

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

Факультет цифровых технологий и химического инжиниринга
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1
ПО КУРСУ
«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»:

СТУДЕНТ группы КС-33

Костяева К.С.

Москва

2024

ТЕОРИЯ

Сортировка вставками (Insertion Sort) — это алгоритм сортировки, который работает по принципу вставки элементов в уже отсортированную часть массива. Он часто используется для сортировки небольших массивов или как часть более сложных алгоритмов сортировки.

Алгоритм:

1. Начнем с того, что первый элемент массива считается отсортированным.
2. Далее, начиная со второго элемента массива, каждый элемент пытаемся вставить в правильную позицию среди уже отсортированных элементов.
3. Для этого мы сравниваем текущий элемент с элементами, которые находятся перед ним, и сдвигаем их вправо, если они больше текущего элемента.
4. После сдвига элемента на нужное место вставляем его в отсортированную часть массива.

Пример:

Предположим, у нас есть массив:

[5, 2, 9, 1, 5, 6]

1. Первый элемент уже считается отсортированным (это 5).
2. Вставляем второй элемент 2:
 - Сравниваем 2 с 5 — 2 меньше, поэтому сдвигаем 5 на место 2.
 - Массив: [2, 5, 9, 1, 5, 6]
3. Вставляем третий элемент 9:
 - Сравниваем 9 с 5 — 9 больше, оставляем как есть.
 - Массив: [2, 5, 9, 1, 5, 6]
4. Вставляем четвертый элемент 1:
 - Сравниваем 1 с 9, сдвигаем 9 вправо.
 - Сравниваем 1 с 5, сдвигаем 5 вправо.
 - Сравниваем 1 с 2, сдвигаем 2 вправо.
 - Массив: [1, 2, 5, 9, 5, 6]
5. Вставляем пятый элемент 5:
 - Сравниваем 5 с 9, сдвигаем 9 вправо.
 - Сравниваем 5 с 5, оставляем как есть.
 - Массив: [1, 2, 5, 5, 9, 6]
6. Вставляем последний элемент 6:
 - Сравниваем 6 с 9, сдвигаем 9 вправо.
 - Сравниваем 6 с 5, оставляем как есть.
 - Массив: [1, 2, 5, 5, 6, 9]

После всех шагов массив отсортирован.

Время работы:

- В худшем случае (когда массив отсортирован в обратном порядке) алгоритм выполняет $O(n^2)$ операций, где n — количество элементов в массиве.

- В лучшем случае (когда массив уже отсортирован) алгоритм работает за **$O(n)$** времени, так как на каждом шаге не требуется сдвигать элементы.
- В среднем случае время работы также **$O(n^2)$** .

ЗАДАНИЕ

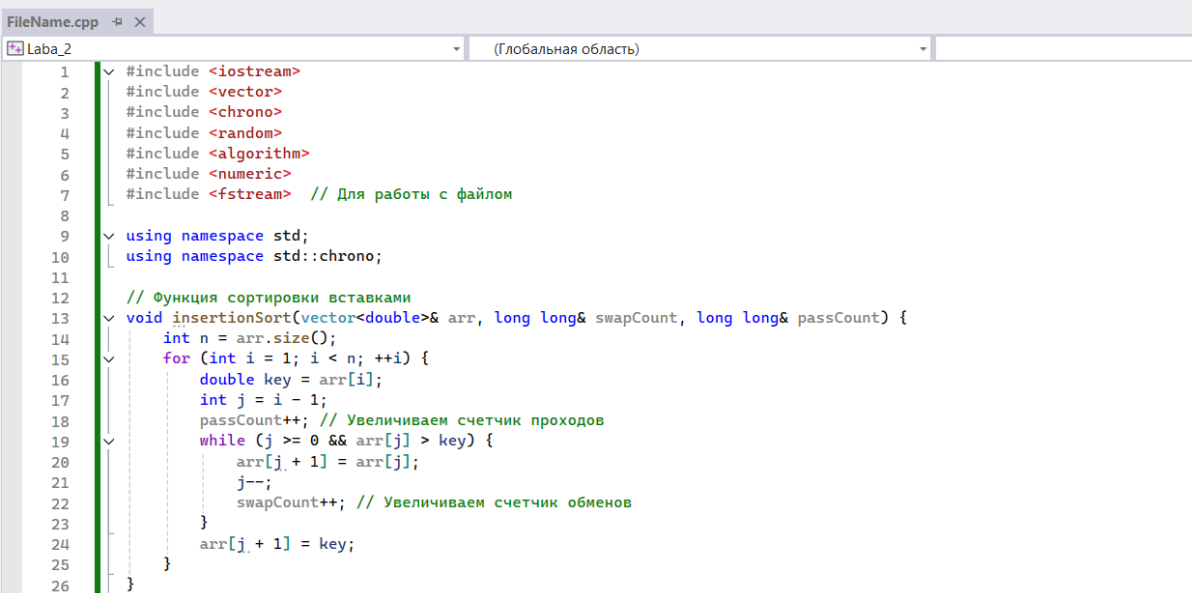
1. Реализовать сортировку вставками.
2. Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров времени.
3. За один расчет выполняется следующие операции.

