

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Российский химико-технологический университет имени Д.И.
Менделеева»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Выполнил студент группы КС-36: Золотухин А.А.

Ссылка на репозиторий: [https://github.com/
MUCTR-IKT-CPP/
ZolotukhinAA_36_ALG](https://github.com/MUCTR-IKT-CPP/ZolotukhinAA_36_ALG)

Принял: Крашенников Роман Сергеевич

Дата сдачи: 17.02.2025

Москва
2025

Оглавление

Описание задачи	1
Описание метода/модели	2
Выполнение задачи	3
Выводы	8

Описание задачи

В лабораторной работе предлагается изучить способ анализа алгоритма, связанный со временем. Рассмотреть для выбранного алгоритма сортировки наилучшее, наихудшее и среднее время и соотнести его с известным для алгоритма показателем эффективности O -большое.

Допускается реализация задания на любом языке программирования, кроме лиспоподобных. Преподаватель может не знать конкретного языка реализации, поэтому вы должны быть способны объяснить алгоритм и нарисовать его без демонстрации непосредственно вашего кода.

Задание:

- Реализовать метод сортировки: **Сортировка вставками**;
- Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров. За один расчёт выполняются следующие операции:
 - Генерируется массив случайных значений;
 - Запоминается время начала расчета алгоритма сортировки;
 - Выполняется алгоритм сортировки
 - * Во время выполнения измерить количество повторных проходов по массиву.
 - * Во время выполнения измерить количество выполнения операций обмена значений.
 - Вычисляется время, затраченное на сортировку: текущее время - время начала;
 - Сохраняется время для одной попытки. После этого расчёт повторяется до окончания серии.
 - * Алгоритм вычисляется 8 сериями по 20 раз за серию;
 - * Алгоритм в каждой серии вычисляется для массива размером M . (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000);
 - * Массив заполняется значениями чисел с плавающей точкой в интервале от -1 до 1;
 - * Для серии запоминаются все времена, которые были замерены.
- По полученным данным времени построить графики зависимости времени от числа элементов в массиве:
 - Совмещенный график наихудшего времени выполнения сортировки и сложности алгоритма, указанной в нотации O большое;

Для построения графика вычисляется O большое для каждого размера массива. При этом при вычислении функции $O(c * g(N))$ подбирается такая константа c , чтобы при значении >1000 график $O(N)$ был выше графика наихудшего случая, но второй график на его фоне не превращался в прямую линию.

- Совмещенный график среднего, наихудшего и наилучшего времени исполнения;
 - График среднего количества обмена значений;
 - График повторных обходов массива.
- По результатам расчётов оформляется отчёт по предоставленной форме, в отчете:
 - Приводится описание алгоритма;
 - Приводится описание выполнения задачи (описание кода и специфических элементов реализации);
 - Приводятся выводы (Графики и их анализ).

Описание метода/модели

Сортировка вставками - это простой алгоритм сортировки, который работает путём итеративной вставки каждого элемента несортированного списка в его правильное положение в отсортированной части списка. Это похоже на сортировку игровых карт в ваших руках. Вы разделяете карты на две группы: отсортированные карты и несортированные карты. Затем вы выбираете карточку из несортированной группы и помещаете её в нужное место в отсортированной группе.

Ход алгоритма:

1. Начинаем со второго элемента массива, поскольку предполагается, что первый элемент в массиве должен быть отсортирован;
2. Сравниваем второй элемент с первым и проверяем, не меньше ли второй элемент, затем меняем их местами;
3. Переходим к третьему элементу, сравниваем его с первыми двумя элементами и устанавливаем в правильное положение;
4. Повторяем до тех пор, пока не будет отсортирован весь массив.

Анализ сложности сортировки вставками:

- **Лучший вариант:** $O(N)$, если список уже отсортирован;
- **Средний вариант:** $O(N^2)$, если список упорядочен случайным образом;

- **Наихудший вариант:** $O(N^2)$, если список находится в обратном порядке, где N - количество элементов в списке.

Algorithm 1 Реализация алгоритма сортировки вставками

```
 $i \leftarrow 2$ 
while  $i \leq n - 1$  do
     $j \leftarrow i - 1$ 
    while  $j \geq 0$  AND  $A[j] > A[j + 1]$  do
         $swap(A[j], A[j + 1])$ 
         $j \leftarrow j - 1$ 
```

Преимущества:

- Эффективно справляется с небольшими и почти отсортированными массивами;
- Экономит место;
- Простой и легкорееализуемый.

