НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

***КУРСОВА РОБОТА***

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

Виконала: Дахал К.

Група: КB-02

Номер залікової книжки: КВ-0207

Допущений до захисту

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

2 семестр 2020/2021

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп’ютерних систем

Узгоджено ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_\_\_\_2021р.

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

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

***Дослідження ефективності методів сортування прямої вставки(з лінійним пошуком місця вставки), методу прямого вибору, сортуванням Шелла на багатовимірних масивах***

Виконавець роботи:

Дахал Крістіна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 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.

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 № 39**

**Задача**

Впорядкувати тривимірний масив Аrr3D [P,M,N] таким чином: переставити перерізи масива за незменшенням значень вектору перших елементів кожного перерізу А [\*,1,1].

**Досліджувані методи та алгоритми**

1. Алгоритм сортування №3 методу прямої вставки з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар’єром (додаток 1, рис. №3).
2. Алгоритм сортування №4 методу прямого вибору (додаток 1, рис. №8).
3. Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2) (додаток 1, рис. №22). 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 Аrr3D[\*,1,1] 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**

1. Залежність часу роботи алгоритмів від розміру перерізів масива.
2. Залежність часу роботи алгоритмів від форми перерізів масива.

**Схема роботи заданих алгоритмів**

1. **Алгоритм сортування №3 методу прямої вставки з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар’єром (додаток 1, рис. №3).**

Принцип:

1. Присвоюємо змінній R значення n.
2. Просуваючись від 1 елементу до R-1 -го обмінюємо поточний елемент з наступним, якщо вони невпорядковані один відносно одного.
3. Змінній k присвоюємо значення змінної i, яка рівна номеру елементу, який востаннє обмінювали з наступним.
4. Змінній R присвоюємо значення змінної k.
5. Повторюємо, доки R>1.

Загальна схема

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1. 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 8 | 2 | 5 | 4 | 1 | 4 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 8 | 5 | 4 | 1 | 4 | 1 |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 5 | 8 | 4 | 1 | 4 | 2 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 5 | 4 | 8 | 1 | 4 | 3 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 5 | 4 | 1 | 8 | 3 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 5 | 4 | 1 | 8 | 3 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 3 | 4 | 1 | 8 | 3 | 2 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 4 | 1 | 5 | 8 | 2 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 4 | 1 | 5 | 8 | 2 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 2 | 1 | 4 | 5 | 8 | 1 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | N-1=4 | R | k |
| 1 | 2 | 4 | 5 | 8 | 0 | 0 |

Алгоритм сортування на мові С

#define VectorLength 10

**int** Vector[VectorLength+1];

clock\_t Insert3(**int** \*A, **int** N)

{

**int** j;

clock\_t time\_start, time\_stop;

time\_start = clock();

**for**(**int** i=2; i<N+1; i++){

A[0]=A[i];

j=i;

**while** (A[0]<A[j-1]) {

A[j]=A[j-1];

j=j-1;

}

A[j]=A[0];

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

1. **Алгоритм сортування №4 методу прямого вибору (додаток 1, рис. №8).**

В кожен момент часу масив розглядається складеним із двох частин. Перша – вже відсортована частина, друга – ще невідсортована.

Спочатку відсортована частина приймається рівною в нуль елементів.

Беруться ліва границя L та права R, тоді невідсортована частина буде знаходитися між правою та лівою границями. На початку присвоюємо границям значення L = 0 та R = N – 1, тобто останньому елементу. Далі виконуємо сортування

Процес сортування:

1) Знаходимо у невідсортованій частині ( діапазон якої спочатку від нуля до N-1 елементів, де N – кількість елементів масиву) найменший та на більший елементи одночасно та запам’ятовуємо їх позиції.

2) Якщо знайдений мінімальний елемент не стоїть на початку невідсортованої частини, то ставимо його туди, а якщо знайдений максимальний не стоїть в кінці, то з останнім елементом масиву.

3) Таким чином два елементи стають відсортованими, тобто відсортована частина збільшується на два елементи, а невідсортована зменшується на два елементи.

4) Виконуємо дії пунктів 1-2 у ще невідсортованій частині, яка залишилась доки не буде впорядкований весь масив, тобто доки L та R не перетнуться.

Алгоритм сортування на мові С

clock\_t Select4(**int** \*A, **int** N)

{

**int** L, R, imin, imax, tmp;

clock\_t time\_start, time\_stop;

time\_start = clock();

L=0; R=N-1;

**while** (L<R){

imin=L; imax=L;

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

**if** (A[i]<A[imin]) imin=i;

**else**

**if** (A[i]>A[imax]) imax=i;

tmp=A[imin];

A[imin]=A[L];

A[L]=tmp;

**if** (imax==L) {

tmp=A[imin];

A[imin]=A[R];

A[R]=tmp;

}

**else** {

tmp=A[imax];

A[imax]=A[R];

A[R]=tmp;

}

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

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

1. **Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2) (додаток 1, рис. №22).** *Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються.*

Сортування Шелла - це алгоритм сортування, який є покращений методом методу прямої вставки .

Принцип сортування Шелла

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

Етап перший:

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

Етап другий:

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

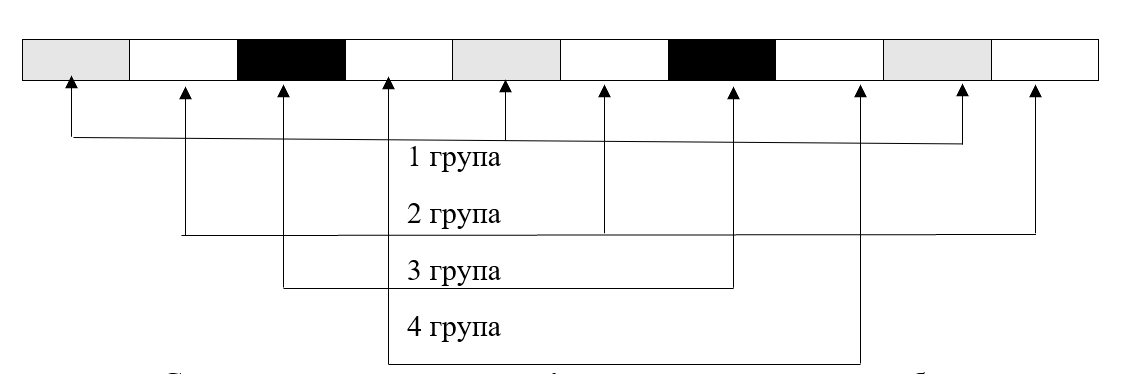
Етап третій:

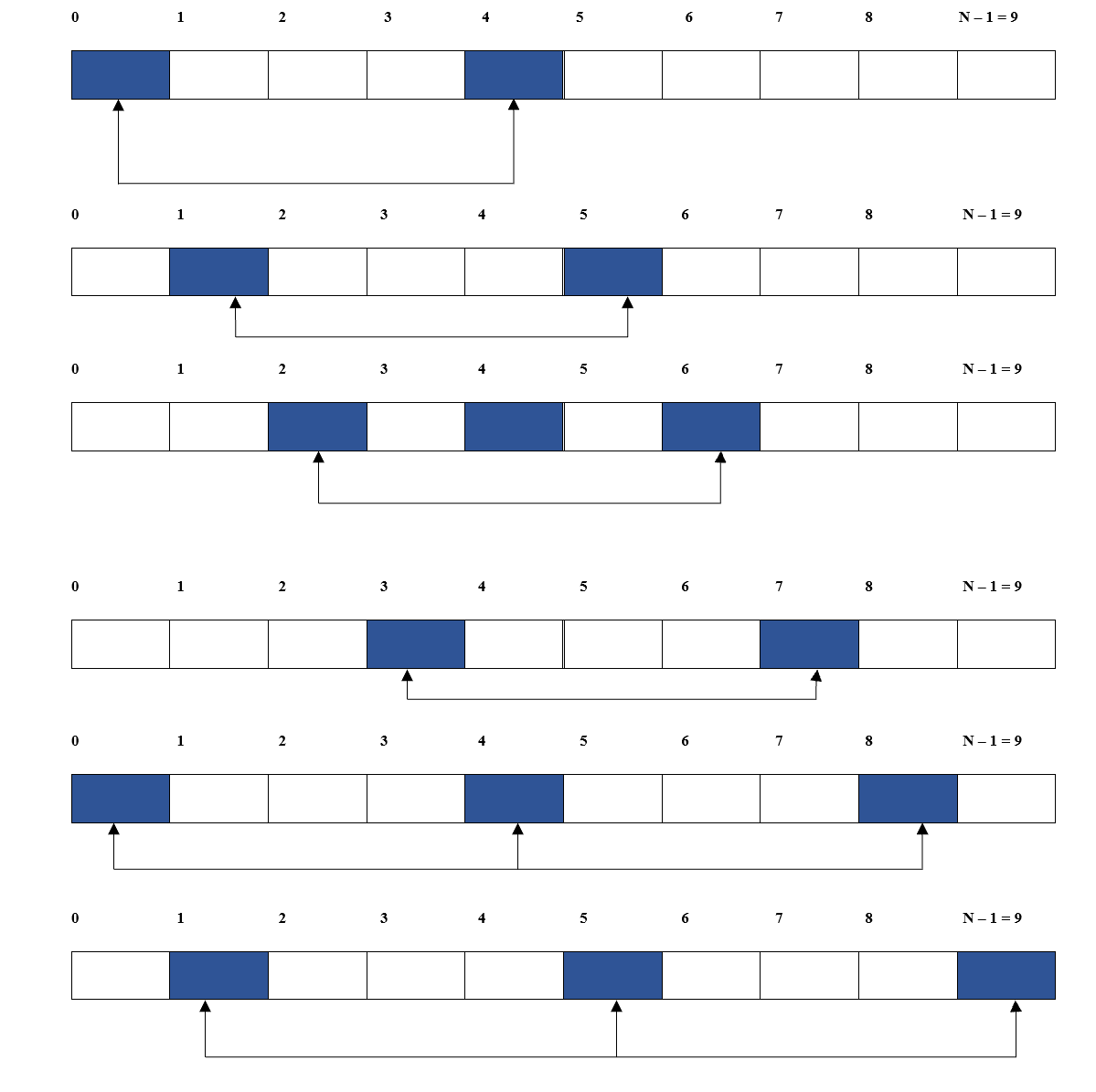
Виконується аналогічно тільки для елементів на відстані в одну позицію, тобто виконується сортування масиву звичайною прямою вставкою.

