МГТУ им. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Алгоритмы сортировки

Работу выполнила: Гаврилов Дмитрий, ИУ7-56Б

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

Москва, 2019

Оглавление

Введение

На данный момент существует огромное количество вариаций сортировок. Эти алгоритмы необходимо уметь сравнивать, чтобы выбирать наилучше подходящие в конкретном случае.

Эти алгоримы оцениваются по:

- Времени быстродействия

Целью данной лабораторной работы является изучение применений алгоритмов сортировки, обучение расчету трудоемкости алгоритмов.

1 | Аналитическая часть

1.1 Описание алгоритмов

Сортировка массива — одна из самых популярных операций над массивом. Алгоритмы реализуют упорядочивание элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки **Область применения:**

- физика,
- математика,
- экономика,
- итд.

1.1.1 Сортировка пузырьком

Алгоритм проходит по массиву n-1 раз или до тех пор, пока массив не будет полностью отсортирован. В каждом проходе элементы попарно сравниваются и, при необходимости, меняются местами. При каждом проходе алгоритма по внутреннему циклу, очереднй наибольший элемент ставится на своё место в конец неотсортированного массива. Таким образом наибольшие элементы "всплывают"как пузырек.

1.1.2 Сортировка вставками

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

1.1.3 Быстрая сортировка

Массив разбивается на два (возможно пустых) подмассива. Таких, что в одном подмассиве каждый элемент меньше либо равен опорному, и при этом не превышает любой элемент второго подмассива. Опорный элемент вычислыется в ходе процедуры разбиения. Подмассивы сортируются с помощью рекурсивного вызова процедуры быстрой сортировки. Поскольку подмассивы сортируются на месте, для их объединения не трубуются никакие действия.

2 | Конструкторская часть

2.1 Схемы алгоритмов

В данном разделе будут рассмотрены схемы алгоритмов пузырьком с флагом (2.1), сортировки вставками (2.2), быстрой сортировки (2.3).

