



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ДИСЦИПЛИНА «Анализ алгоритмов»

Лабораторная работа № 3

Тема Алгоритмы сортировки

Студент Михаил Коротыч

Группа ИУ7-55Б

Преподаватели Волкова Л. Л., Строганов Ю. В.

Москва.
2021 г.

Оглавление

Введение	4
1 Аналитическая часть	5
1.1 Сортировка пузырьком	5
1.2 Сортировка вставками	5
1.3 Быстрая сортировка	5
1.4 Вывод	6
2 Конструкторская часть	7
2.1 Схемы алгоритмов	7
2.2 Трудоёмкость алгоритмов	13
2.2.1 Сортировка пузырьком	13
2.2.2 Сортировка вставками	14
2.2.3 Быстрая сортировка	14
2.3 Вывод	15
3 Технологическая часть	16
3.1 Выбор ЯП	16
3.2 Сведения о модулях программы	16
3.3 Листинг кода	16
3.4 Вывод	20

4	Исследовательская часть	21
4.1	Примеры работы программы	21
4.2	Технические характеристики	21
4.3	Время выполнения алгоритмов	22
4.4	Вывод	24
	Заключение	25
	Список литературы	26

Введение

На сегодняшний день существует не одна вариация алгоритмов сортировки. Все они различаются по скорости и по объёму необходимой памяти.

Цель данной лабораторной работы - обучение расчёту трудоёмкости алгоритмов.

Алгоритмы сортировки часто применяются в практике программирования. В том числе в областях, связанных с математикой, физикой, компьютерной графикой и т. д.

Задачи лабораторной работы:

- изучить алгоритмы сортировок;
- дать теоретическую оценку алгоритмам сортировки;
- реализовать три алгоритма сортировки на одном из языков программирования;
- сравнить алгоритмы сортировок.

1 | Аналитическая часть

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

1.2 Сортировка вставками

На каждом шаге выбирается один из элементов неотсортированной части массива (максимальный или минимальный) и помещается на нужную позицию в отсортированную часть массива.

1.3 Быстрая сортировка

Общая идея алгоритма состоит в следующем:

1. выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность;

2. сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньше опорного», «равные» и «большие»;
3. для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньше опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения

1.4 Вывод

В данном разделе были рассмотрены три алгоритма сортировки.

2 | Конструкторская часть

2.1 Схемы алгоритмов

На рисунке (2.1) приведена схема классического алгоритма сортировки пузырьком.