Можливі два способи сортування груп:

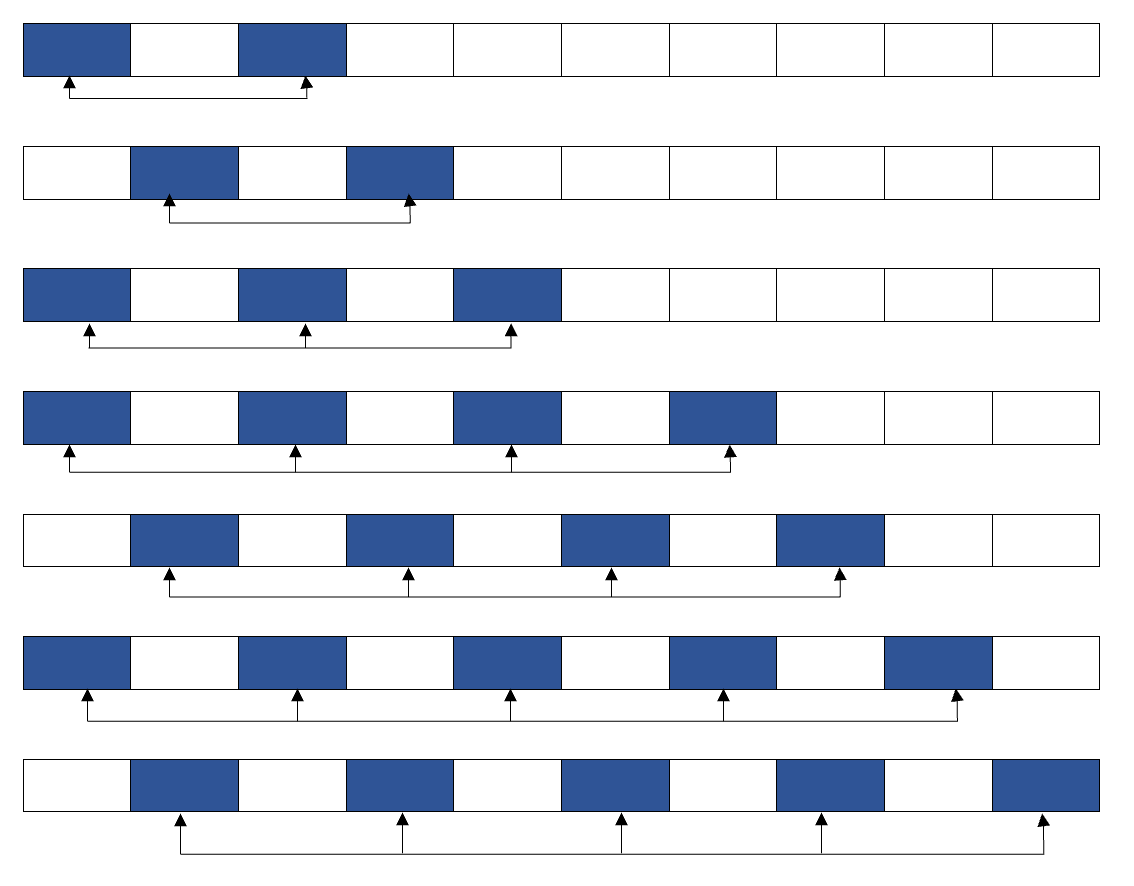
1. Виконується повне сортування методом прямої вставки по черзі всіх елементів кожної групи окремо. Тобто спочатку елемент 1 групи, потім другої і так далі;
2. Виконується вставка для першого елементу 1 групи, потім для першого елементу другої групи, 1 елементу третьої групи і четвертої. Потім аналогічно для 2 елементі, 3 елемента, 4 елемента кожної групи.

Загальна схема





**Схема другого проходу:**



Алгоритм сортування на мові С

clock\_t Shell\_1(**int** \*A, **int** N)

{

**int** Elem, t, j, k;

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** p=0; p<t; p++){

k=Stages[p];

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

Elem=A[i];

j=i;

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

A[j]=A[j-k];

j=j-k;

}

A[j]=Elem;

}

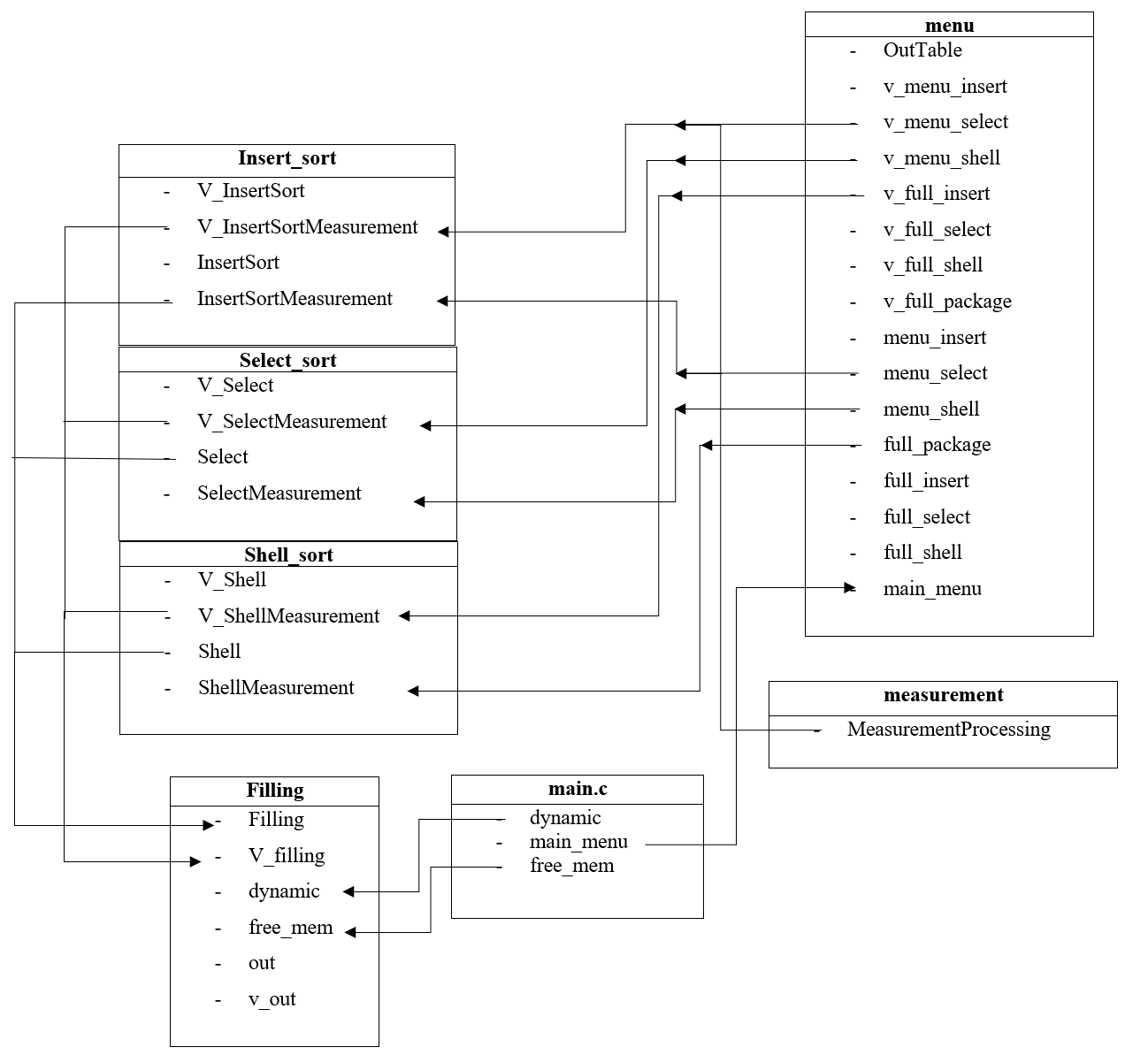
}

time\_stop = clock();

**return** time\_stop - time\_start;

}

**Структурна схема взаємовикликів процедур та функцій**

****

**Схема імпорту/експорту модулів**

global

main

sorting

menu

filling

measurement

**Опис призначення процедур та функцій**

1. **Модуль Filling**

Призначення: у цьому модулі виконується заповнення та виведення тривимірного масиву A та вектору Vector; виділяється та звільняється пам’ять для тривимірного масиву А.

* *Filling* – функція призначена для заповнення тривимірного масиву 3 способами на вибір;
* *V\_filling* – функція призначена для заповнення вектору 3 способами на вибір;
* *dynamic* – виділення пам’яті для тривимірного масиву;
* *free\_mem* – звільнення пам’яті масиву;
* *out* – функція для виведення масиву до та після сортування;
* *v\_out* – функція для виведення вектору до та після сортування;

1. **Модуль Insert\_sort**

Призначення: сортування масиву та вектору методомпрямої вставки з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар’єром.

* *V\_InsertSort* – сортування вектору заданим алгоритмом;
* *V\_InsertSortMeasurement* – вимір часу сортування вектора;
* *InsertSort* – сортування масиву заданим алгоритмом;
* *InsertSortMeasurement* – вимір часу сортування масиву;

1. **Модуль Select\_sort**

Призначення: сортування масиву та вектору методом прямого вибору.

