**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих**

**комп’ютерних систем**

**КУРСОВА РОБОТА**

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

Виконав: Землянський Едуард

Група: КB-22

Допущений до захисту

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

2 семестр 2022/2023 навч. року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

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

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування і спеціалізованих**

**комп’ютерних систем**

|  |  |
| --- | --- |
| Узгоджено  Керівник роботи  \_\_\_\_\_\_\_\_\_ / Марченко О.І. / | ЗАХИЩЕНА "\_\_\_"\_\_\_\_\_\_20\_\_р.  з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_ / Марченко О.І. / |

***Дослідження ефективності методів сортування***

***(назви конкретних методів сортування)***

***на багатовимірних масивах***

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

(підпис)

Прізвище Ім’я По батькові

\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

**Варіант № 94**

**Задача**

5. Впорядкувати тривимірний масив Аrr3D [P,M,N] таким чи-

ном: переставити перерізи масива за незменшенням сум їх

елементів.

**Умова задачі за варіантом.**

**Досліджувані методи та алгоритми**

Алгоритм сортування №3 методу прямої вставки з лінійним

пошуком місця вставки від елемента, що вставляється, або «спра-

ва», з бар’єром

Алгоритм сортування №6 методу прямого вибору (додаток 1,

Алгоритм методу «швидкого сортування» (сортування Хоара)

**Перелік методів та алгоритмів за варіантом.**

3, 10, 24

**Перелік способів обходу за варіантом.**

7. В якості першого етапу сортування сформувати додатковий

вектор Sum, довжина якого дорівнює кількості перерізів і

значеннями якого є суми елементів відповідних перерізів.

Використовуючи елементи вектора Sum як ключі сортуван-

ня, 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 І. Залежність часу роботи алгоритмів від

розміру перерізів масива

Рекомендовані розміри масива для досліджень:

Кількість ключів (перерізів) P = const = 15000

Форма перерізу – однакова (квадрат)

M = var, N = var, M = N

Випадок дослідження ІІ. Залежність часу роботи алгоритмів

від форми перерізів масива

Рекомендовані розміри масива для досліджень:

Кількість ключів (перерізів) P = const = 5000

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

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

**Перелік випадків дослідження, які потрібно розглянути для**

**заданої за варіантом задачі згідно розділу «Випадки дослідження»**

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

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

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

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

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

Вектор довжиною P = 15000

1) M = 2; N = 800;

2) M = 4; N = 400;

3) M = 8; N = 200;

4) M = 16; N = 100;

5) M = 100; N = 16;

6) M = 200; N = 8;

7) M = 400; N = 4;

8) M = 800; N = 2;

Вектор довжиною P = 5000

**Опис теоретичних положень**

**Алгоритм №3 (Insertion Sort)**. Метод сортування вставками - це алгоритм сортування, що працює за принципом вставки нових елементів у відсортовану послідовність елементів. Алгоритм працює наступним чином:

Часова складність - O(n^2), де n - кількість елементів у списку.

Потребує O(1) додаткової пам'яті, оскільки сортування відбувається в межах вхідного списку.

Ефективний для невеликих списків, але може бути дуже повільним для великих списків.

Алгоритм сортування вставками може сортувати впорядкований масив довше, ніж невпорядкований.

Це трапляється через те, що алгоритм сортування вставками базується на перевірці та переміщенні елементів, щоб вставити їх у відсортовану послідовність. У випадку впорядкованого масиву, всі елементи вже будуть відсортовані, тому кожен новий елемент буде порівнюватися з усіма елементами, що передують йому. Це робить сортування довшим, ніж у випадку невпорядкованого масиву, де більшість елементів будуть переміщені на початок послідовності.

Хоча алгоритм сортування вставками може бути менш ефективним для впорядкованих масивів, він все ще є корисним для частково відсортованих масивів або для випадків, коли елементи масиву додаються поступово в хронологічному порядку.

**Алгоритм №10 (Selection Sort)**. Він працює шляхом пошуку мінімального елемента в масиві та обміну його з першим елементом. Потім пошук мінімального елемента відбувається для підмасиву, який складається з усіх елементів масиву, за винятком першого, і т.д.

Аналогічно до сортування вставками, час виконання сортування вибором залежить від впорядкованості вхідного масиву. Якщо масив вже впорядкований, то кількість обмінів буде дорівнювати 0, але складність алгоритму залишиться O(N^2). Якщо масив випадковий або обернено впорядкований, то кількість обмінів буде максимальною і дорівнюватиме O(N^2).

Однак, на відміну від сортування вставками, сортування вибором завжди здійснює однакову кількість порівнянь між елементами масиву. Тому, якщо порівняння між елементами масиву є більш витратними, ніж обміни елементами, то сортування вибором може працювати краще, ніж сортування вставками.

Загальна складність сортування вибором становить O(N^2), що робить його менш ефективним для великих масивів, але при цьому алгоритм дуже простий для реалізації та зрозумілий для розуміння. Також, він може бути корисним для невеликих масивів або як частина складніших алгоритмів сортування.

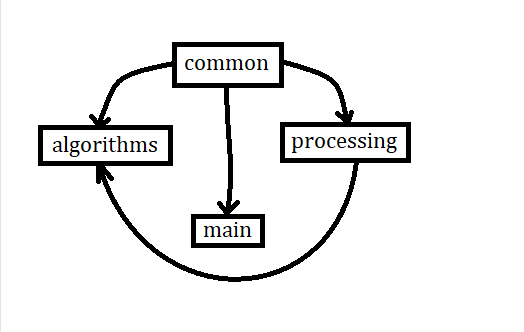
**Алгоритм №24 (QuickSort)**

Цей алгоритм сортування є реалізацією швидкого сортування для тривимірного масиву з паралельними змінами порядку розрізів головної матриці. Він використовує метод роздільного сортування, що дає швидкість O(n\*log n) у середньому, але може досягати O(n^2) у найгіршому випадку.

Сам алгоритм має квадратичну складність у найгіршому випадку, коли масив вже впорядкований в зворотному порядку, оскільки в цьому випадку вибір середнього елементу для опорного не дозволяє ефективно розділити масив на дві частини. У середньому він працює швидше за сортування вибором і вставкою, але повільніше за сортування злиттям та Шелла.

