

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: Ткач Д.

Група: КВ-41

Номер залікової книжки:

Допущений до захисту

---

2 семестр 2024/2025

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
УКРАЇНИ «КПІ ім. Ігоря Сікорського»  
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих комп'ютерних систем**

Узгоджено

ЗАХИЩЕНА " \_\_ " \_\_\_\_ 20 \_\_ р.

Керівник роботи

з оцінкою \_\_\_\_\_

\_\_\_\_\_/Марченко О.І./

\_\_\_\_\_/Марченко О.І./

*Дослідження ефективності методів сортування  
(Алгоритм сортування №1 методу прямого вибору, Алгоритм  
сортування №8 методу прямого вибору, Алгоритм №2 методу  
сортування Шелла (спосіб реалізації на основі гібридного алгоритму  
«вставка-обмін»))  
на багатовимірних масивах*

Виконавець роботи: \_\_\_\_\_  
(підпис)

Прізвище Ім'я По батькові  
\_\_\_\_\_ 20 \_\_\_\_ р.

**ТЕХНІЧНЕ ЗАВДАННЯ**  
*на курсову роботу з дисципліни*  
**“Структури даних і алгоритми”**

**ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

**I.** Описати теоретичні положення, від яких відштовхується дослідження, тобто принцип та схему роботи кожного із досліджуваних алгоритмів сортування для одновимірного масиву, навести загальновідомі властивості цих алгоритмів та оцінки кількості операцій порівняння та присвоєння для них.

**II.** Скласти алгоритми рішення задачі сортування в багатовимірному 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 масиву) для масиву з заданими геометричними розмірами повинна бути такою:

Таблиця №    для масиву  $A[P,M,N]$ , де  $P =$  ;  $M =$  ;  $N =$  ;

	Впорядкований	Невпорядкований	Обернено впорядкований
Назва алгоритму 1			
Назва алгоритму 2			
Назва алгоритму 3			

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

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

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

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

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

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

## **Варіант №119**

### **Задача № 4**

Впорядкувати тривимірний масив  $\text{Arr3D}[P,M,N]$  таким чином: переставити перерізи масиву за незменшенням значень вектору перших елементів кожного перерізу  $A[* ,1,1]$ .

#### **Досліджувані методи та алгоритми**

1. Алгоритм сортування №1 методу прямого вибору (додаток 1, рис. №5);
2. Алгоритм сортування №8 методу прямого вибору (додаток 1, рис. №12);
3. Алгоритм №2 методу сортування Шелла (спосіб реалізації на основі гібридного алгоритму «вставка-обмін») (додаток 1, рис. 25 № 23 ).

#### **Спосіб обходу № 7**

Використовуючи значення вектору перших елементів кожного перерізу  $\text{Arr3D}[*,1,1]$  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**

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

Випадок дослідження II. Залежність часу роботи алгоритмів від форми перерізів масиву.

## **Опис теоретичних положень**

### ***Алгоритм сортування №1 методу прямого вибору (додаток 1, рис. №5)***

Принцип роботи алгоритму:

У кожен момент часу масив вважається поділеним на дві частини: вже відсортовану частину та ще не відсортовану частину. Проте перед початком сортування весь масив вважається не відсортованим.

В ході роботи алгоритму:

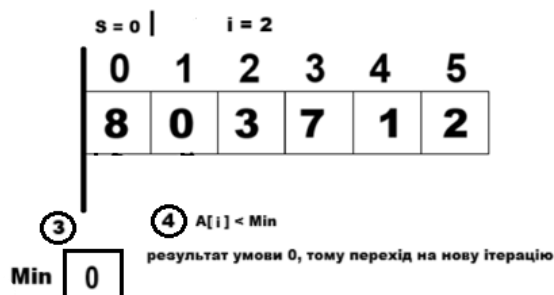
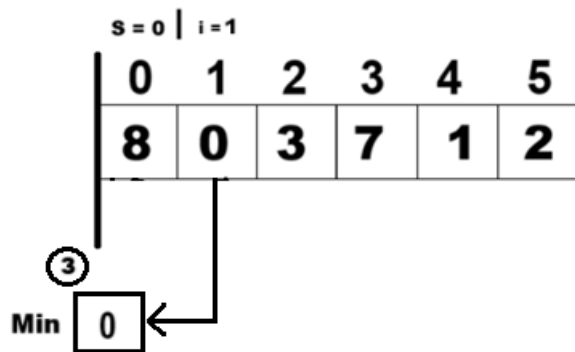
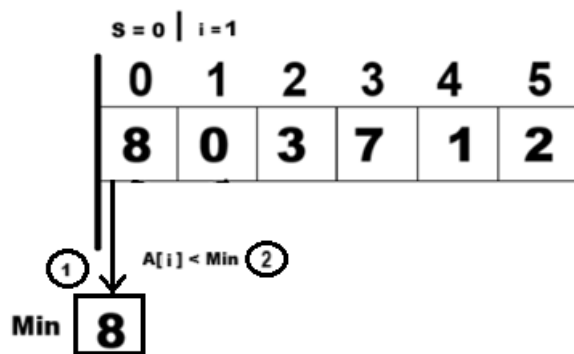
- 1) З усіх елементів масиву у діапазоні індексів позицій від 0 до  $n-1$ , де  $n$  – кількість елементів масиву, вибирається найменший, та у спеціально виділені комірки пам'яті зберігається його значення і позиція (тобто індекс позиції).
- 2) Потім це мінімальне значення міняється місцями з першим елементом масиву (елемента з індексом 0). В результаті виконання цього кроку один елемент масиву вже є відсортованим.
- 3) Кроки 1) і 2) повторюються для ще не відсортованої частини масиву, тобто для частини масиву з індексним діапазоном позицій елементів від 1 до  $n-1$ , потім від 2 до  $n-1$ , потім від 3 до  $n-1$  і т. д., послідовно міняючи черговий мінімальний елемент місцями з елементами відповідно з індексом 1, 2, 3 і т. д. Тобто кількість відсортованих елементів масиву поступово зростає. Останній раціональний індексний діапазон пошуку мінімуму – від  $n-2$  до  $n-1$ .

Схема роботи алгоритму для одновимірного масиву  $A$  на  $n = 6$  елементів:

Початковий стан:

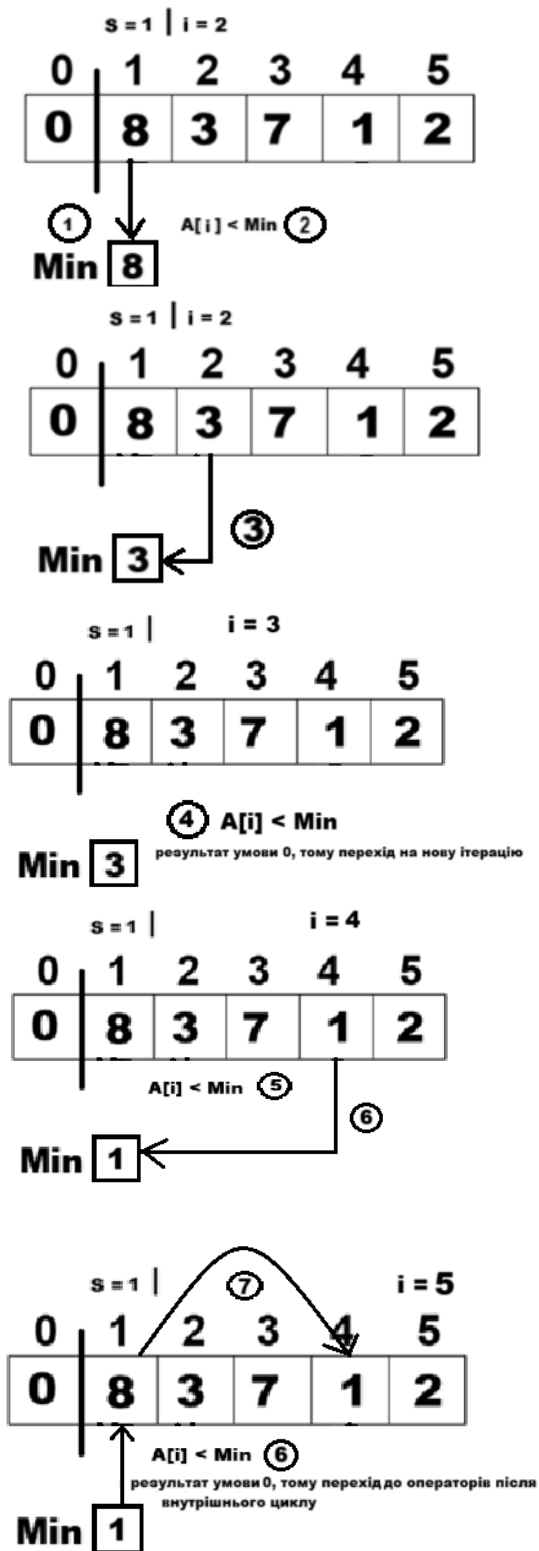
0	1	2	3	4	5
8	0	3	7	1	2

Прохід №1:

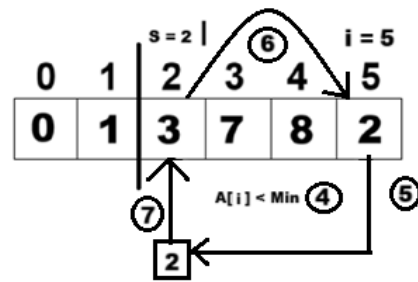
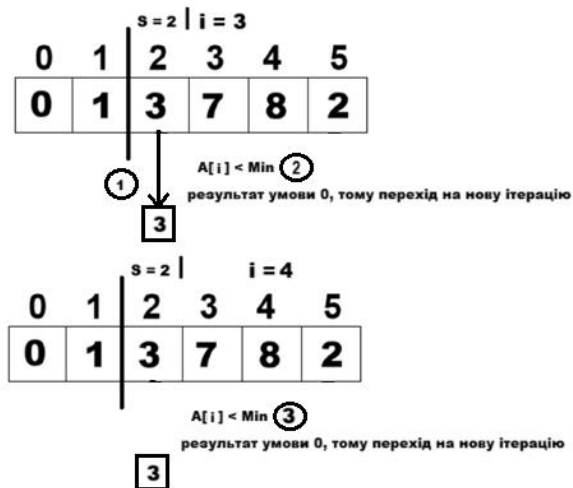




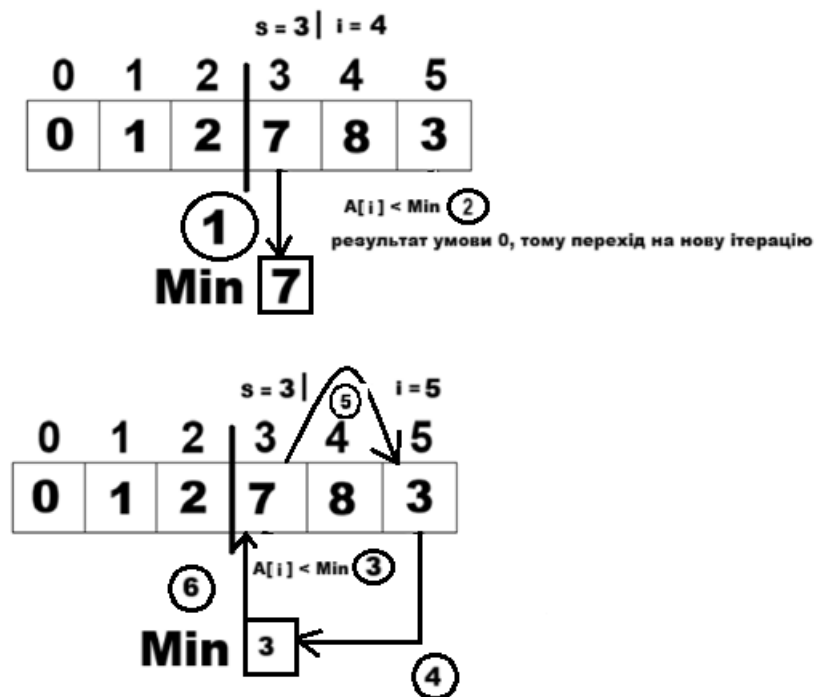
Прохід №2:



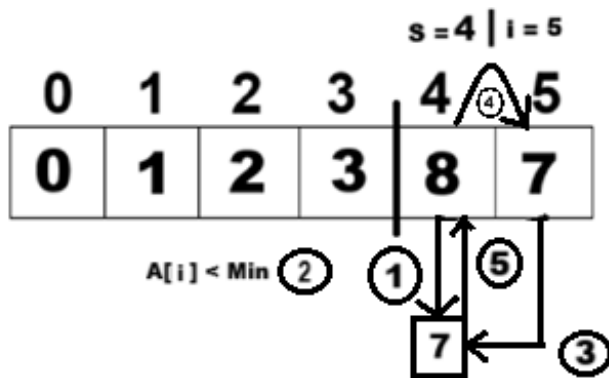
Прохід №3:



Прохід № 4:



Прохід №5:



Результат:

0	1	2	3	4	5
0	1	2	3	7	8

Властивості алгоритму:

Незалежно від того чи початковий стан заданого масиву є відсортованим чи ні, алгоритм завжди буде виконуватися. Повністю відсортований масив, незалежно від того як розташовані елементи масиву, буде отриманий на  $n-1$  проході по ньому (де  $n$  – кількість елементів масиву). Проте такий алгоритм є простим за своїм принципом роботи і структурою. Також кількість присвоєнь у методі сортування прямим вибором – єдина характеристика яка не є квадратичною, кількість порівнянь становить:  $C = (n^2 - n)/2$ .

Кількість операцій порівняння  $C = O(n^2)$ ;

Кількість операцій присвоєння  $R = O(n \times \ln(n))$ ;

Алгоритм сортування №1 методу прямого вибору (за незменшенням) на мові C:

```
clock_t Select1(int *A, int N)
{
    int Min, imin;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int s=0; s<N-1; s++)
    {
        Min=A[s];
        imin=s;
        for(int i=s+1; i<N; i++)
        {
            if (A[i]<Min)
            {
                Min=A[i];
                imin=i;
            }
        }
        A[imin]=A[s];
        A[s]=Min;
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

***Алгоритм сортування №8 методу прямого вибору (додаток 1, рис. №12);***

Принцип роботи алгоритму:

У кожен момент часу масив вважається поділеним на дві частини: вже відсортовану частину та ще не відсортовану частину. Проте перед початком сортування весь масив вважається не відсортованим.

В ході роботи алгоритму:

- 1) Спочатку відбувається ініціалізація змінних, які зберігатимуть поточне значення лівої і правої границь проходу по масиву.
- 2) Далі відбувається пошук індексів відповідно мінімального і максимального елементів заданого масиву.
- 3) Потім відбувається перевірка на правильність позиції поточного мінімального елемента, тобто на місці поточної лівої границі проходу по масиву. Якщо це не так то, елемент який знаходиться на місці поточної лівої границі проходу по масиву, міняється місцями з поточним мінімальним елементом.
- 4) Далі відбувається аналогічна перевірка на правильність позиції для поточного максимального елемента, тобто на місці поточної правої границі проходу по масиву. Якщо це не так, перевіряється чи цей елемент знаходиться на місці поточної лівої границі проходу по масиву, якщо це так то поточний елемент міняється місцями з елементом поточної правої границі проходу по масиву. Проте якщо поточний максимальний елемент не знаходиться на місці поточної правої границі і не знаходиться на місці поточної лівої границі то, цей елемент міняється місцями з елементом який знаходиться на місці поточної правої границі.
- 5) На 5 кроці роботи алгоритму відбувається зміщення поточної лівої і правої границь проходу по масиву на один елемент уперед і назад відповідно.

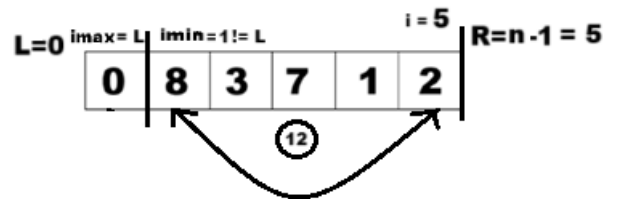
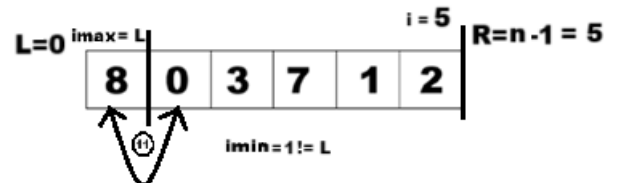
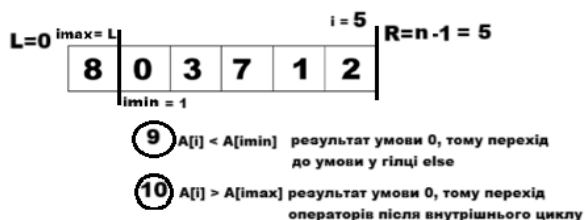
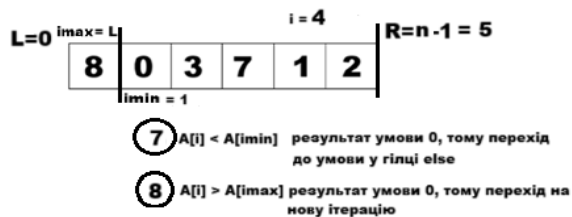
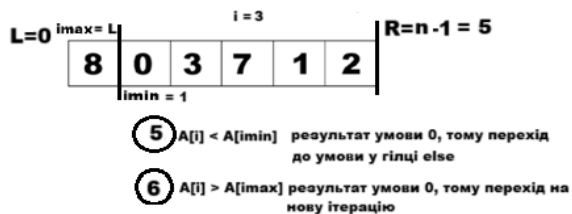
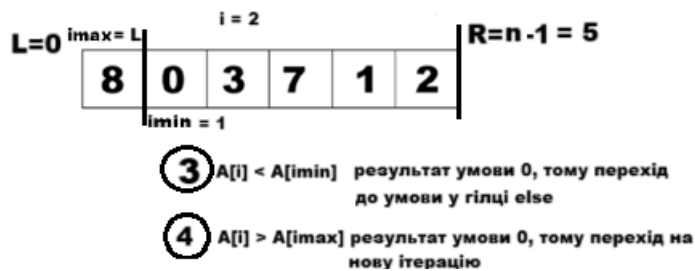
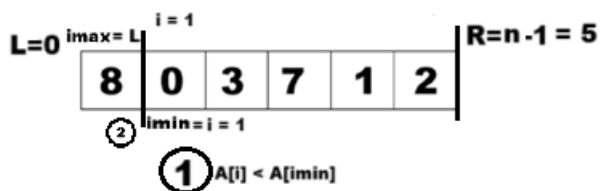
6) Кроки 2), 3), 4), 5) повторюються доки ліва границя менша за праву.

Схема роботи алгоритму для одновимірного масиву  $A$  на  $n = 6$  елементів:

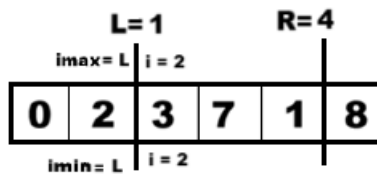
Початковий стан:

0	1	2	3	4	5
8	0	3	7	1	2

Прохід №1:

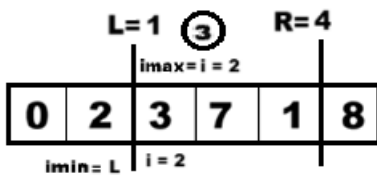


Прохід №2:



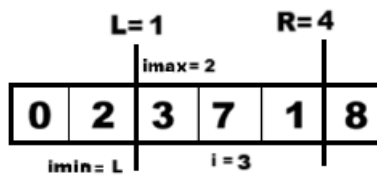
①  $A[i] < A[imin]$  результат умови 0, тому перехід до умови у гілці else

②  $A[i] > A[imax]$



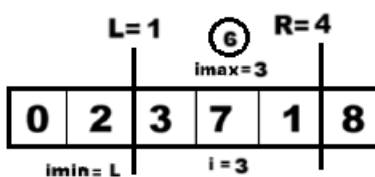
①  $A[i] < A[imin]$  результат умови 0, тому перехід до умови у гілці else

②  $A[i] > A[imax]$



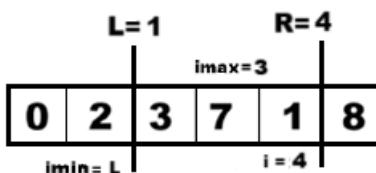
④  $A[i] < A[imin]$  результат умови 0, тому перехід до умови у гілці else

⑤  $A[i] > A[imax]$

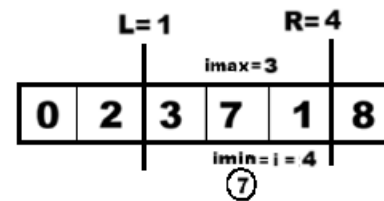


④  $A[i] < A[imin]$  результат умови 0, тому перехід до умови у гілці else

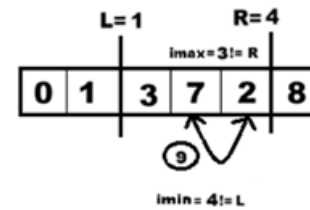
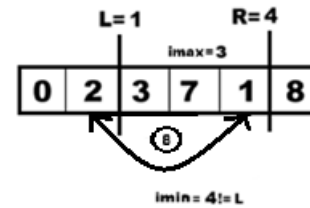
⑤  $A[i] > A[imax]$



⑥  $A[i] < A[imin]$



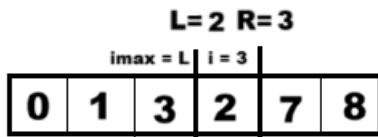
⑥  $A[i] < A[imin]$



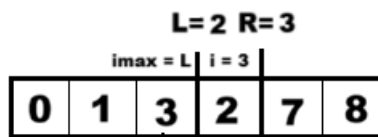
Прохід №3:



①  $A[i] < A[imin]$



② imin = i = 3  
①  $A[i] < A[imin]$



③ imin = i = 3 != L

Результат:

0	1	2	3	4	5
---	---	---	---	---	---

0	1	2	3	7	8
---	---	---	---	---	---

Властивості алгоритму:

Незалежно від початкового стану масиву, алгоритм завжди виконується однаково кількість разів. Навіть якщо масив повністю відсортований, алгоритм все одно проходить по ньому, здійснюючи пошук мінімального та максимального елементів. Повністю впорядкований масив буде отриманий на  $(n/2)$  проході (де  $n$  – кількість елементів масиву). Також оскільки на кожній ітерації встановлюються одразу два елементи на свої місця то він є ефективнішим за звичайний метод сортування прямим вибором.



Алгоритм є простим за своєю структурою та не потребує додаткової пам'яті, оскільки сортування відбувається "на місці". Проте, кількість порівнянь, як і у класичному методі прямого вибору, є квадратичною і становить  $C = (n^2 - n)/2$ . Кількість присвоєнь менша, ніж у стандартному алгоритмі вибору, оскільки за один прохід встановлюються одразу два елементи.

Кількість порівнянь:  $C = O(n^2)$ , де  $n$  – кількість елементів масиву;

Кількість присвоєнь:  $R = O(n)$ , де  $n$  – кількість елементів масиву.

Алгоритм сортування (за незменшенням) №8 методу прямого вибору на мові C:

