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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

КУРСОВА РОБОТА

з дисципліни "Структури даних і алгоритми"

Виконала: Міндер В. Ю.
Група: КВ-22
Номер залікової книжки: КВ-13876184

Допущений до захисту

_____

2 семестр 2022/2023

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Узгоджено

Керівник роботи

_____/Марченко О.І./

ЗАХИЩЕНА "__"_____2022р.

з оцінкою_____

_____ /Марченко О.І./

Дослідження ефективності методів сортування (Гібридний алгоритм "вибір№4 – обмін" ,Алгоритм №1 методу сортування Шелла,Алгоритм сортування №2 методу прямого вибору)на багатовимірних масивах

Виконавець роботи:
Міндер В. Ю.

1._____2023 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 багато-вимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів, тобто виміри часу роботи цих алгоритмів для різних випадків та геометричних розмірів багатовимірних масивів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

Одна таблиця результатів (вимірів часу сортування впорядкованого, випадкового і обернено-впорядкованого масиву) для масиву з заданими геометричними розмірами повинна бути такою:

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

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

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

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

Зробити виміри часу для стандартного випадку одномірного 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 поводять себе саме так, а не інакше**.

VIII. Зробити висновки за зробленим порівняльним аналізом.

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

Варіант № 102

Задача

Впорядкувати окремо кожен переріз тривимірного масива Arr3D [P,M,N] наскрізно по стовпчиках за незменшенням.

Досліджувані методи та алгоритми

- Гібридний алгоритм "вибір№4 – обмін" (додаток 1, рис. №21).
- Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2) (додаток 1, рис. №22). Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються.
- Алгоритм сортування №2 методу прямого вибору (додаток 1, рис. №6).

Способи обходу

Використовуючи елементи першого рядка кожного перерізу як ключі сортування, переставляти відповідні стовпчики кожен раз, коли треба переставляти ключі. При перестановці стовпчиків потрібно саме копіювати їх елементи, а не ко- 27 піювати вказівники на них, використовуючи операції з вказівниками мови C/C++.

Випадки дослідження

Випадок дослідження I.
Залежність часу роботи алгоритмів від довжини стовпчиків масива
Рекомендовані розміри масива для досліджень:
Кількість ключів у перерізі (N) і загальна кількість ключів (N*P) є константами.
P = const = 3, N = const = 155000
1) M = 1;
2) M = 2;
3) M = 4;
4) M = 8;
5) M = 16;
6) M = 32;
7) M = 64;
8) M = 128;
Вектор довжиною кількості ключів у перерізі N = 77500.
Для порівняння час сортування такого вектора помножити на P. 34
Випадок дослідження II.
Залежність часу роботи алгоритмів від форми перерізів масива
Порівняння з вектором для цього випадку не потрібне.
Рекомендовані розміри масива для досліджень:
Кількість елементів у кожному перерізі (M*N) є константою.

P = const = 3, M = var, N = var, M*N = const = 1000000.
1) M = 10; N = 100000;
2) M = 100; N = 10000;
3) M = 1000; N = 1000;
4) M = 10000; N = 100;
5) M = 100000; N = 10;
Випадок дослідження III.
Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масива при однаковій загальній кількості ключів у всьому масиві
Порівняння з вектором для цього випадку не потрібне.
Рекомендовані розміри масива для досліджень:
Кількість ключів у перерізі N = var
Загальна кількість ключів у матриці
P*N = const = 400000
Довжина стовпчика M = const = 10
P = var, N = var, M = const = 100, P*N = const = 400000
1) P = 20; N = 20000;
2) P = 200; N = 2000;
3) P = 2000; N = 200;
4) P = 20000; N = 20;

Опис теоретичних положень

Алгоритм №1 методу сортування Шелла

Принцип роботи:

Сортування виконується в декілька етапів

Загальний принцип методу сортування прямого вибору(на прикладі 3-х етапів):

1. Спочатку методом вставки сортуються лише ті елементи, що розташовані один від одного на відстані 4 позиції, при цьому всі інші елементи в сортуванні участі не приймають. Таких груп елементів у масиві буде 4, і кожна група елементів сортується окремо.

2. Виконується сортування методом вставки тільки елементів, що розташовані один від одного на відстані в 2 позиції. Таких груп буде 2, і кожна сортується окремо.

3. Виконується аналогічно першим двом для елементів на відстані 1 елемент, тобто відбувається звичайне сортування методом прямої вставки.

У даному варіанті сортування Шелла відбувається зсув елементів з подальшою вставкою найменшого елемента на свою позицію

Особливості алгоритму:

1. Основною особливістю сортування Шелла полягає поділ сортування на етапи. В загальному випадку кількість етапів може бути довільною, а відстань між взаємодіючими елементами не обов’язково повинні бути степенями 2, як початково запропонував Шелл.

2. Також в цьому алгоритмі присутнє протиріччя, тобто при збільшенні кількості етапів, збільшується кількість порівнянь, що сповільнює алгоритм, проте зменшується кількість присвоєнь, адже елементи зі збільшенням кількості етапів переставляються на дальшу відстань, тому алгоритм стає більш швидкий.

Хоч алгоритм Шелла показує досить високі показники швидкості, але при аналізі цього алгоритму виникли деякі проблеми, головною проблемою є пошук оптимальної кількості етапів та оптимальних відстаней між взаємодіючими елементами, для збільшення швидкості алгоритму, на основі правильного підбору кількості етапів.

Час виконання алгоритму :

T = O (n ($ln\ ln\ n$) 2)

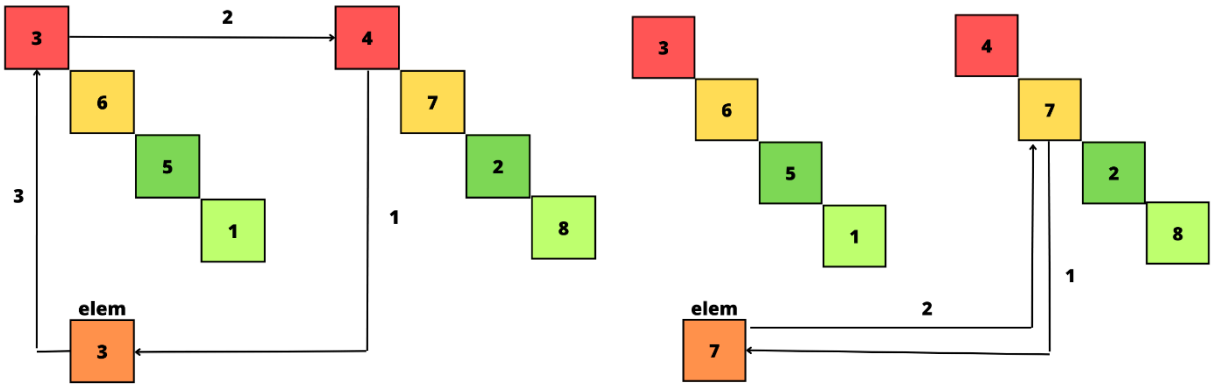
Схема роботи:

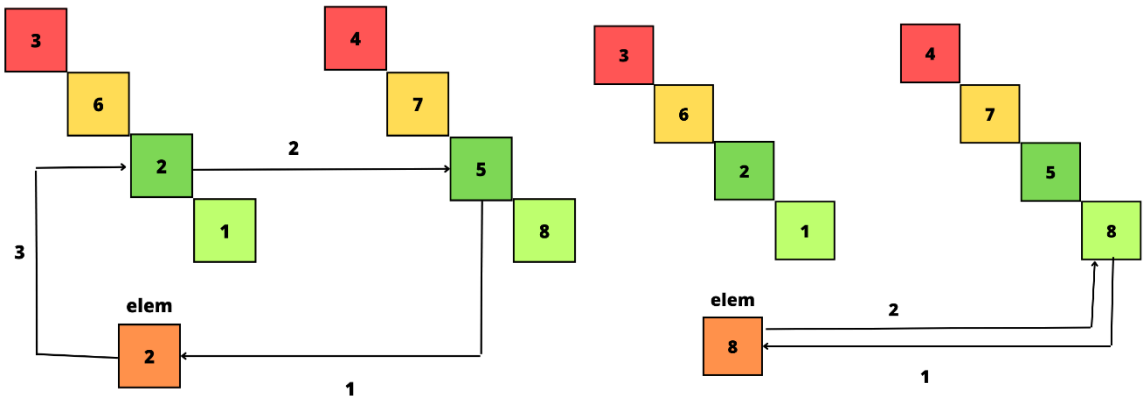
Початковий стан масива:

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

Етап 1:

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



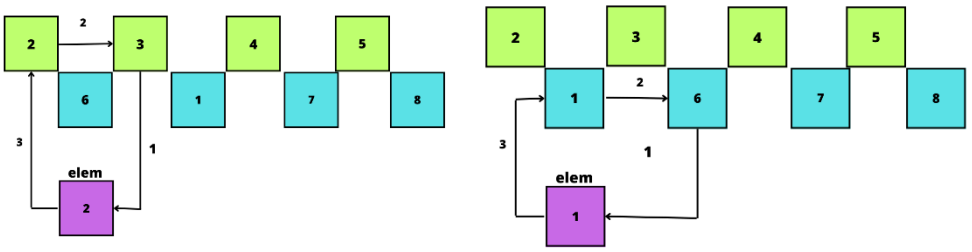


Масив після першого етапу:

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

Етап 2:

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

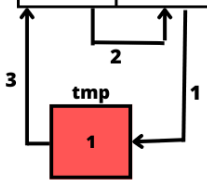


Масив після другого етапу:

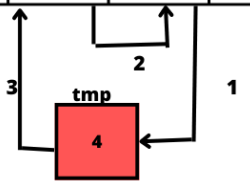
0	1	2	3	4	5	6	n-1=7
2	1	3	6	4	7	5	8

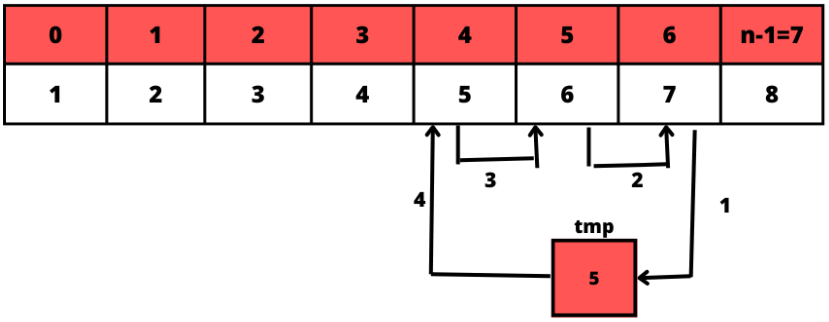
Етап 3:

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

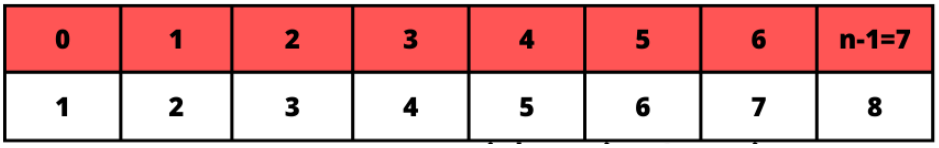


0	1	2	3	4	5	6	n-1=7
1	2	3	4	6	7	5	8





Сортування завершено:



Алгоритм на мові с

```
clock_t Shell1_3D()
{
   int Elem[M][N], t, j, k;
   clock_t time_start, time_stop;
   time_start = clock();
   if (P < 4) t = 1;
   else t = (int)log2f((float)P) - 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 < P; i++)
      {
         for(int r = 0; r < M; r++)
         {
```

```
            for(int u = 0; u < N; u++)
            {
                Elem[r][u] = Arr3D[i][r][u];
            }
        }
        j = i;
        while (j >= k && Elem[0][0] < Arr3D[j - k][0][0])
        {
            for(int r = 0; r < M; r++)
            {
                for(int u = 0; u < N; u++)
                {
                    Arr3D[j][r][u] = Arr3D[j - k][r][u];
                }
            }
            j--;
        }
        for(int r = 0; r < M; r++)
        {
            for(int u = 0; u < N; u++)
            {
                Arr3D[j][r][u] = Elem[r][u];
            }
        }
    }
}
time_stop = clock();
```

```
    return time_stop - time_start;
}
```

Гібридний алгоритм "вибір№4 – обмін" (додаток 1, рис. №21).

Принцип роботи

Беремо початкові значення, які виступають границями, у нас це L та R, яким присвоюються значення першого та останнього елемента відповідно.