* *V\_Select* - сортування вектору заданим алгоритмом;
* *V\_SelectMeasurement* - вимір часу сортування вектора;
* *Select* - сортування масиву заданим алгоритмом;
* *SelectMeasurement* - вимір часу сортування масиву;

1. **Модуль Shell\_sort**

Призначення: сортування масиву та вектору методом сортування Шелла (класичний варіант на основі прямої вставки №2).

* *V\_Shell* - сортування вектору заданим алгоритмом;
* *V\_ShellMeasurement* - вимір часу сортування вектору;
* *Shell* - сортування масиву заданим алгоритмом;
* *ShellMeasurement* - вимір часу сортування масиву;

1. **Модуль Measurement**

Призначення: для врахування похибки використана методика усереднення значень вимірів.

* *MeasurementProcessing* - функція обробки даних виміру часу;

1. **Модуль Menu**

Призначення: у цьому модулі зібрані результати всіх вимірів, які можна продивитися за допомогою діалогового інтерфейсу.

* *OutTable* – шаблон таблиці для виводу результатів сортування
* *main\_menu* – діалоговий інтерфейс з можливістю обрати та подивитися результати сортування будь-яким алгоритмом або вивести результати у пакетному режимі;
* *v\_menu\_insert* – функція для вибору вектора для подальшого сортування методом вставки (упорядкований, рандомний або обернено-упорядкований вектор);
* *v\_menu\_select* - функція для вибору вектора для подальшого сортування методом вибору (упорядкований, рандомний або обернено-упорядкований вектор);
* *v\_menu\_shell* - функція для вибору вектора для подальшого сортування методом Шелла (упорядкований, рандомний або обернено-упорядкований вектор);
* *v\_full\_insert* – вивід таблиці з результатами сортування трьох векторів (упорядкований, рандомний або обернено-упорядкований вектор) методом вставки;
* *v\_full\_select* – вивід таблиці з результатами сортування трьох векторів (упорядкований, рандомний або обернено-упорядкований вектор) методом вибору;
* *v\_full\_shell* – вивід таблиці з результатами сортування трьох векторів (упорядкований, рандомний або обернено-упорядкований вектор) методом Шелла;
* *v\_full\_package* – вивід таблиці з результатами сортування всіх векторів (упорядкований, рандомний або обернено-упорядкований вектор) трьома алгоритмами;
* *menu\_insert* - функція для вибору масиву для подальшого сортування методом вставки (упорядкований, рандомний або обернено-упорядкований масив);
* *menu\_select* - функція для вибору масиву для подальшого сортування методом вибору (упорядкований, рандомний або обернено-упорядкований масив);
* *menu\_shell* - функція для вибору масиву для подальшого сортування методом Шелла (упорядкований, рандомний або обернено-упорядкований масив);
* *full\_insert* - вивід таблиці з результатами сортування трьох масивів (упорядкований, рандомний або обернено-упорядкований масив) методом вставки;
* *full\_select* - вивід таблиці з результатами сортування трьох масивів (упорядкований, рандомний або обернено-упорядкований масив) методом вибору;
* *full\_shell* - вивід таблиці з результатами сортування трьох масивів (упорядкований, рандомний або обернено-упорядкований масив) методом Шелла;
* *full\_package* - вивід таблиці з результатами сортування всіх масивів (упорядкований, рандомний або обернено-упорядкований вектор) трьома алгоритмами;

**Текст головної програми та модулів**

**MAIN.C**

#include "menu.h"

**int** main()

{

dynamic();//оголошення тривимірного масиву та виділення для нього пам'яті

main\_menu();//головне меню

free\_mem();//звільнення пам'яті

//out();

//v\_out();

**return** 0;

}

**FILLING.C**

#include "filling.h"//підключення імпорту

**void** Filling(**int** a)//filling 3D massive

{

**switch** (a)

{

**case** 1:

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

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

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

A[k][i][j] = (i + j + k) \* (j + 1);//упорядковане заповнення

**break**;

**case** 2:

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

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

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

A[k][i][j] = rand()%(P\*M\*N);//рандомне заповнення

**break**;

**case** 3:

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

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

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

A[k][i][j] = j -i -k + 7;//обернено-упорядковане заповнення

**break**;

}

}

**void** out()//функція для виведення масиву до і після сортування

{

Filling(3);

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

{

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

{

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

{

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

}

printf("\n");

}

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

}

printf("SORTED RESULT\n");

InsertSort();

//Select();

//Shell();

//V\_InsertSort(Vector, VectorLength);

//V\_Select(Vector, VectorLength);

//V\_Shell(Vector, VectorLength);

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

{

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

{

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

{

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

}

printf("\n");

}

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

}

}

**void** dynamic()//виділення пам'яті

{

**int** \*\*\*A;

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

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

{

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

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

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

}

}

**void** free\_mem()//звільнення пам'яті

{

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

{

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

free(A[k][i]);

free(A[k]);

}

free(A);

}

**void** V\_Filling(**int** \*A, **int** n, **int** b)//filling vector

{

**switch** (b)

{

**case** 1:

**for** (i = 0; i < n; i++)

A[i] = i + 7 ;//упорядковане заповнення

**break**;

**case** 2:

**for** (i = 0; i < n; i++)

A[i] = rand()%20;//рандомне заповнення

**break**;

**case** 3:

**for** (i = 0; i < n; i++)

A[i] = n - i;//обернено-упорядковане заповнення

**break**;

}

}

**void** v\_out()//функція для виведення вектору до і після сортування

{

V\_Filling(Vector, VectorLength, 3);

**for**(i=0; i<VectorLength; i++)

{

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

}

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

//V\_InsertSort(Vector, VectorLength);

//V\_Select(Vector, VectorLength);

//V\_Shell(Vector, VectorLength);

**for**(**int** i=0; i<VectorLength; i++)

{

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

}

}

**INSERT\_SORT.C**

#include "sorting.h"

clock\_t V\_InsertSort(**int** \*A, **int** n)//сортування вектору методом вставки

{

**int** j;

clock\_t time\_start, time\_stop;

time\_start = clock();

**for**(**int** i=2; i<n+1; i++)

{

A[0]=A[i];

j=i;

**while** (A[0]<A[j-1])

{

A[j]=A[j-1];

j=j-1;

}

A[j]=A[0];

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

**void** V\_InsertSortMeasurement(**int** a)//вимір часу роботи

{

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

{

V\_Filling(Vector, VectorLength, a);

Res[i] = V\_InsertSort(Vector, VectorLength);

}

}

clock\_t InsertSort()//сортування масиву методом вставки

{

**int** m;

clock\_t time\_start, time\_stop;

time\_start = clock();

**for**(**int** k = 2; k<P; k++)

{

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

{

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

{

A[0][i][j]=A[k][i][j];

}

}

m=k;

**while** (A[0][0][0]<A[m-1][0][0])

{

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

{

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

{

A[m][i][j]=A[m-1][i][j];

}

}

m=m-1;

}

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

{

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

{

A[m][i][j]=A[0][i][j];

}

}

}

time\_stop=clock();

**return** time\_stop-time\_start;

}

**void** InsertSortMeasurement(**int** a)//вимір часу роботи

{

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

{

Filling(a);

Res[i] = InsertSort();

}

}

**SELECT\_SORT.C**

#include "sorting.h"

clock\_t V\_Select(**int** \*A, **int** n)//сортування вектору методом вибору

{

**int** L, R, imin, imax, tmp;

clock\_t time\_start, time\_stop;

time\_start = clock();

L=0;

R=n-1;

**while** (L<R)

{

imin=L;

imax=L;

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

**if** (A[i]<A[imin]) imin=i;

**else** **if** (A[i]>A[imax]) imax=i;

tmp=A[imin];

A[imin]=A[L];

A[L]=tmp;

**if** (imax==L)

{

tmp=A[imin];

A[imin]=A[R];

A[R]=tmp;

}

**else**

{

tmp=A[imax];

A[imax]=A[R];

A[R]=tmp;

}

L=L+1;

R=R-1;

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

**void** V\_SelectMeasurement(**int** b)//вимір часу роботи

{

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

{

V\_Filling(Vector, VectorLength, b);

Res[i] = V\_Select(Vector, VectorLength);

}

}

clock\_t Select()//сортування масиву методом вибору

{

**int** L, R, kmin, kmax, tmp;

clock\_t time\_start, time\_stop;

time\_start = clock();

L=0;

R=P-1;

**while** (L<R)

{

kmin=L;

kmax=L;

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

**if** (A[k][0][0]<A[kmin][0][0]) kmin=k;

**else** **if** (A[k][0][0]>A[kmax][0][0]) kmax=k;

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

{

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

{

tmp=A[kmin][0][0];

A[kmin][0][0]=A[L][0][0];

A[L][0][0]=tmp;

}

}

**if** (kmax==L)

{

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

{

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

{

tmp=A[kmin][i][j];

A[kmin][i][j]=A[L][i][j];

A[L][i][j]=tmp;

}

}

}

**else**

{

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

{

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

{

tmp=A[kmax][i][j];

A[kmax][i][j]=A[R][i][j];

A[R][i][j]=tmp;

}

}

}

L=L+1;

R=R-1;

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

**void** SelectMeasurement(**int** a)//вимір часу роботи

{

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

{

Filling(a);

Res[i] = Select();

}

}

**SHELL\_SORT.C**

#include "sorting.h"

clock\_t V\_Shell(**int** \*A, **int** n)//сортування вектору методом Шелла

{

clock\_t time\_start, time\_stop;

**int** **const** max\_t = (n - 1) / 4 + 1;

**int** arr1[max\_t];

**int** temp;

**int** j, k, t, w;

**if** (n < 4)

{

t = 0;

