ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1
Сортировка

Оглавление

Описание задачи.

Необходимо изучить и реализовать метод сортировки вставками на серии тестов для всех N значений из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), где в каждом тесте необходимо по 20 раз генерировать вектор из N элементов.

По окончанию всех тестов необходимо нанести все точки, полученные в результате замеров времени на график. 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.

Задача заключается в следующем: есть часть массива, которая уже отсортирована, и требуется вставить остальные элементы массива в отсортированную часть, сохранив при этом упорядоченность. Для этого на каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор, пока весь набор входных данных не будет отсортирован. Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Так как в процессе работы алгоритма могут меняться местами только соседние элементы, каждый обмен уменьшает число инверсий на единицу. Следовательно, количество обменов равно количеству инверсий в исходном массиве вне зависимости от реализации сортировки. Максимальное количество инверсий содержится в массиве, элементы которого отсортированы по невозрастанию. Число инверсий в таком массиве $n(n-1)/2$.

Алгоритм работает за $O(n+k)$, где k — число обменов элементов входного массива, равное числу инверсий. В среднем и в худшем случае — за $O(n^2)$. Минимальные оценки встречаются в случае уже упорядоченной исходной последовательности элементов, наихудшие — когда они расположены в обратном порядке.

Преимущества:

- эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим;
- эффективен на наборах данных, которые уже частично отсортированы;
- это устойчивый алгоритм сортировки (не меняет порядок элементов, которые уже отсортированы);
- может сортировать список по мере его получения;

Недостатки

- Минусом же является высокая сложность алгоритма: $O(n^2)$.

Выполнение задачи.

Программа для получения тестовых данных была написана на C++. Графики нарисованы в Excel.

Код:

```
#include <iostream>
#include <string>
#include <ctime>
#include <vector>
#include <chrono>
#include <random>
#include <utility>
#include <fstream>
using namespace std;
const int VALUES_COUNT = 8;
int TESTS_LIMIT = 20;
const int TEST_VALUES[VALUES_COUNT]{ 1000, 2000, 4000, 8000, 16000, 32000,64000,128000 };
/*
 * Сортировка массива методом вставок
 * @param vector_values вектор
*/
void insertionSort(vector<double>& vector_values);
/*
 * Программа, тестирующая сортировку вставками
*/
int main()
{
    setlocale(LC_ALL, "Russian");
    srand(time(0));
    mt19937 engine(time(0));
    uniform_real_distribution<double> gen(-1.0, 1.0);
    std::ofstream out;
    out.open("TestingLog.txt");
    if (out.is_open())
    {
        for (int i = 0; i < VALUES_COUNT; i++) {
            out << "N = " << TEST_VALUES[i] << endl;
            for (int j = 0; j < TESTS_LIMIT; j++) {
                vector<double> vector_values(TEST_VALUES[i]);
                /*cout << "\nДо Сортировки: " << endl;*/
                for (int k = 0; k < TEST_VALUES[i]; k++) {
                    vector_values[k] = gen(engine);
                }
               /*cout << "\nПосле Сортировки: " << endl;*/
                chrono::high_resolution_clock::time_point start =
chrono::high_resolution_clock::now();
                insertionSort(vector_values);
                chrono::high_resolution_clock::time_point end =
chrono::high_resolution_clock::now();
                chrono::duration<double, milli> milli_diff = end - start;
                out <<milli_diff.count()<< endl;
            }
        }
    }
    out.close();
}

void insertionSort(vector<double>& vector_values) {
    for (int i = 1; i < vector_values.size(); i++) {
        for(int j = i; j > 0 && vector_values[j - 1] > vector_values[j]; j--) {
            std::swap(vector_values[j-1], vector_values[j]);
        }
    }
  }
```

Реализация:

1)Количество значений записано в массиве `TEST_VALUES, а количество повторений тестов в TESTS_LIMIT`

2) Генерируется массив в функции main, там же, внутри обоих циклов(прохождения по всем тестам и повторений этого теста) создается вектор vector_values, который будет хранить сгенерированные вещественные числа.

3)Далее в той же функции main замеряется время старта до сортировки и после и выводится в файл. Сортируется массив в отдельной функции insertionSort(), сортирующей массив методом вставок.

4)В Excel заношу данные из файла и строю сначала графики худшего, лучшего и среднего случая, выбирая в качестве области значений соответственно все худшие, лучшие средние значения тестов.

