МГТУ им. Н.Э. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Алгоритмы сортировки массивов

Работу выполнил: Луговой Дмитрий, ИУ7-51Б

Преподаватель: Волкова Л.Л.

Москва, 2019

Оглавление

Введение

Целью данной лабораторной работы является исследование существующих алгоритмов сортировки массивов и их трудоемкости.
Примем следующую модель вычислений:

1. Трудоемкость базовых операций
   Операции $+$, $-$, $*$, $/$, $\%$, $=$, $>$, $<$, $\leq$, $\geq$, $==$, $\neq$, $[]$, $+=$, $-=$ - имеют стоимость 1.

2. Трудоемкость условного перехода
   Условный переход имеет стоимость 0, при этом оцениваем расчет условия:

$$
\begin{array}{l}
if\ (n\ \%\ 2\ ==\ 1) \\
\{ \\
\quad // \text{ Тело 1} \\
\} \\
else \\
\{ \\
\quad // \text{ Тело 2} \\
\}
\end{array}
$$

$$
f_{if} = f_{\text{условия}} + \begin{cases} f_{\text{тело 1}}, & \text{при нечетном } N \\ f_{\text{тело 2}}, & \text{при четном } N \end{cases}
$$

3. Трудоемкость цикла for

   $f_{\text{цикла}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}})$,
   где N - число повторений цикла.

Задачи работы

Задачами данной лабораторной являются:

1. Реализовать следующие алгоритмы сортировки массивов:
    - Сортировка вставками
    - Сортировка расческой
    - Быстрая сортировка
2. Проанализировать трудоемкость данных алгоритмов
3. Провести эксперименты с замерами времени

1 | Аналитическая часть

В этом разделе содержатся описания алгоритмов сортировки массивов.

1.1 Сортировка вставками

Сортировка вставками (англ. Insertion sort) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Сортировка вставками делит массив на 2 части — отсортированную и неотсортированную. Из неотсортированной части извлекается текущий элемент. Поскольку другая часть массива отсортирована, то в ней достаточно быстро можно найти место для этого извлечённого элемента. Элемент вставляется куда нужно, в результате чего отсортированная часть массива увеличивается, а неотсортированная уменьшается.

1.2 Сортировка расческой

Сортировка расческой (англ. Comb sort) - это модификация пузырьковой сортировки. Производятся неоднократные прогоны по массиву, при которых сравниваются пары элементов. Если они неотсортированы друг относительно друга - то производится обмен. В результате крупные элементы мигрируют в конец массива, а небольшие по значению - в начало. В пузырьковой сортировке при каждом прогоне по массиву сравниваются соседние элементы. Здесь же сравниваются элементы, между которыми некоторое фиксированное расстояние. При каждом следующем прохождении расстояние уменьшается, пока не достигнет минимальной величины. Уменьшающееся расстояние между сравниваемыми элементами рассчитывается с помощью специальной величины, называемой фактором уменьшения. Длина массива делится на этот фактор, это и есть разрыв между индексами. После каждого прохода расстояние делится на фактор уменьшения и таким образом получается новое значение. В конце концов оно сужается до минимального значения - единицы, и массив просто досортировывается обычным "пузырьком". В результате практических тестов и теоретических исследований оптимальное значение для фактора уменьшения определено следующее: $\frac{1}{1-\frac{1}{e^{\varphi}}}$

1.3 Быстрая сортировка

Быстрая сортировка (англ. Quick sort) — один из самых известных и широко используемых алгоритмов сортировки.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, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

2 | Конструкторская часть

В этом разделе содержатся схемы алгоритмов сортировки массивов и подсчет трудоемкости.

2.1 Схемы алгоритмов

На Рис.2.1 представлена схема алгоритма сортировки массивов вставками.

