

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: Голуб В.В.

Група: КВ-84

Номер залікової книжки: КВ-8411

Допущений до захисту

---

2 семестр 2018/2019

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних  
систем

Узгоджено

ЗАХИЩЕНА "\_\_\_" \_\_\_\_\_ 2019р.

Керівник роботи

з оцінкою \_\_\_\_\_

\_\_\_\_\_/Марченко О.І./

\_\_\_\_\_/Марченко О.І./

***Дослідження ефективності методів сортування  
алгоритм сортування №1 методом прямого  
вибору, алгоритм сортування №4 методом  
прямого вибору, гібридний алгоритм «вставка-  
обмін») на багатовимірних масивах***

Виконавець роботи:

Голуб Володимир Володимирович

\_\_\_\_\_ 2019 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 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 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 № 146**

### **Задача**

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

### **Досліджувані методи та алгоритми**

Алгоритм сортування №1 методу прямого вибору

Алгоритм сортування №4 методу прямого вибору

Гібридний алгоритм "вставка – обмін"

.

### **Способи обходу**

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

### **Випадки дослідження**

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

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

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

## Алгоритм сортування №1 методом прямого вибору

Принцип роботи алгоритму:

Припустимо що потрібно відсортувати одновимірний масив за не зменшенням.

Розділяємо умовно масив на відсортовану і невідсортовану частини.

Спершу вважаємо що весь масив є невідсортованою частиною. Далі виконуємо такі дії:

1. Вважаємо що перший елемент невідсортованої частини є найменший, записуємо його значення та місце знаходження в буфер.
2. Порівнюємо значення буферу з наступними елементами що стоять після першого елемента. При умові що серед інших елементів ми знаходимо менше число що записано у буфері, тоді ми у буфер запам'ятовуємо нове мінімальне число та його місце знаходження у буфер.
3. Продовжуємо пункт 2 поки не дійдемо до останнього елементу масив.
4. Після виконання пункту 3 ми міняємо місцями елементи перший та елемент який записаний у буфері. Після даної операції ми розділяємо умовно масив на відсортовану і невідсортовану частини. (Відсортована частина елемент що стоїть на першому місці масиву, все інше вважаємо не відсортованою)
5. Далі за найменший вважаємо перший елемент невідсортованої частини і повторюємо пункти 1 - 5 до поки не відсортуємо весь масив.

Ілюстрація:

A[N] ; N = 5      1 – відсортована частина      1- невідсортована частина

0	1	2	3	4	N-1 =5
3	5	1	8	7	4

Min < A[1];

Min	iMin
3	0

0	1	2	3	4	N-1 =5
3	5	1	8	7	4

Min < A[2]

Min	iMin
3	0

0	1	2	3	4	N-1 =5
3	5	1	8	7	4

Min < A[3]

Min	iMin
1	2

0	1	2	3	4	N-1 =5
3	5	1	8	7	4

Min	iMin
1	2

Min < A[4];

0	1	2	3	4	N-1 =5
3	5	1	8	7	4

Min	iMin
1	2

Min < A[5];

0	1	2	3	4	N-1 =5
3	5	1	8	7	4

Min	iMin
1	2



Виконуємо попередні дії з невідсортованою частиною поки не відсортуємо все масив

0	1	2	3	4	N-1 =5
<u>1</u>	5	3	8	7	4

Min	iMin
5	1

Алгоритм сортування мовою C:

```
clock_t Select1(int *A, int N)
{
    int Min, imin;
    clock_t time_start, time_stop;

    time_start = clock();

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

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 5. Алгоритм сортування №1 методу прямого вибору.

## Алгоритм сортування №4 методу прямого вибору

Принцип роботи алгоритму:

Задано одновимірний масив , потрібно відсортувати його за незменшенням

Весь масив умовно ділимо на дві частини: 1-відсортована , 2- невідсортована.

На початку сортування весь масив вважаємо не відсортованим. Межі цього масив визначаються так : L – ліва межа ( вважаємо що це є перший елемент масиву), R- права межа( вважаємо елемент який стоїть останній в масиві).

Далі відбувається сортування наступним чином:

1. На початку алгоритму вважаємо що найбільшим та найменшим є перший елемент масиву, далі виконуємо такі дії.
2. Знаходимо(порівнюючи з тими які записані на початку масиву) найбільший і найменший елемент масиву у невідсортованій частині, запам'ятовуємо їхні позиції.
3. Якщо знайдений мінімальний елемент не знаходиться на позиції L, тоді ми міняємо їх між собою місцями. За таким же принципом, якщо найбільший елемент не стоїть на позиції R , теж обмінюємо їх між собою місцями.
4. Пройшовши до кінця невідсортовану частину , і виконавши попередню дію , до відсортованої частини ми додаємо по одному елементу з ліворуч та праворуч, тобто ми зменшуємо невідсортовану частину з обох боків на один елемент( $L = L+1$ ;  $R = R-1$ )
5. Повторюємо пункти 1-4 доти доки ліва границя є меншою від правої ( $L < R$ )



Загальна схема роботи алгоритму:

$A[N]$  ;  $N = 5$       **1 – відсортована частина**      1- невідсортована частина

L = 0	1	2	3	4	R=N-1 =5
<b>3</b>	<b>5</b>	<b>1</b>	<b>8</b>	<b>7</b>	<b>4</b>

iMin	iMax
<b>0</b>	<b>0</b>

L = 0	1	2	3	4	R=N-1 =5
<b>3</b>	<b>5</b>	<b>1</b>	<b>8</b>	<b>7</b>	<b>4</b>

iMin	iMax
<b>2</b>	<b>3</b>

$L = L+1; R = R-1;$

0	L = 1	2	3	R = 4	N-1 =5
<u><b>1</b></u>	<b>5</b>	<b>3</b>	<b>4</b>	<b>7</b>	<u><b>8</b></u>

iMin	iMax
<b>1</b>	<b>1</b>

0	L = 1	2	3	R = 4	N-1 =5
<u><b>1</b></u>	<b>5</b>	<b>3</b>	<b>4</b>	<b>7</b>	<u><b>8</b></u>

iMin	iMax
<b>1</b>	<b>1</b>

0	L = 1	2	3	R = 4	N-1 =5
<u><b>1</b></u>	<b>5</b>	<b>3</b>	<b>4</b>	<b>7</b>	<u><b>8</b></u>

iMin	iMax
<b>2</b>	<b>4</b>

$L = L+1; R = R-1;$

0	1	L = 2	R = 3	4	N-1 =5
<u><b>1</b></u>	<u><b>3</b></u>	<b>5</b>	<b>4</b>	<u><b>7</b></u>	<u><b>8</b></u>

iMin	iMax
<b>3</b>	<b>2</b>

$L = L+1; R = R-1;$

$(L > R)$  – алгоритм відсортований умова не виконується для продовження.

0	1	R = 2	L = 3	4	N-1 =5
<u><b>1</b></u>	<u><b>3</b></u>	<u><b>4</b></u>	<u><b>5</b></u>	<u><b>7</b></u>	<u><b>8</b></u>

iMin	iMax
<b>3</b>	<b>2</b>

Алгоритм мовою С:

```
clock_t Select4(int *A, int N)
{
    int L, R, imin, imax, tmp;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        imin=L; imax=L;

        for(int i=L+1; i<R+1; i++)
            if (A[i]<A[imin]) imin=i;
            else
                if (A[i]>A[imax]) imax=i;

        tmp=A[imin];
        A[imin]=A[L];
        A[L]=tmp;
        if (imax==L) {
            tmp=A[imin];
            A[imin]=A[R];
            A[R]=tmp;
        }
        else {
            tmp=A[imax];
            A[imax]=A[R];
            A[R]=tmp;
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

---

Рис. 8. Алгоритм сортування №4 методу прямого вибору.

## Гібридний алгоритм "вставка – обмін"

Задано одновимірний масив, відсортувати його за незменшенням.

Принцип роботи даного гібридного алгоритму полягає в тому, що ,

1. Як і в звичайній вставці ми йдемо по елементно в данному випадку починаючи з другого елементу одновимірного масива (данна особливість впливає з тої особливості яку запозичили у алгоритму обміну) і аж до кінця.
2. Під час проходження по елементу ми кожен елемент перевіряємо умову чи він є не другим елементом масиву та умову чи він є більшим за попередній, якщо умова виконується ми міняємо їх між собою місцями (данна властивість взята з алгоритму обміну ).
3. Після цього ми перевіряємо цю умову йдучи вже в оберненому шляху починаючи від місця обміну в пункті 2 , якщо умова виконується міняємо, якщо умова не виконується то ми вертаємось до того місця від якого ми йшли у зворотньому шляху і продовжуємо іти по елементно далі вперед до кінця масиву виконуючи дії зазначені в пунктах 2-3.

Після досягнення кінця масиву масив буде відсортовано.

Загальна схема роботи алгоритму

$A[N]$  ;  $N = 5$

$i = 0$ ;  $j = i+1$ ;

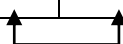
$A[j] < A[j-1]$ ;

0	1	2	3	4	$N-1=5$
3	5	1	8	7	4

$i = 1$ ;  $j = i+1$ ;

$A[j] < A[j-1]$ ;

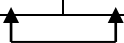
0	1	2	3	4	$N-1=5$
3	5	1	8	7	4



$i = 1$ ;  $j = j-1 = 1$ ;

$A[j] < A[j-1]$ ;

0	1	2	3	4	$N-1=5$
3	1	5	8	7	4



$i = 1; j = j - 1 = 0; j > 0$  – умова не виконується, повертаємось до  $i$ -го елемента і йдемо далі.

0	1	2	3	4	N-1 = 5
<b>1</b>	<b>3</b>	<b>5</b>	<b>8</b>	<b>7</b>	<b>4</b>

$i = 2; j = i + 1 = 3;$

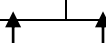
$A[j] < A[j-1];$

0	1	2	3	4	N-1 = 5
<b>1</b>	<b>3</b>	<b>5</b>	<b>8</b>	<b>7</b>	<b>4</b>

$i = 3; j = i + 1 = 4;$

$A[j] < A[j-1];$

0	1	2	3	4	N-1 = 5
<b>1</b>	<b>3</b>	<b>5</b>	<b>8</b>	<b>7</b>	<b>4</b>



$i = 3; j = i + 1 = 4;$

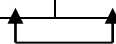
$A[j] < A[j-1];$  - надалі умови не виконується тому, йдемо далі ;

0	1	2	3	4	N-1 = 5
<b>1</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>8</b>	<b>4</b>

$i = 4; j = i + 1 = 5;$

$A[j] < A[j-1];$

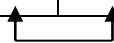
0	1	2	3	4	N-1 = 5
<b>1</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>8</b>	<b>4</b>



$i = 4; j = j - 1 = 4;$

$A[j] < A[j-1];$

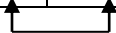
0	1	2	3	4	N-1 = 5
<b>1</b>	<b>3</b>	<b>5</b>	<b>7</b>	<b>4</b>	<b>8</b>



$i = 4; j = j - 1 = 3;$

$A[j] < A[j-1];$

0	1	2	3	4	$N-1 = 5$
1	3	5	4	7	8



$i = 4; j = j - 1 = 2;$

$A[j] < A[j-1];$  - надалі умова не виконується та ми дійшли до кінця масиву тому він вже є відсортований.

