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: Селетков В.Р.

#### Група: КB - 91

Номер залікової книжки: КВ - 9120

#### Допущений до захисту

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

2 семестр 2019/2020

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Узгоджено ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_\_\_\_2020р.

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

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

***Дослідження ефективності методів сортування (прямого вибору №8, гібридний алгоритм "вставка – обмін", сортування Шелла №1)на багатовимірних масивах***

#### Виконавець роботи:

Селетков Владислав Русланович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 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= ;

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

Для варіантів курсової роботи, де крім алгоритмів порівнюються також способи обходу, в назвах рядків таблиць потрібно вказати як назви алгоритмів, так і номери способів обходу.

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

Зробити виміри часу для стандартного випадку одномірного масива, довжина якого вибирається такою, щоб можна було виконати коректний порівняльний аналіз з рішенням цієї ж задачі для багатовимірного масива.

Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибираються так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів.

Рекомендації випадків дослідження з різними геометричними розмірами масивів наведені у розділі «Випадки дослідження».

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

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

* для одномірного масива відносно загальновідомої теорії;
* для багатовимірних масивів відносно результатів для одномірного масива;
* для заданих алгоритмів на багатовимірних масивах між собою;
* дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою;
* **для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше**.

**VIII.** Зробити висновки за зробленим порівняльним аналізом.

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

**Варіант № 85**

**Задача 4**

Впорядкувати тривимірний масив Array[P, M, N] таким чином: переставити перерізи масиву за незменшенням значень вектору перших елементів кожного перерізу Аrray[\*, 0, 0].

**Досліджувані методи та алгоритми**

* Алгоритм сортування №8 методу прямого вибору;
* Гібридний алгоритм "вставка – обмін";
* Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2) Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються.

**Способи обходу**

Використовуючи значення вектору перших елементів кожного перерізу Array[\*, 0, 0] як ключі сортування, переставляти відповідні перерізи кожен раз, коли треба переставляти ключі. При перестановці перерізів потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови C/C++.

**Випадки дослідження**

* Залежність часу роботи алгоритмів від розміру перерізів масиву;
* Залежність часу роботи алгоритмів від форми перерізів масиву.

**Опис теоретичних положень**

**Алгоритм сортування №8 методу прямого вибору**

**Принцип сортування.**

В кожен момент часу масив відзначається складеним з двох частин: вже відсортований та ще не відсортований, спочатку відсортована частина сприймається довжиною в 0 елементів. За ліву межу L приймається перший елемент масиву, за праву межу R приймається останній елемент масиву.

1) Знаходимо найбільший та найменший елементи масиву, які лежать в межах від L до R, запам’ятовуємо їх позиції.

2) Якщо знайдений мінімальний елемент не знаходиться на позиції L, то міняємо місцями цей елемент з елементом на позиції L. Якщо найбільший елемент не знаходиться на позиції R, то міняємо місцями цей елемент з елементом на позиції R.

3) Після проходу по масиву, до відсортованої частини додається перший та останній елемент не відсортованої частини, а ліва границя збільшується на 1 позицію та права зменшується на 1 позицію.

4) Виконуємо пункти 1-3 доки ліва та права границі не перетинаються(L<R).

**Схема алгоритму.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | L = 0 | 1 | 2 | 3 | R = n-1 = 4 |
| 1) | 7 | 4 | 1 | 9 | 3 |

imin = 2, imax = 3. L = L + 1 = 1, R = R - 1 = 3.

4

2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | L = 1 | 2 | R = 3 | 4 |
| 2) | 1 | 4 | 7 | 3 | 9 |

3

1

imin = 3, imax = 2. L = L + 1 = 2, R = R – 1 = 2. L = R.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | L = R = 2 | 3 | 4 |
| 3) | 1 | 3 | 4 | 7 | 9 |

Масив відсортовано.

**Алгоритм сортування на мові С.**

void Vector\_Selection\_Sort(int \*Vector, int N) {

int L, R, imin, imax, temp;

L = 0;

R = N - 1;

while (L < R) {

imin = L;

imax = L;

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

if (Vector[i] < Vector[imin])

imin = i;

else if (Vector[i] > Vector[imax])

imax = i;

if (imin != L) {

temp = Vector[imin];

Vector[imin] = Vector[L];

Vector[L] = temp;

}

if (imax != R)

if (imax == L) {

temp = Vector[imin];

Vector[imin] = Vector[R];

Vector[R] = temp;

} else {

temp = Vector[imax];

Vector[imax] = Vector[R];

Vector[R] = temp;

}

L = L + 1;

R = R - 1;

}

}

**Гібридний алгоритм "вставка – обмін"**

**Принцип сортування.**

В кожен момент часу масив відзначається складеним з двох частин: відсортований та невідсортований. Перед сортуванням за відсортовану частину приймається перший елемент масиву.

1) Перший елемент невідсортованої частини порівнюється з попереднім елементом. Якщо їхні значення не задовольняють умові сортування, то вони міняються місцями.

2) Індекс зменшується на 1, беремо наступну пару сусідніх елементів і виконуємо таке саме порівняння і так далі до останньої пари елементів.

3) Якщо елементи задовольняють умову, то береться наступний елемент невідсортованої частини і повторюються пункти 1-2. Сортування закінчується, коли невідсортована частина має 0 елементів.

**Схема алгоритму.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 1) | 3 | 9 | 2 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 2) | 3 | 9 | 2 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 3) | 3 | 2 | 9 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 4) | 2 | 3 | 9 | 7 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 5) | 2 | 3 | 7 | 9 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 6) | 2 | 3 | 7 | 0 | 9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 7) | 2 | 3 | 0 | 7 | 9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 8) | 2 | 0 | 3 | 7 | 9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | n - 1 = 4 |
| 9) | 0 | 2 | 3 | 7 | 9 |

Масив відсортовано.

**Алгоритм сортування на мові С.**

void Vector\_Insertion\_Exchange\_Sort(int \*Vector, int N) {

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

j = i;

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

temp = Vector[j];

Vector[j] = Vector[j - 1];

Vector[j - 1] = temp;

j = j - 1;

}

}

}

**Алгоритм №1 методу сортування Шелла**

**Принцип сортування.**

Сортування виконується в декілька етапів, кількість етапів як правило залежить від довжини масиву, розглянемо принцип та схему при трьох станах.

**Етап перший.**

Методом прямої вставки сортуються між собою елементи, що знаходяться один від одного на відстані в 4 позиції. Причому, при сортуванні цих елементів, інші елементи участі не приймають. Таких груп елементів буде 4. Аналогічно сортуються елементи з другої, третьої та четвертої груп елементів.

**Етап другий.**

Виконується аналогічно першому етапу, тільки для елементів на відстані в дві позиції. Таких груп буде дві.

**Етап третій.**

Виконується аналогічно тільки для елементів на відстані в одну позицію, тобто виконується сортування масиву звичайною прямою вставкою.

**Можливі два способи сортування груп:**

1) Виконується повне сортування методом прямої вставки по черзі всіх елементів кожної групи окремо. Тобто спочатку елемент 1 групи, потім другої і так далі;

2) Виконується вставка для першого елементу 1 групи, потім для першого елементу другої групи, 1 елементу третьої групи і четвертої. Потім аналогічно для 2 елементі, 3 елемента, 4 елемента кожної групи.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

1 група

2 група

3 група

4 група

Реалізація 2 способу є більш компактною та елегантною ніж першого способу.

**Схема алгоритму другого способу.**

**Етап перший.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

**Етап другий.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | n-1=9 |
|  |  |  |  |  |  |  |  |  |  |

**Етап третій.**

Виконується сортування звичайним алгоритмом прямої вставки (з кроком 1).

**Алгоритм сортування на мові С.**

void Vector\_Shell\_Sort(int \*Vector, int N) {

int Elem, t, j, k;

int Stages[(N - 1) / 4 + 1];

if (N < 4)

t = 1;

else

t = (int) log2f((float) N) - 1;

Stages[t - 1] = 1;

for (int i = t - 2; i >= 0; i--)

Stages[i] = 2 \* Stages[i + 1] + 1;

for (int q = 0; q < t; q++) {

k = Stages[q];

for (int i = k; i < N; i++) {

Elem = Vector[i];

j = i;

while (j >= k && Elem < Vector[j - k]) {

Vector[j] = Vector[j - k];

j = j - k;

}

Vector[j] = Elem;

