Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Выполнил студент группы КС-30 Лобачев Дмитрий Сергеевич

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/DSLobachev\_30/blob/main/Algorithms/Laba1/Laba1.cpp

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 20.02.2023

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать метод сортировки пузырьком. Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), при этом:

* в каждом тесте необходимо по 20 раз генерировать вектор, состоящий из N элементов;
* каждый элемент массива заполняется случайным числом с плавающей запятой от -1 до 1;
* каждый массив после генерации необходимо отсортировать и замерить время, требуемое на сортировку;
* результат замера для каждой попытки каждого теста записать в файл общий файл;

По окончанию всех тестов необходимо нанести все точки, полученные в результате замеров времени на график, где на ось абсцисс(Х) нанести N, а на ось ординат(Y) нанести значения времени на сортировку. По полученным точкам построить график лучшего (минимальное время для каждого N), худшего (максимальное время для каждого N) и среднего (среднее время для каждого N) случая.

# Описание метода/модели.

Сортировка пузырьком является одной из самых простых сортировок, которая приходит на ум практически каждому программисту когда нужно что-то отсортировать. В ее основе лежит идея постоянного пробегания по массиву и постепенному перемещению к концу наибольших элементов неотсортированной части.

Преимущества данной сортировки заключаются в том, что для её реализации не требуется много времени и сил, поскольку данный метод является одним из самых простых видов сортировок. Главный недостаток данной сортировки заключается в наименьшей скорости выполнения, поскольку скорость данной сортировки пропорциональна квадрату количества элементов.

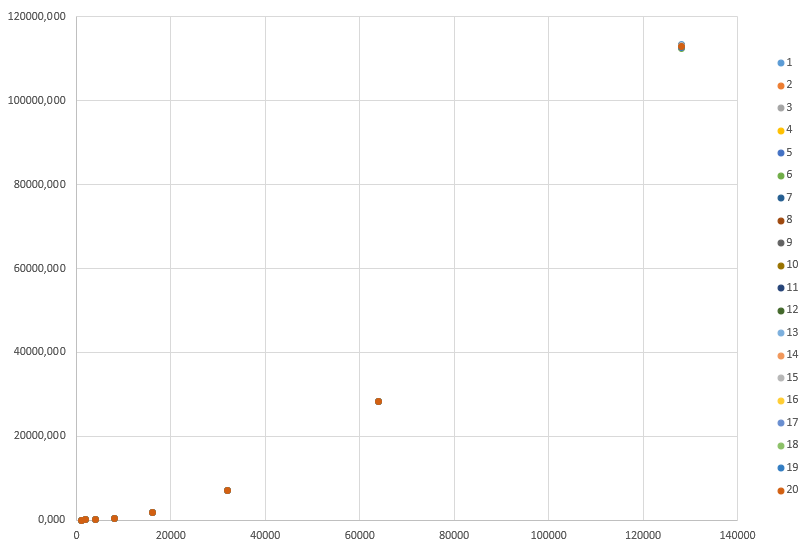
# Выполнение задачи.

Для реализации данного метода сортировки был использован язык программирования C++.

Сортировка выполняется по мере прохождения элементов в массиве от начала до конца. Берутся два соседних элемента и сравниваются по условию (сортировка по возрастанию или убыванию) и если данное условие выполняется, то происходит перестановка двух элементов. Таким образом по мере прохождения по массиву соседние элементы перестанавливаются и с каждым новым проходом начинает выстраиваться правильный порядок.

В рамках задачи было задано провести 20 тестов на массивах разной длины, а также записать время, которое ушло на сортировку. Для записи времени в коде был использован таймер. Значения длин массивов были заданы как 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000.

После реализации программы и анализа полученных данных, был построен график со всеми значениями времени для каждого количества элементов:



После этого было рассчитано среднее значение времени для каждого количества элементов и построен график лучшего, среднего и худшего случая.

Для более лучшего понимания также был построен график по первым трем точкам.

Таким образом был получен график времени, на котором показано, за какое максимальное, среднее и минимальное время будет выполнен алгоритм.

В дополнение к этому также были построены графики худшего случая, и график O(c \* g(N)), где g(N) соответствует асимптотической сложности рассматриваемого метода сортировки. Параметр C был подобран так, что начиная с N ~ 1000 график асимптотической сложности возрастал быстрее чем полученное худшее время, но при этом был различим на графике. За данный параметр было взято значение 0,000015.

Таким образом был построен графики худшего случая и асимптотической функции и такой же график для первых трёх элементов:

**Код реализации программы:**

#include <iostream>

#include <stdlib.h>

#include <chrono>

#include <random>

#include <fstream>

const int TESTS = 20; //Количество тестов

using namespace std;

void fileCreator(); //Создание и очистка файла

void fileWriter(int i, int N, double time); //Запись результата в файл

void bubbleSort(double\* arr, int N); //Сортировка пузырьком

void millisecundomer(double\* arr, int N, int i); //Подсчет времени на сортировку

int main()

{

int N[] = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 }; //Массив длин массива для сортировки

fileCreator(); //Создание файла

for (int j = 0; j < size(N); j++) {

double\* arr = new double[N[j]]; //Выделение памяти под массив элементов

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

//Генерация случайных чисел в [-1; 1]

mt19937 engine(time(0));

uniform\_real\_distribution<double> gen(-1.0, 1.0);

for (int el = 0; el < N[j]; el++) {

arr[el] = gen(engine);

}

millisecundomer(arr, N[j], i); //Замер времени и сортировка

}

delete[] arr; //Удаление массива

}

}

//Сортировка пузырьком

/\*

\* @param arr - Массив

\* @param flag - Флаг требования сортировки

\* @param N - Кол-во элементов

\*/

void bubbleSort(double\* arr, int N) {

bool flag = true;

while (flag) {

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

flag = false;

for (int j = 0; j < N - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

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

flag = true;

}

}

}

}

}

//Создание и очистка файла

void fileCreator() {

ofstream outdatafile("D:\\Users\\DiaZo\\Desktop\\bubblesort.txt");

}

//Запись результата в файл

void fileWriter(int i, int N, double time) {

ofstream outdatafile("D:\\Users\\DiaZo\\Desktop\\bubblesort.txt", ios::app); //Открываем файл для записи

if (outdatafile.is\_open())

{

outdatafile << "Тест " << i << " | Число элементов: " << N << " | Время: " << time << " миллисекунд" << endl;

}

}

//Подсчет времени на сортировку

/\*

\* @param arr - Массив

\* @param i - Номер теста

\* @param N - Кол-во элементов

\*/

void millisecundomer(double\* arr, int N, int i) {

chrono::high\_resolution\_clock::time\_point start = chrono::high\_resolution\_clock::now(); //Время начала сортировки

bubbleSort(arr, N); //Сама сортировка

chrono::high\_resolution\_clock::time\_point end = chrono::high\_resolution\_clock::now(); //Время на конец сортировки

chrono::duration<double, milli> milli\_diff = end - start; //Итоговое время сортировки в миллисекундах

fileWriter((i + 1), N, milli\_diff.count()); //Запись в файл

}

# Заключение.

В заключении можно отметить, что метод сортировки пузырьком является одним из самых медленных методов сортировки, однако он имеет очень простую реализацию. Данный метод можно использовать на небольших массивах без потери большого количество времени, однако, если требуется отсортировать большой массив, лучше прибегнуть к другим методам сортировки.