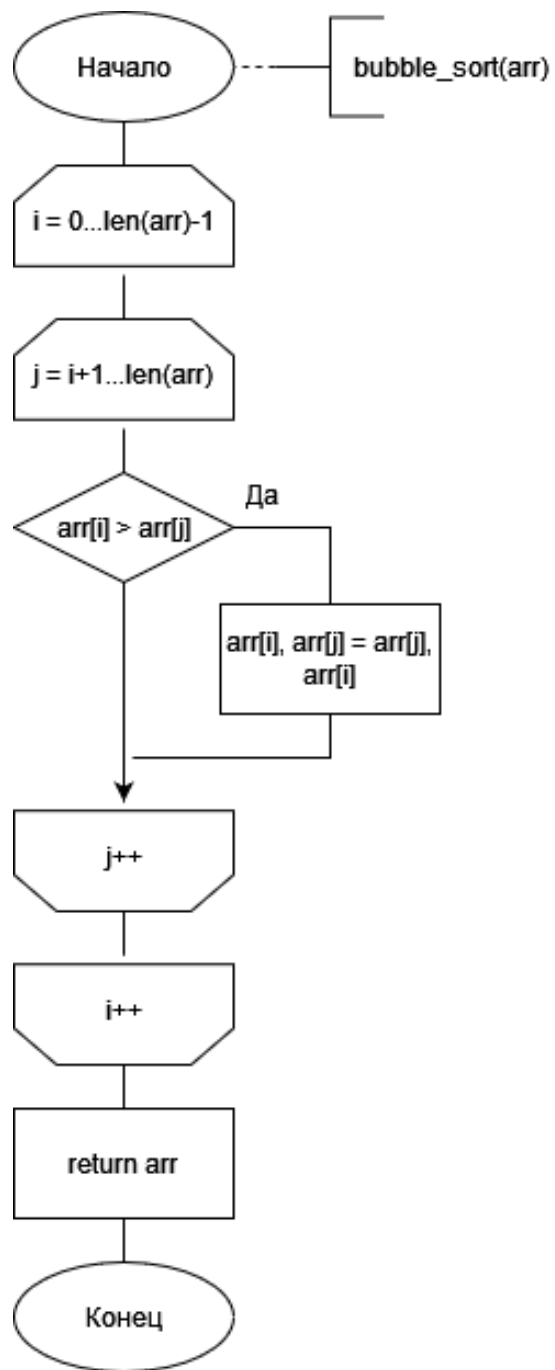


Рис. 2.1: Схема алгоритма сортировки пузырьком

На рисунке (2.2) приведена схема алгоритма сортировки вставками.

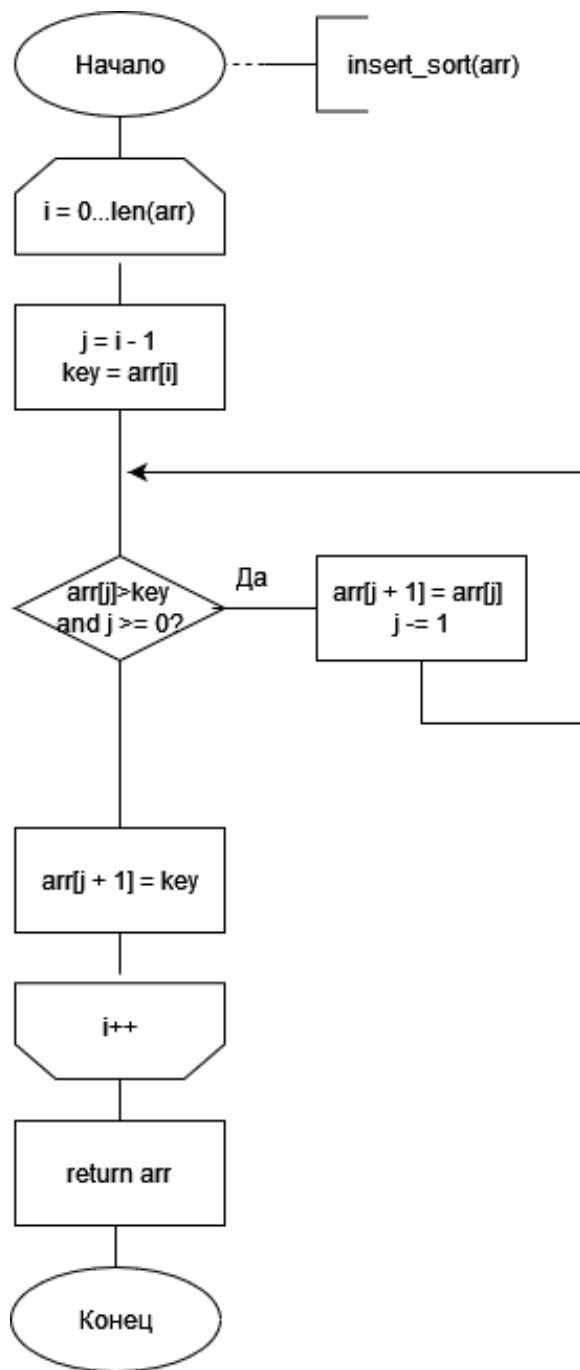


Рис. 2.2: Схема алгоритма сортировки вставками

На рисунке (2.3) приведена схема быстрой сортировки.

