vsort.c*

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

#include "common.h"
#include "Vsort.h"
#include <time.h>
// Функція сортування масиву методом прямого вибору 3
clock_t selection3Vt ()
{
    long int Min, Max, L, R, imin, imax;
    clock_t time_start, time_stop;// Змінні для збереження часу початку
та завершення виконання функції
```

```
    time_start = clock();// Запам'ятовування часу початку виконання
    L=0;
    R=V-1;
    while (L<R)
    {
        Min=Vt[L];//перший невідсортований елемент буде мінімумом та
максимумом,
        imin=L;// а також його індекс(для подальних пошуків мін. та
макс.)
        Max=Vt[L];
        imax=L;
        for(long int i=L+1; i<R+1; i++)//пошук дійсного мінімума та
максимума шляхом перебору невідсортованих ел.
        {
            if (Vt[i] < Min)//поточний може мін?
            {
                Min=Vt[i];
                imin=i;
            }
            else if (Vt[i] > Max)//поточний може макс?
            {
                Max=Vt[i];
                imax=i;
            }
        }//обмін мін і макс до першого і останнього невідсортованого
елемента
        Vt[imin]=Vt[L];
        Vt[L]=Min;
        if (imax==L) Vt[imin]=Vt[R];
        else Vt[imax]=Vt[R];
        Vt[R]=Max;
        L=L+1;//підготовка до наступної ітерації, адже на 2
відсортованих ел. стало більше
        R=R-1;
    }
    time_stop = clock();// Запам'ятовування часу завершення виконання
    return time_stop - time_start;// Повернення часу виконання функції
}
// Функція сортування масиву методом прямого вибору 5
clock_t selection5Vt ()
{
    long int Min, imin;
    clock_t time_start, time_stop;// Змінні для збереження часу початку
та завершення виконання функції
    time_start = clock();// Запам'ятовування часу початку виконання
    for(long int s=0; s<V-1; s++)
    {
        Min=Vt[s];//перший невідсортований елемент буде мінімумом
        imin=s;// а також його індекс(для подальних пошуків мін.)
        for(long int i=s+1; i<V; i++)//пошук дійсного мінімума шляхом
перебору невідсортованих ел.
            if (Vt[i]<Min)//поточний може мін?
            {
                Min=Vt[i];
                imin=i;
            }
```

```
        if (imin!=s)// обмін мінімального елемента з першим
непосортованим елементом
        {
            Vt[imin]=Vt[s];
            Vt[s]=Min;
        }
    }
    time_stop = clock();// Запам'ятовування часу завершення виконання
    return time_stop - time_start;// Повернення часу виконання функції
}
// Функція сортування масиву методом обміну 1
clock_t exchange1Vt ()
{
    long int tmp;
    clock_t time_start, time_stop;// Змінні для збереження часу початку
та завершення виконання функції
    time_start = clock();// Запам'ятовування часу початку виконання
    for(long int R=V-1; R>0; R--)
    {
        for(long int i=0; i<R; i++)
            if (Vt[i]>Vt[i+1])
            {// Обмін сусідніх елементів, якщо вони не відсортовані
                tmp=Vt[i];
                Vt[i]=Vt[i+1];
                Vt[i+1]=tmp;
            }
    }
    time_stop = clock();// Запам'ятовування часу завершення виконання
    return time_stop - time_start;// Повернення часу виконання функції
}
```

## *Msort.h*

```
#ifndef MSORT_H
#define MSORT_H
#include <time.h>

clock_t selection3Mt();
clock_t selection5Mt();
clock_t exchange1Mt();

#endif MSORT_H
```

## *Msort.c*

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

#include "common.h"
#include "Msort.h"
#include <time.h>
clock_t selection3Mt ()// Функція сортування масиву методом прямого
вибору 3
{
    long int L, A, R, B, imin, jmin, imax, jmax, Min, Max;
    clock_t time_start, time_stop;// Змінні для збереження часу початку
та завершення виконання функції
    time_start = clock();// Запам'ятовування часу початку виконання
```

```
    for (long int k = 0; k < P; k++)//цикл для проходу по всіх
перерізах масиву Mt
    {
        L = 0;
        A = 0;
        R = N - 1;
        B = M - 1;
        while (L <= R)
        {//Перший невідсортований елемент буде мінімумом та максимумом,
            imin = A;// а також його індекс(для подальних пошуків мін.
та макс.)
            jmin = L;
            imax = A;
            jmax = L;
            Min = Mt[k][A][L];
            Max = Mt[k][A][L];
            for (long int i = A; i < M; i++)//Пошук дійсного мінімума
та максимума шляхом перебору невідсортованих ел.
            {
                if (Mt[k][i][L] < Min)//Поточний може мін?
                {
                    imin = i;
                    jmin = L;
                    Min = Mt[k][i][L];
                }
                else if (Mt[k][i][L] > Max)//Поточний може макс?
                {
                    imax = i;
                    jmax = L;
                    Max = Mt[k][i][L];
                }
            }
            for (long int j = L + 1; j < R; j++)
            {
                for (long int i = 0; i < M; i++)
                {
                    if (Mt[k][i][j] < Min)//Поточний може мін?
                    {
                        imin = i;
                        jmin = j;
                        Min = Mt[k][i][j];
                    }
                    else if (Mt[k][i][j] > Max)//Поточний може макс?
                    {
                        imax = i;
                        jmax = j;
                        Max = Mt[k][i][j];
                    }
                }
            }
            for (long int i = 0; i < B + 1; i++)
            {
                if (Mt[k][i][R] < Min)//Поточний може мін?
                {
                    imin = i;
                    jmin = R;
```

```
                    Min = Mt[k][i][R];
                }
                else if (Mt[k][i][R] > Max)//поточний може макс?
                {
                    imax = i;
                    jmax = R;
                    Max = Mt[k][i][R];
                }
            }//Обмін мін і макс до першого і останнього
невідсортованого елемента
            Mt[k][imin][jmin] = Mt[k][A][L];
            Mt[k][A][L] = Min;
            if (jmax == L && imax == A)
                Mt[k][imin][jmin] = Mt[k][B][R];
            else
                Mt[k][imax][jmax] = Mt[k][B][R];
            Mt[k][B][R] = Max;
            A++;
            B--;
            if (A == M)// Оновлення індексів та крайніх позицій для
наступної ітерації циклу
            {//підготовка до наступної ітерації, адже на 2
відсортованих ел. стало більше
                L++;
                R--;
                A = 0;
                B = M - 1;
            }
        }//Передбачення випадку, коли кількість рядків більша за
кількість стовпців
        if ((L - R) == 2)
        {
            L -= 1;
            A = 0;
            B = M - 1;
            while (A < B)
            {//Перший невідсортований елемент буде мінімумом та
максимумом,
            imin = A; // а також його індекс(для подальних пошуків мін.
та макс.)
                imax = A;
                Min=Mt[k][A][L];
                Max=Mt[k][A][L];
                for (long int i = A + 1; i < B + 1; i++)// Пошук мін.
та максю у стовпці
                {
                    if (Mt[k][i][L] < Min)//поточний може мін?
                    {
                        imin = i;
                        Min=Mt[k][i][L];
                    }
                    else if (Mt[k][i][L] > Max)//поточний може макс?
                    {
                        imax = i;
                        Max=Mt[k][i][L];
                    }
```

```
                }//Обмін мін і макс до першого і останнього
невідсортованого елемента
                Mt[k][imin][L] = Mt[k][A][L];
                Mt[k][A][L] = Min;
                if (imax == A)
                    Mt[k][imin][L] = Mt[k][B][L];
                else
                    Mt[k][imax][L] = Mt[k][B][L];
                A++;//підготовка до наступної ітерації, адже на 2
відсортованих ел. стало більше
                B--;
            }
        }
    }
    time_stop = clock();// Запам'ятовування часу завершення виконання
    return time_stop - time_start;// Повернення часу виконання функції
}
clock_t selection5Mt ()// Функція сортування масиву методом прямого
вибору 5
{
    long int Min, imin, jmin, o;
    clock_t time_start, time_stop;// Змінні для збереження часу початку
та завершення виконання функції
    time_start = clock();// Запам'ятовування часу початку виконання
    for(long int k = 0; k < P; k++)//цикл для проходу по всіх перерізах
масиву Mt
    {
        for(long int j = 0; j < N; j++)//цикл для проходу по всіх
стовпцях масиву Mt
        {
            for(long int i = 0; i < M; i++)//цикл для проходу по всіх
рядках масиву Mt
            {//Перший невідсортований елемент буде мінімумом та
максимумом,
            jmin = j;// а також його індекс(для подальних пошуків мін.
та макс.)
                imin = i;
                Min = Mt[k][imin][jmin];
                for (long int q = j; q < N; q++)//пошук мін починаючи з
поточного стовпця, бо до нього вже відсортовано або це 0ий
                {//якщо пошук мін іде на поточному стовпці, то пошук
починається з наступного рядка ,
                    o = (q == j) ? (i + 1) : 0;//якщо наступний ст. то
з 0го(стосовно наступного циклу)
                    for(long int b = o ; b < M; b++)
                    {
                        if (Mt[k][b][q] < Min)//може поточний мін?
                        {
                            imin = b;
                            jmin = q;
                            Min = Mt[k][imin][jmin];
                        }
                    }
                }//якщо мін. знайдено, що обмінюється з першим
невідсортованим
                if (jmin != j || imin != i)
                {
```

```
                    Mt[k][imin][jmin] = Mt[k][i][j];
                    Mt[k][i][j] = Min;
                }
            }
        }
    }
    time_stop = clock();// Запам'ятовування часу завершення виконання
    return time_stop - time_start;// Повернення часу виконання функції
}
clock_t exchange1Mt ()// Функція сортування масиву методом обміну 1
{
    long int q, b, tmp;
    clock_t time_start, time_stop;// Змінні для збереження часу початку
та завершення виконання функції
    time_start = clock();// Запам'ятовування часу початку виконання
    for(long int k = 0; k < P; k++)//цикл для проходу по всіх перерізах
масиву Mt
    {
        for(long int j = N - 1; j >= 0; j--)//цикл для проходу по всіх
стовпцях масиву Mt
        {
            for(long int i = M - 1; i >= 0; i--)//цикл для проходу по
всіх рядках масиву Mt
            {
                for(q = 0; q < j; q++)//цикл для проходу по всіх
стовпцях до поточного
                {
                    for(b = 0; b < M - 1; b++)//цикл для проходу по
всіх рядках крім останнього
                    {
                        if(Mt[k][b][q] > Mt[k][b+1][q])//поточний
більший за наступний?
                        {
                            tmp = Mt[k][b][q];//якщо так, то їх обмін
                            Mt[k][b][q] = Mt[k][b+1][q];
                            Mt[k][b+1][q] = tmp;
                        }
                    }
                    if(Mt[k][M-1][q] >
Mt[k][0][q+1])//поточний(кінцевий цього стовпця) більший за
наступний(0ий наст. ст.)
                    {
                        tmp = Mt[k][M-1][q];//якщо так, то їх обмін
                        Mt[k][M-1][q] = Mt[k][0][q+1];
                        Mt[k][0][q+1] = tmp;
                    }
                }
                for(b = 0; b < i; b++)// Перевірка та обмін елементів
                {
                    if(Mt[k][b][j] > Mt[k][b+1][j])//поточний більший
за наступний? (у цьому стовпці)
                    {
                        tmp = Mt[k][b][j];//якщо так, обмін
                        Mt[k][b][j] = Mt[k][b+1][j];
                        Mt[k][b+1][j] = tmp;
                    }
                }
```

```
            }
        }
    }
    time_stop = clock();// Запам'ятовування часу завершення виконання
    return time_stop - time_start;// Повернення часу виконання функції
}
```

