**

Государственное образовательное учреждение высшего профессионального образования «Московский Государственный Технический Университет

имени Н.Э. Баумана»

**ОТЧЕТ**

По лабораторной работе №3

По курсу «Анализ алгоритмов»

Тема: «Трудоёмкость сортировок»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Студент: | | | Жарова Е.А. | |
|  | | Группа | | | ИУ7-51 | |
|  | |  | | |  | |
|  | |  | |  | | |
|  | | Москва, 2017 | | |  | |

**Оглавление**

[Постановка задачи 1](#_Toc507877378)

[Описание модели вычислений 1](#_Toc507877379)

[Сортировка вставками 2](#_Toc507877380)

[1. Описание 2](#_Toc507877381)

[2. Реализация 2](#_Toc507877382)

[3. Теоретическая оценка 2](#_Toc507877383)

[Сортировка «пузырьком» 2](#_Toc507877384)

[1. Описание 2](#_Toc507877385)

[2. Реализация 3](#_Toc507877386)

[3. Теоретическая оценка 3](#_Toc507877387)

[Быстрая сортировка 3](#_Toc507877388)

[1. Описание 3](#_Toc507877389)

[2. Реализация 4](#_Toc507877390)

[3. Теоретическая оценка 4](#_Toc507877391)

[Сравнение алгоритмов 4](#_Toc507877392)

[Заключение 6](#_Toc507877393)

# **Постановка задачи**

Изучить и реализовать алгоритмы сортировки: вставками, «пузырьком» и быстрая сортировка. Оценить трудоёмкость одного из алгоритмов сортировки с указанием лучшего и худшего случаев. Сравнить реализованные алгоритмы сортировки и сделать выводы.

# **Описание модели вычислений**

При оценке трудоёмкости мы будем пользоваться следующей моделью вычисления:

1. Операции, которые имеют трудоемкость “0”:

* логический переход по ветвлению;
* операции обращения к полю структуры/класса (->, .);
* объявление переменных

1. Операции, которые имеют трудоёмкость «1»:

* Арифметические операции {сложение(+), вычитание(-), умножение(\*), деление(/), битовый сдвиг(<<, >>), деление нацело, взятие остатка(%) };
* логические операции { и(&&), не(!), или(||) };
* операции сравнения {<, >, =, !=, >=, <=};
* операции присваивания {=, +=, -=, \*=, /=, %=};
* операция взятия индекса ([]);
* операции побитового И(&) и ИЛИ(|)
* унарный плюс и минус
* операции инкремента и декремента(постфиксные и префиксные) (++, --).

# **Сортировка вставками**

## Описание

Сортировка вставками — 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 - .

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей. O ( n 2 ) {\displaystyle O(n^{2})}

## Реализация

1. template <class T>
2. void insertionSort(T\* begin, T\* end)
3. {
4. T tmp;
5. T\* j;
6. for(T\* i = begin + 1; i < end; i++)
7. {
8. tmp = \*i;
9. j = i - 1;
10. while(j >= begin && tmp >= \*j)
11. {
12. \*(j + 1) = tmp;
13. j--;
14. }
15. }
16. }

## Теоретическая оценка

W – число заходов в цикл while.

f = 3 + 13(n-1) + 9W = 9W + 13n – 10

Лучший случай (массив отсортирован: W = 0):

Худший случай (массив отсортирован в обратную сторону: ):

# **Сортировка «пузырьком»**

## Описание

Сортировка «пузырьком» (сортировка простыми обменами) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма:

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N − 1 {\displaystyle N-1} N – 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде. Отсюда и название алгоритма). ( n 2 ) {\displaystyle (n^{2})}

## Реализация

1. template <class T>
2. void bubleSort(T\* begin, T\* end)
3. {
4. bool flag;
5. for (T \*i = end - 1; i >= begin; i--)
6. {
7. flag = true;
8. for (T \*j = begin + 1; j <= i; j++)
9. {
10. if (\*(j-1) > \*j)
11. {
12. T tmp = \*j;
13. \*j = \*(j-1);
14. \*(j-1) = tmp;
15. flag = false;
16. }
17. }
18. if(flag)
19. break;
20. }
21. }

## Теоретическая оценка

Трудоемкость тела внутреннего цикла оценивается, как O(1), потому что не зависит от количества элементов в массиве. В результате выполнения внутреннего цикла, наибольший элемент смещается в конец массива неупорядоченной части, поэтому через N таких вызовов массив в любом случае окажется отсортирован. Если же массив отсортирован, то внутренний цикл будет выполнен лишь один раз. Тогда в лучшем случае алгоритм будет иметь трудоёмкость:

В худшем и среднем случае:

# **Быстрая сортировка**

## Описание

Быстрая сортировка(сортировка Хоара, qsort) — широко известный алгоритм сортировки, разработанный английским информатиком Чарльзом Хоаром во время его работы в МГУ в 1960 году.

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O ( n log ⁡ n ) {\displaystyle O(n\log n)} обменов при упорядочении n {\displaystyle n} n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Общая идея алгоритма состоит в следующем:

* 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.
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие».
* 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, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

## Реализация

1. template <class T>
2. void quickSort(T\* begin, T\* end)
3. {
4. T \*i = begin, \*j = end, x = \*(i + (j - i) / 2);
5. do
6. {
7. while (\*i < x) i++;
8. while (\*j > x) j--;
9. if(i <= j)
10. {
11. if (\*i > \*j)
12. {
13. T tmp = \*j;
14. \*j = \*i;
15. \*i = tmp;
16. }
17. i++;
18. j--;
19. }
20. } while (i <= j);
21. if (i < end)
22. quickSort(i, end);
23. if (begin < j)
24. quickSort(begin, j);
25. }

## Теоретическая оценка

Лучший случай: с помощью выбранного опорного элемента массив будет делится ровно пополам (+- 1 элемент). Тогда глубина рекурсии будет равна. При этом на каждом уровне рекурсии суммарная трудоемкость разделения массива на две части будет равна O(n). Тогда трудоемкость алгоритма будет равна.