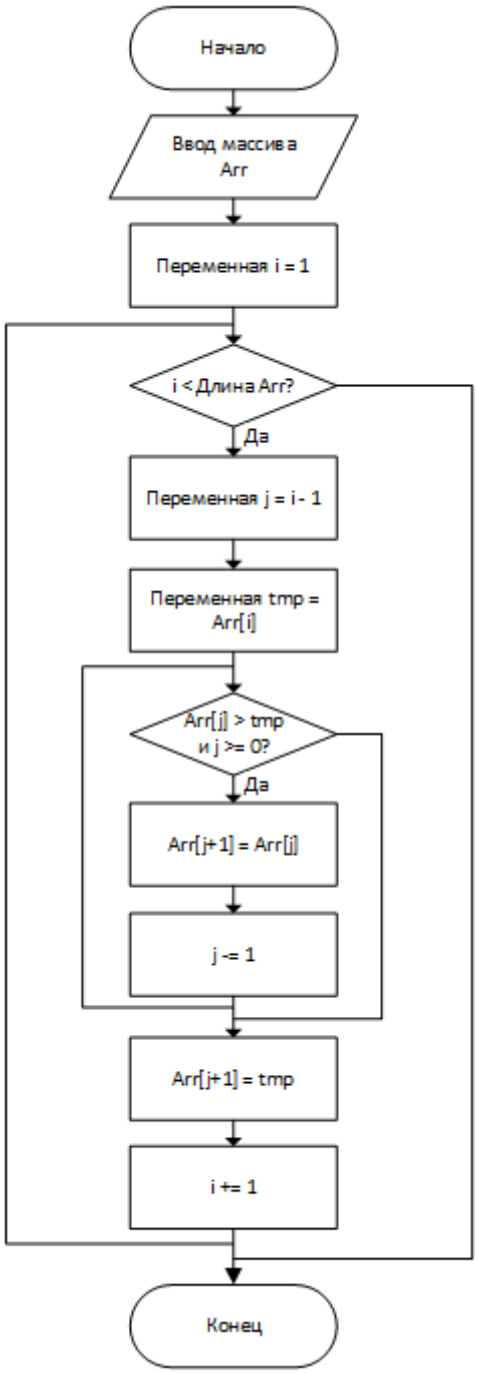

Рис. 2.1: Алгоритм сортировки вставками

На Рис.2.2 представлена схема алгоритма сортировки массивов расческой.