}

**else**

{

t = (log(n) / log(2)) - 1;

}

arr1[t] = 0;

time\_start = clock();

**for** (**int** i = t - 1; i >= 0; i--)

arr1[i] = 2 \* arr1[i + 1] + 1;

**for** (**int** w = 0; w < t; w++)

{

k = arr1[w];

**for** (**int** i = k; i < n; i++)

{

temp = A[i];

j = i;

**while** ((j >= k) && (temp < (A[j - k])))

{

A[j] = A[j - k];

j = j - k;

}

A[j] = temp;

}

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

**void** V\_ShellMeasurement(**int** b)//вимір часу роботи

{

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

{

V\_Filling(Vector, VectorLength, b);

Res[i] = V\_Shell(Vector, VectorLength);

}

}

clock\_t Shell()//сортування масиву методом Шелла

{

clock\_t time\_start, time\_stop;

**int** **const** max\_t = (P - 1) / 4 + 1;

**int** arr1[max\_t];

**int** temp;

**int** m, r, t, w;

time\_start = clock();

**if** (P < 4)

{

t = 0;

}

**else**

{

t = (log(P) / log(2)) - 1;

}

arr1[t-1] = 1;

**for** (**int** k = t - 2; k >= 0; k--)

arr1[k] = 2 \* arr1[k + 1] + 1;

**for** (**int** w = 0; w < t; w++)

{

r = arr1[w];

**for** (**int** k = r; k < P; k++)

{

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

{

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

{

temp = A[k][0][0];

}

}

m = k;

**while** ((m >= r) && (temp < (A[m - r][0][0])))

{

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

{

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

{

A[m][i][j] = A[m - r][i][j];

}

}

m = m - r;

}

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

{

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

{

A[m][0][0] = temp;

}

}

}

}

time\_stop = clock();

**return** time\_stop - time\_start;

}

**void** ShellMeasurement(**int** a)//вимір часу роботи

{

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

{

Filling(a);

Res[i] = Shell();

}

}

**MEASUREMENT.C**

#include "Measurement.h"

clock\_t Res[measurements\_number];

**float** MeasurementProcessing()

{

**long** **int** Sum;

**float** AverageValue;

clock\_t buf;

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

**int** k = rejected\_number;

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

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

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

buf = Res[i];

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

Res[i + 1] = buf;

k = i;

}

}

R = k;

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

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

buf = Res[i];

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

Res[i + 1] = buf;

k = i;

}

}

L = k + 1;

}

Sum=0;

**for** (**int** i = rejected\_number + min\_max\_number; i < measurements\_number - min\_max\_number; i++)

Sum = Sum + Res[i];

AverageValue = (**float**)Sum/(**float**)(measurements\_number - 2\*min\_max\_number - rejected\_number);

printf("");

**return** AverageValue;

}

**MEASUREMENT.H**

#ifndef MEASUREMENT\_H\_INCLUDED

#define MEASUREMENT\_H\_INCLUDED

#include <time.h>

// загальна кількість вимірів часу роботи алгоритма

#define measurements\_number 28

#define rejected\_number 2

#define min\_max\_number 3

// Масив значень часу роботи алгоритма

**extern** clock\_t Res[measurements\_number];

// Функція обробки і усереднення значень вимірів

// часу роботи алгоритма.

// Повертає усереднене значення часу роботи алгоритма.

**float** MeasurementProcessing();

#endif // MEASUREMENT\_H\_INCLUDED

**SORTING.H**

#ifndef SORTING\_H\_INCLUDED

#define SORTING\_H\_INCLUDED

#include "filling.h"

#include "measurement.h"

#include "global.h"

#include <math.h>

clock\_t V\_InsertSort(**int** \*A, **int** n);

clock\_t V\_Select(**int** \*A, **int** n);

clock\_t V\_Shell(**int** \*A, **int** n);

clock\_t InsertSort();

clock\_t Select();

clock\_t Shell();

**void** V\_InsertSortMeasurement(**int** a);

**void** V\_SelectMeasurement(**int** b);

**void** V\_ShellMeasurement(**int** b);

**void** InsertSortMeasurement(**int** a);

**void** SelectMeasurement(**int** a);

**void** ShellMeasurement(**int** a);

**FILLING.H**

#ifndef GENERATOR\_H\_INCLUDED

#define GENERATOR\_H\_INCLUDED

#**pragma once // директива препроцесора, щоб уникнути багаторазове включення модулів**

#include "global.h"

#include "sorting.h"

**void** Filling(**int** a);

**void** V\_Filling(**int** \*A, **int** n, **int** b);

**void** dynamic();

**void** free\_mem();

**void** out();

**void** v\_out();

#endif // GENERATOR\_H\_INCLUDED

**GLOBAL.H**

#ifndef V\_MEASUREMENT\_H\_INCLUDED

#define V\_MEASUREMENT\_H\_INCLUDED

#include <conio.h>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#define M 1

#define N 1

#define P 150000

**int** i, j, k;

**int** A[P][M][N];

**float** select\_choice, ordered, random, backordered;

#define VectorLength 12800

**int** Vector[VectorLength];

#endif // V\_MEASUREMENT\_H\_INCLUDED

**MENU.H**

#include "sorting.h" //Піключення модуля сортування

#include "measurement.h" // Підкючення модуля вимірювання

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

**void** OutTable();

**void** v\_menu\_insert();

**void** v\_menu\_select();

**void** v\_menu\_shell();

**void** menu\_insert();

**void** menu\_select();

**void** menu\_shell();

**void** v\_full\_insert();

**void** v\_full\_select();

**void** v\_full\_shell();

**void** v\_full\_package();

**void** full\_insert();

**void** full\_select();

**void** full\_shell();

**void** full\_package();

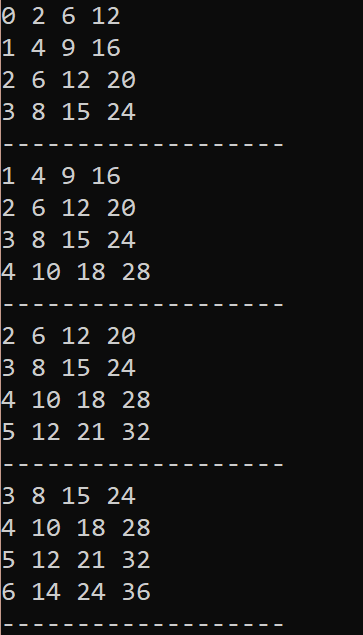
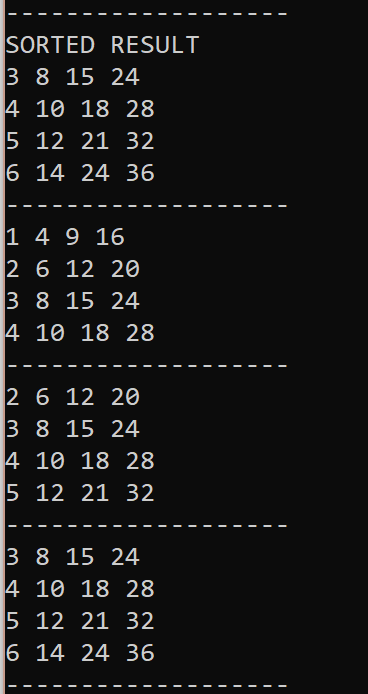
#endif // MENU\_H\_INCLUDED

**Тести на якість сортування масивів**

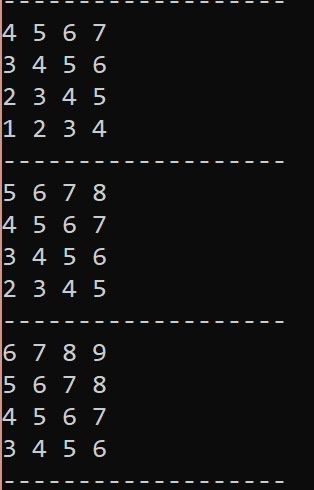
1. **Методом вставки:**

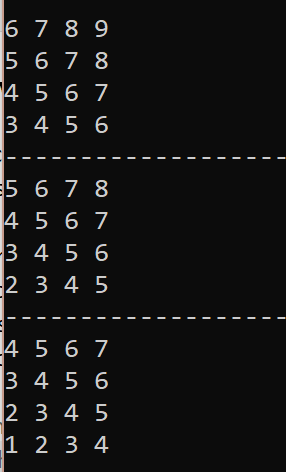
* упорядкований масив

до: після



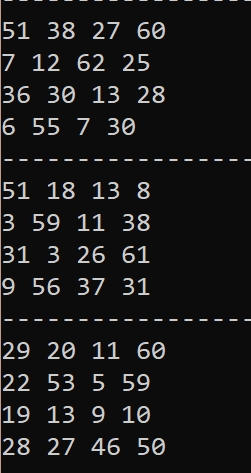
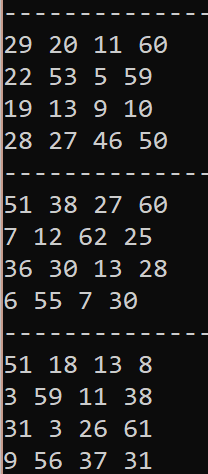
* обернено-впорядкований масив

до після



* масив випадкових чисел

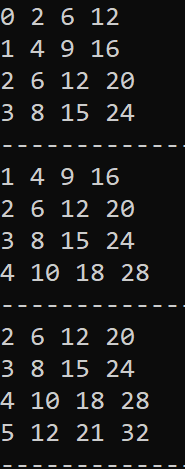
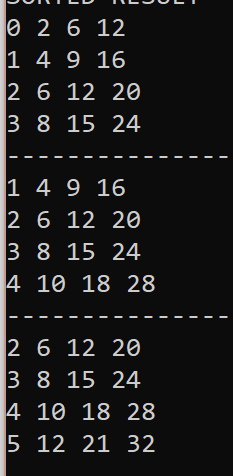
до після