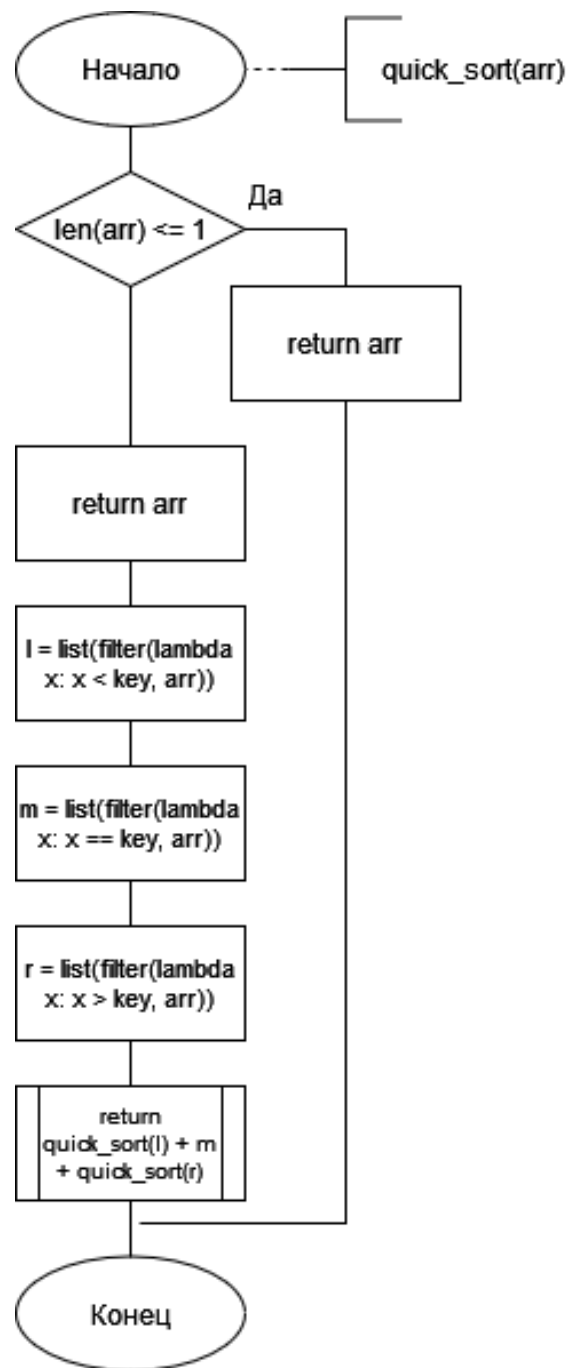


Рис. 2.3: Схема алгоритма быстрой сортировки

2.2 Трудоёмкость алгоритмов

Введем модель трудоёмкости для оценки алгоритмов:

1. базовые операции стоимостью 1: $+$, $-$, $*$, $/$, $=$, $==$, $<=$, $>=$, $!=$, $+=$, $||$, $++$, $-$, получение полей класса;
2. оценка трудоёмкости цикла: $F_{\text{ц}} = a + N^*(a + F_{\text{тела}})$, где a - условие цикла;
3. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будут приведены оценки трудоёмкости алгоритмов. Построчная оценка трудоёмкости сортировки пузырьком с флагом (Табл. 2.1).

2.2.1 Сортировка пузырьком

Лучший случай: массив отсортирован; не произошло ни одного обмена за 1 проход \rightarrow выходим из цикла

Трудоёмкость: $1 + 2n * (1 + 2n * 4) = 1 + 2n + 16n * n = O(n^2)$

Худший случай: массив отсортирован в обратном порядке; в каждом случае происходил обмен

Трудоёмкость: $1 + 2n * (1 + 2n * (4 + 5)) = O(n^2)$

2.2.2 Сортировка вставками

Табл. 2.1 Построчная оценка веса

Код	Вес
for (int i = 1; i < a.size(); i++)	1 + n * 2
int key = a[i]	2
int j = i - 1	2
while (j >= 0 and a[j] > key):	n * 4
a[j + 1] = a[j]	4
j -= 1	1
a[j + 1] = key	3

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.

Трудоёмкость: $T(n) = 1 + 2n * (2 + 2 + 3) = 2n * 7 = 14n + 1 = O(n)$

Худший случай: массив отсортирован в обратном нужном порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.

