Министерство науки и высшего образования Российской Федерации
Российский химико-технологический университет имени Д. И. Менделеева

Факультет информационных технологий и управления
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Сортировка»

Выполнил студент группы КС-30 Колесников Артем Максимович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/AMKolesnikov_30_ALG

Приняли: аспирант кафедры ИКТ Пысин Максим Дмитриевич
аспирант кафедры ИКТ Краснов Дмитрий Олегович

Дата сдачи: 14.03.2022

Москва
2022

Содержание

Описание задачи

Изучить и реализовать метод сортировки вставками (вариант 3). Для реализованного метода сортировки 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 отсортировать и замерить время, требуемое на сортировку
- результат замера для каждой попытки каждого теста записать в общий файл

По окончанию всех тестов нанести все точки, полученные в результате замеров времени на график, где на ось абсцисс(X) 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

Сортировка вставками (англ. Insertion sort) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность — $O(n^2)$.

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.

Выполнение задачи

Алгоритм сортировки вставками был реализован на языке C++. Моя программа состоит из 4 функций ~ numGenerator, vectorGenerator, insertionSort, main.

Функция «numGenerator» ~ возвращает случайное число с плавающей запятой в диапазоне от Min до Max.

```
double numGenerator(double min, double max) {
	return min + ((double)rand() / RAND_MAX * (max - min));
	 }
```

Функция «vectorGenerator» ~ возвращает вектор размера N заполненный случайными дробными числами от -1 до 1, полученными при помощи функции «numGenerator».

```
vector <double> vectorGenerator(int N) {

	const int MIN = -1;
	const int MAX = 1;

	vector <double> v;
	v.resize(N);

	for (int i = 0; i < N; i++)
	{
		v[i] = numGenerator(MIN, MAX);
	}

	return v;
	 }
```

Функция «insertionSort» ~ сортирует полученный вектор методом вставок и возвращает время в миллисекундах, затраченное на сортировку.

```
double insertionSort(vector <double> v) {

	chrono::high_resolution_clock::time_point start =
chrono::high_resolution_clock::now();

	for (int i = 1; i < v.size(); i++)
	{
		double key = v[i];
		int j = i - 1;

		while (j >= 0 && v[j] > key)
		{
			v[j + 1] = v[j];
			v[j] = key;
			j--;
		}
	}

	chrono::high_resolution_clock::time_point end =
chrono::high_resolution_clock::now();
	chrono::duration<double, milli> milli_diff = end - start;


	return milli_diff.count();
	 }
```

Функция «main» ~ состоит из цикла в котором происходит расчет минимального, максимального, среднего времени, затраченного на сортировку вектора, состоящего из N = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 } чисел, каждое из которых рассчитывается 20 раз. На выходе мы получаем файл «Results.txt» с результатами работы нашего алгоритма.

```
int main()
{
	srand(time(0));

	vector <int> N = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 };
	const int NUM_TRIES = 20;
	double time[NUM_TRIES] = { 0 };

	ofstream fout("Results.txt");
	if (fout.is_open())
	{
		for (int i = 0; i < N.size(); i++) {
			double summ = 0;
			fout << "Количество чисел: " << N[i] << endl << "Время (мс): ";

			for (int j = 0; j < NUM_TRIES; j++) {
				time[j] = insertionSort(vectorGenerator(N[i]));
				fout << time[j] << " ; ";
				summ += time[j];
			}

			double min = time[0];
			double max = time[0];

			for (int j = 0; j < NUM_TRIES; j++) {
				if (max < time[j]) max = time[j];
				if (min > time[j]) min = time[j];
			}
			fout << endl << "Минимальное: " << min << endl << "Максимальное: " <<
max << endl << "Среднее: " << summ / NUM_TRIES << endl;
			fout << endl << endl;
		}

		fout.close();
	}

	 }
```

N	мин	сред	макс	асимптотика
1000	14,2668	15,6563	18,1195	21,8
2000	58,8327	65,1411	85,9129	87,2
4000	229,053	234,701	239,739	348,8
8000	925,602	937,063	956,991	1395,2
16000	3727,32	3880,27	4091,7	5580,8
32000	14962,9	15511,6	16315	22323,2
64000	57824,3	58500,1	63656,8	89292,8
128000	231146	232303	233126	357171,2

Таблица 1 – Максимальное, минимальное, среднее время (в миллисекундах) сортировки векторов, состоящих из N элементов.

По полученным результатам был построен график лучшего (минимальное время для каждого N), худшего (максимальное время для каждого N) и среднего (среднее время для каждого N) случая.:

Рисунок 1 – График лучшего (минимальное время для каждого N), худшего (максимальное время для каждого N) и среднего (среднее время для каждого N) случая.

Рисунок 2 – График лучшего, худшего и среднего случая в диапазоне количества чисел массива [1000; 8000].

