**6 Лабораторная работа №6**

**«АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ»**

**6.1 Цель работы**

Научиться оценивать сложность и количество операций для алгоритмов сортировки.

**6.2 Вариант задания ­– 14**

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

Создать структуру и реализовать алгоритм сортировки согласно варианту задания. Критерий сортировки указан в варианте. (Таблица 6.1).

Реализовать более эффективные алгоритмы сортировки согласно варианту задания.

Сравнить производительность различных алгоритмов.

Таблица 6.1 – Вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| Простейший алгоритм 1 | Простейший алгоритм 2 | Быстрый алгоритм 3 | Структура и критерий сортировки |
| Модифицированный метод «пузырька» | Сортировка посредством выбора | Сортировка Шелла | Структура Товар, содержит название, стоимость и код, сортировка по убыванию кода. |

**6.3** **Ход работы**

Была написана программа на языке С++ для выполнения задания по варианту.

#include <iostream>

#include <chrono>

#include <stdlib.h>

#include <time.h>

#include <algorithm>

#include <random>

#include <string.h>

#define ITERATIONS\_COUNT 100

#define ARRAY\_SIZE\_MIN 10

#define ARRAY\_SIZE\_MAX 500

#define ARRAY\_SIZE\_STEP 10

using namespace std;

struct data\_t {

int id;

char name[10];

float price;

void operator=(int v) {

id = v;

strcpy(name,"Item name");

price = 1.1f;

}

bool operator>(const data\_t &v) const {

return id > v.id;

}

};

auto seed = default\_random\_engine(chrono::system\_clock::now().time\_since\_epoch().count());

void fill\_array(data\_t \*a,int n) {

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

a[i] = rand();

}

}

void shuffle\_array(data\_t \*a,int n) {

shuffle(a,a+n,seed);

}

void empty(data\_t \*a,int n) {}

void select\_sort(data\_t \*a, int n) {

int i, j, imax;

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

imax = 1;

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

if(a[j] > a[imax])

imax = j;

swap(a[i],a[imax]);

}

}

void bubble\_sort(data\_t \*a,int n) {

int i,j;

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

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

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

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

}

void shell\_sort(data\_t \*a,int n) {

int i,j,temp,step = n/2;

while(step > 0) {

for(i=0;i<(n-step);i++) {

j = i;

while(j >= 0 && a[j] > a[j+step]) {

swap(a[j],a[j+step]);

j--;

}

}

step = step / 2;

}

}

long long pass(data\_t \*a,int n,void (\*f)(data\_t \*,int)) {

auto start\_time = chrono::high\_resolution\_clock::now();

for(int i=0;i<ITERATIONS\_COUNT;i++) {

shuffle\_array(a,n);

f(a,n);

}

auto end\_time = chrono::high\_resolution\_clock::now();

return chrono::duration\_cast<chrono::milliseconds>(end\_time-start\_time).count();

}

int main(int argc,char \*\*argv) {

srand(time(0));

data\_t \*a = new data\_t[ARRAY\_SIZE\_MAX];

for(int n=ARRAY\_SIZE\_MIN;n<=ARRAY\_SIZE\_MAX;n+=ARRAY\_SIZE\_STEP) {

fill\_array(a,n);

cout << n << " ";

cout << pass(a,n,empty) << " ";

cout << pass(a,n,select\_sort) << " ";

cout << pass(a,n,bubble\_sort) << " ";

cout << pass(a,n,shell\_sort) << " ";

cout << endl;

}

delete[] a;

return 0;

}

Результаты работы алгоритмов, полученные в ходе выполнения программы представлены на рисунке 6.1. По оси абсцисс – размерность массива, по оси ординат – количество элементарных операций.

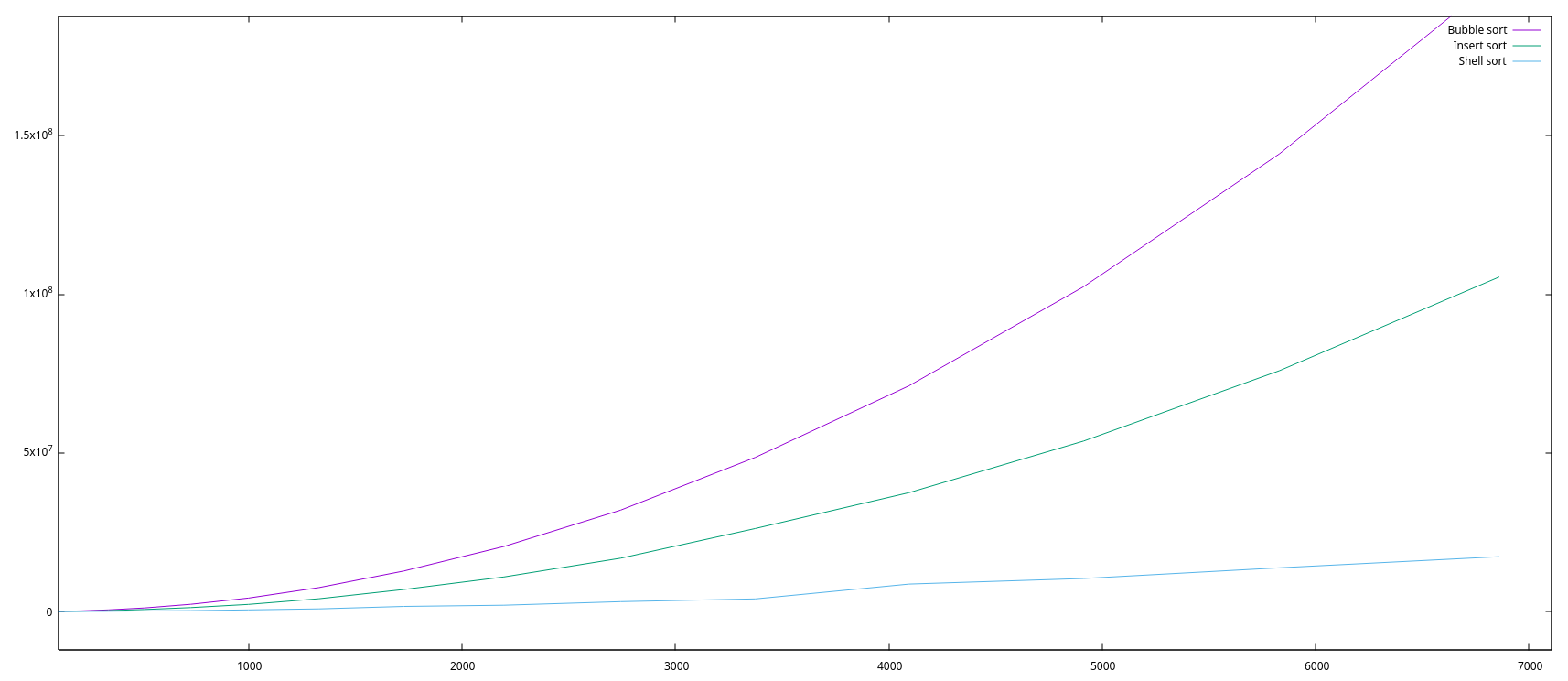


Рисунок 6.1 – График времени выполнения сортировок

**Выводы**

В ходе лабораторной работы были изучены алгоритмы сортировки, оценка их сложности и количества операций. Самой быстрой из предложенных алгоритмов сортировок оказалась сортировка Шела, после неё сортировка прямым выбором, и самой медленной оказалась пузырьковая сортировка, отработавшая при 100 000 раз на массиве из 500 элементов за 176 115 мс.