## Table.h

```
#ifndef TABLE_H
#define TABLE_H

void Table ();

#endif  TABLE_H
```

## Table.c

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

#include "table.h"
#include "common.h"
#include "Measurement.h"

// Функція для виведення таблиці з результатами виміру часу сортувань
на екран
void Table()
{
    float o, b, r; // Змінні для результатів вимірювань
    // Виведення заголовку таблиці з розмірами тривимірного масиву
    printf("\t\t\t\t  Table for the array: P = %d, M = %d, N = %d\n\n",
P, M, N);
    printf("\t\t\tOrdered\t\t\tRandom\t\t\tBack-ordered\n\n");

    // Вимірювання часу сортувань для масиву методом сортування прямого
вибору 3
    selection_measurement3Mt(1);
    o = MeasurementProcessing();
    selection_measurement3Mt(3);
    r = MeasurementProcessing();
    selection_measurement3Mt(2);
    b = MeasurementProcessing();
    printf("Selection3\t\t%f\t\t%f\t\t%f\n\n", o, r, b);//виведення 1
рядка таблиці
// Вимірювання часу сортувань для масиву методом сортування прямого
вибору 5
    selection_measurement5Mt(1);
    o = MeasurementProcessing();
    selection_measurement5Mt(3);
    r = MeasurementProcessing();
    selection_measurement5Mt(2);
    b = MeasurementProcessing();
    printf("Selection5\t\t%f\t\t%f\t\t%f\n\n", o, r, b);//виведення 2
рядка таблиці
// Вимірювання часу сортувань для масиву методом сортування обміну 1
    exchange_measurement1Mt(1);
    o = MeasurementProcessing();
```

```
    exchange_measurement1Mt(3);
    r = MeasurementProcessing();
    exchange_measurement1Mt(2);
    b = MeasurementProcessing();
    printf("Exchange1\t\t%f\t\t%f\t\t%f\n\n", o, r, b);//виведення 3
рядка таблиці

    // Виведення заголовку таблиці з розміром векторного масиву
    printf("\t\t\t\t  Table for the vector: V = %d\n\n", V);
    printf("\t\t\tOrdered\t\t\tRandom\t\t\tBack-ordered\n\n");

    // Вимірювання часу сортувань для масиву методом сортування прямого
вибору 3
    selection_measurement3Vt(1);
    o = MeasurementProcessing();
    selection_measurement3Vt(3);
    r = MeasurementProcessing();
    selection_measurement3Vt(2);
    b = MeasurementProcessing();
    printf("Selection3\t\t%f\t\t%f\t\t%f\n\n", o, r, b);//виведення 1
рядка таблиці
// Вимірювання часу сортувань для масиву методом сортування прямого
вибору 5
    selection_measurement5Vt(1);
    o = MeasurementProcessing();
    selection_measurement5Vt(3);
    r = MeasurementProcessing();
    selection_measurement5Vt(2);
    b = MeasurementProcessing();
    printf("Selection5\t\t%f\t\t%f\t\t%f\n\n", o, r, b);//виведення 2
рядка таблиці
// Вимірювання часу сортувань для масиву методом сортування обміну 1
    exchange_measurement1Vt(1);
    o = MeasurementProcessing();
    exchange_measurement1Vt(3);
    r = MeasurementProcessing();
    exchange_measurement1Vt(2);
    b = MeasurementProcessing();
    printf("Exchange1\t\t%f\t\t%f\t\t%f\n\n", o, r, b);//виведення 3
рядка таблиці
}
```

## *Measurement.h*

```
#ifndef MEASUREMENT_H
#define MEASUREMENT_H
#include <time.h>
// загальна кількість вимірів часу дії алгоритма
#define measurements_number 28
// кількість відкинутих початкових вимірів
#define rejected_number 2
// кількість відкинутих вимірів з мін та макс
#define min_max_number 3
// масив для значень часу роботи алгоритма
extern clock_t Res[measurements_number];
float MeasurementProcessing();
```

```
// функції для виконання вимірів часу для тривимірного масиву(з
уточненням яке заповнення масиву)
void selection_measurement3Mt (int x);
void selection_measurement5Mt (int x);
void exchange_measurement1Mt (int x);
// функції для виконання вимірів часу для векторного масиву(з
уточненням яке заповнення масиву)
void selection_measurement3Vt (int x);
void selection_measurement5Vt (int x);
void exchange_measurement1Vt (int x);

#endif MEASUREMENT_H
```

## *Measurement.c*

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

#include "common.h"
#include "Mfill.h"
#include "Vfill.h"
#include "Vsort.h"
#include "Msort.h"
#include "Measurement.h"

clock_t Res[measurements_number];
// функція обробки значень вимірів часу роботи алгоритма, яка повертає
безпосередньо час дії алгоритма
float MeasurementProcessing()
{
    long int Sum;
    float AverageValue;
    clock_t buf;
    int L = rejected_number, R = measurements_number - 1, 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;
}
// функція для виконання вимірів часу для тривимірного масиву методом
сортування прямого вибору 3
void selection_measurement3Mt(int x)
{
    for (int i = 0; i < measurements_number; i++)
    {
        if (x == 1) // ordered array
        {
            OrderedMtfill();
            Res[i] = selection3Mt();
        }
        else if (x == 2) // backordered array
        {
            Back_orderedMtfill();
            Res[i] = selection3Mt();
        }
        else if (x == 3) // random array
        {
            RandomMtfill();
            Res[i] = selection3Mt();
        }
    }
}// функція для виконання вимірів часу для векторного масиву методом
сортування прямого вибору 3
void selection_measurement3Vt(int x)
{
    for (int i = 0; i < measurements_number; i++)
    {
        if (x == 1) // ordered array
        {
            OrderedVtfill();
            Res[i] = selection3Vt();
        }
        else if (x == 2) // back-ordered array
        {
            Back_orderedVtfill();
            Res[i] = selection3Vt();
        }
        else if (x == 3) // random array
        {
            RandomVtfill();
            Res[i] = selection3Vt();
        }
    }
```

```
}// функція для виконання вимірів часу для тривимірного масиву методом
сортування прямого вибору 5
void selection_measurement5Mt(int x)
{
    for (int i = 0; i < measurements_number; i++)
    {
        if (x == 1) // ordered array
        {
            OrderedMtfill();
            Res[i] = selection5Mt();
        }
        else if (x == 2) // backordered array
        {
            Back_orderedMtfill();
            Res[i] = selection5Mt();
        }
        else if (x == 3) // random array
        {
            RandomMtfill();
            Res[i] = selection5Mt();
        }
    }
}// функція для виконання вимірів часу для векторного масиву методом
сортування прямого вибору 5
void selection_measurement5Vt(int x)
{
    for (int i = 0; i < measurements_number; i++)
    {
        if (x == 1) // ordered array
        {
            OrderedVtfill();
            Res[i] = selection5Vt();
        }
        else if (x == 2) // back-ordered array
        {
            Back_orderedVtfill();
            Res[i] = selection5Vt();
        }
        else if (x == 3) // random array
        {
            RandomVtfill();
            Res[i] = selection5Vt();
        }
    }
}// функція для виконання вимірів часу для тривимірного масиву методом
сортування обміну 1
void exchange_measurement1Mt(int x)
{
    for (int i = 0; i < measurements_number; i++)
    {
        if (x == 1) // ordered array
        {
            OrderedMtfill();
            Res[i] = exchange1Mt();
        }
        else if (x == 2) // back-ordered array
        {
```

```
            Back_orderedMtfill();
            Res[i] = exchange1Mt();
        }
        else if (x == 3) // random array
        {
            RandomMtfill();
            Res[i] = exchange1Mt();
        }
    }
}// функція для виконання вимірів часу для векторного масиву методом сортування обміну 1
void exchange_measurement1Vt(int x)
{
    for (int i = 0; i < measurements_number; i++)
    {
        if (x == 1) // ordered array
        {
            OrderedVtfill();
            Res[i] = exchange1Vt();
        }
        else if (x == 2) // back-ordered array
        {
            Back_orderedVtfill();
            Res[i] = exchange1Vt();
        }
        else if (x == 3) // random array
        {
            RandomVtfill();
            Res[i] = exchange1Vt();
        }
    }
}
```