Рисунок 3 – График лучшего, худшего и среднего случая в диапазоне количества чисел массива [16000; 128000].

В качестве дополнительного задания был построен график худшего случая и график $O(c * n^2)$, где значение C подобрано так, что начиная с N = 1000 график асимптотической сложности возрастал быстрее чем полученное худшее время, но при этом был различим на графике.

$$C = 2{,}18 * 10^{-5}$$

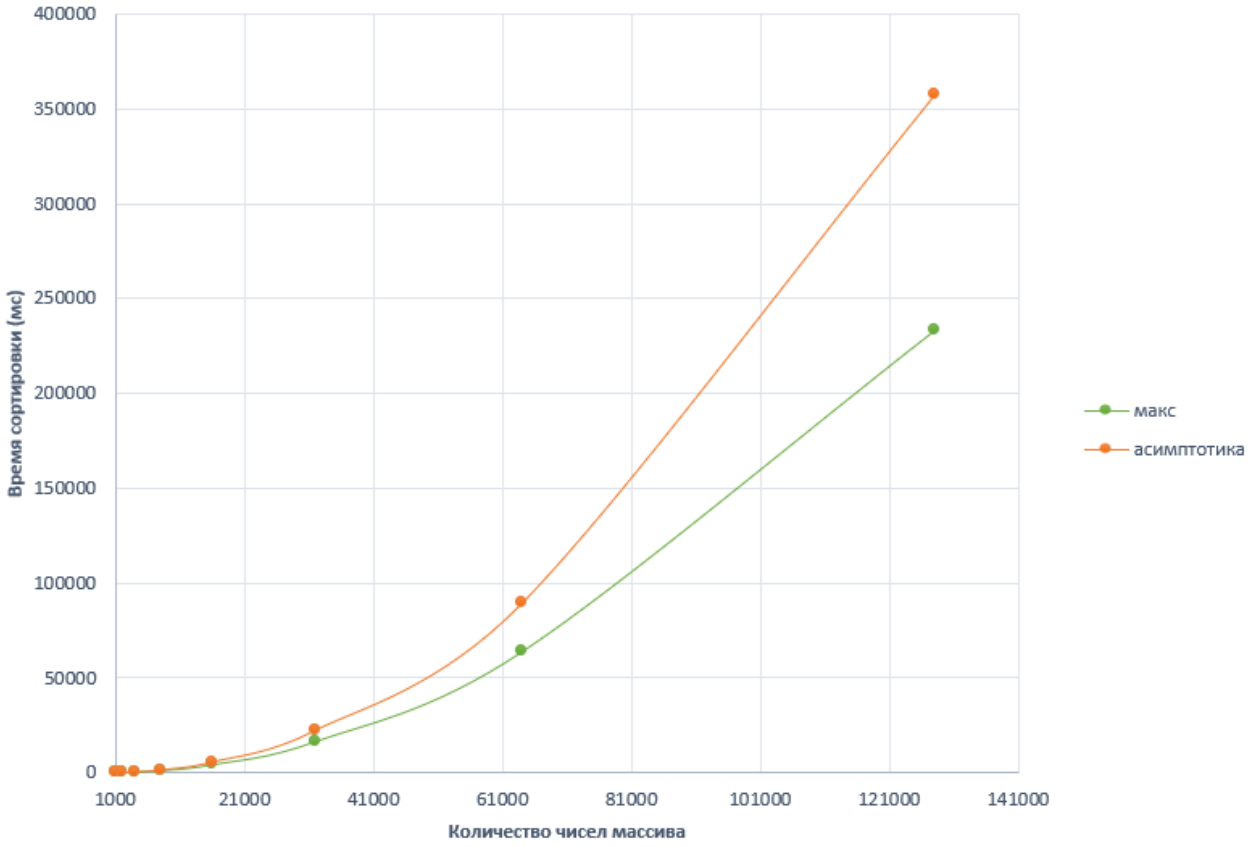

Рисунок 4 – График худшего случая и асимптотической сложности

Заключение

Сортировка вставками достаточно простой алгоритм с квадратичной временной сложностью, поэтому производительность алгоритма прямо пропорциональна размеру входных данных в квадрате. При увеличении количества элементов N времени на сортировку вектора затрачивалось все больше и больше. На сортировку вектора, состоящего из 64000 элементов, в среднем требовалось около 58,5 секунды, а когда N достигло 128000 эл. сортировка одного вектора стала занимать 232,3 секунды, что почти равно 4 минутам, а когда таких векторов 20 работа программы занимает уйму времени. Сортировка вставками работает меньше всего, когда массив уже упорядочен, и больше всего, когда массив отсортирован в обратном порядке. Вторым преимуществом является то, что данный алгоритм не меняет порядок одинаковых ключей ~ то есть алгоритм является устойчивым.