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 № \_

Выполнил студент группы КС-36 Полковникова Д.Д.

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/Polkovnikova\_CS-36

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 17.02.2025

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, соответствующий варианту (здесь: Вариант №3 - метод вставки).

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

Допускает 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, соответствующий вашему варианту:
  1. Сортировка пузырьком
  2. Сортировка перемешиванием
  3. Сортировка вставками
  4. Сортировка выбором
* Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров времени.  
  За один расчет выполняется следующие операции:
  1. Генерируется массив случайных значений
  2. Запоминается время начала расчета алгоритма сортировки
  3. Выполняется алгоритм сортировки
  4. Во время выполнения измерить количество повторных прохождений по массиву.
  5. Во время выполнения измерить количество выполнения операций обмена значений.
  6. Вычисляется время, затраченное на сортировку: текущее время - время начала
  7. Сохраняется время для одной попытки После этого расчет повторяется до окончания серии.
  8. Алгоритм вычисляется 8 сериями по 20 раз за серию.
  9. Алгоритм в каждой серии вычисляется для массива размером M. (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000)
  10. Массив заполняется значения числами с плавающей запятой в интервале от -1 до 1.
  11. Для серии запоминаются все времена которые были замерены
* По полученным данным времени построить графики зависимости времени от числа элементов в массиве:
  1. Совмешеееый 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 с, что бы при значении > 1000 график O(N) был выше графика наихудшего случая, но второй график на его фоне не превращался в прямую линию

* 1. Совмещенный график среднего, наилучшего и наихудшего времени исполнения.
  2. График среднего колличества обмена значений.
  3. График повторных обходов массива.
* По результатом расчетов оформляется отчет по предоставленной форме, в отчете:
  1. Приводится описание алгоритма.
  2. Приводится описания выполнения задачи (Описание кода и спецефических элементов реализации)
  3. Приводятся выводы (Графики и их анализ)

# Описание метода/модели.

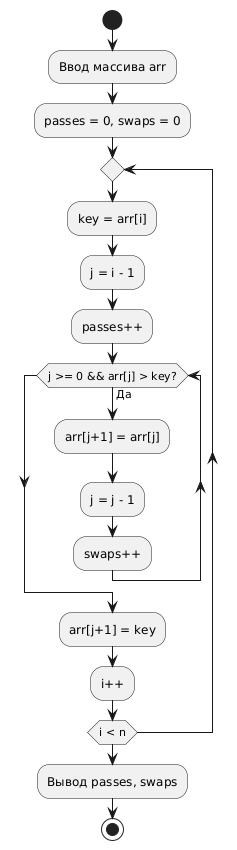


Рис.1 Блок-схема алгоритма метода вставками.

Алгоритм сортировки вставками — это простой алгоритм, который работает по принципу постепенного упорядочивания элементов в массиве. Он имитирует процесс сортировки карт в руках: берём один элемент за раз и вставляем его в правильную позицию относительно уже отсортированной части массива.

**Принцип работы алгоритма:**

1. Начинаем с 1-го элемента (первый элемент считается уже отсортированным).
2. Берём следующий элемент и сравниваем его с предыдущими.
3. Перемещаем элементы, если они больше текущего, освобождая место для вставки.
4. Вставляем текущий элемент в правильное место.
5. Повторяем этот процесс, пока не будет обработан весь массив.

Алгоритм сортировки вставками — это простой алгоритм, который работает по принципу постепенного упорядочивания элементов в массиве. Он имитирует процесс сортировки карт в руках: берём один элемент за раз и вставляем его в правильную позицию относительно уже отсортированной части массива.

**Асимптотическая сложность (Big-O нотация)**

1. **В худшем случае (O(n²))** — когда массив изначально отсортирован в обратном порядке, каждому элементу приходится сдвигать все предыдущие, что даёт квадратичную сложность.
2. **В лучшем случае (O(n))** — если массив уже отсортирован, алгоритм делает только один проход без перестановок.
3. **В среднем случае (O(n²))** — в среднем алгоритм делает n²/4 сравнений и перемещений.

**Потребляемая память**

**O(1)** — алгоритм выполняет сортировку на месте, используя только вспомогательные переменные.

**Преимущества**

1. Простая реализация.
2. Эффективен для небольших массивов.
3. Быстрее, чем другие O(n²) алгоритмы на почти отсортированных данных.

**Недостатки**

1. Медленный на больших объёмах данных.
2. O(n²) сложность в худшем и среднем случаях делает его неэффективным для крупных массивов.