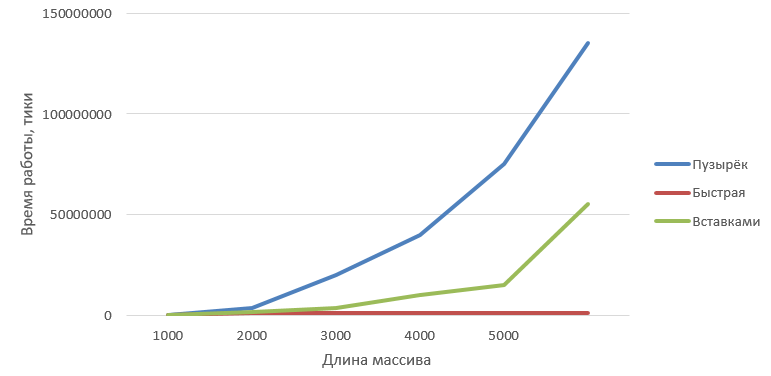
В среднем: временная сложность алгоритма составляет O(nlogn).

Худший случай: выбираемый опорный элемент равен минимальному или максимальному значению на текущем интервале, так как это приведет к тому, что при рекурсивном вызове sort длина массива сокращается не в два раза, а всего лишь на один элемент. Трудоемкость алгоритма быстрой сортировки для этого случая будет равна . В связи с этим существуют различные оптимизации, в том числе и по выбору барьерного элемента, который можно выбирать случайным образом или, например, выбирать средний по значению элемент из первого, последнего и среднего элементов текущего участка массива.

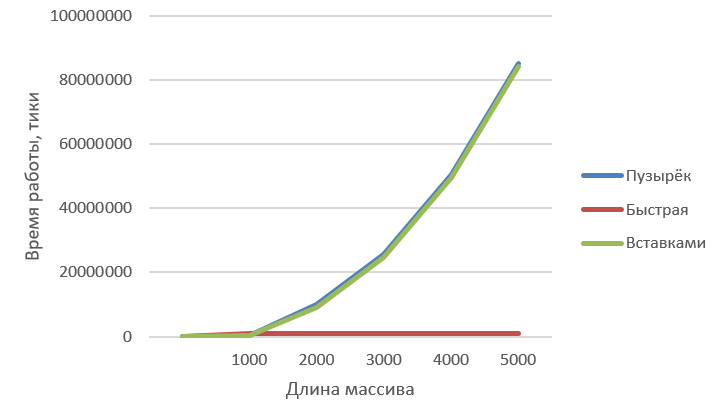
# **Сравнение алгоритмов**

Для сравнения алгоритмов были проведены тесты на отсортированных массивах, произвольных массивах и отсортированных в обратном порядке массивов различных длин: 100, 200, …, 5000.

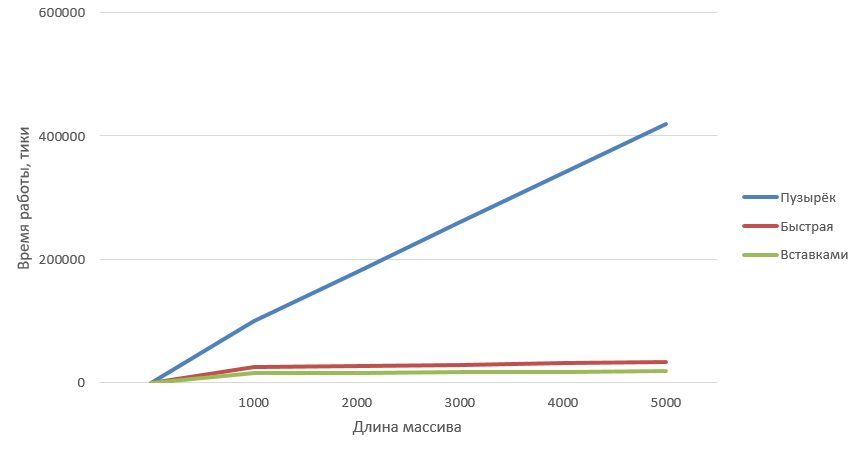
Анализ быстродействия для произвольных массивов:



Анализ быстродействия для отсортированных в обратном порядке массивов:



Анализ быстродействия для отсортированных массивов:

Выводы:

* Для отсортированного массива (лучший случай) алгоритмы сортировки «пузырьком» и вставками работают быстрее алгоритма быстрой сортировки.
* Для произвольного массива и массива отсортированного в обратном порядке время работы алгоритма зависит от длины массива квадратично (для сортировок вставками и «пузырьком») и линейно (для быстрой сортировки). Это подтверждает теоретическую оценку трудоёмкости данных алгоритмов.
* Когда массив имеет небольшое количество элементов и частично отсортирован, наиболее эффективным алгоритмом сортировки является алгоритм сортировки вставками. Быстрая сортировка из-за дополнительных затрат на обслуживание вызовов рекурсивной функции имеет низкую эффективность.

# **Заключение**

Были изучены и реализованы алгоритмы сортировки: вставками, «пузырьком», быстрая сортировка. Во время выполнения работы была произведена оценка трудоёмкости сортировки вставками и сравнение быстродействия реализованных алгоритмов.