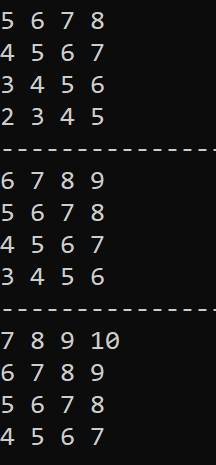
1. **Методом вибору:**

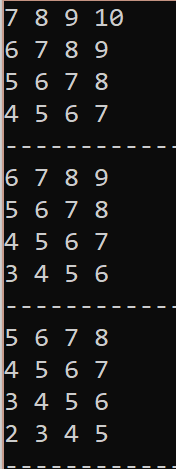
* упорядкований масив

до після



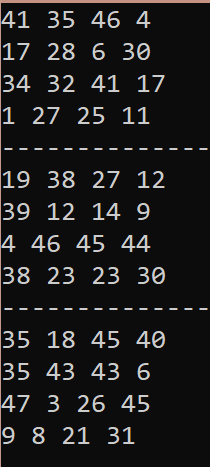
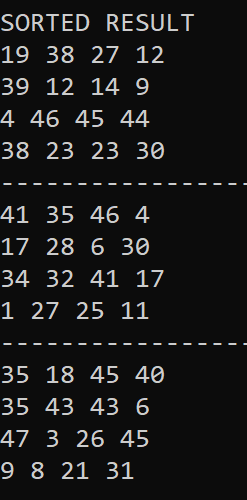
* обернено-впорядкований масив

до після



* масив випадкових чисел

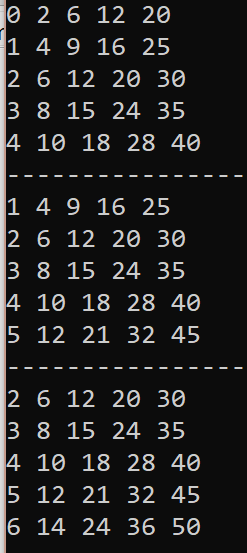
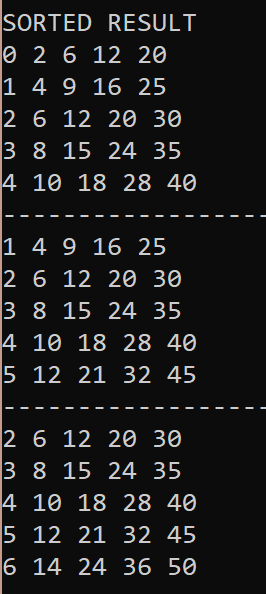
до після



1. **Методом Шелла:**

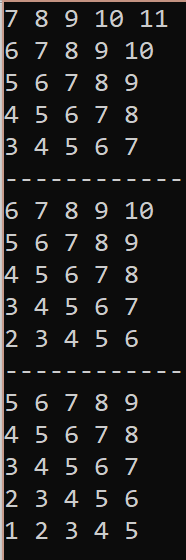
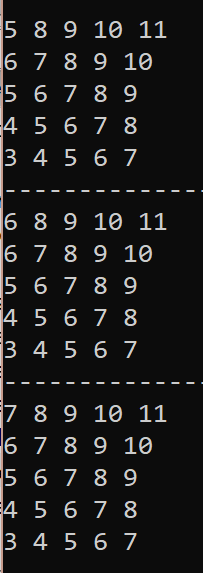
* упорядкований масив

до після

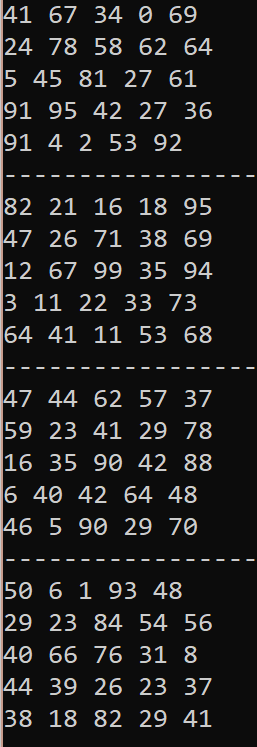
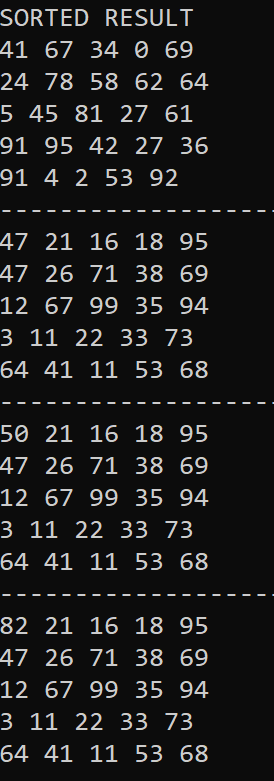


* обернено-впорядкований масив

до після



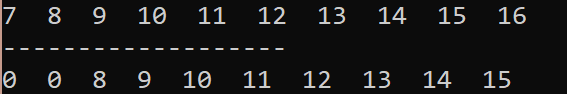
* масив випадкових чисел

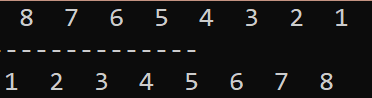
**Тести на якість сортування векторів**

1. **Методом вставки:**

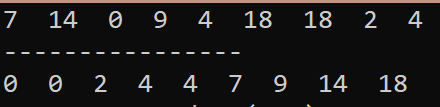
* упорядкований масив



* обернено-впорядкований масив

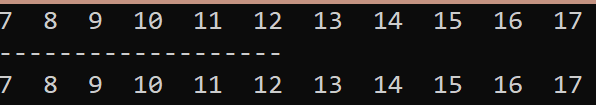


* масив випадкових чисел

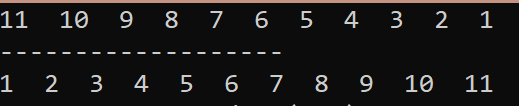


1. **Методом вибору:**

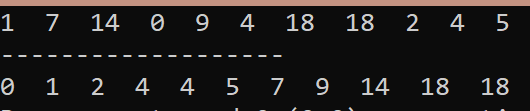
* упорядкований масив



* обернено-впорядкований масив

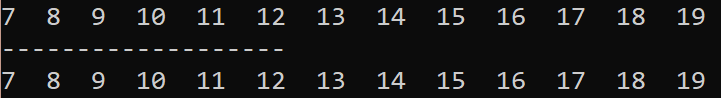


* масив випадкових чисел

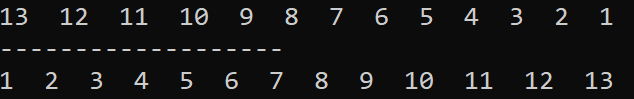


1. **Методом Шелла:**

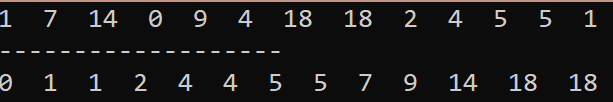
* упорядкований масив



* обернено-впорядкований масив



* масив випадкових чисел



**Тести програми**

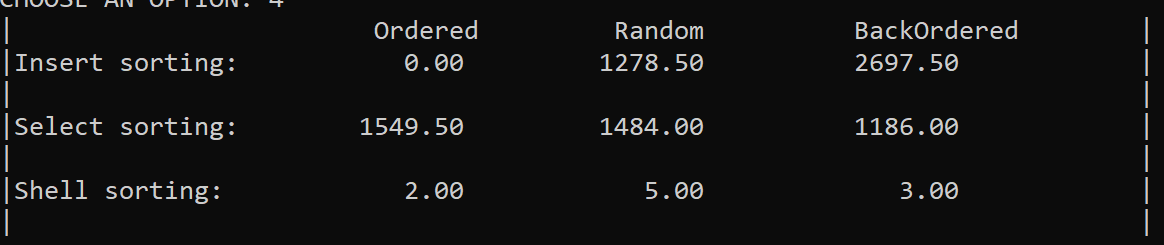
*\*на наданих нижче діаграмах колонки розташовуються безпосередньо під назвою відсортованого масиву або вектору;*

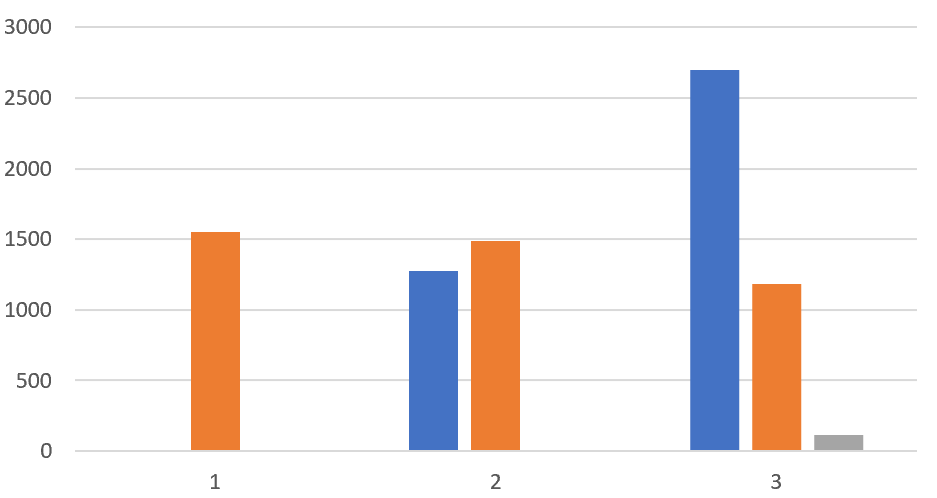
*оранжевий стовпець – сортування методом вибору;*

