Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем та програмної інженерії

Кафедра програмної інженерії

**З В І Т**

до виконання лабораторної роботи №6

з дисципліни «Алгоритми та структури даних»

на тему «Алгоритми сортування»

Виконав:

студент групи СП-11

Бондар Віталій

Варіант 4(1)

Тернопіль 2022

***Мета роботи:*** набути практичних навичок по реалізації різноманітних алгоритмів сортування та порівняти швидкості виконання простих і швидких алгоритмів

**Теоретичні відомості**

**Бульбашкове сортування**

Щоб описати схему роботи даного алгоритму, уявимо собі що структури, які необхідно посортувати лежать в масиві, розташованому вертикально.

Структури з малим значенням ключового поля більш легкі і в спливають догори як бульбашки. При першому проході масиву, береться перший об’єкт і значення його ключа порівнюється із ключовими полями усіх іншими об’єктів в масиві. Коли зустрічається об’єкт із більш легким значенням ключа, то об’єкти міняються місцями, і вже цей об’єкт стає еталоном для порівняння.

В результаті структура з найменшим значенням ключа виявляється на вершині. Під час другого проходу, знаходиться об’єкт із другим по величині ключем і він розміщується на позиції одразу під вершиною.

**Сортування вставками-**при сортуванні вставками на і-тому кроці ми вставляємо і-тий елемент А[i] в потрібну позицію серед елементів A[1], A[2],…, A[i-1], які вже впорядковані. Після цієї вставки перші і елементів будуть впорядкованими.

**Швидке сортування**-цей алгоритм є одним з най ефективніших методів внутрішнього сортування. Для сортування елементів A[1]…A[n], вибирається значення ключа v в якості опорного елементу, відносно якого перевпорядковується всі елементи масиву.

Значення опорного елементу необхідно вибрати близьким по значенню до медіани розподілу значень ключів, так щоб опорний елемент розбивав множину значень ключів на дві приблизно рівні частини.

Далі елементи масиву переставляються так, щоб для деякого індексу j всі переставлені елементи A[1],…,A[j] мали значення ключів, менше від v, а всі елементи A[j+1],…,A[n] – значення ключів, більше або рівне v.

Після цього, процедура швидкого сортування рекурсивно застосовується до множини елементів A[1],…,A[j] та A[j+1],…,A[n] для впорядкування цих множин окремо.

Оскільки значення всіх ключів в першій множині менші за значення ключів в другій множині, то вихідний масив буде відсортовано вірно. Важливим елементом даного сортування є знаходження значення опорного ключа, відносно якого буде проведено перерозподіл елементів масиву. Якщо функція не знаходить різних значень ключів, то повертає 0, в іншому випадку вона повертає індекс найбільшого із перших двох різних ключів. Цей найбільший ключ стає опорним.

**Завдання 1**

1. Посортувати масив із 10ти елементів типу int, використовуючи

 бульбашкове сортування

IDE: Microsoft Visual Studio 2019

Лістинг програми

#include <iostream>

using namespace std;

int main()

{

const int size = 10;

int mas[size];

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

cout << "Enter " << i+1 << " element: ";

cin >> mas[i];

}

cout << endl;

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

cout << mas[i] <<" ";

}

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

for (int j = size - 1; j > i; j--) {

if (mas[j] < mas[j - 1]) {

swap(mas[j], mas[j - 1]);

}

}

}

cout << endl;

cout << endl;

cout <<"Massive after sorting"<< endl;

cout << endl;

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

cout << mas[i] << " ";

}

return 0;

}

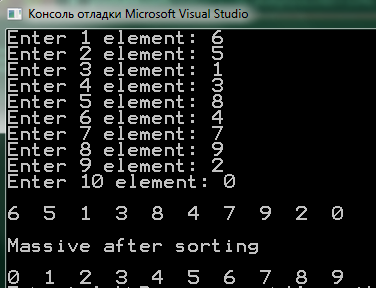


Рис.1 – результат виконання завдання 1

**Завдання 2**

Реалізувати алгоритм швидкого сортування для впорядкування в неспадній послідовності масив структур:

struct INF {

            int key;

            char info;

