**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: Бітлян А.В.

Група: КB-91

Номер залікової книжки: КВ-9102

Допущений до захисту

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

2 семестр 2019/2020 навчального року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ   
УКРАЇНИ «КПІ ІМ. ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

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

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

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

***Дослідження ефективності методів сортування (Алгоритм сортування No5 методу прямого вибору,***

***Алгоритм сортування No8 методу прямого вибору,***

***Алгоритм методу «швидкогосортування» (сортування Хоара)) на багатовимірних масивах***

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

(підпис)

Бітлян Андрій Віталійович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020р.

**ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ**

**I.** Описати принцип та схему роботи кожного із досліджуваних методів сортування або пошуку для одновимірного масиву.

**II.** Скласти алгоритми сортування або пошуку в багатовимірному масиві заданими методами, згідно до варіанту, та написати відповідну програму на мові програмування.

Програма повинна задовольняти наступним вимогам:

1. Всі алгоритми повинні бути реалізовані в рамках ОДНІЄЇ програми з діалоговим інтерфейсом для вибору варіантів тестування та виміру часу кожного алгоритму.
2. Одним з варіантів запуску програми має бути режим запуску виміру часу всіх алгоритмів у пакетному режимі, тобто запуск всіх алгоритмів для всіх випадків і побудова результуючої таблиці за наведеним нижче зразком для масиву з заданими геометричними розмірами.
3. При реалізації програми повинні бути використані модулі (unit).
4. Програма повинна мати коментарі для всіх структур даних, процедур та функцій, а також до основних смислових фрагментів алгоритмів.

**III.** Виконати налагодження та тестування коректності роботи написаної програми.

**IV.** Провести практичні дослідження швидкодії складених алгоритмів.

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

Для виконання ґрунтовного аналізу алгоритмів потрібно виконати виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами, а також для стандартного випадку одномірного масиву. Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибирається самостійно студентом так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів. За отриманими результатами можуть бути також побудовані графіки для наочності подання інформації.

**VI.** Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами:

* 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.

**VII.** Зробити висновки за виконаним порівняльним аналізом.

**VIII.** Програму курсової роботи під час її захисту ОБОВЯЗКОВО мати при собі на електронному носії інформації.

**Варіант № 67**

**Задача 5**

Впорядкувати тривимірний масив Аrr3D [P,M,N] таким чином:

переставити перерізи масива за незменшенням сум їх елементів.

**Досліджувані методи та алгоритми**

1. Алгоритм сортування No5 методу прямого вибору
2. Алгоритм сортування No8 методу прямого вибору
3. Алгоритм методу «швидкого сортування» (сортування Хоара)

**Способи обходу**

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 сортування, переставляти відповідні перерізи кожен раз, коли треба переставляти ключі.

**Випадки дослідження**

* Елементи початкового масиву впорядковані відповідно до заданої ознаки.
* Елементи початкового масиву невпорядковані.
* Елементи початкового масиву впорядковані за протилежно заданою ознакою.

**Принципи роботи та схеми алгоритмів**

1) Алгоритм сортування No5 методу прямого вибору

*Загальні відомості:*

У цьому алгоритмі масив складається з двох частин: відсортована і невідсортована. На початку відсортована складається з одного елементу(який стоїть на 0 позиції), всі інші – невідсортовані.

Принцип роботи:

1. Шукаємо елемент менший, ніж обраний з відсортованої частини, тобто мінімальний елемент в невідсортованій частині.
2. Якщо знайдено, то міняємо його з тим, з яким порівнюємо, на першій ітерації – це 0 позиція, на кожній наступній номер позиції зростає на один.
3. Повторюємо 1 та 2 кроки до тих пір, поки не закінчаться елементи в невідсортованій частині.

*Схема:*

Червоний колір – елемент, з яким порівнюємо 0 1 2 3 4 5 6

Синій колір – знайдений елемент(найменший) **1 2 5 3 6 0 4**

Зелений фон – відсортована частина **0 2 5 3 6 1 4**

**0 1 5 3 6 2 4**

**0 1 2 3 6 5 4**

**0 1 2 3 6 5 4**

**0 1 2 3 4 5 6**

**0 1 2 3 4 5 6**

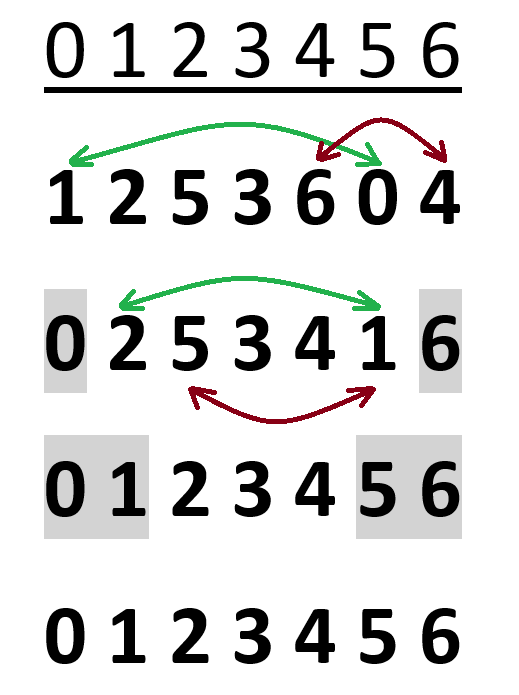
2) Алгоритм сортування No8 методу прямого вибору