}

}

}

**Схема імпорту/експорту модулів**

main

CommonArray

Measurement

InputOutputArray

MeasureArray

SortingArray

InputOutputVector

MeasureVector

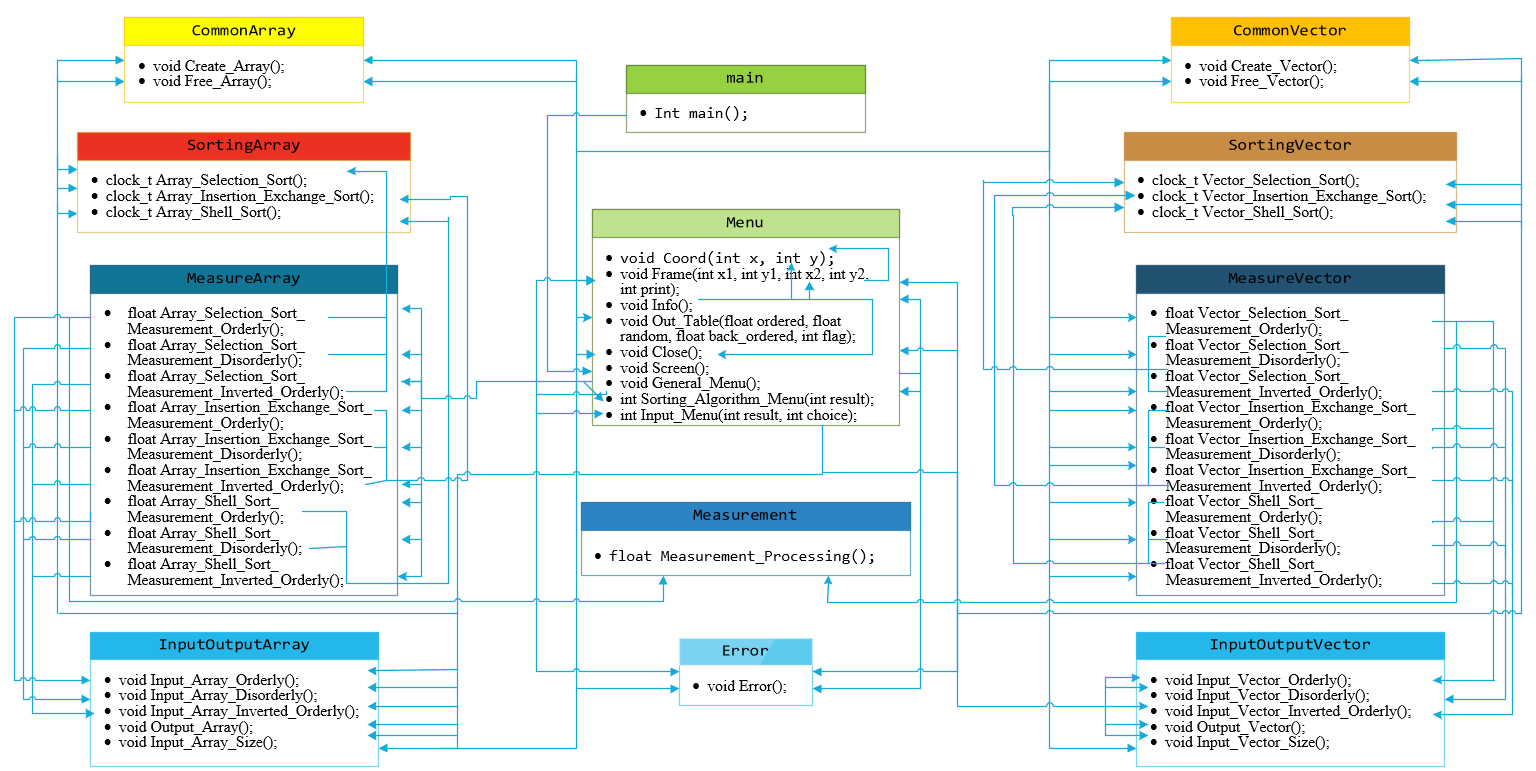
SortingVector

CommonVector

Error

menu

**Структурна схема взаємовикликів процедур та функцій**

****

**Опис призначення всіх функцій і процедур та їх параметрів.**

1. **main** – точка входу до програми, з якої відбувається перехід до меню.

2. **Menu** – створює діалог з користувачем та реалізовує виконання програми.

2.1. **Coord** – приймає два параметри: *int x та int y*, за допомогою яких реалізовується переміщення курсору по консолі.

2.2. **Frame** – створює на консоль рамку. Приймає 5 параметрів: *int x1, int y1, int x2, int y2, int print*. За допомогою координат здійснюється вивід рамки на консоль, а змінна *print* регулює швидкість виводу. Якщо *print* = 1, то можна побачити як друкується рамка, *print* = 0, виводить рамку моментально.

2.3. **Info** – створює заставку на консоль.

2.4. **Out\_Table** – створює табличку з виміром часу алгоритмів сортування в пакетному режимі. Приймає 4 параметри: *float ordered* – виміряний час сортування впорядкованого багатовимірного масиву або вектора*, float random* – не впорядкованого*, float back\_ordered*  - обернено-впорядкованого*, int flag* – перевіряється умовою, щоб заповнювати послідовно рядки таблиці різних алгоритмів*.*

2.5. **Close** – повністю очищає консоль.

2.6. **Screen** – виводить заставку на консоль та реалізовує перехід до основного меню.

2.7. **General\_Menu** – виводить основне меню на консоль, з якого можна перейти до сортування багатовимірного масиву та вектора в пакетному режимі, після чого виведеться табличка з результатами. Або вибрати тестове сортування багатовимірного масиву чи вектора, після чого викликається функція меню для вибору алгоритму.

2.8. **Sorting\_Algorithm\_Menu** – виводить на екран меню для вибору алгоритму сортування в тестовому режимі, приймає значення *result* з попередньої функції, з якого потім визначається масив це чи вектор.

2.9. **Input\_Menu** – виводить на консоль меню алгоритми для вибору заповнення в тестовому режимі, після вибору заповнення виконує сортування та виводить результат на екран. Приймає значення *result та choice\_alg* з попередньої функції, *result* – багатовимірний масив або вектор, *choice\_alg* – вибраний алгоритм сортування.

3. **CommonArray** – модуль, що створює розміри багатовимірного масиву, виділяє пам'ять для багатовимірного масиву та звільняє її.

3.1. **Create\_Array** – виділяє пам'ять від тривимірного масиву.

3.2. **Free\_Array** – звільняє пам'ять від тривимірного масиву.

4. **CommonVector** – модуль, що створює розміри вектора, виділяє пам'ять для вектору та звільняє її.

3.1. **Create\_Vector** – виділяє пам'ять від вектора.

3.2. **Free\_Vector** – звільняє пам'ять від вектора.

5. **Error** – виводить на екран повідомлення про помилку.

6. **InputOutputArray** – виконує заповнення багатовимірного масиву та його розмірів, а також виводить масив на екран.

6.1. **Input\_Array\_Orderly –** заповнює масив впорядковано.

6.2. **Input\_Array\_Disorderly –** заповнює масив не впорядковано.

6.3. **Input\_Array\_Inverted\_Orderly** – заповнює масив обернено-впорядковано.

6.4. **Output\_Array** – виводить масив на екран.

6.5. **Input\_Array\_Sizes** – заповнює, введені користувачем, значення для розміру масиву.

7. **InputOutputVector** – виконує заповнення вектора та його розмірів, а також виводить вектор на екран.

7.1. **Input\_Vector\_Orderly –** заповнює вектор впорядковано.

7.2. **Input\_Vector\_Disorderly –** заповнює вектор не впорядковано.

7.3. **Input\_Vector\_Inverted\_Orderly** – заповнює обернено-впорядковано вектор.

7.4. **Output\_Vector** – виводить вектор на екран.

7.5. **Input\_Vector\_Sizes** – заповнює, введені користувачем, значення для розміру вектора.

8. **MeasureArray** – модуль виміру часу алгоритмів сортування багатовимірного масиву.

8.1. **Array\_Selection\_Sort\_Measurement\_Orderly** - функція, яка заповнює багатовимірний масив впорядковано, виконує сортування вибором та повертає час сортування.

8.2. **Array\_Selection\_Sort\_Measurement\_Disorderly** - функція, яка заповнює багатовимірний масив не впорядковано, виконує сортування вибором та повертає час сортування.

8.3. **Array\_Selection\_Sort\_Measurement\_Inverted\_Orderly -** функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування вибором та повертає час сортування.

8.4. **Array\_Insertion\_Exchange\_Sort\_Measurement\_Orderly** - функція, яка заповнює багатовимірний масив впорядковано, виконує сортування вставкою-обміном та повертає час сортування.

8.5. **Array\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly** - функція, яка заповнює багатовимірний масив не впорядковано, виконує сортування вставкою-обміном та повертає час сортування.

8.6. **Array\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly -** функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування вставкою-обміном та повертає час сортування.

8.7. **Array\_Shell\_Sort\_Measurement\_Orderly** - функція, яка заповнює багатовимірний масив впорядковано, виконує сортування Шелла та повертає час сортування.

8.8. **Array\_Shell\_Sort\_Measurement\_Disorderly** - функція, яка заповнює багатовимірний масив не впорядковано, виконує сортування Шелла та повертає час сортування.

8.9. **Array\_Shell\_Sort\_Measurement\_Inverted\_Orderly** - функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування Шелла та повертає час сортування.

9. **MeasureVector** – модуль виміру часу алгоритмів сортування вектора.

9.1. **Vector\_Selection\_Sort\_Measurement\_Orderly** - функція, яка заповнює вектор впорядковано, виконує сортування вибором та повертає час сортування.

9.2. **Vector\_Selection\_Sort\_Measurement\_Disorderly** - функція, яка заповнює вектор не впорядковано, виконує сортування вибором та повертає час сортування.

9.3. **Vector\_Selection\_Sort\_Measurement\_Inverted\_Orderly -** функція, яка заповнює вектор обернено-впорядковано, виконує сортування вибором та повертає час сортування.

9.4. **Vector\_Insertion\_Exchange\_Sort\_Measurement\_Orderly** - функція, яка заповнює вектор впорядковано, виконує сортування вставкою-обміном та повертає час сортування.

9.5. **Vector\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly** - функція, яка заповнює вектор не впорядковано, виконує сортування вставкою-обміном та повертає час сортування.

9.6. **Vector\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly -** функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування вставкою-обміном та повертає час сортування.

9.7. **Vector\_Shell\_Sort\_Measurement\_Orderly** - функція, яка заповнює вектор впорядковано, виконує сортування Шелла та повертає час сортування.

9.8. **Vector\_Shell\_Sort\_Measurement\_Disorderly** - функція, яка заповнює вектор не впорядковано, виконує сортування Шелла та повертає час сортування.

9.9. **Vector\_Shell\_Sort\_Measurement\_Inverted\_Orderly** - функція, яка заповнює вектор обернено-впорядковано, виконує сортування Шелла та повертає час сортування.

10. **Measurement** – модуль, що створений для виміру часу сортування багатовимірного масиву або вектора.