```
clock_t Select8(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;
        if (imin!=L)
        {
            tmp=A[imin];
            A[imin]=A[L];
            A[L]=tmp;
        }
        if (imax!=R)
            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;
}
```

***Алгоритм №2 методу сортування Шелла (спосіб реалізації на основі гібридного алгоритму «вставка-обмін») (додаток 1, рис. 25 № 23 )***

Принцип роботи:

В кожен момент часу масив вважається поділеним на частково відсортовані групи елементів, які розташовані на певній відстані що поступово зменшується до 1. Сортування масиву цим методом складається з кількох етапів, кількість яких напряму залежить від кількості елементів масиву що сортується.

Для визначення кількості етапів та відповідно оптимальних відстаней між елементами однієї групи у даному алгоритмі використовується формула Хіббарда:

$k = 2^p - 1$ , (1, 3, 7, 15, 31 ... у зворотньому порядку); де  $k$  – поточна відстань між елементами однієї групи.

Відповідно кількість етапів буде:

$T = \lceil \log_2 n \rceil - 1$ ; де  $n$  – кількість елементів масиву що сортується,  $T$  – кількість етапів сортування з різними відстанями.

У загальному випадку не можливо сказати із скількох етапів складатиметься сортування масиву, проте для прикладу, можна припустити що він складається з 4 – етапів. Відстань між елементами однієї групи на останньому, в загальному випадку, етапі сортування масиву становить 1.

В ході роботи алгоритму:

Етап 1:

Спочатку у спеціально виділену комірку пам'яті  $k$ , зберігається значення поточної відстані між елементами однієї групи. Після чого методом сортування обміном – вставкою сортуються елементи, які розташовані на відстані  $k$  позицій, при чому усі інші елементи не сортуються, тобто елементи однієї групи сортуються окремо. Таких груп елементів буде  $k$ .

Етап 2:

Поточна відстань між елементами змінюється на меншу (у змінну  $k$  записується значення поточної, меншої відстані між елементами однієї групи), відповідно і змінюється кількість груп та елементи які входять в ці групи (перегрупування елементів масиву). Після чого методом обміну – вставки сортуються елементи, які розташовані на відстані  $k$  позицій, при чому усі інші елементи не сортуються, тобто елементи однієї групи сортуються окремо. Таких груп елементів вже буде  $k$ .

Етап 3:

Поточна відстань між елементами змінюється на меншу (у змінну  $k$  записується значення поточної, меншої відстані між елементами однієї групи), відповідно і змінюється кількість груп та елементи які входять в ці групи (перегрупування елементів масиву). Після чого методом обміну – вставки сортуються елементи, які розташовані на відстані  $k$  позицій, при чому усі інші елементи не сортуються, тобто елементи однієї групи сортуються окремо. Таких груп елементів вже буде  $k$ .

Етап 4:

На даному етапі відбуватиметься звичайне сортування елементів методом обміну – вставки, оскільки відстань між елементами становить 1 позицію.

Сортування таким методом також складається з кількох етапів:

У загальному випадку, тобто коли відстань між елементами становить  $k$  позицій, спочатку відбувається порівняння елемента з індексом  $j = k$ , і елемента з індексом  $j - k$ . Якщо елемент з індексом  $j$  є меншим ніж елемент з  $j - k$ , то вони міняються місцями (тобто відбувається обмін та вставка меншого елемента на лівішу позицію). Після чого змінна лічильник  $j$  зменшується на  $k$ , і відбувається перехід на нову ітерацію.

Усі ці кроки повторюються доки елементи по даним групам не будуть відсортованими.

Схема роботи алгоритму для одновимірного масиву  $A$  на  $n = 10$  елементів:

Для масиву з такою кількістю елементів,  $T = 2$ ;

Початковий стан масиву  $A$ :

0	1	2	3	4	5	6	7	8	9
8	0	3	7	1	2	4	9	1	5

На першій ітерації масив є умовно поділеним на  $k = 3$  частини:

0	1	2	3	4	5	6	7	8	9
8	0	3	7	1	2	4	9	1	5

Прохід № 1 :

0	1	2	3	4	5	6	7	8	9
8	0	3	7	1	2	4	9	1	5

①  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 1, тому відбувається обмін

Прохід № 2 :

0	1	2	3	4	5	6	7	8	9
7	0	3	8	1	2	4	9	1	5

①  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 0 , тому відбувається перехід на нову ітерацію

Прохід № 3 :

0	1	2	3	4	5	6	7	8	9
7	0	3	8	1	2	4	9	1	5

①  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 1, тому відбувається обмін

Прохід № 4:

0	1	2	3	4	5	6	7	8	9
7	0	2	8	1	3	4	9	1	5

① ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	7	8	9
7	0	2	4	1	3	8	9	1	5

③ ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	7	8	9
4	0	2	7	1	3	8	9	1	5

⑤ ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 0, тому відбувається перехід на нову ітерацію

Прохід № 5 :

0	1	2	3	4	5	6	7	8	9
4	0	2	7	1	3	8	9	1	5

① ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 0, тому відбувається перехід на нову ітерацію

Прохід № 6

0	1	2	3	4	5	6	7	8	9
4	0	2	7	1	3	8	9	1	5

① ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	7	8	9
4	0	2	7	1	1	8	9	3	5

③ ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	7	8	9
4	0	1	7	1	2	8	9	3	5

⑤ ( $j \geq k$  &  $A[j] < A[j-k]$ ) //результат умови 0, тому відбувається перехід на нову ітерацію

Прохід № 7 :

0	1	2	3	4	5	6	7	8	9 <sup>j=9</sup>
4	0	1	7	1	2	8	9	3	5

① ( $j \geq k \ \&\& \ A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6 <sup>j=6</sup>	7	8	9
4	0	1	7	1	2	5	9	3	8

③ ( $j \geq k \ \&\& \ A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

0	1	2	3 <sup>j=3</sup>	4	5	6	7	8	9
4	0	1	5	1	2	7	9	3	8

⑤ ( $j \geq k \ \&\& \ A[j] < A[j-k]$ ) //результат умови 0, тому відбувається перехід на  
// нову ітерацію

На наступному проході відбувається перегрупування елементів, тобто тепер відстань між елементами однієї групи тепер становить  $k = 1$  позицію, тобто цей прохід фактично являє собою звичайне сортування методом обміну – вставки.

Масив А з перегрупованими ( $k = 1$ ) елементами:

0	1	2	3	4	5	6	7	8	9
4	0	1	5	1	2	7	9	3	8

Прохід № 8 :

0	1 <sup>j=1</sup>	2	3	4	5	6	7	8	9
4	0	1	5	1	2	7	9	3	8

① ( $j \geq k \ \&\& \ A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

j = 0

0	1	2	3	4	5	6	7	8	9
0	4	1	5	1	2	7	9	3	8

③ (j >= k && A[j] < A[j-k]) //результат умови 0 , тому відбувається перехід на  
// нову ітерацію

Прохід № 9 :

j = 2

0	1	2	3	4	5	6	7	8	9
0	4	1	5	1	2	7	9	3	8



① (j >= k && A[j] < A[j-k]) //результат умови 1, тому відбувається обмін

j = 1

0	1	2	3	4	5	6	7	8	9
0	1	4	5	1	2	7	9	3	8

③ (j >= k && A[j] < A[j-k]) //результат умови 0 , тому відбувається перехід на  
// нову ітерацію

j = 3

0	1	2	3	4	5	6	7	8	9
0	1	4	5	1	2	7	9	3	8

① (j >= k && A[j] < A[j-k]) //результат умови 0 , тому відбувається перехід на  
// нову ітерацію

Прохід № 10 :

j = 4

0	1	2	3	4	5	6	7	8	9
0	1	4	5	1	2	7	9	3	8



① (j >= k && A[j] < A[j-k]) //результат умови 1, тому відбувається обмін

j = 3

0	1	2	3	4	5	6	7	8	9
0	1	4	1	5	2	7	9	3	8



③ (j >= k && A[j] < A[j-k]) //результат умови 1, тому відбувається обмін

$$j = 2$$


0	1	2	3	4	5	6	7	8	9
0	1	1	4	5	2	7	9	3	8

⑤ ( $j \geq k$  &&  $A[j] < A[j-k]$ ) //результат умови 0 , тому відбувається перехід на  
// нову ітерацію

Прохід № 11 :

$$j = 5$$


0	1	2	3	4	5	6	7	8	9
0	1	1	4	5	2	7	9	3	8



① ( $j \geq k$  &&  $A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

$$j = 4$$

0	1	2	3	4	5	6	7	8	9
0	1	1	4	2	5	7	9	3	8



③ ( $j \geq k$  &&  $A[j] < A[j-k]$ ) //результат умови 1, тому відбувається обмін

$$j = 3$$

0	1	2	3	4	5	6	7	8	9
0	1	1	2	4	5	7	9	3	8

⑤ ( $j \geq k$  &&  $A[j] < A[j-k]$ ) //результат умови 0 , тому відбувається перехід на  
// нову ітерацію

Прохід № 12 :

$$j = 6$$

0	1	2	3	4	5	6	7	8	9
0	1	1	2	4	5	7	9	3	8

① ( $j \geq k$  &&  $A[j] < A[j-k]$ ) //результат умови 0 , тому відбувається перехід на  
// нову ітерацію



Прохід № 13 :

0	1	2	3	4	5	6	$j=7$	8	9
0	1	1	2	4	5	7	9	3	8

①  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 0, тому відбувається перехід на  
// нову ітерацію

Прохід № 14 :

0	1	2	3	4	5	6	7	$j=8$	9
0	1	1	2	4	5	7	9	3	8

①  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	$j=7$	8	9
0	1	1	2	4	5	7	3	9	8

③  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	$j=6$	7	8	9
0	1	1	2	4	5	3	7	9	8	

⑤  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 1, тому відбувається обмін

0	1	2	3	4	5	6	7	8	9
0	1	1	2	4	3	5	7	9	8

⑦  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 1, тому відбувається обмін

0	1	2	3	$j=4$	5	6	7	8	9
0	1	1	2	3	4	5	7	9	8

⑨  $(j \geq k \ \&\& \ A[j] < A[j-k])$  //результат умови 0, тому відбувається перехід на  
// нову ітерацію

Прохід № 15 :



Результат:

0	1	2	3	4	5	6	7	8	9
0	1	1	2	3	4	5	7	8	9

Властивості алгоритму:

Сортуючи масиви даним алгоритмом, можна швидко шукати та переставляти (тобто за незменшенням у даному прикладі) елементи, що знаходяться на порівняно великій відстані один від одного, що забезпечує часткову відсортованість яка збільшується з виконанням кожного етапу.

Одним із найбільш раціональних способів використання цього алгоритму є сортування ним масивів великих розмірів, що значно зменшить час виконання сортування.

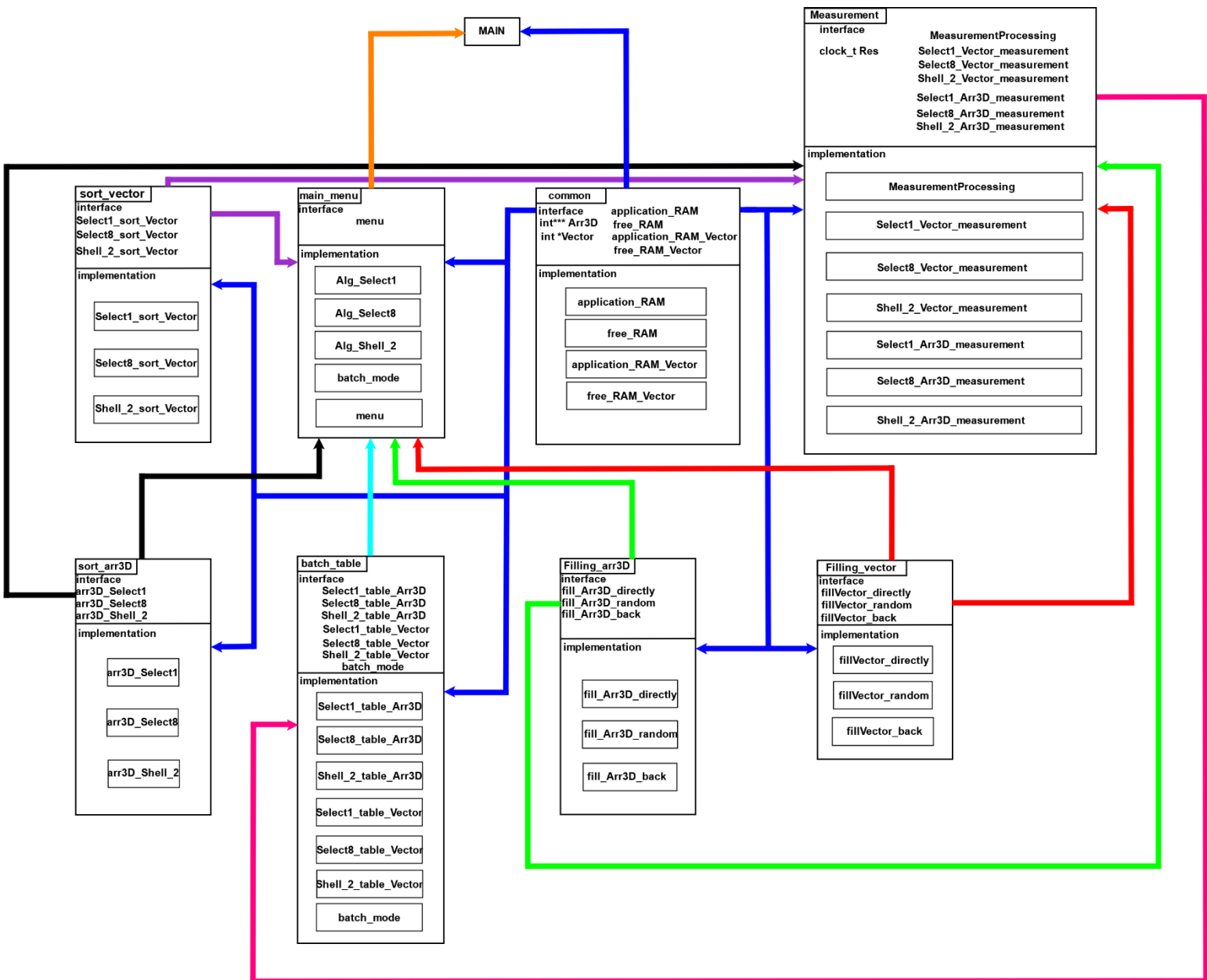
Кількість порівнянь:  $C = O(n^{1.2})$ ;

Кількість присвоєнь:  $R = O(n^{1.2})$ ;

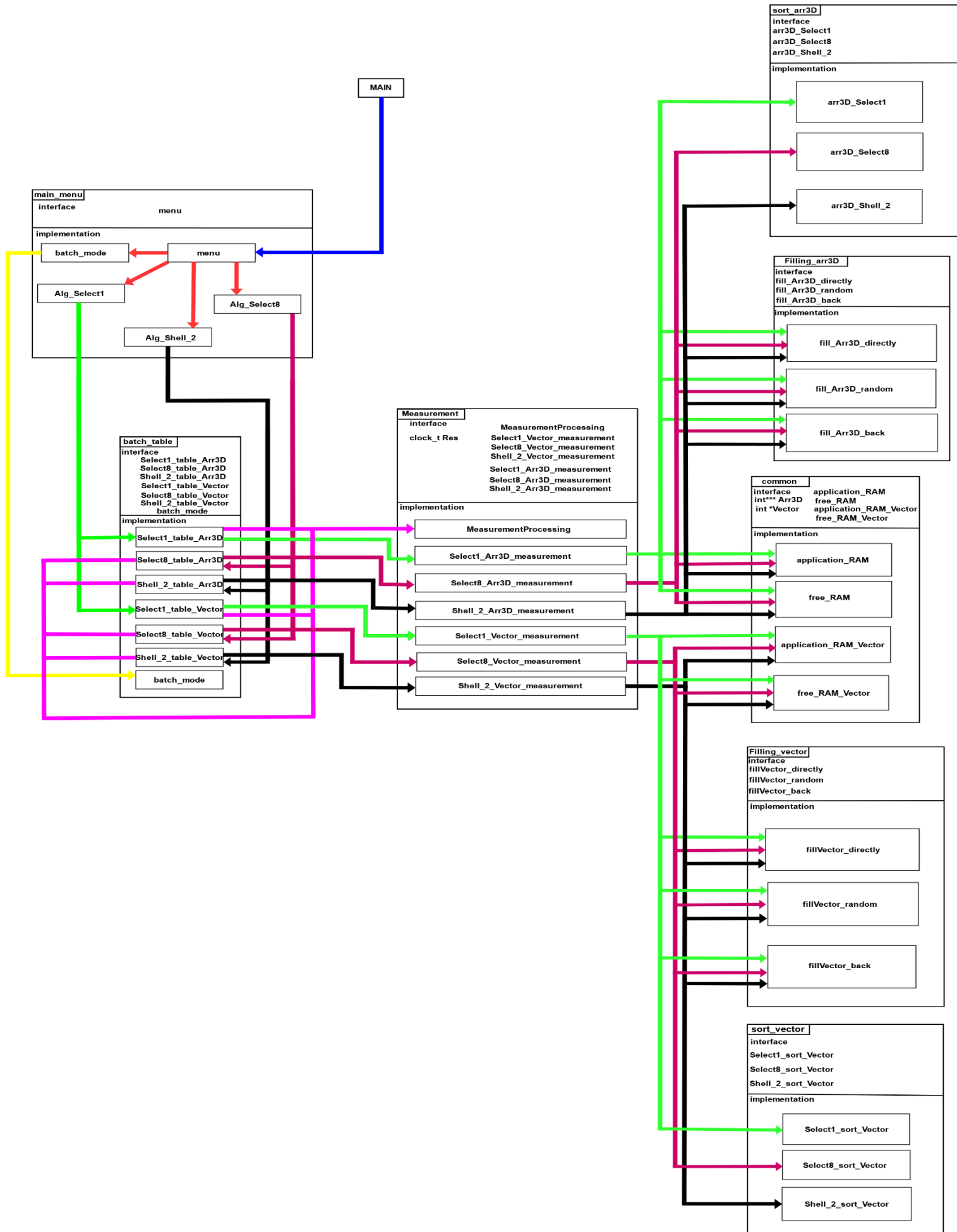
Алгоритм №2 методу сортування Шелла (спосіб реалізації на основі гібридного алгоритму «вставка-обмін») (додаток 1, рис. 25 № 23 ) на мові C :

```
clock_t Shell_2(int *A, int N)
{
    int tmp, 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++)
        {
            j=i;
            while (j>=k && A[j]<A[j-k])
            {
                tmp=A[j];
                A[j]=A[j-k];
                A[j-k]=tmp;
                j=j-k;
            }
        }
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

## Схема імпорту/експорту модулів програми курсової роботи



# Структурна схема взаємо-викликів процедур та функцій програми курсової роботи



## Опис призначення всіх функцій і процедур та їх параметрів

### Модуль *common*:

*common* – це модуль зі спільними, для інших модулів програми, структурами даних і функціями чи процедурами.

У інтерфейсній частині цього модуля, а саме у файлі *common.h*, описане наступне:

1) Розміри багатовимірного масиву та вектора;

2) Прототипи процедур:

*application\_RAM()*; // прототип процедур виділення пам'яті для 3D-масиву

*free\_RAM()*; // прототип процедур звільнення пам'яті для 3D-масиву

*application\_RAM\_Vector()*; // прототип процедур виділення пам'яті для вектора

*free\_RAM\_Vector()*; // прототип процедур звільнення пам'яті для вектора

У реалізацій частині модуля, а саме у файлі *common.c* описані реалізації перелічених вище процедур, і оскільки у програмі курсової будуть виконуватися виміри часу та інші математичні обчислення, то окрім стандартних бібліотеки *<stdio.h>* і *<stdlib.h>*, імпортовано також *<time.h>*, *<math.h>*.

### Модуль *Filling\_vector*:

*Filling\_vector* – модуль призначений для встановлення початкової відсортованості вектора (впорядкований, випадково-впорядкований, обернено-впорядкований).

У інтерфейсній частині цього модуля, а саме у файлі `Filling_vector.h`, описане наступне:

- 1) `fillVector_directly()`; //прототип процедури для заповнення вектора строго за збільшенням
- 2) `fillVector_random()`; //прототип процедури для заповнення вектора випадковими числами
- 3) `fillVector_back()`; //прототип процедури для заповнення вектора строго за зменшенням

У реалізацій частині модуля, а саме у файлі `Filling_vector.c` описані реалізації перелічених вище процедур та імпортовано модуль `common`.

Модуль ***sort\_Vector***:

`sort_Vector` – модуль призначений для сортування вектору заданими за варіантом №119 алгоритмами (прямий вибір №1, прямий вибір №8, метод Шелла №2).

У інтерфейсній частині цього модуля, а саме у файлі `sort_Vector.h`, описане наступне:

- 1) `Select1_sort_Vector()`; //прототип функції для сортування вектора методом прямого вибору №1;
- 2) `Select8_sort_Vector()`; //прототип функції для сортування вектора методом прямого вибору №8;
- 3) `Shell_2_sort_Vector()`; //прототип функції для сортування вектора методом Шелла №2;

У реалізацій частині модуля, а саме у файлі `sort_Vector.c` описані реалізації перелічених вище функцій, і відповідно імпортовано модуль `common`.

### Модуль *Filling\_arr3D*:

Filling\_arr3D – модуль призначений для встановлення початкової відсортованості 3D – масиву(впорядкований, випадково-впорядкований, обернено-впорядкований).

У інтерфейсній частині цього модуля, а саме у файлі Filling\_arr3D.h, описане наступне:

- 1) fill\_Arr3D\_directly();//прототип процедури для заповнення 3D масиву строго за збільшенням;
- 2) fill\_Arr3D\_random();//прототип процедури для заповнення 3D масиву випадковими числами;
- 3) fill\_Arr3D\_back();//прототип процедури для заповнення 3D масиву строго за зменшенням;

У реалізацій частині модуля, а саме у файлі Filling\_arr3D.c описані реалізації перелічених вище процедур, і відповідно імпортовано модуль common.

### Модуль *sort\_arr3D*:

sort\_arr3D – це модуль призначений для сортування 3D – масива, заданими за варіантом №119 алгоритмами(прямий вибір №1, прямий вибір №8, метод Шелла № 2).

У інтерфейсній частині цього модуля, а саме у файлі sort\_arr3D.h, описано наступне:

- 1) arr3D\_Select1();//прототип функції для сортування 3D масиву методом прямого вибору номер 1;
- 2) arr3D\_Select8();//прототип функції для сортування 3D масиву методом прямого вибору номер 8;



3) `arr3D_Shell_2();`//прототип функції для сортування 3D масиву методом Шелла номер 2;

У реалізаційній частині цього модуля, а саме у файлі `sort_arr3D.c` описані реалізації перелічених вище функцій, і відповідно імпортовано модуль `common`.

Модуль ***Measurement***:

`Measurement` – це модуль призначений для усереднення вимірів часу сортування вектору та тривимірного масиву.

У інтерфейсній частині цього модуля, а саме у файлі `Measurement.h`, описано наступне:

- 1) Оголошення глобального вектора який зберігатиме часи сортування вектора чи 3D – масиву, для подальшого їх усереднення;
- 2) `MeasurementProcessing();`// функція обробки і усереднення значень вимірів часу роботи алгоритму;
- 3) `Select1_Vector_measurement(char mode);`//процедура для заповнення масиву вимірів часу для прямого вибору номер 1 (варіант з вектором);
- 4) `Select8_Vector_measurement(char mode);`//процедура для заповнення масиву вимірів часу для прямого вибору номер 8 (варіант з вектором);
- 5) `Shell_2_Vector_measurement(char mode);`//процедура для заповнення масиву вимірів часу для методу Шелла номер 2 (варіант з вектором);
- 6) `Select1_Arr3D_measurement(char mode);`//процедура для заповнення масиву вимірів часу для прямого вибору номер 1 (варіант з 3D масивом);
- 7) `Select8_Arr3D_measurement(char mode);`//процедура для заповнення масиву вимірів часу для прямого вибору номер 8 (варіант з 3D масивом);

8) Shell\_2\_Arr3D\_measurement(char mode); //процедура для заповнення масиву вимірів часу для методу Шелла номер 2 (варіант з 3D масивом);

У реалізаційній частині цього модуля, а саме у файлі Measurement.c описані реалізації перелічених вище процедур , і відповідно імпортовано модулі:

1)common;

2)Filling\_vector;

3) sort\_Vector;

4) Filling\_arr3D;

5)sort\_arr3D;

Модуль **batch\_table**:

batch\_table – це модуль призначений для виводу на екран таблиць з часом виконання сортування у масиві і векторі різними алгоритмами.

У інтерфейсній частині цього модуля, а саме у файлі batch\_table.h, описано наступне:

1) Select1\_table\_Arr3D(); //процедура виводу таблиці з результатами вимірювання швидкодії сортування прямим вибором номер 1 (варіант з 3D масивом);

2) Select8\_table\_Arr3D(); //процедура виводу таблиці з результатами вимірювання швидкодії сортування прямим вибором номер 8 (варіант з 3D масивом);

3) Shell\_2\_table\_Arr3D(); //процедура виводу таблиці з результатами вимірювання швидкодії сортування методом Шелла номер 2 (варіант з 3D масивом);

- 4) `Select1_table_Vector()`; //процедура виводу таблиці з результатами вимірювання швидкодії сортування прямим вибором номер 1 (варіант з вектором);
- 5) `Select8_table_Vector()`; //процедура виводу таблиці з результатами вимірювання швидкодії сортування прямим вибором номер 8 (варіант з вектором);
- 6) `Shell_2_table_Vector()`; //процедура виводу таблиці з результатами вимірювання швидкодії сортування методом Шелла 2 (варіант з вектором);
- 7) `batch_mode()`; //вивід всіх таблиць для всіх випадків відсортованості з усіма алгоритмами і масивами та векторами;

У реалізаційній частині цього модуля, а саме у файлі `batch_table.c` описані реалізації перелічених вище процедур , і відповідно імпортовано модулі:

- 1) `common`;
- 2) `Measurement`;

Модуль ***main\_menu***:

`main_menu` – модуль призначений для створення діалогового інтерфейсу програми.

У інтерфейсній частині цього модуля, а саме у файлі `main_menu.h`, описано прототип процедури `menu`, яка власне і забезпечить діалоговий інтерфейс.

У реалізаційній частині цього модуля написана реалізації кількох процедур:

- 1) `Alg_Select1()` //процедура в меню для вибору структури даних в якій буде відбуватися сортування прямим вибором № 1;

- 2) Alg\_Select8()//процедура в меню для вибору структури даних в якій буде відбуватися сортування прямим вибором №8;
- 3) Alg\_Shell\_2()//процедура в меню для вибору структури даних в якій буде сортування методом Шелла № 2;
- 4) menu()//процедура головного меню;

Також у main\_menu імпортовано такі модулі:

- 1)common;
- 2)batch\_table;
- 3) Filling\_arr3D;
- 4) sort\_arr3D;

Також оскільки у цьому модулі використовується функція з стандартної бібліотеки мови C, а саме <windows.h>, її також імпортовано в цей модуль.