1) Проходимо по масиву починаючи з L-того елементу і порівнюємо його з нульовим та n-1-шим елементом, якщо L-ий елемент менший за нульовий або більший за останній, то робимо відповідні обміни.
2) Таким чином сортуємо всі елементи масиву, збільшуючи L та зменшуючи R.
3) З лівого боку від L та з правого боку від R масив можна вважати відсортованим, а між границями – невідсортованим.

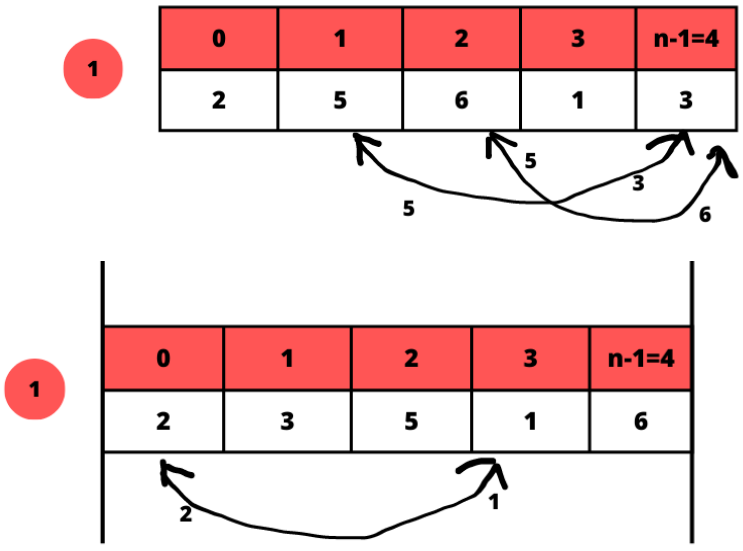
Якщо n парне, то в кінці між границями залишається один елемент. Якщо непарне – порівнюються два елементи між собою.

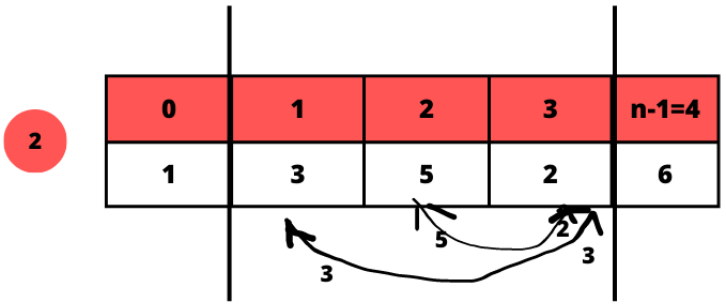
Оцінювання алгоритму

Кількість операцій порівняння C = O(n^2).

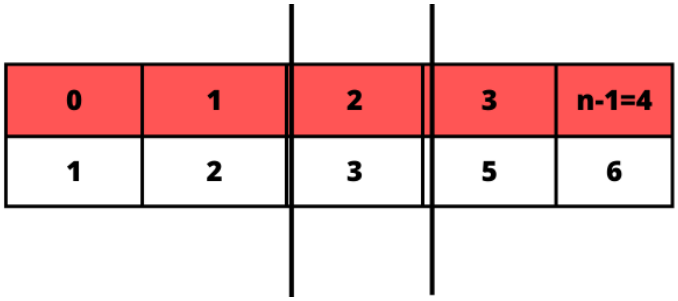
Кількість присвоєнь R = O(n^2).

Схема роботи





Результат :



Алгоритм на мові с

```
clock_t Select4Exchange_3D()
{
   int Min, Max;
   int L, R;
   clock_t time_start, time_stop;
   time_start = clock();
   L = 0;
   R = P - 1;
   while (L < R)
   {
      for(int i = L; i < R + 1; i++)
      {
         if (Arr3D[i][0][0] < Arr3D[L][0][0])
         {
            for(int j = 0; j < M; j++)
```

```
            {
                for(int k = 0; k < N; k++)
                {
                    Min = Arr3D[i][j][k];
                    Arr3D[i][j][k] = Arr3D[L][j][k];
                    Arr3D[L][j][k] = Min;
                }
            }
        }
        else
        {
            if(Arr3D[i][0][0] > Arr3D[R][0][0])
            {
                for(int j = 0; j < M; j++)
                {
                    for(int k = 0; k < N; k++)
                    {
                        Max = Arr3D[i][j][k];
                        Arr3D[i][j][k] = Arr3D[R][j][k];
                        Arr3D[R][j][k] = Max;
                    }
                }
            }
        }
    }
    L = L + 1;
    R = R - 1;
}
```

```
    time_stop = clock();


    return time_stop - time_start;
}
```

Алгоритм сортування №2

Принцип роботи

В кожен момент часу масив ділиться на відсортовану частину і невідсортовану. Перед початком весь масив вважається невідсортованою частиною.

1. Вибираємо найменший елемент в діапазоні від 0 до n-1
2. Знайдений елемент міняємо місцями з нульовим елементом масиву. Таким чином один елемент стає відсортованою частиною.
3. Дії 1-2 повторюються для невідсортованої частини в діапазоні від 1 до n-1, від 2 до n-1 і т.д., міняючи при цьому найменший елемент місцями з відповідно з елементом з індексом 1, з індексом 2 і т.д. . Таким чином відсортована частина зростає.
4. Останній доцільний діапазон пошуку елемента від n-2 до n-1, тому що шукати найменше з одного елементу не матиме сенсу.

Властивості:

Весь масив – невідсортована частина, границя стоїть зліва. S – стартова позиція з якої починаємо шукати мінімум (спочатку S = 0), imin – вказує позицію мінімума. Потім міняємо місцями елемент на стартовій позиції з елементом на позиції мінімум. Індекс стартової позиції збільшується.

Якщо стартова позиція та позиція мінімального співпадають, то елемент міняється місцями сам з собою.

Автоматично на позиції n – 1 стає найбільший елемент. Проходів буде завжди n-1. Використовуємо S як основний лічильник.

Оцінювання алгоритму :

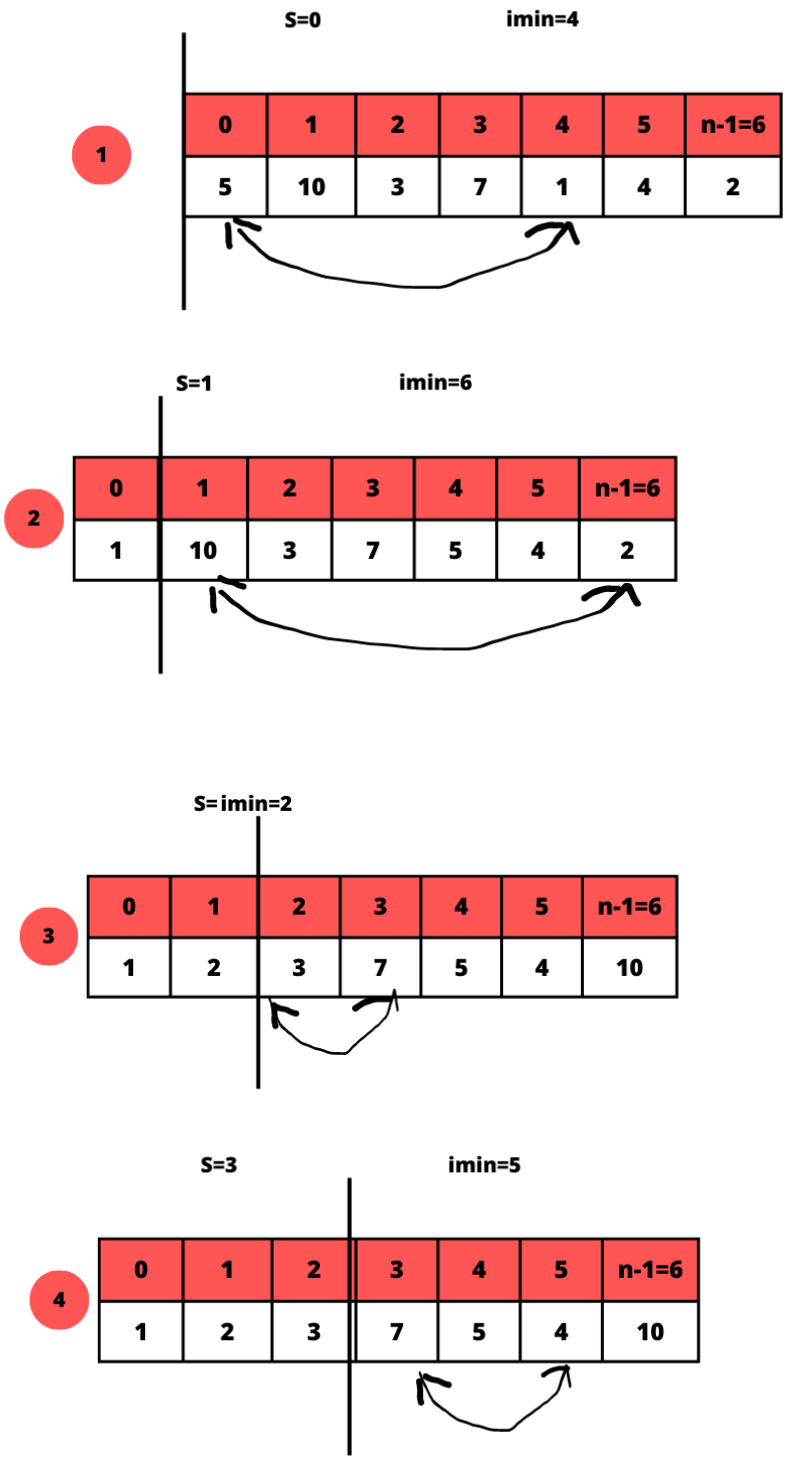
Кількість порівнянь $C = \frac{n^2-n}{2}$. Залежність квадратична. $C = O(n^2)$.

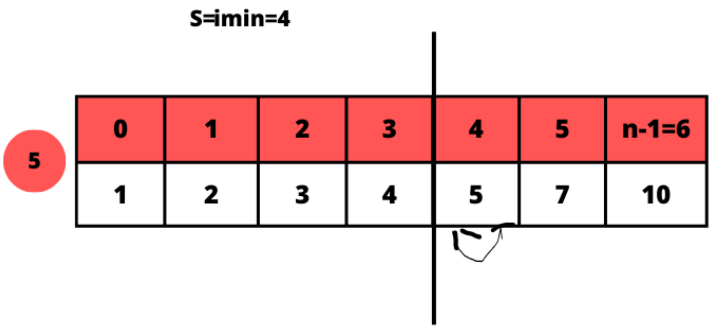
Незалежно від відсортованості масиву кількість порівнянь стала.

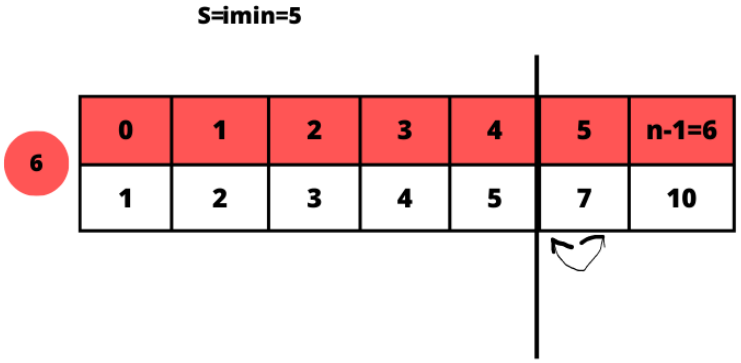
Кількість присвоєнь R = O(n*ln(n)). Єдина характеристика серед усіх прямих методів, яка не є квадратичною.

Загальний час роботи квадратичний (метод належить до прямих).

