ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»

Кафедра МОЭВМ

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1**

**на тему:**

**«Сортировка Шелла»**

**по дисциплине «Построение и анализ алгоритмов»**

Выполнила: студентка группы №1381

Лавринович Т. В.

Проверил: Казаков Б.Б.

Санкт-Петербург

2013 г.

**Цель работы**

Изучить усовершенствованный метод сортировки, принцип его работы. Составить программу с использованием алгоритма сортировки Шелла.

**Задание**

Вариант 12. Сортировка Шелла

Сортировка Шелла — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга.

**Описание алгоритма**

Шаг 1. Пользователь вводит массив чисел, которые необходимо отсортировать.

Шаг 2. Сравниваем и сортируем между собой значения, отстоящие один от другого на некотором расстоянии d .

Шаг 3. Повторяем процедуру для некоторых меньших значений d.

Шаг 4. Завершаем сортировку Шелла упорядочиванием элементов при d=1 (то есть обычной сортировкой вставками).

Выбор длинны промежутков d.

Существует несколько подходов к выбору этиx значений. При выполнении работы использовался метод, предложенный Седжвиком:

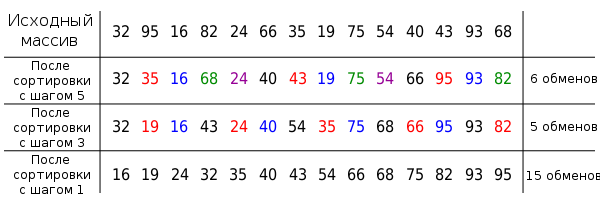
, если i - четное

, если i – нечетное.

Критерий остановки выбора значений d : 3\*d[i] < size, где size – количество элементов массива.

**Пример**

Разер исходного массива =14.



Пусть дан список

А=(32,95,16,82,24,66,35,19,75,54,40,43,93,68) и выполняется его сортировка методом Шелла, а в качестве значений d выбраны 5,3,1.

На первом шаге сортируются подсписки A, составленные из всех элементов A, различающихся на 5 позиций, то есть подсписки

A1 =(32,66,40), A2=(95,35,43), A3=(16,19,93), A4=(82,75,68), A5=(24,54).

В полученном списке на втором шаге вновь сортируются подсписки из отстоящих на 3 позиции элементов.

Процесс завершается обычной сортировкой вставками получившегося списка.

**Контрольные примеры**

Таблица 1. Контрольные примеры

|  |  |  |
| --- | --- | --- |
| Исходная данные | | Результат |
| Размер массива | Массив чисел | Отсортированный массив |
| 5 | 56, 12, 1, 98, 2 | 1, 2, 12, 56, 98 |
| 10 | 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 |
| 15 | 45,3,1,5,9,12,47,98,123,89,14,43,10 | 1,3,5,9,10,12,14,43,45,47,89,98,123 |
| 7 | 4, 9, 15, 3, 45, 110, 250 | 3, 4, 9, 15, 45, 110, 250 |

**Исходная программа**

#include <iostream>

#include <time.h>

using namespace std;

//Алгоритм получения последовательности

int SedgewickAlg(int size, int \*d)

// d массив, в который заносятся инкременты

// size размерность этого массива

{

int p1, p2, p3, i;

p1 = p2 = p3 = 1;

i = -1;

do {// заполняем массив элементов по формуле Роберта Седжвика

if (++i % 2)

d[i] = 8\*p1 - 6\*p2 + 1;

else

{

d[i] = 9\*p1 - 9\*p3 + 1;

p2 \*= 2;

p3 \*= 2;

}

p1 \*= 2;

// заполняем массив, пока текущая инкремента хотя бы

// в 3 раза меньше количества элементов в массиве

} while(3\*d[i] < size);

// возвращаем количество элементов в массиве

return (i > 0) ? (--i) : (0);

}

template<class AnyClass> //Шаблон для использования любого типа данных в //массиве

double shellSort(AnyClass \*a, int size, int &k) { //Возвращает время //выполнения

clock\_t time = clock();

// inc инкремент, расстояние между элементами сравнения

// i и j стандартные переменные цикла

// d[40] массив, в котором хранятся инкременты

int inc = 0, d[40];

int i = 0, j = 0;

int s = 0;//количество элементов в массиве d[40]

// вычисление последовательности приращений

s = SedgewickAlg(size, d);

while (s >= 0)

{

//извлекаем из массива очередную инкременту

inc = d[s--];