Выполняется алгоритм сортировки:

- а) Во время выполнения измерить количество повторных проходов по массиву.
 - б) Во время выполнения измерить количество выполнения операций обмена значений.
4. По полученным данным времени построить графики зависимости времени от числа элементов в массиве: График среднего количества обмена значений.

ПРАКТИЧЕСКАЯ ЧАСТЬ

язык программирования C++



```

1  #include <iostream>
2  #include <vector>
3  #include <chrono>
4  #include <random>
5  #include <algorithm>
6  #include <numeric>
7  #include <fstream> // Для работы с файлом
8
9  using namespace std;
10 using namespace std::chrono;
11
12 // Функция сортировки вставками
13 void insertionSort(vector<double>& arr, long long& swapCount, long long& passCount) {
14     int n = arr.size();
15     for (int i = 1; i < n; ++i) {
16         double key = arr[i];
17         int j = i - 1;
18         passCount++; // Увеличиваем счетчик проходов
19         while (j >= 0 && arr[j] > key) {
20             arr[j + 1] = arr[j];
21             j--;
22             swapCount++; // Увеличиваем счетчик обменов
23         }
24         arr[j + 1] = key;
25     }
26 }

```

```
FileName.cpp  X
Laba_2 (Глобальная область)

28 int main() {
29     vector<int> sizes = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 };
30     int seriesCount = 20;
31     mt19937 engine(time(0));
32     uniform_real_distribution<double> gen(-1.0, 1.0);
33
34     // Открытие CSV файла для записи
35     ofstream outFile("sorting_results.csv");
36     outFile << "Size,Average Time (sec),Min Time (sec),Max Time (sec),Average Swaps,Average Passes\n";
37
38     for (int size : sizes) {
39         vector<double> times;
40         vector<long long> swapCounts;
41         vector<long long> passCounts;
42
43         for (int i = 0; i < seriesCount; ++i) {
44             vector<double> arr(size);
45             for (auto& el : arr) {
46                 el = gen(engine);
47             }
48
49             long long swapCount = 0;
50             long long passCount = 0;
51
52             auto start = high_resolution_clock::now();
53             insertionSort(arr, swapCount, passCount);
54             auto end = high_resolution_clock::now();
55
56             duration<double> diff = end - start;
57             times.push_back(diff.count());
58             swapCounts.push_back(swapCount);
59             passCounts.push_back(passCount);
60         }
    }
```

```
FileName.cpp  X
Laba_2 (Глобальная область)

61
62     // Находим минимальное и максимальное время
63     double minTime = *min_element(times.begin(), times.end());
64     double maxTime = *max_element(times.begin(), times.end());
65
66     // Записываем результаты в файл
67     outFile << size << ", "
68             << accumulate(times.begin(), times.end(), 0.0) / seriesCount << ", "
69             << minTime << ", "
70             << maxTime << ", "
71             << accumulate(swapCounts.begin(), swapCounts.end(), 0LL) / seriesCount << ", "
72             << accumulate(passCounts.begin(), passCounts.end(), 0LL) / seriesCount << "\n";
73
74     // Выводим результаты в консоль
75     cout << "Size: " << size << endl;
76     cout << "Average time: " << accumulate(times.begin(), times.end(), 0.0) / seriesCount << " sec." << endl;
77     cout << "Min time: " << minTime << " sec." << endl;
78     cout << "Max time: " << maxTime << " sec." << endl;
79     cout << "Average swap count: " << accumulate(swapCounts.begin(), swapCounts.end(), 0LL) / seriesCount << endl;
80     cout << "Average pass count: " << accumulate(passCounts.begin(), passCounts.end(), 0LL) / seriesCount << endl;
81     cout << "-----" << endl;
82
83     outFile.close(); // Закрытие файла
84
85     return 0;
86 }
87
88
```