**Эксперимент**

Для оценки эффективности алгоритма выполнены 8 серий тестов по 20 запусков, анализируя:

1. Время выполнения (сек)
2. Число перестановок
3. Число проходов по массиву

Исследования проводились на массивах размером от **1 000 до 128 000 элементов**, заполненных случайными числами в диапазоне **[-1, 1]**.

# Выполнение задачи.

Для расчёта промежутков времени для выполнения сортировки был использован язык программирования С++, а для построения графиков - Python и ПО Excel. Так как С++ считается одним из самых быстрых языков программирования.

**Код:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <cstdint> // Для uint64\_t

using namespace std;

/\*\*

\* Выполняет сортировку вставками и считает количество проходов и перестановок.

\*

\* @param arr Вектор чисел для сортировки.

\* @return Пара (количество проходов, количество перестановок).

\*/

pair<int, uint64\_t> insertionSort(vector<double>& arr) {

int n = arr.size(); //Кол-во эл-тов

int passes = 0; // Количество проходов (итераций внешнего цикла)

uint64\_t swaps = 0; // Количество перестановок элементов

for (int i = 1; i < n; i++) {

double key = arr[i];

int j = i - 1;

passes++; // Фиксируем проход по массиву

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j]; // Сдвигаем элементы вправо

j--;

swaps++; // Фиксируем перестановку

}

// Устанавливаем key на правильное место

arr[j + 1] = key;

if (j + 1 != i) { // Если key не остался на месте, то засчитываем перестановку

swaps++;

}

}

return {passes, swaps};

}

/\*\*

\* Генерирует вектор случайных чисел в диапазоне [-1, 1].

\*

\* @param N Количество чисел.

\* @return Вектор случайных чисел.

\*/

vector<double> generateNumbers(int N) {

vector<double> numbers(N);

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

numbers[i] = (rand() / (double)RAND\_MAX) \* 2.0 - 1.0;

}

return numbers;

}

int main() {

srand(time(nullptr));

vector<int> sizes = {1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000};

for (int size : sizes) {

cout << "Размер массива: " << size << "\n";

for (int series = 0; series < 8; series++) {

vector<double> times, swapsList, passesList;

for (int i = 0; i < 20; i++) {

vector<double> numbers = generateNumbers(size);

clock\_t start = clock();

auto [passes, swaps] = insertionSort(numbers);

clock\_t end = clock();

double elapsedTime = double(end - start) / CLOCKS\_PER\_SEC;

times.push\_back(elapsedTime);

swapsList.push\_back(swaps);

passesList.push\_back(passes);

}

cout << "Серия " << series + 1 << ":\n";

cout << " Время сортировки: ";

for (double t : times) cout << t << " ";

cout << "\n";

cout << " Количество перестановок: ";

for (uint64\_t s : swapsList) cout << s << " ";

cout << "\n";

cout << " Количество проходов по массиву: ";

for (double p : passesList) cout << p << " ";

cout << "\n\n";

}

}

return 0;

}**Результатом** данного кода получаются следующие значения:

Время сортировки для 1000 элементов: 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002 0.001 0.002

Количество перестановок для 1000 элементов: 249826 240400 243643 252387 243553 258917 249687 255428 245327 255443 248898 256059 249737 250149 248976 255208 252246 253661 240253 251944

Количество проходов по массиву для 1000 элементов: 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999

Время сортировки для 2000 элементов: 0.006 0.007 0.006 0.006 0.006 0.006 0.006 0.006 0.005 0.006 0.006 0.006 0.005 0.006 0.005 0.006 0.006 0.006 0.006 0.006

Количество перестановок для 2000 элементов: 996140 1.00449e+006 1.03035e+006 1.00609e+006 995622 998022 993584 998763 1.00423e+006 1.01922e+006 971087 1.01154e+006 986877 1.00986e+006 986504 1.01085e+006 1.01242e+006 1.00261e+006 975877 1.0155e+006

Количество проходов по массиву для 2000 элементов: 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999 1999

Время сортировки для 4000 элементов: 0.024 0.024 0.024 0.024 0.023 0.023 0.023 0.023 0.024 0.027 0.024 0.024 0.023 0.024 0.023 0.023 0.025 0.024 0.022 0.023

Количество перестановок для 4000 элементов: 4.00387e+006 3.97065e+006 3.96687e+006 4.02171e+006 4.0083e+006 3.91235e+006 4.01836e+006 3.96619e+006 4.01777e+006 3.94199e+006 3.96981e+006 3.99933e+006 4.06402e+006 4.07495e+006 4.06249e+006 3.98356e+006 4.06593e+006 3.98082e+006 3.91466e+006 3.96855e+006