Недостатки:

- Неэффективно справляется с большими массивами.

Выполнение задачи

Алгоритм сортировки вставками реализован на языке *C++*. Построение графиков проводить с помощью программы *GNUplot*.

"*main*" функция работает с циклом, в ходе которого производится расчёт минимального, максимального и среднего времени на сортировку массива размером M . Каждая серия просчитывается по 20 раз. В итоге получаются данные, выведенные в определенные файлы, с помощью которых впоследствии строятся графики.

```
1  int main() {
2      std::ofstream worst_and_complexity(Constants::folder + "worst_and_complexity.
dat");
3      std::ofstream average_best_worst(Constants::folder + "average_best_worst.dat");
4      std::ofstream average_swaps(Constants::folder + "average_swaps.dat");
5      std::ofstream average_passes(Constants::folder + "average_passes.dat");
6      if (!worst_and_complexity.is_open() ||
7          !average_best_worst.is_open() ||
8          !average_swaps.is_open() ||
9          !average_passes.is_open()) {
10         std::cerr << "Error of opening the file!" << std::endl;
11         return 1;
12     }
13
14     for (int episode = 0; episode < Constants::M; episode++) {
15         int size = Constants::sizes[episode];
16         double* array = new double[size];
```

```

17     double the_worst_time = 0.0;
18     double the_best_time = std::numeric_limits<double>::max();
19     double total_time = 0.0;
20     int total_swaps = 0;
21     int total_passes = 0;
22
23
24     for (int attempt = 0; attempt < Constants::amount_of_attempts; attempt++) {
25         int amount_of_repeated_passes = 0;
26         int amount_of_swaps = 0;
27
28         generationArray(array, size);
29
30         std::chrono::high_resolution_clock::time_point start = std::chrono::
high_resolution_clock::now();
31
32         insertionSort(array, size, amount_of_repeated_passes, amount_of_swaps);
33
34         std::chrono::high_resolution_clock::time_point end = std::chrono::
high_resolution_clock::now();
35         std::chrono::duration<double, std::milli> milli_diff = end - start;
36
37         double time_taken = milli_diff.count();
38         if (time_taken > the_worst_time)
39             the_worst_time = time_taken;
40         if (time_taken < the_best_time)
41             the_best_time = time_taken;
42
43         total_time += time_taken;
44         total_swaps += amount_of_swaps;
45         total_passes += amount_of_repeated_passes;
46     }
47

```

"generationArray" функция принимает два аргумента: array - массив, size - размер массива. Формирует массив размера size, который заполняется случайными числами с плавающей точкой от -1 до 1.

```

1     void generationArray(double array[], int size) {
2         std::random_device rd;
3         std::mt19937 engine(rd());
4         std::uniform_real_distribution<double> gen(-1.0, 1.0);
5
6         for (int i = 0; i < size; i++)
7             array[i] = gen(engine);
8     }
9