1. Основная функция сортировки вставками (insertionSort):

- Принимает на вход вектор с данными и два счетчика: swapCount (для подсчета количества обменов) и passCount (для подсчета количества проходов по массиву).
- Алгоритм сортирует массив и увеличивает соответствующие счетчики.

2. Генерация случайных данных:

- Массивы для сортировки создаются с разными размерами (от 1000 до 128000 элементов).
- Для генерации случайных чисел используется распределение от -1.0 до 1.0.

3. Тестирование и замеры:

- Для каждого размера массива выполняется несколько серий тестов (в данном случае 20).
- Для каждой серии измеряется время сортировки, количество обменов и проходов.
- Используется high_resolution_clock для точного измерения времени.

4. Запись в файл:

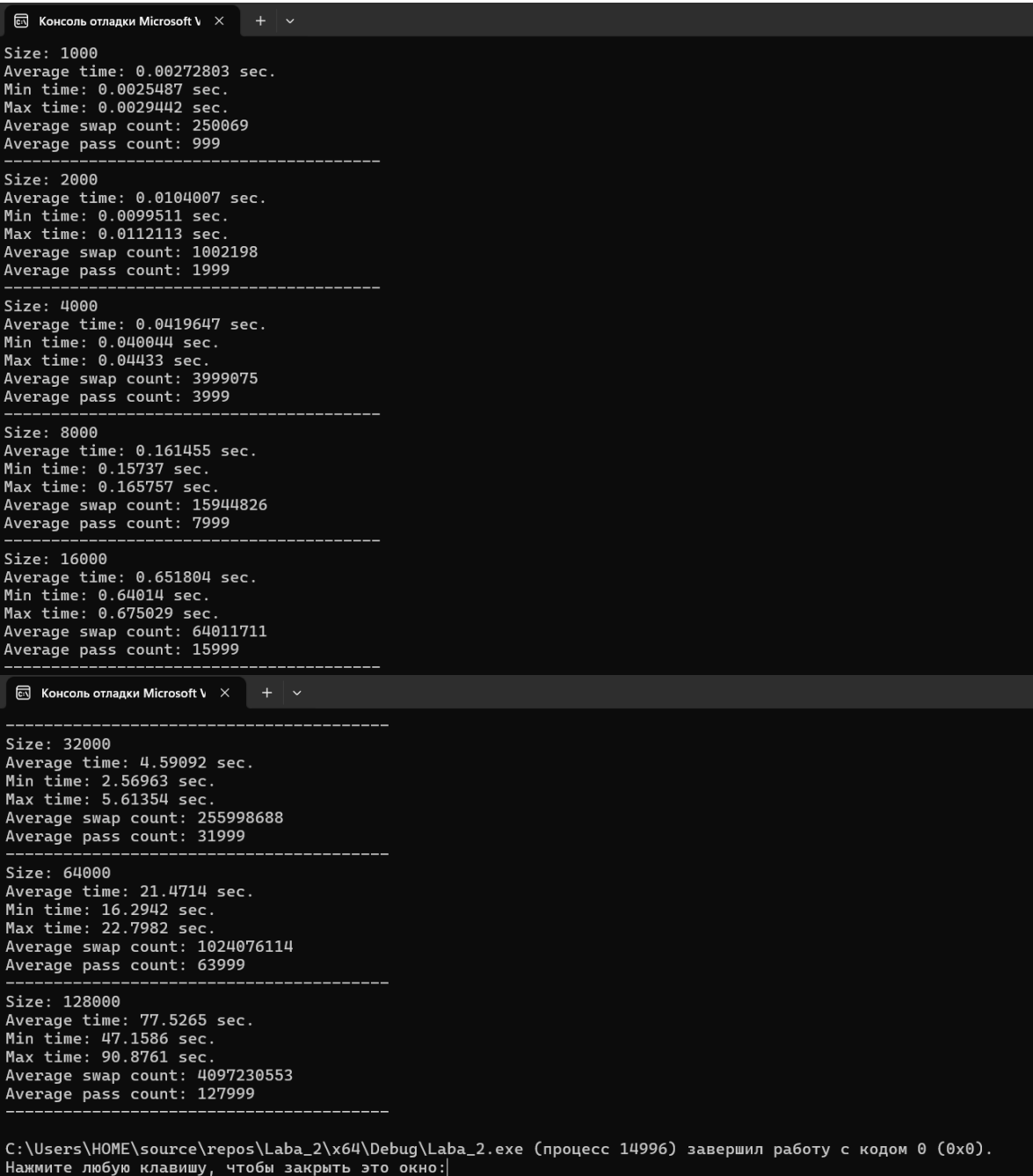
- После проведения тестов для каждого размера массива, результаты (среднее время, минимальное и максимальное время, среднее количество обменов и проходов) записываются в CSV файл `sorting_results.csv`.