## Текст програми з коментарями

### main.c

```
#include "main_menu.h"
int main()
{
    menu();
    return 0 ;
}
```

### common.h

```
#ifndef COMMON_H_INCLUDED
#define COMMON_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
// задання вимірів для тривимірного масиву
#define N_S 100
#define M_S 16
#define P_S 32000
#define VECTOR 32000 // встановлення розміру вектора

int*** Arr3D; // оголошення тривимірного масиву
int *Vector; //оголошення вектору
void application_RAM ();
void free_RAM();
void application_RAM_Vector();
void free_RAM_Vector();

#endif // COMMON_H_INCLUDED
```

### common.c

```
#include "common.h"
void application_RAM() // процедура виділення пам'яті для 3D масиву
{
    Arr3D = (int***) malloc(P_S*sizeof(int**));
    for (int k=0; k < P_S; k++)
    {
        Arr3D[k] = (int**) malloc(M_S*sizeof(int*));
        for (int i=0; i < M_S; i++)
            Arr3D[k][i] = (int*) malloc(N_S*sizeof(int));
    }
}
void free_RAM() // процедура звільнення пам'яті для 3D масиву
{
    for (int k=0; k < P_S; k++)
    {
        for (int i=0; i < M_S; i++)
            free(Arr3D[k][i]);
        free(Arr3D[k]);
    }
    free(Arr3D);
}
void application_RAM_Vector()//процедура виділення пам'яті для вектора
{
    Vector = (int*) malloc(VECTOR*sizeof(int));
}
```

```
void free_RAM_Vector()//процедура звільнення пам'яті для вектора
{
    free(Vector);
}
```

## Filling\_vector.h

```
#ifndef FILLING_VECTOR_H_INCLUDED
#define FILLING_VECTOR_H_INCLUDED

void fillVector_directly();
void fillVector_random();
void fillVector_back();

#endif // FILLING_VECTOR_H_INCLUDED
```

## Filling\_vector.c

```
#include "common.h"
#include "Filling_vector.h"

void fillVector_directly() //заповнення вектору строго за збільшенням
{
    for (int i = 0; i < VECTOR; i++)
    {
        Vector[i] = i + 1;
    }
}
void fillVector_random() //заповнення вектору "випадковими" числами
{
    for (int i = 0; i < VECTOR; i++)
    {
        Vector[i] = rand()%VECTOR;
    }
}
void fillVector_back() //заповнення вектору строго за зменшенням
{
    for (int i = 0; i < VECTOR; i++)
    {
        Vector[i] = VECTOR - i;
    }
}
```

## sort\_Vector.h

```
#ifndef SORT_VECTOR_H_INCLUDED
#define SORT_VECTOR_H_INCLUDED

clock_t Select1_sort_Vector();
clock_t Select8_sort_Vector();
clock_t Shell_2_sort_Vector();

#endif // SORT_VECTOR_H_INCLUDED
```

## sort\_Vector.c

```
#include "common.h"
#include "sort_Vector.h"
clock_t Select1_sort_Vector()//сортування вектора алгоритмом прямого вибору
№ 1
{
    int Min, imin;
    clock_t time_start, time_stop;
    time_start = clock();
    for(int s = 0; s < VECTOR-1; s++)
    {
        Min = Vector[s];
        imin = s;
        for(int i = s+1; i < VECTOR; i++)
        {
            if (Vector[i] < Min)
            {
                Min = Vector[i];
                imin = i;
            }
        }
        Vector[imin] = Vector[s];
        Vector[s] = Min;
    }
    time_stop = clock();
    return time_stop - time_start;
}
clock_t Select8_sort_Vector()//сортування вектора алгоритмом прямого вибору
№ 8
{
    int L, R, imin, imax, tmp;
    clock_t time_start, time_stop;
    time_start = clock();
    L= 0;
    R = VECTOR - 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)
        {
            tmp = Vector[imin];
            Vector[imin] = Vector[L];
            Vector[L] = tmp;
        }
        if (imax!=R)
        {
            if (imax==L)
            {
                tmp=Vector[imin];
                Vector[imin]=Vector[R];
                Vector[R]=tmp;
            }
            else
            {

```

```

        tmp=Vector[imax];
        Vector[imax]=Vector[R];
        Vector[R]=tmp;
    }
}
L=L+1;
R=R-1;
}
time_stop = clock();
return time_stop - time_start;
}
clock_t Shell_2_sort_Vector()//сортування вектора алгоритмом Шелла № 2
{
    int tmp, t, j, k;
    clock_t time_start, time_stop;
    time_start = clock();
    if (VECTOR<4) t=1;
    else t=(int)log2f((float)VECTOR)-1;
    int Stages[t];
    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<VECTOR; i++)
        {
            j=i;
            while (j>=k && Vector[j]<Vector[j-k])
            {
                tmp=Vector[j];
                Vector[j]=Vector[j-k];
                Vector[j-k]=tmp;
                j=j-k;
            }
        }
    }
    time_stop = clock();
    return time_stop - time_start;
}

```



## Filling\_arr3D.h

```
#ifndef FILLING_ARR3D_H_INCLUDED
#define FILLING_ARR3D_H_INCLUDED

void fill_Arr3D_directly();
void fill_Arr3D_random();
void fill_Arr3D_back();

#endif // FILLING_ARR3D_H_INCLUDED
```

## Filling\_arr3D.c

```
#include "common.h"
#include "Filling_arr3D.h"
void fill_Arr3D_directly()//заповнення 3D масиву строго за збільшенням
{
    int number=0;
    for (int k=0; k<P_S; k++)
    {
        for (int j=0; j<N_S; j++)
        {
            for (int i=0; i<M_S; i++)
            {
                Arr3D[k][i][j] = number++;
            }
        }
    }
}
void fill_Arr3D_random()//заповнення 3D масиву випадковими числами
{
    for (int k=0; k<P_S; k++)
    {
        for (int j=0; j<N_S; j++)
        {
            for (int i=0; i<M_S; i++)
            {
                Arr3D[k][i][j] = rand() % (P_S*M_S*N_S);
            }
        }
    }
}

void fill_Arr3D_back()//заповнення 3D масиву строго за зменшенням
{
    int number = P_S*M_S*N_S;
    for (int k=0; k<P_S; k++)
    {
        for (int j=0; j<N_S; j++)
        {
            for (int i=0; i<M_S; i++)
            {
                Arr3D[k][i][j] = number--;
            }
        }
    }
}
```

## sort\_arr3D.h

```
#ifndef SORT_ARR3D_H_INCLUDED
#define SORT_ARR3D_H_INCLUDED

clock_t arr3D_Select1();
clock_t arr3D_Select8();
clock_t arr3D_Shell_2();

#endif // SORT_ARR3D_H_INCLUDED
```

## sort\_arr3D.c

```
#include "common.h"
#include "sort_arr3D.h"

clock_t arr3D_Select1()//сортування 3D масиву алгоритмом прямого вибору № 1
{
    int Min, imin;
    int A[M_S][N_S];
    clock_t time_start, time_stop;
    time_start = clock();
    for (int s = 0; s < P_S - 1; s++)// прохід по всіх перерізах окрім
останнього
    {
        Min = Arr3D[s][0][0];//встановлення початкового мінімуму для
запуску основного алгоритму
        imin = s;
        for (int i = s + 1; i < P_S; i++)// пошук мінімуму серед перших
елементів кожного перерізу
        {
            if (Arr3D[i][0][0] < Min)
            {
                Min = Arr3D[i][0][0];
                imin = i;
            }
        }
        for (int i = 0; i < M_S; i++)//перестановка перерізів за
незменшенням перших елементів
        {
            for (int j = 0; j < N_S; j++)
            {
                A[i][j] = Arr3D[s][i][j];
                Arr3D[s][i][j] = Arr3D[imin][i][j];
                Arr3D[imin][i][j] = A[i][j];
            }
        }
        time_stop = clock();
        return time_stop - time_start;
    }
}

clock_t arr3D_Select8()//сортування 3D масиву алгоритмом прямого вибору № 8
{
    int L, R, imin, imax, tmp;
    clock_t time_start, time_stop;
    time_start = clock();
    L = 0;
    R = P_S - 1;
```

```

while (L < R)
{
    imin = L;
    imax = L;

    /*пошук мінімального і максимального елементів
    серед перших елементів кожного перерізу*/

    for (int i = L + 1; i <= R; i++)
    {
        if (Arr3D[i][0][0] < Arr3D[imin][0][0]) imin = i;
        else if (Arr3D[i][0][0] > Arr3D[imax][0][0]) imax = i;
    }
    if (imin != L) // перестановка мінімального в початок
    {
        for (int m = 0; m < M_S; m++)
        {
            for (int n = 0; n < N_S; n++)
            {
                tmp = Arr3D[L][m][n];
                Arr3D[L][m][n] = Arr3D[imin][m][n];
                Arr3D[imin][m][n] = tmp;
            }
        }
    }
    if (imax != R) // перестановка максимального в кінець
    {
        if (imax == L) imax = imin; // виправлення індекса максимального
        якщо вже був обмін
        for (int m = 0; m < M_S; m++)
        {
            for (int n = 0; n < N_S; n++)
            {
                tmp = Arr3D[R][m][n];
                Arr3D[R][m][n] = Arr3D[imax][m][n];
                Arr3D[imax][m][n] = tmp;
            }
        }
    }
    L = L + 1;
    R = R - 1;
}
time_stop = clock();
return time_stop - time_start;
}
clock_t arr3D_Shell_2() // сортування 3D масиву алгоритмом Шелла № 2
{
    int A[M_S][N_S];
    int t, k, i, j, r, l, m;
    clock_t time_start, time_stop;
    time_start = clock();

    // визначення кількості кроків

    if (P_S < 4) t = 1;
    else t = (int)log2f((float)P_S) - 1;
    int H[t];
    H[t-1] = 1;
    for (int i = t-2; i >= 0; i--) H[i] = 2*H[i+1] + 1; // задання послідовності
    кроків

```

```

    for(m = 0; m < t; m++)
    {
        k = H[m];
        for(l=k; l < P_S; l++)// прохід по масиву з поточним кроком
        {
            r=l;
            while(r>=k && Arr3D[r][0][0]<Arr3D[r-k][0][0])// поки поточний
менший – міняємо місцями
            {
                for(i=0; i<M_S; i++)// копіювання r -> A
                {
                    for(j=0; j<N_S; j++)
                    {
                        A[i][j]=Arr3D[r][i][j];
                    }
                }
                for(i=0; i<M_S; i++)// копіювання r-k -> r
                {
                    for(j=0; j<N_S; j++)
                    {
                        Arr3D[r][i][j]=Arr3D[r-k][i][j];
                    }
                }
                for(i=0; i<M_S; i++)// копіювання A -> r-k
                {
                    for(j=0; j<N_S; j++)
                    {
                        Arr3D[r-k][i][j]=A[i][j];
                    }
                }
                r=r-k;
            }
        }
    }
    time_stop = clock();
    return time_stop - time_start;
}

```

## Measurement.h

```

#ifndef MEASUREMENT_H_INCLUDED
#define MEASUREMENT_H_INCLUDED

#include <time.h>
#define measurements_number 28
#define rejected_number 2
#define min_max_number 3

extern clock_t Res[measurements_number];
float MeasurementProcessing();// функція обробки і усереднення значень
вимірів часу роботи алгоритму

void Select1_Vector_measurement(char mode);
void Select8_Vector_measurement(char mode);
void Shell_2_Vector_measurement(char mode);

void Select1_Arr3D_measurement(char mode);
void Select8_Arr3D_measurement(char mode);
void Shell_2_Arr3D_measurement(char mode);

#endif // MEASUREMENT_H_INCLUDED

```

## Measurement.c

```
#include "common.h"
#include "Filling_vector.h"
#include "sort_Vector.h"
#include "Filling_arr3D.h"
#include "sort_arr3D.h"
#include "Measurement.h"
clock_t Res[measurements_number];
float MeasurementProcessing()//обробка і усереднення вимірів часу
{
    long int Sum;
    float AverageValue;
    // Крок 1. Для виконання Кроку 1 просто починаємо алгоритм з індексу
    rejected_number.

    /*Крок 2.Для знаходження min_max_number мінімальних і максимальних
    значень вимірів

    виконуємо min_max_number ітерацій головного цикла алгоритму шейкерного
    сортування в

    діапазоні індексів від rejected_number до measurements_number-1. В
    результаті,

    min_max_number мінімальних значень вимірів будуть знаходитись на
    позиціях

    в діапазоні індексів від rejected_number до
    rejected_number+min_max_number-1,

    а min_max_number максимальних значень вимірів - на позиціях в діапазоні
    індексів

    від measurements_number-min_max_number до measurements_number-1.*/
    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;

    // Крок 3.

    /* Знаходимо середнє значення вимірів після відкидання
    rejected_number перших

    вимірів та min_max_number вимірів з мінімальними значеннями і
    min_max_number

    вимірів з максима- льними значеннями, тобто середнє значення
    вимірів в

    діапазоні індексів від rejected_number + min_max_number до
    measurements_number

    - min_max_number - 1. */
}
Sum=0;
for (int i = rejected_number + min_max_number; i < measurements_number
- min_max_number; i++)
    Sum = Sum + Res[i];

/* Кількість вимірів, що залишилась для обчислення середнього значення,
дорівнює

measurements_number - 2 * min_max_number - rejected_number */

AverageValue = (float)Sum/(float) (measurements_number -
2*min_max_number - rejected_number);
printf("");
return AverageValue;
}
void Select1_Vector_measurement(char mode)// варіант з вектором
{
    /* Ця процедура заповнює масив вимірів часу для алгоритму
    прямого вибору № 1 для трьох випадків початкової
    відсортованості.

    Ця процедура приймає один параметр який може бути:

    d - встановлення початкової прямої відсортованості
    r - встановлення початкової випадкової відсортованості
    b - встановлення початкової оберненої відсортованості
    */
    for(int i = 0; i < measurements_number; i++)
    {
        if(mode == 'd')
        {
            application_RAM_Vector();
            fillVector_directly();
            Res[i]= Select1_sort_Vector();
            free_RAM_Vector();
        }
        else if(mode == 'r')
        {
            application_RAM_Vector();

```

```

        fillVector_random();
        Res[i] = Select1_sort_Vector();
        free_RAM_Vector();
    }
    else if(mode == 'b')
    {
        application_RAM_Vector();
        fillVector_back();
        Res[i] = Select1_sort_Vector();
        free_RAM_Vector();
    }
}

void Select8_Vector_measurement(char mode) // варіант з вектором
{
    /* Ця процедура заповнює масив вимірів часу для алгоритму
    прямого вибору № 8 для трьох випадків початкової
    відсортованості.

    Ця процедура приймає один параметр який може бути:

    d - встановлення початкової прямої відсортованості
    r - встановлення початкової випадкової відсортованості
    b - встановлення початкової оберненої відсортованості
    */
    for(int i = 0; i < measurements_number; i++)
    {
        if(mode == 'd')
        {
            application_RAM_Vector();
            fillVector_directly();
            Res[i] = Select8_sort_Vector();
            free_RAM_Vector();
        }
        else if(mode == 'r')
        {
            application_RAM_Vector();
            fillVector_random();
            Res[i] = Select8_sort_Vector();
            free_RAM_Vector();
        }
        else if(mode == 'b')
        {
            application_RAM_Vector();
            fillVector_back();
            Res[i] = Select8_sort_Vector();
            free_RAM_Vector();
        }
    }
}

void Shell_2_Vector_measurement(char mode) // варіант з вектором
{
    /* Ця процедура заповнює масив вимірів часу для алгоритму
    Шелла № 2 для трьох випадків початкової відсортованості.

    Ця процедура приймає один параметр який може бути:

```

```

    d - встановлення початкової прямої відсортованості

    r - встановлення початкової випадкової відсортованості

    b - встановлення початкової оберненої відсортованості
*/
for(int i = 0; i < measurements_number; i++)
{
    if(mode == 'd')
    {
        application_RAM_Vector();
        fillVector_directly();
        Res[i] = Shell_2_sort_Vector();
        free_RAM_Vector();
    }
    else if(mode == 'r')
    {
        application_RAM_Vector();
        fillVector_random();
        Res[i] = Shell_2_sort_Vector();
        free_RAM_Vector();
    }
    else if(mode == 'b')
    {
        application_RAM_Vector();
        fillVector_back();
        Res[i] = Shell_2_sort_Vector();
        free_RAM_Vector();
    }
}
}
void Select1_Arr3D_measurement(char mode)//варіант з 3D масивом
{
    /* Ця процедура заповнює масив вимірів часу для алгоритму
    прямого вибору № 1 для трьох випадків початкової
    відсортованості.

    Ця процедура приймає один параметр який може бути:

    d - встановлення початкової прямої відсортованості

    r - встановлення початкової випадкової відсортованості

    b - встановлення початкової оберненої відсортованості
*/
    for (int i = 0; i < measurements_number; i++)
    {
        if (mode == 'd')
        {
            application_RAM();
            fill_Arr3D_directly();
            Res[i] = arr3D_Select1();
            free_RAM();
        }
        else if (mode == 'r')
        {
            application_RAM();
            fill_Arr3D_random();
        }
    }
}

```



```

        Res[i] = arr3D_Select1();
        free_RAM();
    }
    else if (mode == 'b')
    {
        application_RAM();
        fill_Arr3D_back();
        Res[i] = arr3D_Select1();
        free_RAM();
    }
}
}
}
void Select8_Arr3D_measurement(char mode)//варіант з 3D масивом
{
    /* Ця процедура заповнює масив вимірів часу для алгоритму
    прямого вибору № 8 для трьох випадків початкової
    відсортованості.

    Ця процедура приймає один параметр який може бути:

    d - встановлення початкової прямої відсортованості
    r - встановлення початкової випадкової відсортованості
    b - встановлення початкової оберненої відсортованості
    */
    for (int i = 0; i < measurements_number; i++)
    {
        if (mode == 'd')
        {
            application_RAM();
            fill_Arr3D_directly();
            Res[i] = arr3D_Select8();
            free_RAM();
        }
        else if (mode == 'r')
        {
            application_RAM();
            fill_Arr3D_random();
            Res[i] = arr3D_Select8();
            free_RAM();
        }
        else if (mode == 'b')
        {
            application_RAM();
            fill_Arr3D_back();
            Res[i] = arr3D_Select8();
            free_RAM();
        }
    }
}
}
void Shell_2_Arr3D_measurement(char mode)//варіант з 3D масивом
{
    /* Ця процедура заповнює масив вимірів часу для алгоритму
    Шелла № 2 для трьох випадків початкової відсортованості.

    Ця процедура приймає один параметр який може бути:

```

```

        d - встановлення початкової прямої відсортованості

        r - встановлення початкової випадкової відсортованості

        b - встановлення початкової оберненої відсортованості
*/
for (int i = 0; i < measurements_number; i++)
{
    if (mode == 'd')
    {
        application_RAM();
        fill_Arr3D_directly();
        Res[i] = arr3D_Shell_2();
        free_RAM();
    }
    else if (mode == 'r')
    {
        application_RAM();
        fill_Arr3D_random();
        Res[i] = arr3D_Shell_2();
        free_RAM();
    }
    else if (mode == 'b')
    {
        application_RAM();
        fill_Arr3D_back();
        Res[i] = arr3D_Shell_2();
        free_RAM();
    }
}
}

```

## batch\_table.h

```

#ifndef BATCH_TABLE__H_INCLUDED
#define BATCH_TABLE__H_INCLUDED

void Select1_table_Arr3D();
void Select8_table_Arr3D();
void Shell_2_table_Arr3D();
void Select1_table_Vector();
void Select8_table_Vector();
void Shell_2_table_Vector();
void batch_mode();

#endif // BATCH_TABLE__H_INCLUDED

```

## batch\_table.c

```
#include "batch_table.h"
#include "common.h"
#include "Measurement.h"

void Select1_table_Arr3D()//варіант з 3D масивом
{
    float directly, random, back_d;

    /* Формування таблиці з результатами вимірів для
        алгоритму прямого вибору № 1 для усіх випадків
        відсортованості,згідно із зразком у методичці
    */

    printf("\t\t\tTable for the array: P = %d, M = %d, N = %d |array
traversal method #7|\n\n",P_S,M_S,N_S);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Select1_Arr3D_measurement('d');
    directly = MeasurementProcessing();
    Select1_Arr3D_measurement('r');
    random = MeasurementProcessing();
    Select1_Arr3D_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select1\t\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
}

void Select8_table_Arr3D()// варіант з 3D масивом
{
    float directly, random, back_d;

    /* Формування таблиці з результатами вимірів для
        алгоритму прямого вибору № 8 для усіх випадків
        відсортованості,згідно із зразком у методичці
    */

    printf("\t\t\tTable for the array: P = %d, M = %d, N = %d |array
traversal method #7|\n\n",P_S,M_S,N_S);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Select8_Arr3D_measurement('d');
    directly = MeasurementProcessing();
    Select8_Arr3D_measurement('r');
    random = MeasurementProcessing();
    Select8_Arr3D_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select8\t\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
}

void Shell2_table_Arr3D()//варіант з 3D масивом
{
    float directly, random, back_d;

    /* Формування таблиці з результатами вимірів для
        алгоритму Шелла № 2 для усіх випадків
```

```

        відсортованості,згідно із зразком у методичці
    */

    printf("\t\t\tTable for the array: P = %d, M = %d, N = %d |array
traversal method #7|\n\n",P_S,M_S,N_S);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Shell_2_Arr3D_measurement('d');
    directly = MeasurementProcessing();
    Shell_2_Arr3D_measurement('r');
    random = MeasurementProcessing ();
    Shell_2_Arr3D_measurement('b');
    back_d = MeasurementProcessing ();
    printf("Shell_2\t\t\t%.2f\t\t\t%.2f\t\t\t%.2f\n\n", directly, random,
back_d);
}
void Select1_table_Vector()//варіант з вектором
{
    float directly, random, back_d;

    /* Формування таблиці з результатами вимірів для

        алгоритму прямого вибору № 1 для усіх випадків

        відсортованості,згідно із зразком у методичці
    */

    printf("\t\t\tTable for the vector: VECTOR = %d\n\n", VECTOR);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Select1_Vector_measurement('d');
    directly = MeasurementProcessing();
    Select1_Vector_measurement('r');
    random = MeasurementProcessing ();
    Select1_Vector_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select1\t\t\t%.2f\t\t\t%.2f\t\t\t%.2f\n\n", directly, random,
back_d);
}
void Select8_table_Vector()//варіант з вектором
{
    float directly, random, back_d;

    /* Формування таблиці з результатами вимірів для

        алгоритму прямого вибору № 8 для усіх випадків

        відсортованості,згідно із зразком у методичці
    */

    printf("\t\t\tTable for the vector: VECTOR = %d\n\n", VECTOR);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Select8_Vector_measurement('d');
    directly = MeasurementProcessing();
    Select8_Vector_measurement('r');
    random = MeasurementProcessing ();
    Select8_Vector_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select8\t\t\t%.2f\t\t\t%.2f\t\t\t%.2f\n\n", directly, random,
back_d);
}
void Shell_2_table_Vector()//вваріант з вектором
{

```

```

float directly, random, back_d;

/* Формування таблиці з результатами вимірів для
   алгоритму Шелла № 2 для усіх випадків
   відсортованості, згідно із зразком у методичці
*/

printf("\t\t\t\tTable for the vector: VECTOR = %d\n\n", VECTOR);
printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
Shell_2_Vector_measurement('d');
directly = MeasurementProcessing();
Shell_2_Vector_measurement('r');
random = MeasurementProcessing();
Shell_2_Vector_measurement('b');
back_d = MeasurementProcessing();
printf("Shell_2\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
}
void batch_mode()//вивід всіх таблиць для всіх випадків відсортованості з
усіма алгоритмами і масивами
{
    float directly, random, back_d;
    printf("\t\t\t\tTable for the array: P = %d, M = %d, N = %d |array
traversal method #7|\n\n", P_S, M_S, N_S);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Select1_Arr3D_measurement('d');
    directly = MeasurementProcessing();
    Select1_Arr3D_measurement('r');
    random = MeasurementProcessing();
    Select1_Arr3D_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select1\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
    Select8_Arr3D_measurement('d');
    directly = MeasurementProcessing();
    Select8_Arr3D_measurement('r');
    random = MeasurementProcessing();
    Select8_Arr3D_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select8\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
    Shell_2_Arr3D_measurement('d');
    directly = MeasurementProcessing();
    Shell_2_Arr3D_measurement('r');
    random = MeasurementProcessing();
    Shell_2_Arr3D_measurement('b');
    back_d = MeasurementProcessing();
    printf("Shell_2\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
    printf("\t\t\t\tTable for the vector: VECTOR = %d\n\n", VECTOR);
    printf("\t\t\tOrdered\t\t\tRandom Ordered\t\tBack Ordered\n\n");
    Select1_Vector_measurement('d');
    directly = MeasurementProcessing();
    Select1_Vector_measurement('r');
    random = MeasurementProcessing();
    Select1_Vector_measurement('b');
    back_d = MeasurementProcessing();
    printf("Select1\t\t%.2f\t\t%.2f\t\t%.2f\n\n", directly, random,
back_d);
}