0	1	2	3	4	$N-1 = 5$
1	3	4	5	7	8

Алгоритм мовою C:

```
clock_t InsertExchange(int *A, int N)
{
    int j, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

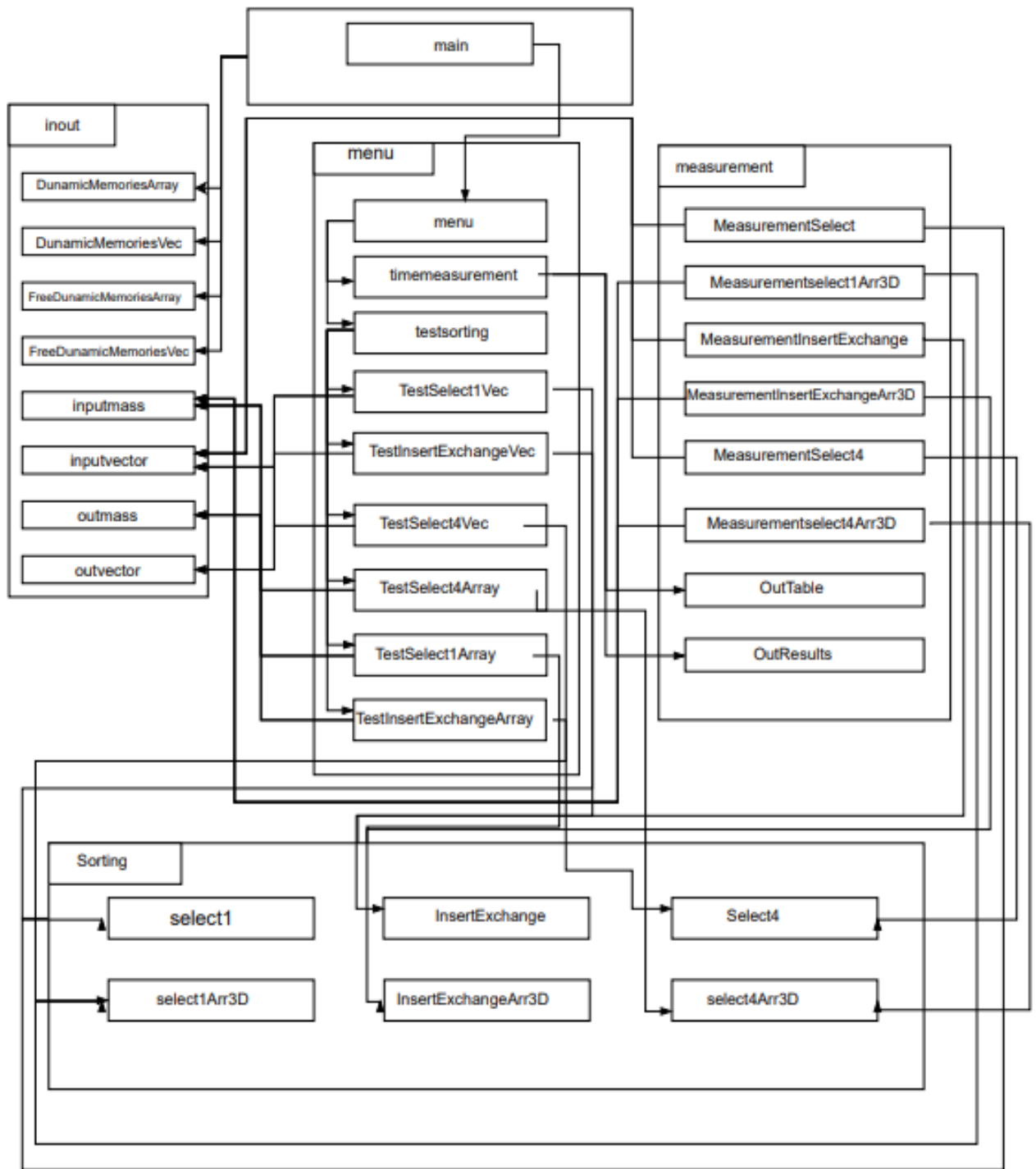
    for(int i=1; i<N; i++){
        j=i;
        while (j>0 && A[j]<A[j-1]) {
            tmp=A[j];
            A[j]=A[j-1];
            A[j-1]=tmp;
            j=j-1;
        }
    }

    time_stop = clock();

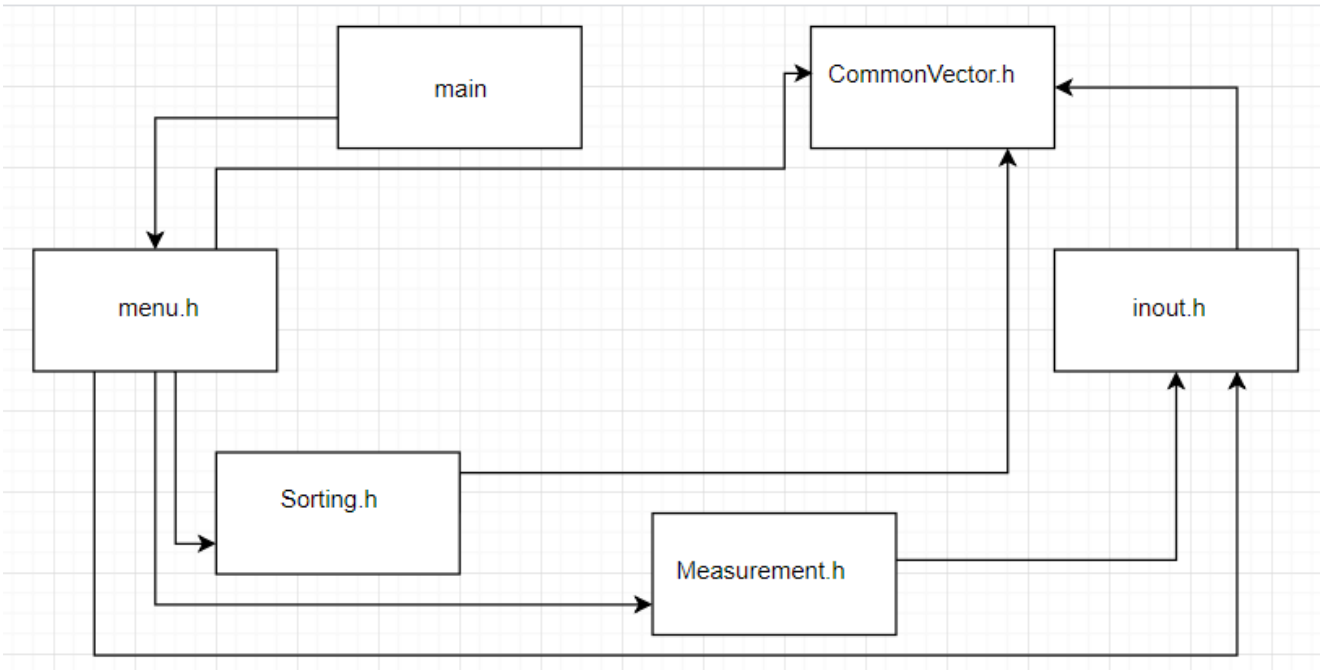
    return time_stop - time_start;
}
```

Рис. 17. Гібридний алгоритм "вставка – обмін".

# Схема взаємовикликів процедур та функцій



## Схема імпорту/експорту модулів



## Опис призначення процедур та функцій

Модуль **CommonVector**- відбувається оголошення тривірного масиву **Arr3D** та вектору **Vec**, оголошення змінних для запам'ятовування значень вимірів взаємності від способу заповнення масиву, а також підключення основних бібліотек.

Модуль **menu** :

- **void menu** - функція яка викликається з **main** що відповідає за діалог з користувачем , вибором режиму роботи програми.
- **void timemeasurement** – функція що відповідає за вивід та діалог з користувачем підпункту меню для виміру часу
- **void testsorting** – функція що відповідає за вивід та діалог з користувачем підпункту меню для тестування правильності роботи алгоритмів
- **void TestSelect1Vec** - відповідає за вивід результатів тестування алгоритму **Select1** для вектору
- **void TestInsertExchangeVec** - відповідає за вивід результатів тестування алгоритму **InsertExchange** для вектору

- ***void TestSelect4Vec*** - відповідає за вивід результатів тестування алгоритму Select4 для вектору.
- ***void TestSelect4Array*** - відповідає за вивід результатів тестування алгоритму Select4 для масиву.
- ***void TestSelect1Array*** - відповідає за вивід результатів тестування алгоритму Select1 для масиву
- ***void TestInsertExchangeArray***- відповідає за вивід результатів тестування алгоритму InsertExchange для масиву

#### Модуль *Sorting* :

- ***clock\_t select1***- вимірює час сортування алгоритму select1 для вектору ;
- ***clock\_t select1Arr3D*** – вимірює час сортування алгоритму select1 для масиву ;
- ***clock\_t InsertExchange*** – вимірює час сортування алгоритму InsertExchange для вектора;
- ***clock\_t InsertExchangeArr3D*** – вимірює час сортування алгоритму InsertExchange для масиву;
- ***clock\_t Select4*** – вимірює час сортування алгоритму Select4 для вектора;
- ***clock\_t select4Arr3D*** – вимірює час сортування алгоритму Select4 для масиву ;

#### Модуль *Measurement* :

- ***void MeasurementSelect*** – процедура для запуску 28 разів алгоритму select1 ( вектора) для подальшої обробки результату всіх вимірів часу;
- ***void Measurementsselect1Arr3D*** – процедура для запуску 28 разів алгоритму select1 (для масиву) для подальшої обробки результату всіх вимірів часу;
- ***void MeasurementInsertExchange*** – процедура для запуску 28 разів алгоритму InsertExchange(для вектору) для подальшої обробки результату всіх вимірів часу;
- ***void MeasurementInsertExchangeArr3D*** – процедура для запуску 28 разів алгоритму InsertExchange (для масиву)для подальшої обробки результату всіх вимірів часу;
- ***void MeasurementSelect4*** – процедура для запуску 28 разів алгоритму Select4 (для вектора) для подальшої обробки результату всіх вимірів часу;



- ***void Measurementsselect4Arr3D*** – процедура для запуску 28 разів алгоритму Select4 (для масиву) для подальшої обробки результату всіх вимірів часу;
- ***void OutTable***- процедура для виводу «шапки» таблиці на екран (відсортований , обернено відсортований і невідсортований собі заповнення):
- ***void OutResults*** – процедура що виводить середні значення часу робити алгоритмів в відповідні позиції для таблиці;
- ***float MeasurementProcessing*** – функція що обчислює і повертає середнє значення часу робити алгоритму;

Модуль ***inout*** :

- ***void inputmass*** - процедура для заповнення масиву відсортовано, обернено-відсортовано та невідсотовано;
- ***void inputvector*** – процедура для заповнення вектора відсортовано , обернено-відсортовано та невідсотовано;
- ***void outmass*** - процедура для виводу на екран масиву;
- ***void outvector*** – процедура для виводу на екран вектора;

## Текст головної програми та модулів

### **main.c**

```
#include <stdio.h> //бібліотека вводу виводу
#include <stdlib.h> // бібліотека виділення пам'яті
#include "menu.h"//підключення модуля меню

int main(){
    menu(); // вивклик меню для діалогу з коритсувачем та роботи програми
    return 0; }
```

### **CommonVector.h**

```
#ifndef COMMONVECTOR_H_INCLUDED
#define COMMONVECTOR_H_INCLUDED
    // розміри масиву
    #define P 3
    #define M 256
    #define N 256
    //оголошення вектора та масива
    int Arr3D[P][M][N];
    int vec[M*N];

    float ordered , random , backordered; // змінні для запам'ятовування результатів
    часу роботи алгоритму

#endif // COMMONVECTOR_H_INCLUDED
```

**inout.h**

```
#ifndef INOUT_H_INCLUDED
#define INOUT_H_INCLUDED

#include <stdio.h>

// прототипи функції заповнення масиву та виведення
void inputmass(int mode);
void inputvector(int mode);
void outmass();
void outvector();

#endif // INOUT_H_INCLUDED
```

**inout.c**

```
#include "inout.h" // підключення інтерфейсної частини
#include <stdio.h> // підключення бібліотеки вводу/виводу
#include "CommonVector.h" // підключення інтерфейсної частини для доступу до
глобальних змінних

int number; // змінна для заповнення масива
int i, j, k; // змінні для циклів
// функція ініціалізації масива залежно від заповнення
void inputmass(int mode){
    switch (mode){
        number = 0;
        case 1: // відсортований масив

            for (k = 0; k < P; k++ )
                for(i = 0; i < N; i++)
                    for (j = 0; j < M; j++)
                        Arr3D[k][j][i] = number++;

            break;
        // random масив
```

```

    case 2:{
for (k = 0; k < P; k++ )
    for(j = 0; j < M; j++)
        for (i = 0; i < N; i++)
            Arr3D[k][j][i] = rand() % (P*M*N);
break;
    }
    case 3:{ // обернено-відсортований
        number = P*N*M;
        for (k = 0; k < P; k++ )
            for(i = 0; i < N; i++)
                for (j = 0; j < M; j++)
                    Arr3D[k][j][i] = number--;
break;
    }
}

// функція ініціалізації вектору залежності від способу заповнення
void inputvector(mode){
    switch(mode){

        case 1:{ // відсортоване заповнення
            number = 0;
            for(i = 0; i < N*M; i++)
                vec[i] = number++;
            break;
        }
        case 2:{ // random заповнення
            for(i = 0; i < N*M; i++)
                vec[i] = rand() % (M*N);
            break;
        }
        case 3:{ // обернено-відсортоване заповнення
            number = N*M;

```

```

        for(i = 0; i < N*M; i++)
            vec[i] = number--;
        break;
    }
}
}

void outmass(){ // функція для виводу трьохвимірної масиви
    for (k = 0; k < P; k++){
        printf("section %d :\n",k);
        for(i = 0; i < M; i++){
            for (j = 0; j < N; j++){
                printf("%-d\t",Arr3D[k][i][j]);
            }
            printf("\n");
        }
        printf("\n\n");
    }
}

void outvector(){ // функція для виведення вектору
    for (i = 0; i < N*M; i++)
        printf("%d ",vec[i]);
}

```

## **sorting.h**

```
#ifndef SORTING_H_INCLUDED
#define SORTING_H_INCLUDED

#include <time.h> //бібліотека для роботи з часом

// прототипи функцій для виміру часу і сортування масиву
clock_t select1();
clock_t select1Arr3D();
clock_t InsertExchange();
clock_t InsertExchangeArr3D();
clock_t Select4();
clock_t select4Arr3D();
#endif // SORTING_H_INCLUDED
```