*синій стовпець – сортування методом вставки;*

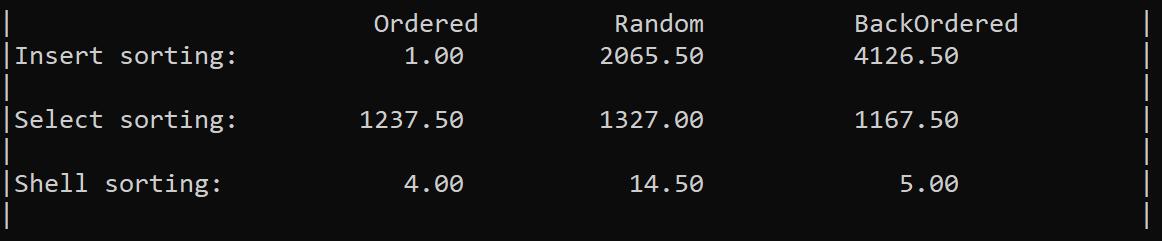
*сірий – сортування методом Шелла;*

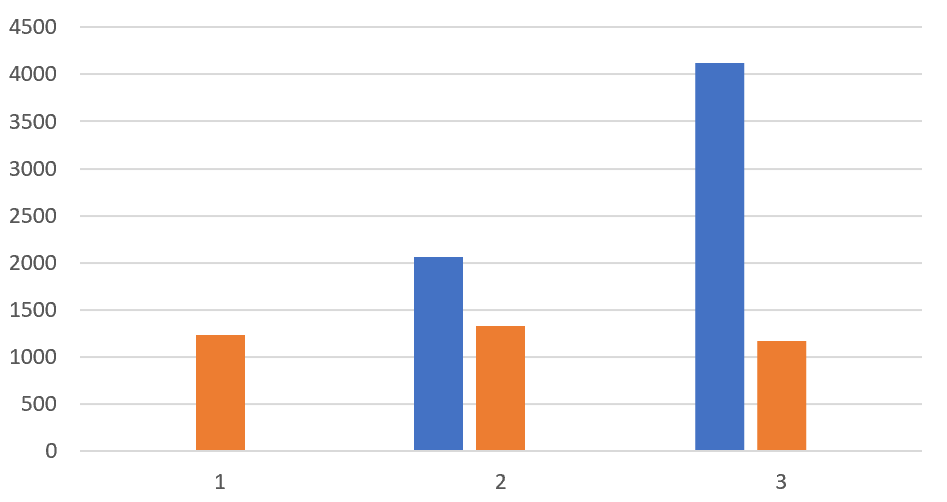
Vector [50000]



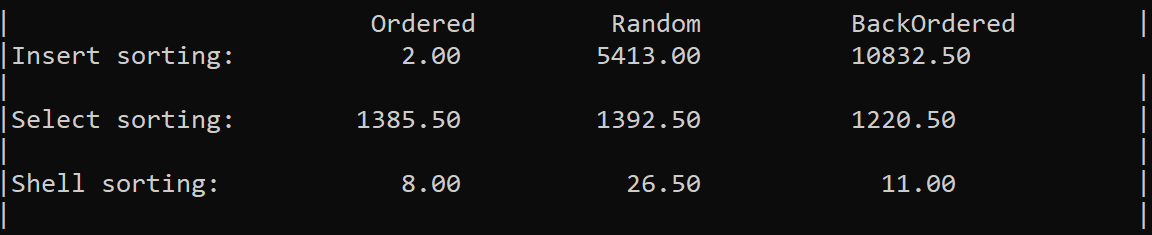


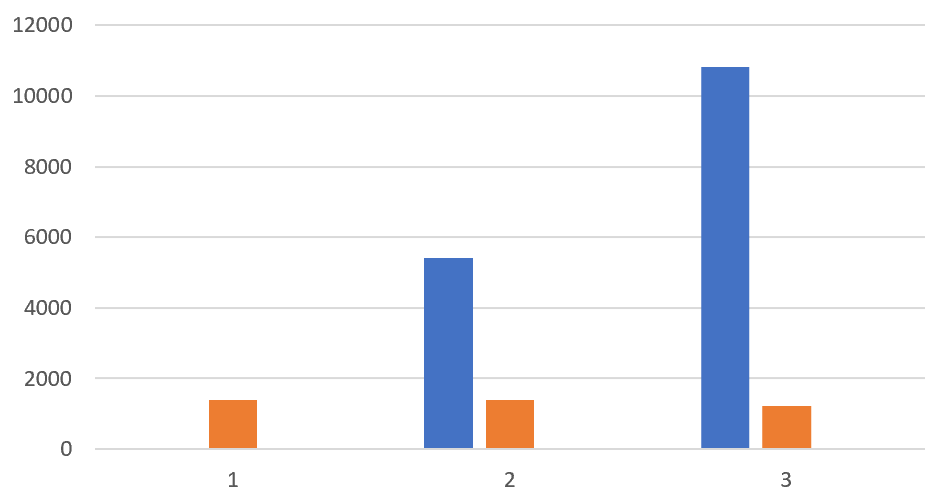
A[50000][1][1]



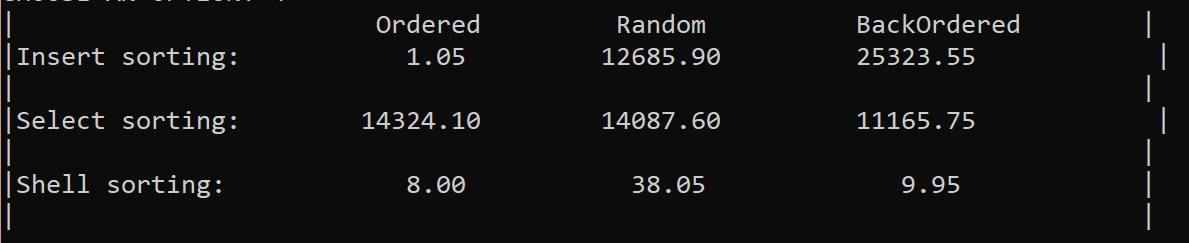


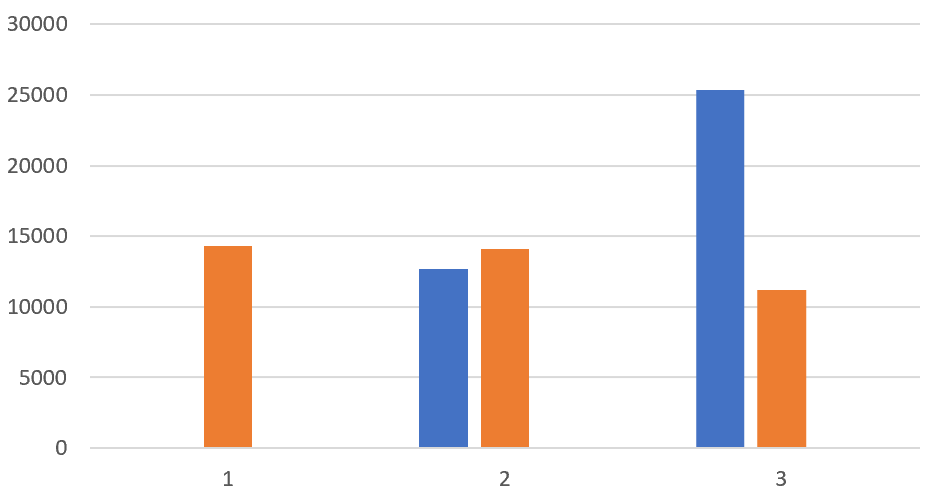
A[50000][2][2]



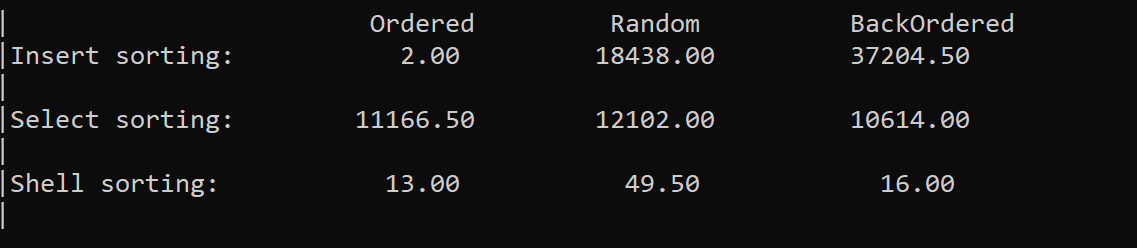


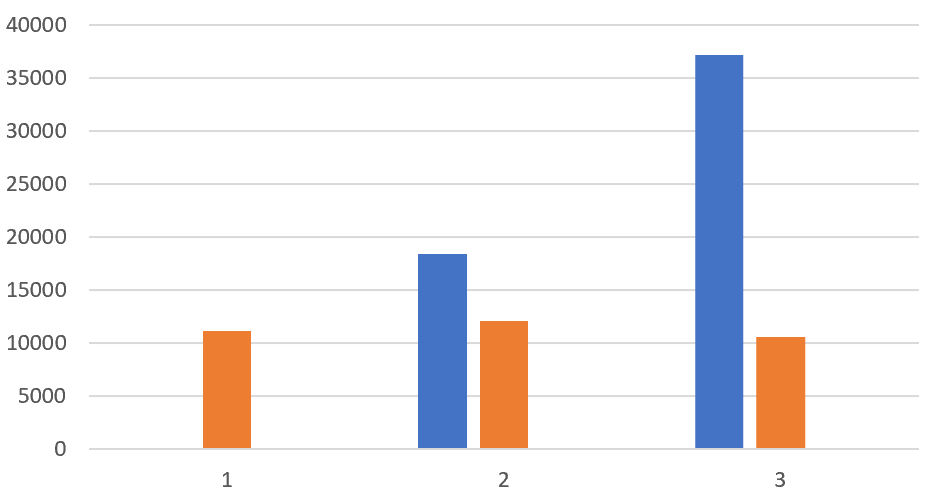
Vector [150000]

