**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

**УКРАЇНИ «КПІ ім. Ігоря Сікорського»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих**

**комп’ютерних систем**

**КУРСОВА РОБОТА**

***з дисципліни "Структури даних і алгоритми"***

Виконав: Савицький Я. В.

Група: КB-84

Номер залікової книжки: КВ-8423

Допущений до захисту

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

2 семестр 2018/2019 навч.року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

**УКРАЇНИ «КПІ ім. Ігоря Сікорського»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих**

**комп’ютерних систем**

Узгоджено ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_20\_\_р.

Керівник роботи з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_/Марченко О.І./ \_\_\_\_\_\_\_\_\_ /Марченко О.І./

**Дослідження ефективності методів сортування**

**(алгоритм сортування №1 методу прямої вставки, гібридний алгоритм "вставка – обмін", гібридний алгоритм "вибір№4 – обмін")**

**на багатовимірних масивах**

Виконавець роботи:\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Савицький Ярослав Вікторович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ 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.

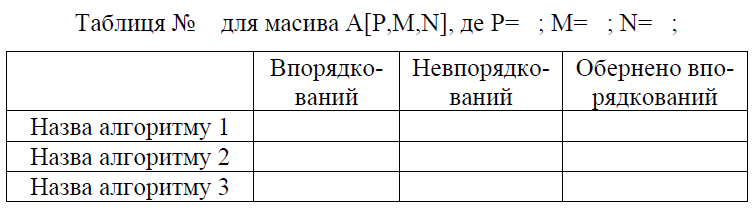
II. Скласти алгоритми рішення задачі сортування в багато-вимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

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 оберненовпорядкованого масива) для масива з заданими геометричними розмірами повинна бути такою:

****

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 алгоритмі 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 № 158**

**Задача**

Впорядкувати окремо кожен переріз тривимірного масива Аrr3D [P,M,N] наскрізно по стовпчиках за незменшенням.

**Досліджувані методи та алгоритми**

Алгоритм сортування №1 методу прямої вставки, гібридний алгоритм "вставка – обмін", гібридний алгоритм "вибір№4 – обмін"

**Способи обходу**

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

**Випадки дослідження**

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

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

**ІІІ.** Залежність часу роботи алгоритмів від розміру перерізів масива.

**Опис теоритичних положень**

Сортування застосовується в усіх без винятку областях програмування, хоч то бази даних чи математичні програми. Методи сортування можна поділити на наступні групи:

1. Методи сортування, які не використовують додаткову пам’ять (ці методи називаються методами сортування на тому ж місці);
2. Методи сортування, які використовують додаткову пам’ять такого ж розміру як задана структура, або більше.

Нас цікавить перша группа методів, тому що вона використовується для сортування масивів і рідко для сортування файлів.

**Загальна характеристика методів сортування на тому ж місці**

Методи сортування на тому ж місці можна поділити на 3 групи:

1. Сортування вставкою;
2. Сортування вибором;
3. Сортування обміном.

Кожна з груп-методів має прямий метод (найпростіший і найповільніший, як правило) і покращеній методи сортування. Покращені методи базуються на тих же методах, що і прямі, але використовують деякі оригінальні ідеї , які суттєво прискорюють процес сортування.

В даній курсовій роботі ми розглянемо:

Алгоритм сортування №1 методу прямої вставки, гібридний алгоритм "вставка – обмін", гібридний алгоритм "вибір№4 – обмін"

## Схема роботи заданих алгоритмів

**1.Алгоритм сортування №1 методу прямої вставки (з ліній-**

**ним пошуком місця вставки від початку послідовності, що сорту-**

**ється , або «зліва»)**

Принцип метода:

Масив розділяється на дві частини відсортовану і невідсортовану.

Елементи з невідсортованої частини почергово вибираються і вставляються в відсортовану частину так, щоб не порушити в ній впорядкованість елементів.

На початку роботи алгоритму в якості відсортованої частини масиву береться

тільки один перший елемент, а в якості невідсортованої частини – всі інші елементи.

Таким чином, алгоритм буде складатися з **n-1**-гопроходу, кожен з яких буде включати чотири дії:

* взяття чергового ***i***-того невідсортованого елемента і збереження його в додатковій змінній ;
* пошук позиції ***j*** у відсортованій частині масиву, в якій присутність взятого елемента не порушить впорядкованість елементів;
* зсув елементів масиву від ***i-1***-го до ***j***-го вправо, щоб звільнити знайдену позицію вставки;
* вставка взятого елемента в знайдену ***j***-ту позицію.

Кількість порівнянь і присвоєнь для цього алгоритму порядку .

**Алгоритм сортування мовою С++**

int Elem, j;

for (int i = 1; i < M \* N; i++)

{

Elem = V[i];

j = 0;

while (Elem > V[j])

j = j + 1;

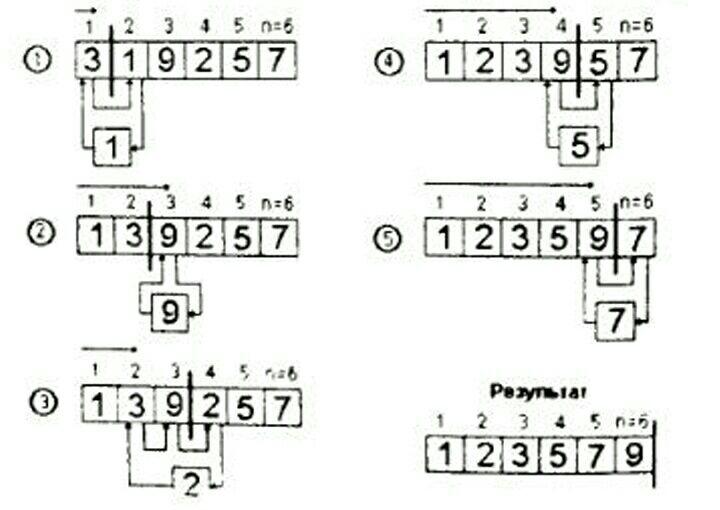
for (int k = i - 1; k >= j; k--)

V[k + 1] = V[k];

V[j] = Elem;

}

**Схема сортування методом вставки**



**2.Гібридний алгоритм "вставка – обмін"**

Принцип метода:

Масив розділяється на дві частини відсортовану і невідсортовану.

Елементи з невідсортованої частини почергово вибираються і почергово порівнюються два сусідні елементи(при першому взятті елемента це останній елемент відсортованої частини і перший з невідсортованої, надалі рухаємося по відсортованій частині адже шукаємо його позицію саме в ній), якщо їх взаєморозміщення не відповідає заданій умові впорядкованості, то вони міняються місцями. Такі дії повторюємо доки не дійдемо до кінця масиву.

На початку роботи алгоритму в якості відсортованої частини масиву береться

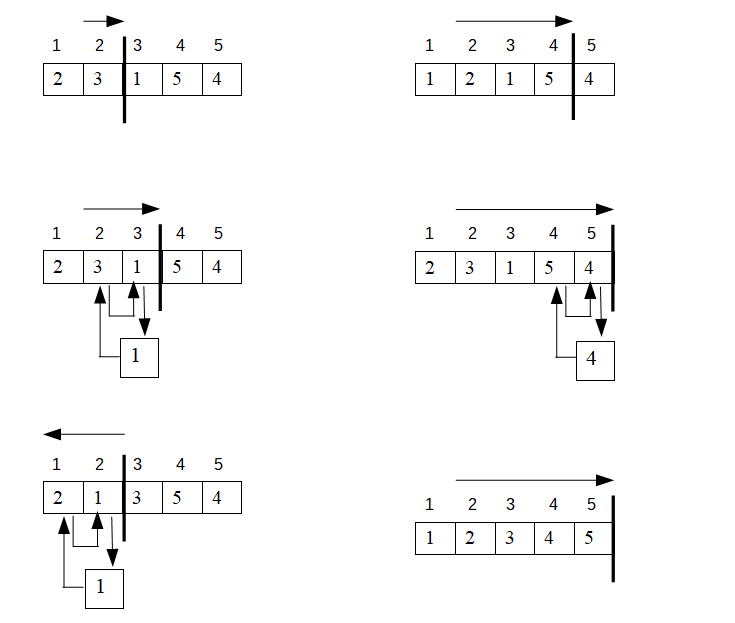
тільки один перший елемент, а в якості невідсортованої частини – всі інші елементи.

Таким чином, алгоритм буде складатися з **n-1**-гопроходу, кожен з яких буде включати три дії:

* взяття чергового ***i***-того невідсортованого елемента і порівння його з ***i-1***-шим елементом масиву;
* якщо елеменет задовільняє умову впорядкованості, то повторюємо 1 пункт;
* якщо елемент не задовільняє умову відсортованості, то міняємо його місцями з ***i-1***-им та повторюємо цей пункт доти, доки умова впорядкованості не буде виконана.

Кількість порівнянь і присвоєнь для цього алгоритму порядку .

**Схема сортування методом “вставки-обміну”**



**Алгоритм сортування мовою С++**

int j, tmp;

for (int i = 1; i < M \* N; i++)

{

j = i;

while (j > 0 && V[j] < V[j - 1])

{

tmp = V[j];

V[j] = V[j - 1];

V[j - 1] = tmp;

j = j - 1;

}

}

**2.Гібридний алгоритм "вибір№4 – обмін"**

Принцип метода:

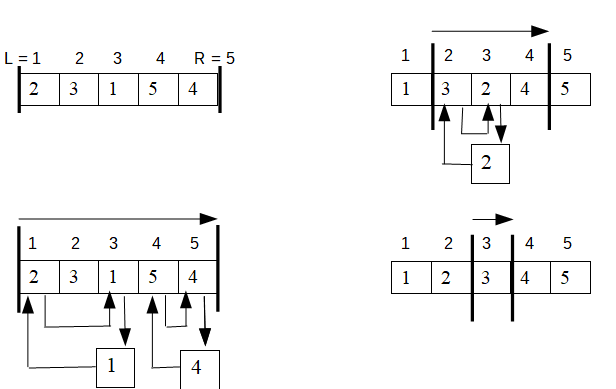
Кожен момент часу масив розглядається з двох частин, вже відсортованої та ще не відсортованої. Відсортована частина на початку дорівнює **0**. Почергово беремо кожен елемент та порівнюємо його з першим та останнім елементом масиву, якщо елемент менший першого, то міняємо їх місцями, якщо ж елемент більший за останній елемент масиву, то міняємо місцями саме їх. За один прохід по масиву ми обов’язково знайдемо мінімальний та максимальний елемент невідсортованої частини так поставимо його на потрібну позицію, яка буде задовільняти умову впорядкованості. Після цього змінюємо межі невідсортованої частина на 1 зліва та справа (**L = L + 1**, **R = R – 1**). Сортування закінчиться, коли невідсортована частина буде дорівнювати **1**.