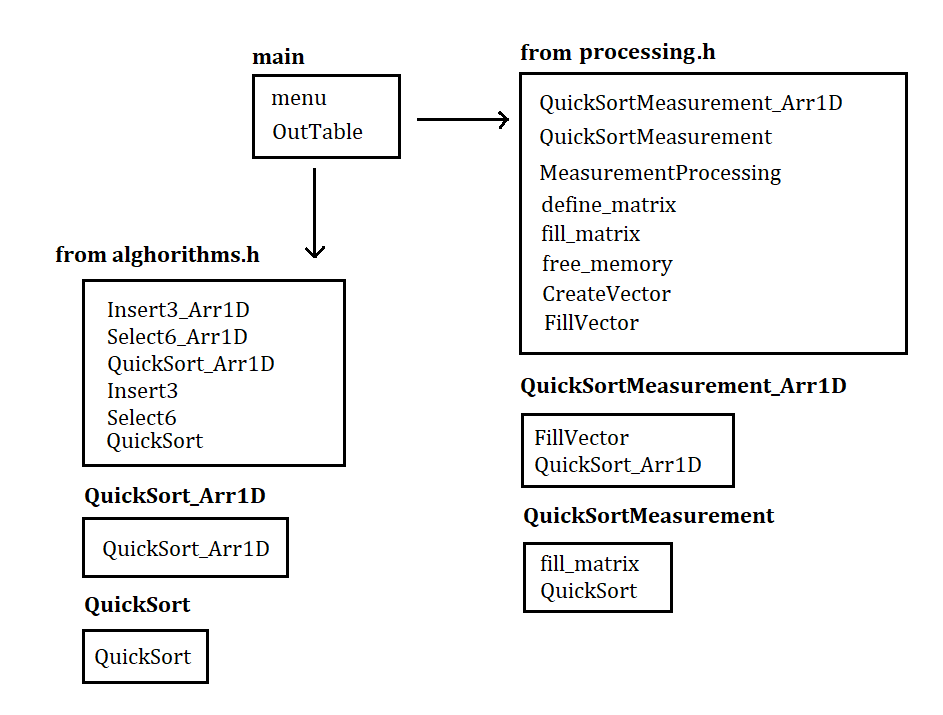
У випадку, коли масив **впорядкований** **по зростанню елементів**, алгоритм QuickSort може працювати не ефективно, якщо використовується стандартний підхід до вибору опорного елемента. У такому випадку опорний елемент завжди буде вибиратись як елемент з середини масиву, що може призвести до того, що алгоритм буде проводити багато непотрібних порівнянь та обмінів елементів, що збільшить час виконання.

**Схема імпорту/експорту модулів програми курсової роботи.**



**Структурна схема взаємовикликів процедур та функцій про-**

**грами курсової роботи.**



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

int main() – Головна функція

void OutTable(float \* measurements) – Вивід таблиці з результатами.

Параметри:

float \* measurements – список з числами типу float, які є кінечними результатами вимірів.

clock\_t Insert3\_Arr1D(int \*Vector, int P) – Функція у якій реалізовано алгоритм сортування №3 для одновимірного масиву.

*int \* Vector – одновимірний масив з елементами, які потрібно відсортувати*

*int P – довжина масиву*

clock\_t Select6\_Arr1D (int \*A, int N) - Функція у якій реалізовано алгоритм сортування №10 для одновимірного масиву.

*int \*A – одновимірний масив з елементами, які потрібно відсортувати*

*int N – довжина масиву*

void QuickSort\_Arr1D(int L, int R, int \* Vector) - Функція у якій реалізовано алгоритм сортування №24 для одновимірного масиву.

*int \* Vector – одновимірний масив з елементами, які потрібно відсортувати*

*int L, int R – границі*

clock\_t Insert3(int P, int M, int N, int \*\*\* Arr3D) - Функція у якій реалізовано алгоритм сортування №3 для тривимірного масиву.

*int P, int M, int N – Розмірність тривимірного масиву*

*int \*\*\* Arr3D – Тривимірний масив*

clock\_t Select6(int P, int M, int N, int \*\*\* Arr3D) - Функція у якій реалізовано алгоритм сортування №10 для тривимірного масиву.

*int P, int M, int N – Розмірність тривимірного масиву*

*int \*\*\* Arr3D – Тривимірний масив*

void QuickSort(int M, int N, int \*\*\* Arr3D, int \* SliceSum, int L, int R) - Функція у якій реалізовано алгоритм сортування №24 для тривимірного масиву.

*int M, int N – Розмірність перерізу масиву*

*int \*\*\* Arr3D – Тривимірний масив*

*int L, int R – границі*

clock\_t \* QuickSortMeasurement\_Arr1D(int P, clock\_t \* Res, int type, int add, int \* Vector) – Функція яка вимірю час виконання сортування алгоритмом №24 одновимірного масиву

*int \* Vector – одновимірний масив з елементами, які потрібно відсортувати*

*int P – довжина масиву*

*clock\_t \* Res – масив часових вимірів для заповнення і повернення*

*int type – цілочисельна змінна яка визначає тип заповнення масиву перед кожною наступною ітерацією для повторного виміру - повернення масиву до свого початкового виду.*

*int add – цілочисельна змінна для додаткового виділення пам’яті і заповнення масиву одним елементом більше. Для QuickSort не використовується, лише для алгоритму №3 (Insertion Sort)*

clock\_t \* QuickSortMeasurement(int P, int M, int N, int \*\*\* Arr3D, clock\_t \* Res, int type) – Функція яка вимірює час виконання сортування алгоритмом №24 тривимірного масиву

*int \*\*\* Arr3D – Тривимірний масив, що сортується*

*int P, int M, int N – Розмірність тривимірного масиву*

*clock\_t \* Res – масив часових вимірів для заповнення і повернення*

*int type – цілочисельна змінна яка визначає тип заповнення масиву перед кожною наступною ітерацією для повторного виміру - повернення масиву до свого початкового виду.*

*int add – цілочисельна змінна для додаткового виділення пам’яті і заповнення масиву одним елементом більше. Для QuickSort не використовується, лише для алгоритму №3 (Insertion Sort)*

float MeasurementProcessing(clock\_t \* Res) – Функція для обробки результатів вимірів. Відкидає перші rejected\_number та по min\_max\_number екстремальних вимірів.

*clock\_t \* Res – масив часових вимірів, що обробляється*

int \*\*\* define\_matrix(int P, int M, int N, int add) – Функція що створює тривимірний масив і виділяє пам’ять для нього.

*int P, int M, int N – Розмірність тривимірного масиву*

*int add – цілочисельна змінна для додаткового виділення пам’яті і заповнення масиву одним елементом більше. Використовується лише для алгоритму №3 (Insertion Sort)*

void free\_memory(int P, int M, int N, int \*\*\* Arr3D, int add) - Функція що приймає тривимірний масив і звільнює пам’ять, виділену під нього раніше.