Количество проходов по массиву для 4000 элементов: 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999 3999

Время сортировки для 8000 элементов: 0.093 0.093 0.094 0.092 0.091 0.095 0.092 0.093 0.092 0.099 0.096 0.093 0.092 0.093 0.094 0.093 0.094 0.092 0.093 0.098

Количество перестановок для 8000 элементов: 1.59631e+007 1.57624e+007 1.60615e+007 1.59677e+007 1.59202e+007 1.61869e+007 1.59387e+007 1.60964e+007 1.56901e+007 1.59942e+007 1.63115e+007 1.57871e+007 1.59241e+007 1.61244e+007 1.5999e+007 1.59739e+007 1.59668e+007 1.59383e+007 1.60293e+007 1.56709e+007

Количество проходов по массиву для 8000 элементов: 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999 7999

Время сортировки для 16000 элементов: 0.38 0.376 0.378 0.382 0.37 0.376 0.374 0.369 0.379 0.371 0.372 0.378 0.374 0.374 0.387 0.371 0.372 0.374 0.376 0.375

Количество перестановок для 16000 элементов: 6.43272e+007 6.42484e+007 6.40726e+007 6.40747e+007 6.38906e+007 6.44689e+007 6.43311e+007 6.36918e+007 6.37568e+007 6.38844e+007 6.4217e+007 6.43296e+007 6.44768e+007 6.36843e+007 6.43666e+007 6.38175e+007 6.38372e+007 6.42913e+007 6.43285e+007 6.40949e+007

Количество проходов по массиву для 16000 элементов: 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999 15999

Время сортировки для 32000 элементов: 1.512 1.502 1.514 1.483 1.498 1.503 1.5 1.501 1.507 1.497 1.491 1.524 1.492 1.507 1.502 1.51 1.498 1.529 1.494 1.525

Количество перестановок для 32000 элементов: 2.55494e+008 2.5625e+008 2.56896e+008 2.54344e+008 2.55727e+008 2.56655e+008 2.55803e+008 2.5482e+008 2.55655e+008 2.55693e+008 2.5594e+008 2.58205e+008 2.55409e+008 2.57238e+008 2.5453e+008 2.56036e+008 2.54986e+008 2.55607e+008 2.55458e+008 2.57455e+008

Количество проходов по массиву для 32000 элементов: 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999 31999

Время сортировки для 64000 элементов: 6.031 6.053 6.004 6.025 6.033 6.032 5.985 6.341 6.102 6.083 6.477 6.41 6.201 6.1 6.245 6.397 6.12 6.083 6.042 6.15

Количество перестановок для 64000 элементов: 1.02238e+009 1.0264e+009 1.02295e+009 1.02336e+009 1.02645e+009 1.02478e+009 1.02492e+009 1.02648e+009 1.02075e+009 1.02674e+009 1.02726e+009 1.02047e+009 1.02305e+009 1.02239e+009 1.01938e+009 1.02543e+009 1.02148e+009 1.0223e+009 1.01944e+009 1.02288e+009

Количество проходов по массиву для 64000 элементов: 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999 63999

Время сортировки для 128000 элементов: 24.493 24.45 24.285 24.412 24.257 24.309 24.174 24.152 24.253 24.155 24.068 24.134 24.27 25.197 25.855 26.453 25.819 24.746 24.524 24.519

Количество перестановок для 128000 элементов: 4097711865 4117541542 4091810689 4091683938 4087168705 4097350901 4084062427 4089592953 4093021567 4091766579 4082747362 4099489703 4100394075 4099711720 4094367421 4096311795 4096739417 4112028035 4093948492 4090365641

Количество проходов по массиву для 128000 элементов: 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999 127999

Из данных получаются следующие **графики**:



# Заключение.

Алгоритм сортировки вставками – это понятный, но неэффективный метод для работы с большими объемами данных. Однако он может пригодиться при обработке небольших массивов, особенно если они частично отсортированы. Реализация этого алгоритма на C++ с измерением времени выполнения и построением графиков позволила наглядно проанализировать его поведение и подтвердить теоретические оценки сложности. В процессе работы были получены стабильные результаты, что дало возможность сделать выводы о пригодности метода в различных условиях. Несмотря на свои ограничения, сортировка вставками остается полезным учебным примером алгоритмических подходов к упорядочиванию данных.