10.1. **Measurement\_Processing** - функція для обробки результату вимірювання часу сортування алгоритмів, яка взята з методички до курсової роботи.

11. **SortingArray** – модуль алгоритмів сортування багатовимірного масиву.

11.1. **Array\_Selection\_Sort** – сортування багатовимірного масиву алгоритмом вибору.

11.2. **Array\_Insertion\_Exchange\_Sort** – сортування багатовимірного масиву алгоритмом вставки-обміну.

11.3. **Array\_Shell\_Sort –** сортування багатовимірного масиву алгоритмом Шелла.

12. **SortingVector** – модуль алгоритмів сортування багатовимірного масиву.

12.1. **Vector\_Selection\_Sort** – сортування вектора алгоритмом вибору.

12.2. **Vector\_Insertion\_Exchange\_Sort** – сортування вектора алгоритмом вставки-обміну.

12.3. **Vector\_Shell\_Sort –** сортування вектора алгоритмом Шелла.

**Текст програми з коментарями**

**main.c**

#include "Menu.h"

int main() {

Screen(); //вивід заставки на екран, після чого відбувається перехід до меню

return 0;

}

**CommonArray.c**

#include "CommonArray.h"

#include <stdlib.h>

#include <stdio.h>

//створення динаміного багатовимірного масиву

void Create\_Array() {

Array = (int \*\*\*) malloc(p \* sizeof(int \*\*));

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

Array[k] = (int \*\*) malloc(m \* sizeof(int \*));

for (int i = 0; i < m; i++) Array[k][i] = (int \*) malloc(n \* sizeof(int));

}

}

//звільнення динамічного багатовимірного масиву

void Free\_Array() {

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

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

free(Array[k][i]);

free(Array[k]);

}

free(Array);

}

**CommonArray.h**

#ifndef COURSE\_WORK\_COMMON\_H

#define COURSE\_WORK\_COMMON\_H

int p;

int m; //розмір багатовимірного масиву

int n;

int \*\*\*Array;

void Create\_Array();

void Free\_Array();

#endif //COURSE\_WORK\_COMMON\_H

**CommonVector.c**

#include "CommonVector.h"

#include <stdlib.h>

#include <stdio.h>

//створення динамічного вектора

void Create\_Vector() {

Vector = (int \*) malloc(N \* sizeof(int));

}

//звільнення динамічного вектора

void Free\_Vector() {

free(Vector);

}

**CommonVector.h**

#ifndef COURSE\_WORK\_COMMONVECTOR\_H

#define COURSE\_WORK\_COMMONVECTOR\_H

long int N; //розмір вектора

int \*Vector;

void Create\_Vector();

void Free\_Vector();

#endif //COURSE\_WORK\_COMMONVECTOR\_H

**Error.c**

#include "Error.h"

#include <stdio.h>

//виводить повідомлення про помилку

void Error() {

printf("\n\nerror, try one more time ");

}

**Error.h**

#ifndef COURSE\_WORK\_FINAL\_ERROR\_H

#define COURSE\_WORK\_FINAL\_ERROR\_H

void Error();

#endif //COURSE\_WORK\_FINAL\_ERROR\_H

**InputOutputArray.c**

#include "InputOutputArray.h"

#include "CommonArray.h"

#include "Error.h"

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <unistd.h>

//заповення багатовимірного масиву впорядковано

void Input\_Array\_Orderly() {

long number = 0;

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

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

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

Array[k][i][j] = number++;

}

//заповення багатовимірного масиву не впорядковано

void Input\_Array\_Disorderly() {

srand(time(NULL));

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

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

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

Array[k][i][j] = rand() % (p \* m \* n);

}

//заповення багатовимірного масиву обернено-впорядковано

void Input\_Array\_Inverted\_Orderly() {

long number = p \* m \* n - 1;

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

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

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

Array[k][i][j] = number--;

}

//вивід багатовимірного масиву

void Output\_Array() {

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

printf("\n");

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

printf("[");

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

printf("%6d", Array[k][i][j]);

printf("]\n");

}

}

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

}

//заповннення користувачем розмірів багатовимірного масиву

void Input\_Array\_Size() {

printf("Input array sizes:\n\np = ");

scanf("%d", &p);

while (p < 0 || p > 999999999) {

Error();

printf("p = ");

scanf("%d", &p);

}

printf("\nm = ");

scanf("%d", &m);

while (m < 0 || m > 999999999) {

Error();

printf("m = ");

scanf("%d", &m);

}

printf("\nn = ");

scanf("%d", &n);

printf("\n");

while (n < 0 || n > 999999999) {

Error();

printf("n = ");

scanf("%d", &n);

}

if (p == 0 || m == 0 || n == 0) {

printf("\n\nerror, emergency exit from the program");

sleep(3);

exit(1);

}

}

**InputOutputArray.h**

#ifndef COURSE\_WORK\_INPUTOUTPUTARRAY\_H

#define COURSE\_WORK\_INPUTOUTPUTARRAY\_H

void Input\_Array\_Orderly();

void Input\_Array\_Disorderly();

void Input\_Array\_Inverted\_Orderly();

void Output\_Array();

void Input\_Array\_Size();

#endif //COURSE\_WORK\_INPUTOUTPUTARRAY\_H

**InputOutputVector.c**

#include "InputOutputVector.h"

#include "CommonVector.h"

#include "Error.h"

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <unistd.h>

//заповнення вектора впорядковано

void Input\_Vector\_Orderly() {

for (long int i = 0; i < N; i++)

Vector[i] = i;

}

//заповнення вектора не впорядковано

void Input\_Vector\_Disorderly() {

srand(time(NULL));

for (long int i = 0; i < N; i++)

Vector[i] = rand() % N;

}

//заповнення вектора обернено-впорядковано

void Input\_Vector\_Inverted\_Orderly() {

for (long int i = 0; i < N; i++)

Vector[i] = N - i - 1;

}

// вивід вектора

void Output\_Vector() {

for (long int i = 0; i < N; i++)

printf("%6d", Vector[i]);

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

}

//заповнення користувачем розмір вектора

void Input\_Vector\_Size() {

printf("Input vector size:\n\nN = ");

scanf("%ld", &N);

printf("\n");

if (N == 0) {

printf("\n\nerror, emergency exit from the program");

sleep(3);

exit(1);

}

while (N < 0 || N > 999999999) {

Error();

printf("N = ");

scanf("%ld", &N);

}}

**InputOutputVector.h**

#ifndef COURSE\_WORK\_INPUTOUTPUT\_H

#define COURSE\_WORK\_INPUTOUTPUT\_H

void Input\_Vector\_Orderly();

void Input\_Vector\_Disorderly();

void Input\_Vector\_Inverted\_Orderly();

void Output\_Vector();

void Input\_Vector\_Size();

#endif //COURSE\_WORK\_INPUTOUTPUT\_H

**MeasureArray.c**

#include "MeasureArray.h"

#include "Measurement.h"

#include "SortingArray.h"

#include "InputOutputArray.h"

//функція, яка заповнює багатовимірний масив впорядковано, виконує сортування вибором та повертає час сортування

float Array\_Selection\_Sort\_Measurement\_Orderly() {

float sorting\_time;

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

Input\_Array\_Orderly();

Res[i] = Array\_Selection\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив не впорядковано, виконує сортування вибором та повертає час сортування

float Array\_Selection\_Sort\_Measurement\_Disorderly() {

float sorting\_time;

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

Input\_Array\_Disorderly();

Res[i] = Array\_Selection\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування вибором та повертає час сортування

float Array\_Selection\_Sort\_Measurement\_Inverted\_Orderly() {

float sorting\_time;

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

Input\_Array\_Inverted\_Orderly();

Res[i] = Array\_Selection\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив впорядковано, виконує сортування вставкою-обміном та повертає час сортування

float Array\_Insertion\_Exchange\_Sort\_Measurement\_Orderly() {

float sorting\_time;

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

Input\_Array\_Orderly();

Res[i] = Array\_Insertion\_Exchange\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив не впорядковано, виконує сортування вставкою-обміном та повертає час сортування

float Array\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly() {

float sorting\_time;

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

Input\_Array\_Disorderly();

Res[i] = Array\_Insertion\_Exchange\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування вставкою-обміном та повертає час сортування

float Array\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly() {

float sorting\_time;

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

Input\_Array\_Inverted\_Orderly();

Res[i] = Array\_Insertion\_Exchange\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив впорядковано, виконує сортування Шелла та повертає час сортування

float Array\_Shell\_Sort\_Measurement\_Orderly() {

float sorting\_time;

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

Input\_Array\_Orderly();

Res[i] = Array\_Shell\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив не впорядковано, виконує сортування Шелла та повертає час сортування

float Array\_Shell\_Sort\_Measurement\_Disorderly() {

float sorting\_time;

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

Input\_Array\_Disorderly();

Res[i] = Array\_Shell\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює багатовимірний масив обернено-впорядковано, виконує сортування Шелла та повертає час сортування

float Array\_Shell\_Sort\_Measurement\_Inverted\_Orderly() {

float sorting\_time;

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

Input\_Array\_Inverted\_Orderly();

Res[i] = Array\_Shell\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

**MeasureArray.h**

#ifndef COURSE\_WORK\_MEASUREARRAY\_H

#define COURSE\_WORK\_MEASUREARRAY\_H

float Array\_Selection\_Sort\_Measurement\_Orderly();

float Array\_Selection\_Sort\_Measurement\_Disorderly();

float Array\_Selection\_Sort\_Measurement\_Inverted\_Orderly();

float Array\_Insertion\_Exchange\_Sort\_Measurement\_Orderly();

float Array\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly();

float Array\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly();

float Array\_Shell\_Sort\_Measurement\_Orderly();

float Array\_Shell\_Sort\_Measurement\_Disorderly();

float Array\_Shell\_Sort\_Measurement\_Inverted\_Orderly();