*int P, int M, int N – Розмірність тривимірного масиву*

*int \*\*\* Arr3D – Тривимірний масив, що буде видалено*

*int add – цілочисельна змінна для додаткового виділення / звільнення пам’яті і заповнення масиву одним елементом більше. Використовується лише для алгоритму №3 (Insertion Sort)*

void fill\_matrix(int P, int M, int N, int \*\*\* Arr3D, int type, int add) – Функція для заповнення елементами створеного раніше тривимірного масиву.

*int P, int M, int N – Розмірність тривимірного масиву*

*int \*\*\* Arr3D – Тривимірний масив, що заповнюється*

*int type – цілочисельна змінна яка визначає тип заповнення масиву*

*int add – цілочисельна змінна для додаткового виділення пам’яті і заповнення масиву одним елементом більше. Використовується лише для алгоритму №3 (Insertion Sort)*

int \* CreateVector(int P, int add) – Функція яка створює одновимірний масив і виділяє під нього пам’ять.

*int P – довжина масиву*

*int add – цілочисельна змінна для додаткового виділення пам’яті, лише для алгоритму №3 (Insertion Sort)*

void FillVector(int P, int \* Vector, int type, int add) - Функція для заповнення елементами створеного раніше одновимірного масиву.

*int \* Vector – одновимірний масив з елементами, які потрібно відсортувати*

*int P – довжина масиву*

*int type – цілочисельна змінна яка визначає тип заповнення масиву*

*int add – цілочисельна змінна для заповнення на add елементів більше, лише для алгоритму №3 (Insertion Sort)*

**Текст програми з коментарями**

**main.c**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <malloc.h>

#include <stdlib.h>

#include "common.h"

#include "processing.h"

#include "algorithms.h"

// Функція яка формує вибір варіанту виконання програми на основі введених юзером даних

int \* menu(int data[4]){

    int choice1, choice2; // Числа що відповідають за формування варіанту виконання програми на основі їх значень

    int counter = 1; // Лічильник для номерів всім можливим комбінаціям (алгоритм - сортування) вибору користувача

    int choice\_matrix[4][4]; // Порядкові номера всіх можливих варінтів вибору складають матрицю (кількість алгоритмів) \* (кількість типів сортування)

    for (int i = 0; i < 4; i++){

        for (int j = 0; j < 4; j++){

            choice\_matrix[i][j] = counter;

            counter++;

        }

    }

    printf("Алгоритми:\n1. Алгоритм №3\n2. Алгоритм №10\n3. Алгоритм №24\n4. Запустити всі алгоритми\n5. Вийти з програми\n\nОберіть пункт що відповідає бажаному номеру алгоритму: ");

    scanf("%d", &choice1);

    if (choice1 < 1 || choice1 > 4){

        printf("\nНевірно введені дані.\n"); exit(0);

    }

    printf("\nТип сортування:\n1. Впорядкований\n2. Невпорядкований\n3. Обернено впорядкований\n4. Всі типи сортування\n\nОберіть тип сортування: ");

    scanf("%d", &choice2);

    if (choice2 < 1 || choice2 > 4){

        printf("\nНевірно введені дані.\n"); exit(0);

    }

    data[0] = choice\_matrix[choice1 - 1][choice2 - 1];

    printf("\nОберіть предмет роботи програми:\n1. Одновимірний масив\n2. Тривимірний масив\n\n");

    printf("Опція: ");

    scanf("%d", &choice1);

    if (choice1 == 1){

        printf("\nВведіть значення довжини вектора (P): ");

        scanf("%d", &data[1]);

        data[2] = 0;

        data[3] = 0;

        return data;

    }

    printf("\nОберіть розмірність матриці (P, M, N): \n");

    printf("Випадок дослідження 1. Форма перерізу - квадрат.\n\n");

    printf("1. 15000 x 1 x 1\n2. 15000 x 2 x 2\n3. 15000 x 4 x 4\n4. 15000 x 8 x 8\n5. 15000 x 16 x 16\n\n");

    printf("Випадок дослідження 2. Залежність часу роботи алгоритмів від форми перерізів масива\n\n");

    printf("6. 4000 x 2 x 800\n7. 4000 x 4 x 400\n8. 4000 x 8 x 200\n9. 4000 x 16 x 100\n10. 4000 x 100 x 16\n11. 4000 x 200 x 8\n12. 4000 x 400 x 4\n13. 4000 x 800 x 2\n14. Задати розмірність власноруч\n\nОпція: ");

    scanf("%d", &choice1);

    switch (choice1)

    {

    case 1:

        data[1] = 15000;

        data[2] = 1;

        data[3] = 1;

        break;

    case 2:

        data[1] = 15000;

        data[2] = 2;

        data[3] = 2;

        break;

    case 3:

        data[1] = 15000;

        data[2] = 4;

        data[3] = 4;

        break;

    case 4:

        data[1] = 15000;

        data[2] = 8;

        data[3] = 8;

        break;

    case 5:

        data[1] = 15000;

        data[2] = 16;

        data[3] = 16;

        break;

    case 6:

        data[1] = 4000;

        data[2] = 2;

        data[3] = 800;

        break;

    case 7:

        data[1] = 4000;

        data[2] = 4;

        data[3] = 400;

        break;

    case 8:

        data[1] = 4000;

        data[2] = 8;

        data[3] = 200;

        break;

    case 9:

        data[1] = 4000;

        data[2] = 16;

        data[3] = 100;

        break;

    case 10:

        data[1] = 4000;

        data[2] = 100;

        data[3] = 16;

        break;

    case 11:

        data[1] = 4000;

        data[2] = 200;

        data[3] = 8;

        break;

    case 12:

        data[1] = 4000;

        data[2] = 400;

        data[3] = 4;

        break;

    case 13:

        data[1] = 4000;

        data[2] = 800;

        data[3] = 2;

        break;

    case 14:

        printf("\nВведіть значення розмірності матриці (P, M, N): ");

        scanf("%d, %d, %d", &data[1], &data[2], &data[3]);

        break;

    default:

        break;

    }

    return data;

}

void OutTable(float \* measurements){ // Функція для виводу усереднених результатів у табличному виді

    printf("Алгоритм\tВпорядкований\tНевпорядкований\tОбернено впорядкований\n");

    printf("Алгоритм 1\t%f\t%f\t%f\n", measurements[0], measurements[1], measurements[2]);

    printf("Алгоритм 2\t%f\t%f\t%f\n", measurements[3], measurements[4], measurements[5]);

    printf("Алгоритм 3\t%f\t%f\t%f\n\n", measurements[6], measurements[7], measurements[8]);

}

int main(){

    // Таск

    // 5. Впорядкувати тривимірний масив Аrr3D [P,M,N] таким чи-

    // ном: переставити перерізи масива за незменшенням сум їх

    // елементів.