5)Строю график O(c * g(N)) , где g(N) соответствует асимптотической сложности рассматриваемого метода сортировки, что бы начиная с N ~ 1000 график асимптотической сложности возрастал быстрее чем полученное худшее время, но при этом был различим на графике.

$$Y=0.000101742*n^2$$

Результаты:

1)Для более наглядной демонстрации протестировал сам процесс сортировки на меньших значениях TEST_VALUES

```
До Сортировки:
0.7908,0.963534,0.167659,-0.542941,

После Сортировки:
-0.542941,0.167659,0.7908,0.963534,

До Сортировки:
-0.69516,0.998865,-0.393761,0.0930644,

После Сортировки:
-0.69516,-0.393761,0.0930644,0.998865,

До Сортировки:
-0.70495,0.085363,0.325407,-0.703666,0.218759,-0.347143,-0.818076,0.537148,

После Сортировки:
-0.818076,-0.70495,-0.703666,-0.347143,0.085363,0.218759,0.325407,0.537148,

До Сортировки:
0.695435,-0.388368,-0.944589,-0.519859,-0.0322989,0.454569,0.335982,0.691403,

После Сортировки:
-0.944589,-0.519859,-0.388368,-0.0322989,0.335982,0.454569,0.691403,0.695435,
```

Рис. 1. Демонстрация работы сортировки

2)Нанесение всех точек на график.