```

```

        Select8_Vector_measurement('d');
        directly = MeasurementProcessing();
        Select8_Vector_measurement('r');
        random = MeasurementProcessing();
        Select8_Vector_measurement('b');
        back_d = MeasurementProcessing();
        printf("Select8\t\t\t%.2f\t\t\t%.2f\t\t\t%.2f\n\n", directly, random,
back_d);
        Shell_2_Vector_measurement('d');
        directly = MeasurementProcessing();
        Shell_2_Vector_measurement('r');
        random = MeasurementProcessing();
        Shell_2_Vector_measurement('b');
        back_d = MeasurementProcessing();
        printf("Shell_2\t\t\t%.2f\t\t\t%.2f\t\t\t%.2f\n\n", directly, random,
back_d);
    }

```

## main\_menu.h

```

#ifndef MAIN_MENU_H_INCLUDED
#define MAIN_MENU_H_INCLUDED

void menu(); // Головные меню

#endif // MAIN_MENU_H_INCLUDED

```

## main\_menu.c

```
#include "common.h"
#include "main_menu.h"
#include "batch_table.h"
#include "Filling_arr3D.h"
#include "Filling_vector.h"
#include "sort_Vector.h"
#include "sort_arr3D.h"
#include <windows.h>

void Alg_Select1()//процедура в меню для вибору структури даних в якій буде
сортування прямим вибором номер 1
{
    int v;
    while(1)
    {
        system("cls");
        printf("Select data structure for sorting by Algorithm - Select1
:\n");
        printf("1. Arr3D\n");
        printf("2. Vector\n");
        printf("3. Return\n");
        if (scanf("%d", &v) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            system("pause");
            while (getchar() != '\n');
            continue;
        }
        switch (v)
        {
            case 1:
            {
                Select1_table_Arr3D();
                system("pause");
                break;
            }
            case 2:
            {
                Select1_table_Vector();
                system("pause");
                break;
            }
            case 3:
            {
                printf("Returning...\n");
                Sleep(2000);
                break;
            }
            default:
            {
                printf("Impossible mode. Please try again.\n");
                Sleep(1000);
                break;
            }
        }
        if (v == 3) break;
    }
}
```

**void** Alg\_Select8()//процедура в меню для вибору структури даних в якій буде сортування прямим вибором номер 8

```
{
    int y;
    while(1)
    {
        system("cls");
        printf("Select data structure for sorting by Algorithm - Select8
:\n");
        printf("1. Arr3D\n");
        printf("2. Vector\n");
        printf("3. Return\n");
        if (scanf("%d", &y) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            system("pause");
            while (getchar() != '\n');
            continue;
        }
        switch (y)
        {
            case 1:
            {
                Select8_table_Arr3D();
                system("pause");
                break;
            }
            case 2:
            {
                Select8_table_Vector();
                system("pause");
                break;
            }
            case 3:
            {
                printf("Returning...\n");
                Sleep(2000);
                break;
            }
            default:
            {
                printf("Impossible mode. Please try again.\n");
                Sleep(1000);
                break;
            }
        }
        if (y == 3) break;
    }
}
```



**void** Alg\_Shell\_2()//процедура в меню для вибору структури даних в якій буде сортування методом Шелла номер 2

```
{
    int y;
    while(1)
    {
        system("cls");
        printf("Select data structure for sorting by Algorithm - Shell_2
:\n");
        printf("1. Arr3D\n");
        printf("2. Vector\n");
        printf("3. Return\n");
        if (scanf("%d", &y) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            Sleep(1000);
            while (getchar() != '\n');
            continue;
        }
        switch (y)
        {
            case 1:
            {
                Shell_2_table_Arr3D();
                system("pause");
                break;
            }
            case 2:
            {
                Shell_2_table_Vector();
                system("pause");
                break;
            }
            case 3:
            {
                printf("Returning...\n");
                Sleep(2000);
                break;
            }
            default:
            {
                printf("Impossible mode. Please try again.\n");
                Sleep(1000);
                break;
            }
        }
        if (y == 3) break;
    }
}
```

```

void menu()//процедура головного меню
{
    int x;
    while(1)
    {
        system("cls");
        printf("MAIN MENU:\n");
        printf("1. Algorithm - Select1\n");
        printf("2. Algorithm - Select8\n");
        printf("3. Algorithm - Shell_2\n");
        printf("4. Batch_mode:\n");
        printf("5. Exit\n");
        printf("Select algorithm or mode:\n");
        if (scanf("%d", &x) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            Sleep(1000);
            while (getchar() != '\n');
            continue;
        }
        switch (x)
        {
            case 1:
            {
                Alg_Select1();
                continue;
            }
            case 2:
            {
                Alg_Select8();
                continue;
            }
            case 3:
            {
                Alg_Shell_2();
                continue;
            }
            case 4:
            {
                batch_mode();
                system("pause");
                continue;
            }
            case 5:
            {
                exit(0);
            }
            default:
            {
                printf("Impossible mode. Please try again.\n");
                Sleep(1000);
            }
        }
    }
}

```

## Тестування

Початковий стан 3D-масиву перед сортуванням методом прямого вибору № 1 є обернено-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
initial state of the array(before sorting by Select_1):  
Section 0:  
96 92 88 84  
95 91 87 83  
94 90 86 82  
93 89 85 81  
  
Section 1:  
80 76 72 68  
79 75 71 67  
78 74 70 66  
77 73 69 65  
  
Section 2:  
64 60 56 52  
63 59 55 51  
62 58 54 50  
61 57 53 49  
  
Section 3:  
48 44 40 36  
47 43 39 35  
46 42 38 34  
45 41 37 33  
  
Section 4:  
32 28 24 20  
31 27 23 19  
30 26 22 18  
29 25 21 17  
  
Section 5:  
16 12 8 4  
15 11 7 3  
14 10 6 2  
13 9 5 1
```

Стан 3D-масиву після сортування методом прямого вибору №1:

```
C:\code_blocks\COURSE_WOR  ×  +  v

state of the array(after sorting by Select_1):
Section 0:
16 12 8 4
15 11 7 3
14 10 6 2
13 9 5 1

Section 1:
32 28 24 20
31 27 23 19
30 26 22 18
29 25 21 17

Section 2:
48 44 40 36
47 43 39 35
46 42 38 34
45 41 37 33

Section 3:
64 60 56 52
63 59 55 51
62 58 54 50
61 57 53 49

Section 4:
80 76 72 68
79 75 71 67
78 74 70 66
77 73 69 65

Section 5:
96 92 88 84
95 91 87 83
94 90 86 82
93 89 85 81
```

Початковий стан 3D-масиву перед сортуванням методом прямого вибору № 1 є випадково-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
initial state of the array(before sorting by Select_1):  
Section 0:  
41 65 82 49  
35 76 80 27  
94 54 41 73  
4 78 17 11  
  
Section 1:  
19 39 4 38  
38 12 94 23  
27 62 45 71  
60 57 92 30  
  
Section 2:  
83 35 95 9  
18 91 3 56  
45 43 26 69  
40 6 93 31  
  
Section 3:  
29 22 83 28  
52 21 45 59  
11 69 73 14  
92 91 10 18  
  
Section 4:  
0 64 50 30  
58 88 9 93  
16 86 82 9  
14 93 86 92  
  
Section 5:  
45 36 11 50  
47 32 4 91  
84 70 48 65  
82 80 87 34
```

Стан 3D-масиву після сортування методом прямого вибору №1:

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
state of the array(after sorting by Select_1):  
Section 0:  
0 64 50 30  
58 88 9 93  
16 86 82 9  
14 93 86 92  
  
Section 1:  
19 39 4 38  
38 12 94 23  
27 62 45 71  
60 57 92 30  
  
Section 2:  
29 22 83 28  
52 21 45 59  
11 69 73 14  
92 91 10 18  
  
Section 3:  
41 65 82 49  
35 76 80 27  
94 54 41 73  
4 78 17 11  
  
Section 4:  
45 36 11 50  
47 32 4 91  
84 70 48 65  
82 80 87 34  
  
Section 5:  
83 35 95 9  
18 91 3 56  
45 43 26 69  
40 6 93 31
```

Початковий стан 3D-масиву перед сортуванням методом прямого вибору № 1 є  
прямо-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
initial state of the array(before sorting by Select_1):  
Section 0:  
0 4 8 12  
1 5 9 13  
2 6 10 14  
3 7 11 15  
  
Section 1:  
16 20 24 28  
17 21 25 29  
18 22 26 30  
19 23 27 31  
  
Section 2:  
32 36 40 44  
33 37 41 45  
34 38 42 46  
35 39 43 47  
  
Section 3:  
48 52 56 60  
49 53 57 61  
50 54 58 62  
51 55 59 63  
  
Section 4:  
64 68 72 76  
65 69 73 77  
66 70 74 78  
67 71 75 79  
  
Section 5:  
80 84 88 92  
81 85 89 93  
82 86 90 94  
83 87 91 95
```

Стан 3D – масиву після сортування методом прямого вибору № 1:

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
state of the array(after sorting by Select_1):  
Section 0:  
0 4 8 12  
1 5 9 13  
2 6 10 14  
3 7 11 15  
  
Section 1:  
16 20 24 28  
17 21 25 29  
18 22 26 30  
19 23 27 31  
  
Section 2:  
32 36 40 44  
33 37 41 45  
34 38 42 46  
35 39 43 47  
  
Section 3:  
48 52 56 60  
49 53 57 61  
50 54 58 62  
51 55 59 63  
  
Section 4:  
64 68 72 76  
65 69 73 77  
66 70 74 78  
67 71 75 79  
  
Section 5:  
80 84 88 92  
81 85 89 93  
82 86 90 94  
83 87 91 95
```



Початковий стан 3D-масиву перед сортуванням методом прямого вибору № 8 є  
обернено-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
initial state of the array(before sorting by Select_8):  
Section 0:  
96 92 88 84  
95 91 87 83  
94 90 86 82  
93 89 85 81  
  
Section 1:  
80 76 72 68  
79 75 71 67  
78 74 70 66  
77 73 69 65  
  
Section 2:  
64 60 56 52  
63 59 55 51  
62 58 54 50  
61 57 53 49  
  
Section 3:  
48 44 40 36  
47 43 39 35  
46 42 38 34  
45 41 37 33  
  
Section 4:  
32 28 24 20  
31 27 23 19  
30 26 22 18  
29 25 21 17  
  
Section 5:  
16 12 8 4  
15 11 7 3  
14 10 6 2  
13 9 5 1
```

Стан 3D – масиву після сортування методом прямого вибору № 8:

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
state of the array(after sorting by Select_8):  
Section 0:  
16 12 8 4  
15 11 7 3  
14 10 6 2  
13 9 5 1  
  
Section 1:  
32 28 24 20  
31 27 23 19  
30 26 22 18  
29 25 21 17  
  
Section 2:  
48 44 40 36  
47 43 39 35  
46 42 38 34  
45 41 37 33  
  
Section 3:  
64 60 56 52  
63 59 55 51  
62 58 54 50  
61 57 53 49  
  
Section 4:  
80 76 72 68  
79 75 71 67  
78 74 70 66  
77 73 69 65  
  
Section 5:  
96 92 88 84  
95 91 87 83  
94 90 86 82  
93 89 85 81
```

Початковий стан 3D-масиву перед сортуванням методом прямого вибору № 8 є випадково-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  v

initial state of the array(before sorting by Select_8):
Section 0:
41 65 82 49
35 76 80 27
94 54 41 73
4 78 17 11

Section 1:
19 39 4 38
38 12 94 23
27 62 45 71
60 57 92 30

Section 2:
83 35 95 9
18 91 3 56
45 43 26 69
40 6 93 31

Section 3:
29 22 83 28
52 21 45 59
11 69 73 14
92 91 10 18

Section 4:
0 64 50 30
58 88 9 93
16 86 82 9
14 93 86 92

Section 5:
45 36 11 50
47 32 4 91
84 70 48 65
82 80 87 34
```

Стан 3D – масиву після сортування методом прямого вибору № 8:

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
  
state of the array(after sorting by Select_8):  
Section 0:  
0 64 50 30  
58 88 9 93  
16 86 82 9  
14 93 86 92  
  
Section 1:  
19 39 4 38  
38 12 94 23  
27 62 45 71  
60 57 92 30  
  
Section 2:  
29 22 83 28  
52 21 45 59  
11 69 73 14  
92 91 10 18  
  
Section 3:  
41 65 82 49  
35 76 80 27  
94 54 41 73  
4 78 17 11  
  
Section 4:  
45 36 11 50  
47 32 4 91  
84 70 48 65  
82 80 87 34  
  
Section 5:  
83 35 95 9  
18 91 3 56  
45 43 26 69  
40 6 93 31
```

Початковий стан 3D-масиву перед сортуванням методом прямого вибору № 8 є  
прямо-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
initial state of the array(before sorting by Select_8):  
Section 0:  
0 4 8 12  
1 5 9 13  
2 6 10 14  
3 7 11 15  
  
Section 1:  
16 20 24 28  
17 21 25 29  
18 22 26 30  
19 23 27 31  
  
Section 2:  
32 36 40 44  
33 37 41 45  
34 38 42 46  
35 39 43 47  
  
Section 3:  
48 52 56 60  
49 53 57 61  
50 54 58 62  
51 55 59 63  
  
Section 4:  
64 68 72 76  
65 69 73 77  
66 70 74 78  
67 71 75 79  
  
Section 5:  
80 84 88 92  
81 85 89 93  
82 86 90 94  
83 87 91 95
```

Стан 3D – масиву після сортування методом прямого вибору № 8:

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
state of the array(after sorting by Select_8):  
Section 0:  
0 4 8 12  
1 5 9 13  
2 6 10 14  
3 7 11 15  
  
Section 1:  
16 20 24 28  
17 21 25 29  
18 22 26 30  
19 23 27 31  
  
Section 2:  
32 36 40 44  
33 37 41 45  
34 38 42 46  
35 39 43 47  
  
Section 3:  
48 52 56 60  
49 53 57 61  
50 54 58 62  
51 55 59 63  
  
Section 4:  
64 68 72 76  
65 69 73 77  
66 70 74 78  
67 71 75 79  
  
Section 5:  
80 84 88 92  
81 85 89 93  
82 86 90 94  
83 87 91 95
```

Початковий стан 3D-масиву перед сортуванням методом Шелла № 2, є обернено-впорядкований :

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
initial state of the array(before sorting by Shell_2):  
Section 0:  
96 92 88 84  
95 91 87 83  
94 90 86 82  
93 89 85 81  
  
Section 1:  
80 76 72 68  
79 75 71 67  
78 74 70 66  
77 73 69 65  
  
Section 2:  
64 60 56 52  
63 59 55 51  
62 58 54 50  
61 57 53 49  
  
Section 3:  
48 44 40 36  
47 43 39 35  
46 42 38 34  
45 41 37 33  
  
Section 4:  
32 28 24 20  
31 27 23 19  
30 26 22 18  
29 25 21 17  
  
Section 5:  
16 12 8 4  
15 11 7 3  
14 10 6 2  
13 9 5 1
```

Стан 3D – масиву після сортування методом Шелла № 2:

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
state of the array(after sorting by Shell_2):  
Section 0:  
16 12 8 4  
15 11 7 3  
14 10 6 2  
13 9 5 1  
  
Section 1:  
32 28 24 20  
31 27 23 19  
30 26 22 18  
29 25 21 17  
  
Section 2:  
48 44 40 36  
47 43 39 35  
46 42 38 34  
45 41 37 33  
  
Section 3:  
64 60 56 52  
63 59 55 51  
62 58 54 50  
61 57 53 49  
  
Section 4:  
80 76 72 68  
79 75 71 67  
78 74 70 66  
77 73 69 65  
  
Section 5:  
96 92 88 84  
95 91 87 83  
94 90 86 82  
93 89 85 81
```



Початковий стан 3D-масиву перед сортуванням методом Шелла № 2, є випадково-впорядкований:

```
C:\code_blocks\COURSE_WOR  X  +  v
initial state of the array(before sorting by Shell_2):
Section 0:
41 65 82 49
35 76 80 27
94 54 41 73
4 78 17 11

Section 1:
19 39 4 38
38 12 94 23
27 62 45 71
60 57 92 30

Section 2:
83 35 95 9
18 91 3 56
45 43 26 69
40 6 93 31

Section 3:
29 22 83 28
52 21 45 59
11 69 73 14
92 91 10 18

Section 4:
0 64 50 30
58 88 9 93
16 86 82 9
14 93 86 92

Section 5:
45 36 11 50
47 32 4 91
84 70 48 65
82 80 87 34
```

Стан 3D – масиву після сортування методом Шелла № 2:

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
state of the array(after sorting by Shell_2):  
Section 0:  
0 64 50 30  
58 88 9 93  
16 86 82 9  
14 93 86 92  
  
Section 1:  
19 39 4 38  
38 12 94 23  
27 62 45 71  
60 57 92 30  
  
Section 2:  
29 22 83 28  
52 21 45 59  
11 69 73 14  
92 91 10 18  
  
Section 3:  
41 65 82 49  
35 76 80 27  
94 54 41 73  
4 78 17 11  
  
Section 4:  
45 36 11 50  
47 32 4 91  
84 70 48 65  
82 80 87 34  
  
Section 5:  
83 35 95 9  
18 91 3 56  
45 43 26 69  
40 6 93 31
```

Початковий стан 3D-масиву перед сортуванням методом Шелла № 2, є  
прямо-впорядкований:

```
C:\code_blocks\COURSE_WOR  ×  +  ∨  
  
initial state of the array(before sorting by Shell_2):  
Section 0:  
0 4 8 12  
1 5 9 13  
2 6 10 14  
3 7 11 15  
  
Section 1:  
16 20 24 28  
17 21 25 29  
18 22 26 30  
19 23 27 31  
  
Section 2:  
32 36 40 44  
33 37 41 45  
34 38 42 46  
35 39 43 47  
  
Section 3:  
48 52 56 60  
49 53 57 61  
50 54 58 62  
51 55 59 63  
  
Section 4:  
64 68 72 76  
65 69 73 77  
66 70 74 78  
67 71 75 79  
  
Section 5:  
80 84 88 92  
81 85 89 93  
82 86 90 94  
83 87 91 95
```

Стан 3D – масиву після сортування методом Шелла № 2:

```
C:\code_blocks\COURSE_WOR  ×  +  ▾  
state of the array(after sorting by Shell_2):  
Section 0:  
0 4 8 12  
1 5 9 13  
2 6 10 14  
3 7 11 15  
  
Section 1:  
16 20 24 28  
17 21 25 29  
18 22 26 30  
19 23 27 31  
  
Section 2:  
32 36 40 44  
33 37 41 45  
34 38 42 46  
35 39 43 47  
  
Section 3:  
48 52 56 60  
49 53 57 61  
50 54 58 62  
51 55 59 63  
  
Section 4:  
64 68 72 76  
65 69 73 77  
66 70 74 78  
67 71 75 79  
  
Section 5:  
80 84 88 92  
81 85 89 93  
82 86 90 94  
83 87 91 95
```

Початковий стан вектору перед сортуванням методом прямого вибору № 1, є обернено-впорядкований:

```
C:\code_blocks\COURSE_WOR x + v
initial state of the vector(before sorting by Select_1):
6      5      4      3      2      1
Process returned 0 (0x0)    execution time : 0.059 s
Press any key to continue.
```

Стан вектору після сортування методом прямого вибору № 1:

```
C:\code_blocks\COURSE_WOR x + v
state of the vector(after sorting by Select_1):
1      2      3      4      5      6
Process returned 0 (0x0)    execution time : 0.090 s
Press any key to continue.
```

Початковий стан вектору перед сортуванням методом прямого вибору № 1, є випадково-впорядкований:

```
C:\code_blocks\COURSE_WOR x + v
initial state of the vector(before sorting by Select_1):
5      5      4      4      5      4
Process returned 0 (0x0)    execution time : 0.076 s
Press any key to continue.
```

Стан вектору після сортування методом прямого вибору № 1:

```
state of the vector(after sorting by Select_1):
4      4      4      5      5      5
Process returned 0 (0x0)    execution time : 0.070 s
Press any key to continue.
```

Початковий стан вектору перед сортуванням методом прямого вибору № 1, є прямо-впорядкований:

```
C:\code_blocks\COURSE_WOR × + v
initial state of the vector(before sorting by Select_1):
1      2      3      4      5      6
Process returned 0 (0x0)    execution time : 0.108 s
```

Стан вектору після сортування методом прямого вибору № 1:

```
C:\code_blocks\COURSE_WOR × + v
state of the vector(after sorting by Select_1):
1      2      3      4      5      6
Process returned 0 (0x0)    execution time : 0.067 s
Press any key to continue.
```

Початковий стан вектору перед сортуванням методом прямого вибору № 8, є обернено-впорядкований:

```
C:\code_blocks\COURSE_WOR × + v
initial state of the vector(before sorting by Select_8):
6      5      4      3      2      1
Process returned 0 (0x0)    execution time : 0.080 s
Press any key to continue.
```

Стан вектору після сортування методом прямого вибору № 8:

```
C:\code_blocks\COURSE_WOR × + v
state of the vector(after sorting by Select_8):
1      2      3      4      5      6
Process returned 0 (0x0)    execution time : 0.084 s
Press any key to continue.
```

Початковий стан вектору перед сортуванням методом прямого вибору № 8, є випадково-впорядкований:

```
C:\code_blocks\COURSE_WOR × + ∨  
initial state of the vector(before sorting by Select_8):  
5      5      4      4      5      4  
Process returned 0 (0x0)    execution time : 0.095 s  
Press any key to continue.
```

Стан вектору після сортування методом прямого вибору № 8:

```
C:\code_blocks\COURSE_WOR × + ∨  
state of the vector(after sorting by Select_8):  
4      4      4      5      5      5  
Process returned 0 (0x0)    execution time : 0.080 s  
Press any key to continue.
```

Початковий стан вектору перед сортуванням методом прямого вибору № 8, є прямо-впорядкований:

```
C:\code_blocks\COURSE_WOR × + ∨  
initial state of the vector(before sorting by Select_8)  
1      2      3      4      5      6  
Process returned 0 (0x0)    execution time : 0.063 s  
Press any key to continue.
```

Стан вектору після сортування методом прямого вибору № 8:

```
C:\code_blocks\COURSE_WOR × + ∨  
state of the vector(after sorting by Select_8):  
1      2      3      4      5      6  
Process returned 0 (0x0)    execution time : 0.092 s  
Press any key to continue.
```

Початковий стан вектору перед сортуванням методом Шелла № 2, є обернено-впорядкований:

```
C:\code_blocks\COURSE_WOR × + ▾  
initial state of the vector(before sorting by Shell_2):  
6      5      4      3      2      1  
Process returned 0 (0x0)    execution time : 0.082 s  
Press any key to continue.  
|
```

Стан вектору після сортування методом Шелла № 2:

```
C:\code_blocks\COURSE_WOR × + ▾  
state of the vector(after sorting by Shell_2):  
1      2      3      4      5      6  
Process returned 0 (0x0)    execution time : 0.090 s  
Press any key to continue.  
|
```

Початковий стан вектору перед сортуванням методом Шелла № 2, є випадково-впорядкований:

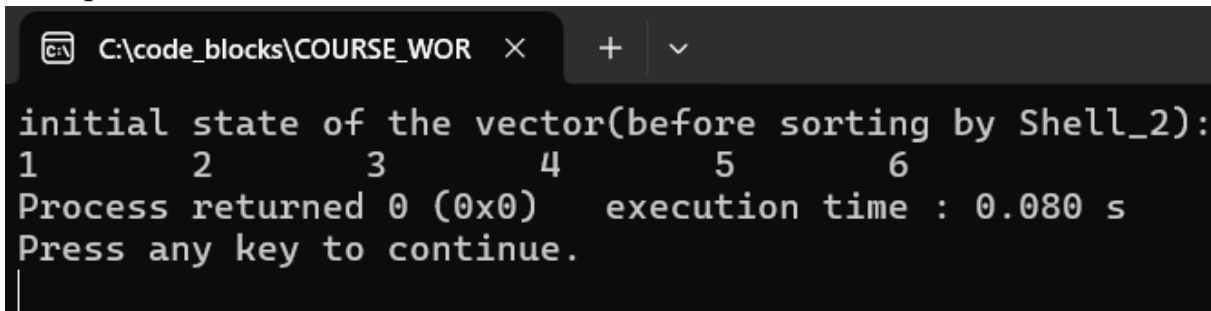
```
C:\code_blocks\COURSE_WOR × + ▾  
initial state of the vector(before sorting by Shell_2):  
5      5      4      4      5      4  
Process returned 0 (0x0)    execution time : 0.077 s  
Press any key to continue.  
|
```

Стан вектору після сортування методом Шелла № 2:

```
C:\code_blocks\COURSE_WOR × + ▾  
state of the vector(after sorting by Shell_2):  
4      4      4      5      5      5  
Process returned 0 (0x0)    execution time : 0.081 s  
Press any key to continue.  
|
```

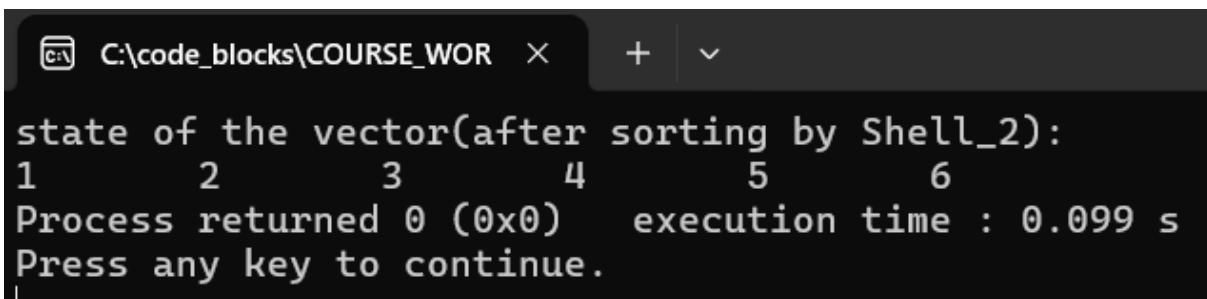


Початковий стан вектору перед сортуванням методом Шелла № 2, є прямо-впорядкований:



```
C:\code_blocks\COURSE_WOR x + v
initial state of the vector(before sorting by Shell_2):
1      2      3      4      5      6
Process returned 0 (0x0)    execution time : 0.080 s
Press any key to continue.
```

Стан вектору після сортування методом Шелла № 2:



```
C:\code_blocks\COURSE_WOR x + v
state of the vector(after sorting by Shell_2):
1      2      3      4      5      6
Process returned 0 (0x0)    execution time : 0.099 s
Press any key to continue.
```

На основі результатів проведеного тестування, виходить логічний висновок, усі 3 алгоритми виконують сортування згідно даної за варіантом задачі та способу обходу.