```

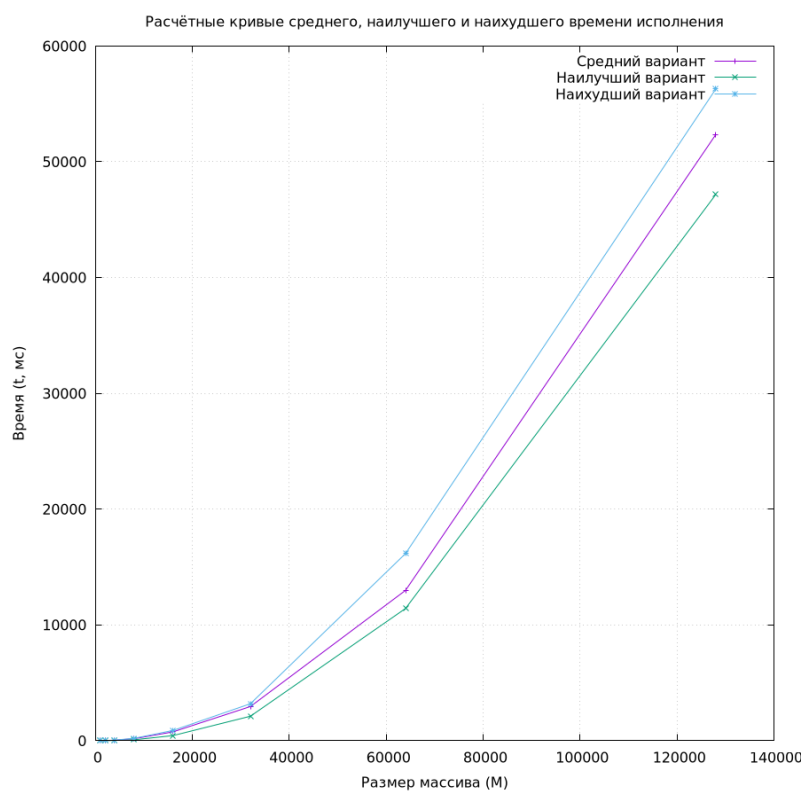
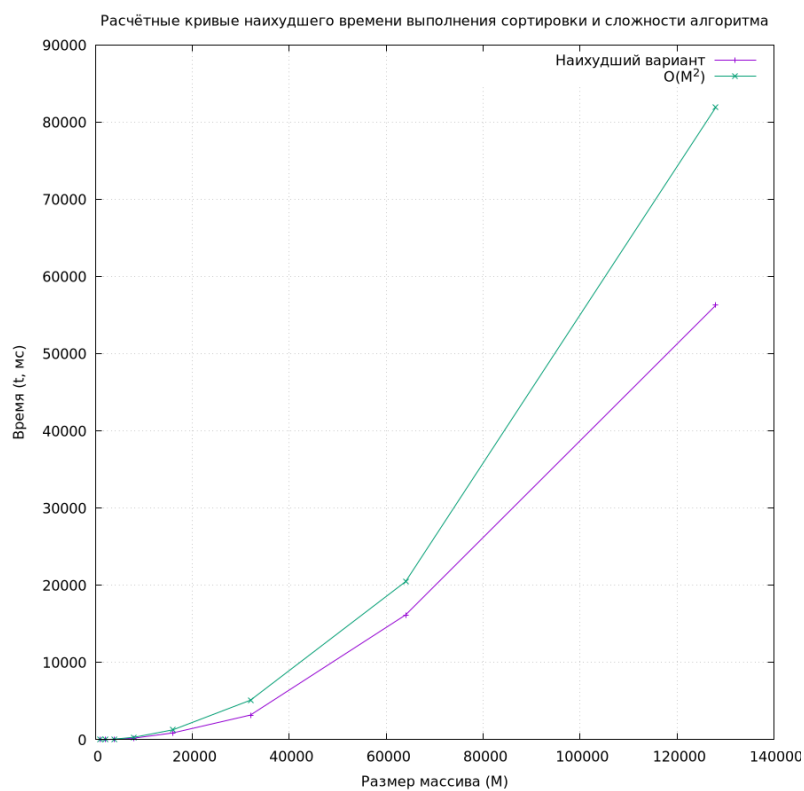
"insertionSort" функция принимает два аргумента: array - заполненный случайными числами массив, size - размер массива, repeated_passes - количество повторных обходов, swaps - количество сделанных обменов значений. Сортирует массив методом вставок и просчитывает количество повторных обходов и обменов значений.

