МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Физтех-школа аэрокосмических технологий

Лабораторная работа
Сортировка данных. C++

Работу выполнил
Лохматов Арсений Игоревич
Б03-303

Долгопрудный 2023

Задача 0: Пузырёк и его товарищи

Цель: Написать три разных сортировки с временной сложностью O(N^2) (пузырек, вставка, шейкер, простой выбор, непростой выбор или что угодно, что можно найти). Построть график зависимости времени работы от размера массива с тремя кривыми – по одной для каждой сортировки. Доказать, что сложность этих алгоритмов действительно O(N^2).

Ход работы: Напишем функции для сортировки: метод пузырька, метод выбора, метод вставки, метод шейкера. Напишем код, в котором будет передавать данным функциям массивы разных размеров, заполненные случайными числами (с помощью библиотеки *random*), для каждого масива будем определять время сортировки.. Поскольку время работы зависит не только от сложности программы, но и от технических требований ПК и проч., будем искать среднее время работы программы для 3 независимых экспериментов. По данным измерениям построим график зависимости длины обрабатываемого массива от времени обработки. Строить будем в логарифмических координатных осях, чтобы наглядно увидеть, что зависимость составляет O(N^2). Как мы это поймём? Заметим, что

$$O = N^2 \to log(O) = 2 \cdot log(N),$$

то есть на графике мы должны получить прямую с коэффициентом наклона равным 2, начиная с некоторой достаточно большой длины массива.

→Сортировка методом пузырька:

```
int r = 1;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n-r; j++) {
        if (array[j] > array[j+1]) {
            int elem = array[j];
            array[j] = array[j+1];
            array[j+1] = elem;
        }
    }
    r += 1;
}
```

Рис. 1: Функция сортировки методом пузырька.

Рис. 2: Зависимость времени работы алгоритма от размера массива.

Видим, что начиная с определённой длины массива ($\sim 10^3$) график представляет собой прямую, значит сложность этого алгоритма $O(N^2)$.

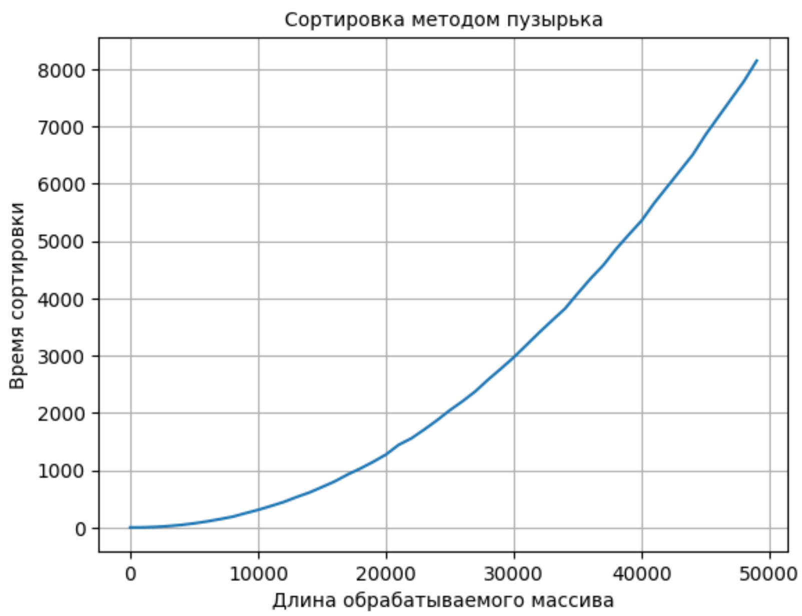


Рис. 3: Зависимость времени работы алгоритма от размера массива.

Если строить график не в логарифмических осях, то получим параболу - так же доказательство того, что сложность данного алгоритма составляет $O(N^2)$. Аналогично проделаем работу для других видов сортировки.

→Сортировка методом вставки:

```
for (int i = 1; i < n; i++) {
    int j = i;
    while (array[j] < array[j-1] && j > 0)
    if (array[j] < array[j-1]) {
        int elem = array[j];
        array[j] = array[j-1];
        array[j-1] = elem;
        j -= 1;
    }
}
```

Рис. 4: Функция сортировки методом вставки.

Рис. 5: Зависимость времени работы алгоритма от размера массива.