Трудоёмкость: $T(n) = 1 + n * (2 + 2 + 4n * (4 + 1) + 3) = 2n * n + 7n + 1 = O(n^2)$

2.2.3 Быстрая сортировка

Лучший случай: сбалансированное дерево вызовов^[2] $O(n * \log(n))$ В наиболее благоприятном случае процедура PARTITION приводит к двум подзадачам, размер каждой из которых не превышает $\frac{n}{2}$, поскольку размер одной из них равен $\frac{n}{2}$, а второй $\frac{n}{2} - 1$. В такой ситуации быстрая сортировка работает намного производительнее, и время её работы описывается следующим рекуррентным соотношением: $T(n) = 2T(\frac{n}{2}) + O(n)$ - где мы не обращаем внимания на неточность, связанную с игнорированием функций “пол” и “потолок”, и вычитанием 1. Это рекуррентное

соотношение имеет решение: $T(n) = O(n \lg n)$. При сбалансированности двух частей разбиения на каждом уровне рекурсии мы получаем асимптотически более быстрый алгоритм.

Фактически, любое разбиение, характеризующееся конечной константой пропорциональности, приводит к образованию дерева рекурсии высотой $O(\lg n)$ со стоимостью каждого уровня, равной $O(n)$. Следовательно, при любой постоянной пропорции разбиения полное время работы быстрой сортировки составляет $O(n \lg n)$.

Худший случай: несбалансированное дерево ^[2] $O(n^2)$ Поскольку рекурсивный вызов процедуры разбиения, на вход которой подаётся массив размером 0, приводит к немедленному возврату из этой процедуры без выполнения каких-либо операций, то $T(0) = O(1)$. Таким образом, рекуррентное соотношение, описывающее время работы процедуры в указанном случае, записывается следующим образом: $T(n) = T(n - 1) + T(0) + O(n) = T(n - 1) + O(n)$. Интуитивно понятно, что при суммировании промежутков времени, затрачиваемых на каждый уровень рекурсии, получается арифметическая прогрессия, что приводит к результату $O(n^2)$.

2.3 Вывод

Сортировка пузырьком: лучший - $O(n)$, худший - $O(n^2)$.

Сортировка вставками: лучший - $O(n)$, худший - $O(n^2)$.

Быстрая сортировка: лучший - $O(n * \lg n)$, худший - $O(n^2)$.

3 | Технологическая часть

3.1 Выбор ЯП

Был выбран C++ в качестве языка программирования, потому как он достаточно удобен и гибок. Среда разработки - Visual Studio Code.

Время работы алгоритмов было замерено с помощью функции `GetCPUTime` из библиотеки `Windows.h`.

3.2 Сведения о модулях программы

Программа состоит из:

- `main.cpp` - файл, в котором находятся функции сортировки.
- `GetCPUTime.c` - файл, в котором находится функция замера времени.

3.3 Листинг кода

В листингах 3.1 - 3.3 приведены реализации алгоритмов сортировки массивов.

Листинг 3.1: Сортировка пузырьком

```
1 std::vector<int> bubble_sort(std::vector<int> array)
2 {
3     for (int i = 0; i < array.size() - 1; ++i)
4         for (int j = i + 1; j < array.size(); ++j)
5             if (array[i] > array[j])
6                 std::swap(array[i], array[j]);
7
8     return array;
9 }
```

Листинг 3.2: Сортировка вставками

```
1 std::vector<int> insert_sort(std::vector<int> array)
2 {
3     for (int i = 0; i < array.size(); ++i)
4     {
5         int j = i - 1;
6         int key = array[i];
7         for (; array[j] > key && j >= 0; --j)
8             array[j + 1] = array[j];
9
10        array[j + 1] = key;
11    }
12
13    return array;
14 }
```

Листинг 3.3: Быстрая сортировка

```
1 void quick_sort(std::vector<int> &array, int low, int high)
2 {
3     int i = low;
4     int j = high;
5     int pivot = array[(i + j) / 2];
6
7     while (i <= j)
8     {
9         while (array[i] < pivot)
10             i++;
11         while (array[j] > pivot)
12             j--;
13         if (i <= j)
14         {
15             std::swap(array[i], array[j]);
16             i++;
17             j--;
18         }
19     }
20
21     if (j > low)
22         quick_sort(array, low, j);
23     if (i < high)
24         quick_sort(array, i, high);
25 }
```

В листинге 3.4 приведена реализация функции замера процессорного времени.