    SetConsoleOutputCP(1251);

    SetConsoleCP(1251);

    srand(time(0));

    float measurements[9]; // Об'ява масиву усереднених результатів для виводу

    int data[4]; // Дані вибору користувачем

    while (TRUE){ // Цикл для перезапуску програми

        for (int i = 0; i < 9; i++){ // Установлення базового значення для вимірів для випадків часткового будування таблиці | Скидання значень до базових після виконання програми перед наступним виконанням

            measurements[i] = -1.0;

        }

        menu(data); // Вибір варіанту виконання програми 1-16 всі можливі випадки

        int P = data[1];

        int M = data[2];

        int N = data[3];

        int \*\*\* Arr3D;

        int \* Vector;

        clock\_t Res[MEASUREMENTS];

        switch (data[0]) // Варіанти виконання програми в залженості від обраних опцій у меню, де кожен case відповідає номеру вибору [1 - 16]

        {

        case 1: // Алгоритм 1 Тип сортування 1

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[0] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

            }

            else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 1, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[0] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 2: // Алгоритм 1 Тип сортування 2

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[1] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

            } else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 2, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[1] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 3: // Алгоритм 1 Тип сортування 3

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[2] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

            } else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 3, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[2] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 4: // Алгоритм 1 Всі типи сортування

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[0] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[1] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[2] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

            } else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 1, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[0] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 2, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[1] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 3, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[2] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 5: // Алгоритм 2 Тип сортування 1

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[3] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 1, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[3] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 6: // Алгоритм 2 Тип сортування 2

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[4] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 2, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[4] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 7: // Алгоритм 2 Тип сортування 3

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[5] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 3, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[5] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 8: // Алгоритм 2 Всі типи сортування

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[3] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[4] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[5] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 1, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[3] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 2, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[4] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 3, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[5] = MeasurementProcessing(Res);

                free(Vector);

            }

            break;

        case 9: // Алгоритм 3 Тип сортування 1

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 1)

                );

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

                measurements[6] =  MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 1, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 10: // Алгоритм 3 Тип сортування 2

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 2)

                );

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 2, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 11: // Алгоритм 3 Тип сортування 3

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 3)

                );

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 0);

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 3, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 12: // Алгоритм 3 Всі типи сортування

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 0);

                fill\_matrix(P, M, N, Arr3D, 1, 0);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 1)

                );

                fill\_matrix(P, M, N, Arr3D, 2, 0);

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 2)

                );

                fill\_matrix(P, M, N, Arr3D, 3, 0);

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 3)

                );

            } else {

                Vector = CreateVector(P, 0);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 1, 0, Vector)

                );

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 2, 0, Vector)

                );

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 3, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 13: // Всі алгоритми, Тип сортування 1

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[0] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[3] = MeasurementProcessing(Res);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 1)

                );

                free\_memory(P, M, N, Arr3D, 0);

            }

            else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 1, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[0] = MeasurementProcessing(Res);

                free(Vector);

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 1, 0);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[3] = MeasurementProcessing(Res);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 1, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 14: // Всі алгоритми, Тип сортування 2

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[1] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[4] = MeasurementProcessing(Res);

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 2)

                );

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 2, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[1] = MeasurementProcessing(Res);

                free(Vector);

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 2, 0);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[4] = MeasurementProcessing(Res);

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 2, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 15: // Всі алгоритми, Тип сортування 3

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 1);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[2] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[5] = MeasurementProcessing(Res);

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 3)

                );

                free\_memory(P, M, N, Arr3D, 0);

            } else {

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 3, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[2] = MeasurementProcessing(Res);

                free(Vector);

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 3, 0);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[5] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 3, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 3, 0, Vector)

                );

                free(Vector);

            }

            break;

        case 16: // Всі алгоритми, Всі типи сортування

            if (M && N){

                Arr3D = define\_matrix(P, M, N, 1);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 0);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[0] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 0);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[1] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 0);

                    Res[i] = Insert3(P, M, N, Arr3D);

                }

                measurements[2] = MeasurementProcessing(Res);

                free\_memory(P, M, N, Arr3D, 1);

                Arr3D = define\_matrix(P, M, N, 0);

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

                    fill\_matrix(P, M, N, Arr3D, 1, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[3] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 2, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[4] = MeasurementProcessing(Res);

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

                    fill\_matrix(P, M, N, Arr3D, 3, 0);

                    Res[i] = Select6(P, M, N, Arr3D);

                }

                measurements[5] = MeasurementProcessing(Res);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 1)

                );

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 2)

                );

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement(P, M, N, Arr3D, Res, 3)

                );

                free\_memory(P, M, N, Arr3D, 0);

            }

            else{

                Vector = CreateVector(P, 1);

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

                    FillVector(P, Vector, 1, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[0] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 2, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[1] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 3, 1);

                    Res[i] = Insert3\_Arr1D(Vector, P);

                }

                measurements[2] = MeasurementProcessing(Res);

                free(Vector);

                Vector = CreateVector(P, 0);

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

                    FillVector(P, Vector, 1, 0);;

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[3] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 2, 0);;

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[4] = MeasurementProcessing(Res);

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

                    FillVector(P, Vector, 3, 0);

                    Res[i] = Select6\_Arr1D(Vector, P);

                }

                measurements[5] = MeasurementProcessing(Res);

                measurements[6] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 1, 0, Vector)

                );

                measurements[7] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 2, 0, Vector)

                );

                measurements[8] = MeasurementProcessing(

                QuickSortMeasurement\_Arr1D(P, Res, 3, 0, Vector)

                );

                free(Vector);

            }

            break;

        default:

            break;

        }

        OutTable(measurements); // Вивід таблиці з даними

    }

    char c = getchar();

    return 0;

}

**algorithms.c**

#include <stdio.h>

#include <time.h>

#include <malloc.h>

#include <stdlib.h>

#include "algorithms.h"

#include "common.h"

clock\_t Insert3\_Arr1D(int \*Vector, int P)

{

    int j;

    clock\_t time\_start, time\_stop;

    time\_start = clock();

    for(int i = 2; i < P + 1; i++){

        Vector[0] = Vector[i];

        j = i;

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

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

            j = j - 1;

            }

            Vector[j] = Vector[0];