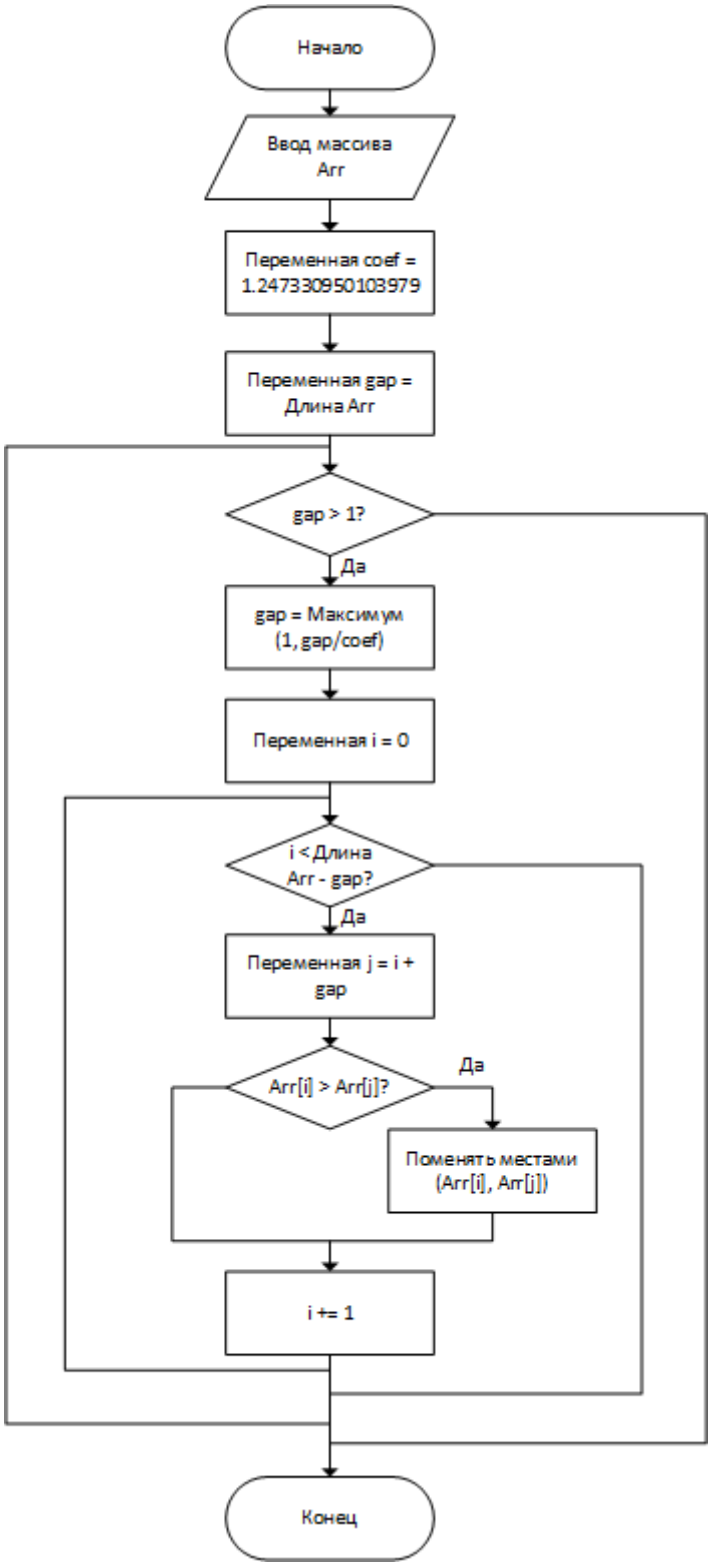

Рис. 2.2: Алгоритм сортировки расческой

На Рис.2.3 представлена схема алгоритма быстрой сортировки массивов.

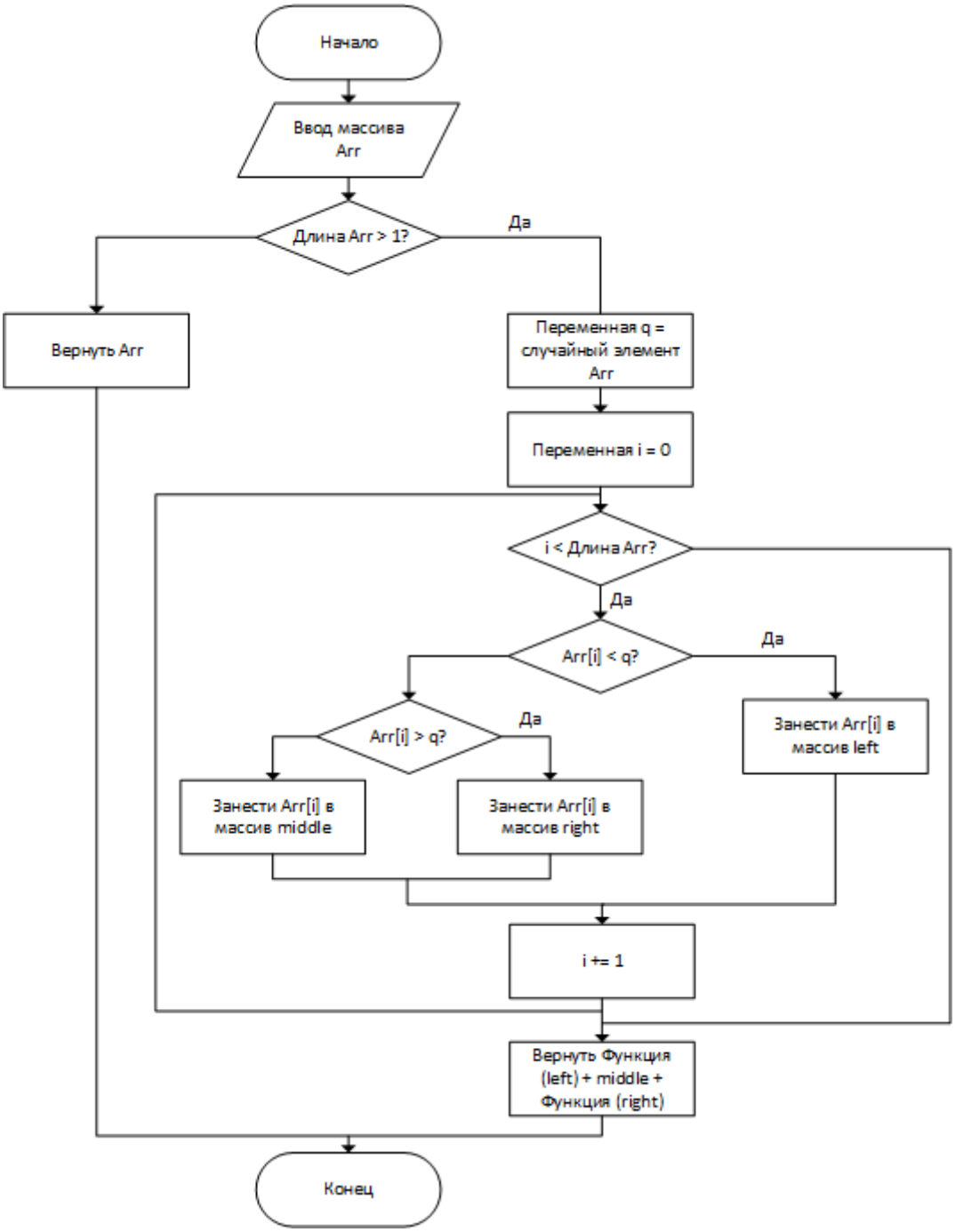

Рис. 2.3: Алгоритм быстрой сортировки

2.2 Расчет трудоемкости

Пусть задан массив размерности N. Используя модель вычислений, заданную ранее, произведем подсчет трудоемкости алгоритмов сортировки:

1. **Сортировка вставками**

   - **Лучший случай** - отсортированный массив. В этом случае внутренний цикл выполняться не будет:

   $$f_{\text{вставками}} = 2 + (N-1)(2+2+2+2+3) = 11N - 9 \approx 11N$$

   - **Худший случай** - обратно отсортированный массив.В этом случае внутренний цикл будет выполняться максимально большое количество раз.

   $$f_{\text{вставками}} = 2+(N-1)(2+2+2+3)+\frac{4+4+1+(4+4+1)(N-1)}{2}(N-1) = \\ = 4.5N^2 + 4.5N - 7 \approx N^2$$

2. **Сортировка расческой** [1].

   - **Лучший случай** - полностью отсортированный массив, как и для сортировки пузырьком, модификацией которого она является. В отличие от сортировки пузырьком, данная сортировка отлично справляется с "черепахами" - элементами которые нужно протащить через весь массив, поэтому имеет трудоемкость: $O(n \cdot \log n)$.
   - **Худший случай** - обратно отсортированный массив, как и для сортировки пузырьком. Трудоемкость: $O(n^2)$, однако является самой быстрой из квадратичных сортировок.

3. **Быстрая сортировка** [2].

   - **Лучший случай** - в наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит $\log_2 n$. В результате количество сравнений, совершаемых быстрой сортировкой, было бы равно значению рекурсивного выражения $C_n = 2 \cdot C_{n/2} + n$, что даёт общую сложность алгоритма $O(n \cdot \log n)$.

- **Худший случай** - в самом несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и $n-1$, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. При простейшем выборе опорного элемента — первого или последнего в массиве, — такой эффект даст уже отсортированный (в прямом или обратном порядке) массив, для среднего или любого другого фиксированного элемента «массив худшего случая» также может быть специально подобран. В этом случае потребуется $n-1$ операций разделения, а общее время работы составит $\sum_{i=0}^{n}(n-i) = O(n^2)$ операций.

Таким образом, в лучшем случае сортировка вставками имеет наименьшую трудоемкость, равную $O(n)$. Однако в худшем случае она сильно проигрывает быстрой сортировке и сортировке расческой из-за коэффициента при n^2, при этом быстрая сортировка и сортировка расческой имеют приблизительно одинаковую трудоемкость во всех случаях.

3 | Технологическая часть

В данном разделе приведены требования к программнму обеспечению, средства реализации и листинги кода

3.1 Требования к ПО

На вход поступает массив чисел, на выходе должен возвращаться результат его сортировки.

3.2 Средства реализации

Для реализации представленных алгоритмов был выбран язык Python. Время работы алгоритмов было замерено с помощью функции perf_counter() из библиотеки time.

3.3 Листинги кода

В Листинге 3.1 показана реализация алгоритма сортировки вставками.

Листинг 3.1: Функция сортировки вставками

```
def insertion(arr):
    result = list(arr)
    for i in range(1, len(result)):
        j = i - 1
        key = result[i]
        while result[j] > key and j >= 0:
            result[j + 1] = result[j]
            j -= 1
        result[j + 1] = key
    return result
```

В Листинге 3.2 показана реализация алгоритма сортировки расческой.

Листинг 3.2: Функция сортировки расческой

```
COMB_COEF = 1.247330950103979

def comb(arr):
    result = list(arr)
    gap = len(result)
    while gap > 1:
        gap = max(1, int(gap / COMB_COEF))
        for i in range(len(result) - gap):
            j = i + gap
            if result[i] > result[j]:
                result[i], result[j] = result[j], result[i]
    return result
```

В Листинге 3.3 показана реализация алгоритма быстрой сортировки.