*Загальні відомості:*

У цьому алгоритмі масив складається з двох частин: відсортована і невідсортована. На початку відсортована складається з одного елементу(який стоїть на 0 позиції), всі інші – невідсортовані.

Принцип роботи:

1. Шукаємо мінімальний і максимальний елемент в масиві.
2. Мінімальний міняємо з першим, максимальний – з останнім.
3. Тепер перший та останній на своїх місцях. Отже, звужуємо коло пошуку – повторюємо 1-2 кроки для масиву з другого до передостанього і так далі. Доки границі не перетнуться.

Примітка.

Стрілками зеленого кольору

вказано обмін мінімальних елементів.

А червоного кольору – максимальних.

Обмін мінімальних елементів

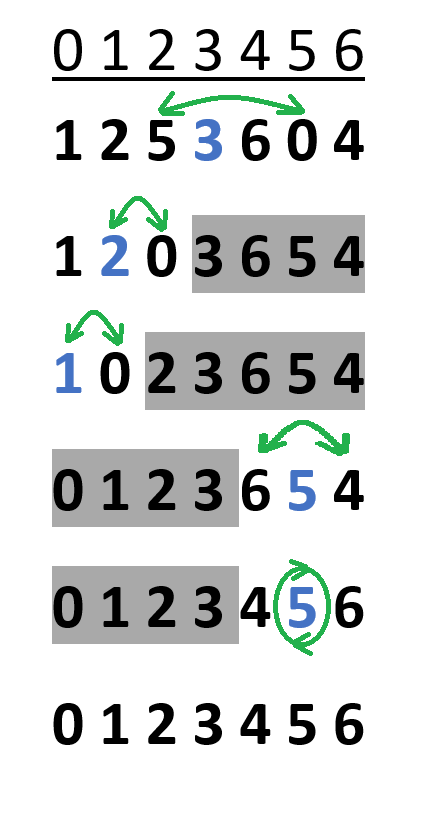
відбувається перед обміном максимальних.

3) ШВИДКЕ СОРТУВАННЯ (сортування Хоара)

Цей алгоритм був створенний професором Хоаром Ч.Е.Р.

*Принцип роботи:*

1. Обираємо довільнйи елемент з масиву, зазвичай, для зручності беруть центральний.
2. Шукаємо елемент більший від обраного(центрального) зліва направо.
3. Шукаємо елемент менший від обраного(центрального) справа наліво.
4. Якщо було знайдено потрібний елемент з обох боків, то вони міняються місцями.
5. Якщо індекси перетнулись, то це означає, що зліва від обраного елемента всі числа менше нього, а справа – більше. Цикл завершується.
6. Знову розбиваємо отримані частини, обираємо центральний елемент і т.д. Повторюємо кроки 1-5.
7. Коли всі частини стануть рівними одному елементу – масив відсортовано.

****

*Схема:*

Обраний елемент позначається синім кольором

На сірому фоні частина масиву, якак не задіяна у сортуванні на даному етапі

**Опис призначення процедур та функцій**

**Модуль main**

menu – меню діалогового інтерфейсу програми

All – запуск виміру часу всіх алгоритмів у пакетному режимі, тобто запуск всіх алгоритмів для всіх випадків і побудова результуючої таблиці

D3 – обчислення швидкодії алгоритму в тривимірному масиву, сортуючи переставляючи перерерізи за сумою

vector – обчислення швидкодії алгоритму сортування на одновимірному масиві

**Модуль fill**

fill\_sorted – заповнення тривимірного масиву числами строго за збільшенням

fill\_reversed – заповнення тривимірного масиву числами строго за зменшенням

fill\_random – заповнення тривимірного масиву випадковими числами

**Модуль sum**

sum – функція для пошуку суми перерізу тривимірного масиву

**Модуль sort**

Swap – функція для перестановки двох перерізів місцями

timeQuickSort - функція для пошуку часу роботи алгоритму швидкого сортування в тривимірному масиві

quickSort - алгоритм швидкого сортування заданої задачі

Select5 - алгоритм вибору номер 5 заданої задачі

Select8 - алгоритм вибору номер 8 заданої задачі

**Модуль vector**

Vtime\_quick\_sort - функція для пошуку часу роботи алгоритму швидкого сортування в одновимірному масиві