Кожен прохід буде включати в себе такі дії:

* взяття чергового ***i***-того невідсортованого елемента і порівння його з лівою та правою границею відсортованості(при першому входженні ліва границя дорівню **0**, а права **N-1**);
* якщо ***i***-тий елемент менший за ліву границю, то міняємо їх місцями, якщо елемент більший за праву границю, то міняємо їх місцями;
* збільшуємо границі відсортованості на 1(**L = L + 1**, **R = R – 1**);
* повторюємо попередні пункти, доки невідсортована частина не буде дорівнювати **0**.

Кількість порівнянь і присвоєнь для цього алгоритму порядку .

**Схема сортування методом “ вибір№4 – обмін ”**

****

**Алгоритм сортування мовою С++**

int Min, Max;

int L, R;

L = 0; R = M \*N - 1;

while (L < R)

{

for (int i = L; i < R + 1; i++)

{

if (V[i] < V[L])

{

Min = V[i];

V[i] = V[L];

V[L] = Min;

}

else

if (V[i] > V[R])

{

Max = V[i];

V[i] = V[R];

V[R] = Max;

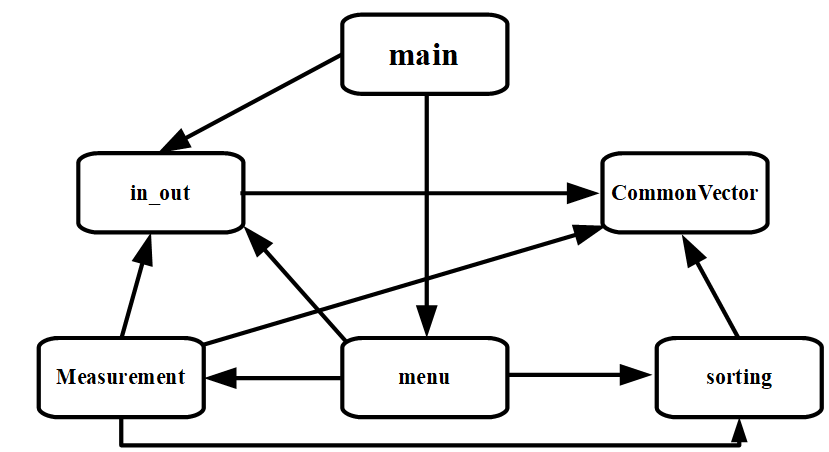
}

}

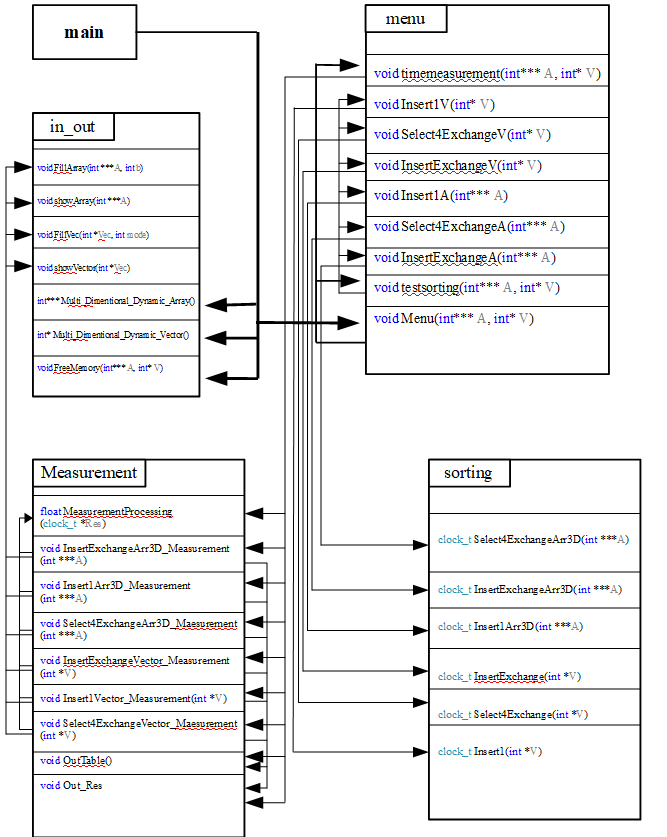
L = L + 1; R = R - 1;

}

**Схема експорту/імпорту модулів**



**Схема взаємовикликів функцій**



**Опис призначення процедур та функцій**

1. Модуль CommonVector – в цьому модулі выдбувається оголошення та визначення розмірів масиву.
2. Модуль menu:

* Menu – процедура для вибору діяльності в програмі;
* testsorting – функція, яка заповнює, сортує та виводить на екран масиви та вектори;
* timemeasurement – функція, яка виводить на екран виміру часу сортування;
* Insert1V – функція, яка заповнює, сортує методом вставки №1 та виводить вектор на екран;
* Select4ExchangeV – функція, яка заповнює, сортує методом «вибір№4 – обмін» та виводить вектор на екран;
* InsertExchangeV – функція, яка заповнює, сортує методом «вставка – обмін» та виводить вектор на екран;
* Insert1A – функція, яка заповнює, сортує методом вставки №1 та виводить масив на екран;
* Select4ExchangeА – функція, яка заповнює, сортує методом «вибір№4 – обмін» та виводить масив на екран;
* InsertExchangeА – функція, яка заповнює, сортує методом «вставка – обмін» та виводить масив на екран.

1. Модуль sorting:

* Select4ExchangeArr3D – функція сортування масиву методом «вибір №4 – обмін»;
* Insert1Arr3D – функція сортування масиву методом вставки №1;
* InsertExchangeArr3D функція сортування масиву методом «вставка – обмін»;
* Select4Exchange – функція сортування вектора методом «вибір №4 – обмін»;
* Insert1 – функція сортування вектора методом вставки №1;
* InsertExchange – функція сортування вектора методом «вставка – обмін».

1. Модуль Measurement:

* MeasurementProcessing – функція обраховування вимірів;
* InsertExchangeArr3D\_Measurement – функція, що заповнює масив, сортує його методом «вставка – обмін» та заповнює масив результатів;
* Insert1Arr3D\_Measurement – функція, що заповнює масив, сортує його методом вставки №1 та заповнює масив результатів;
* Select4ExchangeArr3D\_Maesurement – функція, що заповнює масив, сортує його методом «вибір №4 – обмін» та заповнює масив результатів;
* InsertExchangeVector\_Measurement – функція, що заповнює вектор, сортує його методом «вставка – обмін» та заповнює масив результатів;
* Insert1Vector\_Measurement – функція, що заповнює вектор, сортує його методом вставки №1 та заповнює масив результатів;
* Select4ExchangeVector\_Maesurement – функція, що заповнює вектор, сортує його методом «вибір №4 – обмін» та заповнює масив результатів;
* OutTable – функція, яка виводить заголовок таблиці;
* Out\_Res – функція, яка виводить виміру часу на екран.

1. Модуль in\_out:

* FillArray – функція, яка заповнює масив;
* showArray – функція, ка виводить масив на екран;
* FillVec – функція, яка заповнює вектор;
* showVector – функція, яка виводить вектор на екран;
* Multi\_Dimentional\_Dynamic\_Array – функція, яка виділяє динамічну пам'ять для тривимірного масиву;
* Multi\_Dimentional\_Dynamic\_Vector – функція, яка виділяє динамічну пам'ять для вектора;
* FreeMemory – функція, яка очищує виділену динамічну пам'ять.

**Текст програми**

**main.cpp**

#include <stdio.h>

#include <time.h>

#include <conio.h>

#include "in\_out.h"

#include "menu.h"

int main()

{

int\*\*\* A = NULL;

int\* V = NULL;

srand(time(NULL));

A = Multi\_Dimentional\_Dynamic\_Array();//функція для створення динамічного тривірмного масива

V = Multi\_Dimentional\_Dynamic\_Vector();//функція для створення динамічного масиву

Menu(A, V);//виклик меню

FreeMemory(A, V);//фунцкія для очищення пам'яті

\_getch();

return 0;

}

**CommonVector.h**

#pragma once

#ifndef \_\_CommonVector\_H\_\_

#define \_\_CommonVector\_H\_\_

#define P 3 //перерізи

#define M 512//рядки

#define N 512//стовпці

#endif

**menu.h**

#pragma once

#ifndef MENU\_H\_INCLUDED

#define MENU\_H\_INCLUDED

void Menu(int\*\*\* A, int\* V);

#endif // MENU\_H\_INCLUDED

**menu.cpp**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include "Measurement.h"

#include "sorting.h"

#include "in\_out.h"

void timemeasurement(int\*\*\* A, int\* V) //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);

printf("\nChoose sorting:\n");

printf("1.Insert1(vector)\n2.Select4Exchange (vector)\n");

printf("3.InsertExchange (vector)\n4.Insert1 (array)\n");

printf("5.Select4Exchange (array)\n6.InsertExchange(array)\n");

printf("7.Pack mode(array)\n8.Pack mode(vector)\n");

printf("\nPlease input the number of menu(0-back):");

scanf\_s("%d", &mode);

switch (mode)//вибір пункту меню

{

case 0: return; break;

case 1:

system("cls");

OutTable();

printf("Insert1");

Insert1Vector\_Measurement(V);

system("pause");

break;

case 2:

system("cls");

OutTable();

printf("Select4Exchange");

Select4ExchangeVector\_Maesurement(V);

system("pause");

break;

case 3:

system("cls");

OutTable();

printf("InsertExchange");

InsertExchangeVector\_Measurement(V);

system("pause");

break;

case 4:

system("cls");

OutTable();

printf("Insert1");

Insert1Arr3D\_Measurement(A);

system("pause");

break;

case 5:

system("cls");

OutTable();

printf("Select4Exchange");

Select4ExchangeArr3D\_Maesurement(A);

system("pause");

break;

case 6:

system("cls");

OutTable();

printf("InsertExchange");

InsertExchangeArr3D\_Measurement(A);

system("pause");

break;

case 7:

system("cls");

printf("Sizes of array:\n P=%d M=%d N=%d", P, M, N);

printf("\n");

OutTable();

printf("Insert1");

Insert1Arr3D\_Measurement(A);

printf("Select4Exchange");

Select4ExchangeArr3D\_Maesurement(A);

printf("InsertExchange");

InsertExchangeArr3D\_Measurement(A);

system("pause");

break;

case 8:

system("cls");

printf("Sizes of array:\n M=%d N=%d", M, N);

printf("\n");

