Лабораторная работа №6

“Анализ алгоритмов сортировки”

1 ЦЕЛЬ РАБОТЫ

Научиться оценивать сложность и количество операций для алгоритмов сортировки.

2 ПОСТАНОВКА ЗАДАЧИ

1. Для каждого из приведённых алгоритмов найти оценку для количества шагов и количества требуемой памяти.

2. Создать структуру и реализовать алгоритм сортировки согласно варианту задания.

3. Реализовать более эффективные алгоритмы сортировки согласно варианту задания.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | Простейший алгоритм 1 | Простейший алгоритм 2 | Быстрый алгоритм | Структура и критерий сортировки |
| 3 | Сортировка вставками | Метод «пузырька» | Сортировка слиянием | Структура Студент, содержит ФИО, курс, факультет, сортировка по возрастанию курса |

4. Сравнить производительность различных алгоритмов.

3 ТЕКСТ ПРОГРАММЫ

#include <conio.h>

#include <time.h>

#include <ctime>

#include <windows.h>

#include <iostream>

#include <stdio.h>

using namespace std;

const int N = 10000;

struct student {

char fio[10];

int faculty;

int kurs;

};

void insertSort(student \*a, int n)

{

int i, j;

for(i = 2; i <= n; i++)

{

a[0].kurs = a[i].kurs;

for(j = i-1; a[j].kurs > a[0].kurs; j--)

a[j+1].kurs = a[j].kurs;

a[j+1].kurs = a[0].kurs;

}

}

void bubbleSort(student \*a, int n)

{

int i, j;

for(i = n; i > 1; i--)

for(j = 1; j < i; j++)

if(a[j].kurs > a[j+1].kurs)

swap(a[j].kurs, a[j+1].kurs);

}

void merge(student \*a, int l, int r)

{

student b[N];

int m = (l+r) / 2;

int i, j, k;

i = l;

j = m+1;

k = l;

while(i <= m || j <= r)

{

if(i > m)

{

b[k++].kurs = a[j++].kurs;

continue;

}

if(j > r)

{

b[k++].kurs = a[i++].kurs;

continue;

}

if(a[i].kurs < a[j].kurs)

b[k++].kurs = a[i++].kurs;

else

b[k++].kurs = a[j++].kurs;

}

for(i = l; i <= r; i++)

a[i].kurs = b[i].kurs;

}

void mergeSort(student \*a, int l, int r)

{

if(l == r)

return;

int m = (l + r) / 2;

mergeSort(a, l, m);

mergeSort(a, m+1, r);

merge(a, l, r);

}

int main()

{

setlocale(0, "Russian");

FILE \*def, \*res;

def = fopen("Default.txt", "w+");

res = fopen("Result.txt", "w+");

student a[N];

int n = N;

int key;

for (int i = 0; i < n; i++) a[i].kurs = rand()%100;

for (int i = 0; i < n; i++) fprintf(def, "%i ", a[i].kurs);

cout << "Как отсортировать массив?" << endl;

cout << "1 - Вставками" << endl;

cout << "2 - Пузырьком" << endl;

cout << "3 - Слияниями" << endl;

cin >> key;

double start\_time, end\_time;

switch (key)

{

case 1:

start\_time = clock();

insertSort(a, N);

end\_time = clock();

break;

case 2:

start\_time = clock();

bubbleSort(a, N);

end\_time = clock();

break;

case 3:

start\_time = clock();

mergeSort(a, 0, N-1);

end\_time = clock();

break;

}

double time = end\_time - start\_time;

cout << "Исходный и отсортированный массивы \nсохранены в файлах Default.txt и Result.txt" << endl;

for (int i = 0; i < n; i++) fprintf(res, "%i ", a[i].kurs);

cout << "Время выполнения: = " << time/(double)CLOCKS\_PER\_SEC << endl;

fclose(res);

fclose(def);

return 0;