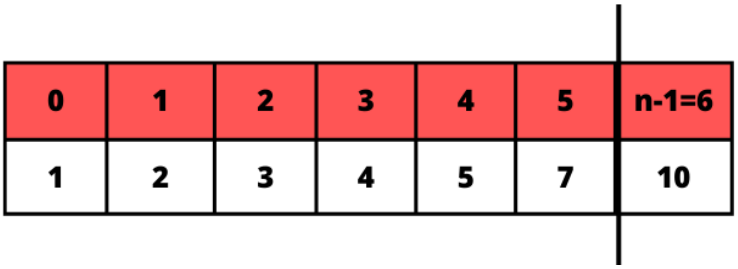
Схема:







Результат:



Алгоритм на мові с

```
clock_t Select2_3D()
{
   int imin, tmp;
   clock_t time_start, time_stop;
   time_start = clock();
   for(int s = 0; s < P - 1; s++)
   {
      imin = s;
```

```
        for(int i = s + 1; i < P; i++)
            if (Arr3D[i][0][0] < Arr3D[imin][0][0])
                imin = i;
        for(int j = 0; j < M; j++)
        {
            for(int k = 0; k < N; k++)
            {
                tmp = Arr3D[imin][j][k];
                Arr3D[imin][j][k] = Arr3D[s][j][k];
                Arr3D[s][j][k] = tmp;
            }
        }
    }
    time_stop = clock();

    return time_stop - time_start;
}
```

Схема імпорту/ експорту модулів

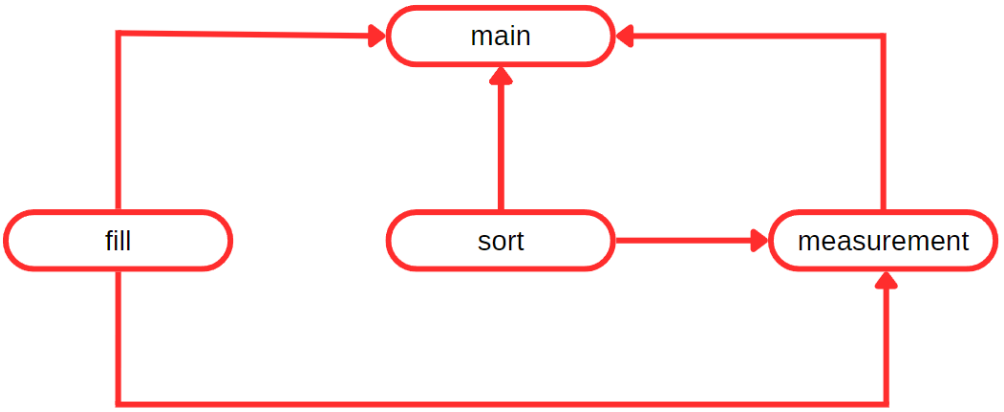
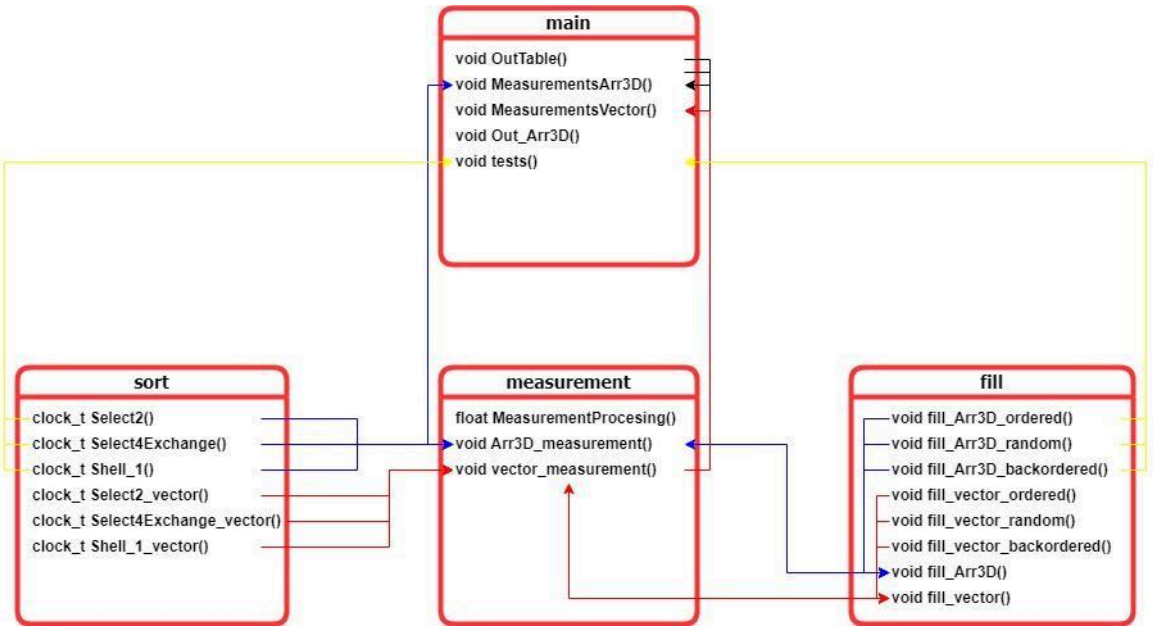


Схема взаємовикликів процедур та функцій



Опис призначення процедур та функцій

main

- void OutTable(float ordered, float random, float backordered, int alg) – функція для виведення результату вимірювання для одного алгоритму;
- void MeasurmentsArr3D(int alg) – функція для вимірювання та виведення часу роботи одного алгоритму для тривимірного масиву;
- void MeasurmentsVector(int alg) – функція для вимірювання та виведення часу роботи одного алгоритму для одновимірного масиву;
- void Out_Arr3D(int ***Arr3D, int p, int m, int n) – функція для виведення на екран тривимірного масиву;
- void tests() - функція для перевірки правильності роботи алгоритмів сортування та заповнення масивів;

measurement

- float MeasurementProcessing() – функція для обробки та усереднення значень вимірування;
- void Arr3D_measurement(int ***Arr3D, int P, int M, int N, int alg, int fill) – функція для вибору та запису часу роботи алгоритму в масив значень для тривимірного масиву;
- void vector_measurement(int *vector, int N, int alg ,int fill) – функція для вибору та запису часу роботи алгоритму в масив значень для одновимірного масиву;

sort

- clock_t Select2(int ***A, int P, int M, int N) – функція для сортування тривимірного масиву алгоритмом Select2;
- clock_t Select4Exchange(int ***A, int P, int M, int N) – функція для сортування тривимірного масиву алгоритмом Select4Exchange;

- clock_t Shell_1(int ***A, int P, int M, int N) – функція для сортування тривимірного масиву алгоритмом Shell_1;
- clock_t Select2_vector(int *A, int N) – функція для сортування одновимірного масиву алгоритмом Select2;
- clock_t Select4Exchange_vector(int *A, int N) – функція для сортування одновимірного масиву алгоритмом Select4Exchange;
- clock_t Shell_1_vector(int *A, int N) – функція для сортування одновимірного масиву алгоритмом Shell_1;

fill

- void fill_Arr3D_ordered(int ***Arr3D, int P, int M, int N) – функція для заповнення тривимірного масиву впорядкованими за зростанням елементами;
- void fill_Arr3D_random(int ***Arr3D, int P, int M, int N) – функція для заповнення тривимірного масиву невпорядкованими елементами;
- void fill_Arr3D_backordered(int ***Arr3D, int P, int M, int N) – функція для заповнення тривимірного масиву впорядкованими за спаданням елементами;
- void fill_vector_ordered (int *vector, int N) – функція для заповнення одновимірного масиву впорядкованими за зростанням елементами;
- void fill_vector_random (int *vector, int N) – функція для заповнення одновимірного масиву невпорядкованими елементами;
- void fill_vector_backordered (int *vector, int N) – функція для заповнення одновимірного масиву впорядкованими за спаданням елементами;
- void fill_Arr3D(int ***Arr3D, int P, int M, int N, int fill) – функція для вибору способу заповнення тривимірного масиву;
- void fill_vector(int *vector, int N, int fill) – функція для вибору способу заповнення для одновимірного масиву;

Код програми

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <time.h>
#include <conio.h>
#include "sort.h"
#include "fill.h"
#include "measurement.h"


#define P 3
#define M 1
#define N 77500

void OutTable(float ordered, float random, float backordered, int alg)
{
  switch(alg){
    case 1:
      printf("%-20s", "Select 2");
    break;
    case 2:
      printf("%-20s","Select 4 Exchange");
    break;
    case 3:
      printf("%-20s", "Shell 1");
    break;
```

```
    }
    printf("%-10.2f%-10.2f%-10.2f \n",ordered, random, backordered);
}

void MeasurmentsArr3D(int alg)
{

    int ***Arr3D;
    Arr3D = (int***) malloc(P*sizeof(int**));
    for (int k=0; k<P; k++)
    {
        Arr3D[k] = (int**) malloc(M*sizeof(int*));
        for (int i=0; i<M; i++)
            Arr3D[k][i] = (int*) malloc(N*sizeof(int));
    }


    Arr3D_measurement(Arr3D, P, M, N, alg, 1);
    float ordered = MeasurementProcessing();

    Arr3D_measurement(Arr3D, P, M, N, alg, 2);
    float random = MeasurementProcessing();

    Arr3D_measurement(Arr3D, P, M, N, alg, 3);
    float backordered = MeasurementProcessing();
    OutTable(ordered, random, backordered, alg);


    for (int k=0; k<P; k++)
    {
        for (int i=0; i<M; i++)
            free(Arr3D[k][i]);
        free(Arr3D[k]);
    }
    free(Arr3D);
}

void MeasurmentsVector(int alg)
{
    int *vector;
    vector = (int*)malloc(sizeof(int)*N);


    vector_measurement(vector, N, alg, 1);
    float ordered = MeasurementProcessing();

    vector_measurement(vector, N, alg, 2);
    float random = MeasurementProcessing();

    vector_measurement(vector, N, alg, 3);
    float backordered = MeasurementProcessing();
    OutTable(ordered, random, backordered, alg);

    free(vector);

}


void Out_Arr3D(int ***Arr3D, int p, int m, int n){
    for(int k = 0; k < p; k++){
        printf("P = %d\n", k);
        for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){
                printf("%4d", Arr3D[k][i][j]);
            }
            printf("\n");
        }
        printf("\n");
    }
}

void tests(){
    int p = 3, m = 8, n = 8;
    int ***Arr3D;
```

```
Arr3D = (int***) malloc(p*sizeof(int**));
for (int k=0; k<p; k++)
{
   Arr3D[k] = (int**) malloc(M*sizeof(int*));
   for (int i=0; i<m; i++)
      Arr3D[k][i] = (int*) malloc(m*sizeof(int));
}

printf("Select algorithm:\n");
printf("1. Select 2\n");
printf("2. Select 4 Exchange\n");
printf("3. Shell 1\n");
int num;
scanf("%d", &num);
system("cls");

switch(num){
   case 1:
      for(int i = 1; i <= 3; i++){
         switch(i){
            case 1:
               printf("Ordered\n");
            break;
            case 2:
               printf("Random\n");
            break;
            case 3:
               printf("Backordered\n");
            break;
         }
         fill_Arr3D(Arr3D, p, m, n, i);
         printf("Before sorting:\n");
         Out_Arr3D(Arr3D, p, m, n);
         Select2(Arr3D, p, m, n);
         printf("After sorting:\n");
         Out_Arr3D(Arr3D, p, m, n);

      }
   break;
   case 2:
      for(int i = 1; i <= 3; i++){
         switch(i){
            case 1:
               printf("Ordered\n");
            break;
            case 2:
               printf("Random\n");
            break;
            case 3:
               printf("Backordered\n");
            break;
         }
         fill_Arr3D(Arr3D, p, m, n, i);
         printf("Before sorting:\n");
         Out_Arr3D(Arr3D, p, m, n);
         Select4Exchange(Arr3D, p, m, n);
         printf("After sorting:\n");
         Out_Arr3D(Arr3D, p, m, n);
      }
   break;
   case 3:
      for(int i = 1; i <= 3; i++){
         switch(i){
            case 1:
               printf("Ordered\n");
            break;
            case 2:
               printf("Random\n");
            break;
            case 3:
               printf("Backordered\n");
            break;
         }
         fill_Arr3D(Arr3D, p, m, n, i);
```

```
            printf("Before sorting:\n");
            Out_Arr3D(Arr3D, p, m, n);
            Shell_1(Arr3D, p, m, n);
            printf("After sorting:\n");
            Out_Arr3D(Arr3D, p, m, n);
          }
        break;
    }

    for (int k=0; k<p; k++)
    {
      for (int i=0; i<m; i++)
        free(Arr3D[k][i]);
      free(Arr3D[k]);
    }
    free(Arr3D);