## *Main*

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
// імпорт модулів, які використовуватимуться в мейні
#include "common.h"
#include "Mfill.h"
#include "Vfill.h"
#include "Vsort.h"
#include "Msort.h"
#include "Measurement.h"
void GotoXY(int X, int Y)//функція для постановки курсора в консольному вікні за координатами (X, Y).
{
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD coord = { X, Y };
    SetConsoleCursorPosition(hStdOut, coord);
}
void colour(int i)//функція для задання кольору виведення наступних символів
{
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(hStdOut,i);
```

```
}
void ConsoleCursorVisible(int show, short size)//функція, що функція
керує видимістю курсора в консольному вікні та його розміром
{
    HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO structCursorInfo;
    GetConsoleCursorInfo(hStdOut, &structCursorInfo);
    structCursorInfo.bVisible = show;
    structCursorInfo.dwSize = size;
    SetConsoleCursorInfo(hStdOut, &structCursorInfo);
}
void submenu( int q)//підменю для вибору виду заповнення масиву
{
    system("cls");
    ConsoleCursorVisible(0, 100);
    const char* submenuOptions[] = {"Ordered", "Back-ordered",
"Random", "Back to main menu"};
    int numSubOptions = sizeof(submenuOptions) /
sizeof(submenuOptions[0]);
    int active_submenu = 0;
    char key;
    float time;
    while (1)
    {
        colour(10);
        GotoXY(33, 12);
        printf("Select the next option from this list:\n");
        GotoXY(35, 13);
        printf("Then press enter on the keyboard.\n");
        colour(9);
        int x = 45, y = 17;
        GotoXY(x, y);

        for (int i = 0; i < numSubOptions; i++)
        {
            GotoXY(x, y++);
            if (i == active_submenu)
            {
                colour(11);
                printf("* %s * ", submenuOptions[i]);
            }
            else
                printf("  %s   ", submenuOptions[i]);
            colour(9);
        }

        key = _getch();
        if (key == -32) key = _getch();
        switch (key)
        {
        case 27:
            exit(0);
        case 72:
            if (active_submenu > 0)
                active_submenu--;
            break;
        case 80:
```

```
            if (active_submenu < numSubOptions - 1)
                active_submenu++;
            break;
        case 13:
            switch (active_submenu)
            {
            case 0:
                system("CLS");
                if(q==1)
                {
                    colour(10);
                    printf("You chosed ordered array and selection3
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    OrderedMtfill();//заповнення
                    OutputMt();//виведення
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection3Mt ();//сортування
                    OutputMt();//виведення
                    colour(10);
                    selection_measurement3Mt(1);//час сортування
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    OrderedVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection3Vt ();
                    OutputVt();
                    colour(10);
                    selection_measurement3Vt(1);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                else if(q==2)
                {
                    colour(10);
                    printf("You chosed ordered array and selection5
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    OrderedMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection5Mt ();
                    OutputMt();
                    colour(10);
```

```
                    selection_measurement5Mt(1);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    OrderedVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection5Vt ();
                    OutputVt();
                    colour(10);
                    selection_measurement5Vt(1);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                else if(q==3)
                {
                    colour(10);
                    printf("You chosed ordered array and exchange1
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    OrderedMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    exchange1Mt ();
                    OutputMt();
                    colour(10);
                    exchange_measurement1Mt(1);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    OrderedVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    exchange1Vt ();
                    colour(7);
                    OutputVt();
                    colour(10);
                    exchange_measurement1Vt(1);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                getch();
                system("CLS");
                break;
            case 1:
```

```
                colour(11);
                system("CLS");
                if(q==1)
                {
                    colour(10);
                    printf("You chosed back-ordered array and
selection3 sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    Back_orderedMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection3Mt ();
                    OutputMt();
                    colour(10);
                    selection_measurement3Mt(2);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    Back_orderedVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    selection3Vt ();
                    colour(7);
                    OutputVt();
                    colour(10);
                    selection_measurement3Vt(2);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                else if(q==2)
                {
                    colour(10);
                    printf("You chosed back-ordered array and
selection5 sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    Back_orderedMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection5Mt ();
                    OutputMt();
                    colour(10);
                    selection_measurement5Mt(2);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
```

```
                    Back_orderedVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    selection5Vt ();
                    colour(7);
                    OutputVt();
                    colour(10);
                    selection_measurement5Vt(2);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                else if(q==3)
                {
                    colour(10);
                    printf("You chosed back-ordered array and exchange1
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    Back_orderedMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    exchange1Mt ();
                    OutputMt();
                    colour(10);
                    exchange_measurement1Mt(2);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    Back_orderedVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    exchange1Vt ();
                    colour(7);
                    OutputVt();
                    colour(10);
                    exchange_measurement1Vt(2);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                getch();
                system("CLS");
                break;
            case 2:
                colour(11);
                system("CLS");
                if(q==1)
                {
                    colour(10);
```

```
                    printf("You chosed random array and selection3
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    RandomMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection3Mt ();
                    OutputMt();
                    colour(10);
                    selection_measurement3Mt(3);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    RandomVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    selection3Vt ();
                    colour(7);
                    OutputVt();
                    colour(10);
                    selection_measurement3Vt(3);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                else if(q==2)
                {
                    colour(10);
                    printf("You chosed random array and selection5
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    RandomMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    selection5Mt ();
                    OutputMt();
                    colour(10);
                    selection_measurement5Mt(3);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    RandomVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    selection5Vt ();
```

```
                    colour(7);
                    OutputVt();
                    colour(10);
                    selection_measurement5Vt(3);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                else if(q==3)
                {
                    colour(10);
                    printf("You chosed random array and exchange1
sorting\n");
                    printf("3D array before:\n");
                    colour(7);
                    RandomMtfill();
                    OutputMt();
                    colour(10);
                    printf("After sorting:\n");
                    colour(7);
                    exchange1Mt ();
                    OutputMt();
                    colour(10);
                    exchange_measurement1Mt(3);
                    time=MeasurementProcessing();
                    printf("  Array[%d][%d][%d], Time=
%f\n\n",P,M,N,time);
                    printf("Vector array before:\n");
                    colour(7);
                    RandomVtfill();
                    OutputVt();
                    colour(10);
                    printf("After sorting:\n");
                    exchange1Vt ();
                    colour(7);
                    OutputVt();
                    colour(10);
                    exchange_measurement1Vt(3);
                    time=MeasurementProcessing();
                    printf("Vector`s size= %d: Time= %f\n",V,time);
                    colour(7);
                }
                getch();
                system("CLS");
                break;
            case 3:
                return;
            }
            break;
        }
    }
}
void menu() //головне меню програми.
{
    system("cls");//очищення консолі перед демонстрацією меню
    ConsoleCursorVisible(0, 100);// встановлення видимості курсора
консолі
```

```
    const char* menuOptions[] = {"selection3","selection5
","exchange1","Table&Exit"};//опції меню
    int numOptions = sizeof(menuOptions) /
sizeof(menuOptions[0]);//кількість опцій
    int active_menu = 0;//початкова опція
    char key;//змінна для збереження введеного символу
    while (1)
    {
        colour(10);//визначення кольру для подальших виведень на екран
        GotoXY(31, 12);//виставлення курсора за координатами
        printf("To select any option, use two battons:\n");
        GotoXY(39, 13);
        printf("the up and down arrows.\n");
        GotoXY(35, 14);
        printf("Then press enter on the keyboard.\n");