OutTable();

printf("Insert1");

Insert1Vector\_Measurement(V);

printf("Select4Exchange");

Select4ExchangeVector\_Maesurement(V);

printf("InsertExchange");

InsertExchangeVector\_Measurement(V);

system ("pause");

break;

default:

printf("Please input CORRECT number (Enter)");

\_getch();

break;

}

} while (1);

}

void Insert1V(int\* V) //тестування сортування вставки на векторі

{

system("cls");

printf("Insert1\n");

printf("Ordered array\n");

FillVec(V, 1);

showVector(V);

Insert1(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\n\nRandom array\n");

FillVec(V, 2);

showVector(V);

Insert1(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\n\nBackOrdered array\n");

FillVec(V, 3);

showVector(V);

Insert1(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\nPress Enter for back");

\_getch();

}

void Select4ExchangeV(int\* V) //тестування сортування "вибір№4 – обмін" на векторі

{

system("cls");

printf("Select4Exchange\n");

printf("Ordered array\n");

FillVec(V, 1);

showVector(V);

Select4Exchange(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\n\nRandom array\n");

FillVec(V, 2);

showVector(V);

Select4Exchange(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\n\nBackOrdered array\n");

FillVec(V, 3);

showVector(V);

Select4Exchange(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\nPress Enter for back");

\_getch();

}

void InsertExchangeV(int\* V) //тестування сортування "вставка – обмін" на векторі

{

system("cls");

printf("InsertExchange\n");

printf("Ordered array\n");

FillVec(V, 1);

showVector(V);

InsertExchange(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\n\nRandom array\n");

FillVec(V, 2);

showVector(V);

InsertExchange(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\n\nBackOrdered array\n");

FillVec(V, 3);

showVector(V);

InsertExchange(V);

printf("\nAfter sorting\n");

showVector(V);

printf("\nPress Enter for back");

\_getch();

}

void Insert1A(int\*\*\* A) {//тестування сортування вставки на масиві

system("cls");

printf("Insert1\n");

printf("Ordered array\n");

FillArray(A, 1);

showArray(A);

Insert1Arr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\n\nRandom array\n");

FillArray(A, 2);

showArray(A);

Insert1Arr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\n\nBackOrdered array\n");

FillArray(A, 3);

showArray(A);

Insert1Arr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\nPress Enter for back");

\_getch();

}

void Select4ExchangeA(int\*\*\* A) //тестування сортування "вибір№4 – обмін" на векторі

{

system("cls");

printf("Select4Exchange\n");

printf("Ordered array\n");

FillArray(A, 1);

showArray(A);

Select4ExchangeArr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\n\nRandom array\n");

FillArray(A, 2);

showArray(A);

Select4ExchangeArr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\n\nBackOrdered array\n");

FillArray(A, 3);

showArray(A);

Select4ExchangeArr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\nPress Enter for back");

\_getch();

}

void InsertExchangeA(int\*\*\* A)//тестування сортування "вставка – обмін" на масиві

{

system("cls");

printf("InsertExchange\n");

printf("Ordered array\n");

FillArray(A, 1);

showArray(A);

InsertExchangeArr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\n\nRandom array\n");

FillArray(A, 2);

showArray(A);

InsertExchangeArr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\n\nBackOrdered array\n");

FillArray(A, 3);

showArray(A);

InsertExchangeArr3D(A);

printf("\nAfter sorting\n");

showArray(A);

printf("\nPress Enter for back");

\_getch();

}

void testsorting(int\*\*\* A, int\* V) {//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.Insert1(vector)\n2.Select4Exchange (vector)\n");

printf("3.InsertExchange (vector)\n4.Insert1 (array)\n");

printf("5.Select4Exchange (array)\n6.InsertExchange(array)\n");

printf("\nIf the window with sorting is closed, please, enter the number again");

printf("\nPlease input the number of menu(0-back):");

scanf\_s("%d", &mode);

switch (mode)//вибір виду сортування

{

case 0: return; break;

case 1:

Insert1V(V);

\_getch();

break;

case 2:

Select4ExchangeV(V);

\_getch();

break;

case 3:

InsertExchangeV(V);

\_getch();

break;

case 4:

Insert1A(A);

\_getch();

break;

case 5:

Select4ExchangeA(A);

\_getch();

break;

case 6:

InsertExchangeA(A);

\_getch();

break;

default:

printf("Please input CORRECT number (Enter)");

\_getch();

break;

}

} while (1);

}

void Menu(int\*\*\* A, int\* V) {//меню з вибором тестування часу і тестування сортувань

do {

system("cls");

printf("\n\n");

printf("\t\t\t\t -------------------------------------------------\n");

printf("\t\t\t\t| COURSE WORK |\n");

printf("\t\t\t\t| The study of the effectiveness of different |\n");

printf("\t\t\t\t| methods for sorting algorithms on multidime- |\n");

printf("\t\t\t\t| nsional arrays |\n");

printf("\t\t\t\t| Variant #158. |\n");

printf("\t\t\t\t| Made by Savytskiy Yaroslav |\n");

printf("\t\t\t\t| group KV - 84 |\n");

printf("\t\t\t\t ------------------------------------------------- \n");

printf("\n\n\n");

printf("\t\t\t\tChoose mode:\n\t\t\t\t1 - time measurement\n");

printf("\t\t\t\t2 - testing sorting\n\t\t\t\t3 - exit: ");

int vote;

scanf\_s("%d", &vote);

switch (vote)//вибір одного з пунктів

{

case 1: timemeasurement(A, V); break;

case 2: testsorting(A, V); break;

case 3: return;

default:

printf("\t\t\t\tPlease input CORRECT number (Press Enter)");

\_getch();

break;

}

} while (1);

}

**Measurements.h**

#pragma once

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

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

#include "CommonVector.h"//модуль з розмірами

#include <time.h>

#define measurements\_number 28//кількість вимірів

#define rejected\_number 2

#define min\_max\_number 3

extern clock\_t Res[measurements\_number];//вектор для часів впорядкованого

extern clock\_t Res1[measurements\_number];//вектор для часів невпорядкованого

extern clock\_t Res2[measurements\_number];//вектор для часів обернено-впорядкованого

//прототипи

void OutTable();

void Out\_Res(float ordered, float random, float backordered);

void Select4ExchangeArr3D\_Maesurement(int \*\*\*A);

void Insert1Arr3D\_Measurement(int \*\*\*A);

void InsertExchangeArr3D\_Measurement(int \*\*\*A);

void Select4ExchangeVector\_Maesurement(int \*V);

void Insert1Vector\_Measurement(int \*V);

void InsertExchangeVector\_Measurement(int \*V);

float MeasurementProcessing(clock\_t\* Res);

#endif

**Measurements.cpp**

#include "Measurement.h"//підключення інтерфейсної частини

#include <stdio.h>//бібліотека вводу-виводу

#include "in\_out.h"////модуль вводу-виводу масива-вектора

#include "sorting.h"

#include <conio.h>

clock\_t Res[measurements\_number];

clock\_t Res1[measurements\_number];

clock\_t Res2[measurements\_number];

void Out\_Res(float ordered, float random, float backordered);

float MeasurementProcessing(clock\_t \*Res)

{

long int Sum;

float AverageValue;

clock\_t buf;

int L = rejected\_number, R = measurements\_number - 1;

int k = rejected\_number;

for (int j = 0; j < min\_max\_number; j++) {

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

if (Res[i] > Res[i + 1]) {

buf = Res[i];

Res[i] = Res[i + 1];

Res[i + 1] = buf;

k = i;

}

}

R = k;

for (int i = R - 1; i >= L; i--) {

if (Res[i] > Res[i + 1]) {

buf = Res[i];

Res[i] = Res[i + 1];

Res[i + 1] = buf;

k = i;

}

}

L = k + 1;

}

Sum = 0;

for (int i = rejected\_number + min\_max\_number; i <

measurements\_number - min\_max\_number; i++)

Sum = Sum + Res[i];

AverageValue = (float)Sum / (float)(measurements\_number -

2 \* min\_max\_number - rejected\_number);

return AverageValue;

}

void InsertExchangeArr3D\_Measurement(int \*\*\*A)//виміру часу при гібридному алгоритмі "вставка – обмін" для масива

{

float ordered, random, backordered;

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

{

FillArray(A, 1);//заповнюємо масив впорядковано

Res[i] = InsertExchangeArr3D(A);//заповнюємо масив вимірами часу, які повертає функція

FillArray(A, 2);//заповнюємо масив рандомними значеннями

Res1[i] = InsertExchangeArr3D(A);//заповнюємо другий масив вимірами часу, які повертає функція

FillArray(A, 3);//заповнюємо масив обернено впорядковано

Res2[i] = InsertExchangeArr3D(A);//заповнюємо третій масив вимірами часу, які повертає функція

}

ordered = MeasurementProcessing(Res);

random = MeasurementProcessing(Res1);

backordered = MeasurementProcessing(Res2);

Out\_Res(ordered, random, backordered);//викликаємо функцію виведення результатів на консоль

}

//в подальших функціях ...\_Measurement, дії відбуваються аналогічно

void Insert1Arr3D\_Measurement(int \*\*\*A)//виміру часу при гібридному алгоритмі "вибір№4 – обмін" для масива

{

float ordered, random, backordered;

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

{

FillArray(A, 1);

Res[i] = Insert1Arr3D(A);

FillArray(A, 2);

Res1[i] = Insert1Arr3D(A);

FillArray(A, 3);

Res2[i] = Insert1Arr3D(A);

}

ordered = MeasurementProcessing(Res);

random = MeasurementProcessing(Res1);

backordered = MeasurementProcessing(Res2);

Out\_Res(ordered, random, backordered);

}

void Select4ExchangeArr3D\_Maesurement(int \*\*\*A)//виміру часу при алгоритмі сортування №1 методу прямої вставки для масива

{

float ordered, random, backordered;

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

{

FillArray(A, 1);

Res[i] = Select4ExchangeArr3D(A);

FillArray(A, 2);

Res1[i] = Select4ExchangeArr3D(A);

FillArray(A, 3);

Res2[i] = Select4ExchangeArr3D(A);

}

ordered = MeasurementProcessing(Res);

random = MeasurementProcessing(Res1);

backordered = MeasurementProcessing(Res2);

Out\_Res(ordered, random, backordered);

}

void InsertExchangeVector\_Measurement(int \*V)//виміру часу при гібридному алгоритмі "вставка – обмін" для вектора

{

float ordered, random, backordered;

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

{

FillVec(V, 1);

Res[i] = InsertExchange(V);

FillVec(V, 2);

Res1[i] = InsertExchange(V);

FillVec(V, 3);