Vquick\_sort - алгоритм швидкого сортування в одновимірному масиві

VSelect5 - алгоритм вибору номер 5 в одновимірному масиві

VSelect8 - алгоритм вибору номер 8 в одновимірному масиві

**Текст програми**

**Main**

//підключення модулів

#include"print.h"

#include"fill.h"

#include"sort.h"

#include "sum.h"

#include"vector.h"

using namespace std;

void vector()

{

//оголошення масиву

int\* A;

int N;

cout << "Input size: ";

cin >> N;

A = new int[N];

//вибір того, як заповнювати масив

cout << "Choose the option(type the digit):" << endl;

cout << "1. Sorted" << endl;

cout << "2. Random" << endl;

cout << "3. Reversed" << endl;

int option;

cin >> option;

switch (option)

{

//впорядкований

case 1: for(int i = 0; i < N; i++)

{

A[i] = i;

}

break;

//невпорядкований

case 2: for (int i = 0; i < N; i++)

{

A[i] = rand() % N;

}

break;

//оберенено впорядкований

case 3: for (int i = 0; i < N; i++)

{

A[i] = N - i;

}

break;

default: cout << "Incorrect data!" << endl;

}

//вибір алгоритму сортування

cout << "Choose the option(type the digit):" << endl;

cout << "1. Quick sort" << endl;

cout << "2. Select 5" << endl;

cout << "3. Select 8" << endl;

cin >> option;

switch (option)

{

case 1: cout << "time = " << Vtime\_quick\_sort(N, A, 0, N - 1) << endl;

break;

case 2: cout << "time = " << VSelect5(A, N) << endl;

break;

case 3: cout << "time = " << VSelect8(A, N) << endl;

break;

default: cout << "Incorrect data!" << endl;

}

}

void D3()

{

//оголошення масиву

int P, M, N;

int\*\*\* Arr3D;

cout << "Input size:" << endl;

cout << "P: ";

cin >> P;

cout << "M: ";

cin >> M;

cout << "N: ";

cin >> N;

//виділення пам'яті для масиву

Arr3D = new int\*\* [P];

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

//вибір того, як заповнювати масив

cout << "Choose the option(type the digit):" << endl;

cout << "1. Sorted" << endl;

cout << "2. Random" << endl;

cout << "3. Reversed" << endl;

int option;

cin >> option;

switch (option)

{

//впорядкований

case 1: fill\_sorted(Arr3D, P, M, N);

break;

//невпорядкований

case 2: fill\_random(Arr3D, P, M, N);

break;

//оберенено впорядкований

case 3: fill\_reversed(Arr3D, P, M, N);

break;

default: cout << "Incorrect data!" << endl;

}

//вибір алгоритму сортування

cout << "Choose the option(type the digit):" << endl;

cout << "1. Quick sort" << endl;

cout << "2. Select 5" << endl;

cout << "3. Select 8" << endl;

cin >> option;

switch (option)

{

case 1: cout << "time = " << timeQuickSort(0, P-1, Arr3D, P, M, N) << endl;

break;

case 2: cout << "time = " << Select5(Arr3D, P, M, N) << endl;

break;

case 3: cout << "time = " << Select8(Arr3D, P, M, N) << endl;

break;

default: cout << "Incorrect data!" << endl;

}

//очищення пам'яті

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

delete[]Arr3D;

}

void All()

{

int P = 50000, M, N;

int\*\*\* Arr3D;

Arr3D = new int\*\* [P];

//масив, в який записуємо дані зі всіх випадків

int tmp[3][3]{};

//підраховуємо значення та записуємо в масив tmp

for (int i = 1; i <= 16; i += i)

{

M = i;

N = i;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

tmp[0][0] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[0][1] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[0][2] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int i = 1; i <= 16; i += i)

{

M = i;

N = i;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

tmp[1][0] += Select5(Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[1][1] += Select5(Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[1][2] += Select5(Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int i = 1; i <= 16; i += i)

{

M = i;

N = i;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

tmp[2][0] += Select8(Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[2][1] += Select8(Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[2][2] += Select8(Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

P = 32000;

for (int s = 2, r = 800; s <= 16; s += s, r /= 2)

{

M = s;

N = r;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

tmp[0][0] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[0][1] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[0][2] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int s = 2, r = 800; s <= 16; s += s, r /= 2)

{

M = r;

N = s;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

tmp[0][0] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[0][1] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[0][2] += timeQuickSort(0, P - 1, Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int s = 2, r = 800; s <= 16; s += s, r /= 2)

{

M = s;

N = r;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

fill\_sorted(Arr3D, P, M, N);

tmp[1][0] += Select5(Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[1][1] += Select5(Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[1][2] += Select5(Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int s = 2, r = 800; s <= 16; s += s, r /= 2)

{

M = r;