        colour(9);
        int x = 45, y = 17;//початкові координати для виведення опцій
        GotoXY(x, y);
// виведення опцій меню з позначкою, яка активна
        for (int i = 0; i < numOptions; i++)
        {
            GotoXY(x, y++);
            if (i == active_menu)
            {
                colour(11);
                printf("* %s * ",  menuOptions[i]);//виведення активної
опції
            }
            else
                printf("  %s   ", menuOptions[i]);//виведення
неактивної опції
            colour(9);
        }
        key = _getch();//отримання введеного символа(натиснення клавіші
х дією)
        if (key == -32) key = _getch();//перевірка чи натиснулися
потрібні клавіші, якими можна керувати меню
        switch (key)//обробка символу
        {
        case 27://якщо натиснуто ESC
            exit(0);//вихід з програми
        case 72://якщо натиснуто стрілку вгору
            if (active_menu > 0)
                active_menu--;//активна опція зміщується на ту, що
вище, якщо це була не 0ва
            break;
        case 80://якщо натиснуто стрілку вниз
            if (active_menu < numOptions - 1)//активна опція зміщується
на ту, що нище, якщо це була не 3тя
                active_menu++;
            break;
        case 13://якщо натиснуто Enter, опція вибирається
            switch (active_menu)
            {
            case 0://якщо обрана 0ва опція:
```

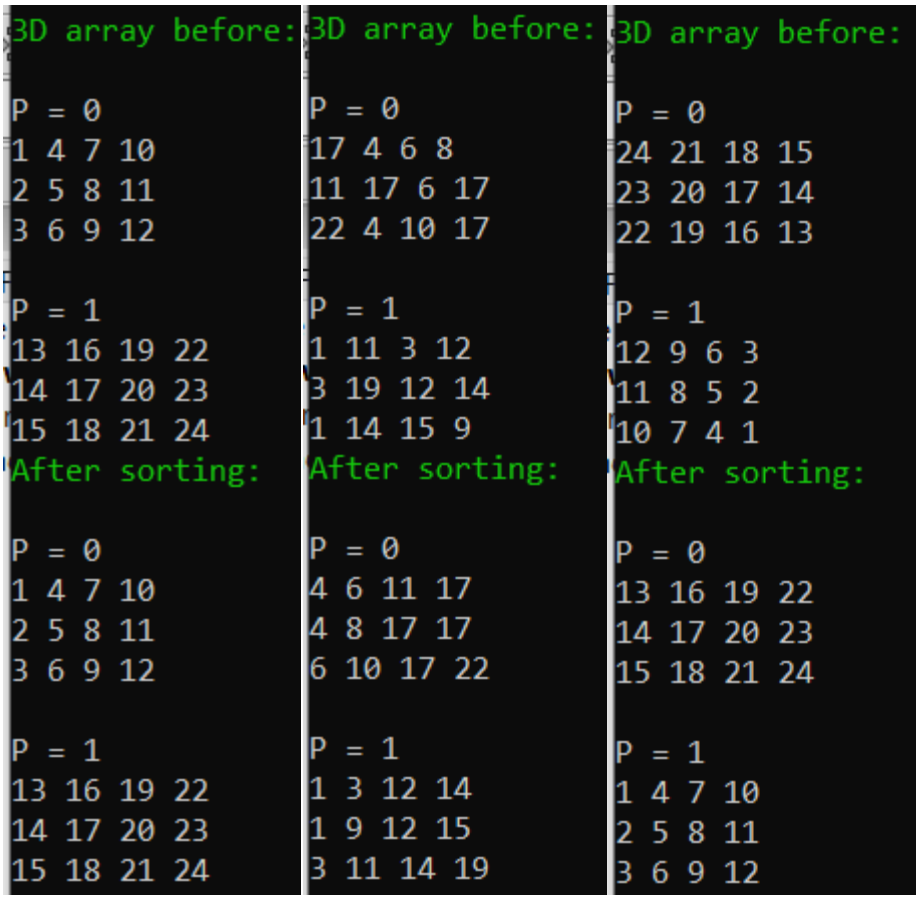
```
                system("CLS");//очищення консолі перед тим, як вивести
результат даної опції
                submenu(1);//виклик підменю з параметром 1, що
відповідає за сортування методом прямого вибору 3
                getch();//очікування натиску клавіші
                system("CLS");//очищення перед наступною опцією
                break;
            case 1://якщо обрана 1ша опція:
                system("CLS");
                submenu(2);//виклик підменю з параметром 1, що
відповідає за сортування методом прямого вибору 5
                getch();
                system("CLS");
                break;
            case 2://якщо обрана 2га опція:
                system("CLS");
                submenu(3);//виклик підменю з параметром 1, що
відповідає за сортування методом обміну 1
                getch();
                system("CLS");
                break;
            case 3://якщо обрана 3тя опція:
                colour(11);
                GotoXY(45, 24);
                system("CLS");
                Table ();//виклик функції, що виводить таблицю
результатів часу різних алгоритмів сортування при різних заповненням
                exit(0);//програма завершується
                break;
            }
            break;
        }
    }
}
int main()
{
    Mt = (long int***) malloc(P * sizeof(long int**));//виділення
пом'яті для кількості перерізів(одновимірний)
    for (long int k = 0; k < P; k++)
    {
        Mt[k] = (long int**) malloc(M * sizeof(long int*));//виділення
пам'яті для рядків(двовимірний)
        for (long int i = 0; i < M; i++)
            Mt[k][i] = (int*) malloc(N * sizeof(long int));//виділення
пам'яті для стовпців(вже став тривимірним масивом)
    }
    menu();//виклик функції, що відповідає за меню
    for (long int k = 0; k < P; k++)//звільнення пам'яті :
    {
        for (long int i = 0; i < M; i++)
            free(Mt[k][i]);//звільнення пам'яті кожного елемента
        free(Mt[k]);//звільнення пам'яті кожного перерізу
    }
    free(Mt);//звільнення пам'яті від масиву повністю
    return 0;
}
```

***Характеристики комп'ютера та компілятор:***
-процесор: Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz
-оперативна пам'ять: 16,0 ГБ
-операційна система: Windows 10 Education 22H2
-компілятор: GNU GCC Compiler

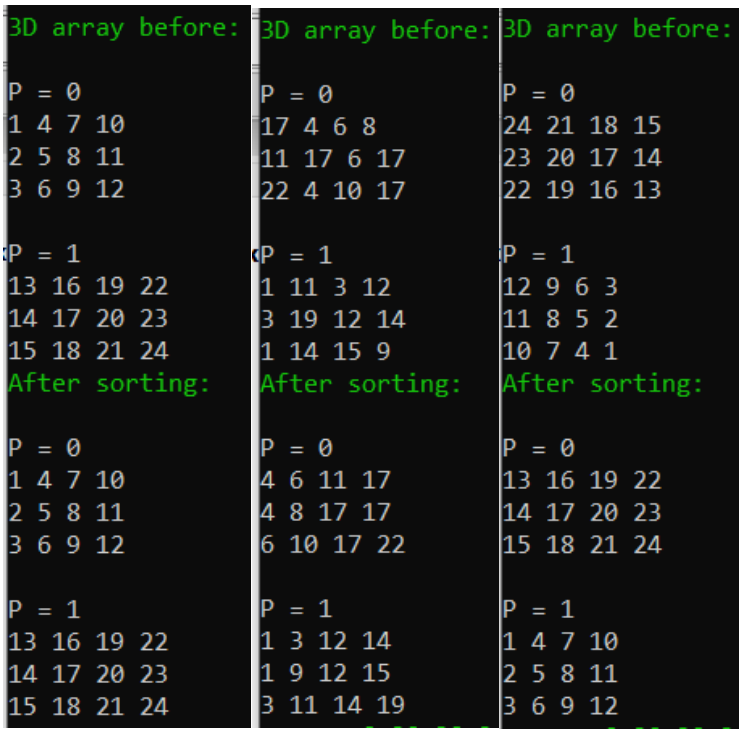
## *Тестування*

***Перевірка правильності сортування:***

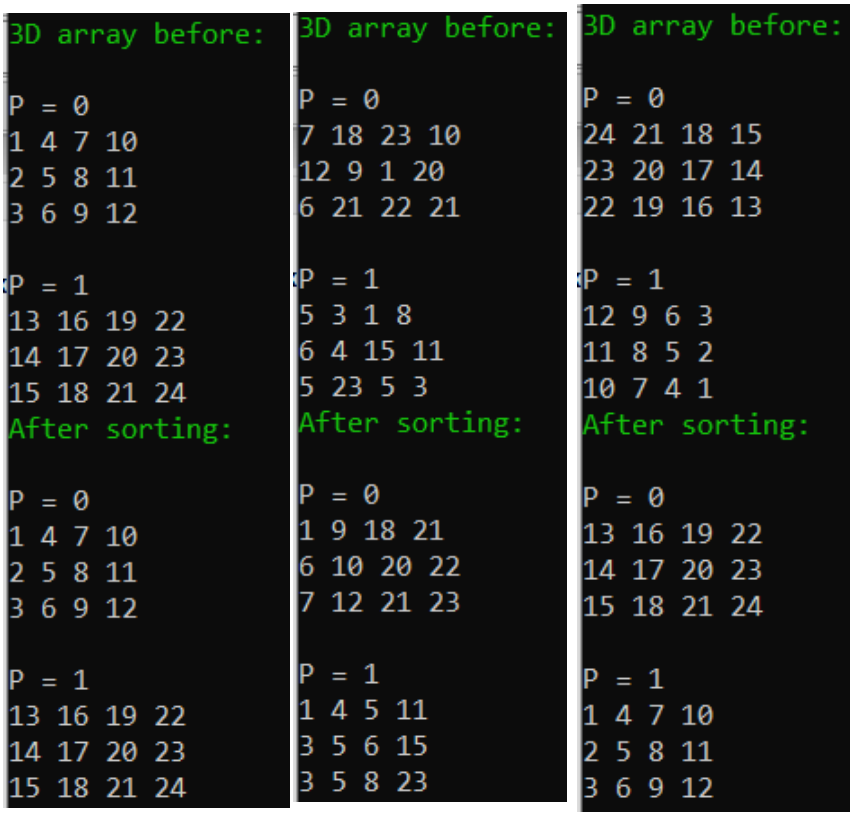
Алгоритм сортування прямого вибору №3




Алгоритм сортування прямого вибору №5




Алгоритм сортування обміну №1



Під час тестувань не було виявлено дифектів роботи програми, сортування відбулося правильно. Висновок: алгоритми сортують правильно.

### *Результати вимірів роботи алгоритмів*

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

P = const = 3

M = var

N = var