    }

    time\_stop = clock();

    return time\_stop - time\_start;

}

// Àëãîðèòì ¹10 äëÿ ñîðòóâàííÿ âåêòîðà

clock\_t Select6\_Arr1D(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;

        if (imin!=s) {

        tmp=A[imin];

        A[imin]=A[s];

        A[s]=tmp;

        }

    }

    time\_stop = clock();

    return time\_stop - time\_start;

}

// Àëãîðèòì ñîðòóâàííÿ âñòàâêàìè ¹3 ç ïîøóêîì ñïðàâà â³ä åëåìåíòó ùî âñòàâëÿºòüñÿ àáî ç áàð'ºðîì çì³íåíèé äëÿ ïàðàëåëüíîãî ïåðåì³ùåííÿ ïåðåð³ç³â

clock\_t Insert3(int P, int M, int N, int \*\*\* Arr3D){

    clock\_t time\_start, time\_stop;

    // Ìàñèâ äëÿ ñóì ïåðåð³ç³â

    int \* SliceSum = (int \*) malloc(sizeof(int) \* (P + 1));

    // int SliceSum[P + 1];

    int KeySlice[M][N];

    time\_start = clock();

    // Çàïîâíåííÿ ìàñèâó ñóìàìè ïåðåð³ç³â

    for (int i = 1; i < P + 1; i++) {

        for (int j = 0; j < M; j++) {

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

                SliceSum[i] += Arr3D[i][j][k];

            }

        }

    }

    // Ñîðòóâàííÿ

    for (int i = 1; i < P; i++) {

        int key = SliceSum[i];

        for (int m = 0; m < M; m++){

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

                KeySlice[m][n] = Arr3D[i][m][n];

            }

        }

        int j = i - 1;

        while (j >= 0 && SliceSum[j] > key) {

            SliceSum[j + 1] = SliceSum[j];

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

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

                    Arr3D[j + 1][m][n] = Arr3D[j][m][n];

                }

            }

            j--;

        }

        SliceSum[j + 1] = key;

        for (int m = 0; m < M; m++){

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

                Arr3D[j + 1][m][n] = KeySlice[m][n];

            }

        }

    }

    time\_stop = clock();

    free(SliceSum);

    return time\_stop - time\_start;

}

clock\_t Select6(int P, int M, int N, int \*\*\* Arr3D){ // Àëãîðèòì ñîðòóâàííÿ ¹10 ìàñèâó ñóì ç ïàðàëåëüíèìè çì³íàìè ïîðÿäêó ðîçð³ç³â ãîëîâíî¿ ìàòðèö³

    int imin, tmp;

    int temp\_slice[M][N]; // ìàòðèöÿ äëÿ òèì÷àñîâîãî çáåð³ãàííÿ ïåðåð³çó

    // int SliceSum[P];

    int \* SliceSum = (int \*) malloc(sizeof(int) \* P); // Îá'ÿâà ìàñèâó ñóì, âèä³ëåííÿ ïàì'ÿò³

    clock\_t time\_start, time\_stop;

    time\_start = clock();

    // Ôîðìóâàííÿ ñóì

    for (int i = 0; i < P; i++) {

        for (int j = 0; j < M; j++) {

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

                SliceSum[i] += Arr3D[i][j][k];

            }

        }

    }

    // Ñîðòóâàííÿ

    for(int s = 0; s < P - 1; s++){

        imin = s;

        for(int i = s + 1; i < P; i++)

            if (SliceSum[i] < SliceSum[imin]) imin = i;

            if (imin != s) {

                tmp = SliceSum[imin];

                SliceSum[imin] = SliceSum[s];

                SliceSum[s] = tmp;

                for (int s1 = 0; s1 < M; s1++){

                    for (int s2 = 0; s2 < N; s2++){

                        temp\_slice[s1][s2] = Arr3D[imin][s1][s2];

                        }

                    }

                for (int s1 = 0; s1 < M; s1++){

                    for (int s2 = 0; s2 < N; s2++){

                        Arr3D[imin][s1][s2] = Arr3D[s][s1][s2];

                    }

                }

                for (int s1 = 0; s1 < M; s1++){

                    for (int s2 = 0; s2 < N; s2++){

                        Arr3D[s][s1][s2] = temp\_slice[s1][s2];

                    }

                }

            }

    }

    time\_stop = clock();

    free(SliceSum);

    return time\_stop - time\_start;

}

// Àëãîðèòì ¹24 äëÿ ñîðòóâàííÿ âåêòîðà

void QuickSort\_Arr1D(int L, int R, int \* Vector)

{

    int B, tmp, i, j;

    B=Vector[(L+R)/2];

    i=L; j=R;

    while (i<=j) {

        while (Vector[i] < B) i=i+1;

        while (Vector[j] > B) j=j-1;

        if (i<=j) {

            tmp=Vector[i];

            Vector[i]=Vector[j];

            Vector[j]=tmp;

            i=i+1;

            j=j-1;

        }

    }

    if (L<j) QuickSort\_Arr1D(L,j, Vector);

    if (i<R) QuickSort\_Arr1D(i,R, Vector);

}

void QuickSort(int M, int N, int \*\*\* Arr3D, int \* SliceSum, int L, int R){ // Àëãîðèòì ñîðòóâàííÿ ¹24 ìàñèâó ñóì ç ïàðàëåëüíèìè çì³íàìè ïîðÿäêó ðîçð³ç³â ãîëîâíî¿ ìàòðèö³

    int B, tmp, i, j; // Çì³íí³ äëÿ ñîðòóâàííÿ àëãîðèòìîì QuickSort

    B = SliceSum[(L + R) / 2]; // Öåíòðàëüíèé îïîðíèé åëåìåíò ïîä³ëó

    i = L; j = R;

    int temp\_slice[M][N];

    while (i <= j) {

        while (SliceSum[i] < B) i = i + 1;

        while (SliceSum[j] > B) j = j - 1;

        if (i <= j) {

            tmp = SliceSum[i];

            SliceSum[i] = SliceSum[j];

            SliceSum[j] = tmp;

            for (int s1 = 0; s1 < M; s1++){

                for (int s2 = 0; s2 < N; s2++){

                    temp\_slice[s1][s2] = Arr3D[i][s1][s2];

                    }

                }

            for (int s1 = 0; s1 < M; s1++){

                for (int s2 = 0; s2 < N; s2++){

                    Arr3D[i][s1][s2] = Arr3D[j][s1][s2];

                }

            }

            for (int s1 = 0; s1 < M; s1++){

                for (int s2 = 0; s2 < N; s2++){

                    Arr3D[j][s1][s2] = temp\_slice[s1][s2];