}

int main()
{
    srand(time(NULL));
    int f = 0;


    while(f != 1){
      system("cls");
      printf("1.Measurement table Arr3D\n");
      printf("2.Measurement table Vector\n");
      printf("3.Measurement algorithm Arr3D\n");
      printf("4.Tests\n");
      printf("5.Exit\n");
      int num;
      scanf("%d", &num);
      system("cls");

      switch(num){
        case 1:
          printf("P = %d\nM = %d\nN = %d\n", P ,M ,N);
          printf("%20s%-10s%-10s%10s \n","", "ordered", "random", "backordered");
          for(int i = 1; i <= 3; i++){
            MeasurmentsArr3D(i);
          }
          getch();
        break;
        case 2:
          printf("N = %d\n", N);
          printf("%20s%-10s%-10s%10s \n","", "ordered", "random", "backordered");
          for(int i = 1; i <= 3; i++){
            MeasurmentsVector(i);
          }
          getch();
        break;
        case 3:
          printf("Select algorithm:\n");
          printf("1. Select 2\n");
          printf("2. Select 4 Exchange\n");
          printf("3. Shell 1\n");
          int num;
          scanf("%d", &num);
          system("cls");
          printf("P = %d\nM = %d\nN = %d\n", P ,M ,N);
          printf("%20s%-10s%-10s%10s \n","", "ordered", "random", "backordered");
          MeasurmentsArr3D(num);
          getch();
        break;
        case 4:
          tests();
          getch();
        break;
        case 5:
          f = 1;
        break;
```

```
        default:

        break;

      }
    }
    return 0;
}
```

sort.h

```
#ifndef SORT_H_INCLUDED
#define SORT_H_INCLUDED


#include <time.h>

clock_t Select2(int ***A, int P, int M, int N);
clock_t Select4Exchange(int ***A, int P, int M, int N);
clock_t Shell_1(int ***A, int P, int M, int N);

clock_t Select2_vector(int *A, int N);
clock_t Select4Exchange_vector(int *A, int N);
clock_t Shell_1_vector(int *A, int N);
#endif // SORT_H_INCLUDED
```

sort.c

```
#include "sort.h"
#include <time.h>
#include <math.h>
#include <stdlib.h>

clock_t Select2(int ***A, int P, int M, int N){
   int imin;

   int buf;

   clock_t time_start, time_stop;
   time_start = clock();
   for(int k = 0; k < P; k++)
      for(int s = 0; s < N-1; s++){
         imin = s;
         for(int i=s+1; i < N; i++){
           if (A[k][0][i] < A[k][0][imin])
              imin=i;
         }

           for(int j = 0; j < M; j++){
              buf = A[k][j][imin];
              A[k][j][imin] = A[k][j][s];
              A[k][j][s] = buf;
           }
      }

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

clock_t Select4Exchange(int ***A, int P, int M, int N){
   int L, R;

   int Min, Max;

   clock_t time_start, time_stop;
   time_start = clock();
   for(int k = 0; k < P; k++){
      L=0; R=N-1;
      while (L<R){
         for(int i=L; i<R+1; i++){
           if (A[k][0][i] < A[k][0][L]){
              for(int j = 0; j < M; j++){
                 Min = A[k][j][i];
                 A[k][j][i] = A[k][j][L];
                 A[k][j][L] = Min;
```

```
                }
            }
            else if(A[k][0][i] > A[k][0][R]){

                for(int j = 0; j < M; j++){
                    Max = A[k][j][i];
                    A[k][j][i] = A[k][j][R];
                    A[k][j][R] = Max;
                }
            }
        }
        L=L+1; R=R-1;
    }
  }

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

clock_t Shell_1(int ***A, int P, int M, int N)
{
  int t, j, k;
  clock_t time_start, time_stop;
  int *Elem;
  Elem = (int*)malloc(sizeof(int)*M);

  time_start = clock();

  if (N<4) t=1;
  else t=(int)log2f((float)N)-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 l = 0; l < P; l++){
    for (int p=0; p<t; p++){
      k=Stages[p];
      for (int i=k; i<N; i++){
        for(int y = 0; y < M; y++){
          Elem[y] = A[l][y][i];
        }
        j=i;
        while (j>=k && Elem[0]<A[l][0][j-k]) {
          for(int y = 0; y < M; y++){
            A[l][y][j]=A[l][y][j-k];
          }
          j=j-k;
        }
        for(int y = 0; y < M; y++){
          A[l][y][j] = Elem[y];
        }
      }
    }
  }
  time_stop = clock();
  free(Elem);
  return time_stop - time_start;
}

clock_t Select2_vector(int *A, int N)
{
  int imin, tmp;
  clock_t time_start, time_stop;
  time_start = clock();
  for(int s=0; s<N-1; s++){
    imin=s;
    for(int i=s+1; i<N; i++)
      if (A[i]<A[imin]) imin=i;

    tmp=A[imin];
    A[imin]=A[s];
    A[s]=tmp;
```

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

clock_t Select4Exchange_vector(int *A, int N)
{
    int Min, Max;
    int L, R;
    clock_t time_start, time_stop;
    time_start = clock();
    L=0; R=N-1;
    while (L<R){
        for(int i=L; i<R+1; i++){
            if (A[i] < A[L]){
                Min=A[i];
                A[i]=A[L];
                A[L]=Min;
            }
            else
            if (A[i] > A[R]){
                Max=A[i];
                A[i]=A[R];
                A[R]=Max;
            }
        }
        L=L+1; R=R-1;
    }
    time_stop = clock();
    return time_stop - time_start;
}

clock_t Shell_1_vector(int *A, int N)
{
    int Elem, t, j, k;
    clock_t time_start, time_stop;
    time_start = clock();
    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;

    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<N; i++){
            Elem=A[i];
            j=i;
            while (j>=k && Elem<A[j-k]) {
                A[j]=A[j-k];
                j=j-k;
            }
            A[j]=Elem;
        }
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

fill.h

```
#ifndef FILL_H_INCLUDED
#define FILL_H_INCLUDED

void fill_Arr3D(int ***Arr3D, int P, int M, int N, int fill);
void fill_vector(int *vector, int N, int fill);

#endif // FILL_H_INCLUDED
```

fill.c

```
#include "fill.h"

#include <stdlib.h>
```

```
void fill_Arr3D_ordered(int ***Arr3D, int P, int M, int N)
{
   int number=0;
   for (int k=0; k<P; k++)
      for (int i=0; i<M; i++)
         for (int j=0; j<N; j++)
            Arr3D[k][i][j] = number++;
}

void fill_Arr3D_random(int ***Arr3D, int P, int M, int N)
{
   for (int k=0; k<P; k++)
      for (int i=0; i<M; i++)
         for (int j=0; j<N; j++)
            Arr3D[k][i][j] = rand()%(P*M*N);
}

void fill_Arr3D_backordered(int ***Arr3D, int P, int M, int N)
{
   int number = P*M*N;
   for (int k=0; k<P; k++)
      for (int i=0; i<M; i++)
         for (int j=0; j<N; j++)
            Arr3D[k][i][j] = number--;
}

void fill_vector_ordered(int *vector, int N){
   for(int i = 0; i < N; i++){
      vector[i] = i;
   }
}

void fill_vector_random(int *vector, int N){
   for(int i = 0; i < N; i++){
      vector[i] = rand() % N;
   }
}

void fill_vector_backordered(int *vector, int N){
   for(int i = 0; i < N; i++){
      vector[i] = N - i;
   }
}

void fill_Arr3D(int ***Arr3D, int P, int M, int N, int fill)
{
   switch(fill)
   {
   case 1:
      fill_Arr3D_ordered(Arr3D, P, M, N);
      break;
   case 2:
      fill_Arr3D_random(Arr3D, P, M, N);
      break;
   case 3:
      fill_Arr3D_backordered(Arr3D, P, M, N);
      break;
   }
}

void fill_vector(int *vector, int N, int fill)
{
   switch(fill)
   {
   case 1:
      fill_vector_ordered(vector, N);
      break;
   case 2:
      fill_vector_random(vector, N);
      break;
   case 3:
      fill_vector_backordered(vector, N);
      break;
   }
```

```
}
```

measurement.h

```
#ifndef MEASUREMENT_H_INCLUDED
#define MEASUREMENT_H_INCLUDED

#define measurements_number 28

#define rejected_number 2

#define min_max_number 3

#include <time.h>

extern clock_t Res[measurements_number];

float MeasurementProcessing();

void Arr3D_measurement(int ***Arr3D, int P, int M, int N, int alg, int fill);
void vector_measurement(int *vector, int N, int alg ,int fill);

#endif // MEASUREMENT_H_INCLUDED
```

measurement.c

```
#include "measurement.h"

#include "fill.h"
#include "sort.h"

#include <time.h>
#include <stdio.h>
clock_t Res[measurements_number];

float MeasurementProcessing()
{
  long int Sum;
  float AverageValue;

  clock_t buf;
  int L = rejected_number, R = measurements_number - 1;
  int k = rejected_number;
  for (int j=0; j < min_max_number; j++) {
        for (int i = L; i < R; i++) {
            if (Res[i] > Res[i + 1]) {
                buf = Res[i];
                Res[i] = Res[i + 1];
                Res[i + 1] = buf;
                k = i;
            }
        }
        R = k;
        for (int i = R - 1; i >= L; i--) {
            if (Res[i] > Res[i + 1]) {
                buf = Res[i];
                Res[i] = Res[i + 1];
                Res[i + 1] = buf;
                k = i;
            }
        }
        L = k + 1;
  }

  Sum=0;
  for (int i = rejected_number + min_max_number; i < measurements_number - min_max_number; i++)
    Sum = Sum + Res[i];

  AverageValue = (float)Sum/(float)(measurements_number - 2*min_max_number - rejected_number);
  return AverageValue;
}

void Arr3D_measurement(int ***Arr3D, int P, int M, int N, int alg, int fill)
{
  switch(alg)
  {
```

```
    case 1:
      {
        for (int i = 0; i < measurements_number; i++)
        {
          fill_Arr3D(Arr3D, P, M, N, fill);
          Res[i] = Select2(Arr3D, P, M, N);
        }
      }
      break;
    case 2:
      {
        for (int i = 0; i < measurements_number; i++)
        {
          fill_Arr3D(Arr3D, P, M, N, fill);
          Res[i] = Select4Exchange(Arr3D, P, M, N);

        }
      }
      break;
    case 3:
      {
        for (int i = 0; i < measurements_number; i++)
        {
          fill_Arr3D(Arr3D, P, M, N, fill);
          Res[i] = Shell_1(Arr3D, P, M, N);

        }
      }
      break;
    }
}

void vector_measurement(int *vector, int N, int alg ,int fill)
{
  switch(alg)
  {
  case 1:
    {
      for (int i = 0; i < measurements_number; i++)
      {
        fill_vector(vector, N, fill);
        Res[i] = Select2_vector(vector, N);
      }
    }
    break;
  case 2:
    {
      for (int i = 0; i < measurements_number; i++)
      {
        fill_vector(vector, N, fill);
        Res[i] = Select4Exchange_vector(vector, N);

      }
    }
    break;
  case 3:
    {
      for (int i = 0; i < measurements_number; i++)
      {
        fill_vector(vector, N, fill);
        Res[i] = Shell_1_vector(vector, N);

      }
    }
    break;
  }
}
```

Тести

Характеристики комп’ютера:

- Процесор: AMD Ryzen 5 3550H
- Оперативна пам’ять: 8 ГБ
- Операційна система: Windows 10 Pro 21H2
- Компілятор: MinGW GCC

```
Select 2
Ordered
Before sorting:
P = 0
  0   1   2   3   4   5   6   7
  8   9  10  11  12  13  14  15
 16  17  18  19  20  21  22  23
 24  25  26  27  28  29  30  31
 32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47
 48  49  50  51  52  53  54  55
 56  57  58  59  60  61  62  63

P = 1
 64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87
 88  89  90  91  92  93  94  95
 96  97  98  99 100 101 102 103
104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127

P = 2
128 129 130 131 132 133 134 135
136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191
```

```
After sorting:
P = 0
  0   1   2   3   4   5   6   7
  8   9  10  11  12  13  14  15
 16  17  18  19  20  21  22  23
 24  25  26  27  28  29  30  31
 32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47
 48  49  50  51  52  53  54  55
 56  57  58  59  60  61  62  63

P = 1
 64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87
 88  89  90  91  92  93  94  95
 96  97  98  99 100 101 102 103
104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127

P = 2
128 129 130 131 132 133 134 135
136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151
152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183
184 185 186 187 188 189 190 191
```

```
Random
Before sorting:
P = 0
 47 174 127  16 127  61  15  39
 35  19  71  15 123 158 175   0
171 131 163  57  11  72  37  60
 31 178  56 154 137  42 106  48
125  37 113 178 182  68 121 103
 39 185  61 117 117 172  85  95
163 157  14 140 142  74  64  10
 90  83  28  82 108  92 133 142

P = 1
 89  51  33  24 159  51  47   5
 83 104  27  34 155 115  79  61
157  48  77 168  55  23  59 121
113 160  61 149 176 171 137 110
 44  30  55  11 100 144  88   8
144 102   9  29  23  59 136  99
130   6   6 147 112  54 125  17
140  33   3 170 125  30  32  26