#endif //COURSE\_WORK\_MEASUREARRAY\_H

**Measurement.c**

#include "Measurement.h"

#include <stdio.h>

//функція, що знаходить час сортування

float Measurement\_Processing() {

long int sum;

float average\_value;

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];

average\_value = (float) sum / (float) (measurements\_number - 2 \* min\_max\_number - rejected\_number);

printf("");

return average\_value;

}

**Measurement.h**

#ifndef COURSE\_WORK\_MEASUREMENT\_H

#define COURSE\_WORK\_MEASUREMENT\_H

#include <time.h>

#define measurements\_number 28

#define rejected\_number 2

#define min\_max\_number 3

clock\_t Res[measurements\_number];

float Measurement\_Processing();

#endif //COURSE\_WORK\_MEASUREMENT\_H

**MeasureVector.c**

#include "MeasureVector.h"

#include "Measurement.h"

#include "SortingVector.h"

#include "InputOutputVector.h"

//функція, яка заповнює вектор впорядковано, виконує сортування вибором та повертає час сортування

float Vector\_Selection\_Sort\_Measurement\_Orderly() {

float sorting\_time;

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

Input\_Vector\_Orderly();

Res[i] = Vector\_Selection\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор не впорядковано, виконує сортування вибором та повертає час сортування

float Vector\_Selection\_Sort\_Measurement\_Disorderly() {

float sorting\_time;

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

Input\_Vector\_Disorderly();

Res[i] = Vector\_Selection\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор обернено-впорядковано, виконує сортування вибором та повертає час сортування

float Vector\_Selection\_Sort\_Measurement\_Inverted\_Orderly() {

float sorting\_time;

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

Input\_Vector\_Inverted\_Orderly();

Res[i] = Vector\_Selection\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор впорядковано, виконує сортування вставкою-обміном та повертає час сортування

float Vector\_Insertion\_Exchange\_Sort\_Measurement\_Orderly() {

float sorting\_time;

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

Input\_Vector\_Orderly();

Res[i] = Vector\_Insertion\_Exchange\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time; }

//функція, яка заповнює вектор не впорядковано, виконує сортування вставкою-обміном та повертає час сортування

float Vector\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly() {

float sorting\_time;

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

Input\_Vector\_Disorderly();

Res[i] = Vector\_Insertion\_Exchange\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор обернено-впорядковано, виконує сортування вставкою-обміном та повертає час сортування

float Vector\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly() {

float sorting\_time;

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

Input\_Vector\_Inverted\_Orderly();

Res[i] = Vector\_Insertion\_Exchange\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор впорядковано, виконує сортування Шелла та повертає час сортування

float Vector\_Shell\_Sort\_Measurement\_Orderly() {

float sorting\_time;

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

Input\_Vector\_Orderly();

Res[i] = Vector\_Shell\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор не впорядковано, виконує сортування Шелла та повертає час сортування

float Vector\_Shell\_Sort\_Measurement\_Disorderly() {

float sorting\_time;

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

Input\_Vector\_Disorderly();

Res[i] = Vector\_Shell\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

//функція, яка заповнює вектор обернено-впорядковано, виконує сортування Шелла та повертає час сортування

float Vector\_Shell\_Sort\_Measurement\_Inverted\_Orderly() {

float sorting\_time;

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

Input\_Vector\_Inverted\_Orderly();

Res[i] = Vector\_Shell\_Sort();

}

sorting\_time = Measurement\_Processing();

return sorting\_time;

}

**MeasureVector.h**

#ifndef COURSE\_WORK\_MEASUREVECTOR\_H

#define COURSE\_WORK\_MEASUREVECTOR\_H

float Vector\_Selection\_Sort\_Measurement\_Orderly();

float Vector\_Selection\_Sort\_Measurement\_Disorderly();

float Vector\_Selection\_Sort\_Measurement\_Inverted\_Orderly();

float Vector\_Insertion\_Exchange\_Sort\_Measurement\_Orderly();

float Vector\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly();

float Vector\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly();

float Vector\_Shell\_Sort\_Measurement\_Orderly();

float Vector\_Shell\_Sort\_Measurement\_Disorderly();

float Vector\_Shell\_Sort\_Measurement\_Inverted\_Orderly();

#endif //COURSE\_WORK\_MEASUREVECTOR\_H

**Menu.c**

#include "Menu.h"

#include "CommonArray.h"

#include "CommonVector.h"

#include "InputOutputArray.h"

#include "InputOutputVector.h"

#include "MeasureArray.h"

#include "MeasureVector.h"

#include "SortingArray.h"

#include "SortingVector.h"

#include "Error.h"

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <unistd.h>

#include <conio.h>

//переміщення по консолі

void Coord(int x, int y) {

COORD coord;

coord.X = x;

coord.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

//створення рамки

void Frame(int x1, int y1, int x2, int y2, int print) {

int i;

Coord(x1, y1);

printf("\311");

for (i = (x1 + 1); i <= (x2 - 1); i++) {

printf("\315");

if (print == 1)

usleep(3000);

}

printf("\273");

for (i = (y1 + 1); i <= (y2 - 1); i++) {

Coord(x1, i);

printf("\272");

if (print == 1)

usleep(3000);

Coord(x2, i);

printf("\272");

if (print == 1)

usleep(3000);

}

Coord(x1, y2);

printf("\310");

for (i = (x1 + 1); i <= (x2 - 1); i++) {

printf("\315");

if (print == 1)

usleep(3000);

}

printf("\274");

}

//вивід рамки та всіх даних на консоль

void Info() {

Close();

Frame(1, 1, 110, 25, 1);

Coord(50, 8);

printf("Course Work\n");

Coord(50, 10);

printf("Subject SDA\n");

Coord(50, 12);

printf("Variant 85\n");

Coord(45, 14);

printf("by Seletkov Vladislav\n");

Coord(52, 16);

printf("KV - 91");

Coord(51, 21);

printf("Kiev 2020\n");

}

//створення таблички з вимірами часу алгоритмів у пакетному режимі

void Out\_Table(float ordered, float random, float back\_ordered, int flag) {

if (flag == 1) {

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

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

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

printf("\t|\tSelection 8\t|\t%7.2f \t|\t%7.2f \t|\t%7.2f \t|\n", ordered, random, back\_ordered);

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

} else if (flag == 2) {

printf("\t| Insertion Exchange\t|\t%7.2f \t|\t%7.2f \t|\t%7.2f \t|\n", ordered, random, back\_ordered);

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

} else if (flag == 3) {

printf("\t|\tShell 1\t\t|\t%7.2f \t|\t%7.2f \t|\t%7.2f \t|\n", ordered, random, back\_ordered);

printf("\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\n\n");

}

}

//очищення екрану

void Close() {

system("cls");

}

//створення заставки

void Screen() {

Info();

char choice;

printf("\n\n\n\nSelect 1 to continue or select 0 to exit: ");

choice = getch();

while (1) {

if (choice != '0' && choice != '1') {

Error();

choice = getch();

} else if (choice == '1')

while (1) { //зациклення меню, що робить неможливим повернення до заставки

Close();

General\_Menu();

}

else exit(0);

}

}

//створення основного меню

void General\_Menu() {

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

char Points[5][60] = {

"1. Sorting multidimensional array algorithms in batch mode.",

"2. Sorting vector algorithms in batch mode.",

"3. Multidimensional array sorting algorithm test.",

"4. Vector sorting algorithm test.",

"5. Exit."

};

for (int i = 0; i < 5; i++)

printf("\t\t\t%s\n", Points[i]);

printf("\n\t\t\tPlease, choose an action");

Frame(1, 1, 110, 25, 0);

float ordered, random, back\_ordered;

int choice, flag;

char temp;

temp = getch();

while (1) {

if (temp != '1' && temp != '2' && temp != '3' && temp != '4' && temp != '5') {

Error();

temp = getch();

} else {

choice = (int) temp - 48;

break;

}

}

//змінна choice, через яку далі здійснюється

//1 - перехід до сортування багатовимірного масиву в пакетному режимі

//2 - перехід до сортування вектора в пакетному режимі

//3 - тестове сортування багатовимірного масиву з вибором заповнення та вибором алгоритма з виводом на екран

//4 - тестове сортування вектора з вибором заповнення та вибором алгоритма з виводом на екран

//5 - вихід з програми

if (choice == 1) {

Close();

flag = 1;

Input\_Array\_Size();

Close();

Create\_Array();

ordered = Array\_Selection\_Sort\_Measurement\_Orderly();

random = Array\_Selection\_Sort\_Measurement\_Disorderly();

back\_ordered = Array\_Selection\_Sort\_Measurement\_Inverted\_Orderly();

Out\_Table(ordered, random, back\_ordered, flag);

flag = 2;

ordered = Array\_Insertion\_Exchange\_Sort\_Measurement\_Orderly();

random = Array\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly();

back\_ordered = Array\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly();

Out\_Table(ordered, random, back\_ordered, flag);

flag = 3;

ordered = Array\_Shell\_Sort\_Measurement\_Orderly();

random = Array\_Shell\_Sort\_Measurement\_Disorderly();

back\_ordered = Array\_Shell\_Sort\_Measurement\_Inverted\_Orderly();

Free\_Array();

Out\_Table(ordered, random, back\_ordered, flag);

system("pause");

Close();

} else if (choice == 2) {

Close();

flag = 1;

Input\_Vector\_Size();

Close();

Create\_Vector();

ordered = Vector\_Selection\_Sort\_Measurement\_Orderly();

random = Vector\_Selection\_Sort\_Measurement\_Disorderly();

back\_ordered = Vector\_Selection\_Sort\_Measurement\_Inverted\_Orderly();