N = s;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

fill\_sorted(Arr3D, P, M, N);

tmp[1][0] += Select5(Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[1][1] += Select5(Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[1][2] += Select5(Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int s = 2, r = 800; s <= 16; s += s, r /= 2)

{

M = s;

N = r;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

fill\_sorted(Arr3D, P, M, N);

tmp[2][0] += Select8(Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[2][1] += Select8(Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[2][2] += Select8(Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

for (int s = 2, r = 800; s <= 16; s += s, r /= 2)

{

M = r;

N = s;

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

Arr3D[i] = new int\* [M];

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

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

Arr3D[i][j] = new int[N];

fill\_sorted(Arr3D, P, M, N);

fill\_sorted(Arr3D, P, M, N);

tmp[2][0] += Select8(Arr3D, P, M, N);

fill\_random(Arr3D, P, M, N);

tmp[2][1] += Select8(Arr3D, P, M, N);

fill\_reversed(Arr3D, P, M, N);

tmp[2][2] += Select8(Arr3D, P, M, N);

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

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

delete[]Arr3D[i][j];

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

delete[]Arr3D[i];

}

//ділимо отриманні значення на 13(кількість тестів), щоб отримати середнє арифметичне

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

tmp[i][j] /= 13;

//виводимо таблицю з результатами

cout << "-------------------------------------------" << endl;

cout << "| | Sorted | Random | Reversed |" << endl;

cout << "-------------------------------------------" << endl;

cout << "| Quick sort |" << setw(5) << tmp[0][0] << setw(4) << "|" << setw(5) << tmp[0][1] << setw(4) << "|"<< setw(6) << tmp[0][2] << setw(5) << "|" << endl;

cout << "-------------------------------------------" << endl;

cout << "| Select 5 |" << setw(5) << tmp[1][0] << setw(4) << "|" << setw(5) << tmp[1][1] << setw(4) << "|" << setw(6) << tmp[1][2] << setw(5) << "|" << endl;

cout << "-------------------------------------------" << endl;

cout << "| Select 8 |" << setw(5) << tmp[2][0] << setw(4) << "|" << setw(5) << tmp[2][1] << setw(4) << "|" << setw(6) << tmp[2][2] << setw(5) << "|" << endl;

cout << "-------------------------------------------" << endl;

delete[]\*\*Arr3D;

}

void menu()

{

int option;

//цикл для повторного введення даних

while (true)

{

cout << "----------------------------------" << endl;

cout << "Choose the option(type the digit):" << endl;

cout << "1. Vector" << endl;

cout << "2. 3D" << endl;

cout << "3. All" << endl;

cout << "4. Exit" << endl;

cin >> option;

switch (option)

{

//перший випадок одновимірний масив

case 1: vector();

break;

//другий випадок тривімірний масив

case 2: D3();

break;

//тест всіх випадків одночасно

case 3: All();

break;

//вихід з меню

case 4: return;

break;

default: cout << "Incorrect data!" << endl;

break;

}

}

}

int main()

{

srand(time(0));

//меню для вибору функцій програми

menu();

}

**Fill.h**

#pragma once

#include<ctime>

#include<cstdlib>

//модуль для заповнення тривимірного масиву числами

void fill\_sorted(int\*\*\* Arr3D, int P, int M, int N);

void fill\_reversed(int\*\*\* Arr3D, int P, int M, int N);

void fill\_random(int\*\*\* Arr3D, int P, int M, int N);

**Fill.cpp**

#include "fill.h"

//впорядкований

void fill\_sorted(int\*\*\* Arr3D, int P, int M, int N)

{

int number = 0;

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

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

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

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

}

//оберенено впорядкований

void fill\_reversed(int\*\*\* Arr3D, int P, int M, int N)

{

int number = P\*M\*N;

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

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

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

Arr3D[k][i][j] = number--;

}

//невпорядкований

void fill\_random(int\*\*\* Arr3D, int P, int M, int N)

{

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

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

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

Arr3D[k][i][j] = rand() % P \* M \* N;

}

**Sum.h**

#pragma once

//модуль для пошуку суми всіх перерізів заданого масиву

void sum(int\*\*\* Arr3D, int\* Sum, int P, int M, int N);

**Sum.cpp**

#include"sum.h"

void sum(int\*\*\* Arr3D, int\* Sum, int P, int M, int N)

{

int tmp;

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

{

tmp = 0;

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

{

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

{

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

}

}

Sum[i] = tmp;

}

}

**Sort.h**

#pragma once

#include <time.h>

#include <windows.h>

#include "sum.h"

//модуль для сортування тривимірного масиву

//функція для перестановки перерізів місцями

void Swap(int\*\*\* Arr3D, int M, int N, int imin, int s);

//функція для пошуку часу алгоритму швидкого сортування

clock\_t timeQuickSort(int L, int R, int\*\*\* Arr3D, int P, int M, int N);