Листинг 3.4: Функция замера процессорного времени

```
1 #include "getCPUTime.h"
2 /*
3  * Author:   David Robert Nadeau
4  * Site:     http://NadeauSoftware.com/
5  * License:  Creative Commons Attribution 3.0 Unported
6              License
7              http://creativecommons.org/licenses/by/3.0/deed
8              .en_US
9  */
10 ...
11 double getCPUTime() {
12 #if defined(_WIN32)
13     FILETIME createTime;
14     FILETIME exitTime;
15     FILETIME kernelTime;
16     FILETIME userTime;
17     if (GetProcessTimes(GetCurrentProcess(), &createTime, &
18         exitTime, &kernelTime, &userTime) != -1) {
19         ULARGE_INTEGER li = {{userTime.dwLowDateTime, userTime.
20             dwHighDateTime }};
21         return li.QuadPart / 10000000.;
22     }
23 }
24 ...
25 return -1;          /* Failed. */
26 }
```

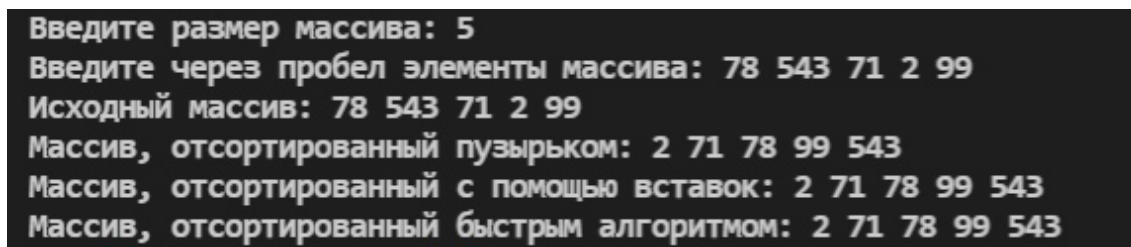
3.4 Вывод

В данной части был описан выбор языка программирования, приведена информация о модулях программы и отображены листинги реализаций трёх алгоритмов сортировки массивов, функции замера процессорного времени.

4 | Исследовательская часть

4.1 Примеры работы программы

На рисунке (4.1) приведена демонстрация работы программы.



Введите размер массива: 5
Введите через пробел элементы массива: 78 543 71 2 99
Исходный массив: 78 543 71 2 99
Массив, отсортированный пузырьком: 2 71 78 99 543
Массив, отсортированный с помощью вставок: 2 71 78 99 543
Массив, отсортированный быстрым алгоритмом: 2 71 78 99 543

Рис. 4.1: Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Windows 10 Домашняя для одного языка 21H1;
- Оперативная память: 8 ГБ;
- Процессор: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99 GHz.

Тестирование проводилось на ноутбуке при включённом режиме производительности. Во время тестирования ноутбук был нагружен только системными процессами.

4.3 Время выполнения алгоритмов

Произведено измерение времени работы алгоритмов на случайно сгенерированных, отсортированных по возрастанию и убыванию массивах. Для замера времени была использована функция `GetTimeCPU`.

Измерение было произведено 50 раз и результат был усреднён.

На рисунках 4.2 - 4.4 приведены графики, отображающие время работы алгоритмов в наносекундах от длины массивов для неупорядоченных, отсортированных по возрастанию, отсортированных по убыванию массивов соответственно.