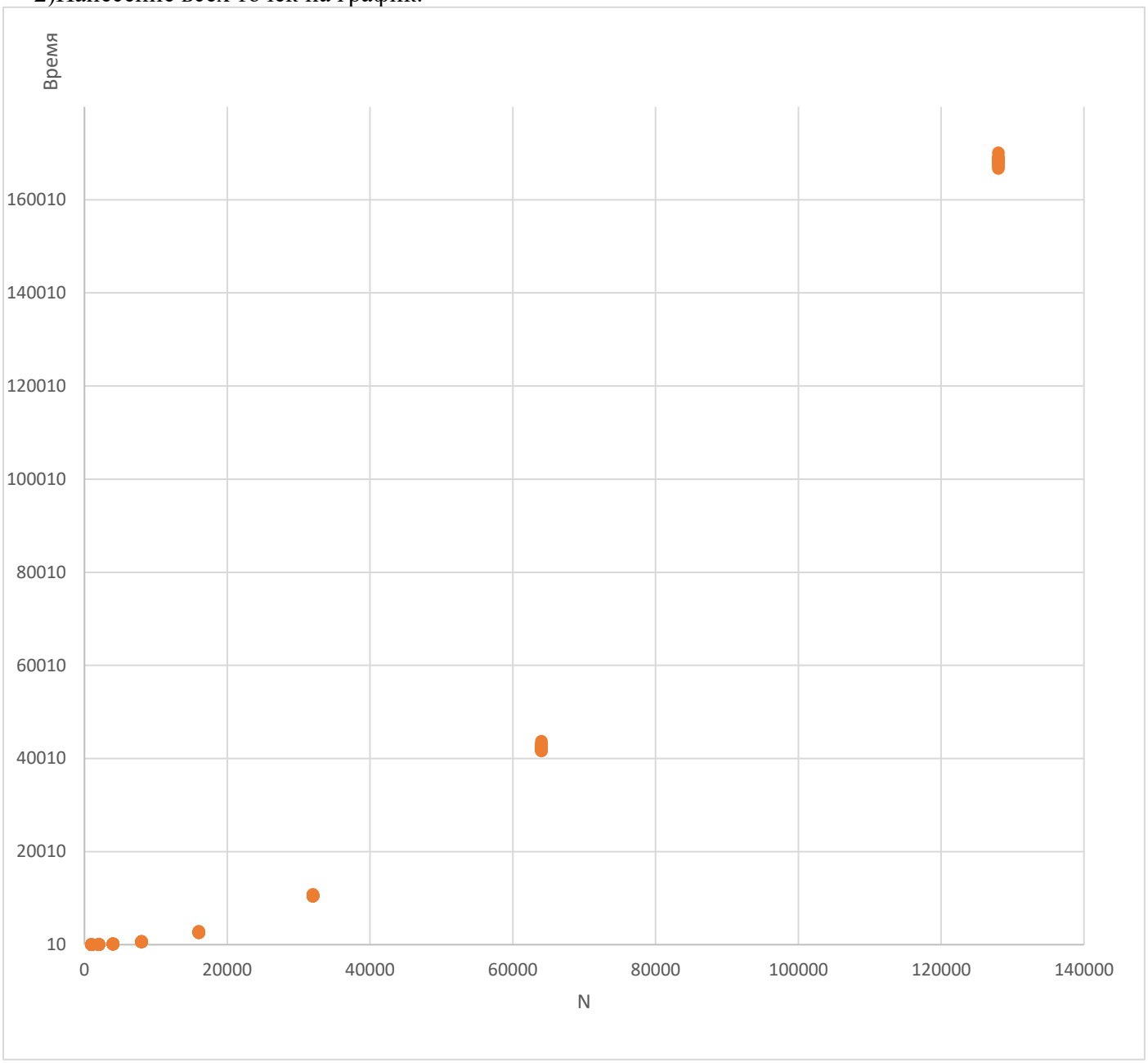

Рис.2 График со всеми точками, полученными в результате тестирования

3) По полученным данным точкам, построил график лучшего, среднего и худшего случая.

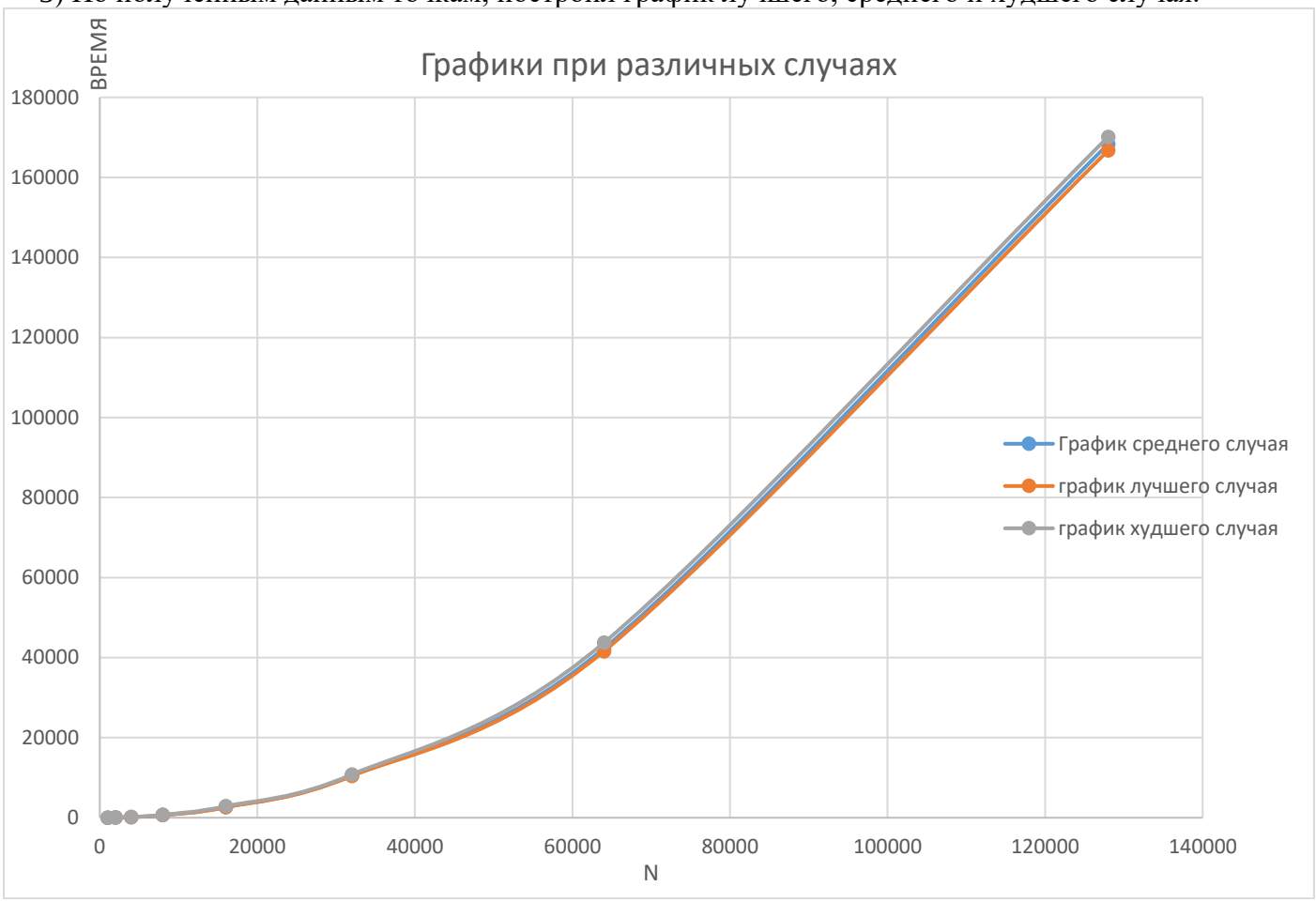

Рис.3. Графики различны случаев.