## **sorting.c**

```
#include <stdio.h>
#include <stdlib.h>
#include "Sorting.h" // підключення інтерфейсної частини
#include <time.h> // підключення бібліотеки для виміру часу
#include "CommonVector.h" // підключення модуля глобальних змінних

clock_t select1(){ //Сортування вектора методом вибору.
    int Min, imin;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int S = 0; S < N*M-1; S++){
        Min = vec[S]; imin = S;
        for(int i = S + 1; i < K; i++ )
            if(vec[i]<Min){
                Min = vec[i];
                imin = i;
            }
    }
    return time_stop - time_start;
}
```

```

    }
    vec[imin] = vec[S];
    vec [S] = Min;
}
time_stop = clock();
return P*(time_stop - time_start);
}
clock_t select1Arr3D() { //Сортування 3-вимірної масиви методом вставки
наскірзно по стовпчиках.
    clock_t time_start, time_stop;
    time_start = clock();
    int Min; // Змінна яка слугує для пошуку мінімального числа.
    int CurrentJmin, CurrentImin; // змінні для запам'ятовування індексів
мінімального елемента для обміну.

    for( int k = 0; k < P; k++){ //
        for( int j = 0; j < N; j++){
            for( int i = 0; i < M; i++){
                Min = Arr3D[k][i][j]; // запам'ятовування першого елемента не
відсортованої частини, та пошук меншого серед інших
                CurrentImin = i; // запам'ятовування координат Min.
                CurrentJmin = j;

                for (int z = i + 1 ; z < M; z++){ // цикл , що  шукає мінімальний елемент
по стовчику де знаходить тимчасовий мінімальний елемент
                    if(Arr3D[k][z][j] < Min ) { //пошук в стовпчику меншого числа за Min.
                        Min = Arr3D[k][z][j]; // перезапис числа Min знайденого меншог
числа , якщо таке існує.
                        CurrentJmin = j; //запам'ятовування координат нового мінімального
числа.
                        CurrentImin = z;
                    }
                }
                // пошук у не відсортованій частині матриці нового ще мінімальнішого
числа ніж було знайдено попередньо.
                for(int p = j+1; p < N; p++){

```

```

        for( int d = 0; d < M; d++){
            if(Arr3D[k][d][p] < Min ){ // перезапис нового знайденого
                мінімального числа для яопдальшого обміну з останнім елементом відсортованої
                частини.
                Min = Arr3D[k][d][p];
                CurrentJmin = p;
                CurrentImin = d;
            }
        }
    }
    // обмін знайденого наменшого мінімального числаб запис його в останнє
    відсортовану комірку масиву.
    Arr3D[k][CurrentImin][CurrentJmin] = Arr3D[k][i][j];
    Arr3D[k][i][j] = Min;
}
}

```

```

}

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

```

```

clock_t InsertExchange(){

    int j, tmp;

    clock_t time_start, time_stop;
    time_start = clock();

    for(int i = 1; i < N*M ; i++){
        j = i ;
        while ( j > 0 && vec[j] < vec[j-1]){
            tmp = vec[j] ;
            vec[j] = vec [j-1];

```



```

        vec[j-1] = tmp;
        j = j - 1;
    }
}

time_stop = clock();
return P*(time_stop - time_start);

}

clock_t InsertExchangeArr3D(){

    int p, g, tmp, Count, start; // додаткові змінні для обміну запам'ятовування
    елементів

    clock_t time_start, time_stop; // змінні для виміру часу роботи алгоритму
    time_start = clock(); // починає відлік початку роботи алгоритму

    for(int k = 0; k < P ;k++){ // перехід по перерізах
        for( int j = 0; j < N; j++){ //перехід по стовпчиках
            int i = 0;
            if(j == 0){ // умова для початку пошуку не враховуючі елемент матриці
Arr3D[P,0,0]
                i = 1;
                while (i < M){ // перехід по рядках
                    // пошук відбувається так що ми розділяємо матрицю на дві
частини, 1 - вектор по g стовпчику , 2- матриця з починаючи від g-1 стовпчика і
до 0 стовпчика.
                    // 1- частина
                    for(g = j, p = i; p >= 0; p--){ // пошук мінімального елемента в j
стовпчику

                        if((p==0) && (g!=0) && (Arr3D[k][p][g] < Arr3D[k][M-1][g-1] )){ //
умова що перевіряє чи обраний елемент стоїть на 0-рядку g -го стовпчика, для
порівняння його з M-1 елементом g-1 стовпчика якщо він менший міняємо їх
місцями.

```

```

    tmp = Arr3D[k][p][g];
    Arr3D[k][p][g] = Arr3D[k][M-1][g-1];
    Arr3D[k][M-1][g-1] = tmp;
} else {
    if((p!=0) && (Arr3D[k][p][g] < Arr3D[k][p-1][g])){ // умова яка
перевіряє чи елемент p менший ніж p-1 у стовпчику g якщо так, міняємо.
        tmp = Arr3D[k][p][g];
        Arr3D[k][p][g] = Arr3D[k][p-1][g];
        Arr3D[k][p-1][g] = tmp;
    }
}
}

// 2- частина

for(g = j-1; g >= 0; g--){ // цикл який робить пошук мінімального
елемента порівнюючи його з обраним у уже відсортованій частині масиву,
починаючи з g-1 стовпчика
    for(p = M-1; p >= 0 ; p--){

        if((p==0) && (g!=0) && (Arr3D[k][p][g] < Arr3D[k][M-1][g-1] )){ //
умова що перевіряє чи обраний елемент стоїть на 0-рядку g -го стовпчика, для
порівняння його з M-1 елементом g-1 стовпчика якщо він менший міняємо їх
місцями.
            tmp = Arr3D[k][p][g];
            Arr3D[k][p][g] = Arr3D[k][M-1][g-1];
            Arr3D[k][M-1][g-1] = tmp;
        } else {
            if((p!=0) && (Arr3D[k][p][g] < Arr3D[k][p-1][g])){ // умова яка
перевіряє чи елемент p менший ніж p-1 у стовпчику g якщо так, міняємо.
                tmp = Arr3D[k][p][g];
                Arr3D[k][p][g] = Arr3D[k][p-1][g];
                Arr3D[k][p-1][g] = tmp;
            }
        }
    }
}

```

```

        }
        i++;
    }
}
}

time_stop = clock(); // час завершення роботи алгоритму
return time_stop - time_start; // повернення результату після закінчення
роботи функції
}

```

```

clock_t Select4(){
    int L, R, imin, imax, tmp;
    L = 0 ;
    R = N*M - 1;
    clock_t time_start, time_stop;
    time_start = clock();

    while( L < R ){
        imin = L;
        imax = L;
        for(int i = L + 1; i < R + 1; i++){
            if( vec[i] < vec[imin] ) imin = i;
            else
                if(vec[i] > vec[imax]) imax = i;
        }
        tmp = vec[imin];
        vec[imin] = vec[L];
        vec[L] = tmp;
        if( imax == L){
            tmp = vec[imin];
            vec[imin] = vec[R];
            vec[R] = tmp;
        }
    }
}

```

```

    }else{
        tmp = vec[imax];
        vec[imax] = vec[R];
        vec[R] = tmp;
    }
    L = L + 1;
    R = R - 1;
}
time_stop = clock();
return P*(time_stop - time_start);
}

```

```

clock_t select4Arr3D(){
    int imin, imax, jmin, jmax, tmp;
    imin = 0; imax = 0; jmax = 0; jmin = 0;
    int i, j;
    clock_t time_start, time_stop;
    time_start = clock();

    for (int k = 0; k < P; k++){
        for (j = 0; j <= N/2; j++){
            for (i = 0; i < M; i++){
                // під час пошуку ми бере за мінімальний елемент перший , шукаємо
                // серед наступних менший ніж є якщо умова виконується запам'ятовуємо його
                // місце знаходження

                //якщо не виконується ми перевіряємо чи він максимальний якщо так
                //ти запам'ятовуємо його позицію

                //при переході на нову ітерацію циклу повторюємо це знову і знову,
                //допоки не буде відсортований весь масив і всі не пройдуть ітерації всіх циклівю

                imin = i;
                jmin = j;
                imax = i;
                jmax = j;

                // ми розділяємо наш масив на три частини 1- вектор по j стовпчику,
                // 2- матриця від( p = j+1) p стовпчика до M-p-1, та 3 -вектор по M - j -1;
            }
        }
    }
    time_stop = clock();
    return time_stop - time_start;
}

```

```

        for(int g = i; (g < M) && (j != N-j-1) ; g++){ // порівнюємо елементи і
запам'ятовуємо мінімальне і максимальне місце знаходження у стопчику j;
            if( Arr3D[k][g][j] < Arr3D[k][imin][jmin] ){
                imin = g;
                jmin = j;
            }else
                if(Arr3D[k][g][j] > Arr3D[k][imax][jmax]){
                    imax = g;
                    jmax = j;
                }
        }
        for(int p = j+1; j < N - p - 1; p++){ //порівнюємо елементи і
запам'ятовуємо мінімальне і максимальне місце знаходження в данній матриці
            for(int g = 0; g < M; g++){
                if( Arr3D[k][g][p] < Arr3D[k][imin][jmin] ){
                    imin = g;
                    jmin = p;
                }else
                    if(Arr3D[k][g][p] > Arr3D[k][imax][jmax]){
                        imax = g;
                        jmax = p;
                    }
            }
        }
        for ( int g=0; (g < M-i) && (j != N-j-1); g++ ){ // порівнюємо елементи і
запам'ятовуємо мінімальне і максимальне місце знаходження у стопчику g;
            if( Arr3D[k][g][N-j-1] < Arr3D[k][imin][jmin] ){
                imin = g;
                jmin = N-j-1;
            }else
                if(Arr3D[k][g][N-j-1] > Arr3D[k][imax][jmax]){
                    imax = g;
                    jmax = N-j-1;
                }
        }

```

```

        for(int g = i; (g < M-i) && (j == N-j-1 ); g++){ // данний цикл буде
        працювати якщо задано масив з непарною кількістю стовпців

                                // порівнюємо елементи і запам'ятовуємо
        мінімальне і максимальне місце знаходження у стопчику j;

        if( Arr3D[k][g][j] > Arr3D[k][imin][jmin] ){

            imin = g;

            jmin = j;

        }else

            if(Arr3D[k][g][j] < Arr3D[k][imax][jmax]){

                imax = g;

                jmax = j;

            }

        }

// обмін мінімального і максимального елемента з першим і останнім відповідно

        tmp = Arr3D[k][imin][jmin];
        Arr3D[k][imin][jmin] = Arr3D[k][i][j];
        Arr3D[k][i][j] = tmp;
        if((imax == i)&&(jmax == j)){

            tmp = Arr3D[k][imin][jmin];
            Arr3D[k][imin][jmin] = Arr3D[k][M-1-i][N-1-j];
            Arr3D[k][M-1-i][N-1-j] = tmp;

        }else{

            tmp = Arr3D[k][imax][jmax];
            Arr3D[k][imax][jmax] = Arr3D[k][M-1-i][N-1-j];
            Arr3D[k][M-1-i][N-1-j] = tmp;

        }

    }

}

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

```

## **menu.h**

```
#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

// прототипи функції для тестування/дослідження для діалогу з користувачем
void menu();

void timemeasurement();
void testsorting();

void TestSelect1Vec();
void TestInsertExchangeVec();
void TestSelect4Vec();

void TestSelect4Array();
void TestSelect1Array();
void TestInsertExchangeArray();

#endif // MENU_H_INCLUDED
```

## **menu.h**

```
#include <stdio.h> // підключення бібліотеки вводу / виводу
#include "CommonVector.h" // підклбчення модуля з глоабльними зміннимита
вказівниками на масив
#include "menu.h"// підключення інтерфейсної частини
#include "Sorting.h" // підключення модуля сортування
#include "Measurement.h" // підключення модуля усереднення часу
#include "inout.h"
// функція для тестування вектора за методом прямої вставки
void TestSelect1Vec(){
```

```

    printf("Sizes of array: P=%d M=%d N=%d\n",P,M,N); // виведення на екран
розмірів масиву
    printf("test select1 (ordered, vector)\n"); //вивід на екран спосіб заповнення
масиву
    inputvector(1);
    outvector();
    select1();
    printf("\nsorting:\n"); // розмежування між відсотованим масивом і не
відсотованим
    outvector();
    printf("\ntest select1 (random, vector)\n"); // вивід на екран спосіб заповнення
масиву
    inputvector(2);
    outvector();
    select1();
    printf("\nsorting:\n"); // розмежування між відсотованим і не відсотованим
масивом
    outvector();
    printf("\ntest select1 (backordered, vector)\n"); // вивід на екра спосіб заповнення
масиву
    inputvector(3);
    outvector();
    select1();
    printf("\nsorting:\n"); // розмежування між відсотованою і не відсотованою
частиною масиву
    outvector();

}

```

```

void TestInsertExchangeVec(){

```

```

    printf("Sizes of array: P=%d M=%d N=%d\n",P,M,N); // виведення на екран
розмірів масиву
    printf("test InsertExchange (ordered, vector)\n"); // вивід на екран спосіб
заповнення масиву
    inputvector(1);

```



```

    outvector();
    InsertExchange();
    printf("\nsorting:\n");// розмежування між відсотованим масивом і не
відсотованим
    outvector();
    printf("\ntest InsertExchange (random, vector)\n");// вивід на екран спосіб
заповнення масиву
    inputvector(2);
    outvector();
    InsertExchange();
    printf("\nsorting:\n");// розмежування між відсотованим масивом і не
відсотованим
    outvector();
    printf("\ntest InsertExchange (backordered, vector)\n");// вивід на екран спосіб
заповнення масиву
    inputvector(3);
    outvector();
    InsertExchange();
    printf("\nsorting:\n");// розмежування між відсотованим масивом і не
відсотованим
    outvector();

}

void TestSelect4Vec(){

    printf("Sizes of array: P=%d M=%d N=%d\n",P,M,N);// виведення на екран
розмірів масиву
    printf("test Select4 (ordered, vector)\n");// вивід на екран спосіб заповнення
масиву
    inputvector(1);
    outvector();
    Select4();
    printf("\nsorting:\n");// розмежування між відсотованим масивом і не
відсотованим
    outvector();
    printf("\nSelect4 (random, vector)\n");// вивід на екран спосіб заповнення масиву