//алгоритм швидкого сортування

void quickSort(int\* Sum, int L, int R, int\*\*\* Arr3D, int P, int M, int N);

//алгоритм вибору номер 5

clock\_t Select5(int\*\*\* Arr3D, int P, int M, int N);

//алгоритм вибору номер 8

clock\_t Select8(int\*\*\* Arr3D, int P, int M, int N);

**Sort.cpp**

#include "sort.h"

void Swap(int\*\*\* Arr3D, int M, int N, int imin, int s)

{

int tmp;

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

{

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

{

tmp = Arr3D[imin][i][j];

Arr3D[imin][i][j] = Arr3D[s][i][j];

Arr3D[s][i][j] = tmp;

}

}

}

clock\_t timeQuickSort(int L, int R, int\*\*\* Arr3D, int P, int M, int N)

{

clock\_t time\_start, time\_stop;

time\_start = clock();

int\* Sum = new int[P];

sum(Arr3D, Sum, P, M, N);

quickSort(Sum, 0, P - 1, Arr3D, P, M, N);

time\_stop = clock();

delete[]Sum;

return time\_stop - time\_start;

}

void quickSort(int\* Sum, int L, int R, int\*\*\* Arr3D, int P, int M, int N)

{

int B, tmp, i, j;

B = Sum[(L + R) / 2]; // центральний елемент

i = L, j = R;

// процедура розділення

while (i <= j)

{

while (Sum[i] < B) i++;

while (Sum[j] > B) j--;

if (i <= j) {

tmp = Sum[i];

Sum[i] = Sum[j];

Sum[j] = tmp;

Swap(Arr3D, M, N, i, j);

i++;

j--;

}

}

// рекурсивні виклики, якщо є, що сортувати

if (L < j) quickSort(Sum, L, j, Arr3D, P, M, N);

if (i < R) quickSort(Sum, i, R, Arr3D, P, M, N);

}

clock\_t Select5(int\*\*\* Arr3D, int P, int M, int N)

{

int Min, imin;

clock\_t time\_start, time\_stop;

time\_start = clock();

int\* Sum = new int[P];

sum(Arr3D, Sum, P, M, N);

for (int s = 0; s < P - 1; s++)

{

Min = Sum[s];

imin = s;

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

if (Sum[i] < Min)

{

Min = Sum[i];

imin = i;

}

if (imin != s)

{

Sum[imin] = Sum[s];

Sum[s] = Min;

Swap(Arr3D, M, N, imin, s);

}

}

delete[]Sum;

time\_stop = clock();

return time\_stop - time\_start;

}

clock\_t Select8(int\*\*\* Arr3D, int P, int M, int N)

{

int L, R, imin, imax, tmp;

clock\_t time\_start, time\_stop;

time\_start = clock();

int\* Sum = new int[P];

sum(Arr3D, Sum, P, M, N);

L = 0; R = P - 1;

while (L < R)

{

imin = L;

imax = L;

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

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

imin = i;

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

imax = i;

if (imin != L)

{

tmp = Sum[imin];

Sum[imin] = Sum[L];

Sum[L] = tmp;

Swap(Arr3D, M, N, imin, L);

}

if (imax != R)

if (imax == L)

{

tmp = Sum[imin];

Sum[imin] = Sum[R];

Sum[R] = tmp;

Swap(Arr3D, M, N, imin, R);

}

else

{

tmp = Sum[imax];

Sum[imax] = Sum[R];

Sum[R] = tmp;

Swap(Arr3D, M, N, imax, R);

}

L = L + 1;

R = R - 1;

}

delete[]Sum;

time\_stop = clock();

return time\_stop - time\_start;

}

**Vector.h**

#pragma once

#include <iostream>

#include <time.h>

#include <windows.h>

#include <iomanip>

//модуль для сортування одновимірного масиву

//функція для пошуку часу роботи алгоритму швидкого сортування

clock\_t Vtime\_quick\_sort(int n, int\* A, int l, int r);

//алгоритм швидкого сортування

void Vquick\_sort(int n, int\* A, int l, int r);

//алгоритм вибору номер 5

clock\_t VSelect5(int\* A, int N);

//алгоритм вибору номер 8

clock\_t VSelect8(int\* A, int N);

**Vector.cpp**

#include"vector.h"

clock\_t Vtime\_quick\_sort(int n, int\* A, int L, int R)

{

clock\_t time\_start, time\_stop;

time\_start = clock();

Vquick\_sort(n, A, L, R);

time\_stop = clock();

return time\_stop - time\_start;

}

void Vquick\_sort(int n, int\* A, int L, int R)

{

int b, i, j;

b = A[(L + R) / 2];

i = L; j = R;

while (i <= j)

{

while (A[i] < b)i++;

while (A[j] > b)j--;

if (i <= j)

{

int tmp = A[i];

A[i] = A[j];

A[j] = tmp;

i++;

j--;

}

}

if (L < j)