4)Доп. Построение графика худшего случая и графика O(c*g(N)), который с N 1000 возрастал бы быстрее, чем полученное худшее время.

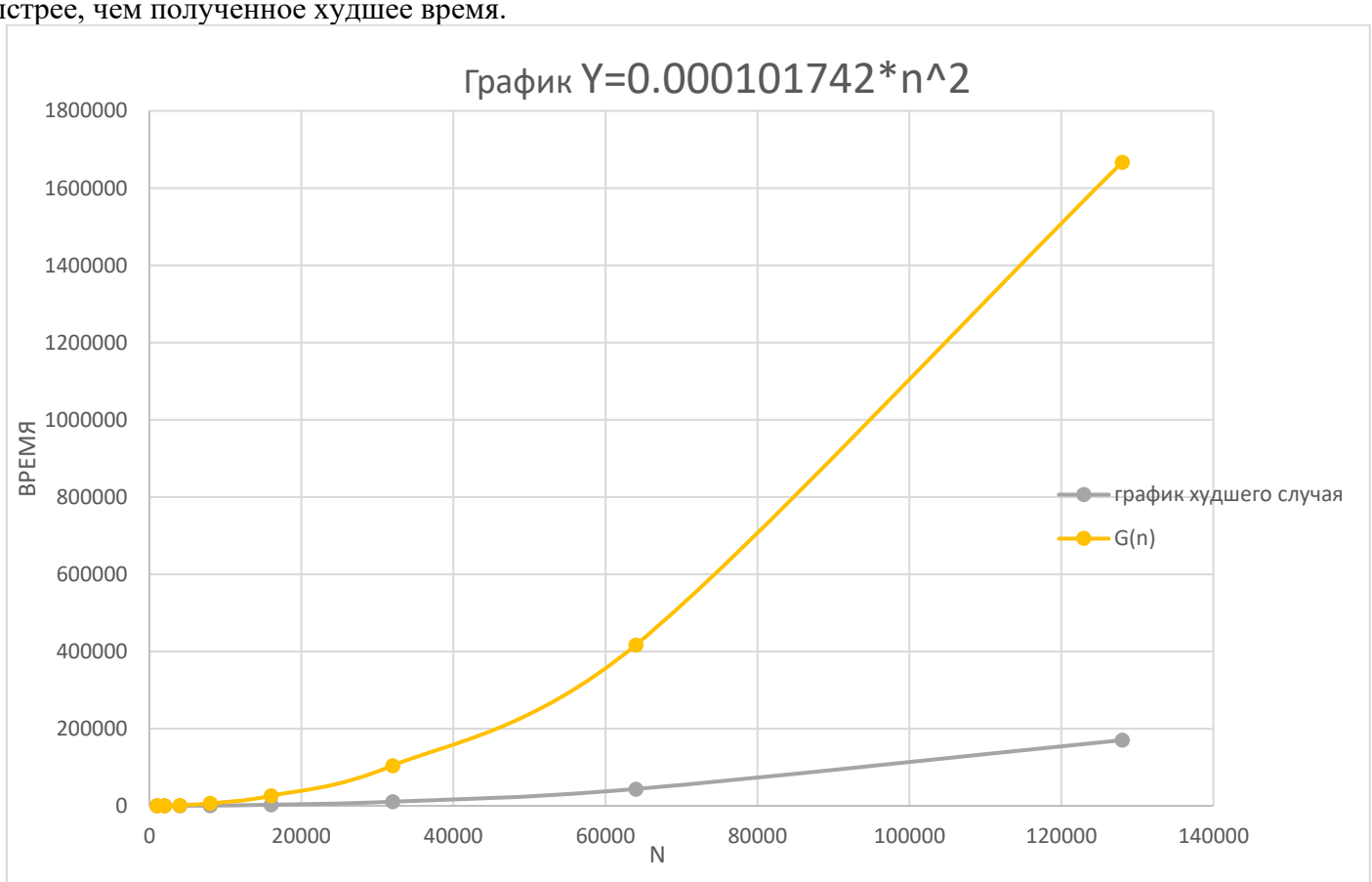

Рис.4. График худшего случая и график O(c*g(N)).

Заключение.

Метод сортировки вставками оказался достаточно прост в реализации, но на больших наборах данных лучше выбрать другой алгоритм. Он наиболее эффективен на небольших набор данных и когда элементы частично отсортированы.