```

```

    inputvector(2);
    outvector();
    Select4();
    printf("\nsorting:\n");// розмежування між відсотованим масивом і не
відсотованим
    outvector();
    printf("\nSelect4 (backordered, vector)\n");// вивід на екран спосіб заповнення
масиву
    inputvector(3);
    outvector();
    Select4();
    printf("\nsorting:\n");// розмежування між відсотованим масивом і не
відсотованим
    outvector();

}

```

```

void TestSelect1Array(){

```

```

    printf("Sizes of array: P=%d M=%d N=%d\n",P,M,N);// виведення на екран
розмірів масиву
    printf("test Select1 (ordered, array)\n");// вивід на екран спосіб заповнення
масиву
    inputmass(1);
    outmass();
    select1Arr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
відсотованим
    outmass();
    printf("test Select1 (random, array)\n");// вивід на екран спосіб заповнення
масиву
    inputmass(2);
    outmass();
    select1Arr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
відсотованим

```

```

    outmass();
    printf("test Select1 (backordered, array)\n");// вивід на екран спосіб заповнення
масиву
    inputmass(3);
    outmass();
    select1Arr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
відсотованим
    outmass();

}

void TestSelect4Array(){

    printf("Sizes of array: P=%d M=%d N=%d\n",P,M,N);// виведення на екран
розмірів масиву
    printf("test Select4 (ordered, array)\n");// вивід на екран спосіб заповнення
масиву
    inputmass(1);
    outmass();
    select4Arr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
відсотованим
    outmass();
    printf("test Select4 (random, array)\n");// вивід на екран спосіб заповнення
масиву
    inputmass(2);
    outmass();
    select4Arr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
відсотованим
    outmass();
    printf("test Select4 (backordered, array)\n");// вивід на екран спосіб заповнення
масиву
    inputmass(3);
    outmass();
    select4Arr3D();

```

```

    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
    відсотованим
    outmass();
}

```

```

void TestInsertExchangeArray(){
    printf("Sizes of array: P=%d M=%d N=%d\n",P,M,N);// виведення на екран
    розмірів масиву
    printf("test InsertExchange (ordered, array)\n");// вивід на екран спосіб
    заповнення масиву
    inputmass(1);
    outmass();
    InsertExchangeArr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
    відсотованим
    outmass();
    printf("test InsertExchange (random, array)\n");// вивід на екран спосіб
    заповнення масиву
    inputmass(2);
    outmass();
    InsertExchangeArr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
    відсотованим
    outmass();
    printf("test InsertExchange (backordered, array)\n");// вивід на екран спосіб
    заповнення масиву
    inputmass(3);
    outmass();
    InsertExchangeArr3D();
    printf("\nAfter sorting :\n");// розмежування між відсотованим масивом і не
    відсотованим
    outmass();
}

```

```

void timemeasurement() { //пункт виміру часу в меню

```

```

    do {

```

```

system("cls");
int mode;
printf("Time measurement\nSizes of array: P=%d M=%d N=%d\n",P,M,N);//
виведення на екран розмірів масиву
printf("\nChoose sorting:\n");
printf("1.Select1(vector)\n2.InsertExchange(vector)\n");
printf("3. Select4 (vector)\n4. select1(array)\n");
printf("5.InsertExchange(array)\n6.Select4(array)\n");
printf("7.Pack mode(vector)\n8.Pack mode(array)\n");
printf("\nPlease input the number of menu(0-back):");
scanf("%d", &mode);
switch (mode)//вибір пункту меню
{

    case 0: return; break;
    case 1:{
        system("cls");
printf("Select1(vector)\n"); // вивід на екран спосіб сортування
        OutTable();
        MeasurementSelect(1);
        ordered = MeasurementProcessing();
        MeasurementSelect(2);
        random = MeasurementProcessing();
        MeasurementSelect(3);
        backordered = MeasurementProcessing();
        OutResults(ordered,random,backordered);
        printf("\nPress Enter for back");
        _getch();

        break;}
    case 2:{
        system("cls");
printf("InsertExchange(vector)\n"); // вивід на екран спосіб
сортування

```

```

        OutTable();
        MeasurementInsertExchange(1);
        ordered = MeasurementProcessing();
        MeasurementInsertExchange(2);
random = MeasurementProcessing();
MeasurementInsertExchange(3);
backordered = MeasurementProcessing();
        OutResults(ordered,random,backordered);
        printf("\nPress Enter for back");
        _getch();
        break;}
case 3:{
        system("cls");
        printf("Select4(vector)\n");// вивід на екран спосіб сотування
        OutTable();
        MeasurementSelect4(1);
        ordered = MeasurementProcessing();
        MeasurementSelect4(2);
random = MeasurementProcessing();
MeasurementSelect4(3);
backordered = MeasurementProcessing();
        OutResults(ordered,random,backordered);
        printf("\nPress Enter for back");
        _getch();
        break;}
case 4:{
        system("cls");
        printf("Select11(array)\n");// вивід на екран спосіб сотування
        OutTable();
        Measurementsselect1Arr3D(1);
        ordered = MeasurementProcessing();
        Measurementsselect1Arr3D(2);
random = MeasurementProcessing();
Measurementsselect1Arr3D(3);

```

```

backordered = MeasurementProcessing();
OutResults(ordered,random,backordered);
    printf("\nPress Enter for back");
    _getch();
    break;}

case 5:{
    system("cls");
    printf("InsertExchange(array)\n");// вивід на екран спосіб
сотування
    OutTable();
    MeasurementInsertExchangeArr3D(1);
    ordered = MeasurementProcessing();
    MeasurementInsertExchangeArr3D(2);
    random = MeasurementProcessing();
    MeasurementInsertExchangeArr3D(3);
    backordered = MeasurementProcessing();
    OutResults(ordered,random,backordered);
    printf("\nPress Enter for back");
    _getch();
    break;}

case 6:{
    system("cls");
    printf("Select4(array)\n");// вивід на екран спосіб сотування
    OutTable();
    Measurementsselect4Arr3D(1);
    ordered = MeasurementProcessing();
    Measurementsselect4Arr3D(2);
    random = MeasurementProcessing();
    Measurementsselect4Arr3D(3);
    backordered = MeasurementProcessing();
    OutResults(ordered,random,backordered);
    printf("\nPress Enter for back");
    _getch();

```

```

        break;
    }
    case 7 :{

        printf("\t\t");
        OutTable();
        printf("Select1(vector)");// вивід на екран спосіб сотування
        MeasurementSelect(1);
            ordered = MeasurementProcessing();
            MeasurementSelect(2);
        random = MeasurementProcessing();
        MeasurementSelect(3);
        backordered = MeasurementProcessing();
        printf("\t");
        OutResults(ordered,random,backordered);
        printf("InsertExchange(vector)");// вивід на екран спосіб сотування
        MeasurementInsertExchange(1);
            ordered = MeasurementProcessing();
            MeasurementInsertExchange(2);
        random = MeasurementProcessing();
        MeasurementInsertExchange(3);
        backordered = MeasurementProcessing();
        OutResults(ordered,random,backordered);
        printf("Select4(vector)");// вивід на екран спосіб сотування
        MeasurementSelect4(1);
            ordered = MeasurementProcessing();
            MeasurementSelect4(2);
        random = MeasurementProcessing();
        MeasurementSelect4(3);
        backordered = MeasurementProcessing();
        printf("\t");
        OutResults(ordered,random,backordered);
        _getch();
    }
}

```



```

    break;
}
case 8:{
    srand(time(NULL));
    printf("\t\t");
    OutTable();
    Measurementsselect1Arr3D(1);
        ordered = MeasurementProcessing();
        Measurementsselect1Arr3D(2);
    random = MeasurementProcessing();
    Measurementsselect1Arr3D(3);
    backordered = MeasurementProcessing();
    printf("Select1(array)");// вивід на екран спосіб сотування
    printf("\t");
    OutResults(ordered,random,backordered);
    MeasurementInsertExchangeArr3D(1);
    ordered = MeasurementProcessing();
    MeasurementInsertExchangeArr3D(2);
    random = MeasurementProcessing();
    MeasurementInsertExchangeArr3D(3);
    backordered = MeasurementProcessing();
    printf("InsertExchange(array)");// вивід на екран спосіб сотування
    OutResults(ordered,random,backordered);

    Measurementsselect4Arr3D(1);
    ordered = MeasurementProcessing();
    Measurementsselect4Arr3D(2);
    random = MeasurementProcessing();
    Measurementsselect4Arr3D(3);
    backordered = MeasurementProcessing();
    printf("Select4(array)");// вивід на екран спосіб сотування
    printf("\t");
    OutResults(ordered,random,backordered);

```

```

        _getch();_getch();
        break;
    }

    default:{
        printf("Please input CORRECT number (Enter)");
        _getch();
        break;}
    }
} while (1);
}

```

void testsorting() {//пункт меню для тестування сортування і виводу його на екран

```

do {
    system("cls");
    int mode;

    printf("Test sortings\nSizes of array: P=%d M=%d N=%d\n",P,M,N);
    printf("\nChoose sorting:\n");
    printf("1.Select1(vector)\n2.InsertExchange(vector)\n");
    printf("3. Select4 (vector)\n4. select1(array)\n");
    printf("5.InsertExchange(array)\n6.Select4(array)\n");
    printf("\nPlease input the number of menu(0-back):");
    scanf("%d", &mode);
    switch (mode)//вибір виду сортування
    {
        case 0: return; break;
        case 1:
            TestSelect1Vec();
            _getch();
            break;
        case 2:
            TestInsertExchangeVec();
            _getch();
            break;
    }
}

```

```

        case 3:
            TestSelect4Vec();
            _getch();
            break;
        case 4:
            TestSelect1Array();
            _getch();
            break;
        case 5:
            TestInsertExchangeArray();
            _getch();
            break;
        case 6:
            TestSelect4Array();
            _getch();
            break;

        default:
            printf("Please input CORRECT number (Enter)");
            _getch();
            break;
    }
} while (1);
}

void menu(){//меню з вибором тестування часу і тестування сортувань
    do {
        system("cls");
        printf("Course work #146\n\n");
        printf("Choose mode:\n1- time measurement\n");
        printf("2-testing sorting\n3-exit:\n");
        int vote;
        scanf("%d", &vote);
        switch (vote){//вибір одного з двох пунктів
            {

```

```

        case 1: timemeasurement(); break;
        case 2: testsorting(); break;
        case 3: return;
        default:
            printf("Please input CORRECT number (Press Enter)");
            _getch();
            break;
    }
} while (1);
}

```

### **measurement.h**

```

#ifndef MEASUREMENT_H_INCLUDED
#define MEASUREMENT_H_INCLUDED
#include "sorting.h"
#include "inout.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define rejected_number 2// кількість вимірів які ми не враховуємо
#define min_max_number 3 //кількість максимальних і мінімальних значень часу
які ми теж не враховуємо

#define measurements_number 28//кількість вимірів

extern clock_t Res[measurements_number]; //вектор для запам'ятовування часу всіх
вимірів

//прототипи функції сортування для виміру часу щоб надалі знайти середнє
значення часу
void MeasurementSelect(int inout);
void Measurementsselect1Arr3D(int inout);
void MeasurementInsertExchange(int inout);

```

```

void MeasurementInsertExchangeArr3D(int inout);
void MeasurementSelect4(int inout);
void Measurementsselect4Arr3D(int inout);
// прототипи функцій виводу результатів на екран у вигляді таблиці
void OutTable();
void OutResults(float ordered, float random, float backordered);
// прототип функції для виміру середнього значення часу
float MeasurementProcessing();
float AverageValue;//змінна для запам'ятовування середнього значення часу
роботи алгоритму

#endif // MEASUREMENT_H_INCLUDED

```

#### **measurement.h**

```

#include "measurement.h" // підключення інтерфейсної частини
#include "inout.h" // підключення інтерфейсної частини для доступу до
заповнення масиву

#define measurements_number 28 // кількість запусків обробки алгоритмів для
знаходження середнього значення

#define rejected_number 2 // кількість вимірів які ми не враховуємо при
обчислення середнього значення часу

#define min_max_number 3 // кількість вимірів які ми не враховуємо при
обчисленні середнього значення часу

int inout; // змінна для визначення запуску конкретного алгоритма з конкретним
заповненням

clock_t Res[measurements_number]; // масив для збереження часу кожного з
вимірів
// функції визначення часу роботи кожного алгоритма за різними випадками
відсортованості

void MeasurementSelect(int inout){
    for ( int i = 0; i < measurements_number; i++){

```

```

switch (inout) {
case 1 :{
inputvector(1); // заповнення впорядковано
Res[i] = select1(); // запис результату до масиву
break;}
case 2:{
inputvector(2); //заповнення неупорядковано
Res[i] = select1(); // запис результату до масиву значень
break; }
case 3:{
inputvector(3); //заповнення обернено-відсортовано
Res[i] = select1(); //запис результату до масиву значень
break;

}
}
}

void Measurementsselect1Arr3D(int inout){
for ( int i = 0; i < measurements_number; i++){
switch (inout) {
case 1 :{
inputmass(1); // заповнення впорядковано
Res[i] = select1Arr3D(); // запис результатів до масиву значень
break;}
case 2:{
inputmass(2); // заповнення неупорядкованого масиву

Res[i] = select1Arr3D(); // запис результатів до масиву значень
break; }
case 3:{
inputmass(3); // заповнення обернено-відсортовано
Res[i] = select1Arr3D(); // запис результатів до масиву значень