Впевнившись у правильності роботи алгоритмів, можна переходити до практичних досліджень швидкодії сортування даними алгоритмами.

## Результати практичних досліджень

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

Довжини вимірів 3 – D масиву:

1)  $P = \text{const} = 25000$ ,  $M = 1$ ,  $N = 1$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	327.90	514.00	847.10
Select8, спосіб обходу № 7	991.25	978.40	620.15
Shell_2, спосіб обходу № 7	0.00	16.35	1.60

2)  $P = \text{const} = 25000$ ,  $M = 2$ ,  $N = 2$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	536.60	913.00	1189.05
Select8, спосіб обходу № 7	1094.90	1011.15	684.60
Shell_2, спосіб обходу № 7	0.00	32.60	9.75

3)  $P = \text{const} = 25000$ ,  $M = 4$ ,  $N = 4$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	1650.95	1791.75	1363.80
Select8, спосіб обходу № 7	792.65	800.10	551.60
Shell_2, спосіб обходу № 7	0.55	50.95	15.80

4)  $P = \text{const} = 25000$ ,  $M = 8$ ,  $N = 8$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	1621.20	1668.35	1754.80
Select8, спосіб обходу № 7	1196.10	1337.20	1013.35
Shell_2, спосіб обходу № 7	2.05	254.45	65.70

5)  $P = \text{const} = 25000$ ,  $M = 16$ ,  $N = 16$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	3974.15	3884.75	4358.25
Select8, спосіб обходу № 7	3745.95	3673.05	1243.10
Shell_2, спосіб обходу № 7	3.30	892.15	263.35

6)  $P = \text{const} = 25000$ ,  $M = 32$ ,  $N = 32$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	4334.25	3324.50	3458.20
Select8, спосіб обходу № 7	1693.00	2888.35	1259.00
Shell_2, спосіб обходу № 7	3.30	2694.10	1431.30

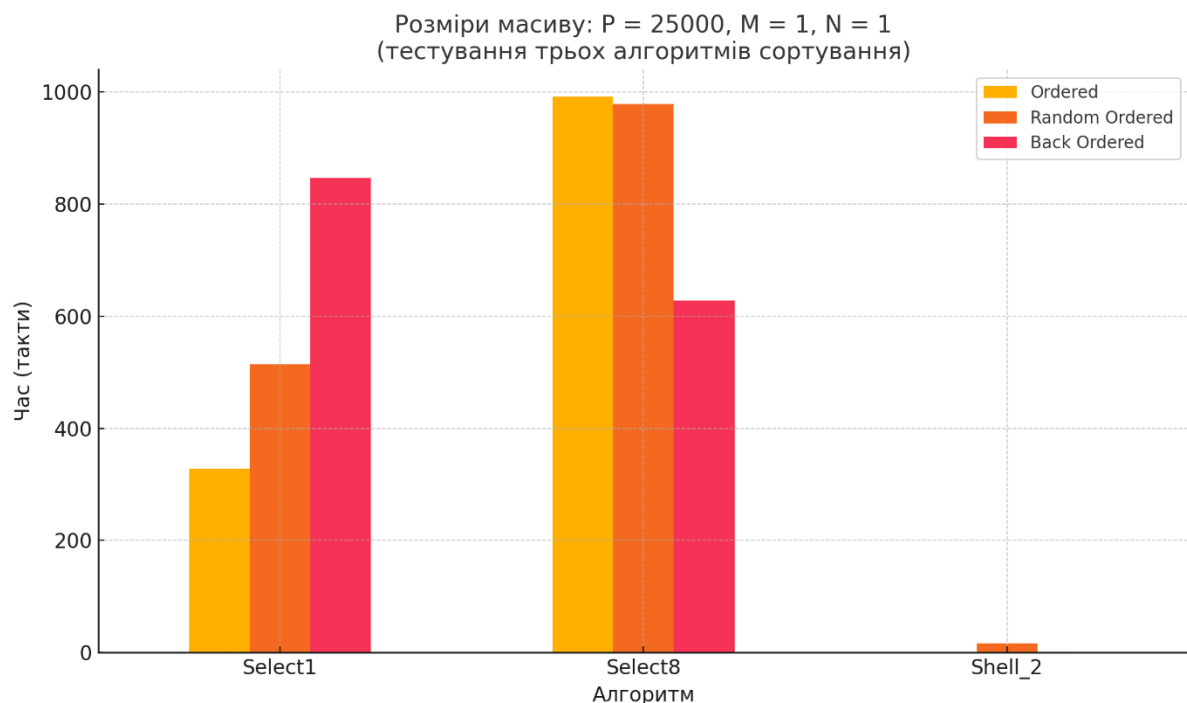
7)  $P = \text{const} = 25000$ ,  $M = 64$ ,  $N = 64$ :

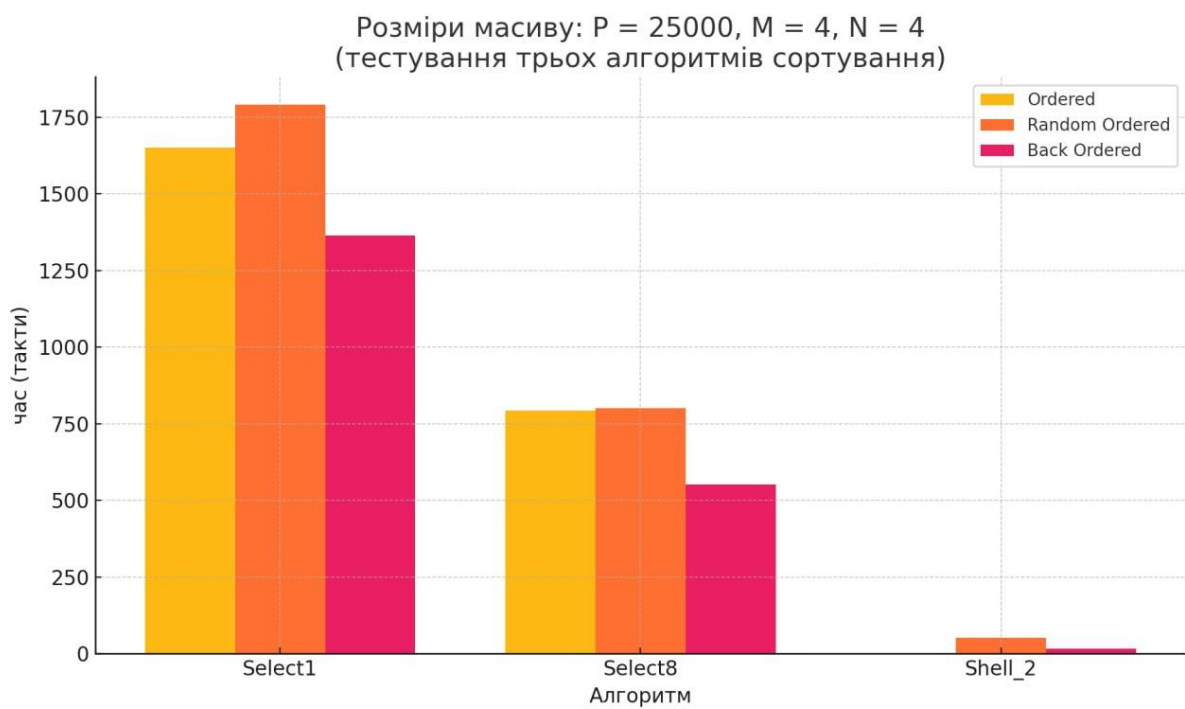
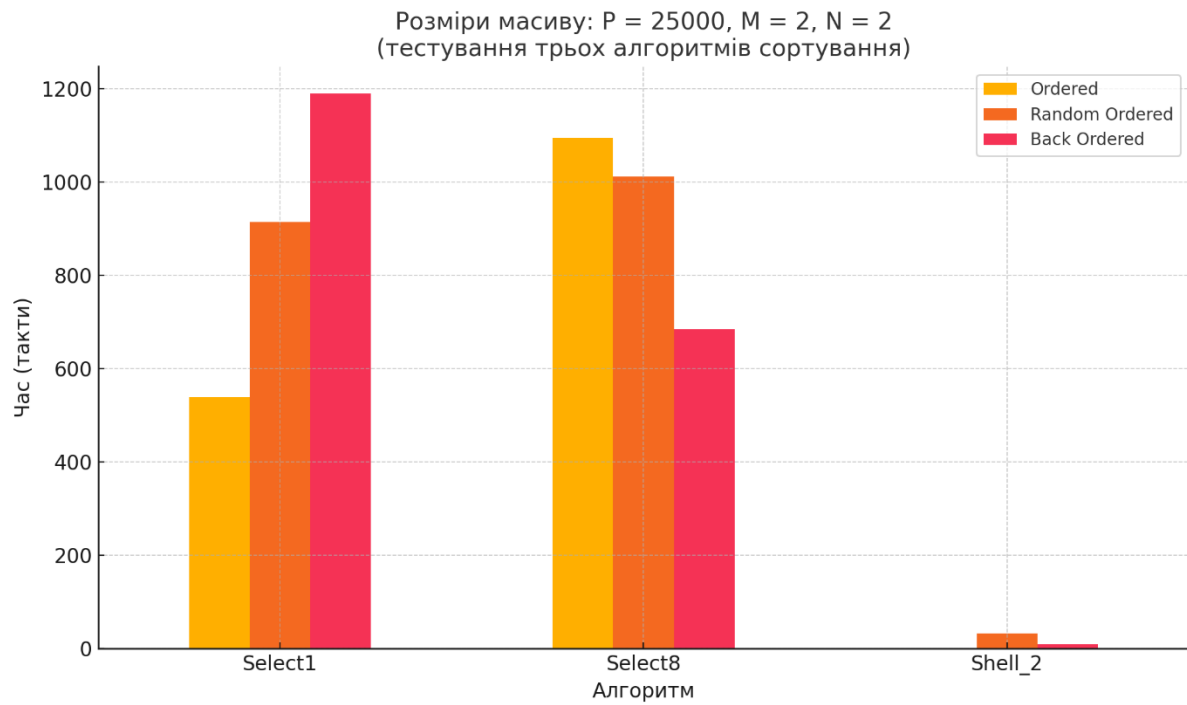
	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	4694.85	3624.64	4358.40
Select8, спосіб обходу № 7	2846.00	3865.22	1800.45
Shell_2, спосіб обходу № 7	4.10	2849.46	1662.23

Для додаткового порівняння було проведено вимірювання часу сортування трьома заданими за варіантом алгоритмами, вектора розміром  $VECTOR = P = 25000$  елементів:

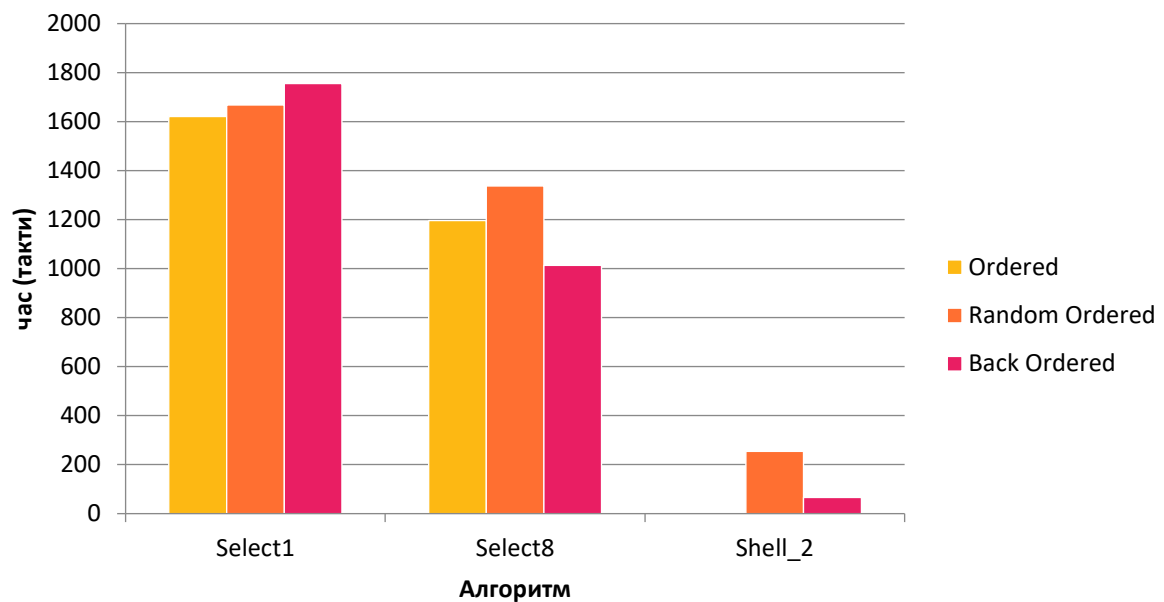
	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	616.40	635.30	714.00
Select8, спосіб обходу № 7	691.35	673.50	519.90
Shell_2, спосіб обходу № 7	0.00	8.65	0.20

Візуалізація результатів вимірювання за допомогою діаграм та графіків для випадку дослідження № 1:

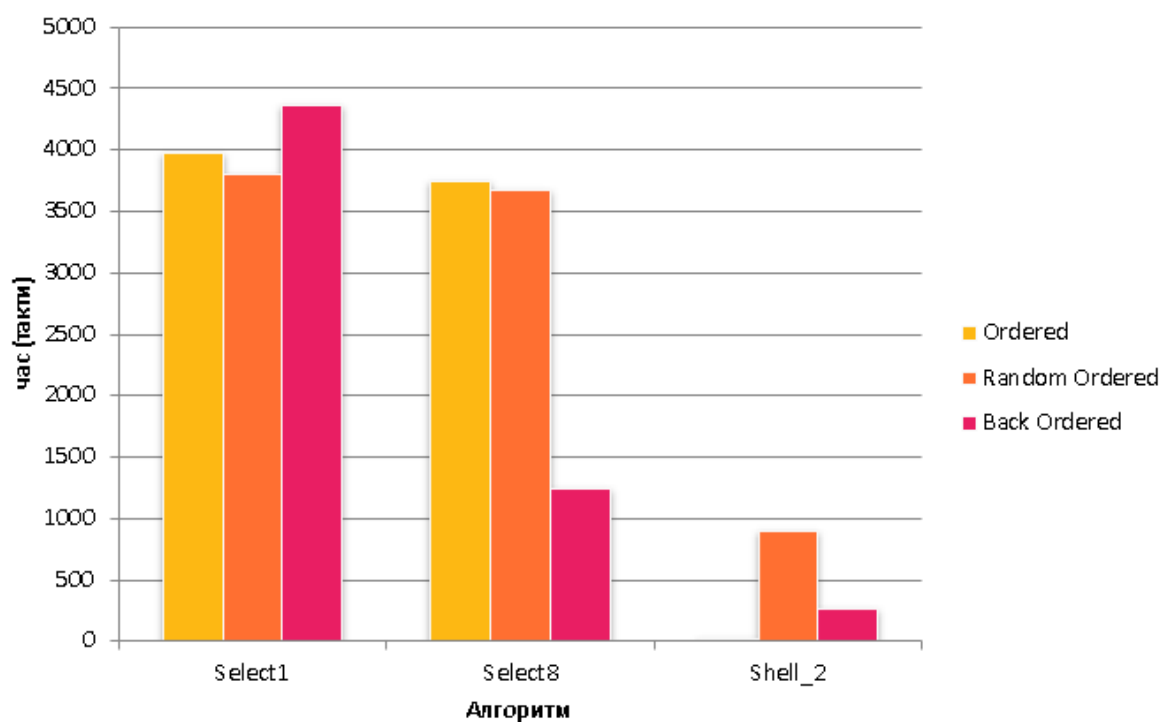




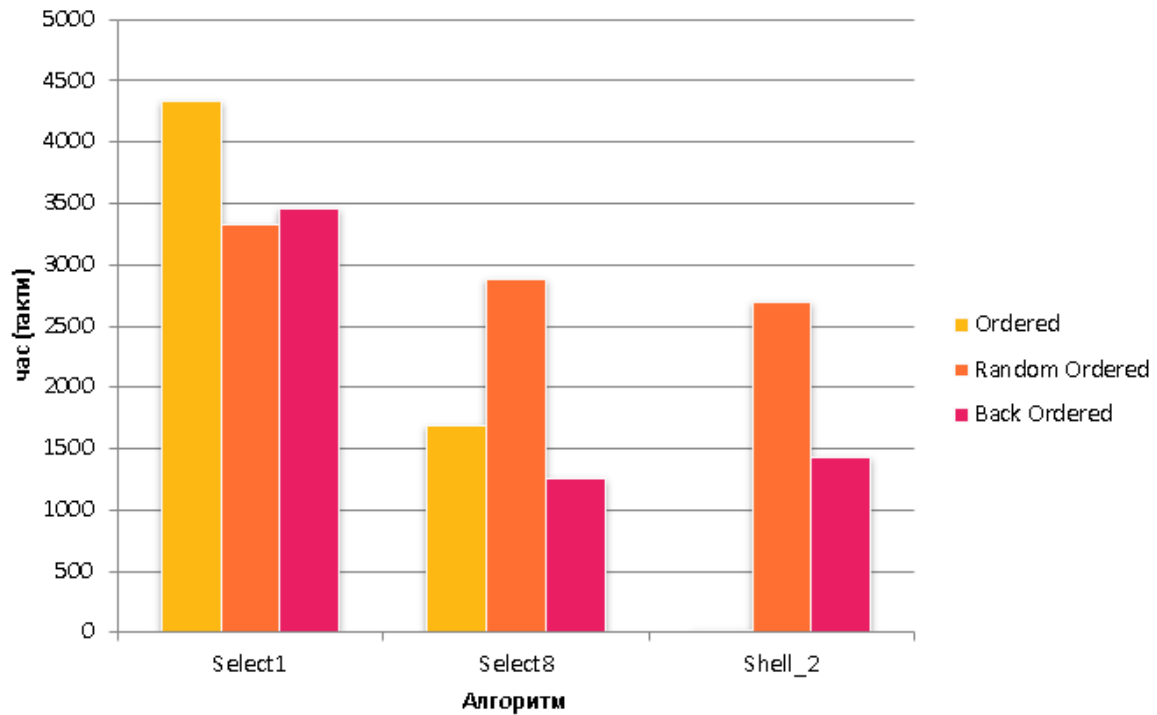
**Розміри масиву: P = 25000, M = 8, N = 8  
(тестування трьох алгоритмів сортування)**



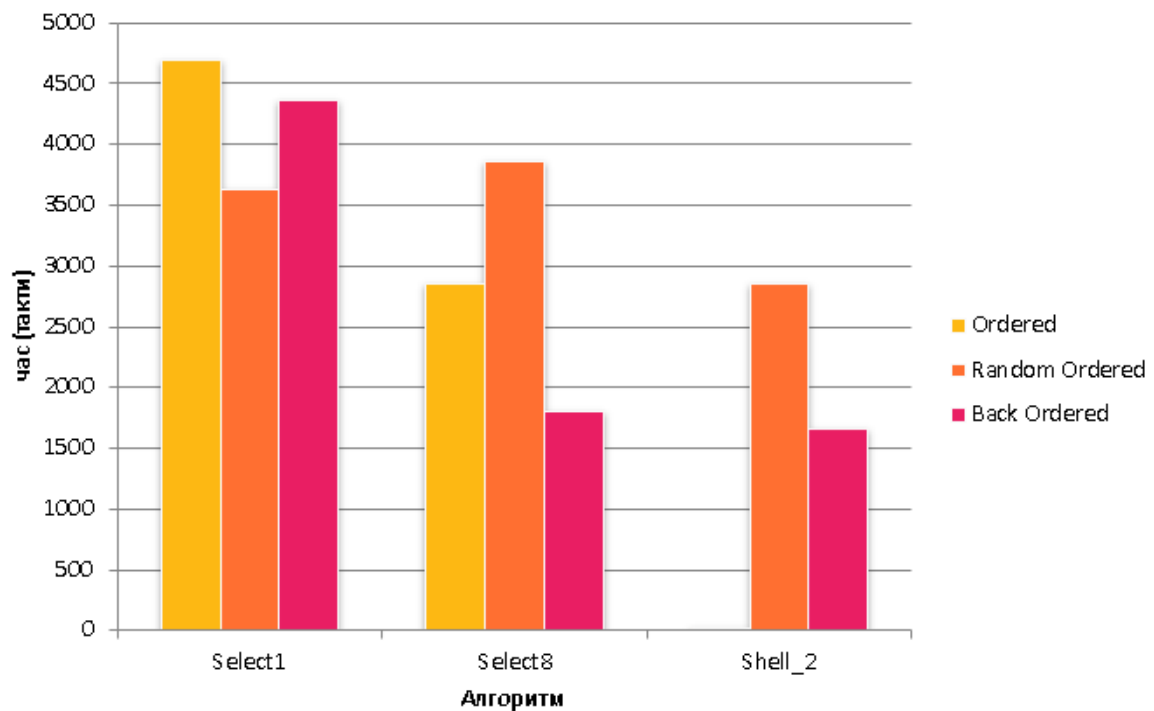
**Розміри масива: P = 25000, M = 16, N = 16  
(тестування трьох алгоритмів сортування)**