P = 2
156 175  93  18 171 100  93  56
132  59 176 110 143  76  38  23
120 101  97  87  97 113  46  91
 83  30  86  26  59  62 111  90
 83  46 184 117  94 182 101  29
 24 113 184  94 109  46  17 163
169 149  71 124  48 142 119  94
175 158 156 108  82  19  97 116
```

```
After sorting:
P = 0
 15  16  39  47  61 127 127 174
175  15   0  35 158 123  71  19
 37  57  60 171  72  11 163 131
106 154  48  31  42 137  56 178
121 178 103 125  68 182 113  37
 85 117  95  39 172 117  61 185
 64 140  10 163  74 142  14 157
133  82 142  90  92 108  28  83

P = 1
  5  24  33  47  51  51  89 159
 61  34  27  79 115 104  83 155
121 168  77  59  23  48 157  55
110 149  61 137 171 160 113 176
  8  11  55  88 144  30  44 100
 99  29   9 136  59 102 144  23
 17 147   6 125  54   6 130 112
 26 170   3  32  30  33 140 125

P = 2
 18  56  93  93 100 156 171 175
110  23 176  38  76 132 143  59
 87  91  97  46 113 120  97 101
 26  90  86 111  62  83  59  30
117  29 184 101 182  83  94  46
 94 163 184  17  46  24 109 113
124  94  71 119 142 169  48 149
108 116 156  97  19 175  82 158
```

```
Backordered
Before sorting:
P = 0
 192 191 190 189 188 187 186 185
 184 183 182 181 180 179 178 177
 176 175 174 173 172 171 170 169
 168 167 166 165 164 163 162 161
 160 159 158 157 156 155 154 153
 152 151 150 149 148 147 146 145
 144 143 142 141 140 139 138 137
 136 135 134 133 132 131 130 129

P = 1
 128 127 126 125 124 123 122 121
 120 119 118 117 116 115 114 113
 112 111 110 109 108 107 106 105
 104 103 102 101 100  99  98  97
  96  95  94  93  92  91  90  89
  88  87  86  85  84  83  82  81
  80  79  78  77  76  75  74  73
  72  71  70  69  68  67  66  65

P = 2
  64  63  62  61  60  59  58  57
  56  55  54  53  52  51  50  49
  48  47  46  45  44  43  42  41
  40  39  38  37  36  35  34  33
  32  31  30  29  28  27  26  25
  24  23  22  21  20  19  18  17
  16  15  14  13  12  11  10   9
   8   7   6   5   4   3   2   1
```

```
After sorting:
P = 0
 185 186 187 188 189 190 191 192
 177 178 179 180 181 182 183 184
 169 170 171 172 173 174 175 176
 161 162 163 164 165 166 167 168
 153 154 155 156 157 158 159 160
 145 146 147 148 149 150 151 152
 137 138 139 140 141 142 143 144
 129 130 131 132 133 134 135 136

P = 1
 121 122 123 124 125 126 127 128
 113 114 115 116 117 118 119 120
 105 106 107 108 109 110 111 112
  97  98  99 100 101 102 103 104
  89  90  91  92  93  94  95  96
  81  82  83  84  85  86  87  88
  73  74  75  76  77  78  79  80
  65  66  67  68  69  70  71  72

P = 2
  57  58  59  60  61  62  63  64
  49  50  51  52  53  54  55  56
  41  42  43  44  45  46  47  48
  33  34  35  36  37  38  39  40
  25  26  27  28  29  30  31  32
  17  18  19  20  21  22  23  24
   9  10  11  12  13  14  15  16
   1   2   3   4   5   6   7   8
```

```
Select 4 Exchange
Ordered
Before sorting:
P = 0
   0   1   2   3   4   5   6   7
   8   9  10  11  12  13  14  15
  16  17  18  19  20  21  22  23
  24  25  26  27  28  29  30  31
  32  33  34  35  36  37  38  39
  40  41  42  43  44  45  46  47
  48  49  50  51  52  53  54  55
  56  57  58  59  60  61  62  63

P = 1
  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87
  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103
 104 105 106 107 108 109 110 111
 112 113 114 115 116 117 118 119
 120 121 122 123 124 125 126 127

P = 2
 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151
 152 153 154 155 156 157 158 159
 160 161 162 163 164 165 166 167
 168 169 170 171 172 173 174 175
 176 177 178 179 180 181 182 183
 184 185 186 187 188 189 190 191
```

```
After sorting:
P = 0
   0   1   2   3   4   5   6   7
   8   9  10  11  12  13  14  15
  16  17  18  19  20  21  22  23
  24  25  26  27  28  29  30  31
  32  33  34  35  36  37  38  39
  40  41  42  43  44  45  46  47
  48  49  50  51  52  53  54  55
  56  57  58  59  60  61  62  63

P = 1
  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87
  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103
 104 105 106 107 108 109 110 111
 112 113 114 115 116 117 118 119
 120 121 122 123 124 125 126 127

P = 2
 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151
 152 153 154 155 156 157 158 159
 160 161 162 163 164 165 166 167
 168 169 170 171 172 173 174 175
 176 177 178 179 180 181 182 183
 184 185 186 187 188 189 190 191
```

```
Random
Before sorting:
P = 0
  55  35  51 125  99  80  89 128
 116  79  72 180 151 168  50  78
 123 160  96   9  70  22   0 163
  70 171   0  40  42 101  91  50
  50  22 115 177  36  56 161 104
   1  28   9  56 183 132 177  33
 152  77 147   6  75  16  48 112
   2 139 168 150  43 188 136  28

P = 1
  43  78 163 153  72  55  36  29
  37 161  98 115 180 136 181  99
 103  35  73  59  39   6 112  15
   9 135 189 191 126  32  78 184
 138  20 169  62 119  21  11 168
 138 166  60 172 118 125  39 156
  80 171 105 113 131 126 104  72
   5 103 166 106   9  88  21 144

P = 2
  54  48 173 166 153  90 189  16
  22  50 126 107 101 106  46 149
  59  45 156  47  69 130 127 161
 156  50 139  31 118  45  70 107
 153 106  24  92  22 142  98 157
 179  79  81 184 169  25 115  85
  16  48 137 125  23  88 157 100
  56 114  22  39 107  39  73 146
```

```
After sorting:
P = 0
  35  51  55  80  89  99 125 128
  79  72 116 168  50 151 180  78
 160  96 123  22   0  70   9 163
 171   0  70 101  91  42  40  50
  22 115  50  56 161  36 177 104
  28   9   1 132 177 183  56  33
  77 147 152  16  48  75   6 112
 139 168   2 188 136  43 150  28

P = 1
  29  36  43  55  72  78 153 163
  99 181  37 136 180 161 115  98
  15 112 103   6  39  35  59  73
 184  78   9  32 126 135 191 189
 168  11 138  21 119  20  62 169
 156  39 138 125 118 166 172  60
  72 104  80 126 131 171 113 105
 144  21   5  88   9 103 106 166

P = 2
  16  48  54  90 153 166 173 189
 149  50  22 106 101 107 126  46
 161  45  59 130  69  47 156 127
 107  50 156  45 118  31 139  70
 157 106 153 142  22  92  24  98
  85  79 179  25 169 184  81 115
 100  48  16  88  23 125 137 157
 146 114  56  39 107  39  22  73
```

```
Backordered
Before sorting:
P = 0
 192 191 190 189 188 187 186 185
 184 183 182 181 180 179 178 177
 176 175 174 173 172 171 170 169
 168 167 166 165 164 163 162 161
 160 159 158 157 156 155 154 153
 152 151 150 149 148 147 146 145
 144 143 142 141 140 139 138 137
 136 135 134 133 132 131 130 129

P = 1
 128 127 126 125 124 123 122 121
 120 119 118 117 116 115 114 113
 112 111 110 109 108 107 106 105
 104 103 102 101 100  99  98  97
  96  95  94  93  92  91  90  89
  88  87  86  85  84  83  82  81
  80  79  78  77  76  75  74  73
  72  71  70  69  68  67  66  65

P = 2
  64  63  62  61  60  59  58  57
  56  55  54  53  52  51  50  49
  48  47  46  45  44  43  42  41
  40  39  38  37  36  35  34  33
  32  31  30  29  28  27  26  25
  24  23  22  21  20  19  18  17
  16  15  14  13  12  11  10   9
   8   7   6   5   4   3   2   1
```

```
After sorting:
P = 0
 185 186 187 188 189 190 191 192
 177 178 179 180 181 182 183 184
 169 170 171 172 173 174 175 176
 161 162 163 164 165 166 167 168
 153 154 155 156 157 158 159 160
 145 146 147 148 149 150 151 152
 137 138 139 140 141 142 143 144
 129 130 131 132 133 134 135 136

P = 1
 121 122 123 124 125 126 127 128
 113 114 115 116 117 118 119 120
 105 106 107 108 109 110 111 112
  97  98  99 100 101 102 103 104
  89  90  91  92  93  94  95  96
  81  82  83  84  85  86  87  88
  73  74  75  76  77  78  79  80
  65  66  67  68  69  70  71  72

P = 2
  57  58  59  60  61  62  63  64
  49  50  51  52  53  54  55  56
  41  42  43  44  45  46  47  48
  33  34  35  36  37  38  39  40
  25  26  27  28  29  30  31  32
  17  18  19  20  21  22  23  24
   9  10  11  12  13  14  15  16
   1   2   3   4   5   6   7   8
```

```
Shell 1
Ordered
Before sorting:
P = 0
   0   1   2   3   4   5   6   7
   8   9  10  11  12  13  14  15
  16  17  18  19  20  21  22  23
  24  25  26  27  28  29  30  31
  32  33  34  35  36  37  38  39
  40  41  42  43  44  45  46  47
  48  49  50  51  52  53  54  55
  56  57  58  59  60  61  62  63

P = 1
  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87
  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103
 104 105 106 107 108 109 110 111
 112 113 114 115 116 117 118 119
 120 121 122 123 124 125 126 127

P = 2
 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151
 152 153 154 155 156 157 158 159
 160 161 162 163 164 165 166 167
 168 169 170 171 172 173 174 175
 176 177 178 179 180 181 182 183
 184 185 186 187 188 189 190 191
```

```
After sorting:
P = 0
   0   1   2   3   4   5   6   7
   8   9  10  11  12  13  14  15
  16  17  18  19  20  21  22  23
  24  25  26  27  28  29  30  31
  32  33  34  35  36  37  38  39
  40  41  42  43  44  45  46  47
  48  49  50  51  52  53  54  55
  56  57  58  59  60  61  62  63

P = 1
  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79
  80  81  82  83  84  85  86  87
  88  89  90  91  92  93  94  95
  96  97  98  99 100 101 102 103
 104 105 106 107 108 109 110 111
 112 113 114 115 116 117 118 119
 120 121 122 123 124 125 126 127

P = 2
 128 129 130 131 132 133 134 135
 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151
 152 153 154 155 156 157 158 159
 160 161 162 163 164 165 166 167
 168 169 170 171 172 173 174 175
 176 177 178 179 180 181 182 183
 184 185 186 187 188 189 190 191
```

```
Random
Before sorting:
P = 0
  90   9  82 165  86 184 164 152
 136   3  29  28 170  80  30 100
  55  62  89 100 159   8 171 152
  64  45 109   9  18  14 133 140
  97 151 130  75 129  97  41 136
 125 151  73  94  80 152  22 138
  88 157 179 171 135  92 145  70
 156 108  56  77  81 169 163  33

P = 1
  22 154  17  21 127  17  89  53
 186  19 190   7 131 185 131  16
  91  86 129  26 182  90 118 117
  51 119 160  60 146  66 138  89
  34  91 103 139  56 144 157 103
 166 127  99 157 107  59 142 189
 104  49  40  58 147 140 131  45
 175 149 140  93  60 159  99 125

P = 2
 171 162  42  54  84  38  29 167
 171  35 161  42 177  38 159 154
 103  54  82 146  72 120 159 119
  55  16  38 120  55   9  86  84
  90 119   5  29 124  90 128 124
 112   6  31  54 123 129 157  45
 128 173 167  42 124 103 179  89
  52 174 147  21 171   8 138  38
```

```
After sorting:
P = 0
   9  82  86  90 152 164 165 184
   3  29 170 136 100  30  28  80
  62  89 159  55 152 171 100   8
  45 109  18  64 140 133   9  14
 151 130 129  97 136  41  75  97
 151  73  80 125 138  22  94 152
 157 179 135  88  70 145 171  92
 108  56  81 156  33 163  77 169