5. Вывод в консоль:

- Результаты для каждого размера массива также выводятся на экран, включая среднее время, минимальное и максимальное время, среднее количество обменов и проходов.

Результат работы программы:

ВЫВОД В КОНСОЛЬ

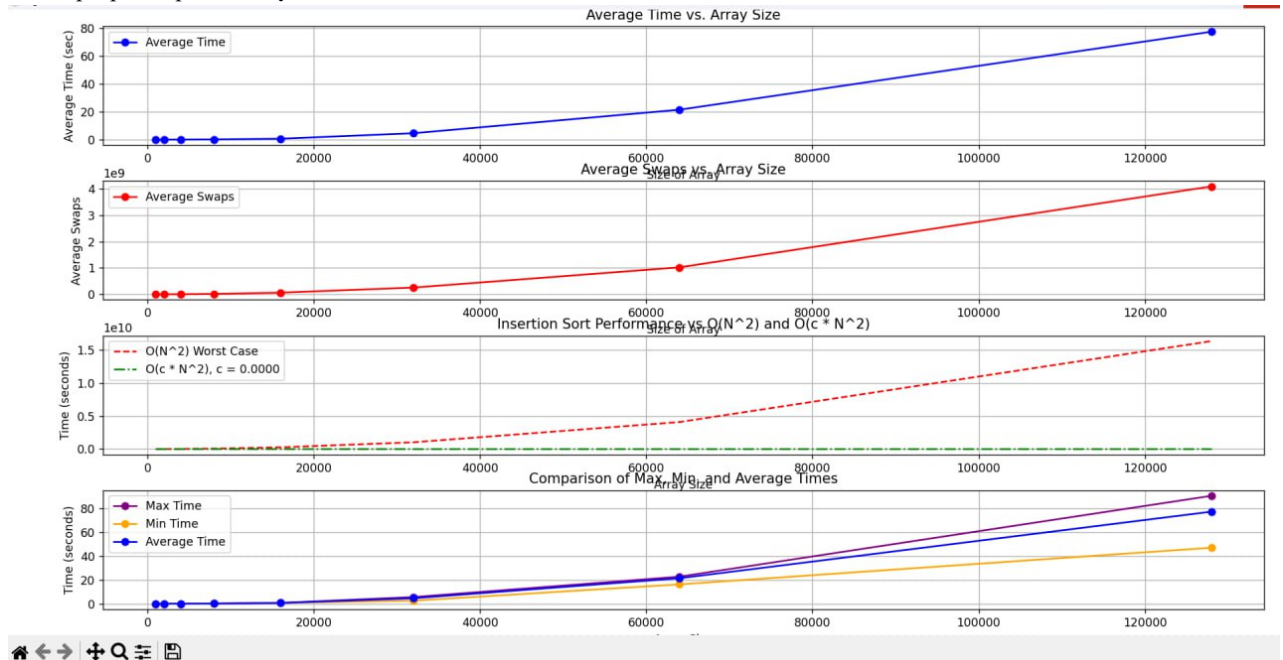


```
Консоль отладки Microsoft V  X + v
Size: 1000
Average time: 0.00272803 sec.
Min time: 0.0025487 sec.
Max time: 0.0029442 sec.
Average swap count: 250069
Average pass count: 999
-----
Size: 2000
Average time: 0.0104007 sec.
Min time: 0.0099511 sec.
Max time: 0.0112113 sec.
Average swap count: 1002198
Average pass count: 1999
-----
Size: 4000
Average time: 0.0419647 sec.
Min time: 0.040044 sec.
Max time: 0.04433 sec.
Average swap count: 3999075
Average pass count: 3999
-----
Size: 8000
Average time: 0.161455 sec.
Min time: 0.15737 sec.
Max time: 0.165757 sec.
Average swap count: 15944826
Average pass count: 7999
-----
Size: 16000
Average time: 0.651804 sec.
Min time: 0.64014 sec.
Max time: 0.675029 sec.
Average swap count: 64011711
Average pass count: 15999
-----
Size: 32000
Average time: 4.59092 sec.
Min time: 2.56963 sec.
Max time: 5.61354 sec.
Average swap count: 255998688
Average pass count: 31999
-----
Size: 64000
Average time: 21.4714 sec.
Min time: 16.2942 sec.
Max time: 22.7982 sec.
Average swap count: 1024076114
Average pass count: 63999
-----
Size: 128000
Average time: 77.5265 sec.
Min time: 47.1586 sec.
Max time: 90.8761 sec.
Average swap count: 4097230553
Average pass count: 127999
-----
C:\Users\HOME\source\repos\Laba_2\x64\Debug\Laba_2.exe (процесс 14996) завершил работу с кодом 0 (0x0).
Нажмите любую клавишу, чтобы закрыть это окно:
```