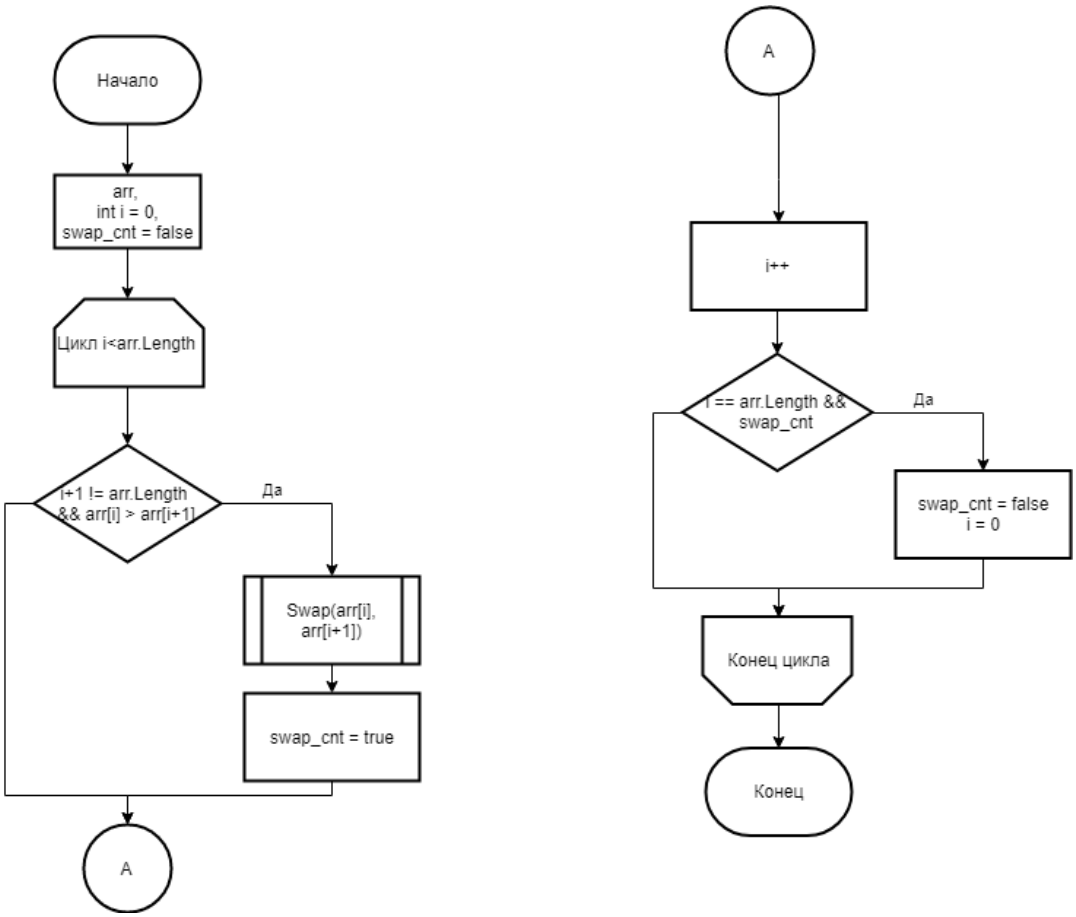

Рис. 2.1: Схема алгоритма сортировки пузырьком с флагом

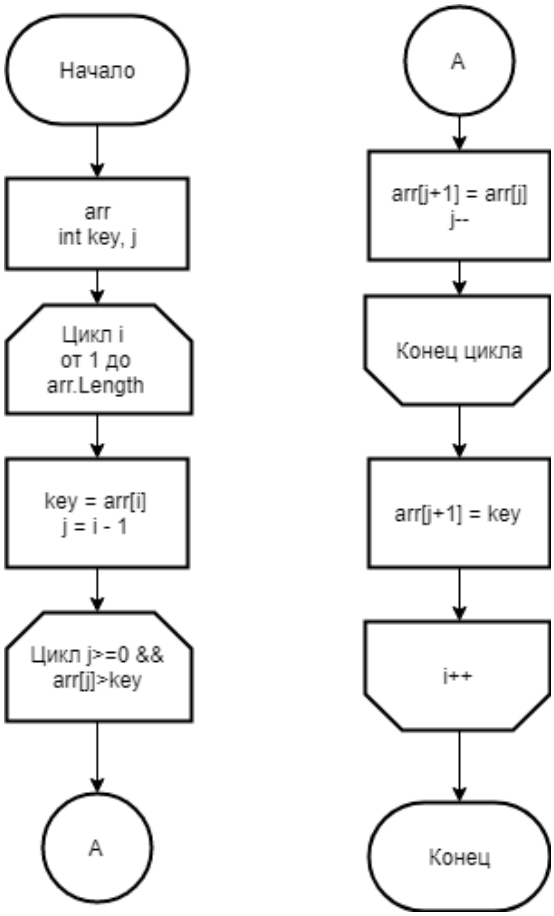

Рис. 2.2: Схема алгоритма сортировки вставками

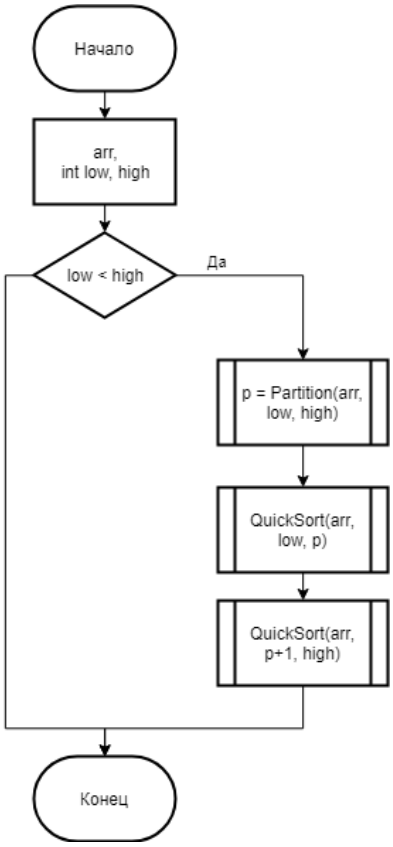

Рис. 2.3: Схема алгоритма быстрой сортировки

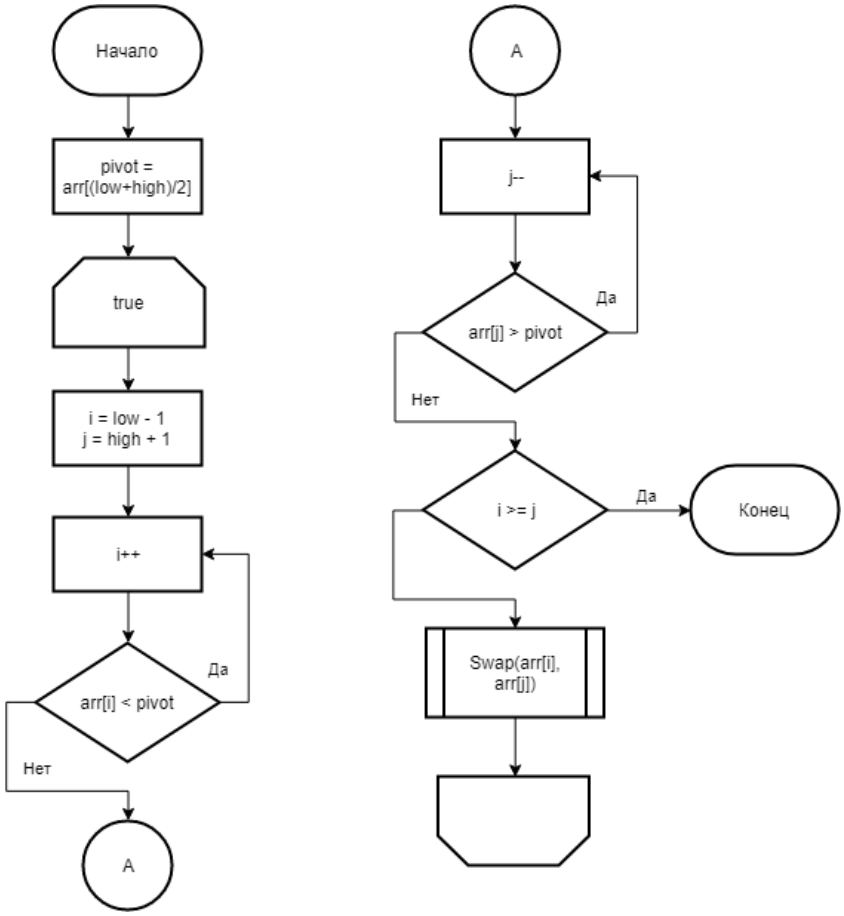

Рис. 2.4: Схема функции partition

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса

2. оценка трудоемкости цикла: Fц = a + N*(a + Fтела), где a - условие цикла

3. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будет приведены оценки трудоемкости алгоритмов. Построчная оценка трудоемкости сортировки пузырькм с флагом (Табл. 2.1).

2.2.1 Сортировка пузырьком с флагом

Табл. 2.1 Построчная оценка веса

Код	Вес
int i = 0;	1
bool swapcnt = false;	1
while (i < arr.Length)	2
{	0
if(i + 1 != arr.Length && arr[i] > arr[i + 1])	7
{	0
Swap(ref arr[i], ref arr[i + 1]);	2+3
swapcnt = true;	1
}	0
i++;	1
if (i == arr.Length && swapcnt)	3
{	0
swapcnt = false;	1
i = 0;	1
}	0
}	0

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $1 + 1 + n * (2 + 7 + 1 + 3) + 2 = 13n + 4 = O(n)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость: $1+1+n*(n*(7+5+1+3)+1+1)+2 = n*(16n+2)+4 = 16n^2 + 2n + 4 = O(n^2)$