Vquick\_sort(n, A, L, j);

if (i < R)

Vquick\_sort(n, A, i, R);

}

clock\_t VSelect5(int\* A, int N)

{

int Min, imin;

clock\_t time\_start, time\_stop;

time\_start = clock();

for (int s = 0; s < N - 1; s++)

{

Min = A[s];

imin = s;

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

if (A[i] < Min)

{

Min = A[i];

imin = i;

}

if (imin != s)

{

A[imin] = A[s];

A[s] = Min;

}

}

time\_stop = clock();

return time\_stop - time\_start;

}

clock\_t VSelect8(int\* A, int N)

{

int L, R, imin, imax, tmp;

clock\_t time\_start, time\_stop;

time\_start = clock();

L = 0; R = N - 1;

while (L < R)

{

imin = L;

imax = L;

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

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

imin = i;

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

imax = i;

if (imin != L)

{

tmp = A[imin];

A[imin] = A[L];

A[L] = tmp;

}

if (imax != R)

if (imax == L)

{

tmp = A[imin];

A[imin] = A[R];

A[R] = tmp;

}

else

{

tmp = A[imax];

A[imax] = A[R];

A[R] = tmp;

}

L = L + 1;

R = R - 1;

}

time\_stop = clock();

return time\_stop - time\_start;

}

**Інформація про комп'ютер**

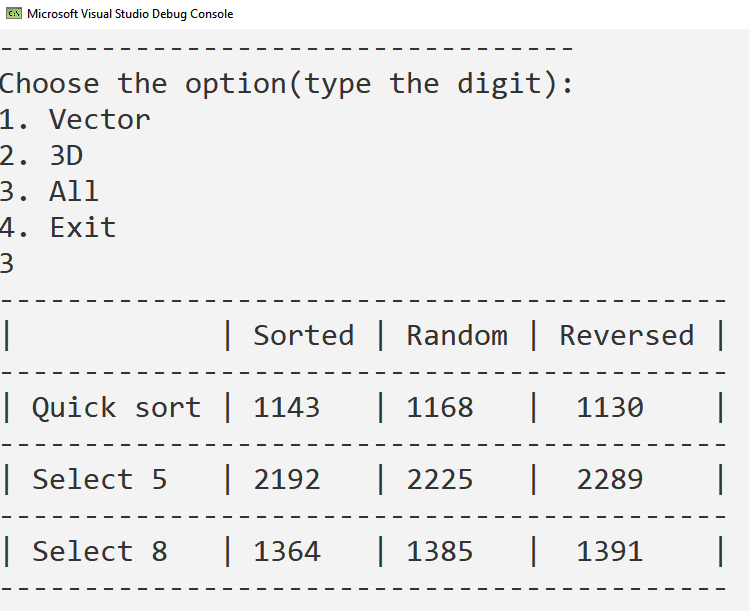
Операційна система – Windows 10 (x64)

Процесор - AMD Ryzen 5 2600

Оперативна пам’ять – 16 ГБ

Компілятор – Microsoft Visual Studio 2019

**Тести**

****

**Результати**

Таблиця №1 для вектора A[N], де N = 150,000

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 8 | 19 | 8 |
| Вибору №5 | 25498 | 25466 | 27676 |
| Вибору №8 | 13719 | 13782 | 14188 |

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

Рекомендовані розміри масива для досліджень:

Кількість ключів (перерізів) P = const = 1,500,000

Форма перерізу – однакова (квадрат)

M = var, N = var, M = N

Було обрано P = 50000, тобто менше рекомендованого в 30 разів, оскільки при більших значеннях виникала помилка переповнення стеку при використанні алгоритму швидкого сортування.

Таблиця №2 для масива A[P, M, N], де P = 50,000; M = 1; N = 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 4 | 13 | 5 |
| Вибору №5 | 2830 | 2835 | 3073 |
| Вибору №8 | 1524 | 1543 | 1585 |

Таблиця №3 для масива A[P, M, N], де P = 50,000; M = 2; N = 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 7 | 20 | 9 |
| Вибору №5 | 2827 | 2845 | 3074 |
| Вибору №8 | 1520 | 1545 | 1582 |

Таблиця №4 для масива A[P, M, N], де P = 50,000; M = 4; N = 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 14 | 44 | 21 |
| Вибору №5 | 2839 | 2855 | 3082 |
| Вибору №8 | 1525 | 1558 | 1590 |

Таблиця №5 для масива A[P, M, N], де P = 50,000; M = 8; N = 8

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 36 | 131 | 51 |
| Вибору №5 | 2838 | 2885 | 3098 |
| Вибору №8 | 1501 | 1590 | 1613 |

Таблиця №6 для масива A[P, M, N], де P = 50,000; M = 16; N = 16

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 254 | 529 | 156 |
| Вибору №5 | 2961 | 2990 | 3052 |
| Вибору №8 | 1702 | 1703 | 1627 |

**Випадок дослідження ІI. Залежність часу роботи алгоритмів від форми перерізів масива**

Рекомендовані розміри масива для досліджень:

Кількість ключів (перерізів) P = const = 128,000

Форма перерізу – однакова (квадрат)

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

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

Було обрано P = 32,000, тобто менше рекомендованого в 4 рази, оскільки при більших значеннях виникала помилка переповнення стеку при використанні алгоритму швидкого сортування.