Out\_Table(ordered, random, back\_ordered, flag);

flag = 2;

ordered = Vector\_Insertion\_Exchange\_Sort\_Measurement\_Orderly();

random = Vector\_Insertion\_Exchange\_Sort\_Measurement\_Disorderly();

back\_ordered = Vector\_Insertion\_Exchange\_Sort\_Measurement\_Inverted\_Orderly();

Out\_Table(ordered, random, back\_ordered, flag);

flag = 3;

ordered = Vector\_Shell\_Sort\_Measurement\_Orderly();

random = Vector\_Shell\_Sort\_Measurement\_Disorderly();

back\_ordered = Vector\_Shell\_Sort\_Measurement\_Inverted\_Orderly();

Free\_Vector();

Out\_Table(ordered, random, back\_ordered, flag);

system("pause");

Close();

} else if (choice == 3 || choice == 4) {

Close();

while (Sorting\_Algorithm\_Menu(choice)); //передача вибору тестовго сортуання 3 - багатовимірного масиву або 4 - вектора до наступної функції

} else if (choice == 5)

exit(0);

Close();

}

//вибір алгоритма тестового сортування

//змінна result зберігає значення того, що було вибрано користувачем, багатовимірний масив чи вектор

int Sorting\_Algorithm\_Menu(int result) {

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

char Points[4][41] = {

"1. Selection sorting algorithm.",

"2. Insertion exchange sorting algorithm.",

"3. Shell sorting algorithm.",

"4. Go back to menu." };

for (int i = 0; i < 4; i++)

printf("\t\t\t\t%s\n", Points[i]);

printf("\n\t\t\t\tPlease, choose an action");

Frame(1, 1, 110, 25, 0);

int choice\_alg;

char temp;

temp = getch();

while (1) {

if (temp != '1' && temp != '2' && temp != '3' && temp != '4') {

Error();

temp = getch();

} else if (temp == '4')

return 0;

else {

choice\_alg = (int) temp - 48;

break;

}

}

Close();

while (Input\_Menu(result, choice\_alg));

}

//передача, вибраного користувачем, алгоритму сортування та передача //змінної result

//яка зберігає значення того, що було вибрано користувачем, //багатовимірний масив чи вектор

//вибір заповнення масиву для тестового сортування

int Input\_Menu(int result, int choice\_alg) {

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

char Points[4][27] = {

"1. Input orderly.",

"2. Input disorderly.",

"3. Input inverted orderly.",

"4. Go back to menu."

};

for (int i = 0; i < 4; i++)

printf("\t\t\t\t%s\n", Points[i]);

printf("\n\t\t\t\tPlease, choose an action");

Frame(1, 1, 110, 25, 0);

int choice\_input;

char temp;

temp = getch();

while (temp != '1' && temp != '2' && temp != '3' && temp != '4') {

Error();

temp = getch();

}

choice\_input = (int) temp - 48; //вибране користувачем заповнення масиву

//result: 3 - багатовимірний масив, 4 - вектор

//choice\_ald: 1 - сортування вибором, 2 - сортування вставкою-обміном, 3 - сортування Шелла

//choice\_input: 1 - впорядковане заповнення, 2 - не впорядковане заповнення, 3 - обернено-впорядковане заповнення, 4 - повернення до меню

switch (result) {

case 3:

switch (choice\_alg) {

case 1:

switch (choice\_input) {

case 1:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Orderly();

printf("Ordered Array: \n\n");

Output\_Array();

Array\_Selection\_Sort();

printf("Selection sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 2:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Disorderly();

printf("Disordered Array: \n\n");

Output\_Array();

Array\_Selection\_Sort();

printf("Selection sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 3:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Inverted\_Orderly();

printf("Inverted orderly Array: \n\n");

Output\_Array();

Array\_Selection\_Sort();

printf("Selection sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 4:

Close();

return 0;

}

break;

case 2:

switch (choice\_input) {

case 1:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Orderly();

printf("Ordered Array: \n\n");

Output\_Array();

Array\_Insertion\_Exchange\_Sort();

printf("Insertion exchange sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 2:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Disorderly();

printf("Disordered Array: \n\n");

Output\_Array();

Array\_Insertion\_Exchange\_Sort();

printf("Insertion exchange sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 3:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Inverted\_Orderly();

printf("Inverted orderly Array: \n\n");

Output\_Array();

Array\_Insertion\_Exchange\_Sort();

printf("Insertion exchange sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 4:

Close();

return 0;

}

break;

case 3:

switch (choice\_input) {

case 1:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Orderly();

printf("Ordered Array: \n\n");

Output\_Array();

Array\_Shell\_Sort();

printf("Shell sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 2:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Disorderly();

printf("Disordered Array: \n\n");

Output\_Array();

Array\_Shell\_Sort();

printf("Shell sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 3:

Close();

Input\_Array\_Size();

Create\_Array();

Input\_Array\_Inverted\_Orderly();

printf("Inverted orderly Array: \n\n");

Output\_Array();

Array\_Shell\_Sort();

printf("Shell sorted Array: \n\n\n");

Output\_Array();

Free\_Array();

printf("\n\n");

system("pause");

Close();

break;

case 4:

Close();

return 0;

}

break;

}

break;

case 4:

switch (choice\_alg) {

case 1:

switch (choice\_input) {

case 1:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Orderly();

printf("Ordered Vector: \n\n");

Output\_Vector();

Vector\_Selection\_Sort();

printf("Selection sorted Vector: \n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 2:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Disorderly();

printf("Disordered Vector: \n\n");

Output\_Vector();

Vector\_Selection\_Sort();

printf("Selection sorted Vector: \n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 3:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Inverted\_Orderly();

printf("Inverted orderly Vector: \n\n");

Output\_Vector();

Vector\_Selection\_Sort();

printf("Selection sorted Vector: \n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 4:

Close();

return 0;

}

break;

case 2:

switch (choice\_input) {

case 1:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Orderly();

printf("Ordered Vector: \n\n");

Output\_Vector();

Vector\_Insertion\_Exchange\_Sort();

printf("Insertion exchange sorted Vector:\n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 2:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Disorderly();

printf("Disordered Vector: \n\n");

Output\_Vector();

Vector\_Insertion\_Exchange\_Sort();

printf("Insertion exchange sorted Vector:\n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 3:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Inverted\_Orderly();

printf("Inverted orderly Vector: \n\n");

Output\_Vector();

Vector\_Insertion\_Exchange\_Sort();

printf("Insertion exchange sorted Vector:\n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 4:

Close();

return 0;

}

break;

case 3:

switch (choice\_input) {

case 1:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Orderly();

printf("Ordered Vector: \n\n");

Output\_Vector();

Vector\_Shell\_Sort();

printf("Shell sorted Vector: \n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 2:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Disorderly();

printf("Disordered Vector: \n\n");

Output\_Vector();

Vector\_Shell\_Sort();

printf("Shell sorted Vector: \n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 3:

Close();

Input\_Vector\_Size();

Create\_Vector();

Input\_Vector\_Inverted\_Orderly();

printf("Inverted orderly Vector: \n\n");

Output\_Vector();

Vector\_Shell\_Sort();

printf("Shell sorted Vector: \n\n\n");

Output\_Vector();

Free\_Vector();

printf("\n\n");

system("pause");

Close();

break;

case 4:

Close();

return 0;

}

break;

}

break;

}

Close();

}

**Menu.h**

#ifndef COURSE\_WORK\_MENU\_H

#define COURSE\_WORK\_MENU\_H

void Coord(int x, int y);

void Frame(int x1, int y1, int x2, int y2, int print);

void Info();

void Out\_Table(float ordered, float random, float back\_ordered, int flag);

void Close();

void Screen();

void General\_Menu();

int Sorting\_Algorithm\_Menu(int result);

int Input\_Menu(int result, int choice);

#endif //COURSE\_WORK\_MENU\_H

**SortingArray.c**

#include "SortingArray.h"

#include "CommonArray.h"

#include <math.h>

//сортування багатовимірного масиву вибором

clock\_t Array\_Selection\_Sort() {

int i, j, k, temp, L, R, imin, imax;

clock\_t time\_start, time\_stop;

time\_start = clock();

L = 0;

R = p - 1;

while (L < R) {

imin = L;

imax = L;

for (k = L + 1; k < R + 1; k++)

if (Array[k][0][0] < Array[imin][0][0])

imin = k;

else if (Array[k][0][0] > Array[imax][0][0])

imax = k;

if (imin != L)

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

for (j = 0; j < n; j++) {

temp = Array[imin][i][j];

Array[imin][i][j] = Array[L][i][j];

Array[L][i][j] = temp;

}

if (imax != R)

if (imax == L)

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

for (j = 0; j < n; j++) {

temp = Array[imin][i][j];

Array[imin][i][j] = Array[R][i][j];

Array[R][i][j] = temp;

}

else

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

for (j = 0; j < n; j++) {

temp = Array[imax][i][j];

Array[imax][i][j] = Array[R][i][j];

Array[R][i][j] = temp;

}

L = L + 1;

R = R - 1;

}

time\_stop = clock();

return time\_stop - time\_start;

}

//сортування багатовимірного масиву вставкою-обміном

clock\_t Array\_Insertion\_Exchange\_Sort() {

int i, j, k, temp;

clock\_t time\_start, time\_stop;

time\_start = clock();

for (int r = 1; r < p; r++) {

k = r;

while (k > 0 && Array[k][0][0] < Array[k - 1][0][0]) {

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

for (j = 0; j < n; j++) {

temp = Array[k][i][j];

Array[k][i][j] = Array[k - 1][i][j];

Array[k - 1][i][j] = temp;

}

k = k - 1;