                }

            }

            i = i + 1;

            j = j - 1;

        }

    }

    if (L < j) QuickSort(M, N, Arr3D, SliceSum, L, j); // Âèêëèêè äëÿ ñîðòóâàííÿ ìåíøèõ ÷àñòèí

    if (i < R) QuickSort(M, N, Arr3D, SliceSum, i, R);

}

**algorithms.c**

#include <stdio.h>

#include <time.h>

#include <malloc.h>

#include <stdlib.h>

#include "processing.h"

#include "common.h"

#include "algorithms.h"

clock\_t \* QuickSortMeasurement\_Arr1D(int P, clock\_t \* Res, int type, int add, int \* Vector)

{

    clock\_t time\_start, time\_stop;

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

    {

        FillVector(P, Vector, type, add);

        time\_start = clock();

        QuickSort\_Arr1D(0, P - 1, Vector);

        time\_stop = clock();

        Res[i] = time\_stop - time\_start;

    }

    return Res;

}

clock\_t \* QuickSortMeasurement(int P, int M, int N, int \*\*\* Arr3D, clock\_t \* Res, int type){ // Ôóíêö³ÿ äëÿ âèì³ðó ÷àñó âèòðà÷åíîãî íà ñîðòóâàííÿ àëãîðèòìîì ¹24 QuickSort

    int L = 0;

    int R = P - 1;

    // int \* SliceSum = (int \*) malloc(sizeof(int) \* P);

    int SliceSum[P];

    clock\_t time\_start, time\_stop;

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

        fill\_matrix(P, M, N, Arr3D, type, 0);

        time\_start = clock();

        for (int i = 0; i < P; i++) {

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

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

                    SliceSum[i] += Arr3D[i][j][k];

                }

            }

        }

        QuickSort(M, N, Arr3D, SliceSum, L, R);

        time\_stop = clock();

        Res[i] = time\_stop - time\_start;

    }

    // free(SliceSum);

    return Res;

}

float MeasurementProcessing(clock\_t \* Res){ // Óñåðåäíåííÿ ðåçóëüòàò³â âèì³ðþâàííÿ â³äêèäàííÿì ïåðøèõ òà åêñòðåìàëüíèõ çíà÷åíü

    long int Sum;

    float AverageValue;

    clock\_t buf;

    int L = rejected\_number, R = MEASUREMENTS - 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 - min\_max\_number; i++)

        Sum = Sum + Res[i];

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

    return AverageValue;

}

// Ôóíêö³ÿ ùî âèä³ëÿº ïàìÿòü äëÿ òðèâèì³ðíîãî ìàñèâó

// int add - çñóâ äëÿ ñîðòóâàííÿ Insert3

int \*\*\* define\_matrix(int P, int M, int N, int add){ // Ôóíêö³ÿ äëÿ âèä³ëåííÿ ïàì'ÿò³ ï³ä òðèâèì³ðíèé ìàñèâ

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

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

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

        for (int i = 0; i < M; i++)

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

    }

    return Arr3D;

}

// Ôóíêö³ÿ ùî çâ³ëüíÿº ïàìÿòü âèä³ëåíó ðàí³øå äëÿ òðèâèì³ðíîãî ìàñèâó

// int add - çñóâ äëÿ ñîðòóâàííÿ Insert3

void free\_memory(int P, int M, int N, int \*\*\* Arr3D, int add){ // Ôóíêö³ÿ äëÿ çâ³ëåííÿ ïàì'ÿò³ âèä³ëåíî¿ ï³ä òðèâèì³ðíèé ìàñèâ

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

        for (int i = 0; i < M; i++)

            free(Arr3D[k][i]);

        free(Arr3D[k]);

    }

    free(Arr3D);

}

// Ôóíêö³ÿ ÿêà çàïîâíþº ìàòðèöþ ÷èñëàìè.

// int type - âèçíà÷àº âïîðÿäêîâàí³ñòü

// int add - çñóâ äëÿ ñîðòóâàííÿ Insert3

void fill\_matrix(int P, int M, int N, int \*\*\* Arr3D, int type, int add){ // Ôóíêö³ÿ äëÿ çàïîâíåííÿ ìàòðèö³ â çàëåæíîñò³ â³ä îáðàíîãî òèïó (âïîðÿäêîâàíà, íåâïîðÿäêîâàíà, îáåðíåíî âïîðÿäêîâàíà)

    int counter;

    if (type == 1){

        counter = 1 - add;

        for (int p = 0; p < P + add; p++){

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

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

                    Arr3D[p][m][n] = counter;

                    counter++;

                }

            }

        }

    }

    else if (type == 2){

        int upper = 100;

        int lower = 0;

        for (int p = 0; p < P + add; p++){

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

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

                    Arr3D[p][m][n] = rand() % (P\*M\*N);

                }

            }

        }

    }

    else{

        counter = P \* M \* N + add;

        for (int p = 0; p < P; p++){

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

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

                    Arr3D[p][m][n] = counter;

                    counter--;

                }

            }

        }

    }

}

// Ôóíêö³ÿ ùî ñòâîðþº ³ ïîâåðòàº îäíîâèì³ðíèé ìàñèâ

int \* CreateVector(int P, int add){

    int \* Vector = (int \*) malloc(sizeof(int) \* (P + add));

    return Vector;

}

// Ôóíêö³ÿ äëÿ çàïîâíåííÿ îäíîâèì³ðíîãî ìàñèâà ÷èñëàìè

void FillVector(int P, int \* Vector, int type, int add){

    // int Vector[P+add];

    // Çàïîâíåííÿ

    switch (type)

    {

    case 1:

        for (int p = 0; p < P + add; p++){

            Vector[p] = p - 1;

        }

        break;

    case 2:

        for (int p = 0; p < P + add; p++){

            Vector[p] == rand() % P;

        }

        break;

    case 3:

        for (int p = (P + add); p > 0; p--){

            Vector[p] = p;

        }

        break;

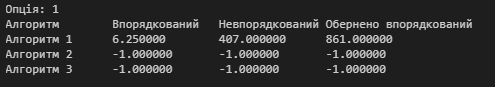
    default:

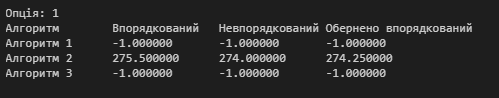
        break;