M*N = const = 65536

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

1) Mt[3][16][4096]:

Table for the array: P = 3, M = 16, N = 4096

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 14981.700195 | 10573.150391 | 10595.250000 |
| Selection5 | 14569.349609 | 14578.700195 | 18439.199219 |
| Exchange1 | 17831.400391 | 53530.500000 | 47973.750000 |

2) Mt[3][32][2048]:

Table for the array: P = 3, M = 32, N = 2048

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 14901.700195 | 9951.000000 | 11345.049805 |
| Selection5 | 12938.200195 | 13080.799805 | 17949.199219 |
| Exchange1 | 18694.750000 | 52232.699219 | 47302.148438 |

3) Mt[3][64][1024]:

Table for the array: P = 3, M = 64, N = 1024

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 16174.000000 | 10518.700195 | 11710.000000 |
| Selection5 | 12518.150391 | 12706.099609 | 18176.900391 |
| Exchange1 | 20747.550781 | 55189.050781 | 52952.750000 |

4) Mt[3][128][512]:

Table for the array: P = 3, M = 128, N = 512

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 15687.000000 | 10054.599609 | 11304.950195 |
| Selection5 | 12211.700195 | 12201.349609 | 17400.699219 |
| Exchange1 | 19897.750000 | 53614.800781 | 50425.949219 |

5) Mt[3][256][256]:

Table for the array: P = 3, M = 256, N = 256

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 15289.549805 | 9528.099609 | 10915.250000 |
| Selection5 | 11440.099609 | 11485.450195 | 16839.000000 |
| Exchange1 | 18806.449219 | 52268.851563 | 49679.449219 |

6) Mt[3][512][128]:

Table for the array: P = 3, M = 512, N = 128

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 15751.450195 | 10318.849609 | 11520.349609 |
| Selection5 | 11968.450195 | 11985.349609 | 17428.849609 |
| Exchange1 | 20385.150391 | 53815.648438 | 50650.800781 |

7) Mt[3][1024][64]:

Table for the array: P = 3, M = 1024, N = 64

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 15977.250000 | 10516.549805 | 11673.150391 |
| Selection5 | 12099.549805 | 12127.099609 | 17633.150391 |
| Exchange1 | 20819.849609 | 54144.750000 | 51478.148438 |

8) Mt[3][2048][32]:

Table for the array: P = 3, M = 2048, N = 32

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 17436.650391 | 11830.349609 | 12747.849609 |
| Selection5 | 13217.799805 | 13216.400391 | 19352.500000 |
| Exchange1 | 23446.699219 | 56946.300781 | 55447.148438 |

9) Mt[3][4096][16]:

Table for the array: P = 3, M = 4096, N = 16

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 17350.750000 | 11673.549805 | 12784.950195 |
| Selection5 | 14709.400391 | 14767.299805 | 20179.650391 |
| Exchange1 | 23199.750000 | 56693.300781 | 56437.148438 |

10) Vt[65536]  V=M*N:

Table for the vector: V = 65536

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 2173.000000 | 1833.199951 | 1991.199951 |
| Selection5 | 3926.949951 | 3941.149902 | 3869.500000 |
| Exchange1 | 4530.649902 | 9592.799805 | 5651.000000 |

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

P = var

M = N = const = 100,  M*N=10000

Час сортування вектора довжиною NV = M*N (дорівнює кількості елементів у перерізі) помножити на P. Вимір і порівняння для вектора достатньо зробити тільки для найбільшого P

1) Mt[2][100][100]:

Table for the array: P = 2, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 228.649994 | 138.000000 | 161.649994 |
| Selection5 | 177.000000 | 182.350006 | 251.399994 |
| Exchange1 | 271.899994 | 724.650024 | 731.400024 |

2) Mt[4][100][100]:

Table for the array: P = 4, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 456.049988 | 274.649994 | 323.799988 |
| Selection5 | 353.700012 | 357.049988 | 499.950012 |
| Exchange1 | 542.349976 | 1444.699951 | 1463.250000 |

3) Mt[8][100][100]:

Table for the array: P = 8, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 916.500000 | 552.450012 | 649.549988 |
| Selection5 | 711.549988 | 719.599976 | 1005.599976 |
| Exchange1 | 1089.050049 | 2902.350098 | 2942.050049 |

4) Mt[16][100][100]:

Table for the array: P = 16, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 1799.550049 | 1083.099976 | 1273.050049 |
| Selection5 | 1397.949951 | 1412.199951 | 1974.750000 |
| Exchange1 | 2133.250000 | 5699.350098 | 5755.350098 |

5) Mt[32][100][100]:

Table for the array: P = 32, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 3619.250000 | 2203.949951 | 2555.750000 |
| Selection5 | 2807.100098 | 2827.949951 | 3958.649902 |
| Exchange1 | 4278.649902 | 11417.950195 | 11544.500000 |

6) Mt[64][100][100]:

Table for the array: P = 64, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 7279.200195 | 4384.200195 | 5159.649902 |
| Selection5 | 5656.549805 | 5711.549805 | 7971.750000 |
| Exchange1 | 8636.849609 | 23062.099609 | 23318.800781 |

7) Mt[128][100][100]:

Table for the array: P = 128, M = 100, N = 100

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 14588.200195 | 8785.700195 | 10322.250000 |
| Selection5 | 11134.650391 | 11246.700195 | 15735.049805 |
| Exchange1 | 16993.900391 | 45400.851563 | 46584.750000 |

8) Vt[10000]:

Table for the vector: V = 10000

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 51.000000 | 43.299999 | 47.000000 |
| Selection5 | 92.349998 | 92.500000 | 91.800003 |
| Exchange1 | 106.750000 | 184.600006 | 134.449997 |

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

P = const = 3  M = N = var  M*N = var

1) Mt[3][16][16]  (M*N=256):

Table for the array: P = 3, M = 16, N = 16

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 0.200000 | 0.100000 | 0.100000 |
| Selection5 | 0.150000 | 0.150000 | 0.200000 |
| Exchange1 | 0.200000 | 0.850000 | 0.750000 |

2) Mt[3][32][32]  (M*N=1024):

Table for the array: P = 3, M = 32, N = 32

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 3.750000 | 2.400000 | 2.750000 |
| Selection5 | 3.000000 | 3.150000 | 4.100000 |
| Exchange1 | 4.450000 | 11.150000 | 11.150000 |

3) Mt[3][48][48]  (M*N=2304):

Table for the array: P = 3, M = 48, N = 48

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 19.450001 | 12.400000 | 13.850000 |
| Selection5 | 15.600000 | 15.700000 | 20.900000 |
| Exchange1 | 22.250000 | 56.700001 | 57.900002 |

4) Mt[3][64][64]  (M*N=4096):

Table for the array: P = 3, M = 64, N = 64

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 57.799999 | 35.200001 | 40.950001 |
| Selection5 | 45.200001 | 46.400002 | 63.700001 |
| Exchange1 | 68.400002 | 176.000000 | 180.000000 |

5) Mt[3][128][128]  (M*N=16384):

Table for the array: P = 3, M = 128, N = 128

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 908.200012 | 545.799988 | 642.799988 |
| Selection5 | 700.450012 | 704.349976 | 993.450012 |
| Exchange1 | 1081.300049 | 2970.699951 | 2917.800049 |

6) Mt[3][192][192]  (M*N=36864):

Table for the array: P = 3, M = 192, N = 192

| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 4759.049805 | 2856.800049 | 3368.500000 |
| Selection5 | 3625.449951 | 3638.500000 | 5177.450195 |
| Exchange1 | 5701.399902 | 15444.150391 | 14781.849609 |

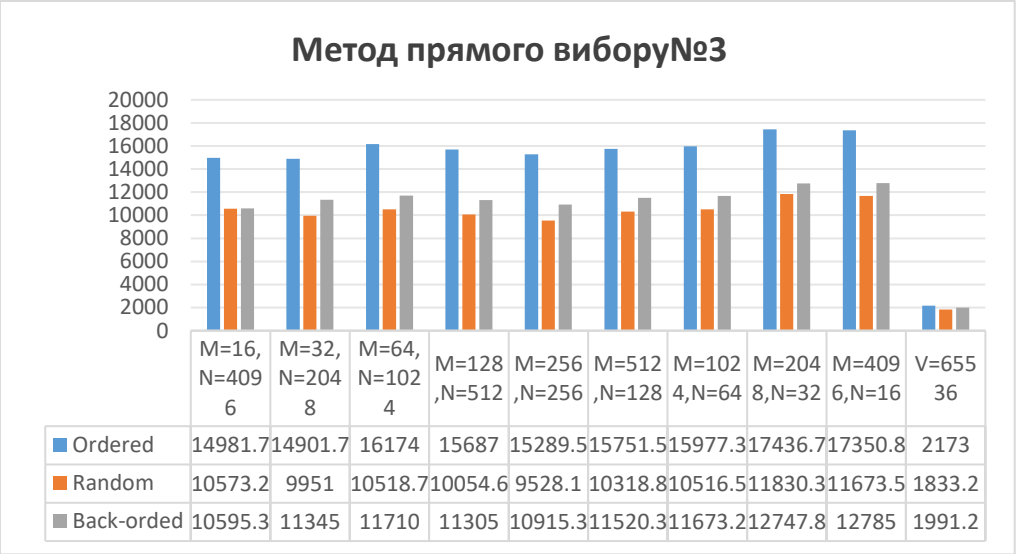
7) Mt[3][256][256]  (M*N=65536):

Table for the array: P = 3, M = 256, N = 256

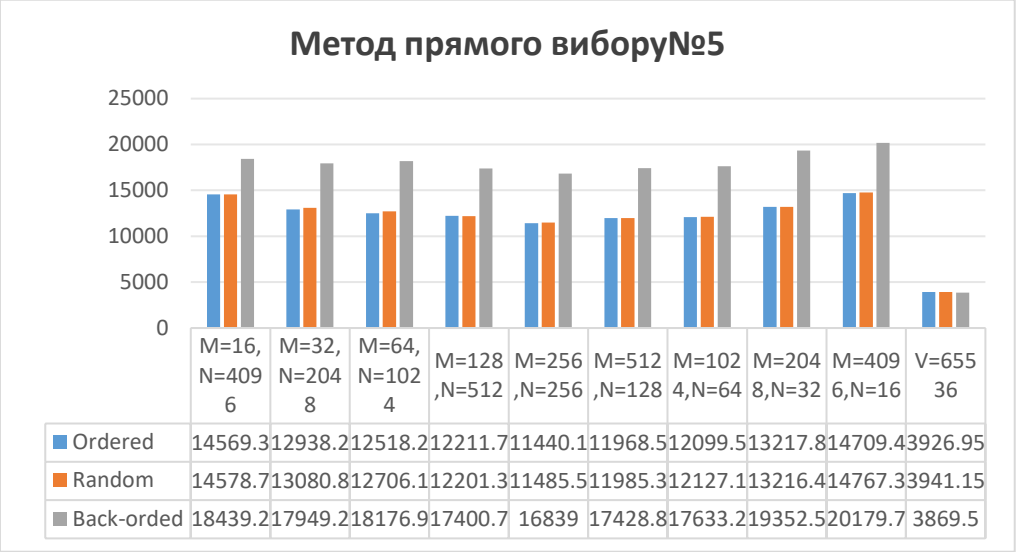
| | Ordered | Random | Back-ordered |
|---|---|---|---|
| Selection3 | 15141.250000 | 9477.700195 | 10919.700195 |
| Selection5 | 11606.500000 | 11631.400391 | 16710.650391 |
| Exchange1 | 18871.300781 | 51801.949219 | 48930.000000 |

## *Подання даних графіками*

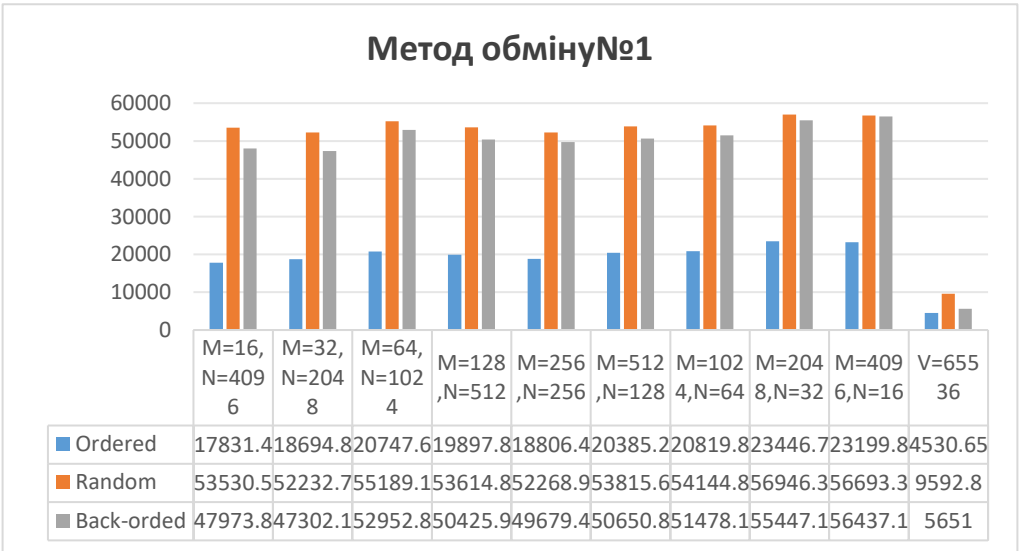
Дослідження1



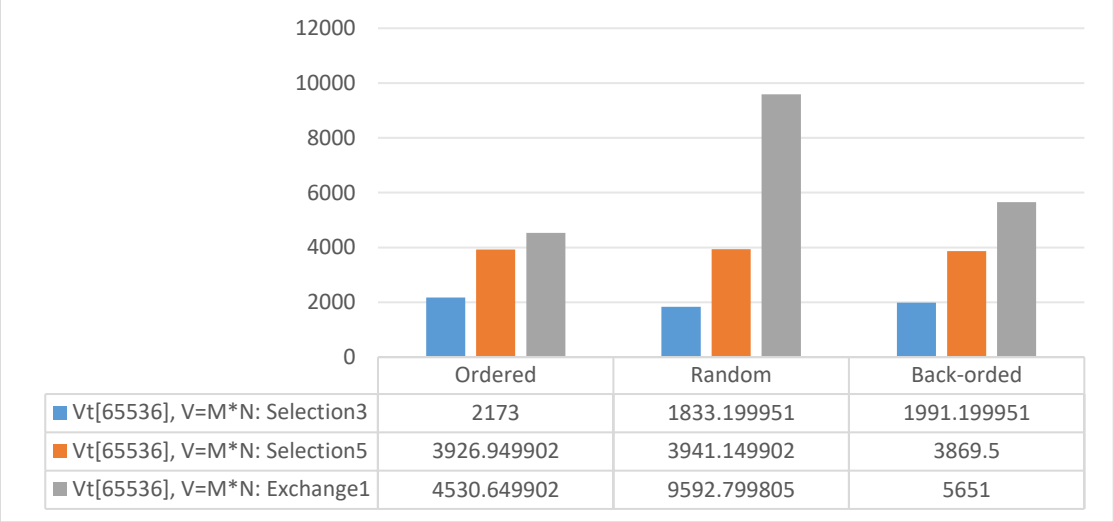

| | M=16, N=4096 | M=32, N=2048 | M=64, N=1024 | M=128 ,N=512 | M=256 ,N=256 | M=512 ,N=128 | M=1024,N=64 | M=2048,N=32 | M=4096,N=16 | V=65536 |
|---|---|---|---|---|---|---|---|---|---|---|
| Ordered | 14981.7 | 14901.7 | 16174 | 15687 | 15289.5 | 15751.5 | 15977.3 | 17436.7 | 17350.8 | 2173 |
| Random | 10573.2 | 9951 | 10518.7 | 10054.6 | 9528.1 | 10318.8 | 10516.5 | 11830.3 | 11673.5 | 1833.2 |
| Back-orded | 10595.3 | 11345 | 11710 | 11305 | 10915.3 | 11520.3 | 11673.2 | 12747.8 | 12785 | 1991.2 |




| | M=16, N=4096 | M=32, N=2048 | M=64, N=1024 | M=128 ,N=512 | M=256 ,N=256 | M=512 ,N=128 | M=1024,N=64 | M=2048,N=32 | M=4096,N=16 | V=65536 |
|---|---|---|---|---|---|---|---|---|---|---|
| Ordered | 14569.3 | 12938.2 | 12518.2 | 12211.7 | 11440.1 | 11968.5 | 12099.5 | 13217.8 | 14709.4 | 3926.95 |
| Random | 14578.7 | 13080.8 | 12706.1 | 12201.3 | 11485.5 | 11985.3 | 12127.1 | 13216.4 | 14767.3 | 3941.15 |
| Back-orded | 18439.2 | 17949.2 | 18176.9 | 17400.7 | 16839 | 17428.8 | 17633.2 | 19352.5 | 20179.7 | 3869.5 |