**Розміри масива:  $P = 25000$ ,  $M = 32$ ,  $N = 32$**   
**(тестування трьох алгоритмів сортування)**

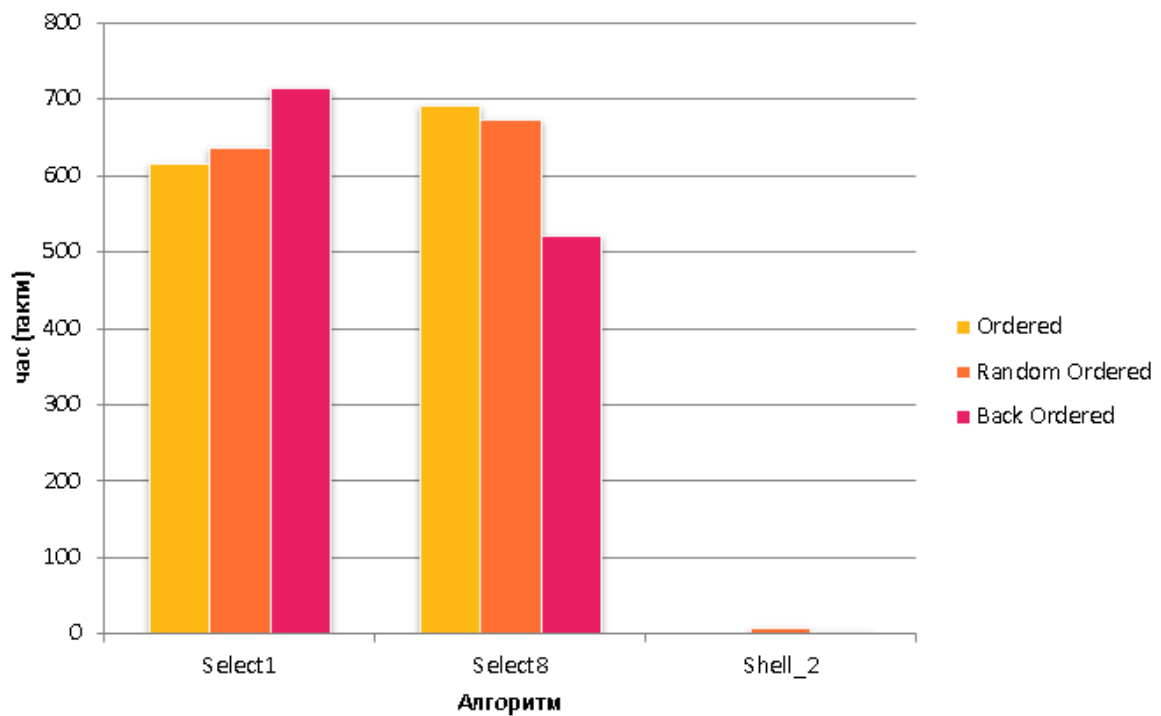


**Розміри масива:  $P = 25000$ ,  $M = 64$ ,  $N = 64$**   
**(тестування трьох алгоритмів сортування)**

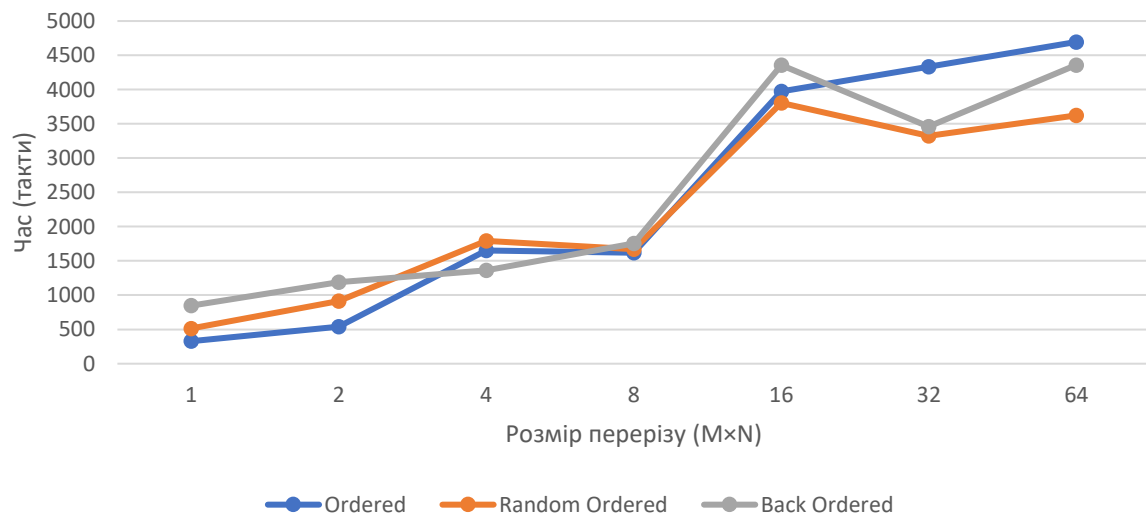


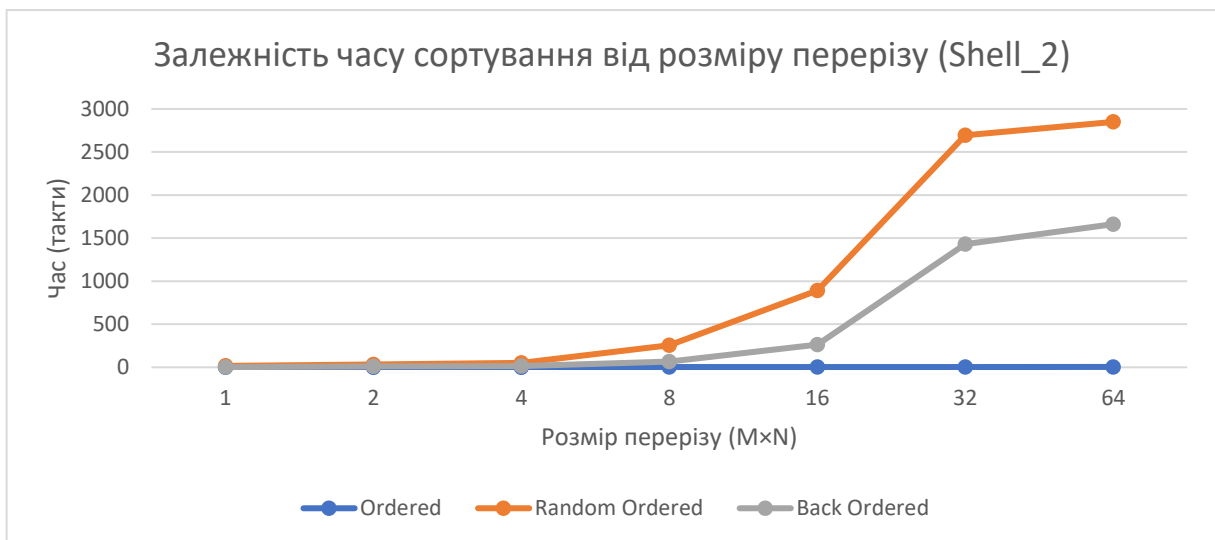
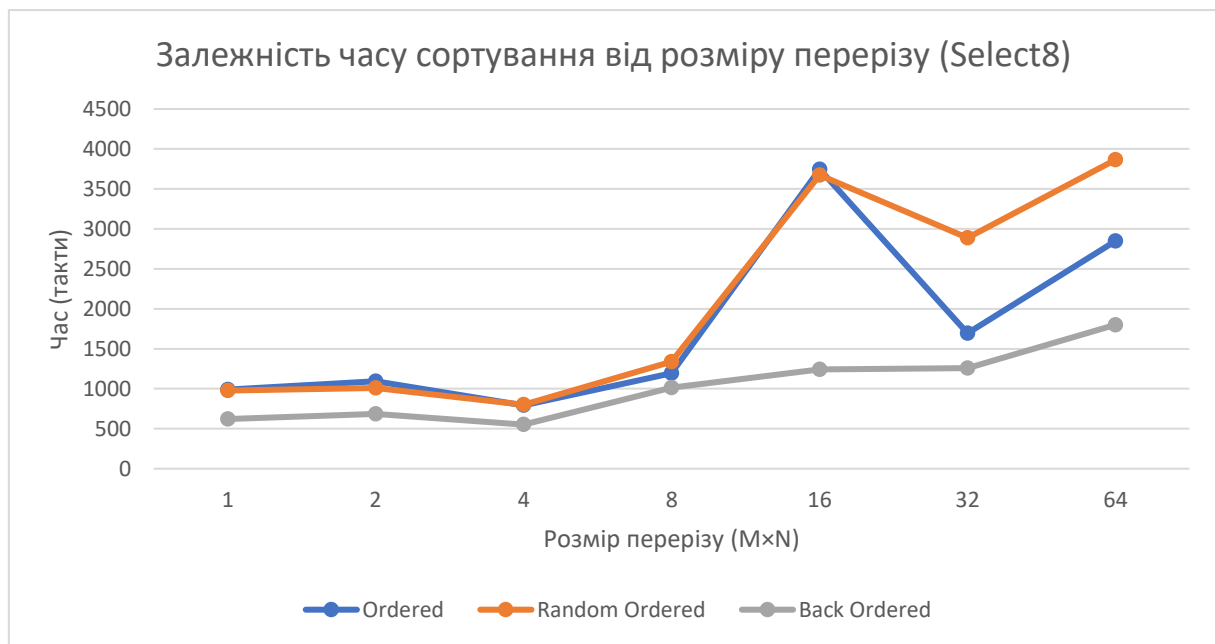


### Розміри вектора: P = 25000 (тестування трьох алгоритмів сортування)



### Залежність часу сортування від розміру перерізу (Select1)





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

Довжини вимірів 3 – D масиву:

1)  $P = 32000$ ;  $M = 2$ ;  $N = 800$  :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	2508.50	2524.30	2750.75
Select8, спосіб обходу № 7	2061.60	2265.95	1543.45
Shell_2, спосіб обходу № 7	1.25	6950.20	1910.20

2)  $P = 32000$ ;  $M = 4$ ;  $N = 400$ :

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	4363.15	3784.85	2880.30
Select8, спосіб обходу № 7	2280.95	2372.35	1610.10
Shell_2, спосіб обходу № 7	2.15	7153.65	3270.05

3) P = 32000; M = 8; N = 200:

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	2652.80	2662.70	2941.65
Select8, спосіб обходу № 7	2310.95	2444.85	1633.15
Shell_2, спосіб обходу № 7	3.30	7079.15	2005.75

4) P = 32000; M = 16; N = 100:

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	3429.20	3885.00	3331.60
Select8, спосіб обходу № 7	2466.00	2572.30	2110.70
Shell_2, спосіб обходу № 7	14.65	11115.60	3419.95

5) P = 32000; M = 100; N = 16:

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	7200.25	9475.65	15178.25
Select8, спосіб обходу № 7	11219.70	12729.35	8117.95
Shell_2, спосіб обходу № 7	17.30	13791.05	4078.10

6) P = 32000; M = 200; N = 8:

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	4030.15	4029.50	4282.85
Select8, спосіб обходу № 7	3466.30	3240.30	2287.35
Shell_2, спосіб обходу № 7	5.00	10393.80	4269.60

7) P = 32000; M = 400; N = 4:

	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	4730.00	4754.80	5275.10
Select8, спосіб обходу № 7	13520.90	5544.25	2541.40
Shell_2, спосіб обходу № 7	4.60	8189.75	2557.75

8) P = 32000; M = 800; N = 2:

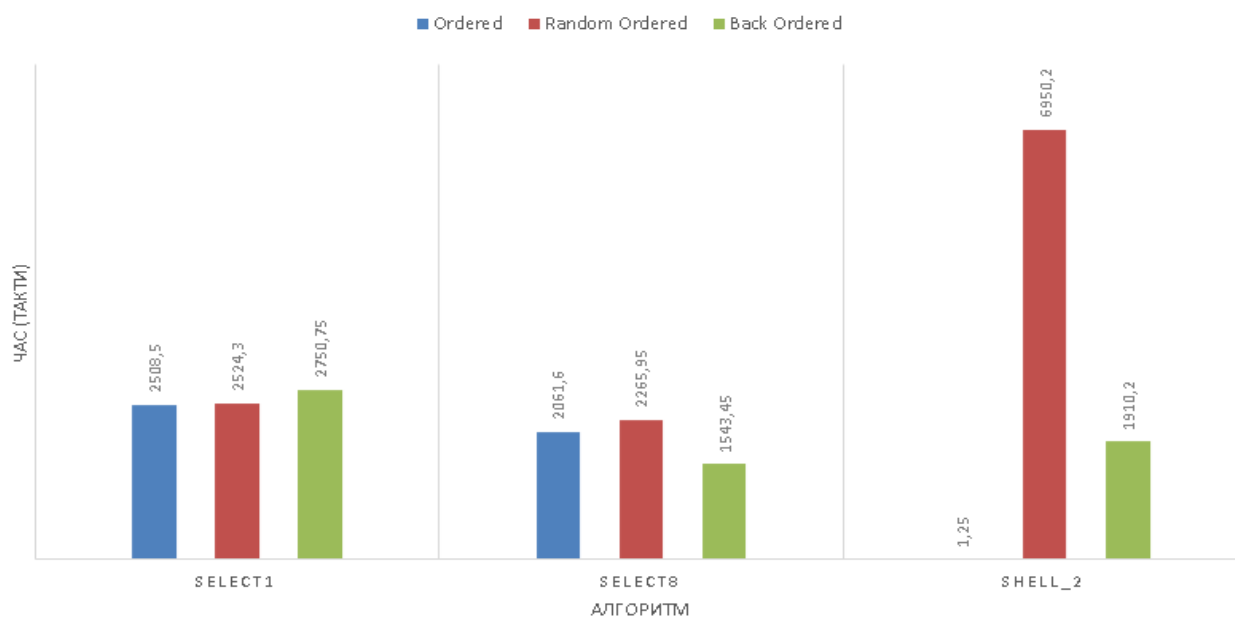
	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	10200.35	10677.15	17208.55
Select8, спосіб обходу № 7	15933.90	3595.15	2829.20
Shell_2, спосіб обходу № 7	3.50	7963.25	2381.80

Для додаткового порівняння було проведено вимірювання часу сортування трьома заданими за варіантом алгоритмами, вектора розміром  $VECTOR = P = 32000$  елементів:

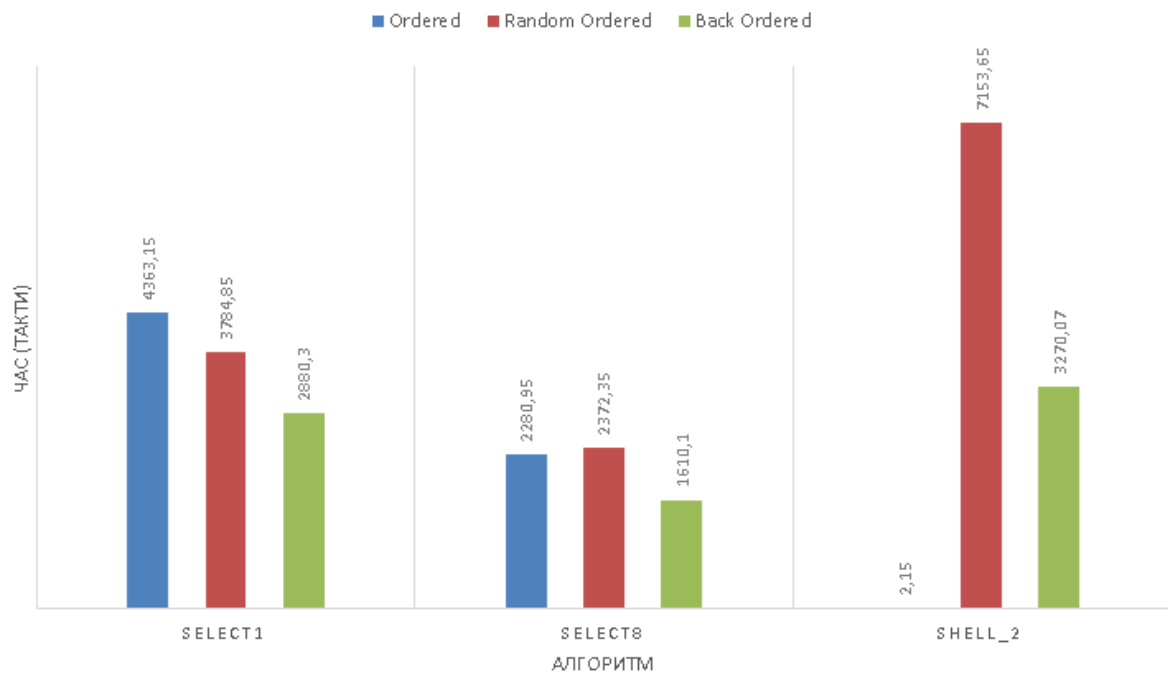
	Впорядкований	Невпорядкований	Обернено впорядкований
Select1, спосіб обходу № 7	386.75	386.65	522.90
Select8, спосіб обходу № 7	507.85	518.75	312.10
Shell_2, спосіб обходу № 7	0.00	7.30	0.00

Візуалізація результатів вимірювання за допомогою діаграм та графіків для випадку дослідження № 2:

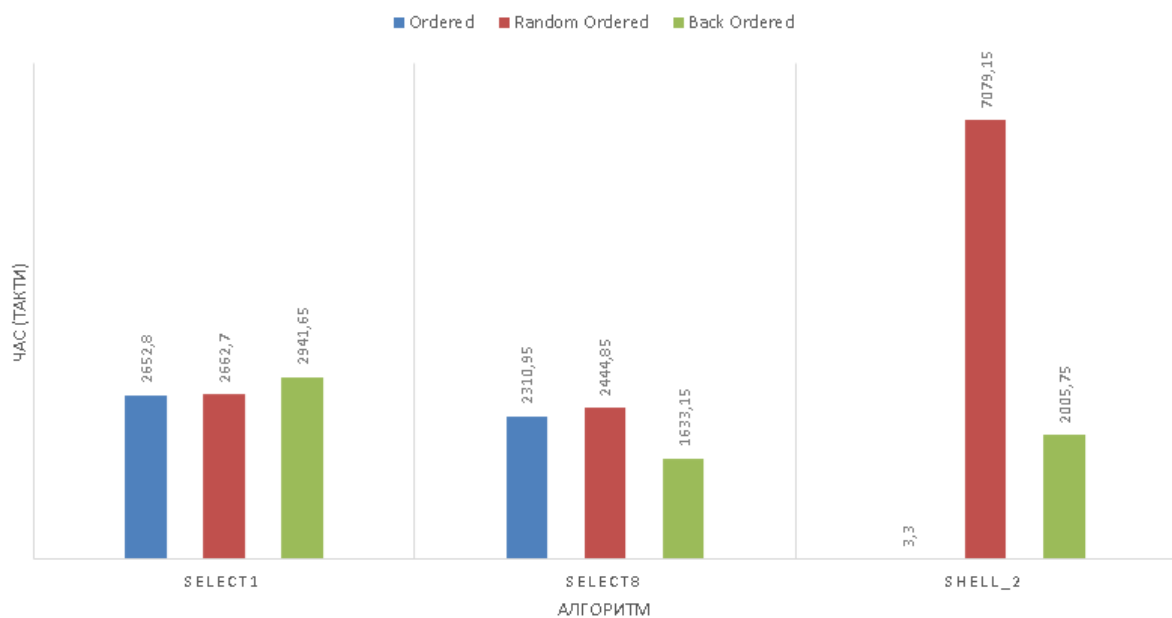
**РОЗМІРИ МАСИВУ:  $P = 32000$ ,  $M = 2$ ,  $N = 800$   
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



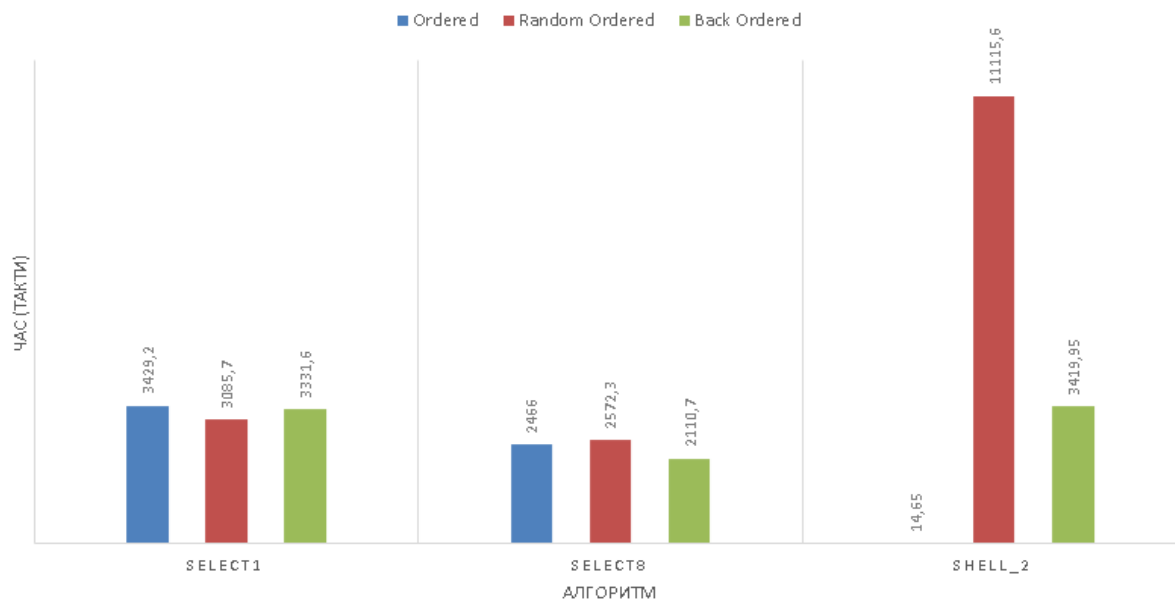
**РОЗМІРИ МАСИВУ: P = 32000, M = 4 , N = 400  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



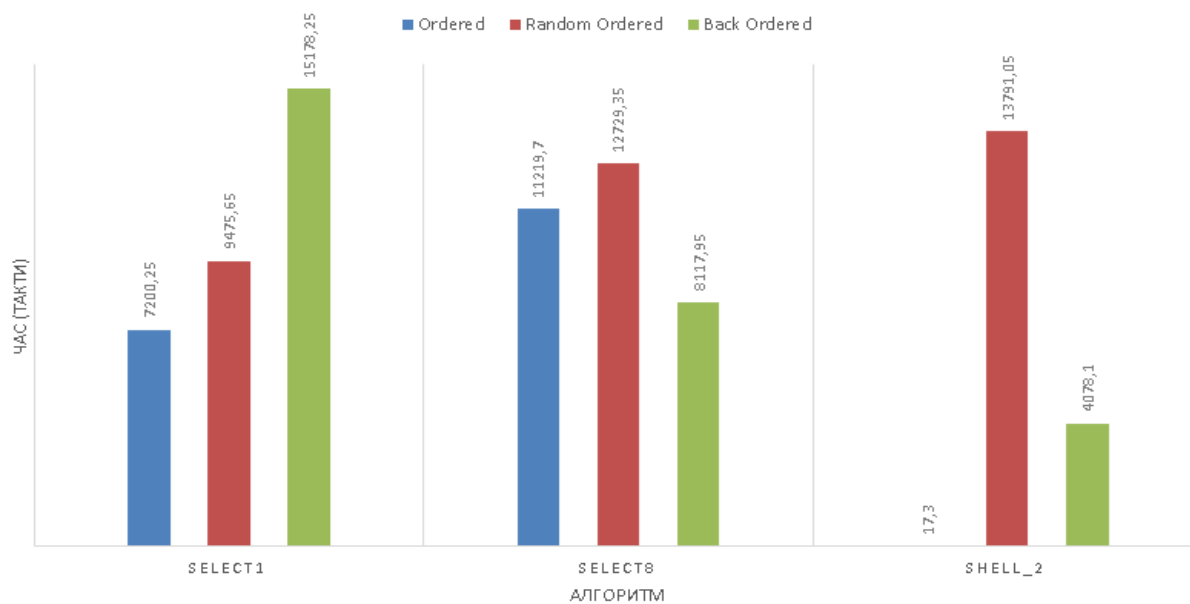
**РОЗМІРИ МАСИВУ: P = 32000, M = 8 , N = 200  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



