БУ ВО Ханты-Мансийского автономного округа – Югры

«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

ОТЧЁТ

по лабораторной работе №2

по дисциплине: «Алгоритмы и структуры данных»

Выполнил: студент(ка) группы №609-31,

Гаврилов Е.Е.

Принял: старший преподаватель кафедры АиКС

Назаров Е.В.

Сургут

2025г.

**Цель работы**: изучение алгоритмов сортировки, исследование сложности алгоритмов сортировки при различных исходных данных, закрепление навыков алгоритмизации.

**Задание:**

1. Реализовать алгоритмы сортировки согласно варианту. Для проверки реализованных функций сортировки разработать функцию, проверяющую упорядоченность элементов в последовательности.
2. Разработать функции, позволяющие производить оценку временной сложности алгоритма T(n) (n – количество элементов сортируемой последовательности) и суммарного количества операций сравнения S(n), выполняемых в ходе его работы.
3. На основе функций формирования последовательностей, реализованных в ходе предыдущей лабораторной работы, и результатов выполнения пп. 1 и 2, разработать программу, с помощью которой по экспериментальным данным построить зависимости T(n) и S(n) для заданных алгоритмов сортировки по четырем последовательностям (упорядоченная, случайная, упорядоченная в обратном порядке и указанная в варианте). Размеры сортируемых последовательностей следует выбирать самостоятельно в соответствии с характеристиками исследуемого алгоритма (например, 5⋅10^3 , 10⋅10^3 , …, 50⋅10^3 ). Кроме того, необходимо оценить зависимости T(n) и S(n) для функции qsort из стандартной библиотеки C.
4. Составить отчет, в котором привести графики зависимостей T(n) и S(n), результаты анализа полученных экспериментальных данных и теоретических оценок сложности алгоритмов, сравнительную оценку реализованных алгоритмов (по сложности, устойчивости, естественности, требованиям к памяти), выводы по работе.

Вариант 9.

1. Согласно варианту, необходимо реализовать:
   1. Сортировку Шелла для последовательности S1: hi = 3\*hi-1+1
   2. Сортировку выбором
   3. Поразрядную сортировку подсчетом (LSD) для 1, 2, 4, 8 разрядов.
2. Измерение времени работы алгоритма сортировки T и количество операций сравнения S от размера сортируемой последовательности n.

Таблица 1.T(n) и S(n) для сортировки Шелла

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время сортировки S(n) (µs) | | | |
| Количество сравнений T(n) | | | |
| Количество  элементов n  (\*10^3) | Возрастающая | Случайная | Убывающая | Ступенчатая |
| 5 | 127 | 113 | 607 | 178 |
| 33364 | 38363 | 101455 | 40311 |
| 10 | 233 | 258 | 1400 | 382 |
| 75084 | 85083 | 239011 | 89301 |
| 15 | 336 | 394 | 2219 | 572 |
| 115084 | 130083 | 373638 | 136454 |
| 20 | 453 | 531 | 3130 | 852 |
| 155084 | 175083 | 537339 | 183209 |
| 25 | 706 | 690 | 4091 | 955 |
| 195084 | 220083 | 715678 | 230748 |
| 30 | 788 | 880 | 5822 | 1269 |
| 255243 | 285242 | 988220 | 297363 |
| 35 | 893 | 1022 | 5827 | 1471 |
| 300243 | 335242 | 1077373 | 350041 |
| 40 | 1037 | 1177 | 7279 | 1723 |
| 345243 | 385242 | 1277015 | 401498 |
| 45 | 1191 | 1338 | 8056 | 1827 |
| 390243 | 435242 | 1404792 | 453686 |
| 50 | 1329 | 1497 | 9032 | 2110 |
| 435243 | 485242 | 1622973 | 505445 |