Res2[i] = InsertExchange(V);

}

ordered = MeasurementProcessing(Res);

random = MeasurementProcessing(Res1);

backordered = MeasurementProcessing(Res2);

Out\_Res(ordered, random, backordered);

}

void Insert1Vector\_Measurement(int \*V)//виміру часу при гібридному алгоритмі "вибір№4 – обмін" для вектора

{

float ordered, random, backordered;

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

{

FillVec(V, 1);

Res[i] = Insert1(V);

FillVec(V, 2);

Res1[i] = Insert1(V);

FillVec(V, 3);

Res2[i] = Insert1(V);

}

ordered = MeasurementProcessing(Res);

random = MeasurementProcessing(Res1);

backordered = MeasurementProcessing(Res2);

Out\_Res(ordered, random, backordered);

}

void Select4ExchangeVector\_Maesurement(int \*V)//виміру часу при алгоритмі сортування №1 методу прямої вставки для вектора

{

float ordered, random, backordered;

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

{

FillVec(V, 1);

Res[i] = Select4Exchange(V);

FillVec(V, 1);

Res1[i] = Select4Exchange(V);

FillVec(V, 1);

Res2[i] = Select4Exchange(V);

}

ordered = MeasurementProcessing(Res);

random = MeasurementProcessing(Res1);

backordered = MeasurementProcessing(Res2);

Out\_Res(ordered, random, backordered);

}

void OutTable()//функція для виведення "шапки" таблиці

{

printf("\n\t\t TABLEs WITH MEASUREMENTS: \n");

printf("\n\t\t------------------------------------------------------\n");

printf("\t\t|\t Ordered |\t Random |\t BackOrdered | \n");

printf("\t\t------------------------------------------------------\n");

}

void Out\_Res(float ordered, float random, float backordered)//функція для виводу значень, отриманих при обробці результатів

{

printf("\n\t\t| %7.2f \t|%7.2f \t| %7.2f\t |\n", (float)ordered, (float)random, (float)backordered);

printf("\t\t------------------------------------------------------\n");

}

**sorting.h**

#pragma once

#ifndef SORTING\_H\_INCLUDED

#define SORTING\_H\_INCLUDED

//прототипи

clock\_t Insert1Arr3D(int \*\*\*A);

clock\_t Select4ExchangeArr3D(int \*\*\*A);

clock\_t InsertExchangeArr3D(int \*\*\*A);

clock\_t Insert1(int \*V);

clock\_t Select4Exchange(int \*V);

clock\_t InsertExchange(int \*V);

#endif // SORTING\_H\_INCLUDED

**sorting.cpp**

#include <stdlib.h>

#include <time.h>

#include "CommonVector.h"//розміри масиву, вектора

clock\_t Select4ExchangeArr3D(int \*\*\*A)

{

int k, j\_1, L, R, min, max, counter\_1, counter\_2;//оголошення змінних

clock\_t time\_start, time\_stop;

time\_start = clock();//початок виміру часу

for (k = 0; k < P; k++)

{

j\_1 = 0; //ініціалізація змінних, що повинні мати саме такі значення при сортуванні нового перерізу

L = 0;

R = M - 1;

counter\_1 = 0;

counter\_2 = N - 1;

while (L < R || counter\_1 < counter\_2)//сортування відбувається доки відсортовані частини не перетнуться

{

j\_1 = counter\_1;

if (j\_1 > counter\_2) break;

for (int i = L; i < R + 1 || j\_1 < counter\_2; i++)//цикл, що проводить саме сортування

{

if (i == M)

{

i = 0;

j\_1++;

}

if (A[k][i][j\_1] < A[k][L][counter\_1])

{

min = A[k][i][j\_1];

A[k][i][j\_1] = A[k][L][counter\_1];

A[k][L][counter\_1] = min;

}

else

if (A[k][i][j\_1] > A[k][R][counter\_2])

{

max = A[k][i][j\_1];

A[k][i][j\_1] = A[k][R][counter\_2];

A[k][R][counter\_2] = max;

}

}

L = L + 1;

R = R - 1;

if (L == M)//якщо ліва границя виходить за межі масива, присвоюємо їй значення 0 і переходимо на новий стовпець

{

L = 0;

counter\_1++;

}

if (R == -1)//якщо права границя виходить за межі масива, присвоюємо їй значення останнього елемента стопця і переходимо на новий стовпець

{

R = M - 1;

counter\_2--;

}

}

}

time\_stop = clock();//кінець виміру

return time\_stop - time\_start;}

clock\_t Select4Exchange(int \*V)

{

int Min, Max;

int L, R;

clock\_t time\_start, time\_stop;

time\_start = clock();

L = 0; R = M \*N - 1;

while (L < R) {

for (int i = L; i < R + 1; i++) {

if (V[i] < V[L]) {

Min = V[i];

V[i] = V[L];

V[L] = Min;

}

else

if (V[i] > V[R]) {

Max = V[i];

V[i] = V[R];

V[R] = Max;

}

}

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

}

time\_stop = clock();

return P \* (time\_stop - time\_start);

}

clock\_t Insert1Arr3D(int \*\*\*A)

{

int i, j, k, p, v, d, elem, g;//ініціалізація змінних

clock\_t time\_start, time\_stop;

time\_start = clock();//початок виміру

for (g = 0; g < P; g++)

{

for (j = 0; j <= N - 1; j++)

{

for (i = 0; i <= M - 1; i++)

{

if (j == 0 && i == 0)//пропускаємо нульовий елемент за умовою сортування

{

continue;

}

elem = A[g][i][j];

k = 0;

p = 0;

while (elem > A[g][k][p])// нарощуємо лічильник, який вкаже потрібну позицію для елемента

{

k++;

if (k == M)//якщо лічильник більший довжини стовпця, збільшуємо значення лічильника стовпців, а лічильнику рядків присвоюємо 0

{

p = p + 1;

k = 0;

}

}

v = i - 1;

d = j;

for (v = i - 1; v >= k || d != p; v--)

{

if (v != -1)//якщо елмент у 0 рядку, потрібно поміняти його місцями з останнім елементом попереднього стовпця

{

A[g][v + 1][d] = A[g][v][d];

}

if (v == -1)

{

A[g][0][d] = A[g][M - 1][d - 1];

v = M - 1;

d--;

}

}

A[g][k][p] = elem;

}

}

}

time\_stop = clock();//кінець виміру

return time\_stop - time\_start;

}

clock\_t Insert1(int \*V)

{

int Elem, j;

clock\_t time\_start, time\_stop;

time\_start = clock();

for (int i = 1; i < M \* N; i++) {

Elem = V[i];

j = 0;

while (Elem > V[j]) j = j + 1;

for (int k = i - 1; k >= j; k--)

V[k + 1] = V[k];

V[j] = Elem;

}

time\_stop = clock();

return P \* (time\_stop - time\_start);

}//повертаємо час пропорційно перерізам у масиві

clock\_t InsertExchangeArr3D(int \*\*\*A)

{

int p, g, tmp, i, j, k;//шніціалізація змінних

clock\_t time\_start, time\_stop;

time\_start = clock();//початок виміру

for (k = 0; k < P; k++)

{

for (j = 0; j < N; j++)

{

for (i = 0; i < M; i++)

{

if (i == 0 && j == 0)//попускаємо нульовий елмент за умовою задачі

{

continue;

}

p = i;

g = j;

while (!(p == 0 && g == 0))//виконуємо сортування доки не дійдемо до 0 елмента масива

{

if (p == 0) {

if (A[k][p][g] < A[k][M - 1][g - 1]) {

tmp = A[k][p][g];

A[k][p][g] = A[k][M - 1][g - 1];

A[k][M - 1][g - 1] = tmp;

}

g--;//зменшуємо значення лічильників, щоб цикл був скінченним

p = M - 1;//при переході на новий стовпец втановлюємо лічильник на останній елемент стовпця

}

else {

if (A[k][p][g] < A[k][p - 1][g]) {

tmp = A[k][p][g];

A[k][p][g] = A[k][p - 1][g];

A[k][p - 1][g] = tmp;

}

p--;//зменшуємо значення лічильників, щоб цикл був скінченним

}

}

}

}

}

time\_stop = clock();//кінець виміру

return time\_stop - time\_start;

}

clock\_t InsertExchange(int \*V)

{

int j, tmp;

clock\_t time\_start, time\_stop;

time\_start = clock();

for (int i = 1; i < M \* N; i++) {

j = i;

while (j > 0 && V[j] < V[j - 1]) {

tmp = V[j];

V[j] = V[j - 1];

V[j - 1] = tmp;

j = j - 1;

}

}

time\_stop = clock();

return P \* (time\_stop - time\_start);

}

**in\_out.h**

#pragma once

#ifndef INO\_OUT\_H\_INCLUDED

#define INO\_OUT\_H\_INCLUDED

#include <stdio.h>//бібліотека вводу-виводу

#include <stdlib.h>//бібліотека для виділення пам'яті

//прототипи

void FillArray(int \*\*\*A, int b);

void showArray(int \*\*\*A);

void FillVec(int \*Vec, int mode);

void showVector(int \*Vec);

int\*\*\* Multi\_Dimentional\_Dynamic\_Array();

int\* Multi\_Dimentional\_Dynamic\_Vector();

void FreeMemory(int\*\*\*A, int\* Vec);

#endif // INO\_OUT\_H\_INCLUDED

**in\_out.cpp**

#include <iostream>

#include "CommonVector.h"

void FillArray(int \*\*\*A, int b)//заповнення масива

{

switch (b)

{

case 1:

{

long double number = 0;

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

{

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

{

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

{

A[k][i][j] = number++ ;//заповнення масива по зростанню елементів у стовпцях

}

}

}

break;

}

case 2:

{

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

{

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

{

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

{

A[k][i][j] = rand() % (P \* M \* N);// рандомно заповнений масив

}

}

}

break;

}

case 3:

{

int number = P \* M \* N;

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

{

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

{

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

{

A[k][i][j] = number--;//заповнення масива за спаданням елментів у стовпцях

}

}

}

}

default:

break;

}

}

void showArray(int \*\*\*A)

/\*

Функція, яка призначення для виводу масиву на екран

у вигляді матриці перерізів масиву

\*/

{

for (int k = 0; k < P; k++) {

printf("Number %d\n", k);

for (int i = 0; i < M; i++) {

for (int j = 0; j < N; j++) {

printf("|%4d", A[k][i][j]);

}

printf("|\n");

}

printf("\n");

}

}

void FillVec(int \*Vec, int mode)

{ // функція для заповнення вектора

int i;

long double number;

switch (mode)

{

case 1: // прямо-відсортований масив А

number = 0;

for (i = 0; i < M\*N; i++)