}

4 РЕЗУЛЬТАТЫ

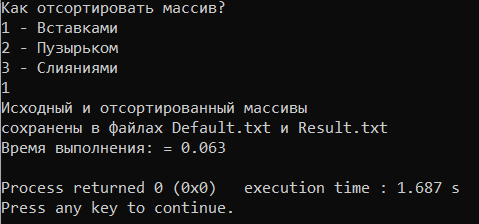


Рисунок 1 – Результат сортировки вставками

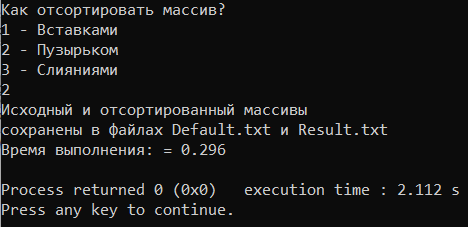


Рисунок 2 – Результат сортировки пузырьком

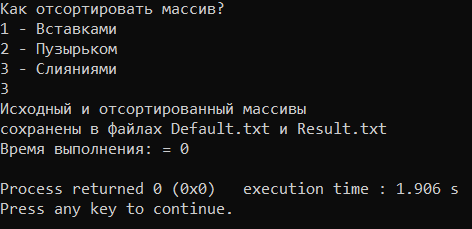


Рисунок 3 – Результат сортировки слиянием



Рисунок 4 – Исходный массив

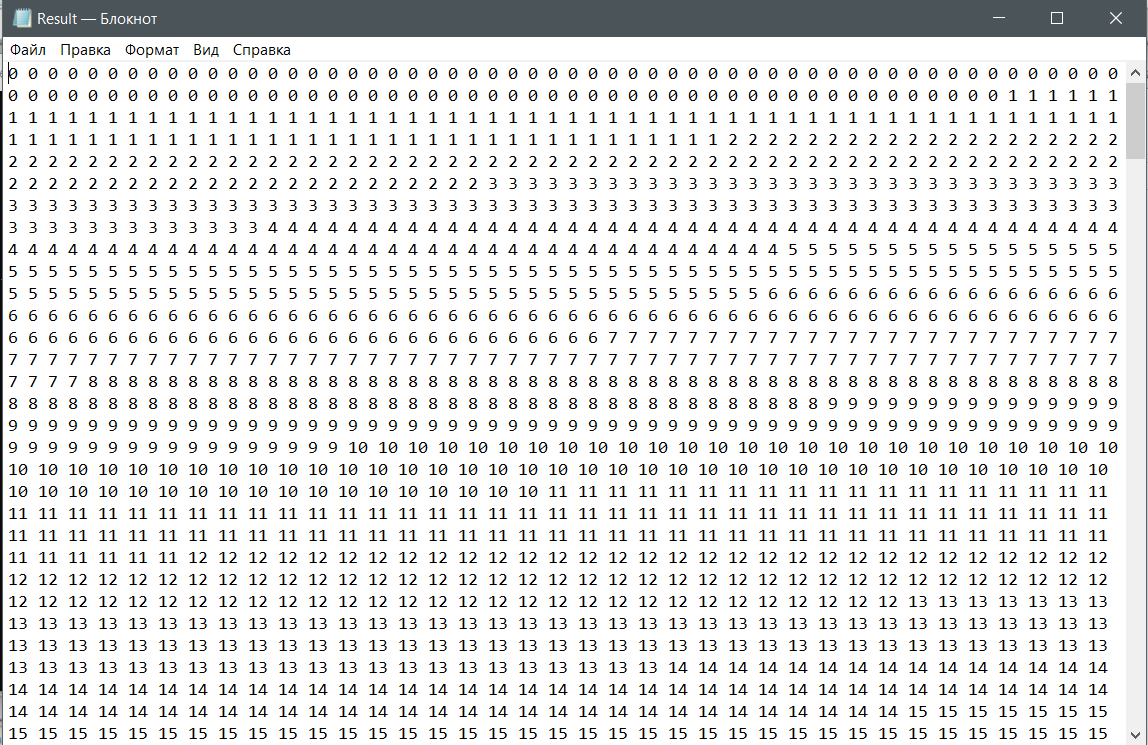


Рисунок 5 – Результат сортировки

ВЫВОД

В ходе выполнения лабораторной работы были приобретены навыки оценки сложности и эффективности различных алгоритмов сортировки. Были написаны три алгоритма сортировки: вставками, пузырьком и слиянием. Оценка эффективности показала, что наиболее быстрым является алгоритм сортировки слиянием, а наиболее долгим – сортировка пузырьком.