```

```

        break;

    }

}

}

}

void MeasurementInsertExchange(int inout){
    for ( int i = 0; i < measurements_number; i++){
        switch (inout) {
            case 1 :{
                inputvector(1); // заповнення відсортоване
                Res[i] = InsertExchange();
                break;}
            case 2:{
                inputvector(2); // заповнення невідсортоване
                Res[i] = InsertExchange();
                break; }
            case 3:{
                inputvector(3); // заповнення обернено-відсортовано
                Res[i] = InsertExchange();
                break;
            }
        }
    }
}

```

```

void MeasurementInsertExchangeArr3D(int inout){
    for ( int i = 0; i < measurements_number; i++){
        switch (inout) {
            case 1 :{
                inputmass(1); // заповнення відсотоване
                Res[i] = InsertExchangeArr3D();
                break;}
            case 2:{

```

```

    inputmass(2); // заповнення невідсортоване
    Res[i] = InsertExchangeArr3D();
    break; }
    case 3:{
    inputmass(3); // заповнення обернено-відсортоване
    Res[i] = InsertExchangeArr3D();
    break;

    }
    }
}
}

```

```

void MeasurementSelect4(int inout){
    for ( int i = 0; i < measurements_number; i++){
        switch (inout) {
            case 1 :{
                inputvector(1); // заповнення відсортоване
                Res[i] = Select4();
                break;}
            case 2:{
                inputvector(2); // заповнення невідсортоване
                Res[i] = Select4();
                break; }
            case 3:{
                inputvector(3); // заповнення обернено- відсортоване
                Res[i] = Select4();
                break;
                }
            }
        }
    }
}

```

```

void Measurementsselect4Arr3D(int inout){
    for ( int i = 0; i < measurements_number; i++){

```



```

switch (inout) {
case 1 :{
inputmass(1); // заповнення відсортоване
Res[i] = select4Arr3D();
break;}
case 2:{
inputmass(2); // заповнення невідсортоване
Res[i] = select4Arr3D();
break; }
case 3:{
inputmass(3); // заповнення обернено-відсортоване
Res[i] = select4Arr3D();
break;
    }
}
}
}

```

// функція для визначення середнього значення часу робити кожного алгоритма

```

float MeasurementProcessing(){
    long int Sum;
    AverageValue = 0;
    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;
}

```

```

    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));

```

```

/* !!!!! Пустий printf потрібен для правильного повернення результату */
printf("");
/* !!!!! */

printf("",AverageValue); // для коректного виводу результатів по іншому
                          виводить “мусор”

return AverageValue;
}

```

```

void OutTable()
{
    // Усереднений результат вимірів буде виведено на екран у портібну позицію
    printf("\t Ordered \t Random \t BackOrdered \n");}

```

```
void OutResults(float ord, float rand, float backord){//процедура виводу результатів  
на екран  
    printf("\t %7.2f \t %7.2f \t %7.2f \n",ord , rand ,backord);  
    printf("\n\n");  
}
```

## Тести програми

```
Sizes of array: P=3 M=4 N=5  
test Select1 (ordered, array)
```

```
section 0 :  
0      4      8      12     16  
1      5      9      13     17  
2      6     10      14     18  
3      7     11      15     19
```

```
section 1 :  
20     24     28     32     36  
21     25     29     33     37  
22     26     30     34     38  
23     27     31     35     39
```

```
section 2 :  
40     44     48     52     56  
41     45     49     53     57  
42     46     50     54     58  
43     47     51     55     59
```

After sorting :

```
section 0 :  
0      4      8      12     16  
1      5      9      13     17  
2      6     10      14     18  
3      7     11      15     19
```

```
section 1 :  
20     24     28     32     36  
21     25     29     33     37  
22     26     30     34     38  
23     27     31     35     39
```

```
section 2 :  
40     44     48     52     56  
41     45     49     53     57  
42     46     50     54     58  
43     47     51     55     59
```

```
test Select1 (random, array)  
section 0 :  
16     35     50     42     48  
46     40     2      4      28  
26     45     50     9      10  
50     6      21     33     8
```

```
section 1 :  
9      23     24     34     36  
20     46     56     11     28  
24     39     26     43     17  
58     38     42     9      41
```

```
section 2 :  
33     35     19     18     24  
30     57     26     53     6  
41     5      44     52     30  
9      17     33     57     12
```

After sorting :

```
section 0 :  
2      9      26     40     48  
4      10     28     42     50  
6      16     33     45     50  
8      21     35     46     50
```

```
section 1 :  
9      20     26     38     43  
9      23     28     39     46  
11     24     34     41     56  
17     24     36     42     58
```

```
section 2 :  
5      17     26     33     52  
6      18     30     35     53  
9      19     30     41     57  
12     24     33     44     57
```

```
test Select1 (backordered, array)
```

```
section 0 :
```

60	56	52	48	44
59	55	51	47	43
58	54	50	46	42
57	53	49	45	41

```
section 1 :
```

40	36	32	28	24
39	35	31	27	23
38	34	30	26	22
37	33	29	25	21

```
section 2 :
```

20	16	12	8	4
19	15	11	7	3
18	14	10	6	2
17	13	9	5	1

```
After sorting :
```

```
section 0 :
```

41	45	49	53	57
42	46	50	54	58
43	47	51	55	59
44	48	52	56	60

```
section 1 :
```

21	25	29	33	37
22	26	30	34	38
23	27	31	35	39
24	28	32	36	40

```
section 2 :
```

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

```

test InsertExchange (ordered, array)
section 0 :
0      4      8      12     16
1      5      9      13     17
2      6     10     14     18
3      7     11     15     19

section 1 :
20     24     28     32     36
21     25     29     33     37
22     26     30     34     38
23     27     31     35     39

section 2 :
40     44     48     52     56
41     45     49     53     57
42     46     50     54     58
43     47     51     55     59

After sorting :
section 0 :
0      4      8      12     16
1      5      9      13     17
2      6     10     14     18
3      7     11     15     19

section 1 :
20     24     28     32     36
21     25     29     33     37
22     26     30     34     38
23     27     31     35     39

section 2 :
40     44     48     52     56
41     45     49     53     57
42     46     50     54     58
43     47     51     55     59

```

```

test InsertExchange (random, array)
section 0 :
16     35     50     42     48
46     40     2      4      28
26     45     50     9      10
50     6      21     33     8

section 1 :
9      23     24     34     36
20     46     56     11     28
24     39     26     43     17
58     38     42     9      41

section 2 :
33     35     19     18     24
30     57     26     53     6
41     5      44     52     30
9      17     33     57     12

After sorting :
section 0 :
2      9      26     40     48
4      10     28     42     50
6      16     33     45     50
8      21     35     46     50

section 1 :
9      20     26     38     43
9      23     28     39     46
11     24     34     41     56
17     24     36     42     58

section 2 :
5      17     26     33     52
6      18     30     35     53
9      19     30     41     57
12     24     33     44     57

```

```

test InsertExchange (backordered, array)
section 0 :
60      56      52      48      44
59      55      51      47      43
58      54      50      46      42
57      53      49      45      41

section 1 :
40      36      32      28      24
39      35      31      27      23
38      34      30      26      22
37      33      29      25      21

section 2 :
20      16      12      8       4
19      15      11      7       3
18      14      10      6       2
17      13      9       5       1

After sorting :
section 0 :
41      45      49      53      57
42      46      50      54      58
43      47      51      55      59
44      48      52      56      60

section 1 :
21      25      29      33      37
22      26      30      34      38
23      27      31      35      39
24      28      32      36      40

section 2 :
1       5       9       13      17
2       6       10      14      18
3       7       11      15      19
4       8       12      16      20

```

test Select4 (ordered, array)

section 0 :

0	4	8	12	16
1	5	9	13	17
2	6	10	14	18
3	7	11	15	19

section 1 :

20	24	28	32	36
21	25	29	33	37
22	26	30	34	38
23	27	31	35	39

section 2 :

40	44	48	52	56
41	45	49	53	57
42	46	50	54	58
43	47	51	55	59

After sorting :

section 0 :

0	4	8	12	16
1	5	9	13	17
2	6	10	14	18
3	7	11	15	19

section 1 :

20	24	28	32	36
21	25	29	33	37
22	26	30	34	38
23	27	31	35	39

section 2 :

40	44	48	52	56
41	45	49	53	57
42	46	50	54	58
43	47	51	55	59

test Select4 (random, array)

section 0 :

41	47	34	40	29
4	18	18	22	44
5	5	1	27	1
11	55	2	27	36

section 1 :

51	24	2	33	52
22	21	56	38	35
47	6	11	18	9
52	47	19	55	54

section 2 :

23	51	2	33	33
44	21	31	53	28
47	44	22	57	57
19	23	21	49	58

After sorting :

section 0 :

1	5	18	29	41
1	5	22	34	44
2	11	27	36	47
4	18	27	40	55

section 1 :

2	18	24	47	52
6	19	33	47	54
9	21	35	51	55
11	22	38	52	56

section 2 :

2	22	31	44	53
19	23	33	47	57
21	23	33	49	57
21	28	44	51	58



```
test Select4 (backordered, array)
```

```
section 0 :
```

60	56	52	48	44
59	55	51	47	43
58	54	50	46	42
57	53	49	45	41

```
section 1 :
```

40	36	32	28	24
39	35	31	27	23
38	34	30	26	22
37	33	29	25	21

```
section 2 :
```

20	16	12	8	4
19	15	11	7	3
18	14	10	6	2
17	13	9	5	1

```
After sorting :
```

```
section 0 :
```

41	45	49	53	57
42	46	50	54	58
43	47	51	55	59
44	48	52	56	60

```
section 1 :
```

21	25	29	33	37
22	26	30	34	38
23	27	31	35	39
24	28	32	36	40

```
section 2 :
```

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

```

Please input the number of menu(0-back):1
Sizes of array: P=3 M=4 N=5
test select1 (ordered, vector)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
sorting:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
test select1 (random, vector)
1 7 14 0 9 4 18 18 2 4 5 5 1 7 1 11 15 2 7 16
sorting:
0 1 1 1 2 2 4 4 5 5 7 7 7 9 11 14 15 16 18 18
test select1 (backordered, vector)
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
sorting:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```

```

Please input the number of menu(0-back):2
Sizes of array: P=3 M=4 N=5
test InsertExchange (ordered, vector)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
sorting:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
test InsertExchange (random, vector)
11 4 2 13 12 2 1 16 18 15 7 6 11 18 9 12 7 19 15 14
sorting:
1 2 2 4 6 7 7 9 11 11 12 12 13 14 15 15 16 18 18 19
test InsertExchange (backordered, vector)
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
sorting:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```

```

Please input the number of menu(0-back):3
Sizes of array: P=3 M=4 N=5
test Select4 (ordered, vector)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
sorting:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Select4 (random, vector)
3 11 2 13 13 4 1 11 13 8 7 4 2 17 17 19 3 1 9 18
sorting:
1 1 2 2 3 3 4 4 7 8 9 11 11 13 13 13 17 17 18 19
Select4 (backordered, vector)
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
sorting:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```

## О системе

### Характеристики устройства

#### Personal Computer

Имя устройства	Vovka
Процессор	Intel(R) Pentium(R) CPU B950 @ 2.10GHz 2.10 GHz
Оперативная память	6,00 ГБ
Код устройства	C35310FF-3FC0-4304-88CB-29616CA8 5D87
Код продукта	00326-10000-00000-AA248
Тип системы	64-разрядная операционная система, процессор x64
Перо и сенсорный ввод	Для этого монитора недоступен ввод с помощью пера и сенсорный ввод

## Результати досліджень

### Дослідження I:

Залежність часу роботи алгоритмів від форми перерізу масива(для дослідження кількість перерізів беру меншу ніж потрібно це обумовлено станом ноутбука)

```
Time measurement
Sizes of array: P=2 M=10000 N=10

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
Ordered      Random      BackOrdered
Select1(array) 40241.70    39167.60    48692.90

InsertExchange(array) 59635.80    100615.80    141302.84

Select4(array) 45283.85    42932.50    28950.45

Time measurement
Sizes of array: P=2 M=1000 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
Ordered      Random      BackOrdered
Select1(array) 40822.90    40782.30    47814.30

InsertExchange(array) 52240.70    81862.05    111543.30

Select4(array) 34739.75    32651.85    23658.70
```

```

Time measurement
Sizes of array: P=2 M=100 N=1000

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8

```

	Ordered	Random	BackOrdered
Select1(array)	42253.35	41907.30	48544.55
InsertExchange(array)	52954.60	85723.80	114162.00
Select4(array)	35180.80	33023.95	23163.10

```

Time measurement
Sizes of array: P=2 M=10 N=10000

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8

```

	Ordered	Random	BackOrdered
Select1(array)	35235.25	36140.60	45767.00
InsertExchange(array)	59487.50	95696.75	114458.95
Select4(array)	34607.80	32254.10	22850.55