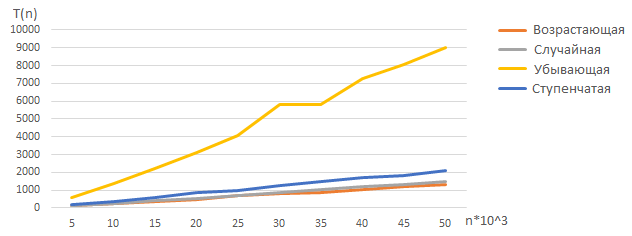


Рисунок 1.График T(n) для сортировки Шелла

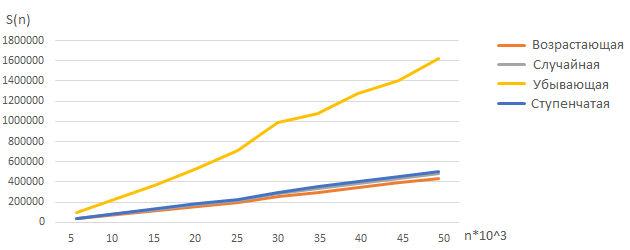


Рисунок 2.График S(n) для сортировки Шелла

Таблица 2.T(n) и S(n) для сортировки выбором

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время сортировки S(n) (µs) | | | |
| Количество сравнений T(n) | | | |
| Количество  элементов n  (\*10^3) | Возрастающая | Случайная | Убывающая | Ступенчатая |
| 5 | 43530 | 43530 | 44250 | 43148 |
| 12497500 | 12497500 | 12497500 | 12497500 |
| 10 | 175660 | 190646 | 176444 | 175808 |
| 49995000 | 49995000 | 49995000 | 49995000 |
| 15 | 404432 | 394899 | 399937 | 397393 |
| 112492500 | 112492500 | 112492500 | 112492500 |
| 20 | 703629 | 707854 | 717234 | 706760 |
| 199990000 | 199990000 | 199990000 | 199990000 |
| 25 | 1113956 | 1117779 | 1242359 | 1102668 |
| 312487500 | 312487500 | 312487500 | 312487500 |
| 30 | 1676897 | 1729480 | 1659705 | 1601084 |
| 449985000 | 449985000 | 449985000 | 449985000 |
| 35 | 2140591 | 2283669 | 2154644 | 2161333 |
| 612482500 | 612482500 | 612482500 | 612482500 |
| 40 | 2814863 | 2964313 | 2854022 | 2797836 |
| 799980000 | 799980000 | 799980000 | 799980000 |
| 45 | 3745011 | 3615451 | 3525206 | 3601422 |
| 1012477500 | 1012477500 | 1012477500 | 1012477500 |
| 50 | 4462305 | 4452957 | 4584839 | 4450908 |
| 1249975000 | 1249975000 | 1249975000 | 1249975000 |

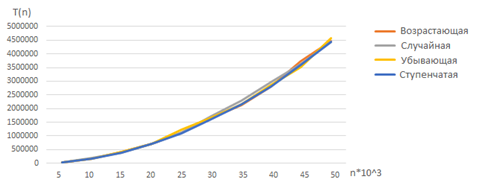


Рисунок 3.График T(n) для сортировки выбором

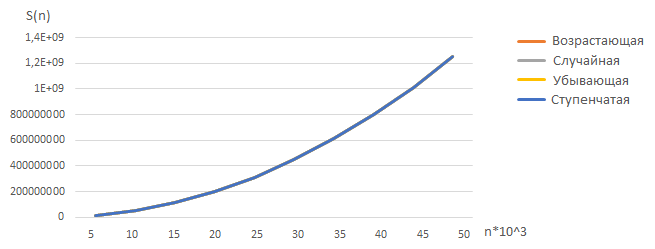


Рисунок 4.График S(n) для сортировки выбором

Ввиду особенностей реализации поразрядной сортировки, операции сравнения не используются

Таблица 3.T(n) для поразрядной сортировки подсчетом (1 разряд)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время сортировки S(n) (µs) | | | |
| Количество  элементов n  (\*10^3) | Возрастающая | Случайная | Убывающая | Ступенчатая |
| 5 | 1158 | 1129 | 1144 | 1105 |
| 10 | 2331 | 2285 | 2363 | 2269 |
| 15 | 3484 | 3517 | 3395 | 3330 |
| 20 | 4683 | 4664 | 4624 | 4498 |
| 25 | 5845 | 5854 | 5782 | 5884 |
| 30 | 7037 | 6985 | 6919 | 6778 |
| 35 | 8089 | 7851 | 8064 | 8081 |
| 40 | 9637 | 9329 | 9295 | 9011 |
| 45 | 10451 | 10386 | 10463 | 10496 |
| 50 | 11573 | 11783 | 11775 | 11566 |

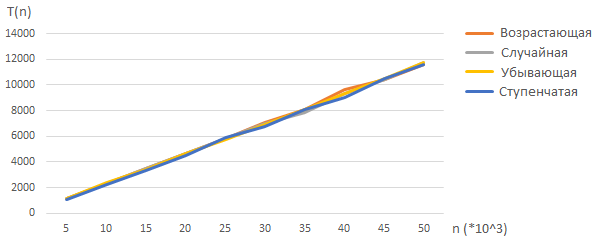


Рисунок 5.График T(n) для поразрядной сортировки подсчетом (1 разряд)