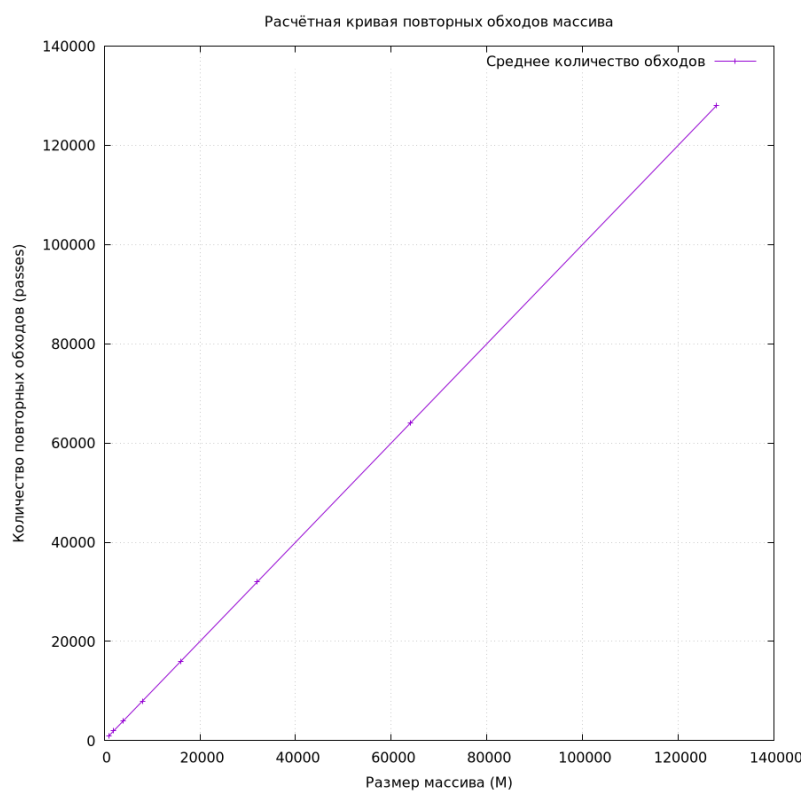
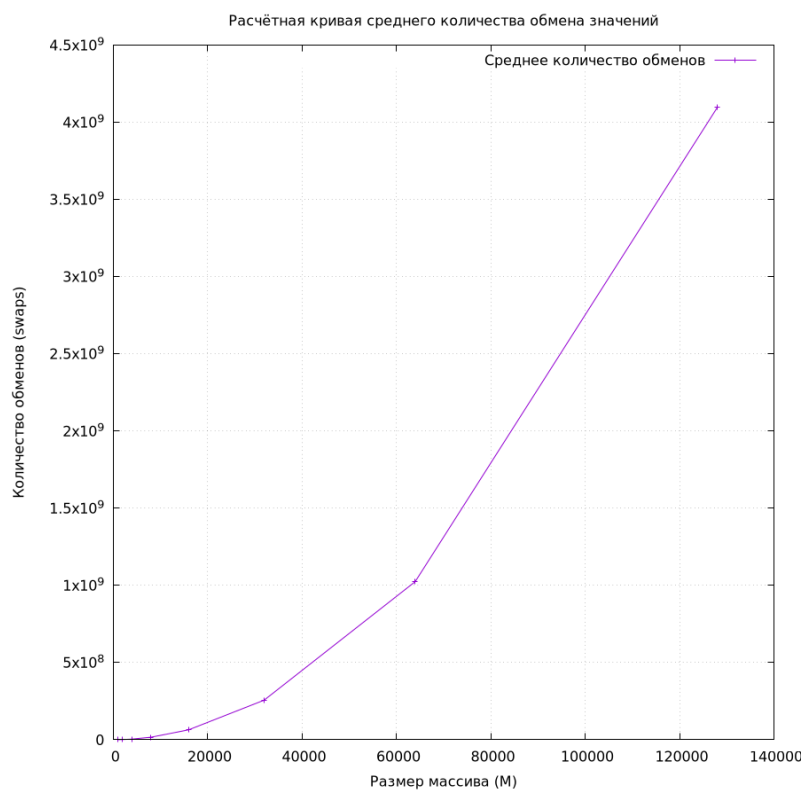
```

1     void insertionSort(double array[], int size, int& repeated_passes, int& swaps) {
2         for (int i = 1; i < size; i++) {
3             repeated_passes++;
4
5             int j = i - 1;
6

```

```
6     while (j > -1 && array[j] > array[j + 1]) {
7         double temp = array[j];
8         array[j] = array[j + 1];
9         array[j + 1] = temp;
10
11         swaps++;
12
13         j--;
14     }
15 }
16 }
17 }
```





Выводы

Сортировка вставками - простой алгоритм, который имеет временную сложность, равную квадратичной ($O(N^2)$). Это означает, что время выполнения алгоритма растёт пропорционально квадрату количества элементов в массиве. Чем больше данных нужно обработать, тем заметнее становится этот эффект.

Для сортировки массива, который состоит из 64000 элементов, в среднем требуется около 15 секунд, а для сортировки массива, состоящего из 128000 элементов, в среднем требуется около 55 секунд. При 20 таких сериях занимает много времени.

Данная сортировка наиболее эффективна, когда массив уже частично или полностью упорядочен. В таком варианте количество операций значительно сокращается, и алгоритм работает быстрее. Напротив, если массив отсортирован в обратном порядке, то это наихудший сценарий для данного алгоритма, т.к. количество сравнений и обменов элементов достигает максимума.