}

}

time\_stop = clock();

return time\_stop - time\_start;

}

//сортування Шелла багатовимірного масиву

clock\_t Array\_Shell\_Sort() {

int Elem[m][n], Stages[(p - 1) / 4 + 1];

int t, j, k, r, q;

clock\_t time\_start, time\_stop;

time\_start = clock();

if (p < 4)

t = 1;

else

t = (int) log2f((float) p) - 1;

Stages[t - 1] = 1;

for (int i = t - 2; i >= 0; i--)

Stages[i] = 2 \* Stages[i + 1] + 1;

for (int g = 0; g < t; g++) {

k = Stages[g];

for (int i = k; i < p; i++) {

for (r = 0; r < m; r++)

for (q = 0; q < n; q++)

Elem[r][q] = Array[i][r][q];

j = i;

while (j >= k && Elem[0][0] < Array[j - k][0][0]) {

for (r = 0; r < m; r++)

for (q = 0; q < n; q++)

Array[j][r][q] = Array[j - k][r][q];

j = j - k;

}

for (r = 0; r < m; r++)

for (q = 0; q < n; q++)

Array[j][r][q] = Elem[r][q];

}

}

time\_stop = clock();

return time\_stop - time\_start;

}

**SortingArray.h**

#ifndef COURSE\_WORK\_SORTINGARRAY\_H

#define COURSE\_WORK\_SORTINGARRAY\_H

#include <time.h>

clock\_t Array\_Selection\_Sort();

clock\_t Array\_Insertion\_Exchange\_Sort();

clock\_t Array\_Shell\_Sort();

#endif //COURSE\_WORK\_SORTINGARRAY\_H

**SortingVector.c**

#include "SortingVector.h"

#include "CommonVector.h"

#include <math.h>

//сортування вектора вибором

clock\_t Vector\_Selection\_Sort() {

int L, R, imin, imax, temp;

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 (Vector[i] < Vector[imin])

imin = i;

else if (Vector[i] > Vector[imax])

imax = i;

if (imin != L) {

temp = Vector[imin];

Vector[imin] = Vector[L];

Vector[L] = temp;

}

if (imax != R)

if (imax == L) {

temp = Vector[imin];

Vector[imin] = Vector[R];

Vector[R] = temp;

} else {

temp = Vector[imax];

Vector[imax] = Vector[R];

Vector[R] = temp;

}

L = L + 1;

R = R - 1;

}

time\_stop = clock();

return time\_stop - time\_start;

}

//сортування вектора вставкою-обміном

clock\_t Vector\_Insertion\_Exchange\_Sort() {

int j, temp;

clock\_t time\_start, time\_stop;

time\_start = clock();

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

j = i;

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

temp = Vector[j];

Vector[j] = Vector[j - 1];

Vector[j - 1] = temp;

j = j - 1;

}

}

time\_stop = clock();

return time\_stop - time\_start;

}

//сортування Шелла вектора

clock\_t Vector\_Shell\_Sort() {

int Elem, t, j, k;

int Stages[(N - 1) / 4 + 1];

clock\_t time\_start, time\_stop;

time\_start = clock();

if (N < 4)

t = 1;

else

t = (int) log2f((float) N) - 1;

Stages[t - 1] = 1;

for (int i = t - 2; i >= 0; i--)

Stages[i] = 2 \* Stages[i + 1] + 1;

for (int q = 0; q < t; q++) {

k = Stages[q];

for (int i = k; i < N; i++) {

Elem = Vector[i];

j = i;

while (j >= k && Elem < Vector[j - k]) {

Vector[j] = Vector[j - k];

j = j - k;

}

Vector[j] = Elem;

}

}

time\_stop = clock();

return time\_stop - time\_start;

}

**SortingVector.h**

#ifndef COURSE\_WORK\_SORTINGVECTOR\_H

#define COURSE\_WORK\_SORTINGVECTOR\_H

#include <time.h>

clock\_t Vector\_Selection\_Sort();

clock\_t Vector\_Insertion\_Exchange\_Sort();

clock\_t Vector\_Shell\_Sort();

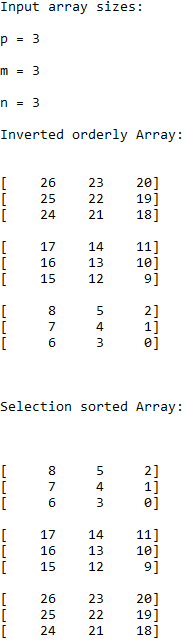
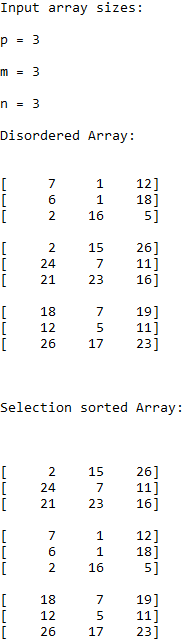
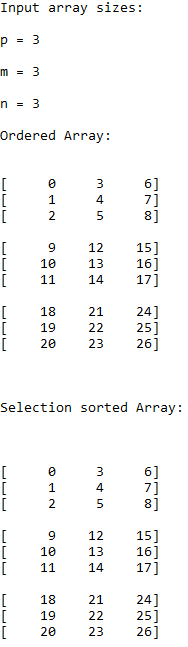
#endif //COURSE\_WORK\_SORTINGVECTOR\_H

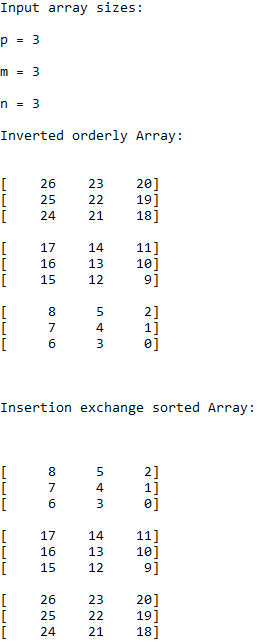
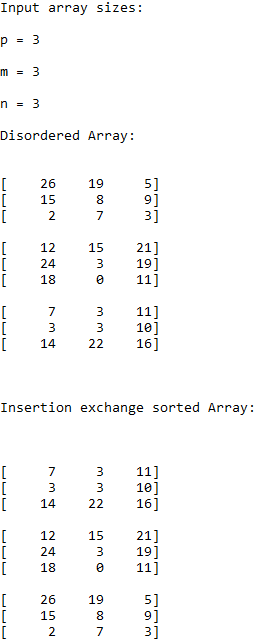
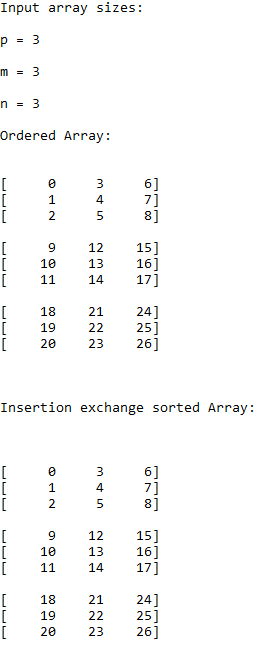
**Тести, що демонструють коректну роботу кожного із заданих алгоритмів для кожного випадку початкової впорядкованості масива.**

*Характеристики комп’ютера:*

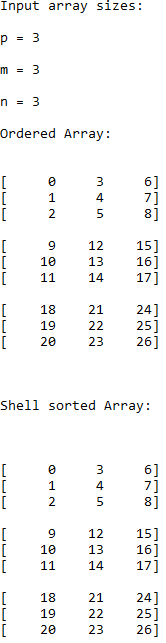
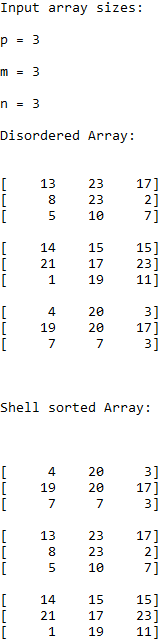
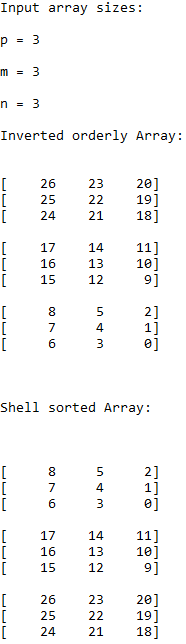
* Процесор: Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz;
* Встановлена пам'ять (ОЗУ): 6,00 ГБ;
* Тип системи: 64-розрядна операційна система, процесор х64;
* Операційна система: Windows 10 Pro, версія 1903;
* Компілятор: MinGW-w64 GDB.