```

Time measurement
Sizes of array: P=2 M=10 N=10000

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

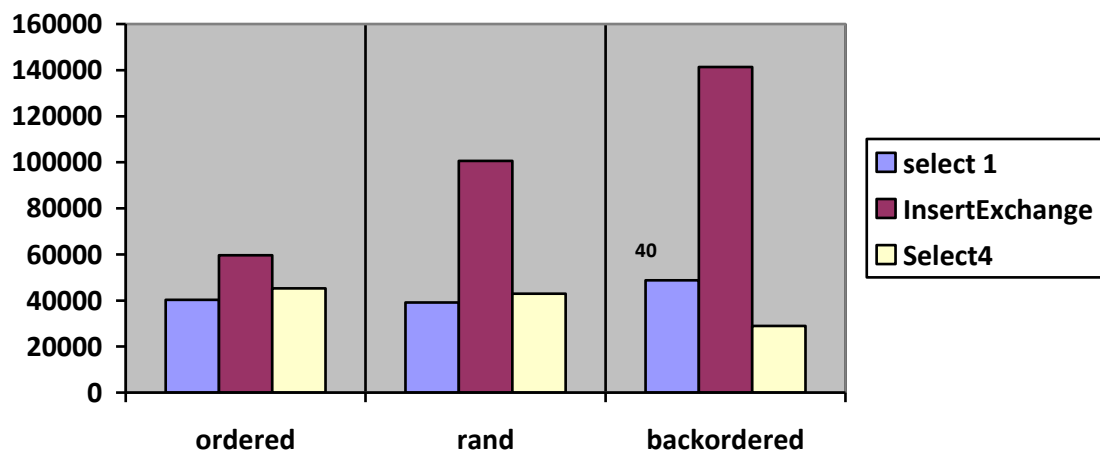
Please input the number of menu(0-back):7

```

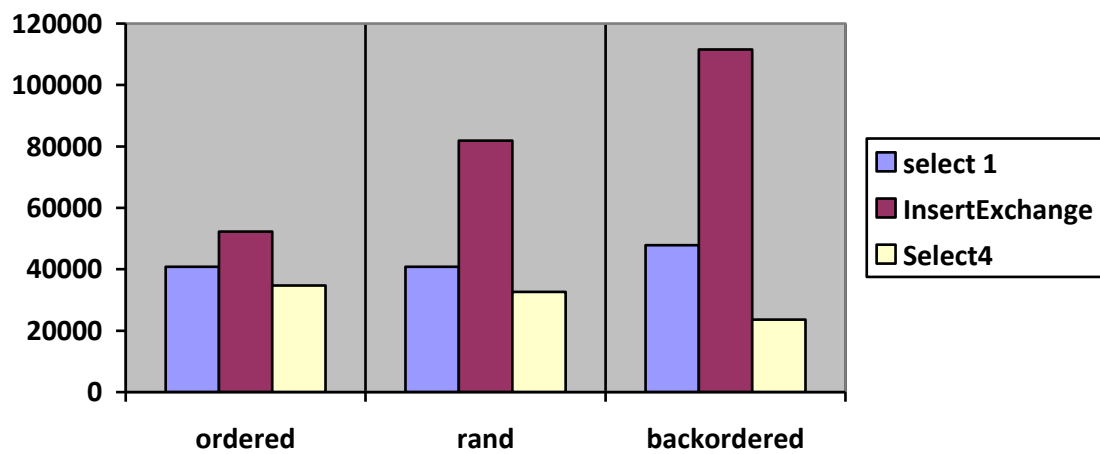
	Ordered	Random	BackOrdered
Select1(vector)	35434.60	36175.40	35119.90
InsertExchange(vector)	0.70	31097.70	62520.70
Select4(vector)	9936.75	9959.95	8832.60

### Графіки результатів

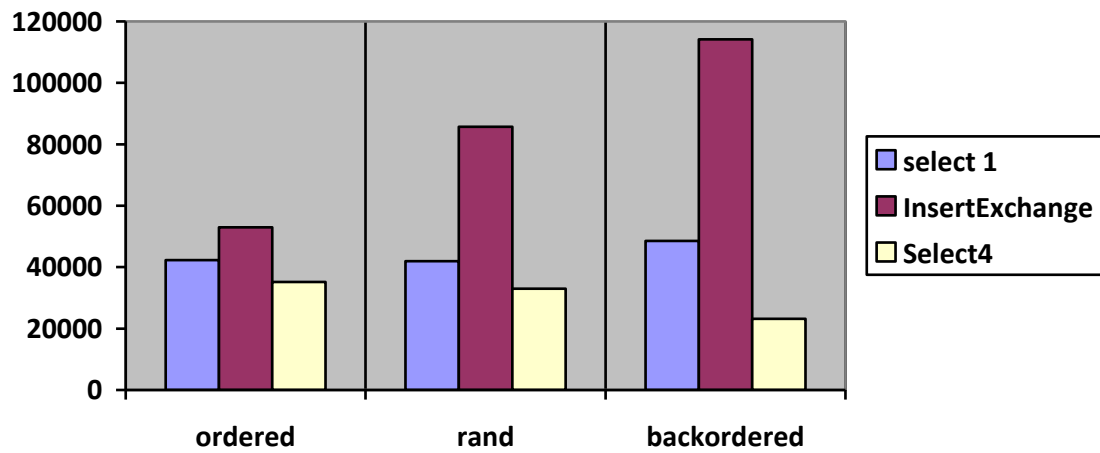
$P = 2$   $N = 10000$   $M = 10$



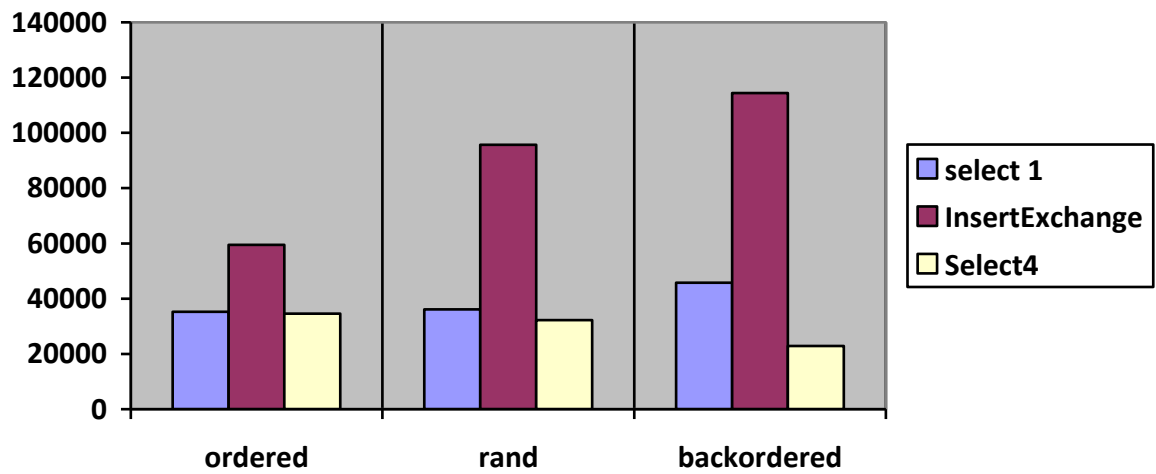
$P = 2$   $N = 1000$   $M = 100$



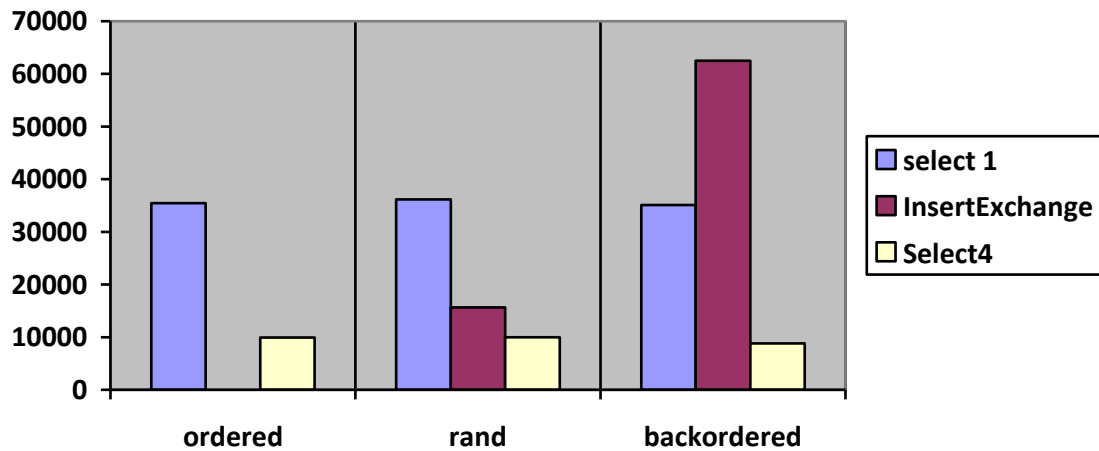
$P = 2$   $N = 100$   $M = 1000$



$P = 2 \quad N = 10 \quad M = 10000$



для вектору(  $P = 2$   $N = 10$   $M = 10000$ )



## Результати дослідження II. Залежність часу роботи алгоритмів від кількості перерізів масива

```
Time measurement
Sizes of array: P=2 M=100 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8

      Ordered      Random      BackOrdered
Select1(array)      432.55      432.55      498.25

InsertExchange(array)      593.35      915.65      1240.85

Select4(array)      378.10      364.20      252.25
```



```
Time measurement
Sizes of array: P=4 M=100 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
```

	Ordered	Random	BackOrdered
Select1(array)	909.85	894.45	1003.15
InsertExchange(array)	1242.10	1827.40	2616.15
Select4(array)	858.40	767.05	527.95

```
Time measurement
Sizes of array: P=8 M=100 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
```

	Ordered	Random	BackOrdered
Select1(array)	1904.70	1811.50	2118.45
InsertExchange(array)	2424.10	3784.85	5138.20
Select4(array)	1409.10	1343.25	935.80

```
Time measurement
Sizes of array: P=16 M=100 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
```

	Ordered	Random	BackOrdered
Select1(array)	3194.20	3237.80	3619.00
InsertExchange(array)	4142.90	6375.75	8763.05
Select4(array)	2673.20	2603.45	1759.75

```
Time measurement
Sizes of array: P=32 M=100 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
```

	Ordered	Random	BackOrdered
Select1(array)	6153.65	6199.05	6932.90
InsertExchange(array)	8013.35	12585.95	17833.45
Select4(array)	5375.35	5191.40	3666.20

```
Time measurement
Sizes of array: P=64 M=100 N=100

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8
```

	Ordered	Random	BackOrdered
Select1(array)	15389.05	15236.50	16669.75
InsertExchange(array)	18869.00	29692.35	40319.50
Select4(array)	12402.90	10346.70	8182.75