Таблица 4.T(n) для поразрядной сортировки подсчетом (2 разряда)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время сортировки S(n) (µs) | | | |
| Количество  элементов n  (\*10^3) | Возрастающая | Случайная | Убывающая | Ступенчатая |
| 5 | 614 | 568 | 566 | 552 |
| 10 | 1164 | 1166 | 1155 | 1113 |
| 15 | 1715 | 2073 | 1657 | 1789 |
| 20 | 2320 | 2326 | 2322 | 2232 |
| 25 | 2964 | 2942 | 2966 | 2917 |
| 30 | 3634 | 3536 | 3658 | 3395 |
| 35 | 4072 | 3958 | 4066 | 4092 |
| 40 | 4641 | 4601 | 4756 | 4545 |
| 45 | 5263 | 5228 | 5091 | 5250 |
| 50 | 5926 | 5896 | 5854 | 5784 |

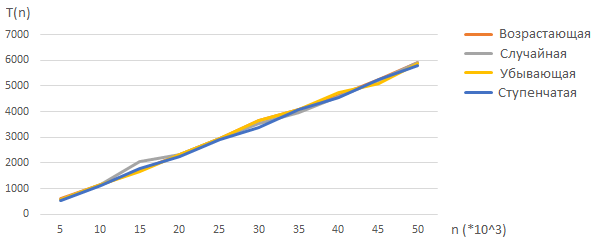


Рисунок 6.График T(n) для поразрядной сортировки подсчетом (2 разряда)

Таблица 5.T(n) для поразрядной сортировки подсчетом (4 разряда)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время сортировки S(n) (µs) | | | |
| Количество  элементов n  (\*10^3) | Возрастающая | Случайная | Убывающая | Ступенчатая |
| 5 | 282 | 289 | 288 | 281 |
| 10 | 588 | 607 | 575 | 658 |
| 15 | 879 | 1070 | 851 | 850 |
| 20 | 1205 | 1192 | 1186 | 1140 |
| 25 | 1508 | 1522 | 1536 | 1518 |
| 30 | 1817 | 2078 | 1854 | 1734 |
| 35 | 2088 | 2114 | 2082 | 2081 |
| 40 | 2370 | 2347 | 2478 | 2319 |
| 45 | 2729 | 2682 | 2595 | 2723 |
| 50 | 3067 | 3017 | 3009 | 3001 |

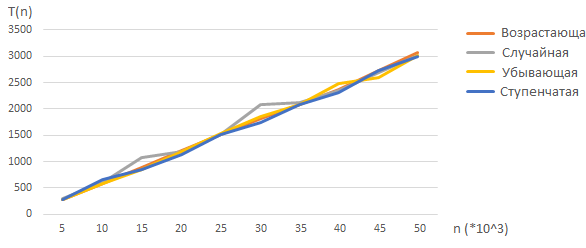


Рисунок 7.График T(n) для поразрядной сортировки подсчетом (4 разрядов)

Таблица 6.T(n) для поразрядной сортировки подсчетом (8 разрядов)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Время сортировки S(n) (µs) | | | |
| Количество  элементов n  (\*10^3) | Возрастающая | Случайная | Убывающая | Ступенчатая |
| 5 | 149 | 153 | 154 | 149 |
| 10 | 303 | 303 | 302 | 294 |
| 15 | 464 | 596 | 440 | 440 |
| 20 | 631 | 622 | 634 | 618 |
| 25 | 796 | 790 | 777 | 786 |
| 30 | 963 | 1338 | 951 | 918 |
| 35 | 1115 | 1080 | 1103 | 1098 |
| 40 | 1306 | 1234 | 1281 | 1249 |
| 45 | 1455 | 1413 | 1376 | 1449 |
| 50 | 1586 | 1600 | 1596 | 1575 |

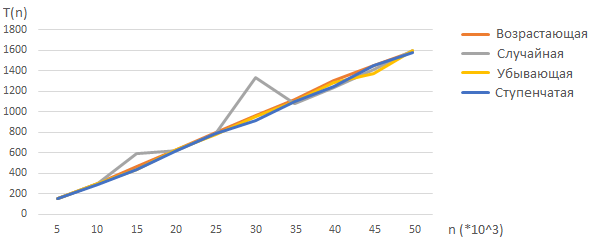


Рисунок 8.График T(n) для поразрядной сортировки подсчетом (8 разрядов)

3.Анализ полученных данных.