**Сортування тривимірного масиву**

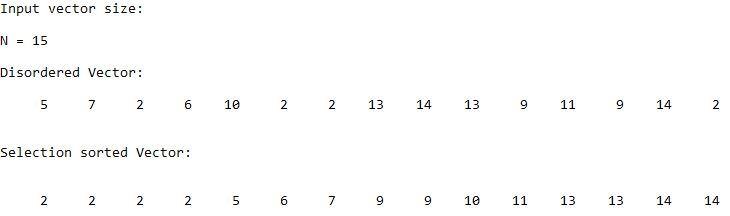
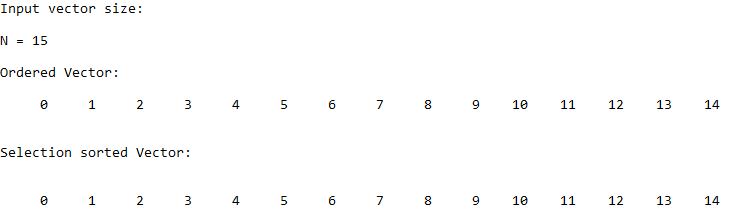
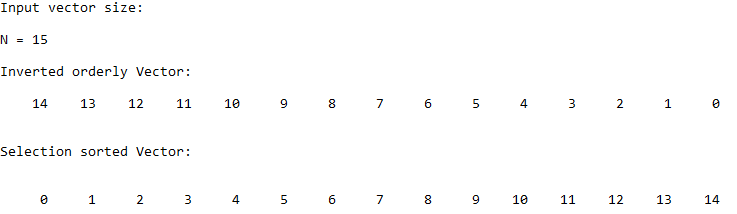
********Алгоритм сортування №8 методу прямого вибору**

**Гібридний алгоритм "вставка – обмін"**

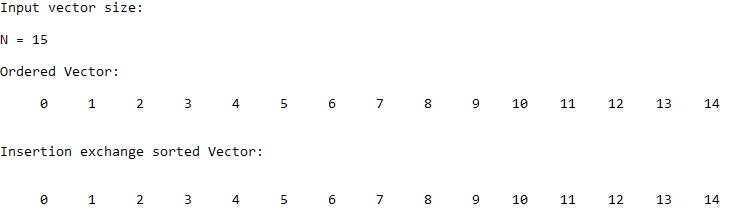
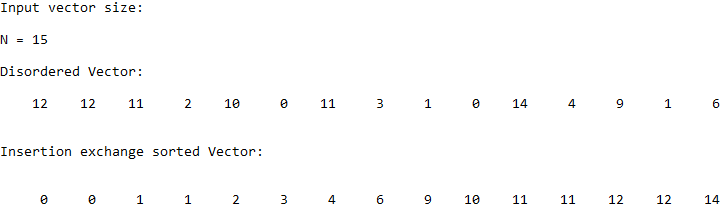
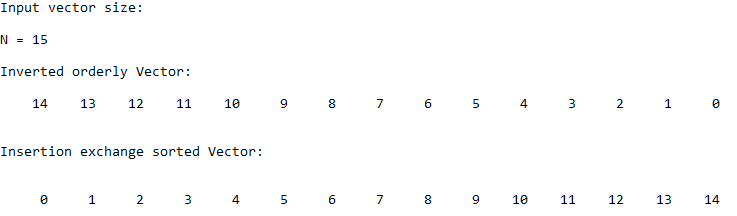
**Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2). Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються**



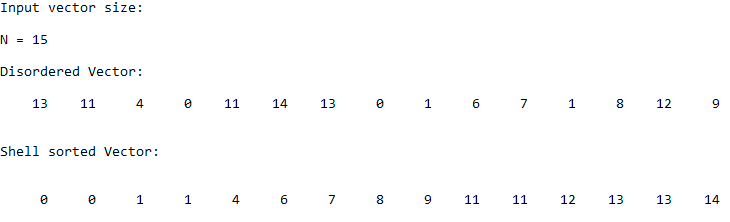
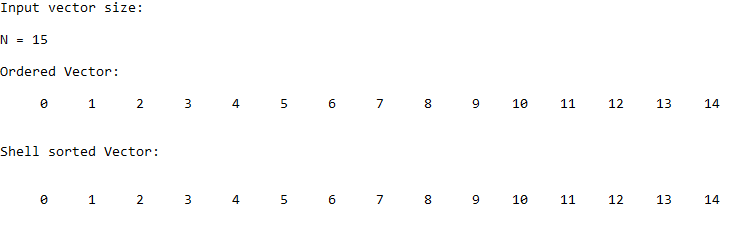
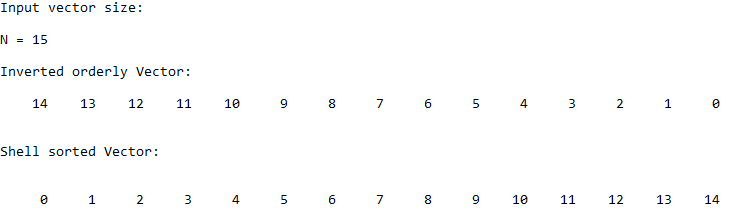
**Сортування вектора**

**Алгоритм сортування №8 методу прямого вибору**

**Гібридний алгоритм "вставка – обмін"**

****

**Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2). Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються**

****

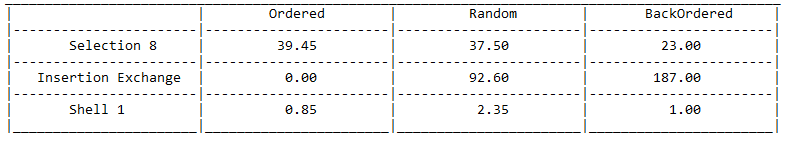
**Результати виміру часу роботи алгоритмів**

**Випадок дослідження І. Залежність часу роботи алгоритмів від розміру перерізів масиву**

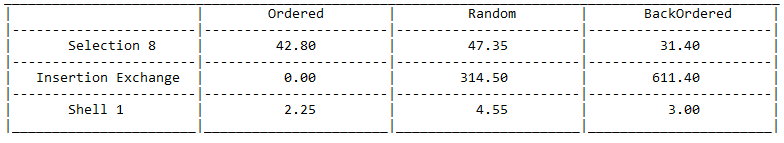
Кількість ключів (перерізів) P = const = 5000.

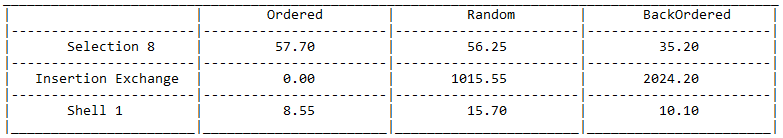
Форма перерізу – однакова (квадрат) M = var, N = var, M = N.

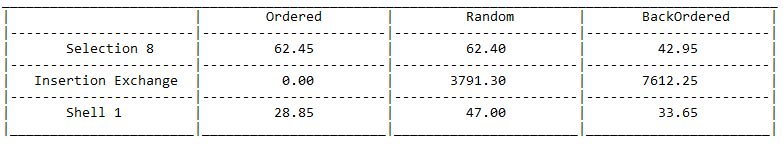
1) M = N = 1 (M\*N = 1).

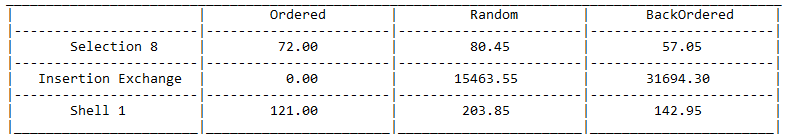


2) M = N = 2 (M\*N = 4).



3) M = N = 4 (M\*N = 16).

4) M = N = 8 (M\*N = 64).

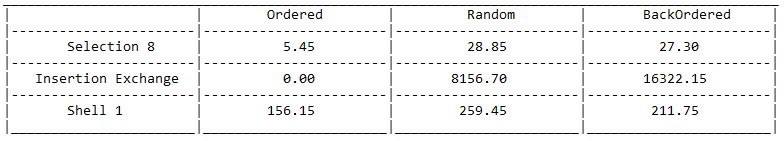
5) M = N = 16 (M\*N = 256).

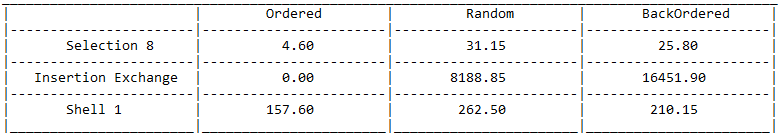
**Випадок дослідження ІІ. Залежність часу роботи алгоритмів від форми перерізів масиву**

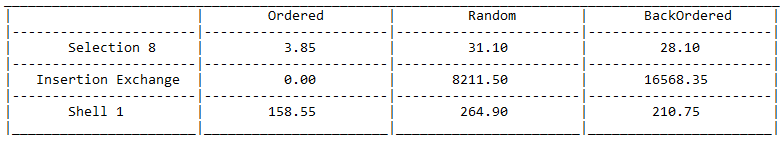
Кількість ключів (перерізів) P = const = 1500.

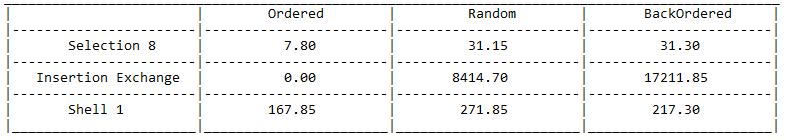
M = var, N = var, M\*N = const = 1600.

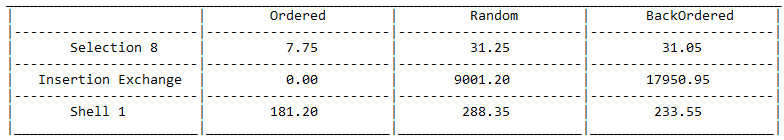
Загальна кількість елементів P\*M\*N = const = 2400000.

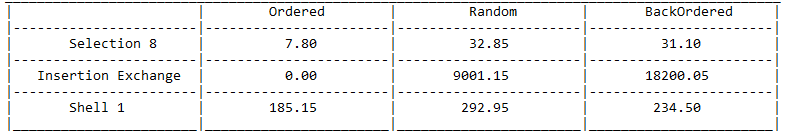
1) M = 2; N = 800.

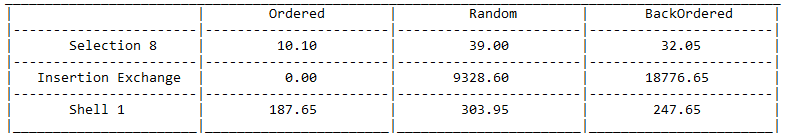
2) M = 4; N = 400.