P = 1
  17  17  21  22  53  89 127 154
 190 185   7 186  16 131 131  19
 129  90  26  91 117 118 182  86
 160  66  60  51  89 138 146 119
 103 144 139  34 103 157  56  91
  99  59 157 166 189 142 107 127
  40 140  58 104  45 131 147  49
 140 159  93 175 125  99  60 149

P = 2
  29  38  42  54  84 162 167 171
 159  38 161  42 177  35 154 171
 159 120  82 146  72  54 119 103
  86   9  38 120  55  16  84  55
 128  90   5  29 124 119 124  90
 157 129  31  54 123   6  45 112
 179 103 167  42 124 173  89 128
 138   8 147  21 171 174  38  52
```

```
Backordered
Before sorting:
P = 0
 192 191 190 189 188 187 186 185
 184 183 182 181 180 179 178 177
 176 175 174 173 172 171 170 169
 168 167 166 165 164 163 162 161
 160 159 158 157 156 155 154 153
 152 151 150 149 148 147 146 145
 144 143 142 141 140 139 138 137
 136 135 134 133 132 131 130 129

P = 1
 128 127 126 125 124 123 122 121
 120 119 118 117 116 115 114 113
 112 111 110 109 108 107 106 105
 104 103 102 101 100  99  98  97
  96  95  94  93  92  91  90  89
  88  87  86  85  84  83  82  81
  80  79  78  77  76  75  74  73
  72  71  70  69  68  67  66  65

P = 2
  64  63  62  61  60  59  58  57
  56  55  54  53  52  51  50  49
  48  47  46  45  44  43  42  41
  40  39  38  37  36  35  34  33
  32  31  30  29  28  27  26  25
  24  23  22  21  20  19  18  17
  16  15  14  13  12  11  10   9
   8   7   6   5   4   3   2   1
```

```
After sorting:
P = 0
 185 186 187 188 189 190 191 192
 177 178 179 180 181 182 183 184
 169 170 171 172 173 174 175 176
 161 162 163 164 165 166 167 168
 153 154 155 156 157 158 159 160
 145 146 147 148 149 150 151 152
 137 138 139 140 141 142 143 144
 129 130 131 132 133 134 135 136

P = 1
 121 122 123 124 125 126 127 128
 113 114 115 116 117 118 119 120
 105 106 107 108 109 110 111 112
  97  98  99 100 101 102 103 104
  89  90  91  92  93  94  95  96
  81  82  83  84  85  86  87  88
  73  74  75  76  77  78  79  80
  65  66  67  68  69  70  71  72

P = 2
  57  58  59  60  61  62  63  64
  49  50  51  52  53  54  55  56
  41  42  43  44  45  46  47  48
  33  34  35  36  37  38  39  40
  25  26  27  28  29  30  31  32
  17  18  19  20  21  22  23  24
   9  10  11  12  13  14  15  16
   1   2   3   4   5   6   7   8
```

Результати

Випадок 1.

Залежність часу роботи алгоритмів від довжини стовпчиків масива

P = const = 3, M = var, N = const = 77500

```
P = 3
M = 1
N = 77500
                    ordered    random     backordered
Select 2            29605.29   30321.57   30293.43
Select 4 Exchange   26929.43   40996.14   26934.29
Shell 1             32.71      97.86      41.43
```

```
P = 3
M = 2
N = 77500
                    ordered    random     backordered
Select 2            27497.00   28921.43   30816.14
Select 4 Exchange   25722.29   48427.00   27752.29
Shell 1             68.57      178.71     91.29
```

```
P = 3
M = 4
N = 77500
                    ordered    random     backordered
Select 2            26300.29   29302.57   28619.71
Select 4 Exchange   25196.71   66031.57   30752.29
Shell 1             150.57     275.29     175.86
```

```
P = 3
M = 8
N = 77500
                    ordered    random     backordered
Select 2            25427.29   25350.29   35638.00
Select 4 Exchange   23095.14   92637.71   24846.86
Shell 1             269.29     435.57     271.57
```

```
P = 3
M = 16
N = 77500
                    ordered    random     backordered
Select 2            25189.75   27540.13   29057.25
Select 4 Exchange   22942.88   174614.88  23009.00
Shell 1             450.25     806.25     523.63
```

```
P = 3
M = 32
N = 77500
                    ordered    random     backordered
Select 2            29108.67   29725.00   30265.67
Select 4 Exchange   26616.00   478494.66  25195.00
Shell 1             965.00     1703.67    1069.67
```

```
P = 3
M = 64
N = 77500
                    ordered    random     backordered
Select 2            28167.13   28688.75   29098.13
Select 4 Exchange   24798.38   1027372.8825838.50
Shell 1             2040.88    3879.00    2420.00
```

```
P = 3
M = 128
N = 77500
                    ordered    random     backordered
Select 2            28577.50   30356.25   30137.75
Select 4 Exchange   25780.25   2408739.2525421.25
Shell 1             3937.25    8049.25    4686.75
```

```
N = 77500
                    ordered    random     backordered
Select 2            6951.20    7030.10    7467.90
Select 4 Exchange   4586.30    7550.60    4617.30
Shell 1             6.20       21.70      7.30
```

Випадок 2.

Залежність часу роботи алгоритмів від форми перерізів масива

P = const = 3, M = var, N = var, M*N = 1000000

```
P = 3
M = 10
N = 100000
                    ordered    random     backordered
Select 2            46987.50   42702.00   71113.75
Select 4 Exchange   64432.75   418737.75  46768.50
Shell 1             420.75     734.75     477.75
```

```
P = 3
M = 100
N = 10000
                    ordered    random     backordered
Select 2            496.13     542.38     496.23
Select 4 Exchange   420.17     44197.98   487.52
Shell 1             359.93     586.92     440.82
```

```
P = 3
M = 1000
N = 1000
                    ordered    random     backordered
Select 2            49.30      112.12     50.23
Select 4 Exchange   4.18       5811.38    25.47
Shell 1             314.80     477.07     376.18
```

```
P = 3
M = 10000
N = 100
                    ordered    random     backordered
Select 2            37.25      56.63      39.50
Select 4 Exchange   0.00       538.40     28.27
Shell 1             156.22     265.38     221.58
```