```
Time measurement
Sizes of array: P=64 M=100 N=100

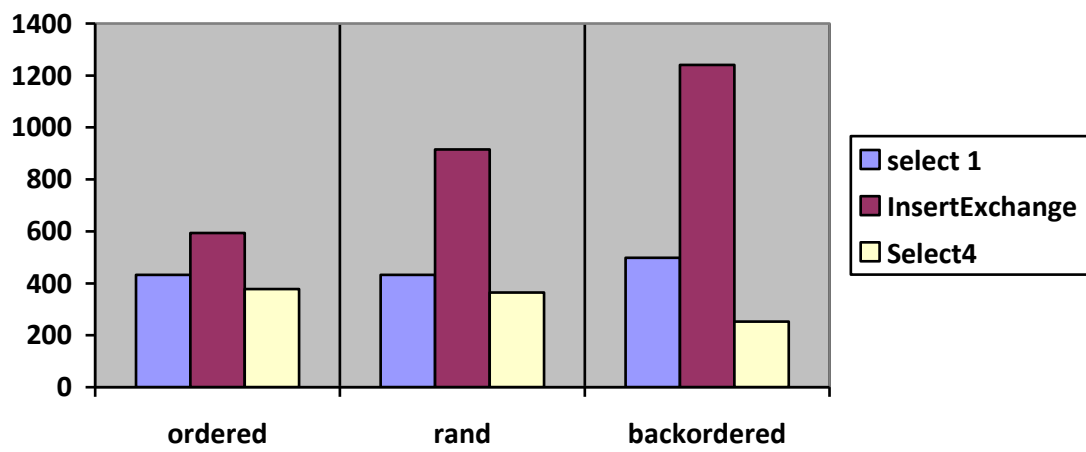
Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):7
```

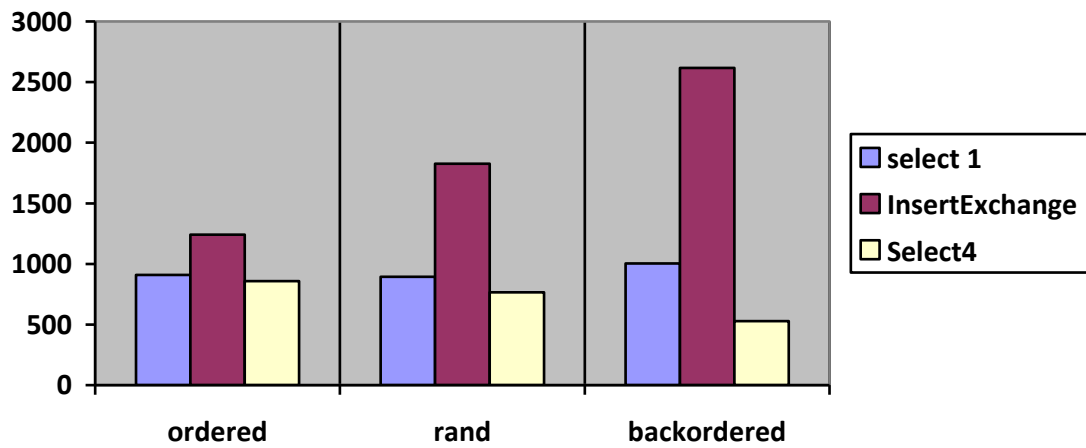
	Ordered	Random	BackOrdered
Select1(vector)	11382.40	12201.60	11753.60
InsertExchange(vector)	0.00	10297.60	20755.20
Select4(vector)	102.75	103.30	89.80

графіки результатів

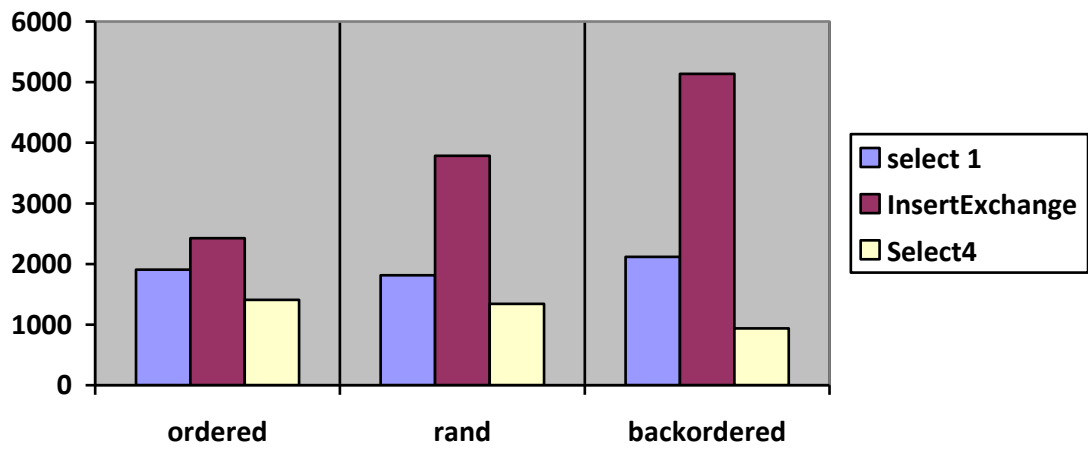
$P = 2$   $N = 100$   $M = 100$



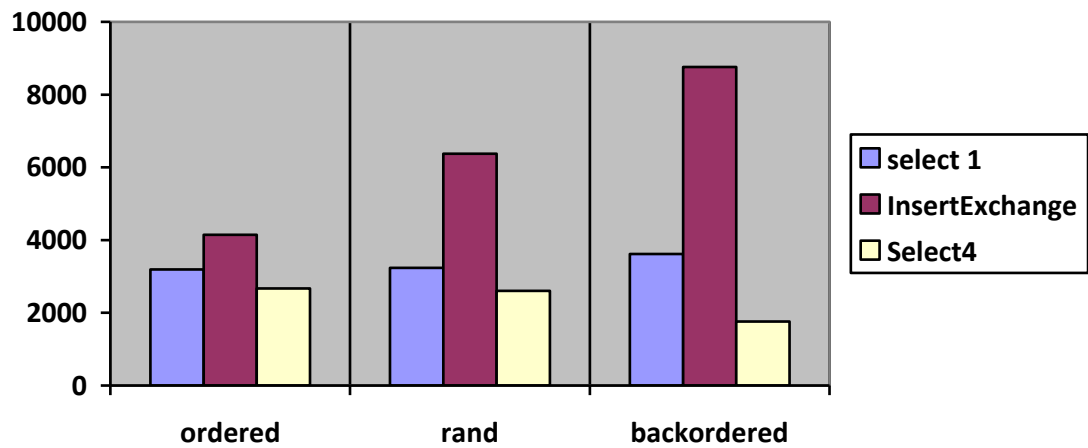
$P = 4$   $N = 100$   $M = 100$



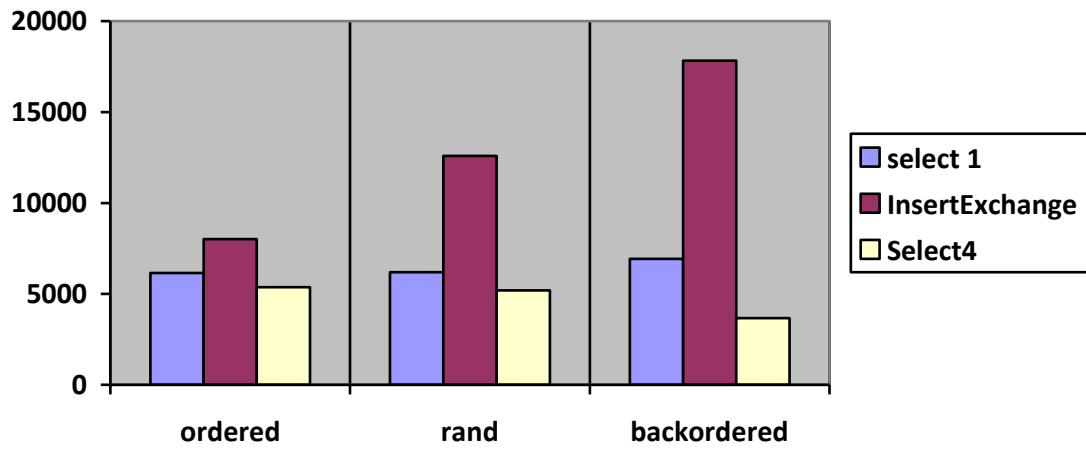
P = 8 N = 100 M = 100



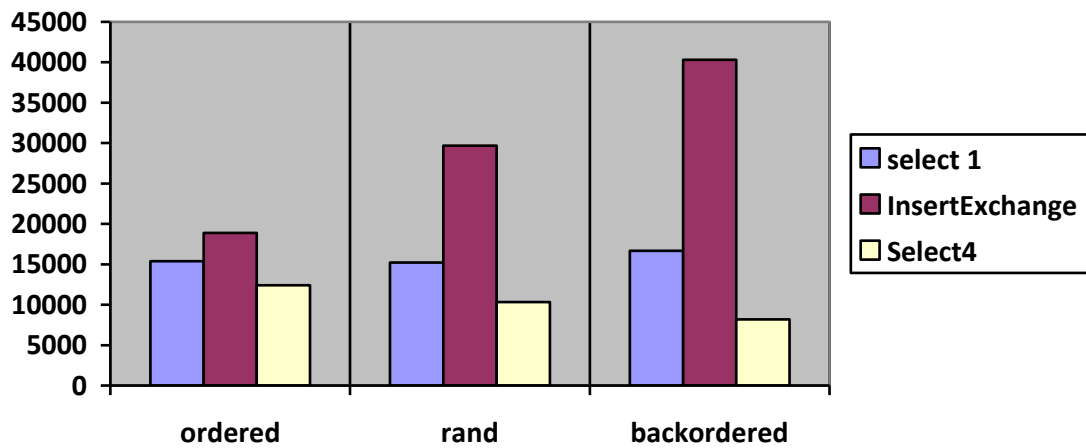
P = 16 N = 100 M = 100



P = 32 N =100 M =100

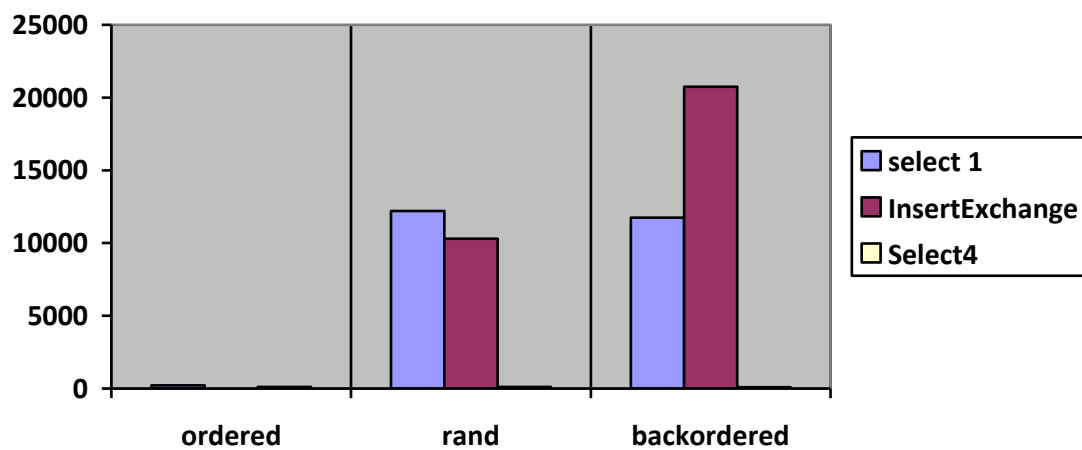


P = 64 N =100 M =100



*Для вектора*

P = 64 N = 100 M = 100



### Результати дослідження III. Залежність часу роботи алгоритмів від розміру перерізів масива

Time measurement Sizes of array: P=3 M=4 N=4			
Choose sorting: 1.Select1(vector) 2.InsertExchange(vector) 3. Select4 (vector) 4. select1(array) 5.InsertExchange(array) 6.Select4(array) 7.Pack mode(vector) 8.Pack mode(array)			
Please input the number of menu(0-back):8			
	Ordered	Random	BackOrdered
Select1(array)	0.00	0.00	0.00
InsertExchange(array)	0.00	0.00	0.00
Select4(array)	0.00	0.00	0.00
Time measurement Sizes of array: P=3 M=8 N=8			
Choose sorting: 1.Select1(vector) 2.InsertExchange(vector) 3. Select4 (vector) 4. select1(array) 5.InsertExchange(array) 6.Select4(array) 7.Pack mode(vector) 8.Pack mode(array)			
Please input the number of menu(0-back):8			
	Ordered	Random	BackOrdered
Select1(array)	0.00	0.00	0.00
InsertExchange(array)	0.00	0.00	0.00
Select4(array)	0.00	0.00	0.00
Time measurement Sizes of array: P=3 M=16 N=16			
Choose sorting: 1.Select1(vector) 2.InsertExchange(vector) 3. Select4 (vector) 4. select1(array) 5.InsertExchange(array) 6.Select4(array) 7.Pack mode(vector) 8.Pack mode(array)			
Please input the number of menu(0-back):8			
	Ordered	Random	BackOrdered
Select1(array)	0.40	0.50	0.55
InsertExchange(array)	0.55	1.00	1.20
Select4(array)	0.40	0.40	0.20

```

Time measurement
Sizes of array: P=3 M=32 N=32

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8

```

	Ordered	Random	BackOrdered
Select1(array)	7.50	7.65	8.00
InsertExchange(array)	9.00	14.60	19.85
Select4(array)	6.05	6.00	3.90

```

Time measurement
Sizes of array: P=3 M=64 N=64

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8

```

	Ordered	Random	BackOrdered
Select1(array)	114.80	115.70	127.60
InsertExchange(array)	132.95	223.85	415.85
Select4(array)	108.00	143.60	61.35

```

Time measurement
Sizes of array: P=3 M=128 N=128

Choose sorting:
1.Select1(vector)
2.InsertExchange(vector)
3. Select4 (vector)
4. select1(array)
5.InsertExchange(array)
6.Select4(array)
7.Pack mode(vector)
8.Pack mode(array)