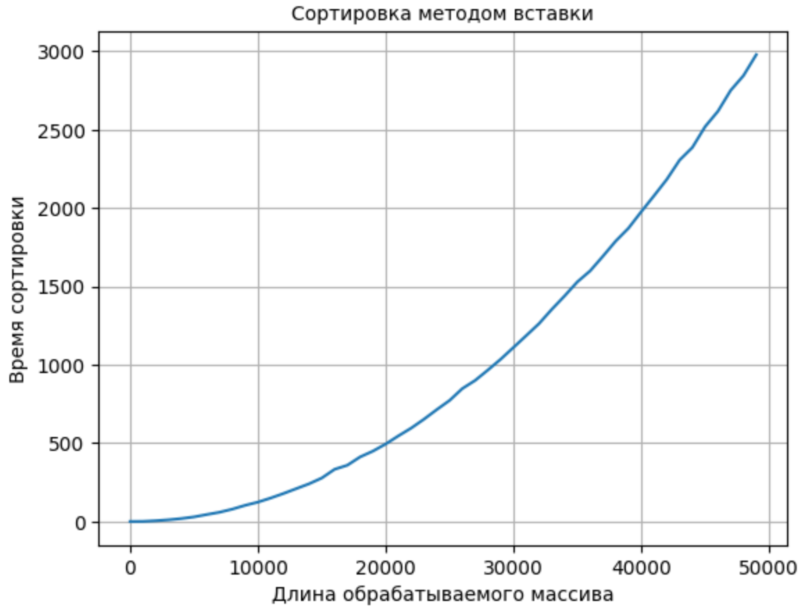


Рис. 6: Зависимость времени работы алгоритма от размера массива.

→Сортировка методом шейкера:

```
int l = 0, r = 1;
for (int i = 0; i < n; i++) {
    for (int j = l; j < n-r; j++) {
        if (array[j] > array[j+1]) {
            int elem = array[j];
            array[j] = array[j+1];
            array[j+1] = elem;
        }
    }
    for (int j = n-r; j > l; j--) {
        if (array[j] < array[j-1]) {
            int elem = array[j];
            array[j] = array[j-1];
            array[j-1] = elem;
        }
    }
    l += 1;
    r += 1;
}
```

Рис. 7: Функция сортировки методом шейкера.

Рис. 8: Зависимость времени работы алгоритма от размера массива.

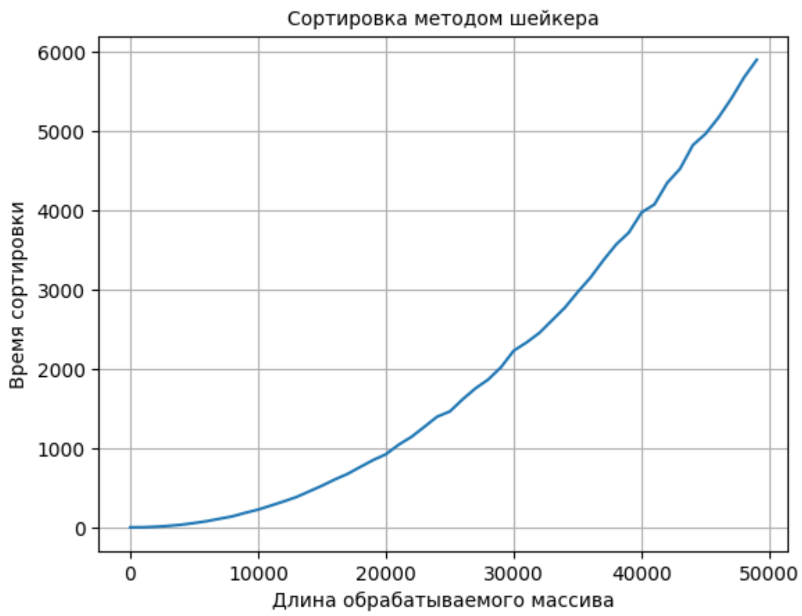


Рис. 9: Зависимость времени работы алгоритма от размера массива.

Построим графики в одной системе координат (обычных и логарифмических) для наглядного сравнения:

Рис. 10: Сравнение алгоритмов.

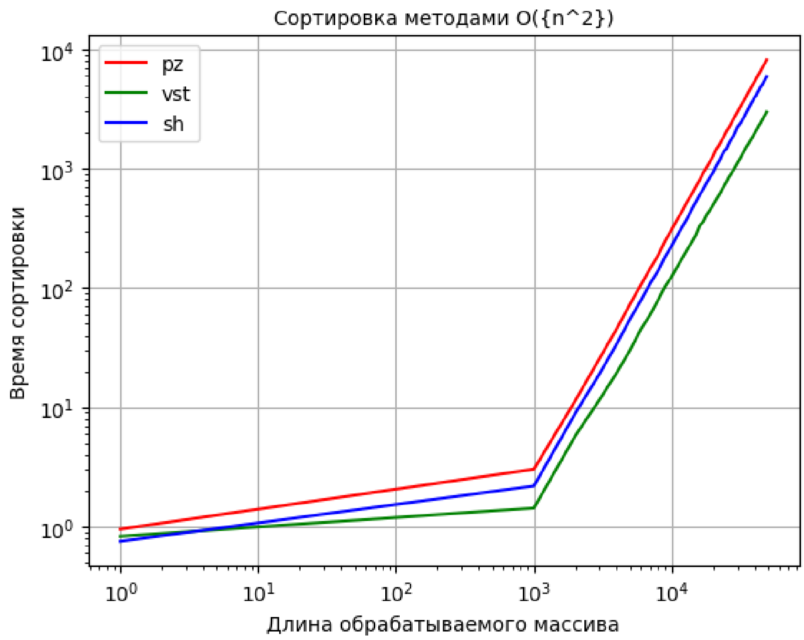


Рис. 11: Сравнение алгоритмов.