```
P = 3
M = 100000
N = 10
                    ordered    random     backordered
Select 2            106.60     109.38     108.60
Select 4 Exchange   0.00       135.15     56.93
Shell 1             170.55     301.77     282.63
```

Випадок 2.

Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масива при однаковій загальній кількості ключів у всьому масиві

P = var, M = const = 10, N = var, P*N = 400000

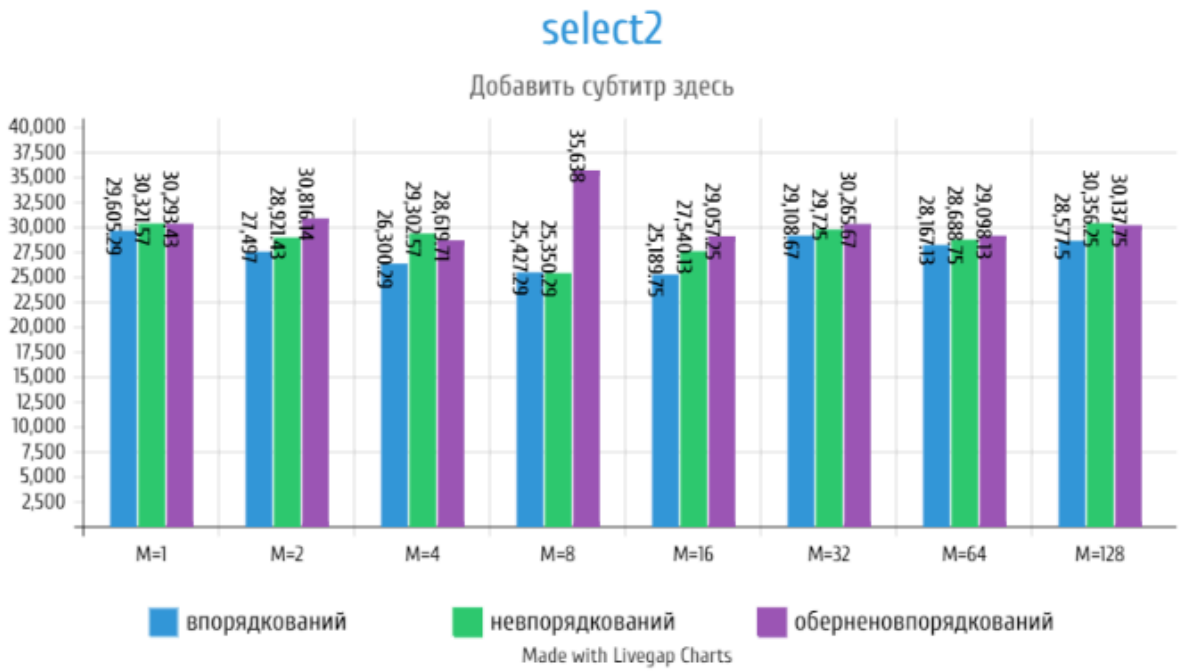
```
P = 20
M = 10
N = 20000
                    ordered    random     backordered
Select 2            13862.90   13737.40   14351.70
Select 4 Exchange   13208.60   68607.40   12168.70
Shell 1             526.40     934.10     605.30
```

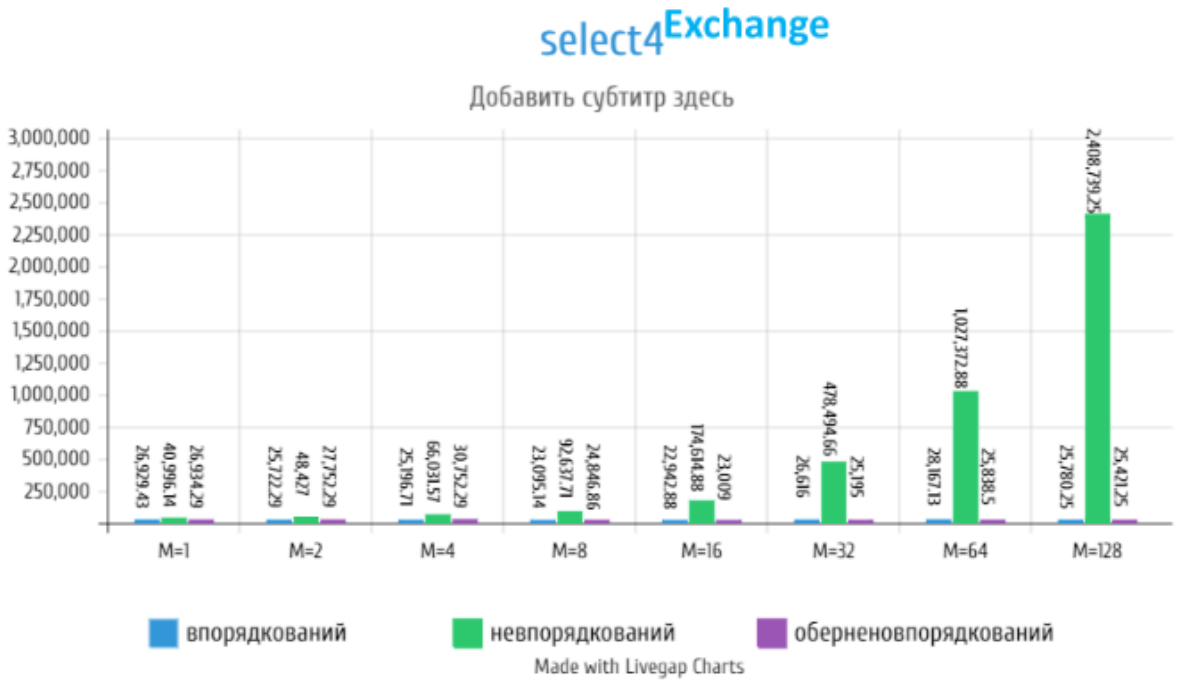
```
P = 200
M = 10
N = 2000
                    ordered    random     backordered
Select 2            1353.14    1396.14    1402.43
Select 4 Exchange   1181.57    6823.00    1243.86
Shell 1             317.43     505.71     371.57
```

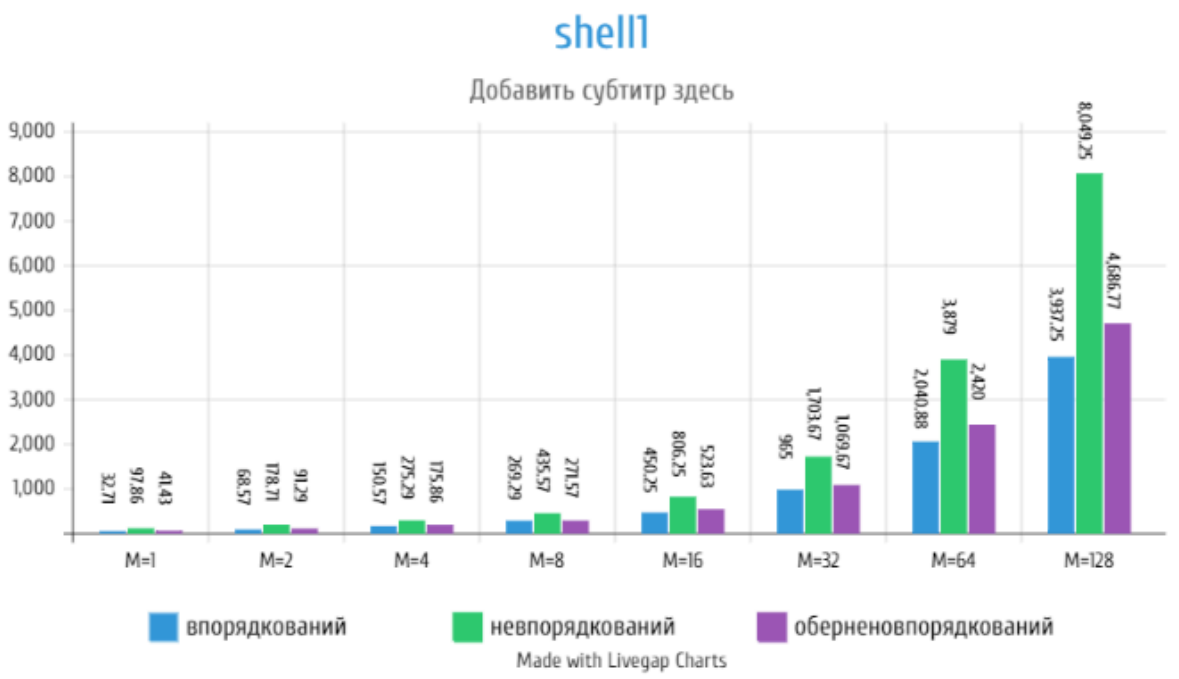
```
P = 2000
M = 10
N = 200
                    ordered    random     backordered
Select 2            173.14     196.29     189.00
Select 4 Exchange   128.57     713.00     158.86
Shell 1             219.57     328.57     260.57
```

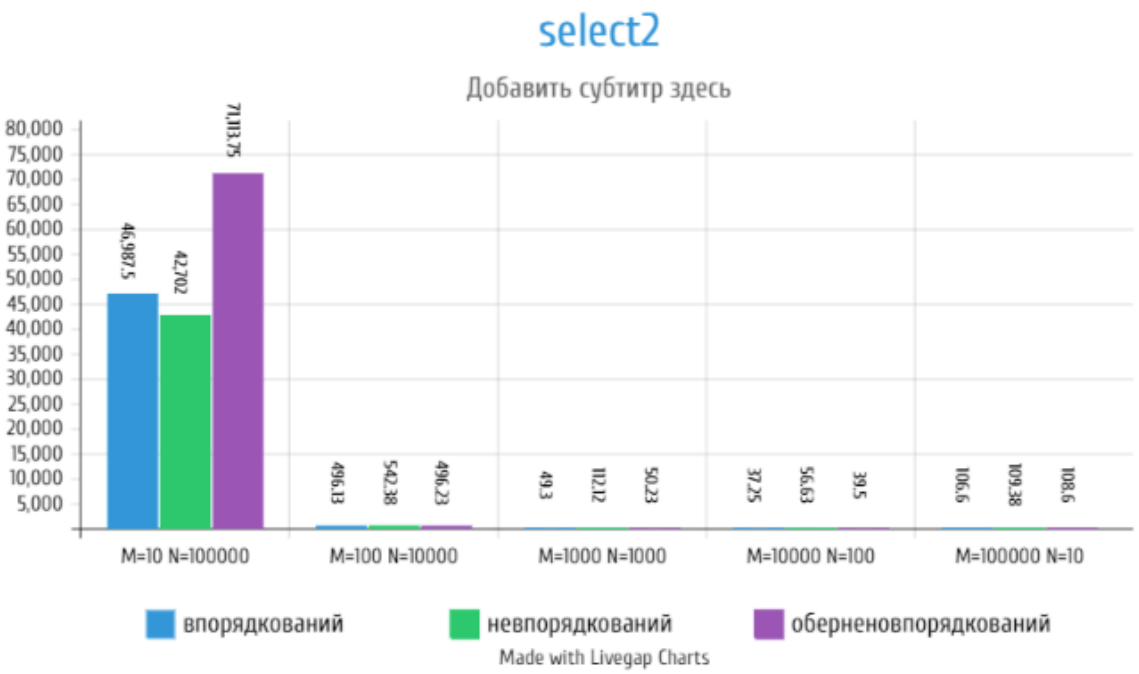
```
P = 20000
M = 10
N = 20
                    ordered    random     backordered
Select 2            65.29      72.43      61.00
Select 4 Exchange   19.43      102.71     42.71
Shell 1             103.14     153.43     159.43
```

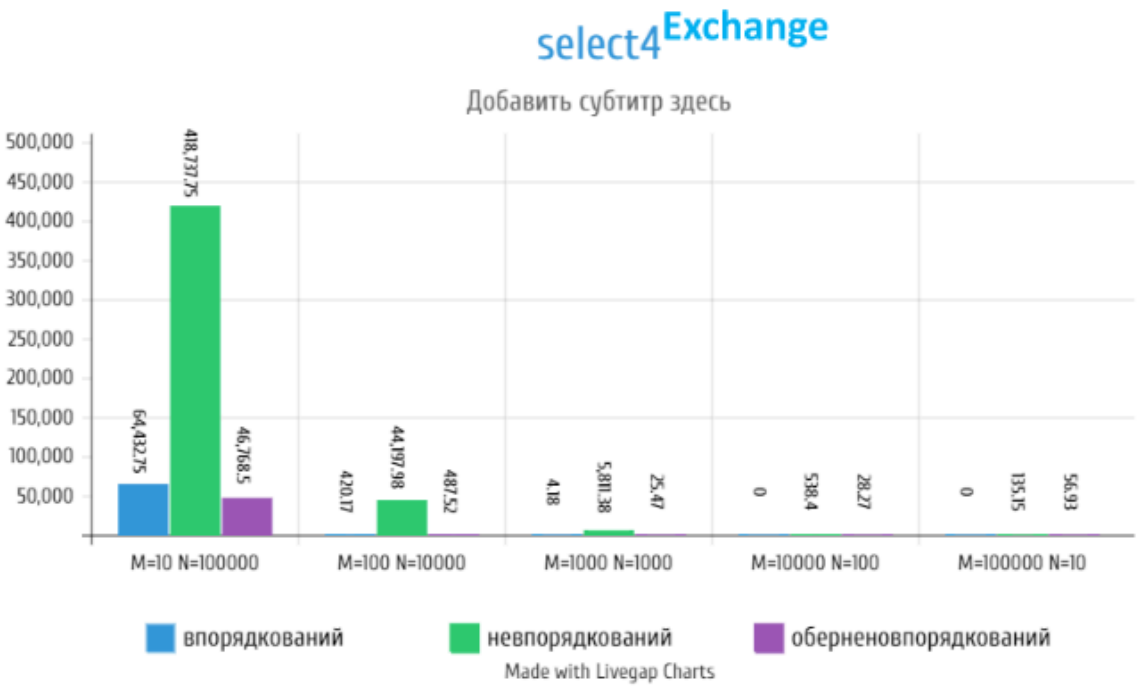
Візуалізація результатів

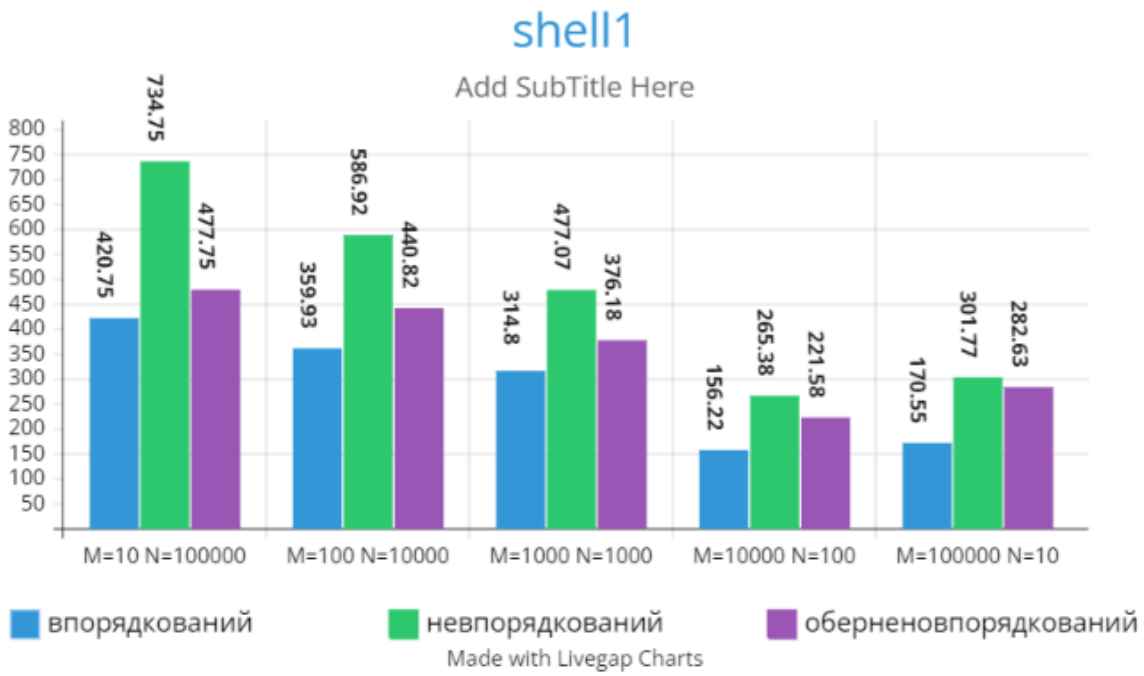


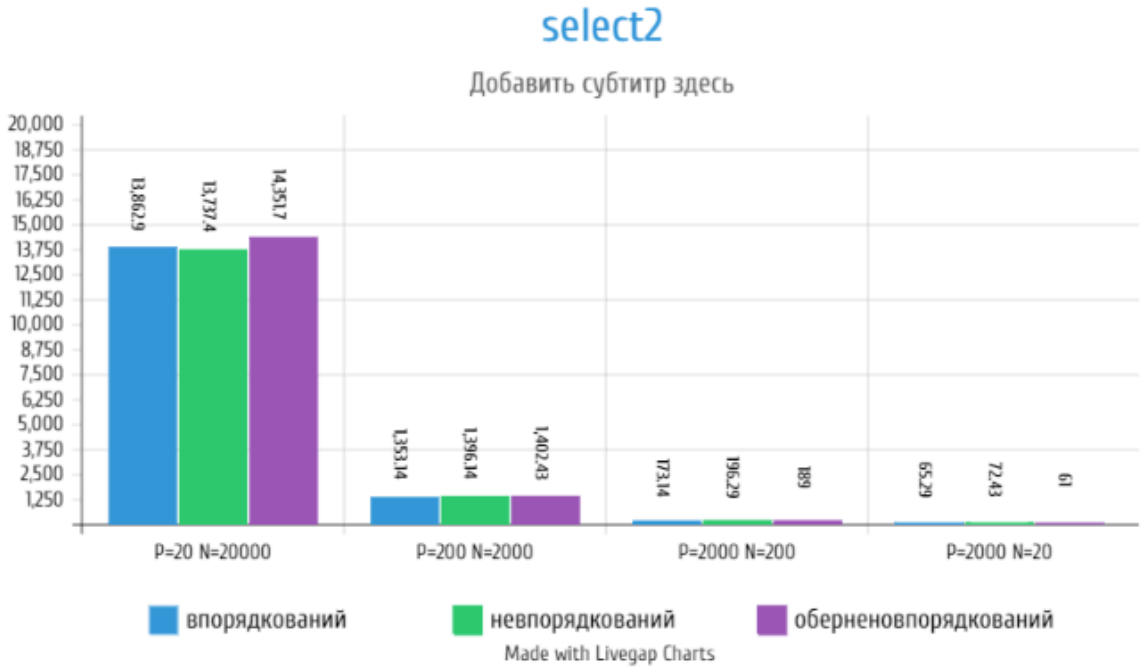

select4Exchange
Добавить субтитр здесь
3,000,000
2,750,000
2,500,000
2,250,000
2,000,000
1,750,000
1,500,000
1,250,000
1,000,000
750,000
500,000
250,000
26,929.43
40,996.14
26,934.29
25,722.29
48,427
27,752.29
25,196.71
66,031.57
30,752.29
23,095.14
92,637.71
24,846.86
22,942.88
174,614.88
23,009
26,616
478,494.66
25,195
28,167.13
1,027,372.88
25,838.5
25,780.25
2,408,739.25
25,421.25
M=1
M=2
M=4
M=8
M=16
M=32
M=64
M=128
впорядкований
невпорядкований
оберненовпорядкований
Made with Livegap Charts

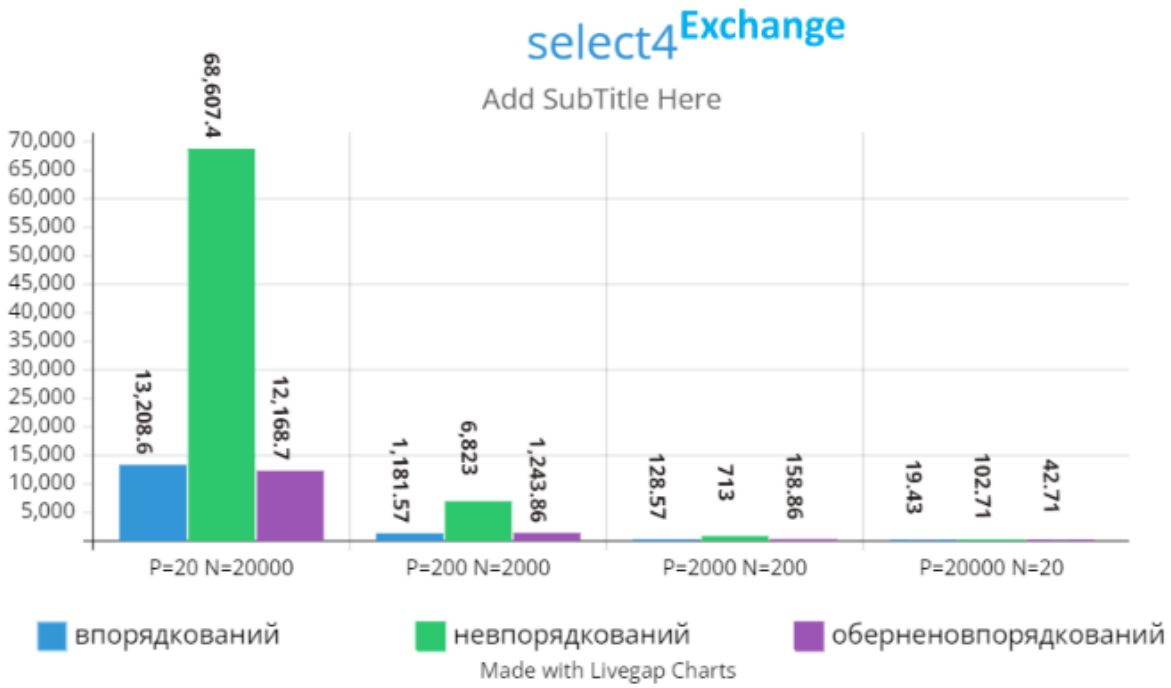

shell1
Добавить субтитр здесь
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
32.71
97.86
41.43
68.57
178.71
91.29
150.57
275.29
175.86
269.29
435.57
271.57
450.25
806.25
523.63
965
1,703.67
1,069.67
2,040.88
3,879
2,420
3,937.25
8,049.25
4,686.77
M=1
M=2
M=4
M=8
M=16
M=32
M=64
M=128
впорядкований
невпорядкований
оберненовпорядкований
Made with Livegap Charts

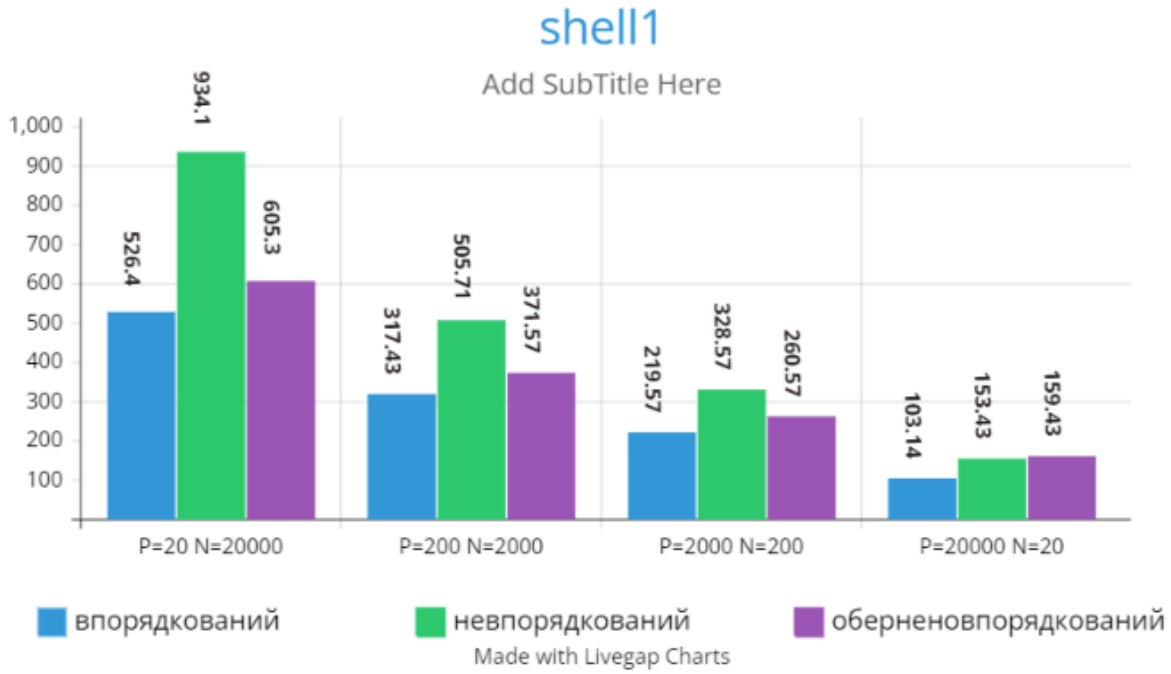

select2
Добавить субтитр здесь
80,000
75,000
70,000
65,000
60,000
55,000
50,000
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
46,987.5
42,702
71,113.75
496.13
542.38
496.23
49.3
112.12
50.23
37.25
56.63
39.5
106.6
109.38
108.6
M=10 N=100000
M=100 N=10000
M=1000 N=1000
M=10000 N=100
M=100000 N=10
впорядкований
невпорядкований
оберненовпорядкований
Made with Livegap Charts


select4Exchange
Добавить субтитр здесь
500,000
450,000
400,000
350,000
300,000
250,000
200,000
150,000
100,000
50,000
64,432.75
418,737.75
46,768.5
420.17
44,197.98
487.52
4.18
5,811.38
25.47
0
538.4
28.27
0
135.15
56.93
M=10 N=100000
M=100 N=10000
M=1000 N=1000
M=10000 N=100
M=100000 N=10
впорядкований
невпорядкований
оберненовпорядкований
Made with Livegap Charts


shell1
Add SubTitle Here
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
420.75
734.75
477.75
359.93
586.92
440.82
314.8
477.07
376.18
156.22
265.38
221.58
170.55
301.77
282.63
M=10 N=100000
M=100 N=10000
M=1000 N=1000
M=10000 N=100
M=100000 N=10
впорядкований
невпорядкований
оберненовпорядкований
Made with Livegap Charts


select2
Добавить субтитр здесь
20,000
18,750
17,500
16,250
15,000
13,750
12,500
11,250
10,000
8,750
7,500
6,250
5,000
3,750
2,500
1,250
13,862.9
13,737.4
14,351.7
1,353.14
1,396.14
1,402.43
173.14
196.29
189
65.29
72.43
61
P=20 N=20000
P=200 N=2000
P=2000 N=200
P=2000 N=20
впорядкований
невпорядкований
оберненовпорядкований
Made with Livegap Charts





Порівняльний аналіз