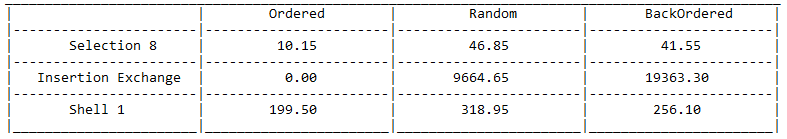
3) M = 8; N = 200.

4) M = 16; N = 100.

5) M = 100; N = 16.

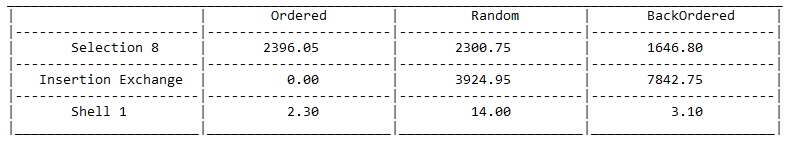
6) M = 200; N = 8.

7) M = 400; N = 4.

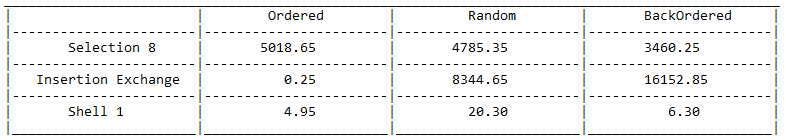
8) M = 800; N = 2.

**Результати виміру часу роботи алгоритмів вектора**

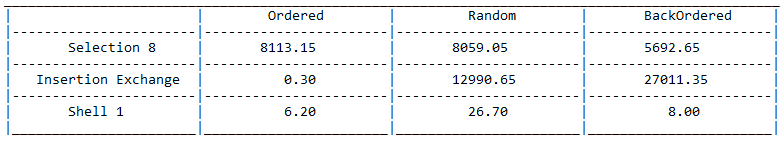
1) N = 50000.



2) N = 70000.



3) N = 90000.



**Стовпчикові діаграми отриманих результатів**

**Випадок дослідження І. Залежність часу роботи алгоритмів від розміру перерізів масиву**

**Випадок дослідження ІІ. Залежність часу роботи алгоритмів від форми перерізів масиву**

**Стовпчикові діаграми отриманих результатів для вектора**

**Порівняльний аналіз алгоритмів**

В курсовій роботі було розглянуто три алгоритми сортування: метод прямого вибору №8, гібридний алгоритм вставки-обміну та сортування методом Шелла №1. На основі цих алгоритмів було проведено порівняльне дослідження їхньої швидкодії. Задача полягала в тому, щоб відсортувати тривимірний масив даними алгоритмами таким чином, щоб перерізи масиву впорядковувалися за незменшенням значень вектору, який включає у себе перші елементи всіх перерізів масива. Виконано сортування вектора для трьох випадків, з яких видно, що сортування Шелла являє собою найкращий алгоритм для сортування одновимірного масиву через його принцип роботи, адже на відміну від інших алгоритмів його швидкодія T = 0(n \* (ln(n))2. Гібридний алгоритм являється найгіршим для сортування вектора, тому що виконується постійна перестановка елементів, в оберненому порядку виконується максимальна кількість обмінів між елементами, що позначається на швидкодії роботи алгоритму (найгірший випадок), але є найкращим для сортування впорядкованого вектора через мінімальну кількість ітерацій. Сортування алгоритму вибору №8 через постійний пробіг по масиву та значну кількість присвоєнь, показує досить поганий результат виміру часу сортування.

Згідно інформації, яку було вивчено із методички курсової роботи, були проведені заміри часу роботи кожного із алгоритмів в двох випадках дослідження. Після дослідження обох випадків, було виявлено, що у порівнянні з вектором, швидкодія деяких алгоритмів відрізняється в залежності від розмірів масиву.

**Аналіз методу сортування прямого вибору**

**Випадок дослідження І.**

З проведеного дослідження, можна сказати, що швидкість сортування алгоритму вибору №8 залежить від розміру перерізів, адже чим більше стає розмір, тим довше сортується масив. Згідно даних досліджень, сортування вибору №8 виконується довше у впорядкованому масиві, оскільки при малих взятих розмірах перерізи в оберненому порядку сортуються мінімальну кількість разів, а у впорядкованому випадку перевіряємо умови і постійно виконуємо ітерації imax елементу. Тому виходить, що обернено-впорядкований швидший за випадкове та впорядковане заповнення.

**Випадок дослідження II.**

В даному випадку, можна побачити, що час роботи алгоритмів сортування збільшується, тому що збільшується кількість рядків перерізу та зменшується кількість стовпчиків. На відміну від першого випадку, впорядкований масив сортується набагато швидше, аніжобернено-впорядкований, тому що маємо меншу кількість перерізів, що означає меншу кількість перевірок та ітерацій. Проте в цьому випадку дослідження, розміри перерізів набагато більші аніж в першому випадку. Це означає, шо при перестановці перерізів необхідно витратити набагато більше часу аніж в першому випадку, це пояснює більший час обернено-впорядкованого заповнення. Виходить, що найдовше сортується випадково заповнений масив, тому що там відбувається найбільша кількість перестановок, а найшвидше сортується впорядковано заповнений масив.

**Аналіз гібридного методу сортування "вставка – обмін"**

**Випадок дослідження І.**

Через збільшення розмірів масиву, відбувається збільшення часу роботи сортування масиву. Згідно даних досліджень та через малі взятих розміри маємо, що впорядковане сортування практично не виконує ніякої роботи сортування масиву, оскільки не виконується ніяких перестановок перерізів через перевірку умови на незбільшення, виконуються тільки ітерації змінної k. Якщо взяти окремо набагато більші розміри, то вже виконається деякий час роботи алгоритму. При сортуванні вектора в пакетному режимі та при немалих розмірах, було видно вже деякий час сортування. При оберненому порядку заповнення масиву, можна побачити найбільший час сортування, адже в цьому випадку при кожній ітерації ключа сортування виконується постійна перестановка сусідніх перерізів до найпершого перерізу масиву. Випадкове заповнення має також значну кількість перестановок. Таким чином, цей алгоритм являється дуже повільним для даної задачі. Впорядковане заповнення завжди буде найшвидшим, а обернено-впорядковане найповільнішим.

**Випадок дослідження II.**

В цьому випадку сортування відбувається аналогічно першому випадку, але час сортування менший, тому що кількість перерізів набагато менша.

**Аналіз методу сортування Шелла №1**

**Випадок дослідження І.**

В цілому, сортування Шелла показує себе не погано, але у порівнянні з алгоритмом вибору №8 гірше, тому що задана мала кількість перерізів та великий розмір масиву. Зазвичай, сортування Шелла виконується стабільно не довго у всіх випадках відсортованості, що однозначно є перевагою цього алгоритму. Очевидно, що впорядкований масив у всіх випадках буде сортуватися швидше за інші впорядкованості. Обернено-впорядкований масив виконує сортування швидше за випадково впорядкований, тому що алгоритм сортування виконує обмін елементів на відстані, що дорівнює довжині самого вектора та поділеній на поточний крок алгоритма, завжди в правильно заданому напрямку, а у випадково відсортованому масиві значення елементів можуть переставлятися в іншому напрямку та набагато далі ніж необхідно. Отже, впорядкований випадок є найшвидшим, а випадковий – найгірший.

**Випадок дослідження II.**

В даному випадку, сортування масиву алгоритмом Шелла та залежність результатів вимірювання аналогічні з першим випадком дослідження.

**Вплив різних геометричних розмірів багатовимірних масивів**

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

У першому випадку, можна побачити, що зі збільшенням розміру перерізів масиву, збільшується час сортування кожного алгоритму, тому що зі збільшенням кількості рядків збільшується час сортування. Особливо негативно себе проявляє гібридний алгоритм, який квадратично залежить від розміру масиву.

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

В другому випадку, спостерігається аналогічне збільшення часу сортування усіх алгоритмів як і у першому випадку. Через те, що кількість рядків збільшується та зменшується кількість стовпчиків, то реалізується перестановка перерізів через подвійний цикл з двома лічильниками, де другий цикл виконується M разів, переміщення M\*N разів. Отже, чим більша кількість рядків в масиві, то тим більше знадобиться часу для його сортування.

**Висновки**

Завершивши всі дослідження щодо виміру часу роботи алгоритмів масиву, та різного впливу розмірів на нього, можна зробити такі висновки:

1. Алгоритм методу сортування вибору №8 показав, що він працює швидко та оптимально, але враховуючи те, що через швидкодію комп’ютера, на якому виконано дослідження, була взята не велика кількість перерізів, за заданими розмірами алгоритм показав себе непогано.

2. Гібридний метод алгоритму сортування вставкою-обміном проявив себе найгіршим чином практично у всіх випадках заповнення, але показав себе найкраще в обох дослідах з впорядкованим заповненням масиву.

3. Сортування методом Шелла проявило себе стабільно та швидко, особливо для сортування вектора, але трохи гірше ніж сортування вибором №8 для тривимірного масиву. Я вважаю, що якщо взяти значно більшу кількість перерізів ніж було використано, то сортування Шелла буде найшвидшим.

4. Також, можна помітити залежність від кількості перерізів для всіх алгоритмів сортування та залежність від розміру перерізів масиву. Чим більше переріз, тим повільніша перестановка.

Отже, якщо потрібне стабільне, оптимальне сортування та не звертати уваги на розміри масиву, то метод Шелла підходить краще за всі алгоритми.

З іншого боку, якщо розмір масива є невеликим, то найкращим вибором алгоритму сортування є вибір №8.

Категорично не рекомендую використовувати гібридний метод алгоритму сортування "вставка – обмін".

**Список використаної літератури**

1) Марченко О. І., Марченко О., О «СТРУКТУРИ ДАНИХ ТА АЛГОРИТМИ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ СОРТУВАННЯ НА БАГАТОВИМІРНИХ МАСИВАХ» Київ, 2018.

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