****

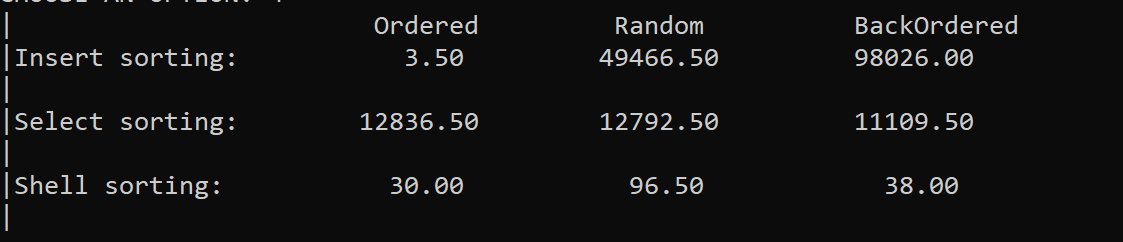


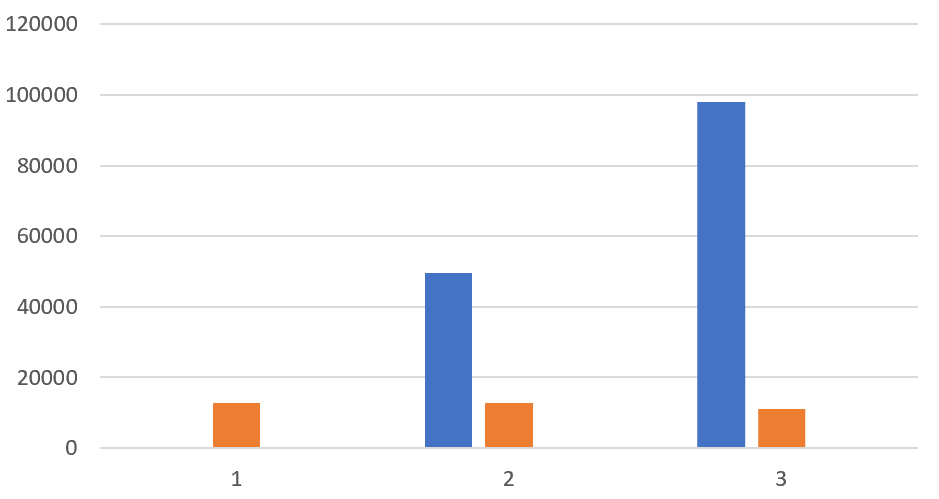
A[150000][1][1]



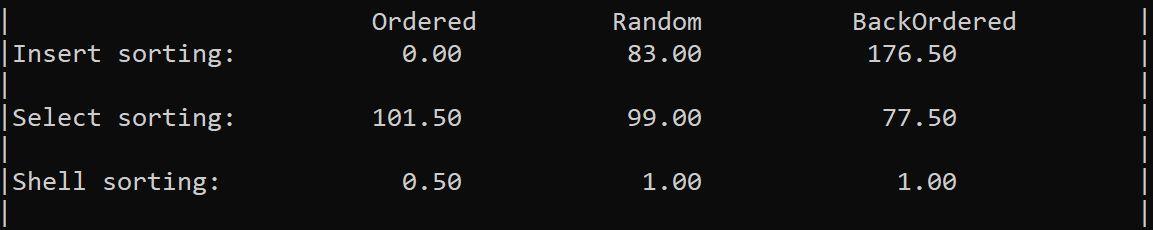


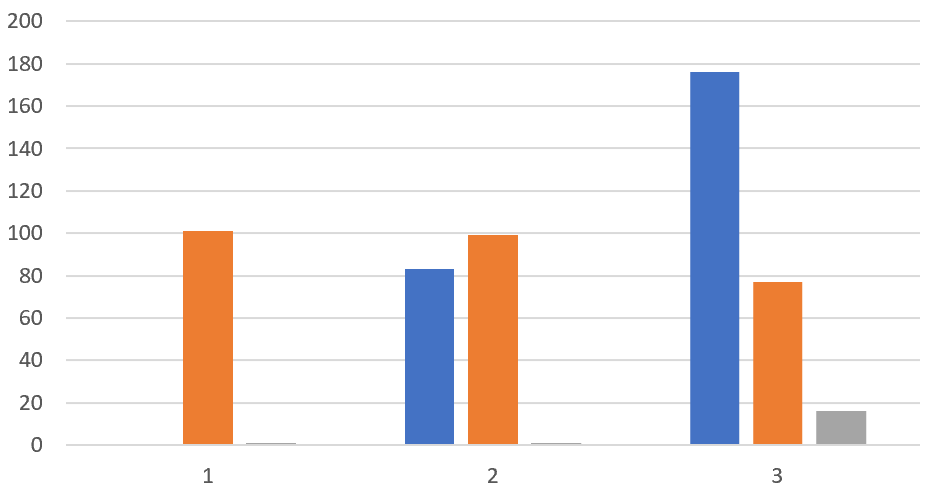
A[150000][2][2]



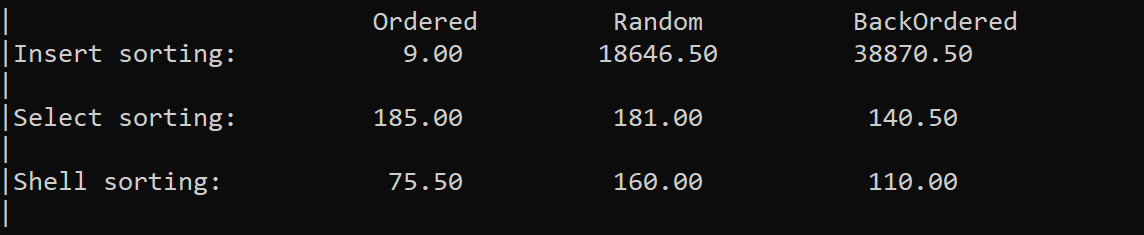
****

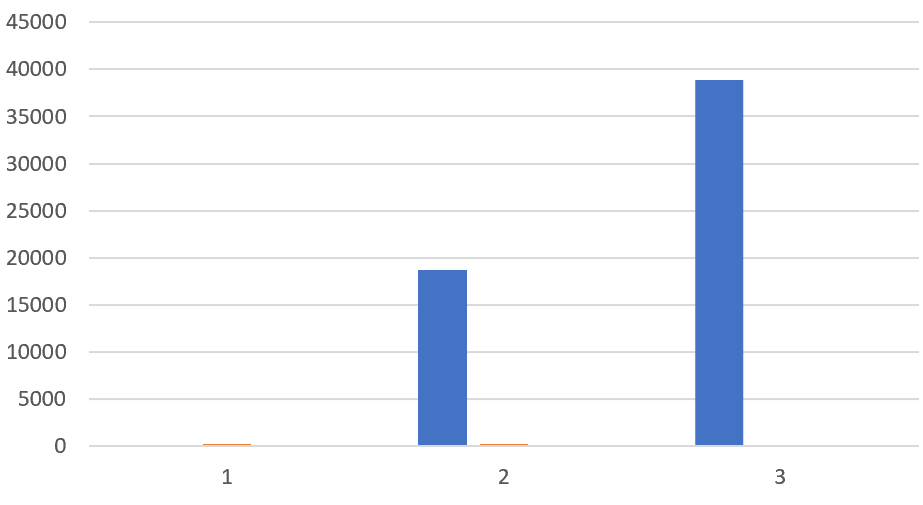
Vector[12800]

****

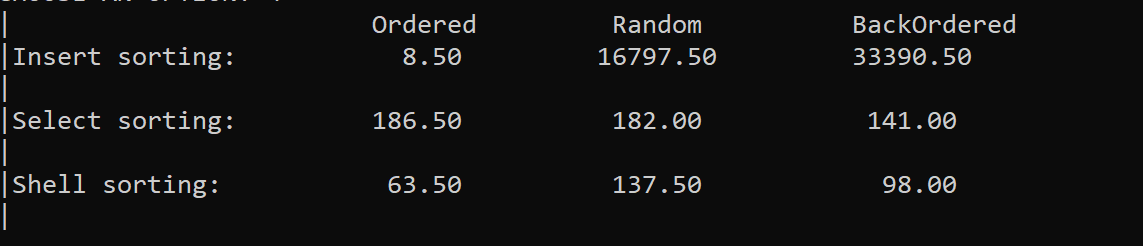
****

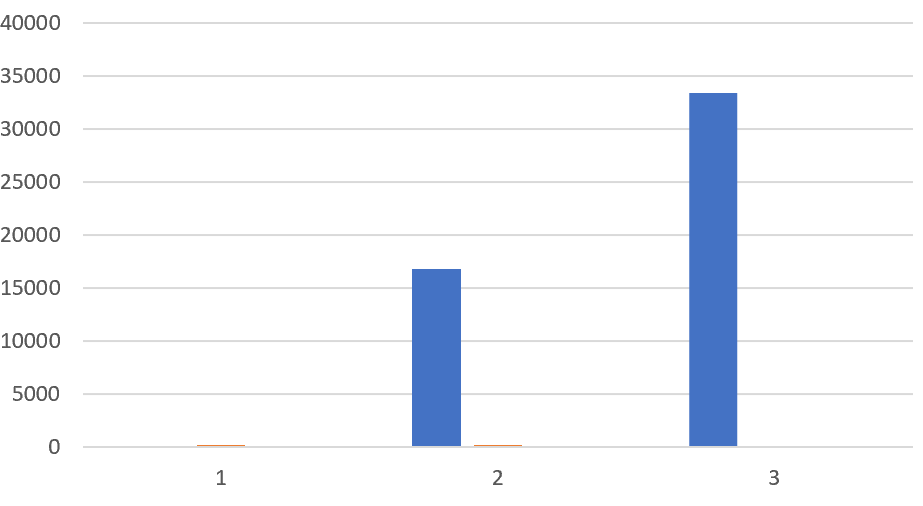
A[12800][2][80]