В ході виконання курсової роботи було реалізовано 3 алгоритми:

- Алгоритм сортування №2 методу прямого вибору
- Гібридний алгоритм "вибір№4 – обмін"
- Алгоритм №1 методу сортування Шелла

Дані алгоритми були використані при сортування одновимірних та багатовимірних масивів. За умовою завдання потрібно відсортувати окремо кожен переріз, переставляючи стовпчики, в якості ключів сортування використовувався перший елемент кожного стовпчика.

З метою подальшого аналізу були проведені виміри для 3-ох випадків дослідження:

- Залежність часу роботи алгоритмів від довжини стовпчика.
- Залежність часу роботи алгоритмів від форми перерізу масива.
- Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масива при однаковій загальній кількості ключів у всьому масиві.

Порівняльна характеристика алгоритмів

1. **Алгоритм сортування №2 методу прямого вибору**

Час роботи алгоритму не суттєво змінювався від довжини стовпчика, також вимірювання показали, що час роботи для відсортованого, невідсортованого, обернено відсортованого майже такий самий, оскільки навіть якщо елемент стоїть на потрібному місці, він всеодно переставляється сам з собою. Цей алгоритм у будь якому разі здійснить n-1 перестановку

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

Це алгоритм на відміну від попереднього краще справляється у випадку відсортованого або обернено відсортованого масиву. Але у випадку збільшення довжини стовпчиків час роботи цього алгоритму для невідсортованого масиву дуже сильно зростає, а час роботи обернено відсортованого залишається стабільним.

3. **Алгоритм №1 методу сортування Шелла**

Найшвидший майже у всіх випадках алгоритм. При збільшені довжини стовпчика або менших по довжині масивах різниця між іншими алгоритмами стає не такою суттєвою.

Порівнняння часу роботи алгоритмів в залежності від розмірів та форми масиву

Випадок дослідження I.

Тут Алгоритм сортування №2 показує свою стабільність, в той же час як час роботи у випадку невідсортованого масиву у Гібридного алгоритму "вибір№4 – обмін" кратно зростає. Та зростає для всіх випадків відсортованості для Алгоритму №1 методу сортування Шелла, хоча навіть при цьому даний алгоритм є найшвидшим в цьому випадку.

Випадок дослідження II.

В даному випадку найгірше спочатку себе показали Алгоритм сортування методом прямого вибору та Гібридний алгоритм "вибір№4 – обмін", але при зменшенні кількості ключів, та при збільшенні дожини стовпців Алгоритм сортування методом прямого вибору став найкращим, на другому місці Гібридний алгоритм "вибір№4 – обмін" і тільки потім сортування методом Шелла.

Випадок дослідження III.

Знову при великій кількості ключів в масиві сортування методом Шелла є в дестяки разів швидшим за інші алгоритми, але при зменшенні кількості ключів різниця стає не настільки суттєвою, а потім навпаки інші алгоритми стають в два рази швидшими.

Висновок

Алгоритм сортування №2 методу прямого вибору

Даний алгоритм з однаковою швидкістю сортує масиви з різною можливою відсортованістю. Хоча він і є досить повільним для великих довгих масивів, але для коротких або краще за метод сортування Шелла, або не сильно відстає від нього. Цей алгоритм найкраще підходить для роботи з малею кількість ключів сортування та довгими стовпчиками.

Гібридний алгоритм "вибір№4 – обмін"

При невеликих значення довжини стовпчика хоч і гірше, але на набагато з інші алгоритми у випадку невідсортованості, у інших випадках алгоритм на рівні з іншим. Алгоритм не підходить для сортування масивів з великою кількістю ключів сортування та довгими стовпчиками.

Алгоритм №1 методу сортування Шелла

В більшості випадків цей алгоритм є найшвидшим. Чим довший масив там більший розрив в порівнянні з іншими алгоритмами. Він виявився настільки швидкий завдяки поділу сортування на етапи і сортуванню елементів на певній відстанні, що і зробило його таким швидким. Даний алгоритм є ефективним при будь яких формах масиву, але найкраще в порівнянні з іншими він себе показує при великих значеннях довжини масиву.

Список літератури

1. YouTube канал «Марченко Олександр Іванович».
2. Конспект лекцій з СДА.