Please input the number of menu(0-back):8

```

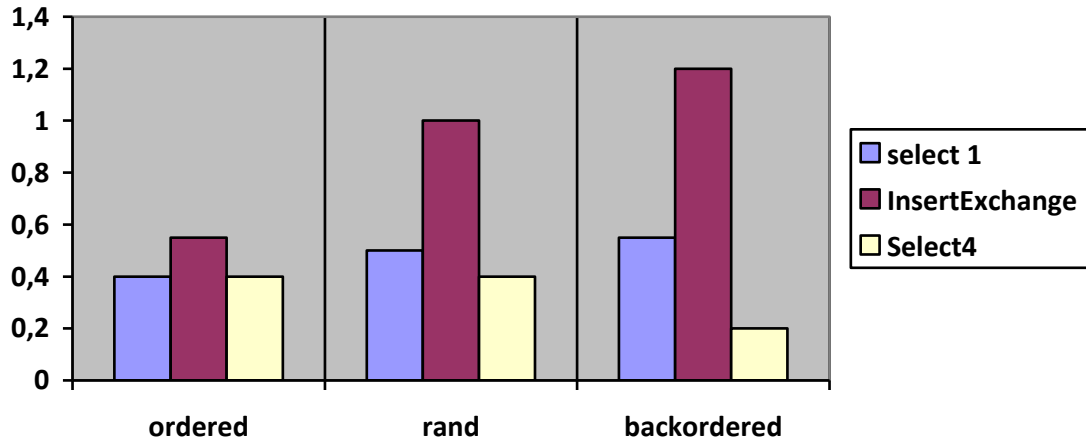
	Ordered	Random	BackOrdered
Select1(array)	2080.45	2331.05	2701.25
InsertExchange(array)	2621.50	4048.95	5444.45
Select4(array)	1959.45	1803.05	1264.95

графіки результатів данного дослідження

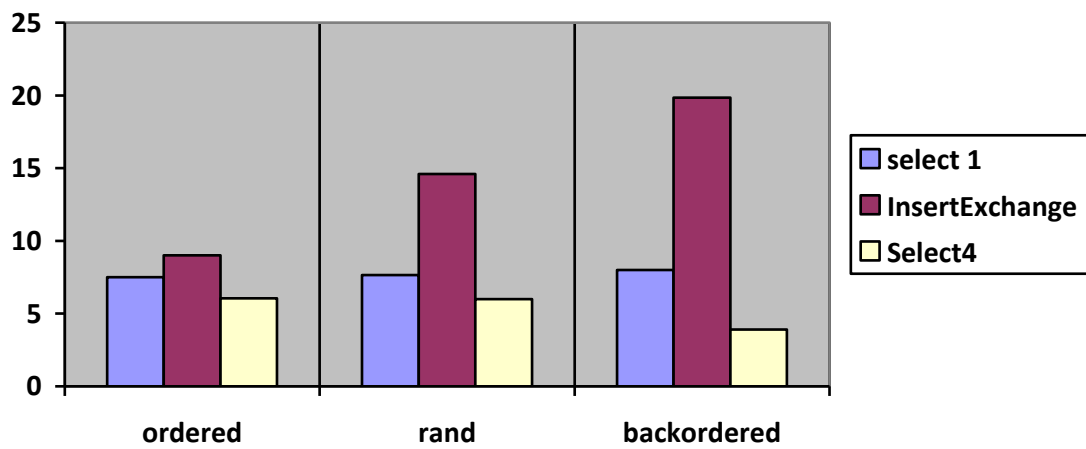
Оскільки перші два дослідження показали 0 виміру часу у всіх алгоритмах , тому для них графіка не буде , оскільки тут очевидно що всі вони працюють швидко.



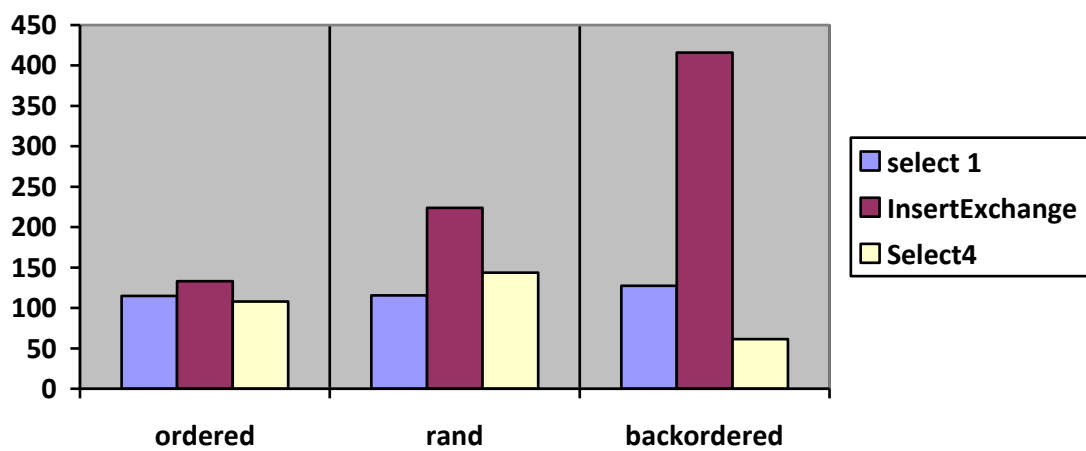
P=3 M=16 N=16



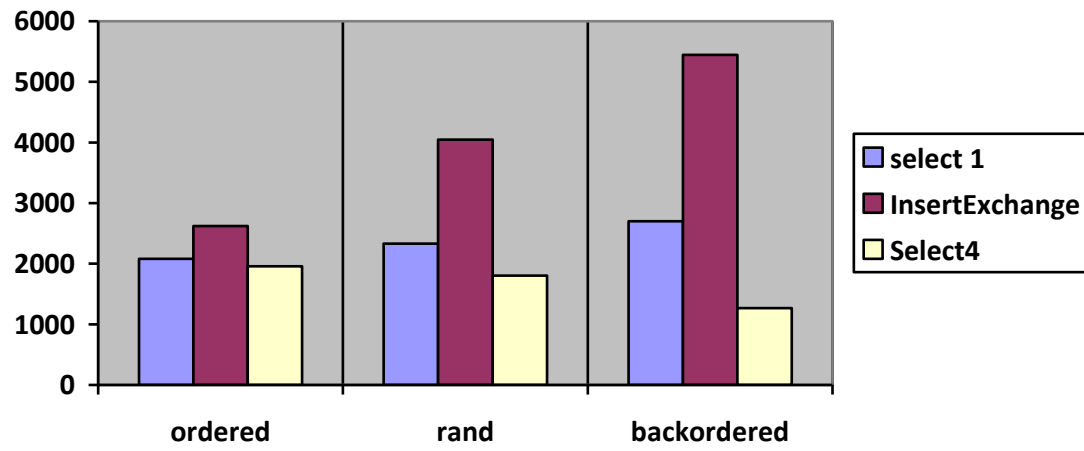
P=3 N=32 M=32;



P=3 N=64 M=64;



P = 3 N = 128 M = 128;



# Порівняльний аналіз отриманих результатів

Провдемо аналіз кожного алгоритму окремо за такою умовою «Чому при різному заповненні масиву саме такий час як наведений на знімках екрану?»

## Select 1

2.2. Сортировка массивов				107
Таблица 2.9. Время выполнения программ сортировки				
	Упорядоченный массив		Случайный массив	Упорядоченный в обратном порядке массив
Простое включение	12	23	366	1444
Бинарное включение	56	125	373	1327
Простой выбор	489	1907	509	1956
Метод пузырька	540	2165	1026	4054
Метод пузырька с ограничением	5	8	1104	4270
Шейкер-сортировка	5	9	961	3642
Сортировка Шелла	58	116	127	349
Пирамидальная сортировка	116	253	110	241
Быстрая сортировка	31	69	60	146
Сортировка слиянием*)	99	234	102	242

\*) См. разд. 2.3.1.

число в каждой колонке дано для массива из 256 элементов, а правое — для 512 элементов. Эти данные демонстрируют

Порівнюючи результати які отримані під час дослідження і результатами Вірта не можна спиратись остаточно це буде не коректно , оскільки хоч і результати відносно пропорційні і очікувані що найшвидше сортуватиметься відсортований масив, а найповільніше обрєнено відсортований( це зумовлено тим що при обрєнено відсортованому масиві алгоритм прямого вибору

$C = (n^2 - n) / 2$  , та  $M = (n^2) / 4 + 3(n - 1)$ , в той час як відсортованому С лишиться без змін, а от  $M = 3(n - 1)$  ) адже ми сортуємо більший за розмірами масив.

## InsertExchange

Оскільки його немає в таблиці Вірта тому оцінимо його наступним чином.

Він є гібридом тому швидше за все очікування що він працюватиме найкраще, проте це є не так і данні нашого дослідження це підтвердять (знімки океранів з результатами часу роботу алгоритмів).Чому саме так, даний алгоритм є поєднанням двох простих алгоритмів так звана бульбашка та вставки( принцип роботи і які особливості взято з кожного описані на початку роботи ).Оскільки велику кількість операцій проходить під час обміну елементами між собою за принципом бульбашки але взявши те що ми шукаємо найменший елемент за принципом вставки робить його кращим проте не настільки скільки нам хотілося.

Якщо порівнювати алгоритм по кількості проходів циклу та операцій що виконуються в них під час сортування, то для відсотованого –  $C = n(n^2 - n + 1)$  , а для обернено відсотованого  $C = (n^2) * (n^2 - n + 1)$ .

## Select 4

Як і попередній алгоритм з результатами Вірта ми порівнювати не можемо адже це його там і немає.

Даний алгоритм є модифікацією Select1 що одразу нас наводить на думку він є кращим ніж Select1. Так, він є кращим. Це пов'язано з тим що здавалося б і виконуємо більше перевірок та присвоєнь, але через те що ми одночасно і

шукаємо мінімальне та максимальне і після проходження пошуку одразу ставимо їх на місця це і пришвидшує алгоритм.

Адже всі присвоєння і порівняння які відбуваються відбуваються у гіршому випадку  $C = (n/2 + 8n) + n$  разів, в кращому випадку  $C = n/2 + 8n$ .

### ***Порівняємо данні алгоритми між собою***

Найкращим серед них за аналізом select4 далі select1 та insertexchange.

Це все зумовлено їхніми певними особливостями кількістю присвоєнь характеристикою робити та властивостями.

### **Аналіз дослідження I:**

Залежність часу роботи алгоритмів від форми перерізу масива

З вище наведених результатів роботи алгоритмів та гістограм приведенених до них ми може зробити такий висновок що

коли  $M \ll N$  час роботи всіх алгоритмів є меншим. це спричинено тим що під час проходження по масиву ми проходимось швидше по елементах кожного стовпчика а оскільки за умовою наша задача стоїть така щоб відсортувати наскрізного по стовпчиках то з відси слідує чим менше стовпчиків тим швидше буде відсортовано масив. При  $M \gg N$  все відбуватиметься повільніше адже стовпчиків є багато по декілька елементів що і зумовлює більше операцій які будуть виконуватись під час сортування.

Чому найкраще себе поводить саме select4 серед усіх це обумовлено тим що як і у select1 основною властивістю є те що ми переставляємо елементи на потрібну позицію на великій відстані. Але select4 працює краще оскільки він одразу ставить на свої місця і мінімальний і максимальний елементи.

А insertexchange працює найгірше бо переставляє елементи що стоять поруч.

Якщо порівнювати результати часу роботи алгоритмів які сортували тривимірний масив та вектор, чітко прослідковується що select1 працює завжди в межах одного і того ж часу незалежно чи вектор чи масив одного і того ж розміру. Це обумовлено тим що сортування завжди проходить по всіх елементах чи то є масив чи вектор.

Через те що insertexchange працює за умови коли минулий елемент є більший ніж теперішній тому для відсортованого масиву це все проходить досить швидко, а для інших типів відсортованості росте прямопропорційно їхній впорядкованості.

Select4 працює навпаки чим більш невідсортований масив тим менш його часу потрібно для сортування це зумовлено в умові пошуку мінімального і максимального елементів в тілі циклу. (тим що ми шукаємо спершу мінімальний а якщо умова невиконується переходимо в гуклу на перевірку для максимального, тому наочно бачимо що при відсортованому масиві порівняння умов виконується більше ніж при обрешено-відсортованому.

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

Проаналізуємо результати між тривимірними масивами, на гістограмах чітко прослідковуються те що сортування залежить в більшості не від кількості перерізів і від кількості елементів що знаходяться в данному перерізі. Це є наслідком того що розміри  $M = N$  і завжди є сталими, тобто ми завжди сортуємо одну і ту саму кількість елементів, лише з більшою кількістю перерізів. також це видно з графіків якщо не дивитись на шкалу то чітко видно що всі розміри стовпців приблизно однакові.

Тому серед них можна розподілити так найкращий є алгоритм select4, далі select1 та найгірший insertexchange.

Порівнюючі час сортування з вектором який задано в нас за умовою, то ми

бачимо що час сортування вектора надзвичайно відрізняється від часу сортування тривимірної масиву. Це спричинено тим що під час сортування тривимірної масиву при переході на новий переріз ми сортуємо елементи в цьому перерізі (звісно було б і припустити що оскільки кількість елементів в перерізах однакова завжди то логічно було б помножити час сортування вектора на кількість перерізів)

і виконуємо сортування всіх елементів перерізу що рази при переході на новий переріз, а при сортуванні вектора ми один раз сортуємо і все. А поведінку та час роботи даних алгоритмів сортування на векторах ми знаємо тому як вони були описані вище.

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

З попередніх двох досліджень ми вивчили що час сортування масивів не залежить від кількості перерізів, а від розміру масиву. З першого дослідження ми виявили що чим менше стовпчиків тим швидше сортується, тому тут ми вважаємо переріз квадратним і збільшуємо його. Чітко прослідковується те що чим більший розмір тим довше сортує, з гістограм якщо не зважати на шкалу то всі алгоритми поводяться однотипно притаманним їм. Як і в попередніх дослідженнях найгіршим серед трьох є insertexchange далі друге місце займає select1 та найкращим серед даних є select4.

Також це дослідження чітко підкреслює особливості часу сортування кожного з алгоритмів в залежності від способу їх заповнення.

## ***Висновок***

Порівнявши всі результати всіх алгоритмів, можна сказати що найкращим є select4 далі йде select1 та найгірший показує себе insertexchange.

Якщо потрібно потрібно відсортувати обернено- впорядкований масив то найкраще підійде для цього select4. Для сортування невідсортованого показує себе краще select 4 проте select1 не відстає від лідера. За допомогою insertexchange можна легко визначити чи впорядкований масив чи ні , якщо час сортування був наближений до 0 тоді масив дійсно впорядкований від початку.

З данної роботи ми дізнались те що при сортування наскрізно по стовчиках ідеальним варіантом буде коли в матриці найменша кількість стовпчиків, найгіршим варіантом є найбільша кількість стовпчиків і найменша рядків. І великої ролі не грає кількість перерізів в масиві.

Тому кожен з методів потрібно використовувати там де він працює як найкращий.

## Використана література

1 Конспект з СДА

2 « Алгоритми +структури даних = програми» Н. Вірт (1977)