};

Лістинг програми

#include <iostream>

using namespace std;

struct INF {

int key;

char info;

};

void quickSort(INF\* mas, int first, int last)

{

int i = first;

int j = last;

int mid = mas[(first + last) / 2].key;

do

{

while (mas[i].key < mid)

{

i++;

}

while (mid < mas[j].key)

{

j--;

}

if (i <= j)

{

swap(mas[i], mas[j]);

i++;

j--;

}

} while (i < j);

if (first < j)

{

quickSort(mas, first, j);

}

if (i < g)

{

quickSort(mas, i, last);

}

}

int main()

{

int n;

start:

cout << "Enter n:";

cin >> n;

if (n < 1) {

cout << "Error num.Try again" << endl;

goto start;

}

INF \*mas=new INF[n];

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

{

cout << "Enter " << i + 1 << " key: ";

cin >> mas[i].key;

cout << "Enter info:";

cin >> mas[i].info;

} quickSort(mas, 0, n - 1);

cout << "Massive after sorting:" << endl;

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

{

cout << mas[i].key << " - " << mas[i].info << endl;

}

return 0;

}



Рис.2 – результат виконання завдання 2

**Завдання 3**

Провести порівняльний аналіз  часу сортування масиву з 50000 елементів типу int для алгоритму швидкого сортування, та для алгоритму сортування вставками

Лістинг програми

#include <iostream>

#include<time.h>

using namespace std;

void quickSort(int mas[], int first,int last)

{

int i = first;

int j = last;

int mid = mas[(first + last) / 2];

do

{

while (mas[i] < mid)

{

i++;

}

while (mid < mas[j])

{

j--;

}

if (i <= j)

{

swap(mas[i], mas[j]);

i++;

j--;

}

} while (i < j);

if (first < j)

{

quickSort(mas, first, j);

}

if (i < last)

{

quickSort(mas, i, last);

}

}

void insertionSort(int mas[],int NUM) {

int t;

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

t = mas[i];

for (int j = i - 1; j >= 0 && mas[j] > t; j--) {

mas[j + 1] = mas[j];

mas[j + 1] = t;

}

}

}

void CreateArray(int A[], int B[], int NUM) {

srand(1);

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

A[i] = B[i] = (int)rand();

}

}

void RestoreArray(int A[], int B[], int NUM)

{

for (int i = 0; i < NUM; i++) A[i] = B[i];

}

int main()

{

const unsigned int NUM = 50000;

int A[NUM];

int B[NUM];

CreateArray(A, B, NUM);

clock\_t begt, endt;

begt = clock();

quickSort(A, 0, NUM - 1);

endt = clock();

cout << "Quick sort" << endl;

printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock();

quickSort(A, 0, NUM - 1);

endt = clock();

printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock();

quickSort(A, 0, NUM - 1); endt = clock(); printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock(); quickSort(A, 0, NUM - 1); endt = clock(); printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock(); quickSort(A, 0, NUM-1); endt = clock(); printf("time=%d\n", endt - begt);

insertionSort(A, NUM);

endt = clock();

cout << endl;

cout << "Insertion sort" << endl;

printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock();

insertionSort(A, NUM);

endt = clock();

printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock(); insertionSort(A, NUM); endt = clock(); printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock(); insertionSort(A, NUM); endt = clock(); printf("time=%d\n", endt - begt);

RestoreArray(A, B, NUM);

begt = clock(); insertionSort(A, NUM); endt = clock(); printf("time=%d\n", endt - begt);

return 0;

}

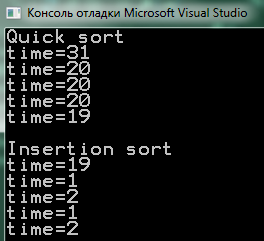
****

Рис.3 – результат виконання завдання 3

**Висновок.** набув практичних навичок по реалізації різноманітних алгоритмів сортування та порівняв швидкості виконання простих і швидких алгоритмів