Vec[i] = number++;

break;

case 2: // заповнення масиву А рандомними числами

for (i = 0; i < M\*N; i++)

Vec[i] = rand() % (N \* M);

break;

case 3: // обернено-відсортований масив А

number = N \* M;

for (i = 0; i < M\*N; i++)

Vec[i] = number--;

break;

default: break;

}

}

void showVector(int \*Vec)

/\*

Функція, яка призначення для виводу вектора на екран

\*/

{

for (int j = 0; j < M\*N; j++)

printf("|%4d", Vec[j]);

}

int\*\*\* Multi\_Dimentional\_Dynamic\_Array()//динамічне виділення пам'яті для тривірмного масива

{

int\*\*\* Arr3D = (int\*\*\*)malloc(P \* sizeof(int\*\*));

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

{

Arr3D[k] = (int\*\*)malloc(M \* sizeof(int\*));

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

Arr3D[k][i] = (int\*)malloc(N \* sizeof(int));

}

return Arr3D;

}

int\* Multi\_Dimentional\_Dynamic\_Vector()//динамічне виділення пам'яті для вектора

{

int\* Vec = (int\*)malloc(N \* M \* sizeof(int\*));

return Vec;

}

void FreeMemory(int\*\*\* A, int\* V)//звільнення пам'яті після роботи з динамічним масивом

{

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

{

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

free(A[k][i]);

free(A[k]);

}

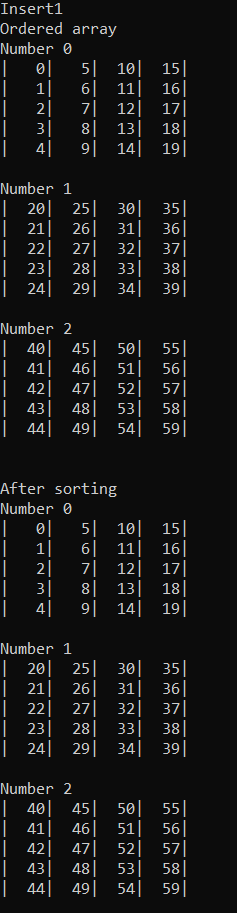
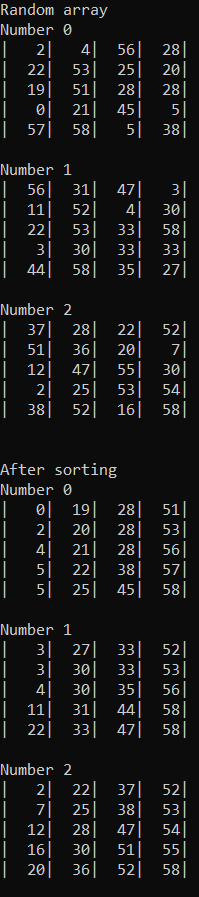
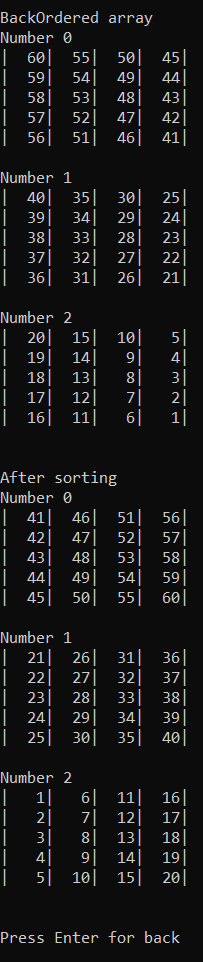
free(A);

free(V);

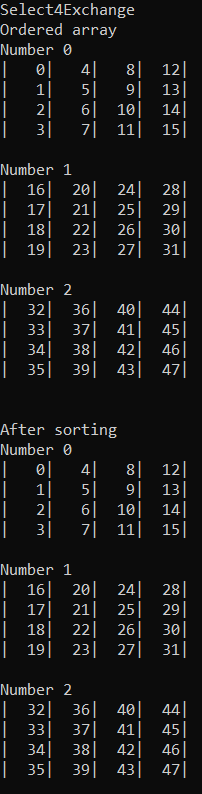
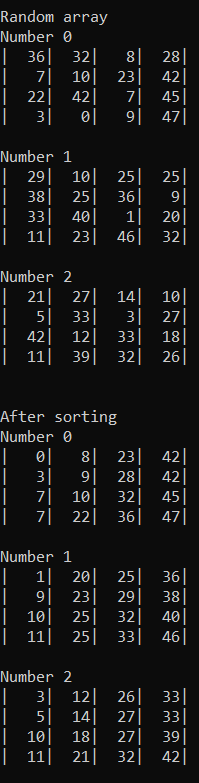
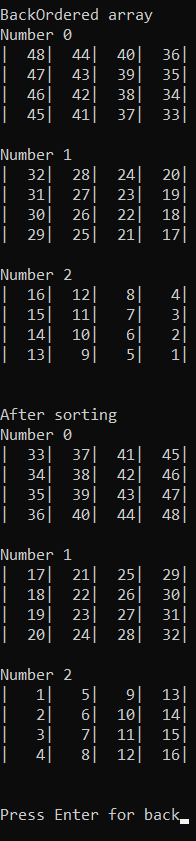
}

**Тести програми**

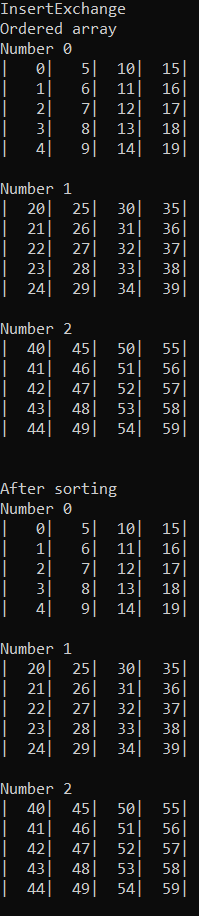
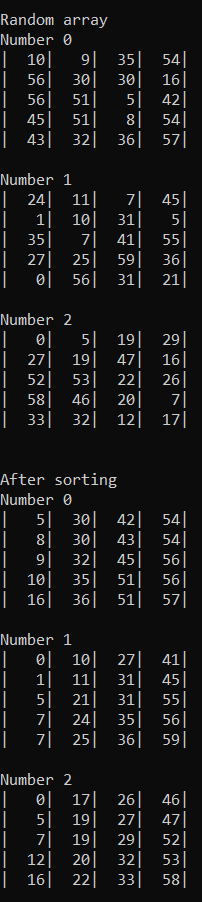
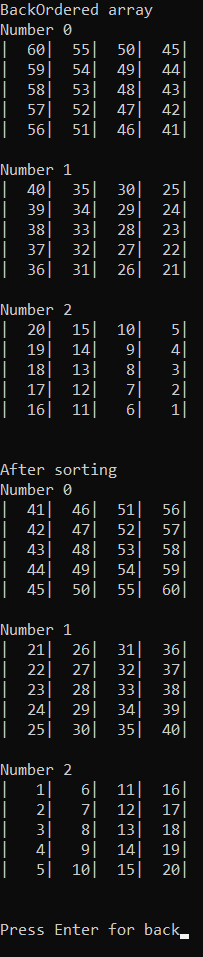
1. **Тести сортування для масиву:**
   1. **Для тривимірного масиву**
2. Алгоритм сортування вставкою №1

Б) Гібридний алгоритм «вибір №4 – обмін»

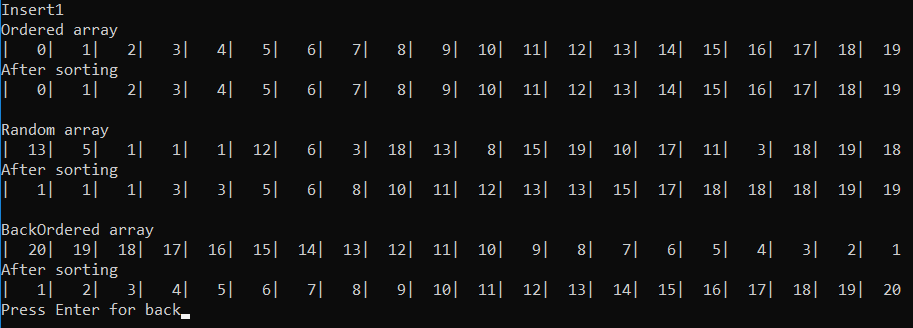
  

1. Гібридний алгоритм «вставка – обмін»

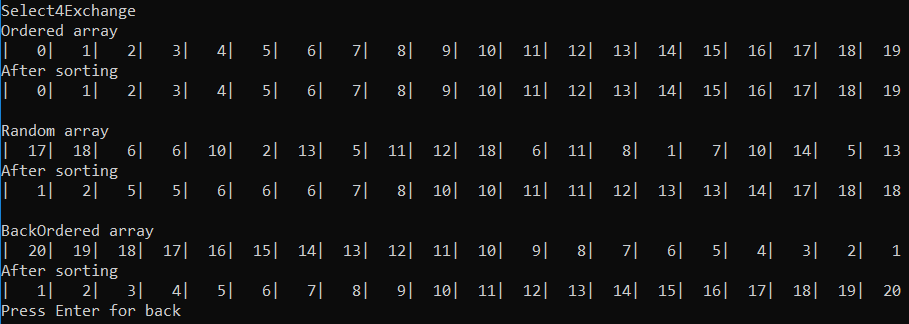
  

* 1. **Для одновимірного масиву**

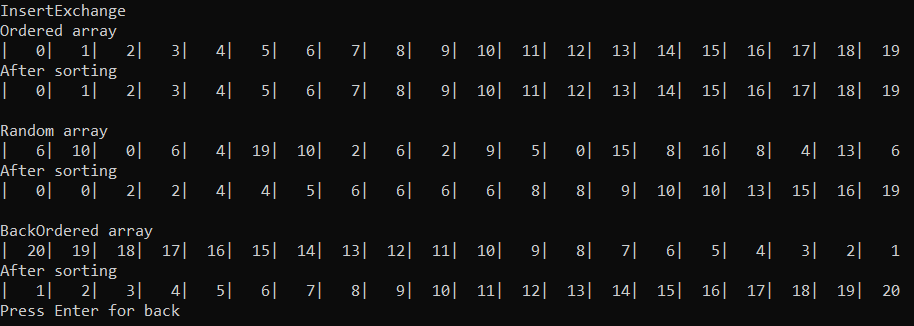
А) Алгоритм сортування вставкою №1

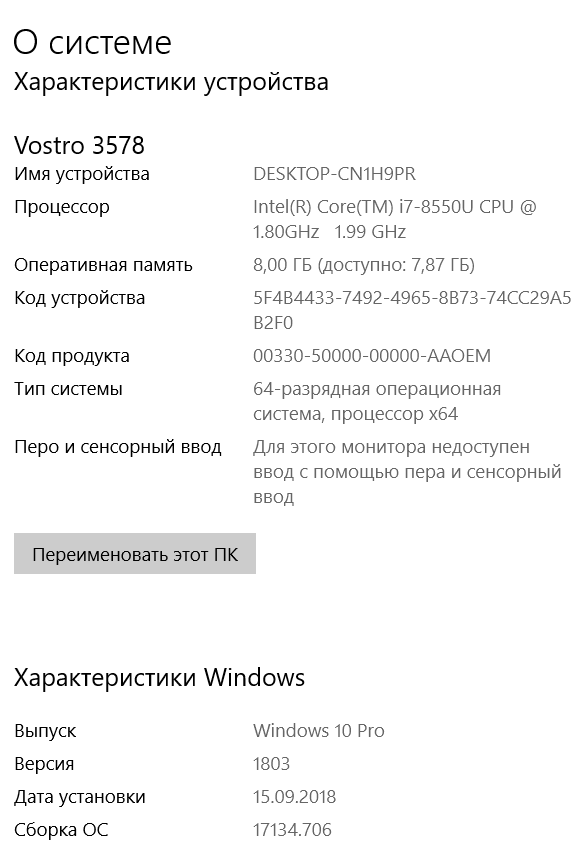


Б) Гібридний алгоритм «вибір №4 – обмін»