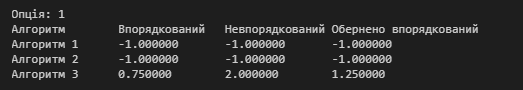
    }

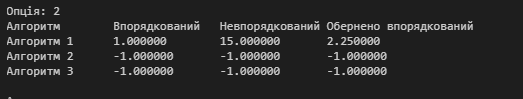
}

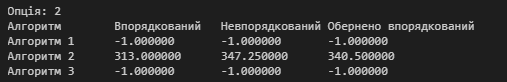
**Тести, що демонструють коректну роботу** (-1 в полях значень які не обраховувались, тобто користувач обрав певні алгоритми чи типи сортування)

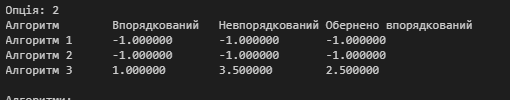


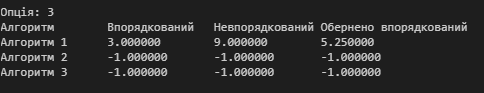


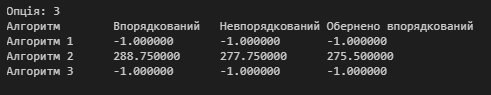


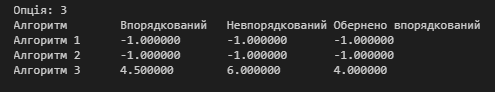


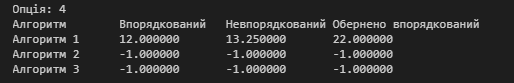


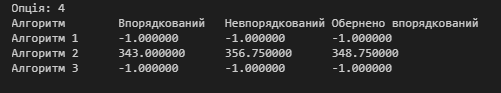


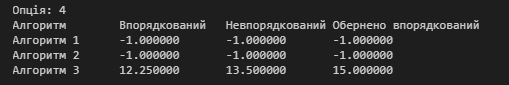


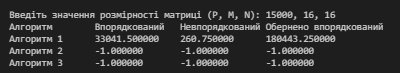


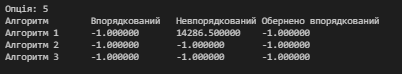


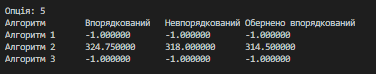


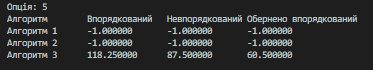


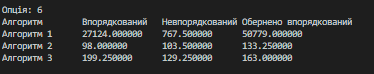


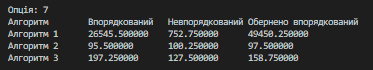


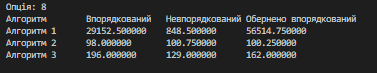


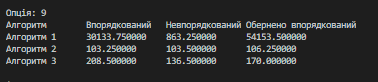


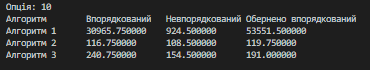


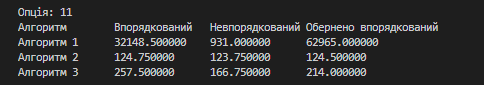


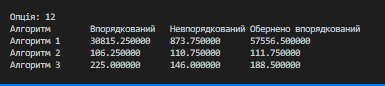


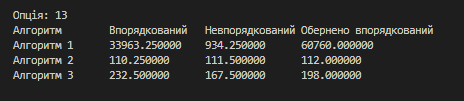












**Результати** (У clocks-ах)

1. P x M x N = 15000 x 1 x 1

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 6.25 | 407 | 861 |
| Алгоритм №10 | 275 | 274 | 274.25 |
| Алгоритм №24 | 0.75 | 2 | 1.25 |

1. P x M x N = 15000 x 2 x 2

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 1 | 15 | 2.25 |
| Алгоритм №10 | 313 | 347.25 | 340.5 |
| Алгоритм №24 | 1 | 3.5 | 2.5 |

1. P x M x N = 15000 x 4 x 4

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 3 | 9 | 5.25 |
| Алгоритм №10 | 288.75 | 277.75 | 275.5 |
| Алгоритм №24 | 4.5 | 6 | 4 |

1. P x M x N = 15000 x 8 x 8

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 12 | 13.25 | 22 |
| Алгоритм №10 | 343 | 356.75 | 348.75 |
| Алгоритм №24 | 12.25 | 13.5 | 15 |

1. P x M x N = 15000 x 16 x 16

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 33041.5 | 14286.5 | 180443.25 |
| Алгоритм №10 | 324.75 | 318 | 314.5 |
| Алгоритм №24 | 118.25 | 87.5 | 60.5 |

1. P x M x N = 4000 x 2 x 800

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 27124 | 6054 | 50779 |
| Алгоритм №10 | 98 | 103.5 | 133.25 |
| Алгоритм №24 | 199.25 | 129.25 | 163 |

1. P x M x N = 4000 x 4 x 400

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 26545.5 | 5768.5 | 49540.25 |
| Алгоритм №10 | 95.5 | 100.25 | 97.5 |
| Алгоритм №24 | 197.25 | 127.5 | 158.75 |

1. P x M x N = 4000 x 8 x 200

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 29152.5 | 5883.25 | 56514.75 |
| Алгоритм №10 | 98 | 100.75 | 100.25 |
| Алгоритм №24 | 196 | 129 | 162 |

1. P x M x N = 4000 x 16 x 100

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 30133.75 | 5223 | 54153.5 |
| Алгоритм №10 | 103.25 | 103.5 | 106.25 |
| Алгоритм №24 | 208.5 | 136.5 | 170 |

1. P x M x N = 4000 x 100 x 16

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 30965.75 | 5001.5 | 53551.5 |
| Алгоритм №10 | 116.75 | 108.5 | 119.75 |
| Алгоритм №24 | 240.75 | 154.5 | 191 |

1. P x M x N = 4000 x 200 x 8

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 32148.5 | 5996.5 | 62965 |
| Алгоритм №10 | 124.75 | 123.75 | 124.5 |
| Алгоритм №24 | 257.5 | 166.75 | 214 |

1. P x M x N = 4000 x 400 x 4

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 30815.25 | 5153.25 | 57556.5 |
| Алгоритм №10 | 106.25 | 110.75 | 111.75 |
| Алгоритм №24 | 225 | 146 | 188.5 |

1. P x M x N = 4000 x 800 x 2

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Впорядкований | Невпорядкований | Обернено впорядкований |
| Алгоритм №3 | 33963.25 | 5803.25 | 60760 |
| Алгоритм №10 | 110.25 | 111.5 | 112 |
| Алгоритм №24 | 232.5 | 167.5 | 198 |

Довжина в 15000 чи 4000 елементів для одновимірного масиву занадто мала, алгоритми №3 і №24 відправьовують миттєво, тому підберемо зручні значення окремо для кожного алгоритму.