2.2.2 Сортировка вставками

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.
Трудоемкость: $T(n) = 3n + ((2+2+4+2) * (n-1)) = 3n + 10(n-1) = 13n - 10 = O(n)$

Худший случай: массив отсортирован в обратном нужному порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.
Трудоемкость: $T(n) = 3n+(2+2)(n-1)+4\left(\frac{n(n+1)}{2} - 1\right)+5\frac{n(n-1)}{2}+3(n-1) = 3n+4n-4+2n^2+2n-4+2.5n^2-2.5n+3n-3 = 4.5n^2+9.5n-11 = O(n^2)$

2.2.3 Быстрая сортировка

Лучший случай: сбалансированное дерево вызовов $O(n * log(n))$ 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 ее 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 ; $T(n) = O(nlgn)$. 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 $O(lgn)$ со стоимостью каждого уровня, равной $O(n)$. Следовательно, прилюбой постоянной пропорции разбиения полное время работы быстрой сортировки составляет $O(nlgn)$.

Худший случай: несбалансированное дерево $O(n^2)$ Поскольку рекурсивный вызов процедуры разбиения, на вход которой подается массив размером 0, приводит к немедленному возврату из этой процедуры без выполнения каких-ли-бо операций, $T(0) = O(1)$. 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 - $O(nlgn)$, худший - $O(n^2)$

При этом сортировка вставками быстрее пузырька с флагом в худшем случае т.к. имеет меньший коэффициент. Вставки $4.5n^2$, пузырек $16n^2$.

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран Java, а средой разработки Intellij IDEA. Время работы алгоритмов было замерено с помощью класса Instant.

3.2 Описание структуры ПО

3.3 Сведения о модулях программы

Программа состоит из:

- Main.java - главный файл программы, в котором располагается точка входа в программу и функция замера времени.
- SortUtils.java - файл класса sorting, в котором находятся алгоримы сортировки
- RandomArrayFactory.java - файл-фабрика для генерации различных массивов

3.4 Листинг кода

Листинг 3.1: Алгоритм сортировки пузырьком

```
1 public static void bubbleSort(int[] array) {
2     boolean sorted = false;
```

```
3       int temp;
4       while(!sorted) {
5           sorted = true;
6           for (int i = 0; i < array.length - 1; i++) {
7               if (array[i] > array[i+1]) {
8                   temp = array[i];
9                   array[i] = array[i+1];
10                  array[i+1] = temp;
11                  sorted = false;
12              }
13          }
14      }
15  }
```

Листинг 3.2: Алгоритм сортировки вставками

```
public static void insertionSort(int[] array) {
    for (int i = 1; i < array.length; i++) {
        int current = array[i];
        int j = i - 1;
        while(j >= 0 && current < array[j]) {
            array[j+1] = array[j];
            j--;
        }

        array[j+1] = current;
    }
}
```

Листинг 3.3: Алгоритм быстрой сортировки

```
public static void quickSort(int[] array, int begin, int
    end) {
    if (end <= begin) return;
    int pivot = partition(array, begin, end);
    quickSort(array, begin, pivot-1);
    quickSort(array, pivot+1, end);
}
```

Листинг 3.4: Алгоритм поиска опорного элемента

```
public static int partition(int[] array, int begin, int end
    ) {
    int pivot = end;

    int counter = begin;
    for (int i = begin; i < end; i++) {
        if (array[i] < array[pivot]) {
            int temp = array[counter];
            array[counter] = array[i];
            array[i] = temp;
            counter++;
        }
    }
    int temp = array[pivot];
    array[pivot] = array[counter];
```

```
15      array[counter] = temp;
16
17      return counter;
18  }
```

3.5 Вывод

В данном разделе были представлены реализации алгоритмов сортировки пузырьком, вставками и быстрой сортировки.

4 | Исследовательская часть

Был проведен замер времени работы каждого из алгоритмов.