Таблиця №7 для масива A[P, M, N], де P = 32,000; M = 2; N = 800

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 1317 | 1345 | 1302 |
| Вибору №5 | 1563 | 1675 | 1689 |
| Вибору №8 | 1028 | 1058 | 1028 |

Таблиця №8 для масива A[P, M, N], де P = 32,000; M = 4; N = 400

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 1442 | 1395 | 1329 |
| Вибору №5 | 1592 | 1572 | 1560 |
| Вибору №8 | 1099 | 1056 | 1062 |

Таблиця №9 для масива A[P, M, N], де P = 32,000; M = 8; N = 200

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 1607 | 1391 | 1442 |
| Вибору №5 | 1563 | 1592 | 1558 |
| Вибору №8 | 1068 | 1060 | 1054 |

Таблиця №10 для масива A[P, M, N], де P = 32,000; M = 16; N = 100

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 1419 | 1507 | 1400 |
| Вибору №5 | 1619 | 1730 | 1611 |
| Вибору №8 | 1091 | 1087 | 1103 |

Таблиця №11 для масива A[P, M, N], де P = 32,000; M = 100; N = 16

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 1624 | 1734 | 1859 |
| Вибору №5 | 1778 | 1735 | 1806 |
| Вибору №8 | 1275 | 1207 | 1255 |

Таблиця №12 для масива A[P, M, N], де P = 32,000; M = 200; N = 8

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 1926 | 1786 | 1935 |
| Вибору №5 | 1840 | 1798 | 1850 |
| Вибору №8 | 1339 | 1312 | 1325 |

Таблиця №13 для масива A[P, M, N], де P = 32,000; M = 400; N = 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 2263 | 2420 | 2266 |
| Вибору №5 | 2051 | 1978 | 1981 |
| Вибору №8 | 1504 | 1494 | 1506 |

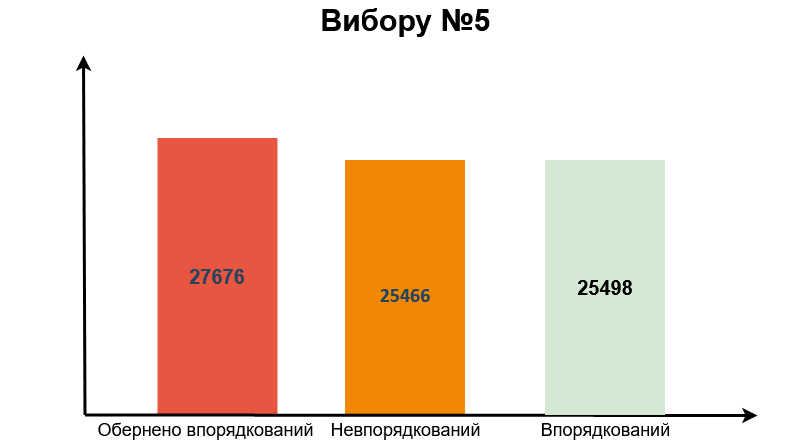
Таблиця №14 для масива A[P, M, N], де P = 32,000; M = 800; N = 2

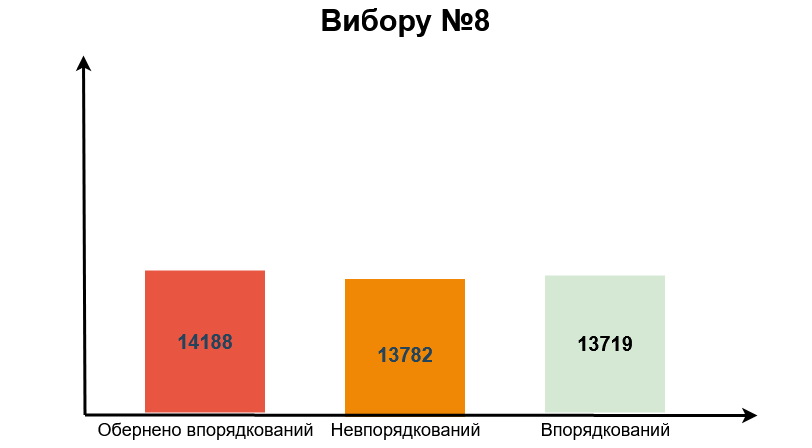
|  |  |  |  |
| --- | --- | --- | --- |
|  | Впорядкований | Невпорядкований | Обернено впорядкований |
| Швидке сортування | 2529 | 2546 | 2586 |
| Вибору №5 | 2236 | 2238 | 2212 |
| Вибору №8 | 1712 | 1708 | 1718 |