3.1 Сортировка Шелла имеет асимптотическую сложность в среднем O(n\*log(n)). Она показывает себя лучше всего на отсортированной последовательности, хуже всего – на убывающей. Никаких дополнительный затрат памяти не требуется.

3.2 Сортировка выбором имеет асимптотическую сложность в среднем O(n^2). Ее скорость не зависит от типа обрабатываемой последовательности. Никаких дополнительный затрат памяти не требуется.

3.3 Поразрядная сортировка подсчетом k двоичными разрядами имеет асимптотическую сложность в среднем O(w \* n), где w – длинна ключа в байтах, деленная на k. Скорость не зависит от типа последовательности. Затраты памяти: w+n;

**Вывод:** были разработаны функции для сортировки последовательностей целых чисел методами Сортировки Шелла, Поразрядной сортировки LSD (для 1, 2, 4, 8 разрядов) и Сортировки выбором; измерена сложность работы алгоритмов; наиболее эффективной является поразрядная сортировка со сложностью O(n+w); ее эффективность прямо пропорциональна используемой памяти; наименее эффективной является сортировка выбором со сложностью O(n^2).

Листинг 1. arrgen.h

#ifndef ARRGEN\_H

#define ARRGEN\_H

#include <iostream>

#include <random>

#include <cmath>

#include <fstream>

namespace ArrGen

{

template <typename T>

void IncreasingSequence(T \*array, size\_t len, T min, T max);

template <typename T>

void DecreasingSequence(T \*array, size\_t len, T min, T max);

template <typename T>

void RandomSequence(T \*array, size\_t len, T min, T max);

template <typename T>

void SawtoothSequence(T \*array, size\_t len, T start, T end, size\_t period, size\_t offset);

template <typename T>

void SinSequence(T \*array, size\_t len, T start, T end, double period = 2 \* M\_2\_PI, T offset = 0);

template <typename T>

void StepSequence(T \*array, size\_t len, T start, T end, size\_t period = 10);

template <typename T>

void QuasiOrderedSequence(T \*array, size\_t len, T start, T end, T spread);

template <typename T>

void SawtoothSequence(T \*array, size\_t len, T start, T end);

template <typename T>

void SinSequence(T \*array, size\_t len, T start, T end);

template <typename T>

void StepSequence(T \*array, size\_t len, T start, T end);

template <typename T>

void QuasiOrderedSequence(T \*array, size\_t len, T start, T end);

extern void (\*GenFunArrInt[])(int \*, size\_t, int, int);

extern void (\*GenFunarrDouble[])(double \*, size\_t, double, double);

extern std::string GenFunNames[];

}

#endif

Листинг 2. arrgen.cpp

#include "arrgen.h"

#include <string>

namespace ArrGen

{

template <typename T>

void IncreasingSequence(T \* array, size\_t len, T min, T max)

{

if (nullptr == array || 0 == len)

return;

if (min > max)

{

T temp = min;

min = max;

max = temp;

}

double step = 1.0 \* (max - min) / len;

array[0] = (T)min;

double prev = (T)min;

for (size\_t i = 1; i < len; i++)

{

array[i] = prev + step;

prev += step;

}

}

template <typename T>

void DecreasingSequence(T \* array, size\_t len, T min, T max)

{

if (nullptr == array || 0 == len)

return;

if (min > max)

{

T temp = min;

min = max;

max = temp;

}

double step = 1.0 \* (max - min) / len;

array[0] = max;

double prev = (T)min;

for (size\_t i = 1; i < len; i++)

{

array[i] = prev + step;

prev += step;

}

}

template <typename T>

void RandomSequence(T \* array, size\_t len, T min, T max)

{

if (nullptr == array || 0 == len)

return;

if (min > max)

{

T temp = min;

min = max;

max = temp;

}

std::random\_device random\_device;

std::mt19937 generator(random\_device());

std::uniform\_real\_distribution distribution((double)min, (double)max);

for (size\_t i = 0; i < len; i++)

{

array[i] = distribution(generator);

}

}

template <typename T>

void SawtoothSequence(T \* array, size\_t len, T start, T end, size\_t period, size\_t offset)

{

if (nullptr == array || 0 == len)

return;

double local\_step = 1.0 \* (end - start) / period;

for (size\_t i = 0; i < len; i++)