// сортировка вставками с инкрементами inc

for (i = inc; i < size; i++)

{

AnyClass temp = a[i];

// сдвигаем элементы до тех пор, пока не дойдем

// до конца или не упорядочим в нужном порядке

for (j = i-inc; (j >= 0) && (a[j] > temp); j -= inc)

{

a[j+inc] = a[j];

++k;

}

// после всех сдвигов ставим на место j+inc элемент,

// который находился на i месте

a[j+inc] = temp;

}

}

return (double(clock() - time));

}

int compare (const void \* a, const void \* b)

{

return ( \*(int\*)a - \*(int\*)b );

}

int main()

{

setlocale(LC\_CTYPE, "Russian");

int n = 0, d;

int k = 0;//Количество базовых опереций (сравнение и перемещение)

cout<<"Введите размер массива.\n";

cin>>n;

int \*a = new int[n];

int \*b = new int[n];

srand((signed int)time(0));

cout<<"\n----------Меню-----------\n";

cout<<" 1)Генерация случайного массива\n";

cout<<" 2)Ввод массива пользователем";

do

{

cout<<"\n Введите d (1 или 2)\n";

cin>>d;

if(d<1||d>2)

cout<<"\n Ошибка! Повторите попытку! \n";

}

while(d<1||d>2);

switch(d)

{

case 1:

{

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

{

a[i] = rand()%10000000; // Генерация чисел в диапазоне //от 0 до 10000000

cout<<a[i]<<" ";

b[i] = a[i];

}

}break;

case 2:

{

cout<<"Введите массив чисел.\n";

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

{

cin>>a[i];

b[i] = a[i];

}

}

}

cout<<"\nВремя выполнения:"<<shellSort(a, n, k)<<endl;

cout<<"Количество базовых операций:"<<k<<endl;

cout<<"отсортированный массив:"<<endl;

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

cout<<a[i]<<" ";

clock\_t time = clock();

qsort(b, n, sizeof(int), compare);

cout<<endl<<"Время сортировки функцией qsort: "<<double(clock() - time)<<endl;

cout<<"отсортированный массив qsort:"<<endl;

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

cout<<b[i]<<" ";

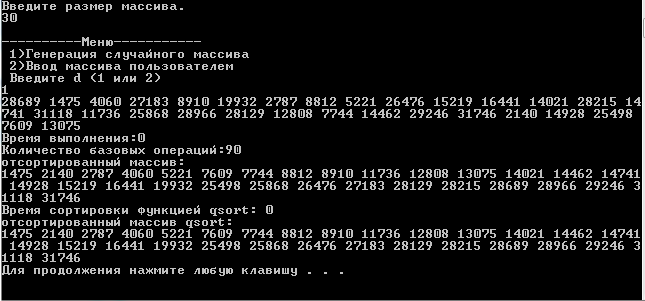
cout<<endl;

system("pause");

return 0;

}

**Результаты выполнения программы**

****

При выполнении программы полученные результаты совпадают с данными Таблицы 1. Ошибки не обнаружены.

**Результаты испытаний**

Оценка сложности алгоритма.

По теоретическим сведениям сложность алгоритма в среднем составляет O(n7/6), а в худшем случае O(n4/3). Сравним практические результаты с теоретическими на сгенерированных наборах.

Таблица 2. Оценка сложности алгоритма

|  |  |  |
| --- | --- | --- |
| Размер массива n | Количество базовых операций k (практика) | Сложность алгоритма в худшем случае |
| 20 | 26 | 54 |
| 100 | 418 | 464 |
| 500 | 3601 | 3969 |
| 1000 | 8137 | 10000 |
| 10000 | 114945 | 215443 |
| 100000 | 1417027 | 4641588 |

Оценка эффективности алгоритма.

Основной показатель эффективности - это время выполнения. Сравним время выполнения разработанной функции сортировки системной функцией qsort на наборе из 100000 символов. Из таблицы 3 видно, что при большом размере массива наша функция работает быстрее.

Таблица 3. Оценка эффективности алгоритма

|  |  |  |
| --- | --- | --- |
| Исходный массив | shellSort, мс | qsort, мс |
| Случайная генерация | 53 | 75 |
| 0,1,2,3,…,100000 | 42 | 55 |
| 100000,…,3,2,1,0 | 47 | 52 |

**Вывод**

При выполнении лабораторной работы №1«Сортировка Шелла» был составлен алгоритм сортировки Шелла и проведен его анализ.