**РОЗМІРИ МАСИВУ: P = 32000, M = 16 , N = 100  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**

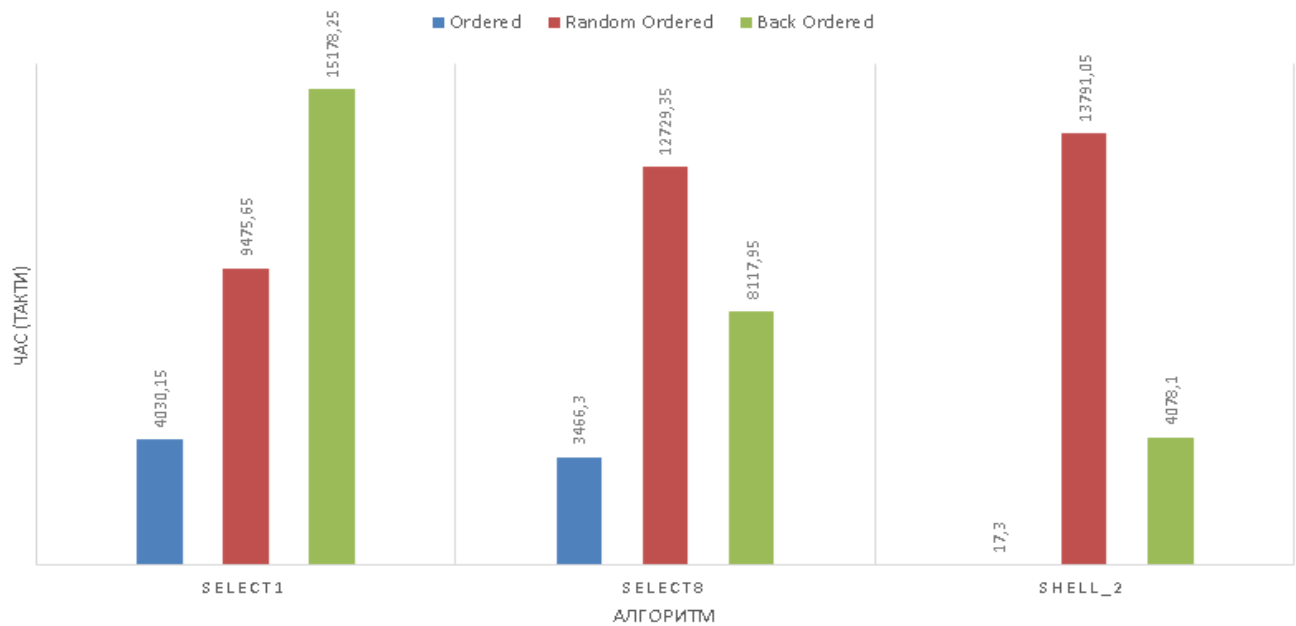


**РОЗМІРИ МАСИВУ: P = 32000, M = 100 , N = 16  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**

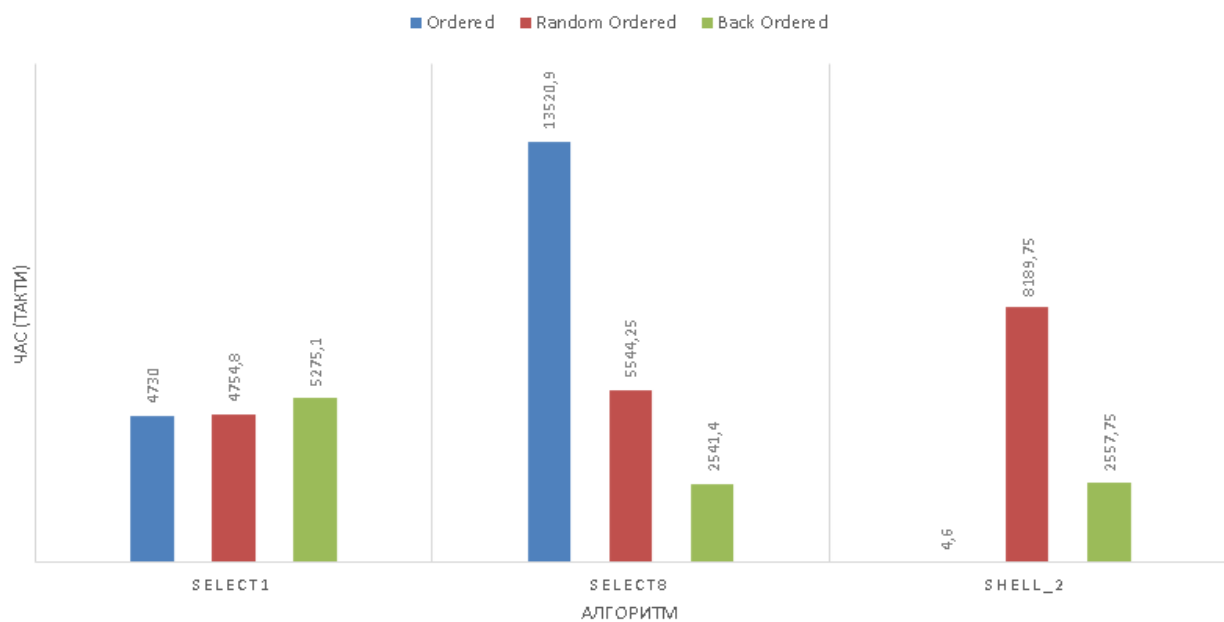




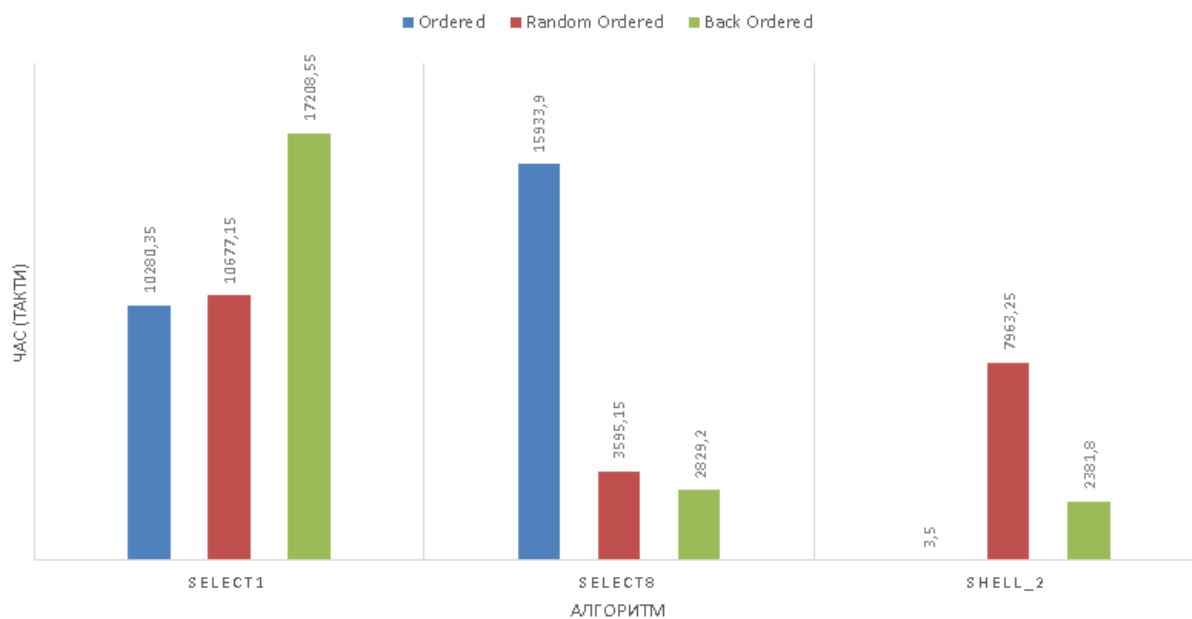
**РОЗМІРИ МАСИВУ: P = 32000, M = 200, N = 8  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



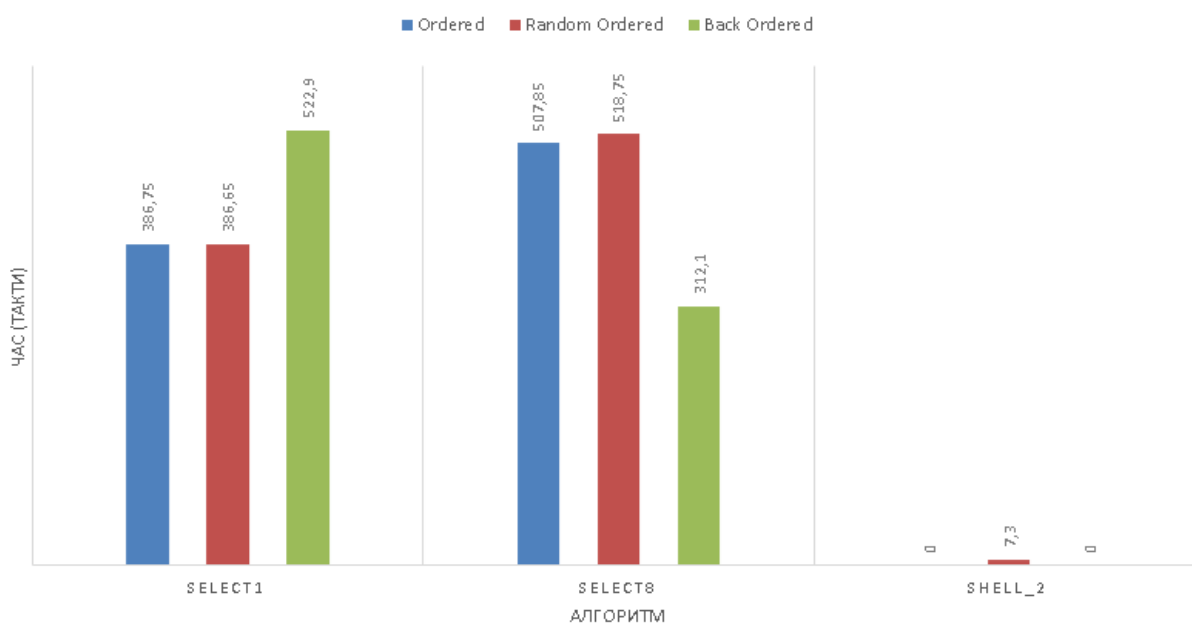
**РОЗМІРИ МАСИВУ: P = 32000, M = 400, N = 4  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



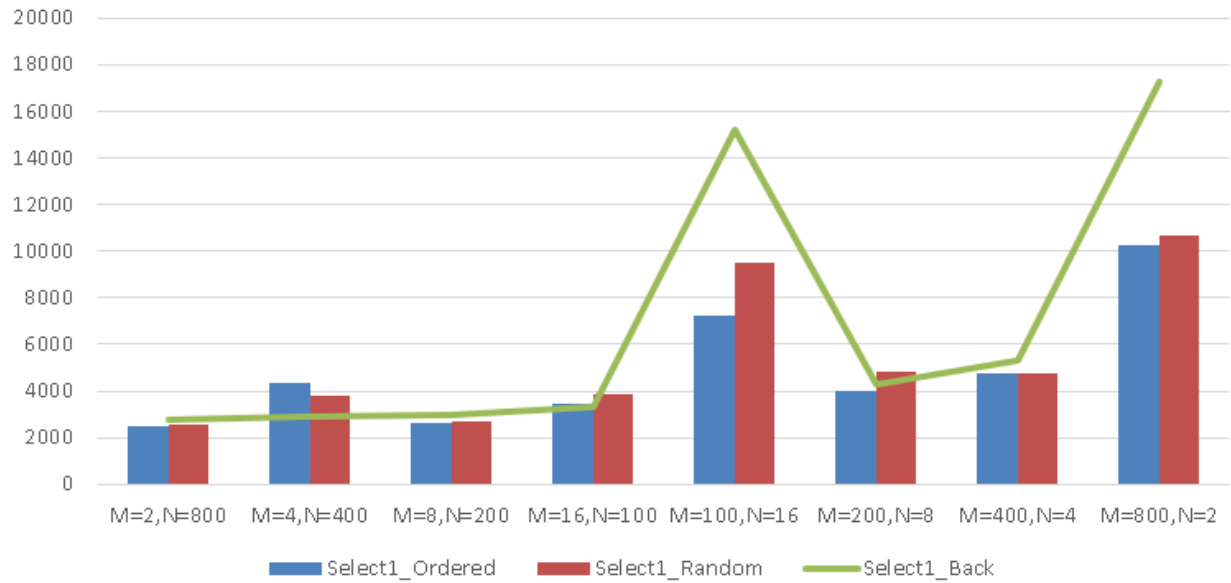
**РОЗМІРИ МАСИВУ: P = 32000, M = 800 , N = 2  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



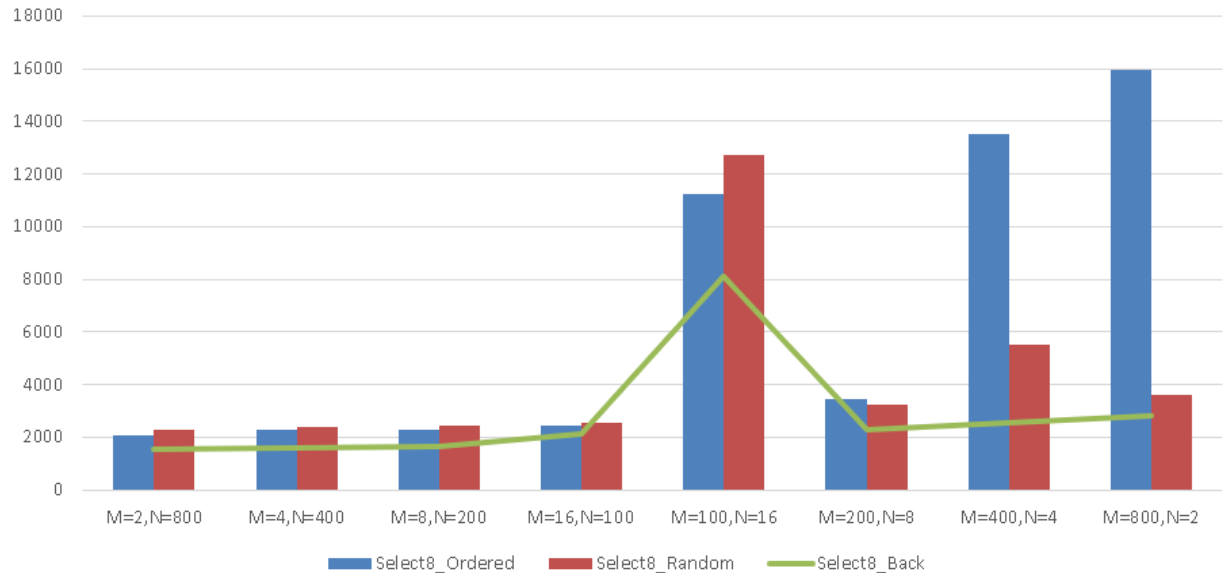
**РОЗМІР ВЕКТОРА: P = 32000  
(ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)**



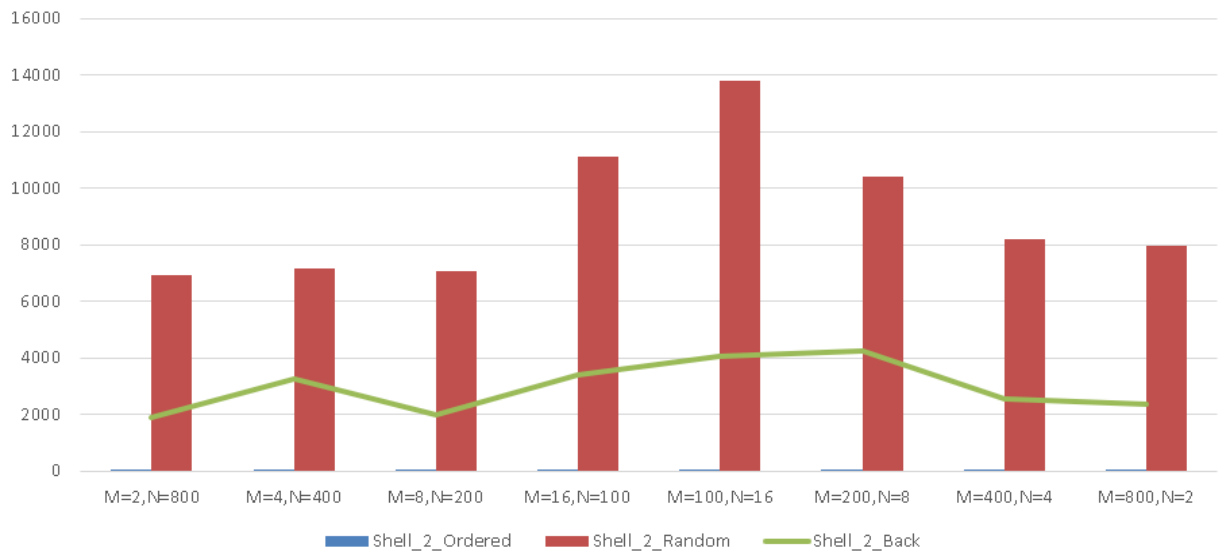
Залежність часу сортування від форми перерізу (Select1)



Залежність часу сортування від форми перерізу (Select8)



Залежність часу сортування від форми перерізу (Shell\_2)



## Порівняльний аналіз алгоритмів

У ході написання курсової роботи, було досліджено залежність часу сортування тривимірному масиву  $Arr3D$ , для трьох випадків відсортованості (прямо-впорядкований, випадково-впорядкований, обернено-впорядкований), від розміру перерізів цього масиву, а також від форми його перерізів.

Згідно варіанту масив сортувався трьома різними алгоритмами, а саме:

- 1) Алгоритм сортування № 1 на основі методу прямого вибору;
- 2) Алгоритм сортування № 8 на основі методу прямого вибору;
- 3) Алгоритм сортування № 2 на основі методу Шелла;

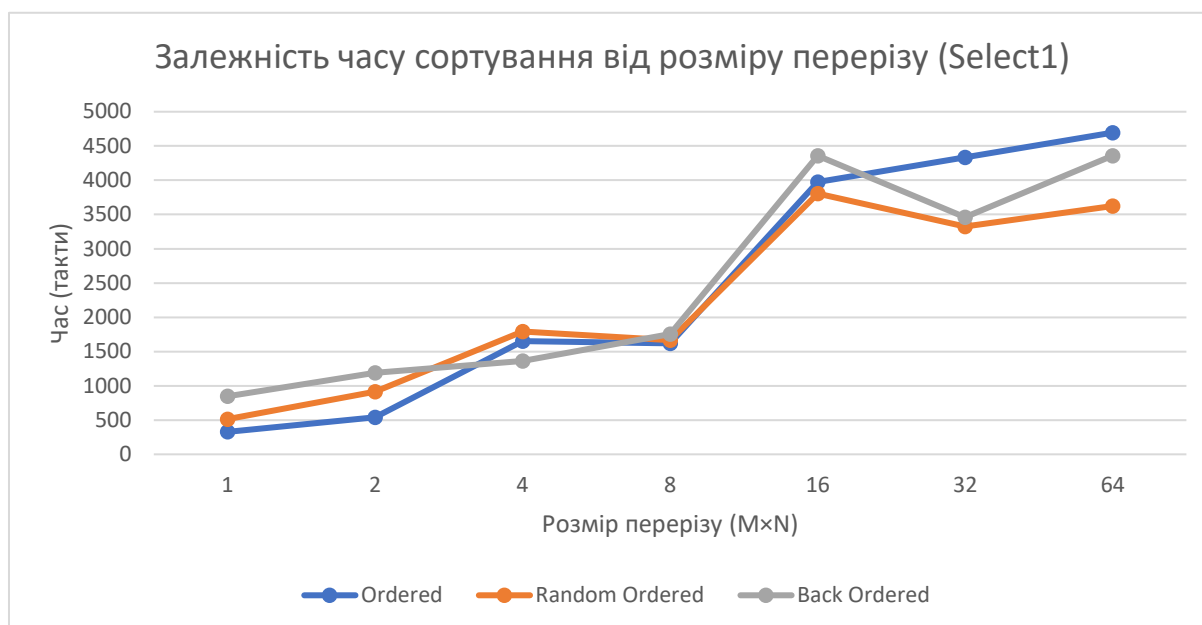
При дослідженні швидкодії всіх алгоритмів використовувався, один і той самий, спосіб обходу по елементах масиву, заданий згідно варіанту №119.

Аналіз отриманих результатів для двох випадків дослідження (дослідження швидкодії алгоритмів залежно від розміру перерізів та від форми перерізів):

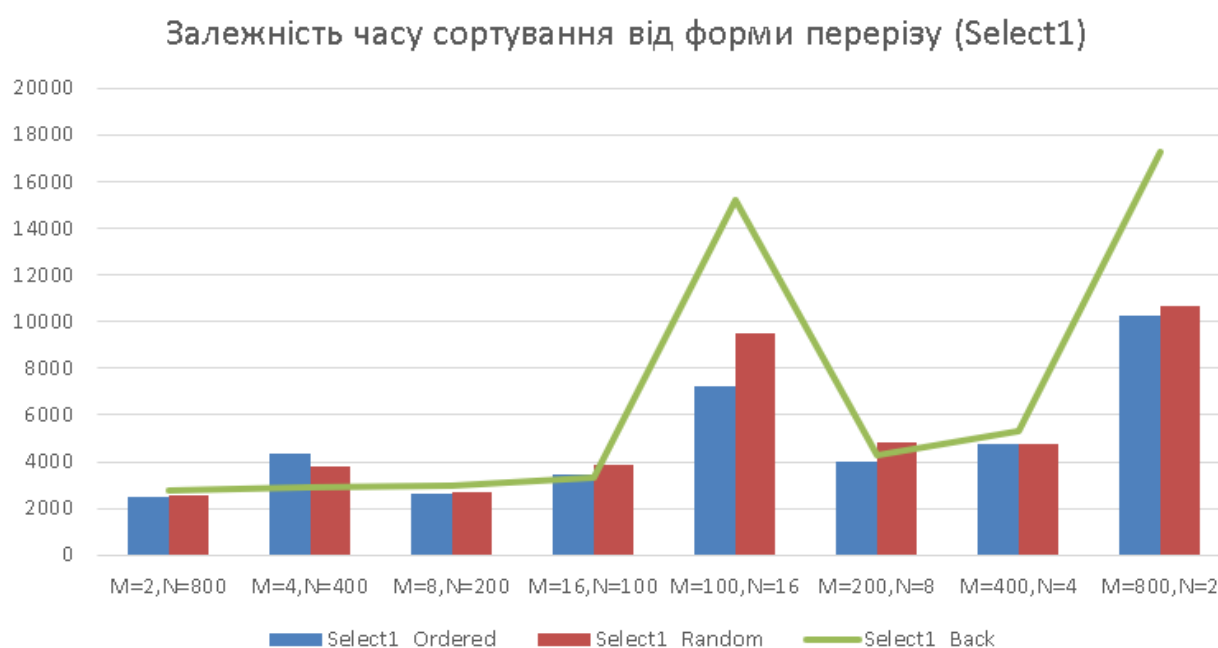
Для Select1:

Алгоритм Select1 є класичним алгоритмом сортування методом прямого вибору, у якому відбувається пошук мінімального елемента який міняється місцями з першим (елемент з індексом 0).

До переваг цього алгоритму можна віднести його просту реалізацію, не тільки для векторів, а й для 3 – D масивів. Проте у цього алгоритму є вади, наприклад величезна кількість порівнянь та переприсвоювань для великих масивів чисел, цей недолік стає особливо часозатратним коли згадати, що згідно умови задачі та способу обходу, кожен раз коли треба переставляти ключі сортування (елементи  $Arr3D[i][0][0]$ ) треба переставляти відповідні перерізи масиву.

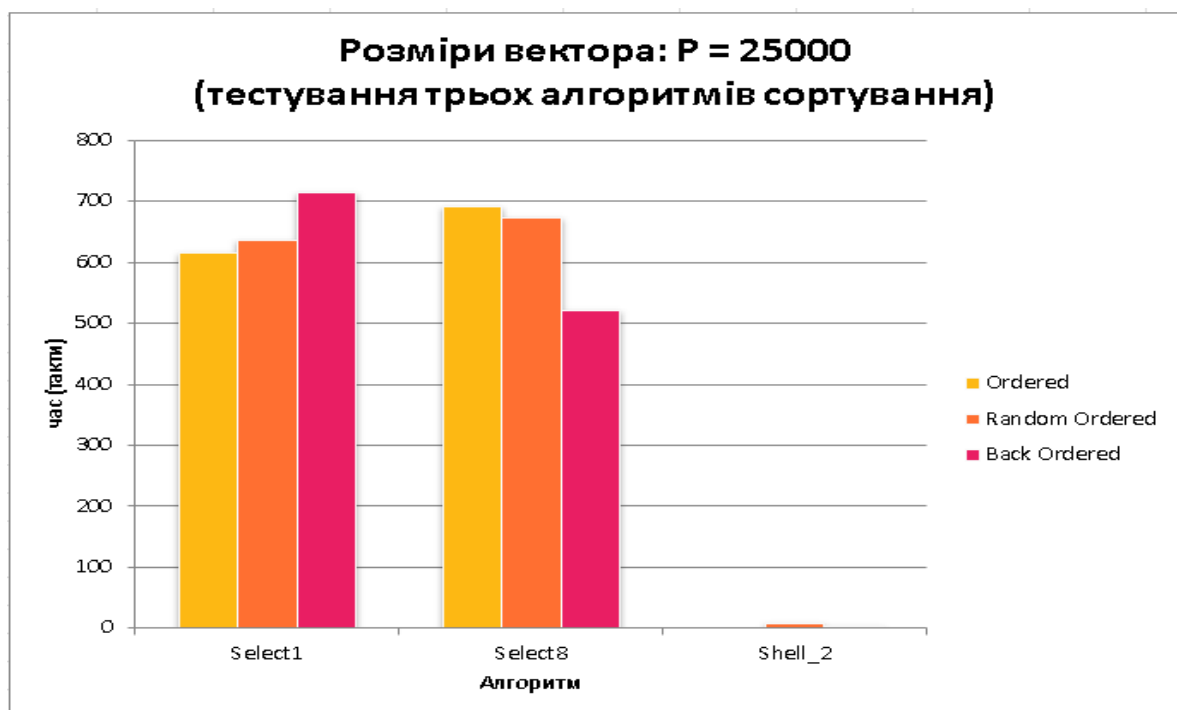
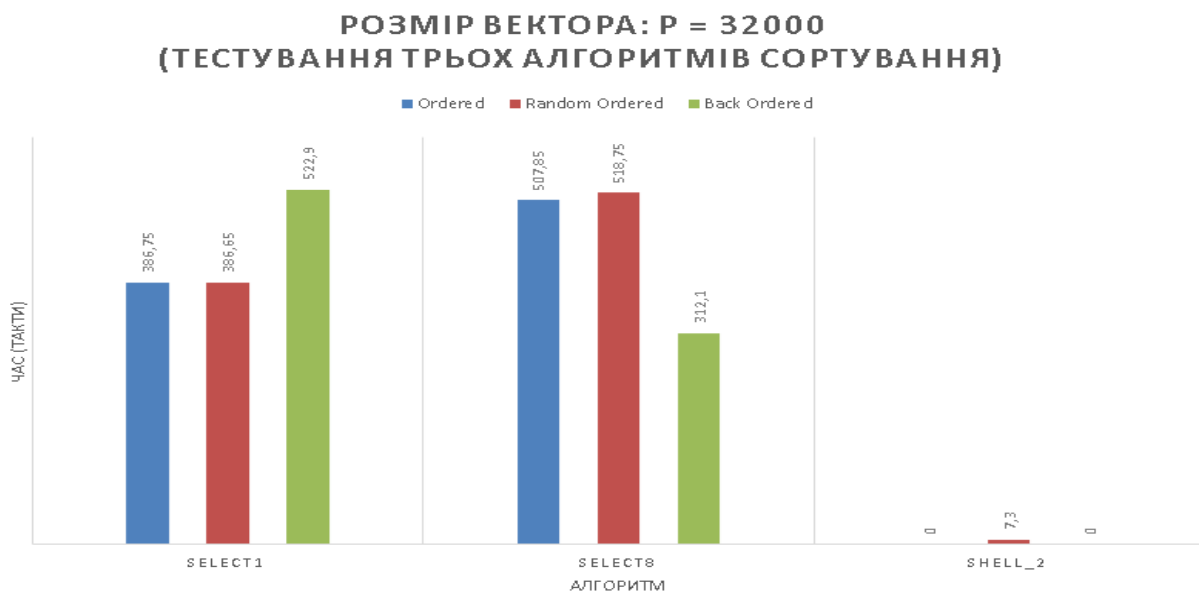


Аналізуючи графік залежності часу сортування від розміру перерізу масиву, можна помітити, що цей алгоритм має найкращі результати на невеликих перерізах та при прямій або випадковій, початковій відсортованості. Також можна зауважити, що при зростанні розміру перерізів, спостерігається майже лінійне збільшення часу сортування для всіх випадків початкової відсортованості.