Листинг 3.3: Функция быстрой сортировки

```
def quick(arr):
    if len(arr) <= 1:
        return arr
    else:
        q = random.choice(arr)
        left = []
        middle = []
        right = []
        for elem in arr:
            if elem < q:
                left.append(elem)
            elif elem > q:
                right.append(elem)
            else:
                middle.append(elem)
        return quick(left) + middle + quick(right)
```

4 | Экспериментальная часть

В данном разделе приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1 Функциональное тестирование

Тест 1: Пустой массив
Массив: [ ]
Ожидаемый результат: [ ]
Сортировка вставками: [ ]
Сортировка расческой: [ ]
Быстрая сортировка: [ ]

Тест 2: Массив из 1 элемента
Массив: [2]
Ожидаемый результат: [2]
Сортировка вставками: [2]
Сортировка расческой: [2]
Быстрая сортировка: [2]

Тест 3: Отсортированный массив
Массив: [1, 2, 3, 4, 5, 6]
Ожидаемый результат: [1, 2, 3, 4, 5, 6]
Сортировка вставками: [1, 2, 3, 4, 5, 6]
Сортировка расческой: [1, 2, 3, 4, 5, 6]
Быстрая сортировка: [1, 2, 3, 4, 5, 6]

Тест 4: Случайный массив
Массив: [6, 2, 4, -4, 1]
Ожидаемый результат: [-4, 1, 2, 4, 6]
Сортировка вставками: [-4, 1, 2, 4, 6]
Сортировка расческой: [-4, 1, 2, 4, 6]
Быстрая сортировка: [-4, 1, 2, 4, 6]

Тест 5: Обратно отсортированный массив
Массив: [6, 5, 4, 3, 2, 1]
Ожидаемый результат: [1, 2, 3, 4, 5, 6]
Сортировка вставками: [1, 2, 3, 4, 5, 6]
Сортировка расческой: [1, 2, 3, 4, 5, 6]
Быстрая сортировка: [1, 2, 3, 4, 5, 6]

Программа успешно прошла все тестовые случаи, все полученные результаты совпали с ожидаемыми.

4.2 Сравнение алгоритмов по времени

Для экспериментов использовались массивы, размер которых варьируется от 100 до 1000 с шагом 100, при этом рассматривались 3 случая:

- Массив заполнен случайными числами
- Массив отсортирован
- Массив отсортирован в обратном порядке

Количество повторов каждого эксперимента = 100. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

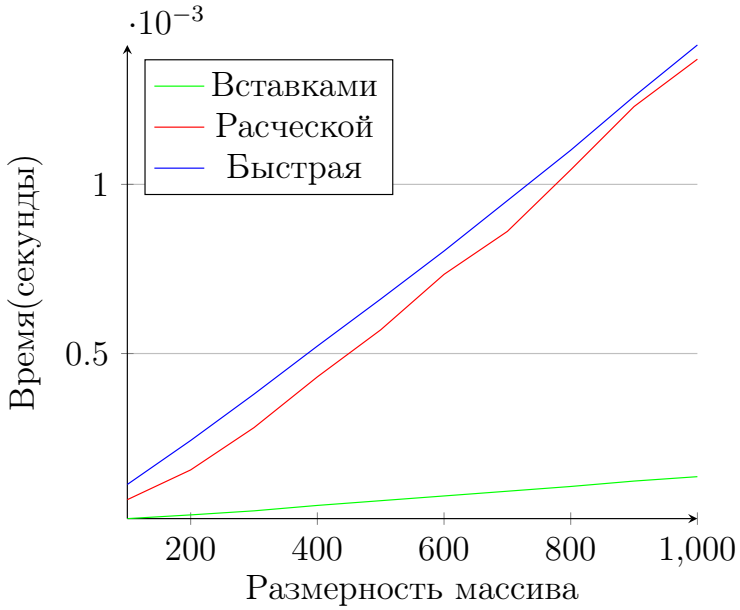

Рис. 4.1: Алгоритмы сортировки массивов(отсортированных)

На Рис. 4.1 видно, что алгоритм сортировки вставками значительно превосходит другие алгоритмы по времени(в ≈ 10 раз). При этом алгоритм сортировки расческой превосходит алгоритм быстрой сортировки на $\approx 3\%$.