4.1 Примеры работы программы

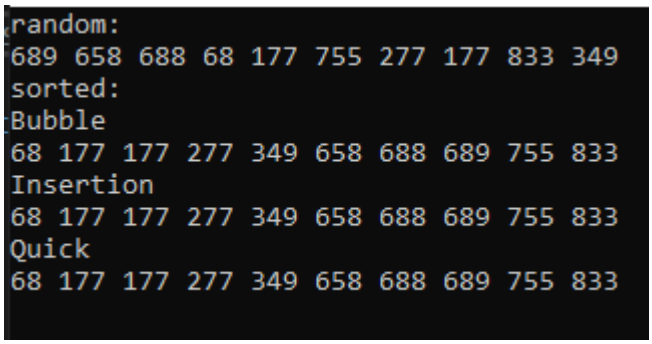

Рис. 4.1: Сортировка массива, заполненного случайными числами

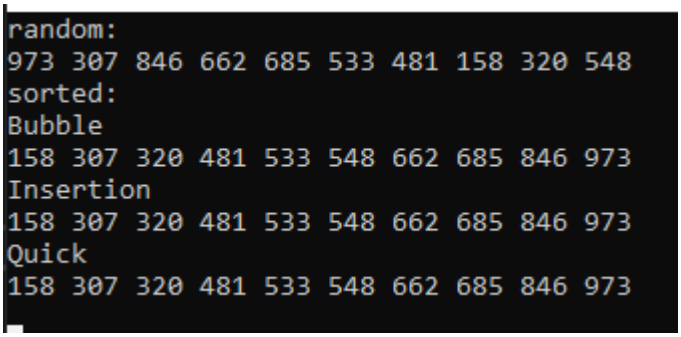

Рис. 4.2: Сортировка массива, заполненного случайными числами

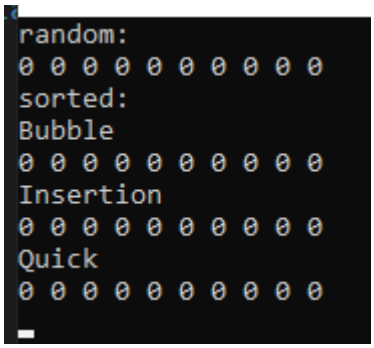

Рис. 4.3: Сортировка массива, заполненного одинаковыми числами

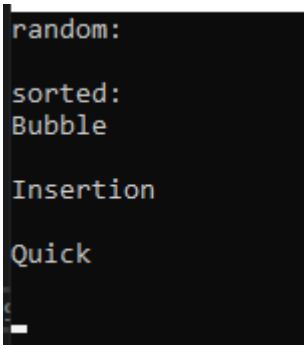

Рис. 4.4: Сортировка пустого массива

4.2 Эксперимент

В рамках данного эксперимента было произведено сравнение времени выполнения трех алгоритмов в лучшем/худшем/случайном случае заполения массива. При длине массивов от 100 до 1000 элементов с шагом 100. На предоставленных ниже графиках Рис. 4.1, Рис. 4.2, Рис. 4.3 по оси l идет длина массива, а по оси t - время сортировки в милисекундах.