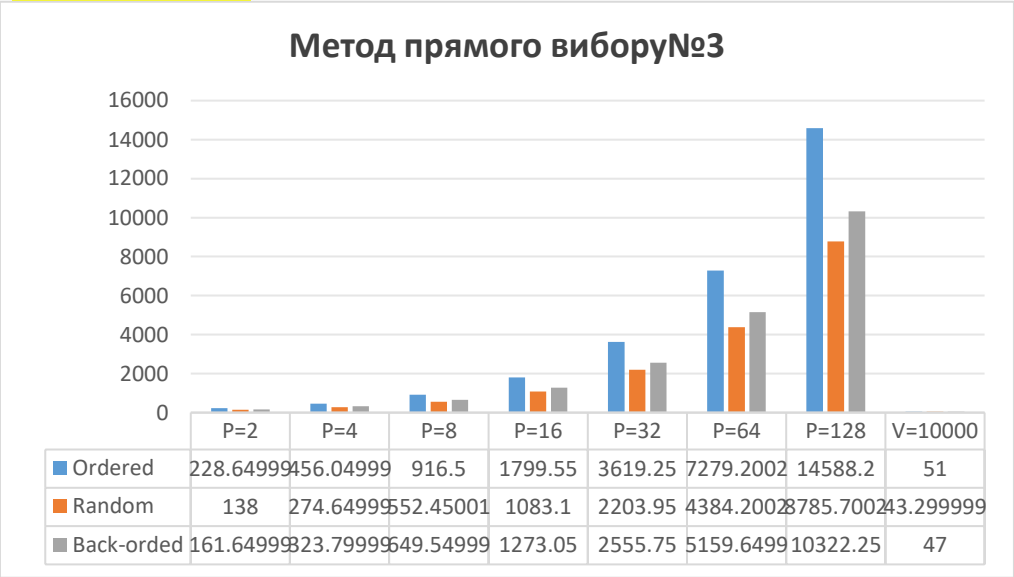



| | M=16, N=4096 | M=32, N=2048 | M=64, N=1024 | M=128 ,N=512 | M=256 ,N=256 | M=512 ,N=128 | M=1024,N=64 | M=2048,N=32 | M=4096,N=16 | V=65536 |
|---|---|---|---|---|---|---|---|---|---|---|
| Ordered | 17831.4 | 18694.8 | 20747.6 | 19897.8 | 18806.4 | 20385.2 | 20819.8 | 23446.7 | 23199.8 | 4530.65 |
| Random | 53530.5 | 52232.7 | 55189.1 | 53614.8 | 52268.9 | 53815.6 | 54144.8 | 56946.3 | 56693.3 | 9592.8 |
| Back-orded | 47973.8 | 47302.1 | 52952.8 | 50425.9 | 49679.4 | 50650.8 | 51478.1 | 55447.1 | 56437.1 | 5651 |

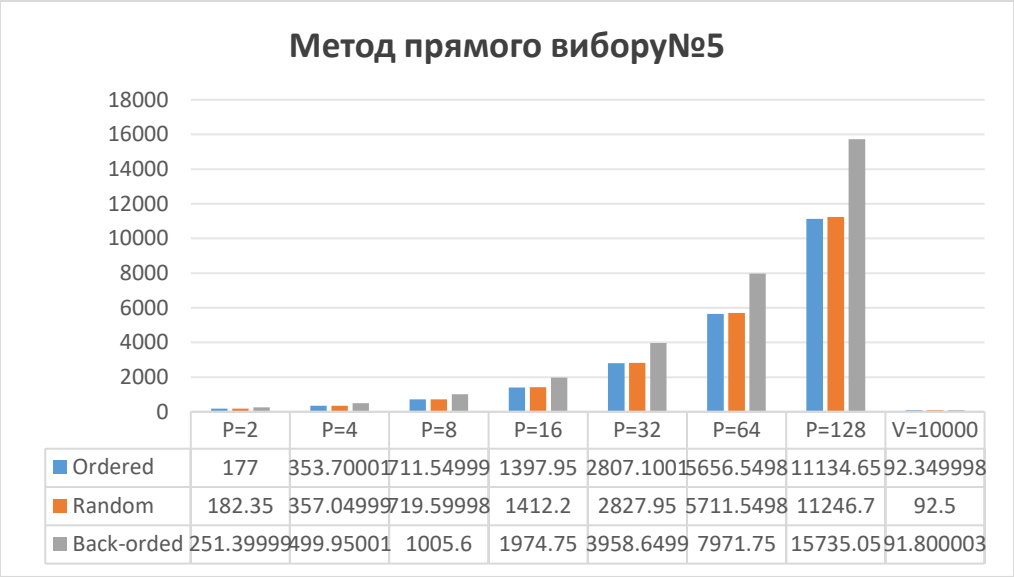



| | Ordered | Random | Back-orded |
|---|---|---|---|
| Vt[65536], V=M*N: Selection3 | 2173 | 1833.199951 | 1991.199951 |
| Vt[65536], V=M*N: Selection5 | 3926.949902 | 3941.149902 | 3869.5 |
| Vt[65536], V=M*N: Exchange1 | 4530.649902 | 9592.799805 | 5651 |

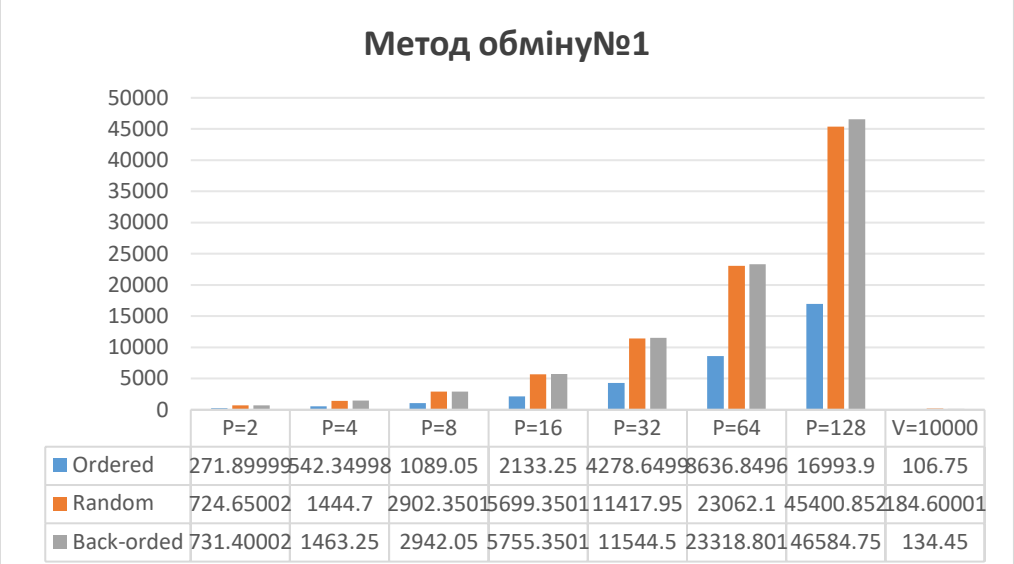
Дослідження2



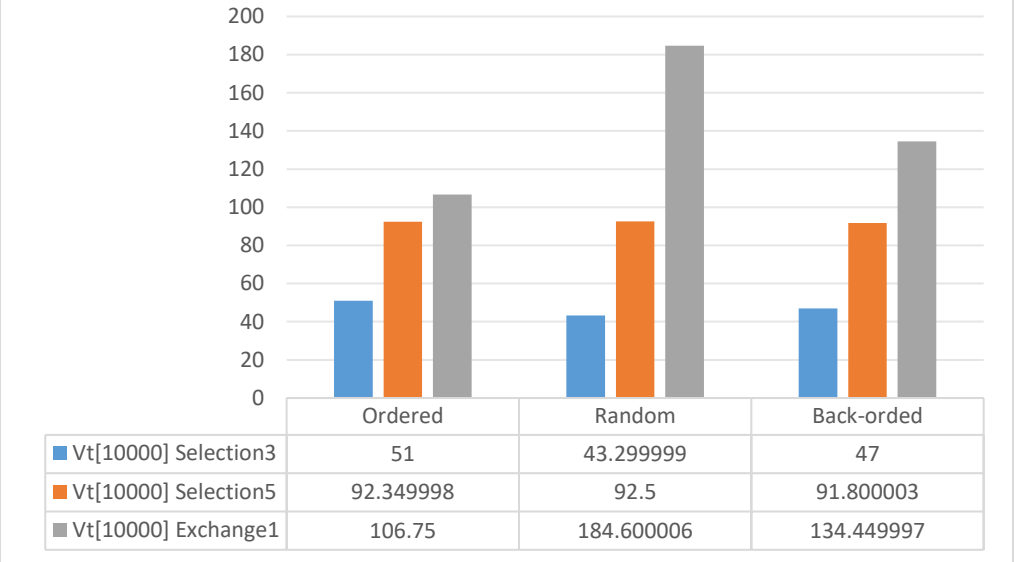

| | P=2 | P=4 | P=8 | P=16 | P=32 | P=64 | P=128 | V=10000 |
|---|---|---|---|---|---|---|---|---|
| Ordered | 228.64999 | 456.04999 | 916.5 | 1799.55 | 3619.25 | 7279.2002 | 14588.2 | 51 |
| Random | 138 | 274.64999 | 552.45001 | 1083.1 | 2203.95 | 4384.2002 | 8785.7002 | 43.299999 |
| Back-orded | 161.64999 | 323.79999 | 649.54999 | 1273.05 | 2555.75 | 5159.6499 | 10322.25 | 47 |




| | P=2 | P=4 | P=8 | P=16 | P=32 | P=64 | P=128 | V=10000 |
|---|---|---|---|---|---|---|---|---|
| Ordered | 177 | 353.70001 | 711.54999 | 1397.95 | 2807.1001 | 5656.5498 | 11134.65 | 92.349998 |
| Random | 182.35 | 357.04999 | 719.59998 | 1412.2 | 2827.95 | 5711.5498 | 11246.7 | 92.5 |
| Back-orded | 251.39999 | 499.95001 | 1005.6 | 1974.75 | 3958.6499 | 7971.75 | 15735.05 | 91.800003 |

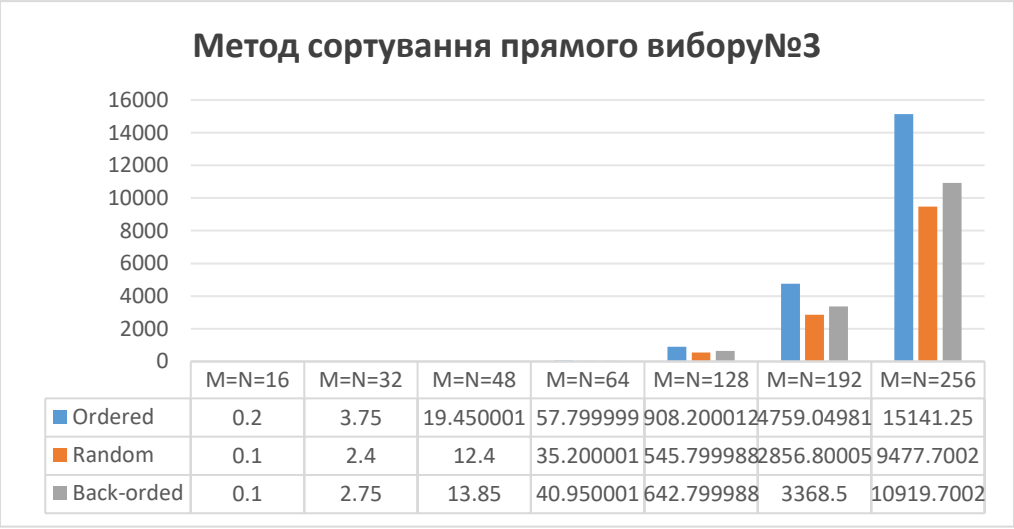



| | P=2 | P=4 | P=8 | P=16 | P=32 | P=64 | P=128 | V=10000 |
|---|---|---|---|---|---|---|---|---|
| Ordered | 271.89999 | 542.34998 | 1089.05 | 2133.25 | 4278.6499 | 8636.8496 | 16993.9 | 106.75 |
| Random | 724.65002 | 1444.7 | 2902.3501 | 5699.3501 | 11417.95 | 23062.1 | 45400.852 | 184.60001 |
| Back-orded | 731.40002 | 1463.25 | 2942.05 | 5755.3501 | 11544.5 | 23318.801 | 46584.75 | 134.45 |

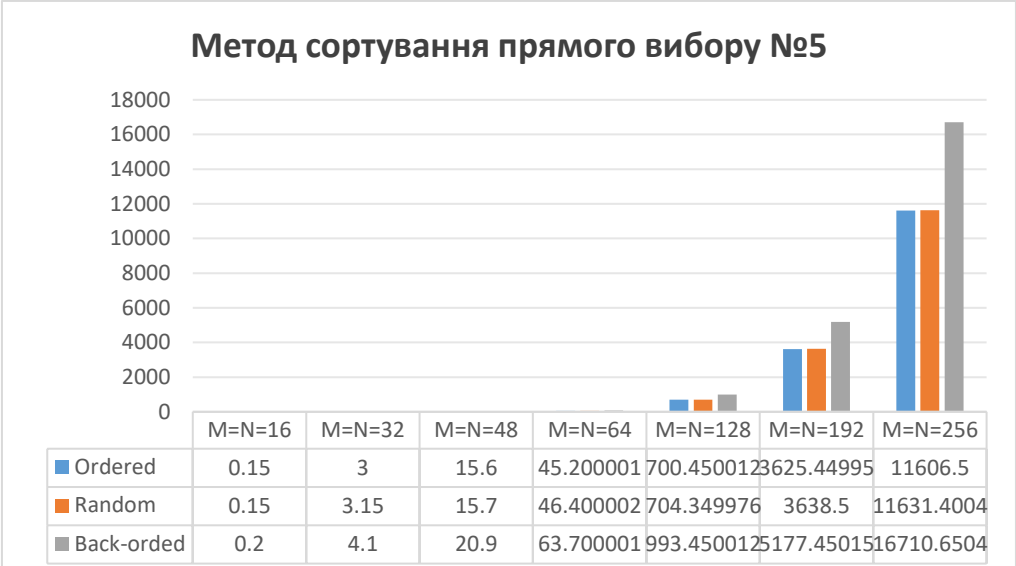



| | Ordered | Random | Back-orded |
|---|---|---|---|
| Vt[10000] Selection3 | 51 | 43.299999 | 47 |
| Vt[10000] Selection5 | 92.349998 | 92.5 | 91.800003 |
| Vt[10000] Exchange1 | 106.75 | 184.600006 | 134.449997 |

Дослідження3




| | M=N=16 | M=N=32 | M=N=48 | M=N=64 | M=N=128 | M=N=192 | M=N=256 |
|---|---|---|---|---|---|---|---|
| Ordered | 0.2 | 3.75 | 19.450001 | 57.799999 | 908.200012 | 4759.04981 | 15141.25 |
| Random | 0.1 | 2.4 | 12.4 | 35.200001 | 545.799988 | 2856.80005 | 9477.7002 |
| Back-orded | 0.1 | 2.75 | 13.85 | 40.950001 | 642.799988 | 3368.5 | 10919.7002 |




| | M=N=16 | M=N=32 | M=N=48 | M=N=64 | M=N=128 | M=N=192 | M=N=256 |
|---|---|---|---|---|---|---|---|
| Ordered | 0.15 | 3 | 15.6 | 45.200001 | 700.450012 | 3625.44995 | 11606.5 |
| Random | 0.15 | 3.15 | 15.7 | 46.400002 | 704.349976 | 3638.5 | 11631.4004 |
| Back-orded | 0.2 | 4.1 | 20.9 | 63.700001 | 993.450012 | 5177.45015 | 16710.6504 |

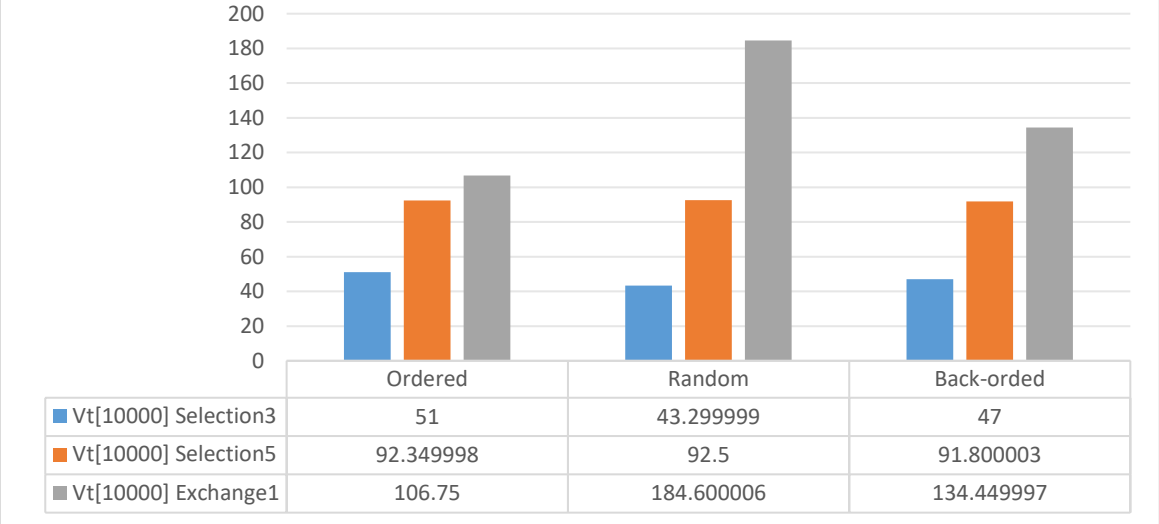

| | M=N=16 | M=N=32 | M=N=48 | M=N=64 | M=N=128 | M=N=192 | M=N=256 |
|---|---|---|---|---|---|---|---|
| Ordered | 0.2 | 4.45 | 22.25 | 68.400002 | 1081.30005 | 5701.3999 | 18871.3008 |
| Random | 0.85 | 11.15 | 56.700001 | 176 | 2970.69995 | 15444.1504 | 51801.9492 |
| Back-orded | 0.75 | 11.15 | 57.900002 | 180 | 2917.80005 | 14781.8496 | 48930 |

## *Аналіз даних*

Зважаючи на 86-ий варіант курсової роботи, при вирішенні задачі№1, а саме було впорядкувано окремо кожен переріз тривимірного наскрізно по стовпчикахта одновимірний масиви за незменшенням, мною було пристосовано, досліджено та протестовано алгоритми:

- сортування методу прямого вибору№3;
- сортування методу прямого вибору№5;
- сортування методу обміну№1;

***Порівняння властивостей алгоритмів при сортуванні одновимірних масивів***




| | Ordered | Random | Back-orded |
|---|---|---|---|
| Vt[10000] Selection3 | 51 | 43.299999 | 47 |
| Vt[10000] Selection5 | 92.349998 | 92.5 | 91.800003 |
| Vt[10000] Exchange1 | 106.75 | 184.600006 | 134.449997 |