{

array[i] = start + local\_step \* ((i + offset) % period);

}

}

template <typename T>

void SinSequence(T \* array, size\_t len, T start, T end, double period, T offset)

{

if (nullptr == array || 0 == len)

return;

for (size\_t i = 0; i < len; i++)

{

array[i] = sin(i \* M\_2\_PI \* len / period / period + offset) \* (end - start) / 2 + (end + start) / 2;

}

}

template <typename T>

void StepSequence(T \* array, size\_t len, T start, T end, size\_t period)

{

if (nullptr == array || 0 == len || 0 == period || len < period)

return;

std::random\_device random\_device;

std::mt19937 generator(random\_device());

std::uniform\_real\_distribution distribution((double)0, (double)(end - start) / (len / period));

for (size\_t i = 0; i < len; i++)

{

double ran = distribution(generator);

array[i] = (i / period) \* (end - start) / (len / period) + ran;

}

}

template <typename T>

void QuasiOrderedSequence(T \* array, size\_t len, T start, T end, T spread)

{

if (nullptr == array || 0 == len)

return;

double step = (end - start) / len;

std::random\_device random\_device;

std::mt19937 generator(random\_device());

T max = start > end ? start : end;

T min = start < end ? start : end;

double direct = start;

for (size\_t i = 0; i < len; i++, direct += step)

{

direct = start + 1.0 \* i \* (end - start) / len;

double up\_limit = max > direct + spread ? direct + spread : max;

double low\_limit = min < direct - spread ? direct - spread : min;

std::uniform\_real\_distribution distribution(low\_limit, up\_limit);

double ran = distribution(generator);

array[i] = ran;

}

}

template <typename T>

void SawtoothSequence(T \* array, size\_t len, T start, T end)

{

SawtoothSequence(array, len, start, end, 10, 0);

}

template <typename T>

void SinSequence(T \* array, size\_t len, T start, T end)

{

SinSequence<T>(array, len, start, end, 10 \* M\_2\_PI, 0);

}

template <typename T>

void StepSequence(T \* array, size\_t len, T start, T end)

{

StepSequence<T>(array, len, start, end, 10);

}

template <typename T>

void QuasiOrderedSequence(T \* array, size\_t len, T start, T end)

{

QuasiOrderedSequence<T>(array, len, start, end, 10);

}

}

Листинг 3. lab02.cpp

#include <iostream>

#include <cstdlib>

#include "arrgen.h"

#include <cstdio>

#include <chrono>

#include <fstream>

template <typename T>

bool IsSorted(int (\*Compare)(T, T), T \*array, size\_t n)

{

for (size\_t i = 0; i < n - 1; i++)

{

if (Compare(array[i], array[i + 1]) < 0)

{

return false;

}

}

return true;

}

template <typename T>

void PrintArray(T \*array, size\_t n)

{

for (size\_t i = 0; i < n; i++)

std::cout << array[i] << " ";

std::cout << std::endl;

}

/\*\*

\* h(i) = 3\*h(i-1)-1

\*/

template <typename T>

size\_t ShellSort(int (\*Compare)(T, T), T \*array, size\_t n)

{

size\_t CompTimes = 0;

size\_t h;

for (h = 1; h <= n / 9; h = 3 \* h + 1)

;

for (; h > 0; h /= 3)

{

for (size\_t i = h; i < n; i++)

{

T temp = array[i];

size\_t j = i;

for (; j >= h && Compare(array[j-h],temp) < 0; j -= h)

{

CompTimes++;

array[j] = array[j - h];

}

array[j] = temp;

CompTimes++;

}

}

return CompTimes;

}

void PrintBin(int \*array, size\_t n)

{

for (size\_t i = 0; i < n; i++)

{

for (int j = sizeof(int) \* 8 - 1; j >= 0; j--)

std::cout << ((array[i] & (1 << j)) ? 1 : 0);

std::cout << " ";

}

std::cout << std::endl;

}

void PrintBin(unsigned int \*array, size\_t n)

{

for (size\_t i = 0; i < n; i++)

{

for (int j = sizeof(unsigned int) \* 8 - 1; j >= 0; j--)

std::cout << ((array[i] & (1 << j)) ? 1 : 0);

std::cout << " ";

}

std::cout << std::endl;

}

void RadixSortLSD1(unsigned \*array, size\_t n, int power)