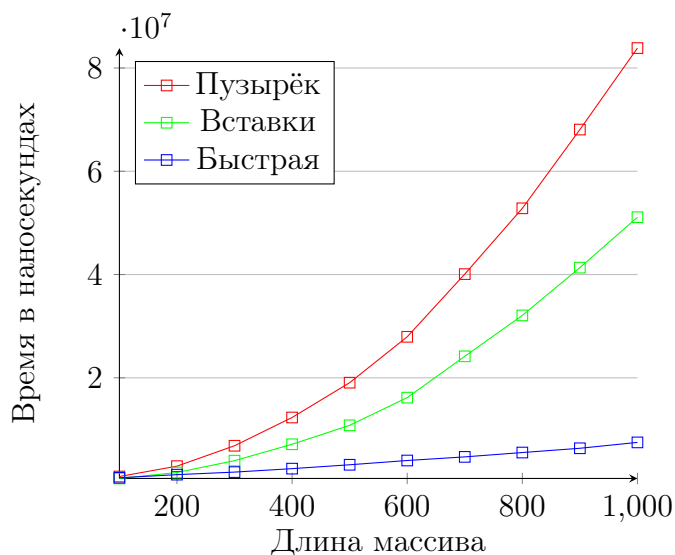


Рис. 4.2: Сравнение времени сортировок неупорядоченных массивов

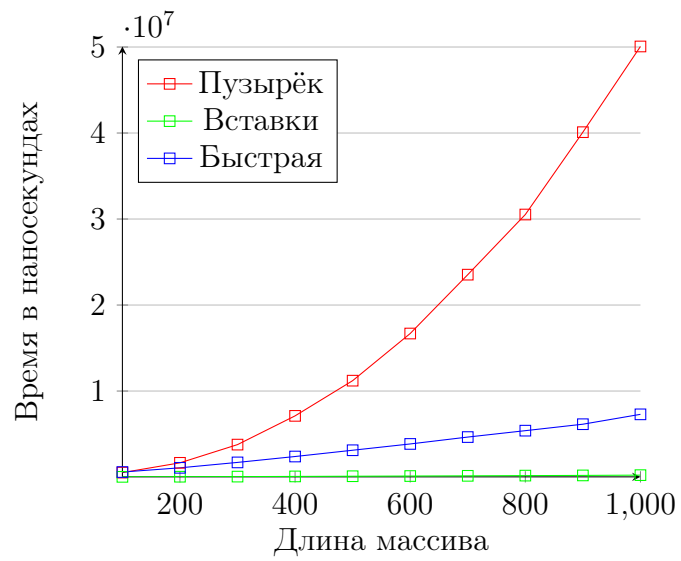


Рис. 4.3: Сравнение времени сортировок отсортированных по возрастанию массивов

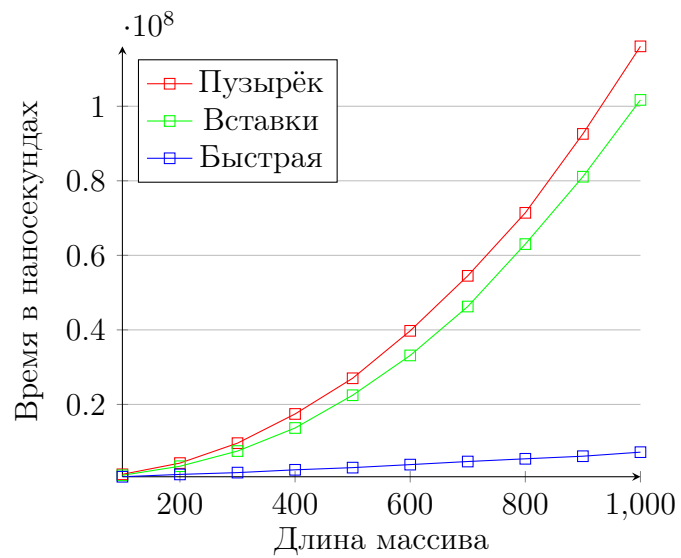


Рис. 4.4: Сравнение времени сортировок отсортированных по убыванию массивов

4.4 Вывод

Были протестированы алгоритмы сортировки на массивах размерами 100...1000 с шагом 100. Рассмотрены неупорядоченные, отсортированные по возрастанию, отсортированные по убыванию массивы.

Экспериментально было подтверждено, что при сортировке отсортированных по возрастанию массивов сортировка вставками показывает наилучший результат.

Исходя из графиков наглядно видно, что сортировка пузырьком является самой медленной.

Заключение

В ходе выполнения данной лабораторной работы были реализованы три алгоритма сортировки: сортировка пузырьком, сортировка вставками и быстрая сортировка. Был проведён анализ каждого алгоритма и измерено время работы алгоритмов для массивов разных размеров. Была оценена трудоёмкость алгоритмов.

Список литературы

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 7. Быстрая сортировка // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005.
2. Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.
3. CPP Reference [Электронный ресурс]. Режим доступа: <https://en.cppreference.com/w/>
4. Процессор Intel Core 17-85500 [Электронный ресурс] Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html>
5. Windows 10 [Электронный ресурс] Режим доступа: <https://www.microsoft.com/ru-ru/windows/get-windows-10>