*1) Алгоритм сортування методу прямого вибору №3:*

Даний алгоритм є вдосконаленішим в порівнянні з класичною версією,а дже при пошуці мінімума, одночасно оновлювався і максимум,також при кожній ітерації на своє місце стають відразу 2 елементи, що й скорочує роботу сортування. Коли масив заповнений впорядкованими числами, то при кожній ітерації міняються перший та останній елементи невідсортованої частини, хоч вони на своїх місцях, тому час витрачається на зайві порівняння та обміни. Коли масив заповнений випадковими числами, такої проблеми нема( рідко можна натрапити на елемент, що є вже на своєму місці) тому кількість порівнянь зменшується. Коли масив заповнений оберннено впорядкованими числами, це ускладнює роботу, але не настільки, коли масив впорядкований. Практично підтверджено, що цей алгоритм оптимізований відносно класичного.

2) *Алгоритм сортування методу прямого вибору №5:*

Даний алгоритм дуже схожий на класичний метод прямого вибору, що має квадратичну характеристику, хоч і, коли масив заповнений

випадковими числами, кількість присвоєть є логарифмічною, це є характерим і для алгоритму сорування методу прямого вибору№5. Цю властивість можна побачити, проаналізувавши 2 рядок діаграми, і зробити висновок, що час впорядковано заповненого масиву, заповненого масиву випадковими числами і обернено впорядково заповненого масив ± однакові, але перші 2 все ж таки займають більше часу. Цей алгоритм проходиться через всі елементи вектора за раз, при цьому йде пошук мінімума і його перестановка на перше місце невідсортованої частини. Цей підхід скорочує час сортування впрорядковано заповненого і заповненого випадковими числами масивів, але не дуже впливає на обернено впорядкований масив. Практично підтверджено, що така зміна не дуже впливає на оптимізованість алгоритму для масивів середнього та великого розміру в порівнянні з класичним.

*3)Алгоритм сортування методу обміну №1:*

Даний алгоритм - це класичний метод сортування обміну. Його основною ідеєю є послідовне проходження по елементах та порівняння сусідніх, якщо вони відповідають умові(за якою потрібно відсортувати масив, наприклад: за незменшенням чи незбільшенням)то алгоритм переходить до 2 наступних елементів, включаючи останній з них. Найшвидше сортується впорядкований масив, адже перестановки не потрібно робити, наступний вид заповнення по швидкості є обернено впорядкований масив, адже невідсортовані елементи, що повинні буди відсортовані всеодно стоять поряд, тому кількість обмінів не дуже велика, що є відмінним від заповненого випадковими числами масиву, адже там в хаотичному порядку розташовані елементи, що повинні стояти поряд після сортування

***Залежність часу роботи алгоритмів від форми перерізу масива.***

Помітна залежність в цьому, адже чим більша кількість рядків у перерізі, в результаті більше часу буде затрачено на сортування даного масиву. Це можна простежити в усіх трьох алгоритмів та заповненнях.

***Залежність часу роботи алгоритмів від кількості перерізів масива***

Помітна тенденція при другому дослідженні в тому, що при вищій кількості перерізів масиву, він значно довше сортується.

***Залежність часу роботи алгоритмів від розміру перерізів масиву***

Помітна тенденція, що чим більший переріз, тоді й час сортування перевищує в кілька деятків, а то й тисяч разі порівнюючи найменший та найбільший масиви в перерізах.

## *Висновок*

Впродовж виконання курсової роботи було досліджено 3 алгоритми в різних випадках при різному заповненні. Кожен з них працював по різному, але концепція результатів була схожою.

Найбільш оптимізованим алгоритмом виявився метод прямого вибору№3, наступний по цій шкалі є метод прямого вибору№5, та найменш оптимальним варіантом є метод обміну№1.

От для методу прямого вибору №3 характерно найшвидше сортувати масиви із випадковими числами, а найдовше впорядковані масиви. Метод прямого вибору№5 найменше витрачає часу на сортування впорядкованого і випадкового вмісту. А метод обміну№1 найшвидше сортує впорядковані масиви, а інші 2 випадки приблизно на одному рівні.

Отже кожен алгоритм є унікальним, більш придатним для окремих випадків.

## *Список літератури*

1) Інструкції з курсової роботи
2) YouTube канал «Марченко Олександр Іванович»
3) Конспект СДА лекцій