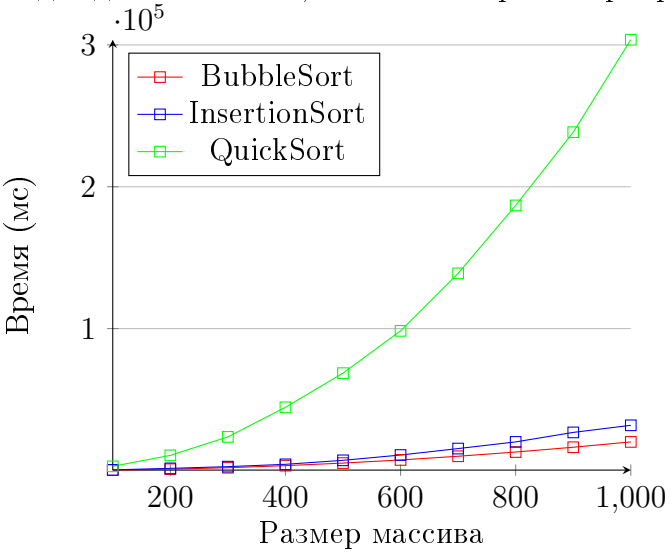

Рис. 4.1: Сравнение времени для отсортированного массива

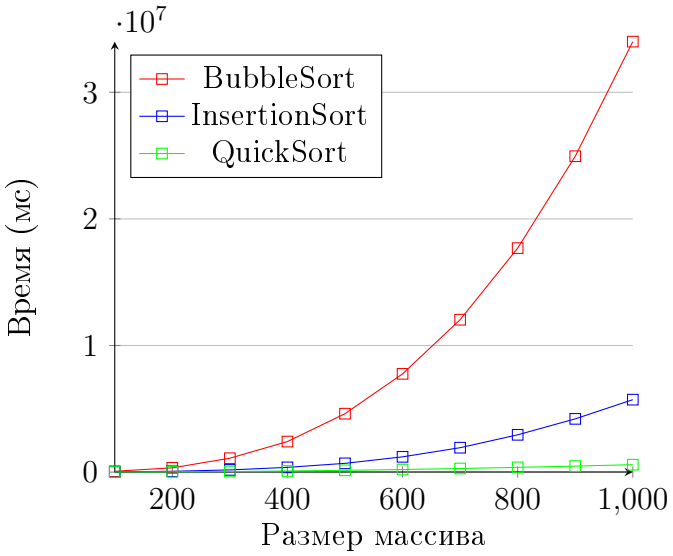

Рис. 4.2: Сравнение времени для отсортированного массива в обратном порядке

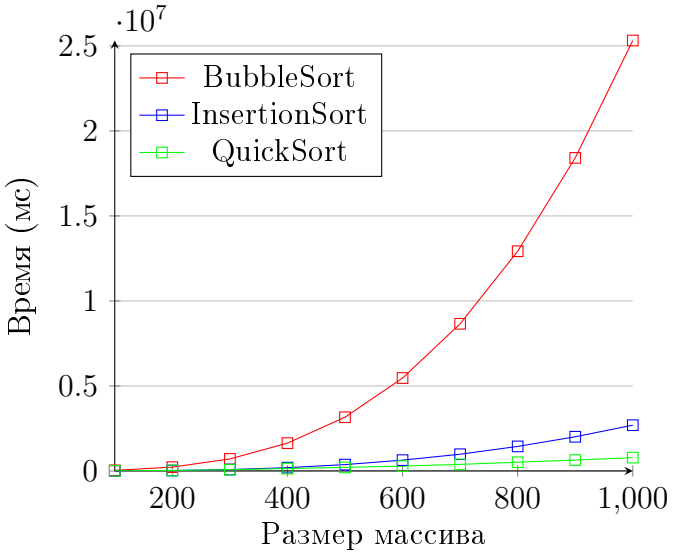

Рис. 4.3: Сравнение времени при случайном заполнении массива

4.3 Вывод

По результатам тестирования выявлено, что все рассматриваемые алгоритмы реализованы правильно. Самым быстрым алгоритмом, при использовании случайного заполнения, оказался алгоритм быстрой сортировки, а самым медленным — алгоритм сортировки пузырьком.

Заключение

В ходе работы были изучены алгоритмы сортировки массива: пузырьком с флагом, вставки, быстрая сортировка. Выполнено сравнение всех рассматриваемых алгоритмов. В ходе исследования был найден оптимальный алгоритм. Изучены зависимости выполнения алгоритмов от длины массива. Также реализован программный код продукта.

Список литературы

1. Кормен Т. Алгоритмы: построение и анализ [Текст] / Кормен Т. - Вильямс, 2014. - 198 с. - 219 с.