вывод в файл CSV

	A	B	C	D	E	F	G	H
1	Size,Average Time (sec),Min Time (sec),Max Time (sec),Average Swaps,Average Passes							
2	1000,0.00272803,0.0025487,0.0029442,250069,999							
3	2000,0.0104007,0.0099511,0.0112113,1002198,1999							
4	4000,0.0419647,0.040044,0.04433,3999075,3999							
5	8000,0.161455,0.15737,0.165757,15944826,7999							
6	16000,0.651804,0.64014,0.675029,64011711,15999							
7	32000,4.59092,2.56963,5.61354,255998688,31999							
8	64000,21.4714,16.2942,22.7982,1024076114,63999							
9	128000,77.5265,47.1586,90.8761,4097230553,127999							
10								

язык программирования Python



- Среднее время сортировки.
- Среднее количество обменов.
- Теоретическая сложность и фактическая зависимость времени от N.
- Сравнение максимального, минимального и среднего времени.

1. Функция `calculate_O_N_squared(sizes):`

- Эта функция вычисляет теоретическую сложность $O(N^2)$ для каждого размера массива, где `sizes` — это список размеров массива.

2. Функция `calculate_c_factor(avg_times, sizes):`

- Мы подбираем константу `c`, чтобы она масштабировала теоретическое время ($O(N^2)$) так, чтобы оно не становилось прямой линией, но было выше реального времени работы сортировки. Мы вычисляем максимальное теоретическое время и максимальное фактическое время сортировки, а затем находим константу `c`, которая масштабирует $O(N^2)$ относительно реального времени работы.

3. Построение графиков:

- В первой части кода строится график, который показывает среднее время работы сортировки вставками в зависимости от размера массива.

- Во второй части строится график среднего количества обменов, чтобы продемонстрировать, как количество обменов зависит от размера массива.
- В третьей части строятся графики для $O(N^2)$ (красный, худший случай) и $O(c * N^2)$ (зеленый, теоретическая оценка с подогнанной константой).

Как работает подбираемая константа c :

- Мы используем максимальное время работы алгоритма и теоретическое время ($O(N^2)$) для нахождения коэффициента, который позволяет масштабировать теоретическое время так, чтобы оно было выше фактического времени сортировки, но не становилось прямой линией.