В) Гібридний алгоритм «вставка – обмін»



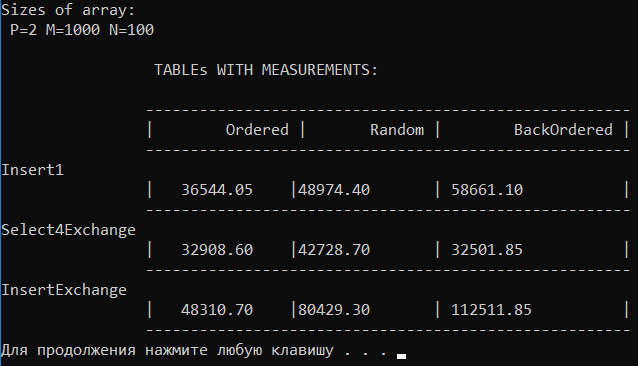
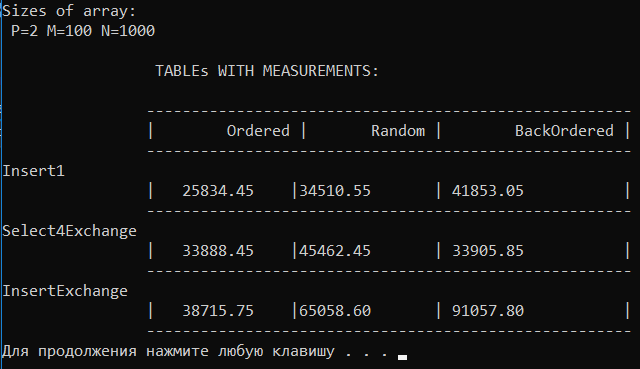
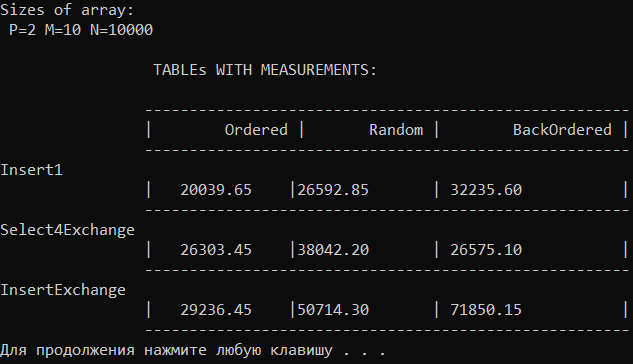
****

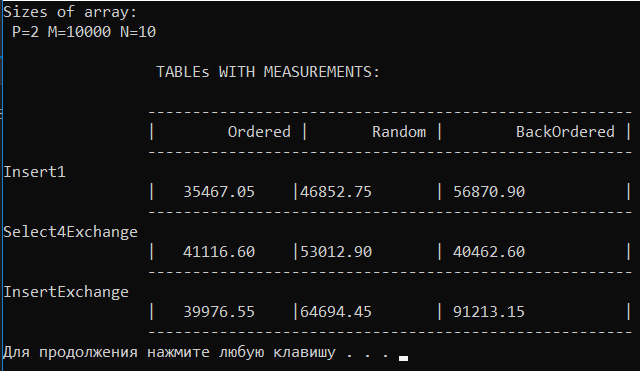
Компілятор: Visual Studio 2017

**Таблиці часу сортування**

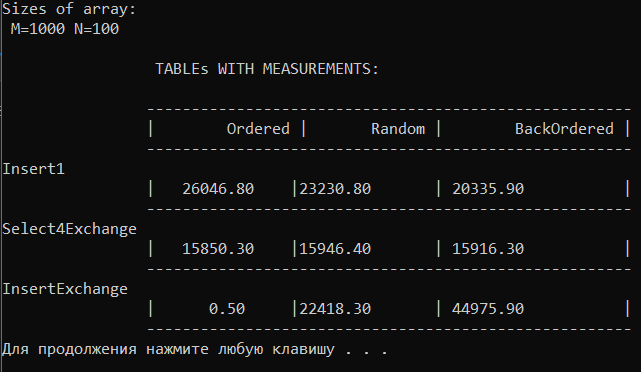
***Випадок дослідження І. Залежність часу роботи алгоритмів від***

***форми перерізу масива.***



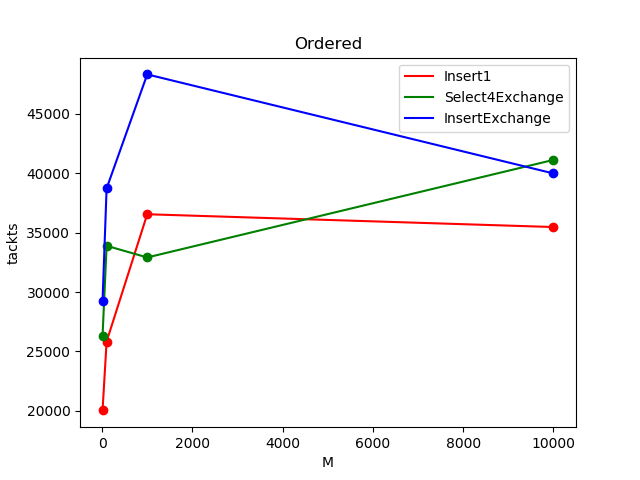


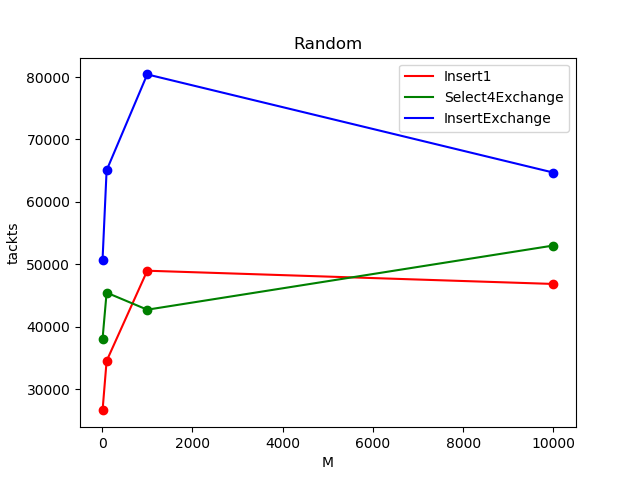
Для вектора довжиною M \* N(Р = 2, на Р домножені всі результати):

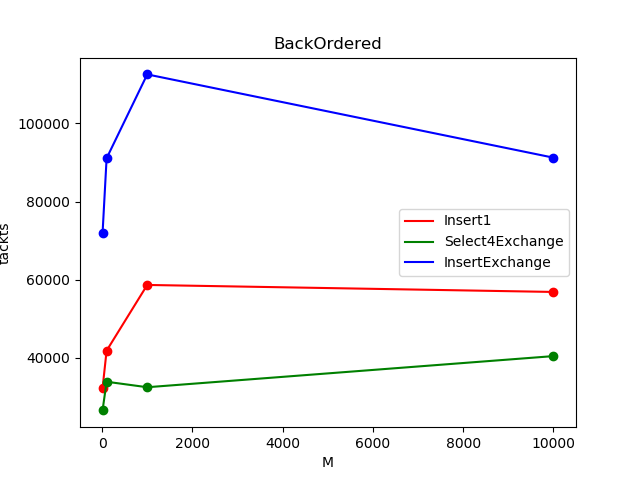


Можна зробити висновок, що час сортування залежить від форми перерізу.

Детальний аналіз у розділі Аналіз алгоритмів сортування.