Алгоритм №3

|  |  |  |  |
| --- | --- | --- | --- |
| Вектор (100.000) | 0.75 | 1 | 1.25 |
| (100.000 х 2 х 2) | 4 | 61.5 | 12.5 |
| (100.000 х 4 х 4) | 18.5 | 80.5 | 42.25 |

Алгоритм №10

|  |  |  |  |
| --- | --- | --- | --- |
| Вектор (10.000) | 118 | 116 | 116 |
| Тривимірний масив (10.000 х 2 х 2) | 112 | 114 | 111 |
| Тривимірний масив (10.000 х 4 х 4) | 112.5 | 117.75 | 115.75 |

Алгоритм №24

|  |  |  |  |
| --- | --- | --- | --- |
| Вектор (100.000) | 3.25 | 3.25 | 4 |
| (100.000 х 2 х 2) | 10 | 19.25 | 13 |
| (100.000 х 4 х 4) | 23.75 | 37.75 | 30 |
| (100.000 х 8 х 8) | 199 | 98.25 | 117 |

Отже, можливо в деяких випадках час, що пішов на сортування одновимірного масиву помножений на кількість елементів у перерізі тривимірного масиву буде схожий на час, що пішов на сортування самого тривимірного масиву, проте не для всіх типів сортування та не для всіх використаних алгоритмів, що видно з таблиць, що б перевірити, яка буде реальна різниця, потрібно брати більші довжини для одновимірних масивіві, проте тоді буде нереально дочекатись результатів програми для тривимірних масивів, адже наявність перерізів дуже сповільнює роботу програми, наприклад в той час як одновимірний масив довжиною в 10.000.000 обробляється за секунд 20 програмою, результатів для такої ж кількості будь яких за розмірністю перерізів (>1) не дочекатись. Тому результати що наявні зараз дуже поверхневі і на їх основі важко зробити якісь адекватні висновки. Але очевидно, що сортування будь якого тривимірного масиву завжди займатиме більше часу, адже програма здійснює багато додаткових дій: переходи між розмірностями, переміщення елементів перерізів і т.д.

**Для випадків 1 – 5 побудуємо графіки залежностей часу від зростання кількості елементів у перерізі**

**(M x N), P – const для всіх випадків, тому його не враховуємо**

1. **Масив впорядкований**
2. **Масив невпорядкований**
3. **Масив обернено впорядкований**

**Порівняльний аналіз алгоритмів**

Метод прямої вставки з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар’єром. Цей метод має часову складність O(n^2) в найгіршому та середньому випадках, тому для великих масивів він може бути дуже повільним. Однак, для впорядкованих масивів з невеликою кількістю елементів, цей метод може бути швидшим, ніж інші методи. Хоча в загальному класичний метод вставки показує себе найкраще при роботі з впорядкованим масивом, гірше при роботі з невпорядкованим та найгірше при роботі з обернено впорядкованим, ця версія методу вставки, з пошуком справа від елемента що вставляється, з бар’єром показала себе трохи інакше, при значному збільшенні розмірів масиву найкращим випадком для неї був невпорядкований масив, обернено впорядкований залишився найгіршим випадком, в той час як при відносно невеликих масивах

Метод прямого вибору має також часову складність O(n^2) в найгіршому та середньому випадках. Цей метод не потребує додаткової пам'яті, як в методі прямої вставки, але має схожі проблеми з ефективністю на великих масивах.

Алгоритм методу «швидкого сортування» (сортування Хоара) має часову складність O(n log n) в середньому та найкращому випадках, тому для великих масивів він може бути набагато швидшим, ніж попередні два методи. Проте, в найгіршому випадку (коли обраний елемент не є оптимальним для розбиття масиву) часова складність може становити O(n^2), що робить його повільним для певних видів масивів.

**Висновки по отриманих результатах**

Точність вимірювання часу залежить від багатьох факторів, таких як розмір масиву, обсяг пам'яті, швидкість процесора та багато інших. Тому результати вимірювання часу можуть відрізнятися від теоретичних показників. Однак, у загальному випадку, порівняльний аналіз показав, що метод "швидкого сортування" має кращу складність у середньому та найкращому випадку, а метод прямого вибору та метод прямої вставки мають значно гіршу часову складність.

Також можна зробити висновок, що для невеликих масивів метод прямої вставки може працювати швидше, а для масивів з великою кількістю елементів краще використовувати метод "швидкого сортування".

В деяких таблицях видно, що невпорядкований масив вставкою сортується швидше, ніж впорядкований або обернено впорядкований, що доволі незвично, проте метод прямої вставки має часову складність O(n^2), що означає, що час сортування збільшується пропорційно квадрату кількості елементів в масиві. Коли масив вже впорядкований, метод прямої вставки все одно проходиться по всіх елементах масиву, порівнюючи їх між собою, щоб знайти правильне місце для вставки. Оскільки вже впорядкований масив має відносно невелику кількість "обмінів" елементів місцями, то це може зайняти більше часу, ніж сортування невпорядкованого масиву, де більше елементів потрібно буде перемістити.

Загалом, метод прямої вставки - це алгоритм, який добре працює з малими масивами, але може бути неефективним з великими масивами, незалежно від того, чи вони вже впорядковані чи ні. Тому існують інші алгоритми сортування, які працюють краще з великими масивами, наприклад, швидке сортування.

Метод вибору в основному пройде через той же порядок порівнянь, незалежно від того, чи впорядкований масив чи ні. Якщо масив уже впорядкований, то все ж потрібно порівнювати кожен елемент з кожним, незалежно від того, чи менший він чи більший за попередній. Це може займати більше часу, ніж у випадку невпорядкованого масиву, де порівнювати елементи можна зупинитись, як тільки знайдено менший елемент. Таким чином, метод вибору може потребувати більше порівнянь у впорядкованому масиві, що може призвести до більшого часу сортування. Але в цілому метод вибору показує схожі результати у різних випадках.

**Список літератури**

<https://programm.top/uk/c-sharp/algorithm/array-sort/insertion-sort/>

<https://uk.wikipedia.org/wiki/Сортування_вибором>

<https://www.wikiwand.com/uk/Сортування_вибором>

<https://uk.wikipedia.org/wiki/Швидке_сортування>

<https://training.epam.ua/News/Items/483?lang=ua>