З графіка залежності часу сортування від форми перерізів видно, що на досить видовжених перерізах (коли  $M \ll N$ ) час сортування є невеликим для всіх випадків початкової відсортованості масиву, такий результат не є дивним оскільки, за умовою задачі треба переставляти саме перерізи масиву, а оскільки при  $M \ll N$  перерізи є малі, переставлятимуться порівняно малі блоки даних, що значно скорочує час сортування подібних масивів.

Справедливо й наступне, якщо  $M$  приблизно таке ж або більше ніж  $N$ , то кожен раз, будуть переставлятися величезні блоки даних, що відображено на графіку різким зростанням часу (більш ніж 15000 тактів) при  $M = 100$ ,  $N = 16$  і сягає піку при  $M = 800$ ,  $N = 2$ .



З результатів наведених на діаграмах, можна сказати, що найменший час сортування зафіксовано для випадково-впорядкованих даних, трохи гірший результат є для впорядкованого вектора, що свідчить про відсутність значної оптимізації для вже впорядкованих послідовностей. Найгірший результат — на обернено-впорядкованому масиві, що демонструє суттєве зниження ефективності Select1. Це може бути наслідком того, що базовий Select1 реалізує просту стратегію вибору мінімального елемента, і в оберненому порядку йому доводиться проходити весь залишок масиву щоразу, що суттєво уповільнює процес. Таким чином, Select1 є порівняно повільним варіантом сортування для великих векторів, особливо коли дані мають несприятливу конфігурацію.

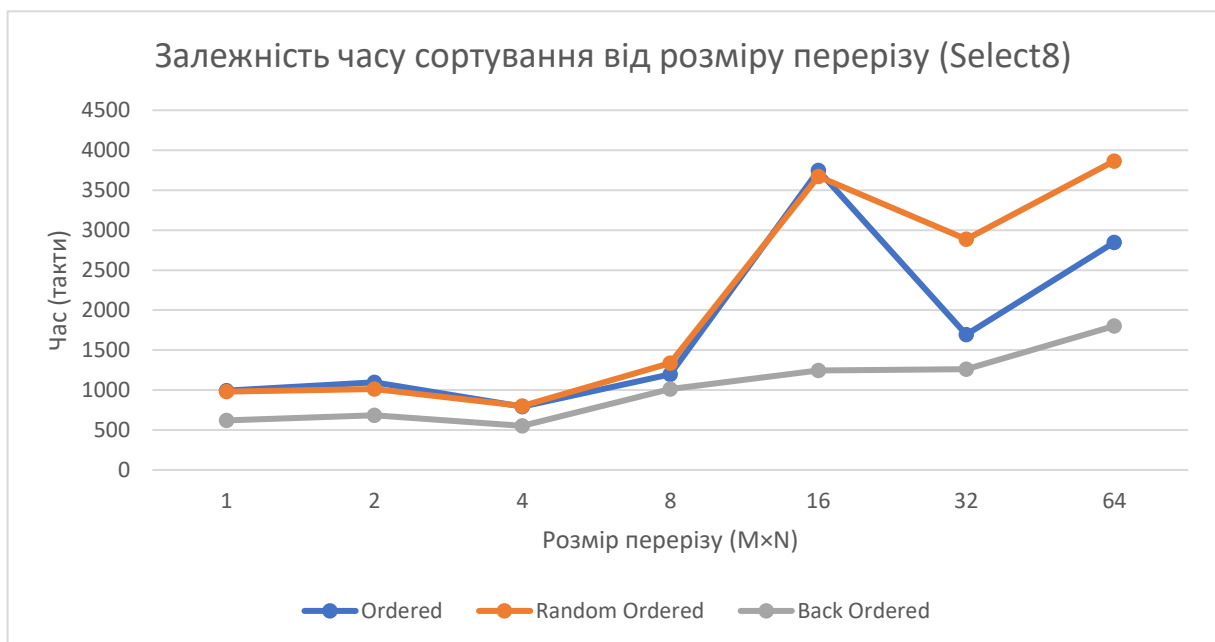
Тому підсумовуючи можна сказати, що сортування алгоритмом Select1, може бути ефективним на масивах з невеликими перерізами або витягнутими перерізами ( $M \ll N$ ) та з прямою або випадковою початковою впорядкованістю, елементів на цих перерізах.

Проте, якщо у конкретній задачі важливим фактором її виконання, є не мінімальний час, а саме простота реалізації рішення, то цей алгоритм можна сміло використовувати і на великих масивах.

Для Select8:

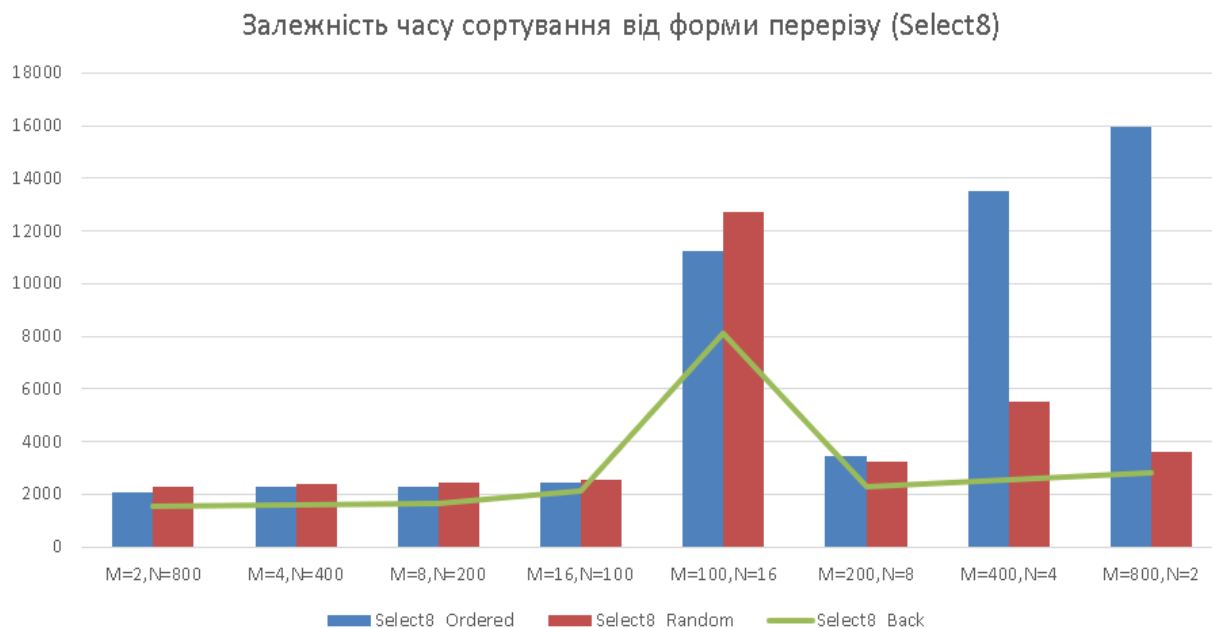
Select8 фактично є покращеною версією алгоритму Select1, оскільки за рахунок одночасного пошуку мінімуму і максимуму та їх подальшою перестановкою на кінці масиву, сортування відбувається одразу «з обох сторін», що забезпечує значно меншу кількість операцій, і відповідно має зменшити час сортування загалом.

Проте у цього методу сортування також є недоліки, один з яких це наприклад досить складна логіка, порівняно з Select1, обробки граничних випадків, також значним недоліком є ризик зайвих обмінів, через постійні перестановки перерізів із країв, блоки, які ще не впорядковані, постійно "переміщуються", це призводить до більшого часу сортування.



З графіка залежності часу сортування алгоритмом Select8 від розміру перерізів масиву, що сортується, видно, що найкращі результати виходять при невеликих розмірах перерізів та випадковій або обернено – впорядкованій, початковій відсортованості масиву. Також, варто зазначити що поведінка цього алгоритму є більш стабільна порівняно з Select1.





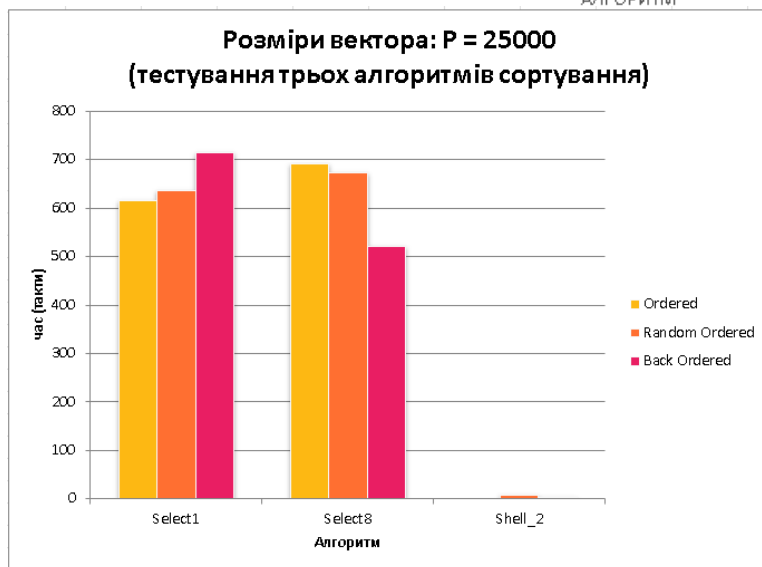
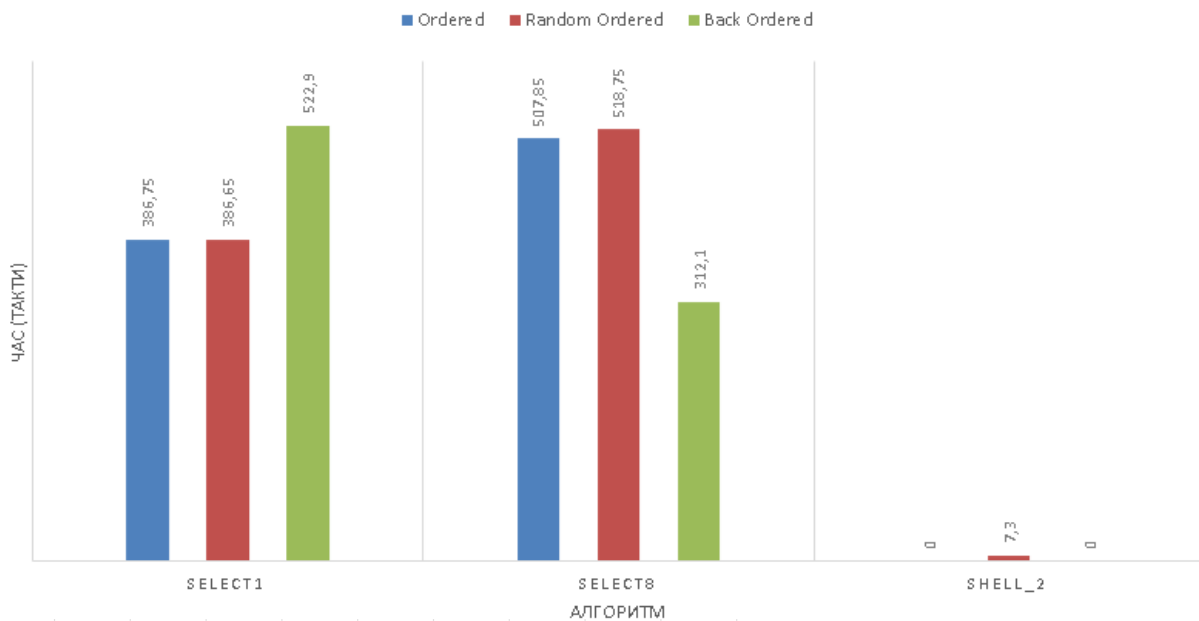
Проте алгоритм Select8 демонструє нестабільність часу сортування залежно від форми перерізу масиву. Найнижчі значення часу спостерігаються для видовжених форм (наприклад,  $M=2, N=800$ ), тоді як для наближено квадратних перерізів ( $M=100, N=16$ ) спостерігається різкий стрибок часу сортування — до 13000–14000 тактів в окремих випадках початкової відсортованості.

Така поведінка пояснюється особливостями реалізації Select8. Подвійний обмін (міняються одразу два перерізи на крок) в умовах невеликої кількості перерізів ( $P\_S$ ) дозволяє швидше зменшити розмір невідсортованої частини, але при великій кількості елементів усередині перерізу кожен обмін стає дуже часозатратним, бо копіюються всі  $M \times N$  елементів. Подібна проблема спостерігалася при аналізі Select1.

Для масивів де  $M > N$  і водночас велика кількість перерізів ( $P = 32000$ ) і значний розмір кожного перерізу, що передує великому часу виконання сортування.

У випадку обернено впорядкованих Select8 поводитьсся трохи краще, ніж у випадку випадкової впорядкованості, але загальна картина схожа: найбільший час на широких перерізах, і найменший — на видовжених.

### РОЗМІР ВЕКТОРА: P = 32000 (ТЕСТУВАННЯ ТРЬОХ АЛГОРИТМІВ СОРТУВАННЯ)



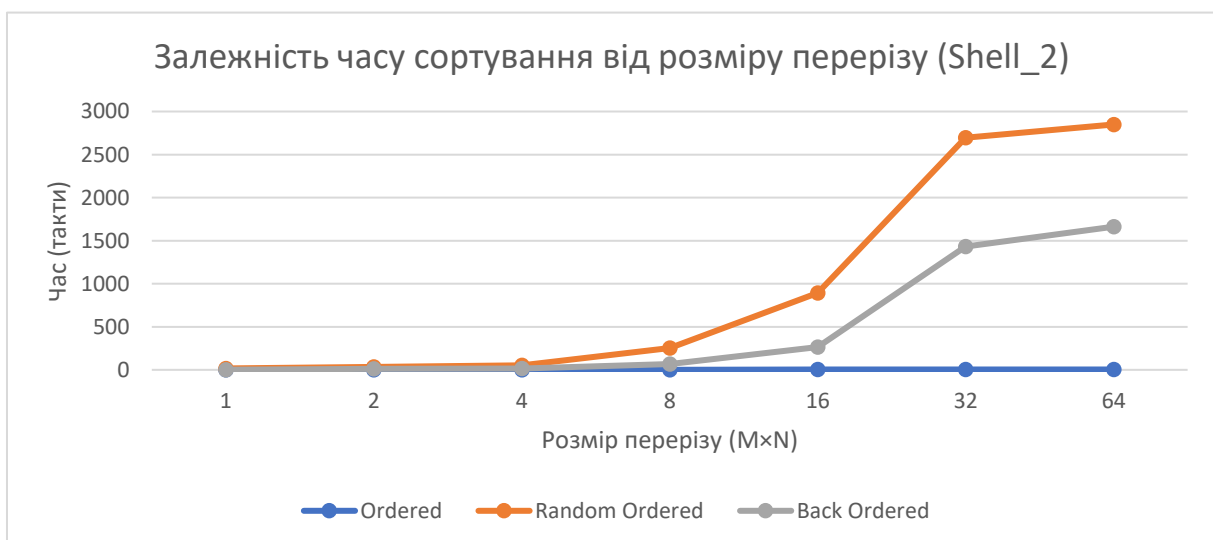
Для алгоритму Select8 найгірший час сортування вектора спостерігається при прямо-впорядкованому та випадково-впорядкованому початковому стані, що вказує на серйозне зниження ефективності за сприятливих або частково сприятливих, вхідних умов.

Алгоритм Select8, незважаючи на свою блочну структуру та можливість виконувати одночасні операції з кількома частинами перерізу, не показує значного виграшу порівняно з базовим Select1. Отже, Select8 не є оптимальним вибором для обробки великих векторів чи масивів, особливо якщо дані не є обернено-впорядкованими.

У підсумку можна сказати що Select8 найкраще підходить для видовжених перерізів з невеликою кількістю елементів, особливо коли  $M \ll N$  або  $N \ll M$ . У всіх інших випадках, особливо при великих квадратних формах, цей алгоритм є одним з найменш ефективних через складність подвійного обміну та низьку адаптивність до впорядкованості даних.

Для Shell\_2:

Алгоритм Shell\_2 реалізований на основі гібридного алгоритму «вставка-обмін» та алгоритму Шелла. Він працює таким чином: спочатку визначається початкова відстань між елементами однієї групи (як методі Шелла), а далі методом сортування обміном – вставкою сортуються елементи, які розташовані на цій відстані.

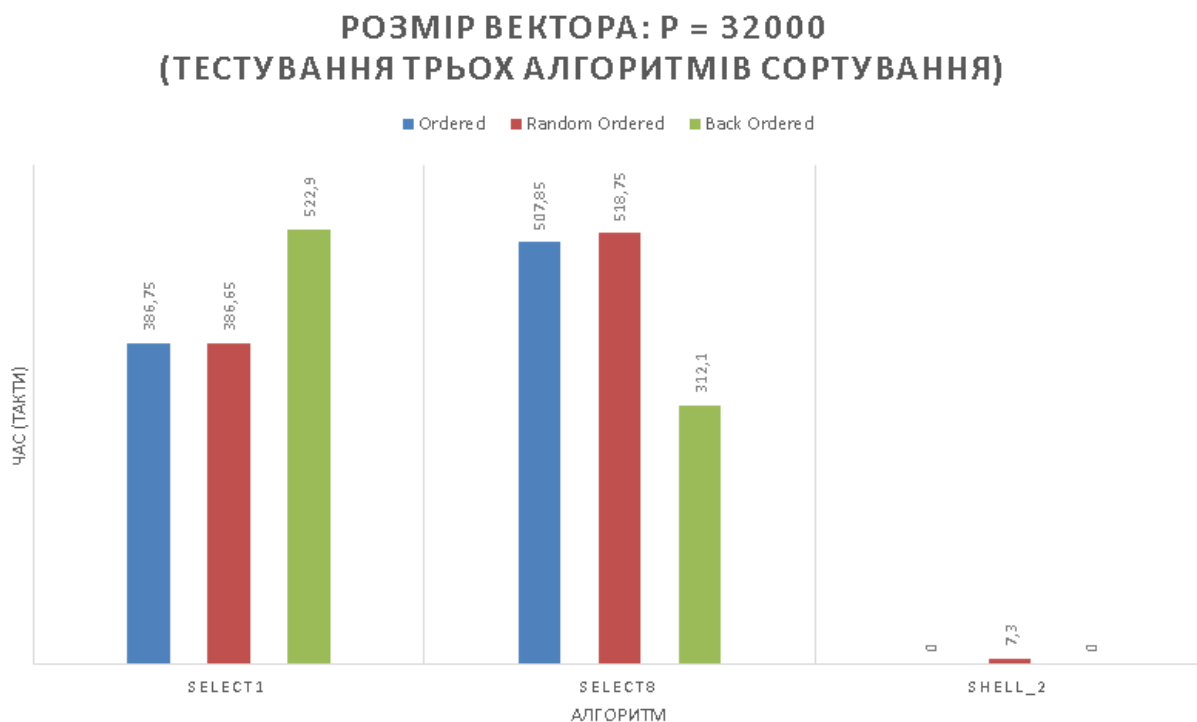
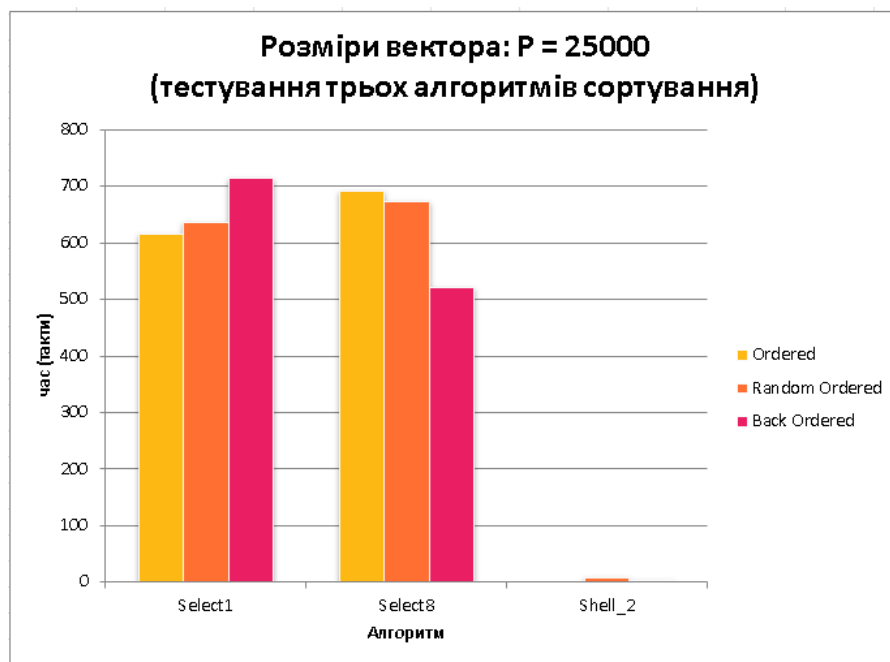


Алгоритм Shell\_2 виявляє найстабільнішу поведінку серед розглянутих алгоритмів. Його час виконання зростає плавно, без різких піків, що чітко відрізняє його від Select1 і особливо від Select8.



Найменший час виконання спостерігається при впорядкованих даних, оскільки масив уже впорядкований, то кожен крок перевірки не вимагає жодних змін. Алгоритм просто проходить по елементах, майже нічого не змінюючи — ні обмінів, ні вставок.

При випадкових даних (червоні стовпці) алгоритм демонструє порівняно низький рівень швидкодії. Пік часу на випадкових даних (майже 14000 тактів) — це найгірший сценарій для Shell\_2. У цьому випадку великі кроки не дають жодної переваги, оскільки не існує початкової структури, яку можна було б ефективно скоригувати. У результаті алгоритм виконує зайві операції перестановки на кожному етапі, що веде до значного зростання часу виконання. На графіку це чітко видно — для випадково впорядкованого вектора час сортування набагато більший, ніж для інших варіантів.



Алгоритм Шелла №2 демонструє нестабільну ефективність у залежності від початкової впорядкованості даних. Найкращі результати спостерігаються у випадку повністю впорядкованого або обернено впорядкованого вектора, де часи виконання мінімальні.

У підсумку можна сказати, що алгоритм Shell\_2 доцільно використовувати переважно для масивів великого розміру з частково впорядкованим або обернено – впорядкованим вмістом. Для коротких або сильно витягнутих масивів краще цей алгоритм не використовувати, особливо якщо масив є випадково впорядкованим.

## Висновки по отриманих результатах

У ході написання курсової роботи було досліджено та проаналізовано ефективність використання заданих за варіантом № 119 алгоритмів, для сортування тривимірних масивів заданим способом обходу № 7.

Досліджувалися наступні алгоритми:

- 1) Алгоритм сортування № 1 на основі методу прямого вибору;
- 2) Алгоритм сортування № 8 на основі методу прямого вибору;
- 3) Алгоритм сортування № 2 на основі методу Шелла;

Select1, може бути ефективним на масивах з невеликими перерізами або витягнутими перерізами ( $M \ll N$ ) та з прямою або випадковою початковою впорядкованістю, елементів на цих перерізах.

Проте, якщо у конкретній задачі важливим фактором її виконання, є не мінімальний час, а саме простота реалізації рішення, то цей алгоритм можна сміло використовувати і на великих масивах.

Select8 найкраще підходить для видовжених перерізів з невеликою кількістю елементів, особливо коли  $M \ll N$  або  $N \ll M$ . У всіх інших випадках, особливо при великих квадратних формах, цей алгоритм є одним з найменш ефективних через складність подвійного обміну та низьку адаптивність до впорядкованості даних.

Shell\_2 доцільно використовувати переважно для масивів великого розміру з частково впорядкованим або обернено – впорядкованим вмістом. Для коротких або сильно витягнутих масивів краще цей алгоритм не використовувати, особливо якщо масив є випадково впорядкованим.

## **Список використаної літератури**

- 1) Youtube канал «Марченко Олександр Іванович»;