*Графіки*

**Вектор**







**Порівняльний аналіз алгоритмів**

Результати тестів сходяться з наведеними числами на лекції:

Для прямого вибору час сортування:

Мінімальний Середній Максимальний

0,94 0,96 1,18

Для швидкого час сортування:

Мінімальний Середній Максимальний

0,08 0,12 0,08

**Вектор**

Як видно з графіків алгоритм сортування №5 і №8 *методу прямого* *вибору* практично не залежать від впорядкованості масиву. Це можна пояснити тим, що ці алгоритми мають однакову кількість повторень циклів незалежно від впорядкованості масиву.

Проте для обернено впорядкованого масиву, ці алгоритми сортування при пошуку мінімального елементу на кожній ітерації знаходять новий – тобто витрачають час на присвоєння. А при впорядкованому або не впорядкованому це відбується значно рідше. Також для обернено впорядкованого масиву на кожній ітерації зовнішнього циклу гарантовано відбувається обмін елементів між відсортованими та невідсортованими частинами масиву.

Також хочу зазначити, що алгоритм номер 8 працює швидше близько на 50-100%, ніж 5. Цього вдалося досягти завдяки перестановці двух, замість одного, елементів за одну ітерацію.

*Швидке* сортування працює приблизно з однаковою швидкістю для відсортованого та обернено відсортованого масиву, оскільки на першому рекурсивному виклику всі елементи, які менші обраного(в нашому випадку центрального), розміщуються зліва, а більші – справа. Отже, при відсортованому масиві швидкому сортуванню потрібен лише один рекурсивний виклик для перевірки цього. А при обернено відсортованому масиві також потрібен тільки один рекурсивний виклик, адже при пошуку на кожний менший елемент справа знайдеться більший зліва.

Натомість при невпорядкованому масиві кількість рекурсивних викликів невідома – отже час роботи також може бути різним, зазвичай більшим, ніж в перших двох випадках.

**Багатовимірний масив**

Багатовимірний масив розташований в пам’яті так само, як і одновимірний. Він зроблений лише для зручності роботи. Тобто сортування в багатовимірному масиві відбувається так само, як і у векторі.

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

З графіків можна побачити, що зі збільшенням розміру перерізу час роботи алгоритмів вибору майже не зростає. Це зумовлено тим, що ці алгоритми переставляють перерізи відразу на правильні місця, тобто за мінімальну кількість присвоєнь. Оскільки, кількість перерізів значно більша ніж їх розмір, тому сортування вибору працює приблизно з однаковою швидкістю.

Алгоритм швидкого сортування спочатку також працює приблизно з тією ж швидкістю на різних за розміром перерізах. Але коли розмір перерізу зростає суттєво, то алгоритм швидкого сортування сильно сповільнюється. Це зумовлено тим, що цей алгоритм не завжди відразу ставить перерізи на праввильну позицію, отже виконує зайві перестановки, які зі збільшенням перерізу впливають сильніше і сильніше на швидкодію алгоритму.

Отже, для масивів з великими перерізами алгоритм сортування вибору підійде краще.

**Випадок дослідження ІI. Залежність часу роботи алгоритмів від форми перерізів масива**

Вимірявши час роботи функції Swap, я зрозумів, що все спирається на неї. Чомусь ця функція обмінювала елементи довше при формі перерізу, коли M більше N. Тобто в сортуванні з перестановкою перерізів все спирається лише на швидкість роботи функції обміну. Саме тому швидке сортування стало таким повільним після зміни форми перерізу, навіть повільніше алгоритму вибору 5.

**Висновки**

Завдяки цій курсовій роботі я навчився коректно та швидко оцінювати алгоритми, обирати найбільш ефективний для кожної ситуації. Я розлядав 3 алгоритми, два з яких були прямого вибору, тобто я міг порівняти дві різні модифікації методу вибору. Виявилось, що алгоритм номер 8 працює на 50-100% швидше п’ятого алгоритму. Також обидва алгоритми методу вибору працювали гірше з обернено впорядкованим масивом.

Алгоритм швидкого сортування виявився найшвидшим у більшості ситуацій, крім тих, коли потрібно переставляти великі перерізи. Алгоритм сортування Хоара найшвидше працює з відсортованими та обернено відсортованими масивами – добре підійде для сортування таких.

Було виявлено, що обмін елементів у функції Swap відбувається повільніше при M>N формі перерізу. Відповідно це повпливало на швидкодію алгоритмів сортування.

**Список використаної літератури**

1. Конспект лекцій з курсу «Структури даних та алгоритми».
2. Методичні матеріали для виконання курсової роботи.