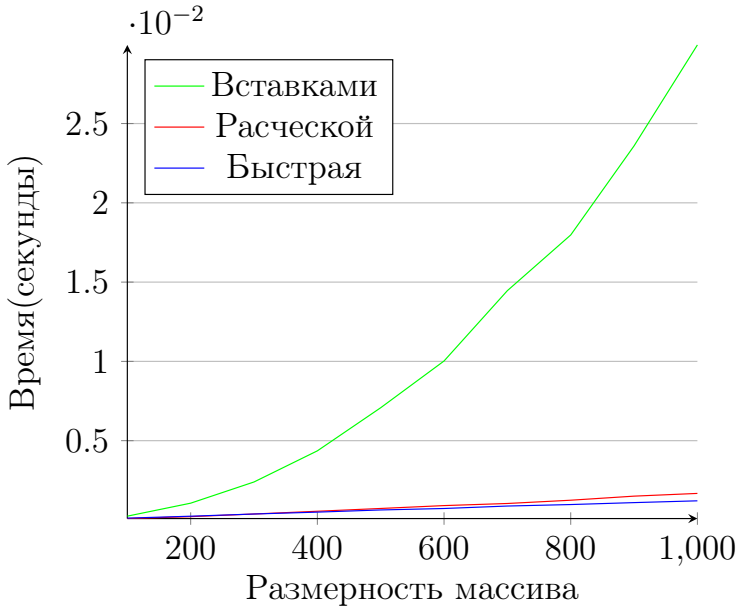

Рис. 4.2: Алгоритмы сортировки массивов(случайных)

На Рис. 4.2 видно, что алгоритм сортировки вставками в случае сортировки случайного массива потерял свое превосходство и отставание от других алгоритмов растет практически в геометрической прогрессии от размера массива. Отставание алгоритма сортировки расческой от алгоритма быстрой сортировки составляет $\approx 30\%$, однако оно несущественно по сравнению с временем сортировки вставками.

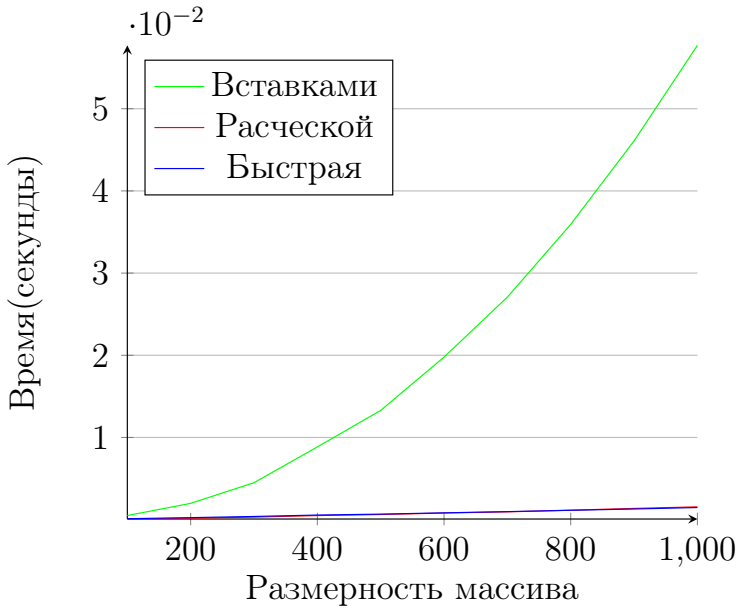

Рис. 4.3: Алгоритмы сортировки массивов(обратно отсортированных)

На Рис. 4.3 видно, что алгоритм сортировки вставками также проигрывает другим алгоритмам и в худшем случае, причем отставание также растет в геометрической прогрессии от размера массива. Алгоритм сортировки расческой отстает от алгоритма быстрой сортировки на $\approx 5\%$, что по сравнению с сортировкой вставками делает их результаты практически идентичными.

Заключение

В ходе работы были изучены и реализованы такие алгоритмы сортировки массивов чисел,как алгоритм сортировки вставками, алгоритм сортировки расческой и алгоритм быстрой сортировки. Был проведен сравнительный анализ перечисленных алгоритмов по трудоемкости и экспериментально подтверждено различие во временной эффективности. Было показано, что алгоритм сортировки вставками является лучшим выбором, если массив уже отсортирован или практически отсортирован, однако он значительно проигрывает другим алгоритмам в случаях случайного массива и обратно отсортированного массива. В этих случаях алгоритм быстрой сортировки показывает лучшие результаты, но алгоритм сортировки расческой не сильно ему уступает.

Список литературы

[1] TutorialsPoint Comb Sort [Электронный ресурс]. – Режим доступа: URL: https://www.tutorialspoint.com/Comb-Sort. – (Дата обращения: 25.10.2019)

[2] Wikipedia Быстрая сортировка [Электронный ресурс]. – Режим доступа: URL: https://ru.wikipedia.org/wiki/Быстрая сортировка. – (Дата обращения: 25.10.2019)