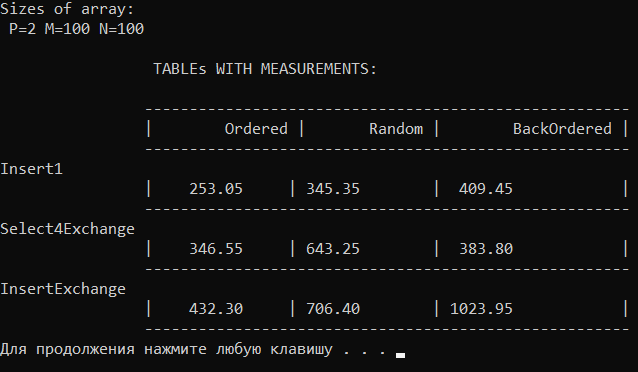


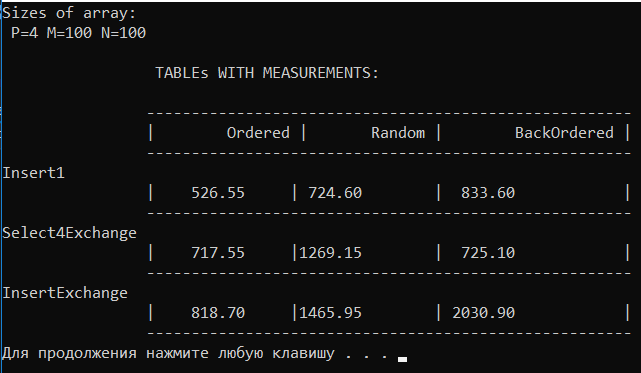


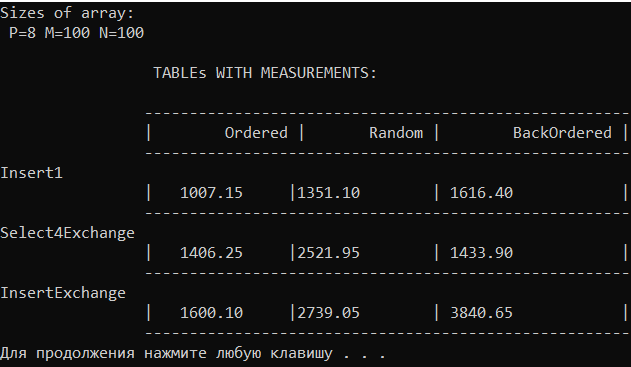


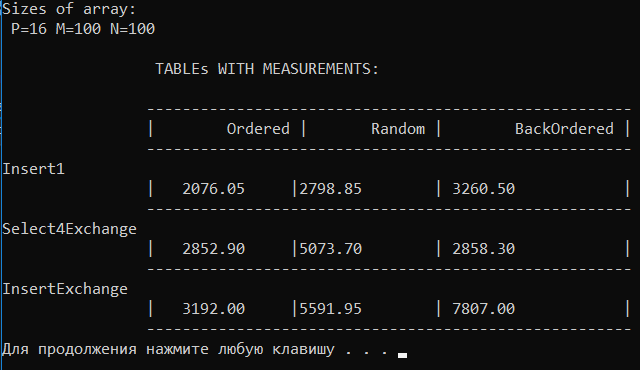
***Випадок дослідження ІІ. Залежність часу роботи алгоритмів***

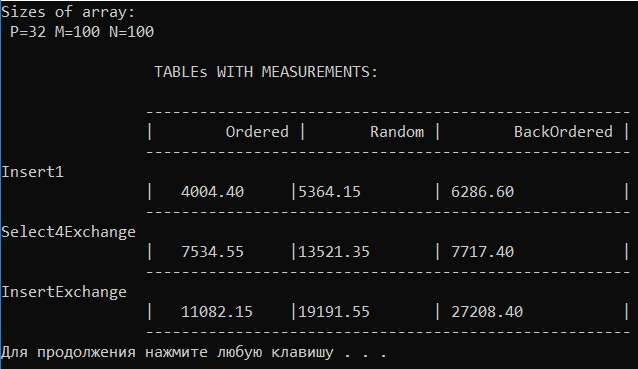
***від кількості перерізів масива***



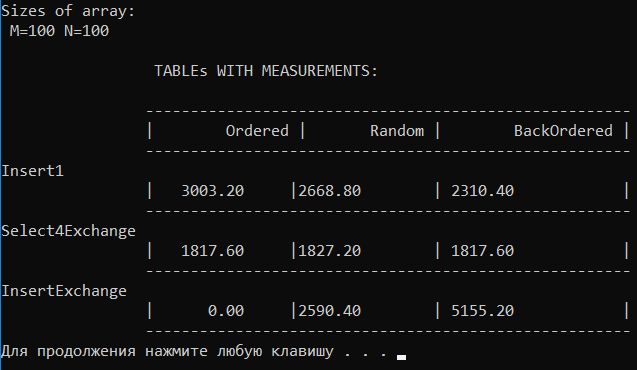






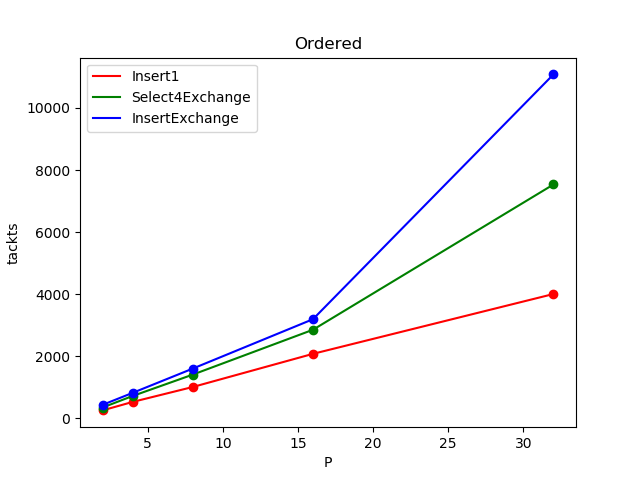


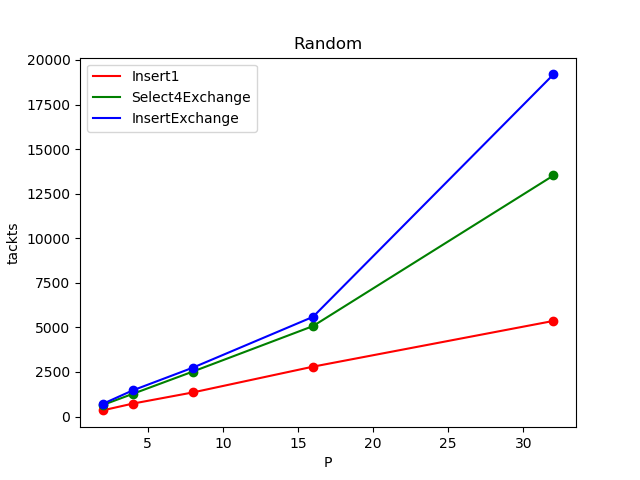
Для вектора довжиною M \* N(Р = 32, на Р домножені всі результати):

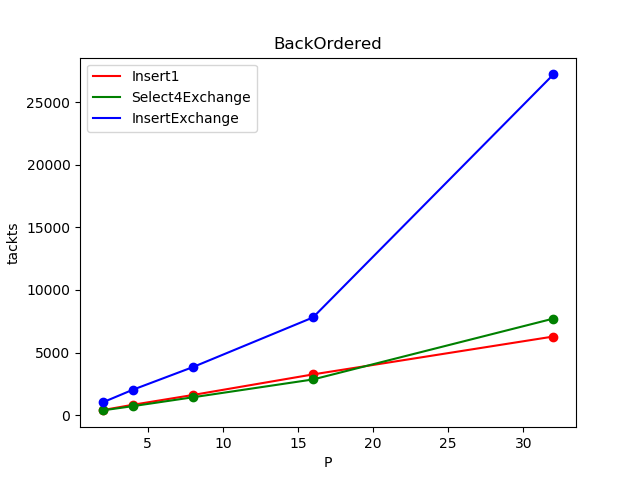


Прослідковується лінійна залежність часу роботи від кількості перерізів, час збільшується на стільки, на скільки ми збільшили кількість перерізів.

Детальний аналіз у розділі Аналіз алгоритмів сортування.

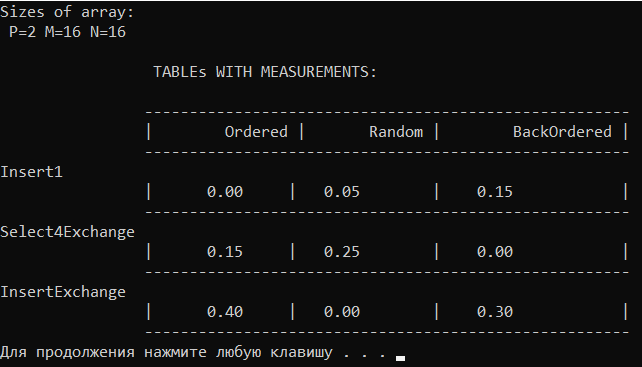


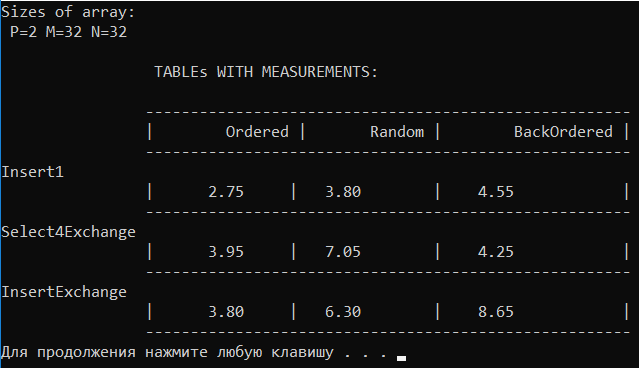


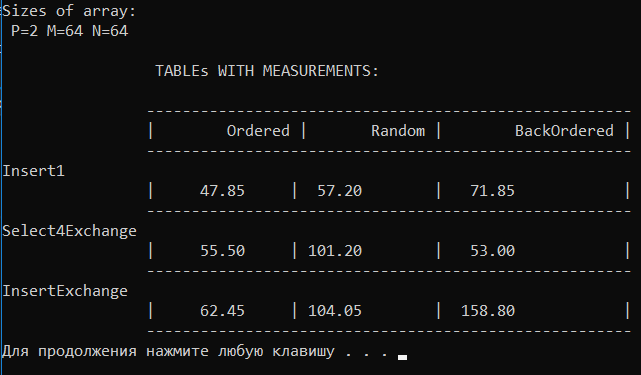


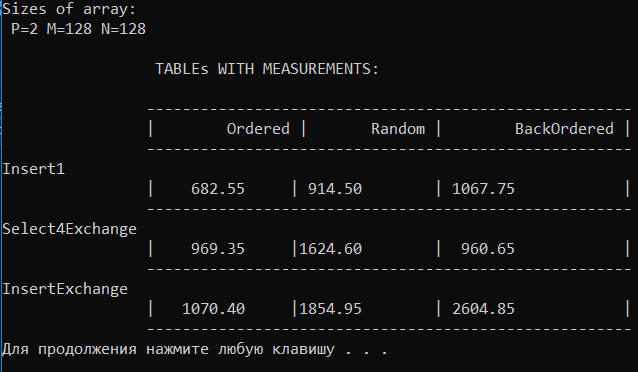
***Випадок дослідження ІІІ. Залежність часу роботи алгоритмів***