****

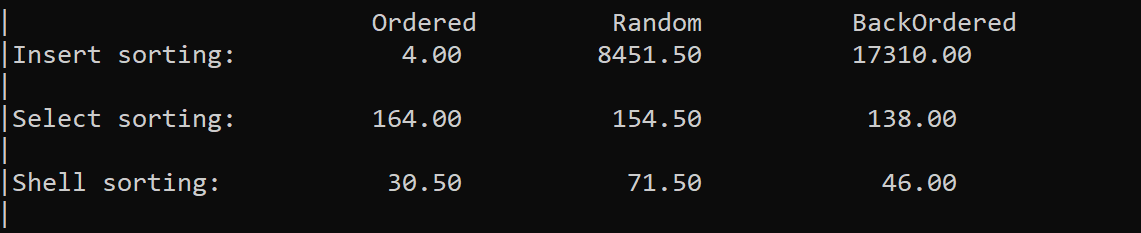
****

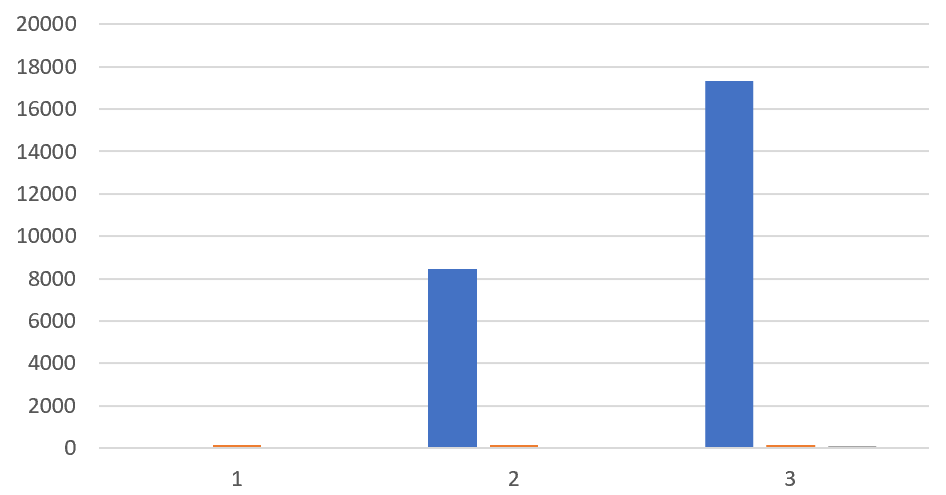
А[12800][80][2]



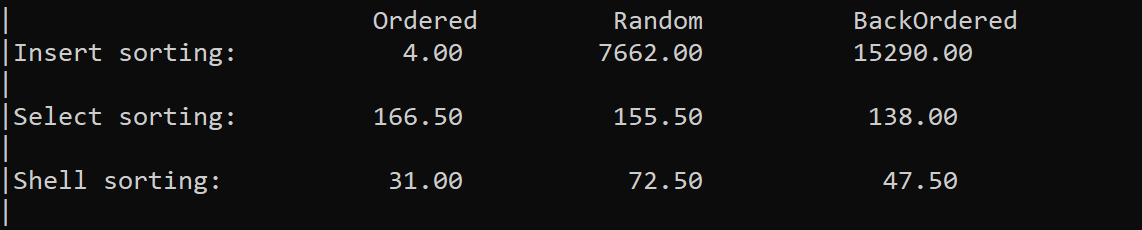


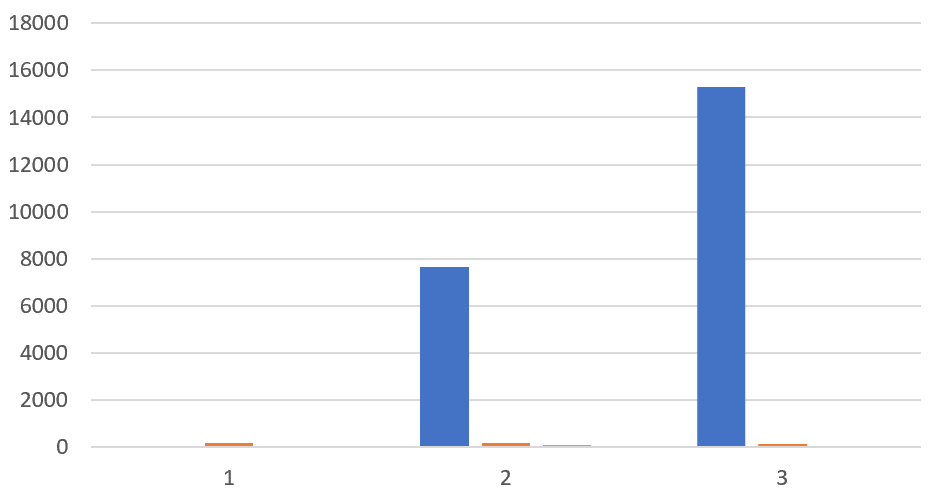
А[12800][4][20]





А[12800][20][4]



****

**Порівняльний аналіз отриманих результатів**

У ході курсової роботи було порівняно 3 алгоритми сортування на прикладі вектору та тримірного масиву. Були задіяні наступні алгоритми: метод прямої вставки з лінійним пошуком місця вставки, метод прямого вибору №4 та сортування Шелла. Порівняння виконувалось шляхом перестановки перерізів масива за незменшенням значень вектору перших елементів кожного перерізу А [\*,1,1]. Під час досліджень було виявлено:

1. сортування тривимірного масиву відбувається значно повільніше вектору;
2. кожен алгоритм має свої переваги при роботі з різними видами масивів та векторів.

**Метод прямої вставки з лінійним пошуком місця вставки**

Сортування впорядкованого масиву показало найкращі результати, адже принцип даного алгоритму полягає у тому, що ми шукаємо місце вставки зліва-направо та виконуємо зсув елементів для звільнення місця вставки, а оскільки масив впорядкований, то зсув не відбувається. Тому навіть змінюючи розміри масиву, суттєвої різниці не було виявлено. З іншого боку, сортування обернено-впорядкованого масиву показало найгірші результати саме через те, що пошук місця вставки йде зліва-направо і відбувається зсув усіх елементів для звільнення позиції. Для масиву, що заповнений випадково цей алгоритм поступається усім іншим алгоритмам, що були представлені у даній курсовій роботі. Як для невпорядкованого так і для обернено-впорядкованого швидкість сортування зменшувалася зі збільшенням N.

**Метод прямого вибору № 4**

На відміну від методу прямої вставки з лінійним пошуком місця вставки, для всіх видів масиву ми отримали приблизно однакові результати. Більше того сортування обернено-впорядкованого масиву відбулося швидше ніж сортування впорядкованого масиву. Це можна пояснити тим, що кількість порівнянь буде більшою при впорядкованому масиві ніж при обернено-впорядкованому, отже збільшується кількість операцій та час сортування масиву.

**Сортування Шелла**Якщо враховувати загальну схему роботи даного алгоритму можна сказати, що з одного боку, чим більше етапів, тим більше операцій порівняння. У той же час, з кожним етапом буде переставлятися більша кількість елементів і відповідно кількість перестановок буде менша. Тому треба шукати оптимальну кількість етапів сортування і оптимальні відстані між елементами на кожному етапі. Найкращим випадком для сортування Шелла є впорядкований масив, але не відстає від нього і обернено-впорядкований масив, адже оскільки сортування Шелла виконується по групах поетапно, то не має значення як впорядкований масив і лише при зовсім випадковому масиві сортування виконується значно повільніше, адже відбуватиметься багато порівнянь і перестановок в групах елементів. Згідно з тестів програми сортування Шелла – дуже нестабільний алгоритм, це пояснюється логарифмічною складовою вибору.

**Висновки**

У ході дослідження та аналізу були отримані наступні результати:

1. Час сортування тривимірного масиву заданою задачею не залежить як і від розміру перерізів масиву, так і від їх форми. Причиною цього є те, що з вектором перших елементів масиву перебираються також дані кожного перерізу масиву. Проте, ключову роль відграє саме розмір перерізів масиву.

-найліпше це твердження доводить порівняння результатів вимірів тримірного масиву з вектором.

1. Метод сортування з лінійним пошуком місця вставки є найгіршим вибором з перелічених вище алгоритмів для сортування обернено-впорядкованого, або заповненого випадковими числами тривимірного масиву. Рекомендується для використанні при сортуванні впорядкованого масиву.
2. Метод прямого вибору забезпечить стабільні результати для будь-якого масиву.
3. Сортування Шелла демонструє швидкий відносно інших алгоритмів результат, але має доволі нестабільні показники при сортуванні впорядкованого, обернено-впорядкованого або заповненого випадковими числами тривимірного масиву.

Отже, як було зазначено раніше, кожен алгоритм має свої сильні та слабкі сторони. Але я вважаю найуніверсальнішим методом сортування алгоритм Шелла, адже він показав найкращі результати у вимірах часу.

* Він є на 34% ефективнішим за метод вибору в роботі з впорядкованим масивом.
* На 110% ефективніший за метод вставки в роботі з масивом заповненим випадковими числами.

**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.
4. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2, Issue 10, October 2013

http://ijarcet.org/wp-content/uploads/IJARCET-VOL-2-ISSUE-10-2720-2724.pdf