{

if (power < 0)

return;

int bits = 1 << power;

int counter\_size = (1 << bits) + 1;

unsigned int counter[counter\_size];

unsigned int mask;

int \*buf\_array = (int \*)malloc(n \* sizeof(int));

int hight;

int max\_hight = sizeof(int) \* 8 / bits - 1;

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

mask |= array[i];

for (hight = max\_hight; hight >= 0; hight--)

if (mask & ((counter\_size - 2) << (hight \* bits)))

break;

for (unsigned int b = 0; b <= hight; b++)

{

mask = (counter\_size - 2) << (b \* bits);

for (int i = 0; i < counter\_size; i++)

counter[i] = 0;

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

counter[((array[i] & mask) >> (b \* bits)) + 1]++;

for (int i = 1; i < counter\_size; i++)

counter[i] += counter[i - 1];

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

buf\_array[counter[((array[i] & mask) >> (b \* bits))]++] = array[i];

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

array[i] = buf\_array[i];

}

free(buf\_array);

}

template <typename T>

size\_t SelectionSort(int (\*Compare)(T, T), T \*array, size\_t n)

{

size\_t CompTimes = 0;

for (size\_t i = 0; i < n - 1; i++)

{

size\_t min\_index = i;

for (size\_t j = i + 1; j < n; j++)

{

CompTimes++;

if (Compare(array[min\_index], array[j]) < 0)

{

min\_index = j;

}

}

if (min\_index != i)

{

T temp = array[i];

array[i] = array[min\_index];

array[min\_index] = temp;

}

}

return CompTimes;

}

int CompFunc(int a, int b)

{

return b - a;

}

int CompFunc(unsigned int a, unsigned int b)

{

if (a > b)

return -1;

if (a == b)

return 0;

return 1;

}

void (\*GenFunArr[])(unsigned \*, size\_t, unsigned, unsigned) = {ArrGen::IncreasingSequence, ArrGen::DecreasingSequence, ArrGen::RandomSequence, ArrGen::StepSequence};

std::string GenFunNames[]{"IncreasingSequence", "DecreasingSequence", "RandomSequence", "StepSequence"};

using namespace std;

using namespace std::chrono;

int main()

{

ofstream outstream("out2.txt");

outstream << "Sort Name | Sequence | Size | Comparations | Time (mics) | Status\n";

auto begin = steady\_clock::now();

auto end = steady\_clock::now();

size\_t count;

unsigned \*arr;

int c = pow(10,3);

for (int i = 5; i <= 50; i += 5)

{

unsigned \*arr = (unsigned \*)malloc(sizeof(unsigned) \* i \* pow(10, 3));

for (int j = 0; j < 4; j++)

{

arr = (unsigned \*)malloc(sizeof(unsigned) \* i \* c);

GenFunArr[j](arr, i \* pow(10, 3), 0, 4294967295);

begin = steady\_clock::now();

count = ShellSort(CompFunc, arr, i \* c);

end = steady\_clock::now();

outstream << "ShellSort" << " " << GenFunNames[j] << " " << i << "\*10^3" << " " << count << " " << duration\_cast<microseconds>(end - begin).count() << "mics" << " " << (IsSorted(CompFunc, arr, i \* c) ? "SUCCSES" : "FAIL") << endl;

free(arr);

arr = (unsigned \*)malloc(sizeof(unsigned) \* i \* c);

GenFunArr[j](arr, i \* pow(10, 3), 0, 4294967295);

begin = steady\_clock::now();

count = SelectionSort(CompFunc, arr, i \* c);

end = steady\_clock::now();

outstream << "SelectionSort" << " " << GenFunNames[j] << " " << i << "\*10^3" << " " << count << " " << duration\_cast<microseconds>(end - begin).count() << "mics" << " " << (IsSorted(CompFunc, arr, i \* c) ? "SUCCSES" : "FAIL") << endl;

free(arr);

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

{

arr = (unsigned \*)malloc(sizeof(unsigned) \* i \* c);

GenFunArr[j](arr, i \* pow(10, 3), 0, 4294967295);

begin = steady\_clock::now();

RadixSortLSD1(arr, i \* c, k);

end = steady\_clock::now();

count = 0;

outstream << "RadixSort[" << (1 << k) << "]" << " " << GenFunNames[j] << " " << i << "\*10^3" << " " << count << " " << duration\_cast<microseconds>(end - begin).count() << "mics" << " " << (IsSorted(CompFunc, arr, i \* c) ? "SUCCSES" : "FAIL") << endl;

free(arr);

}

}

}

outstream.close();

}

#include "arrgen.cpp"