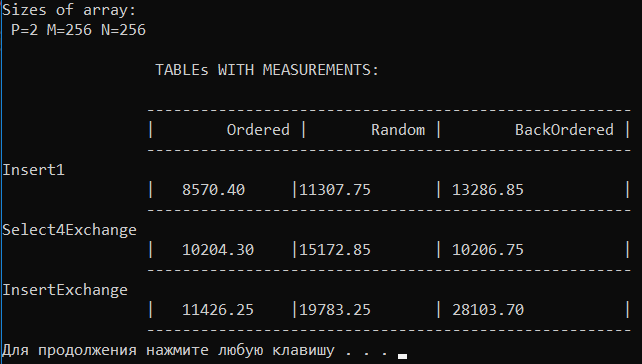
***від розміру перерізів масива***







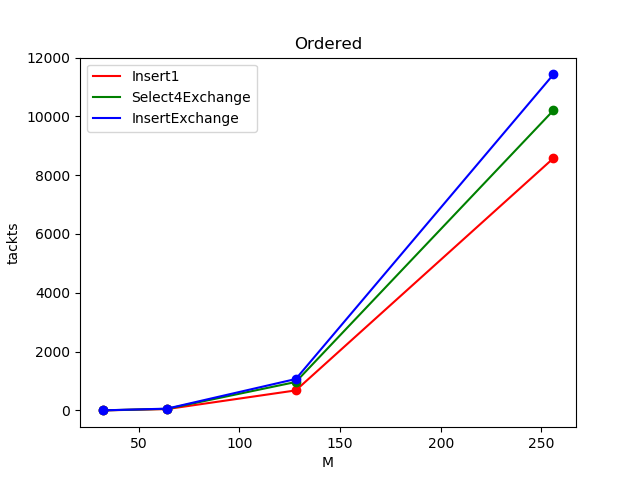


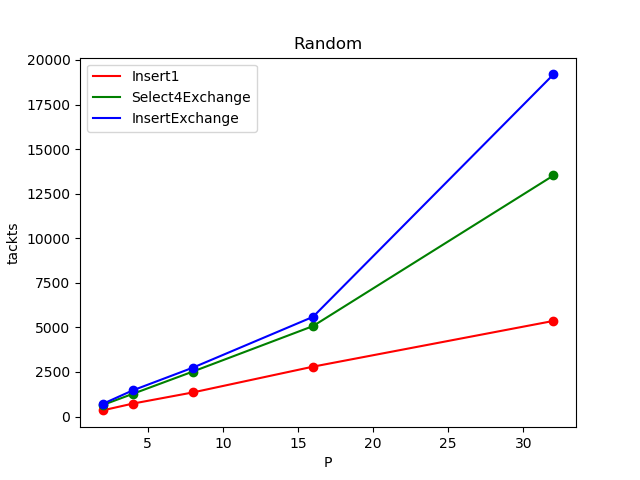


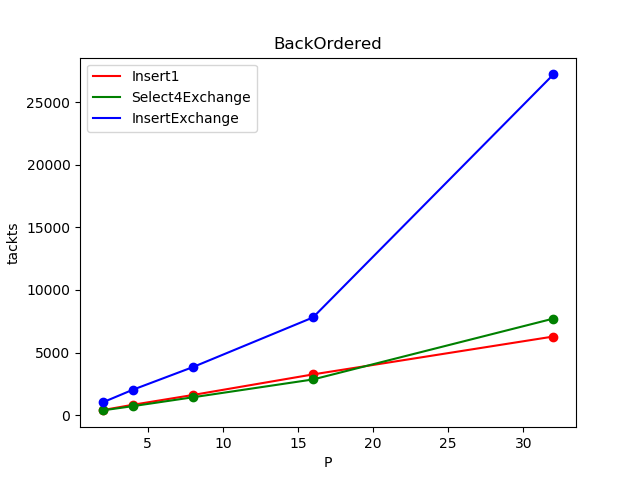
З отриманих вимірів стає очевидним, що швидкість сортування збільшується

зі збільшенням розмірів масиву. (Виміри для масивів малих розмірів можна не розглядати, тому що похибка вимірів там дуже велика)

Детальний аналіз у розділі Аналіз алгоритмів сортування.







**Порівняльний аналіз отриманих результатів**

Перед початку аналізу хочу зазначити, що на одній з лекцій було дозволено не аналізувати поведінку алгоритму «вибір №4 – обмін» для випадку оберненої відсортованості масиву, тому цього опису в мене не буде.

**Залежність часу роботи алгоритмів від довжини форми перерізів масиву**

За даними таблиць вимірів часу і графіків наведених мною вище, прослідковується сильна залежність часу сортування від форми перерізів. Для сортування методом «вибір №4 – обмін» цю залежність можна сформулювати так, чим більше форма перерізу подібна до квадрата, тим швидше відбувається сортування.

Для сортування «вставкою №1» найкращою формою для сортування буде форма, наближена до вектора, що наглядно зображено у таблицях вимірів та графіках. Очевидно це пов’язано з кешуванням при роботі алгоритму.

Для сортування «вставка – обмін» найкращою формою буде також форма наближена до вектора, цей метод буде проводити найдовше сортування для перерізів квадратної форми. Очевидно це пов’язано з кешуванням при роботі алгоритму.

Найкращий сортування для цього випадку дослідження виявився алгоритм «вставка №1», саме він буде оптимальним варіантом для сортування різних форм перерізів. Алгоритм виявився найкращим саме тому, що в ньому немає елементів сортування “бульбашкою”, ми знаходимо позицію вставки в циклі, не виконуючи обмін елементів місцями при кожному проході, а просто робимо зсув на один елемент, що значно скорочує час роботи.

На другому місці по швидкодії для даного випадку дослідження алгоритм «вибір№4 – обмін». Найкращим випадком очевидно виявився уже відсортований масив, а найгіршим рандомно заповнений. Алгоритм програє у швидкодії вставці тому, що навіть коли мінімальний і максимальний елементи масиву були знайдені і поставлені на портібні позиції, алгоритм змушений пройти по всій невідсортованій частині, адже ми не можемо бути впевнені, що знайшли мінімум та максимум, доки не перевіримо всі елементи, за рахунок цього витрачається більше часу на сортування.

Найгіршим для цього дослідження виявився алгоритм сортування «вставка – обмін». Найкращим випадком, як не дивно, для цього алгоритму виявився уже відсортований масив, найгіршим – обернено-відсортований масив, тому що в даному алгоритмі ми шукаємо позицію елемента, щоразу міняючи його з попереднім, якщо порядок елементів не буде порушувати умову відсортованості масиву.

Сортування вектора для даних розмірів дає нам дані, що для вектора сортування відбувається швидше, ніж для матриці, сортування «вибрі№4 – обмін» показує досить хороші результати для всіх випадків заповнення масиву, тому що за один прохід по масиву ставимо одразу два елементи на свої позиції, але найкращим для вже відсортованого масиву виявися метод «вставка – обмін» тому, що ми не виконуємо фактично ніяких дій, завдяки тому, що умова відсортованості не є порушеною і ми просто не заходимо в цикл.

Висвітлений мною порядок алгоритмів по швидкодії однаковий для всіх випадків досліджень, тому надалі причини програшу в швидкості не будуть детально висвітлюватися.

Щодо сортування вставкою №1 та сортування вставкою №2, то принцих їх роботи абсолютно різний, при сортуванні вставкою ми одразу міняємо місцями елементи, якщо попередній виявився меншим, доки не знайдемо потрібну позицію для елемента, яка буде задовольняти умову відсортованості. При сортуванні вставкою ми спочатку знаходимо позицію в одному циклі і тільки тоді уже робимо зсув, та вставляємо елемент на потрібну позицію. Середня кількість присвоєнь та порівнянь порядку для обох алгоритмів. Можливо компілятор генерує код, для сортування вставкою №2, вважаючи його швидшим для даного сортування вектора, адже кількість порівнянь і присвоєнь у даних алгоритмів досить подібна, хоч і різні стилі обходу. Також можна зауважити, що середня кількість присвоєнь для алгоритму сортування вставкою №1 рівна максимальній кількості порівнянь для алгоритму сортування вставкою №2. Можливо це також вплинуло на генерацію асемблерного коду компілятором VS2017.

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

З графіків та таблиць вимірів часу для даного випадку дослідження можна помітити, що всі алгоритми мають однакову лінійну залежність від кількості перерізів (в цілому на скільки збільшується кількість перерізів, на стільки і збільшується час сортування). По швидкодії алгоритм сортування «вставка№1» залишається найкращим, на другому місці залишається «вибір№4 – обмін» і найгіршим «вставка – обмін» через описані мною вище причини.

Час отриманий при сортуванні вектора і помножений на кількість перерізів значно відрізняються адже знову ж таки, довжина вектора виходить досить малою, через що похибка стає більш вагомою. Очевидно, що вектор буде відсортовано швидше адже там немає переходів на новий рядок та лишніх перевірок чи не досягла потрібна змінна кінця рядка чи стовпця.

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

З графіків та таблиць вимірів часу для даного випадку дослідження прослідковується сильна залежність (більше, ніж на порядок) часу сортування від розмірів перерізів. При збільшенні кількості елементів залежність вирівнюється до такої, що виміру часу збільшуються майже рівно на порядок, для масивів малих розмірів, помітити саме таку числову залежність важко, тому що є похибка вимірів є досить великою. Така залежність отримана тому, що разом зі збільшенням кількості елементів збільшується не тільки кількість перестановок, а і перевірок чи не виходимо ми за межі масиву та деяких інших в залежності від алгоритму.

**Висновки**

Після дослідження методу прямої вставки з лінійним пошуком місця вставки зліва, гібридного методу «вставка – обмін», гібридного методу «вибір№4-обмін», я отримав такі результати:

1. Вставка №1 з пошуком місця вставки зліва виявився найоптимальнішим із перевірених мною алгоритмів для сортування тривимірних масивів, він найкраще себе показав себе фактично у всіх випадках дослідження.
2. Для впорядкованого вектора краще використовувати гібридний алгоритм «вставка – обмін», а для рандомно заповненого та обернено-впорядкованого найоптимальнішим буде гібридний алгоритм «вибір№4 – обмін».
3. Я остаточно переконався, що час сортування масивів від форми перерізів, та вияснив, який з методів буде найбільш оптимальним для сортування перерізів різної форми.
4. Гібридний алгоритм сортування «вставка – обмін» є найгіршим для сортування багатовимірних масивів, час його роботи значно більший, ніж у методу «вибур№4 – обмін» та методу вставки №1.

Отже, якщо необхідно відсортувати тривимірний масив, то найкраще використовувати сортування методом вставки №1 з пошуком місця вставки зліва. Він найшвидше сортує впорядкований, невпорядкований та обернено-впорядкований масив серед представлених мною трьох методів. А от гібридний алгоритм «вставка – обмін» для цієї задачі підходить найгірше, через надзвичайно великий час сортування.

**Список використаної літератури**

1. Марченко А. И., Марченко Л., А «Программирование в среде Turbo Pascal 7.0» Киев «ВЕК+», Санкт-Петербург «КОРОНА-Век», 2007
2. Конспект з СДА;

3. Вирт Н. «Алгоритмы и структуры данных» / Н.Вирт – СПб. : Невский диалект, 2005.