Задача 1: Пузырек, но быстрее

Цель: А теперь лезем в настройки компилятора: добавляем при компиляции флаги -O0, -O1, -O2 и -O3 соответственно.

Ход работы: Построим график с четырьмя кривыми для одной из сортировок за O(N2) – по одной кривой на каждую степень оптимизации.

→Сортировка методом пузырька:

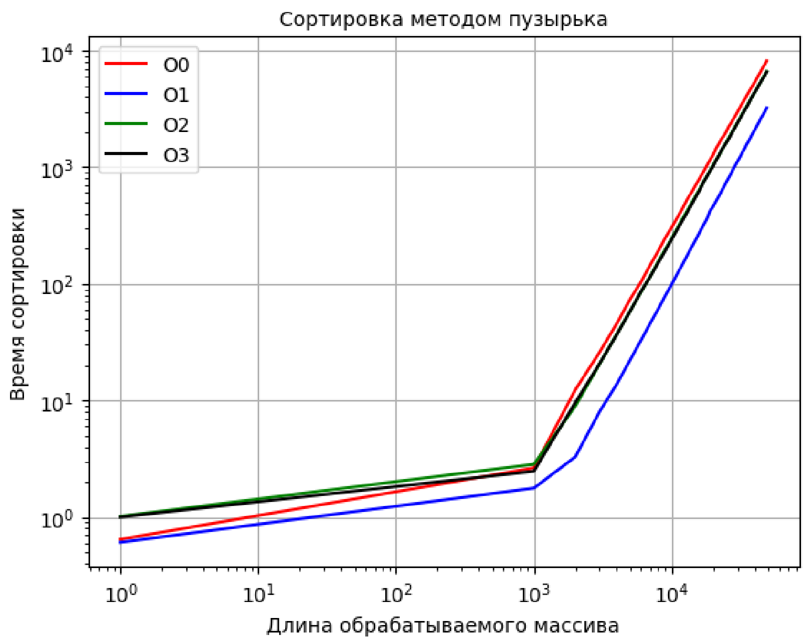


Рис. 12: Метод пузырька на O0, O1, O2, O3.

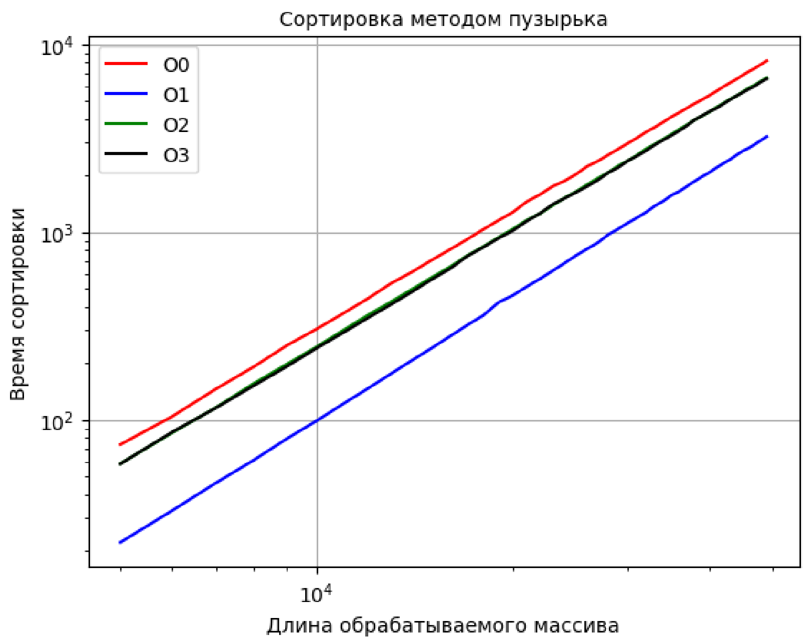


Рис. 13: Метод пузырька на O0, O1, O2, O3 (участок графиков).

Можно заметить, что при сортировке методом пузырька оптимизация O2 и O3 не сильно ускоряют работу программы, нежели O1.

→Сортировка методом вставки:

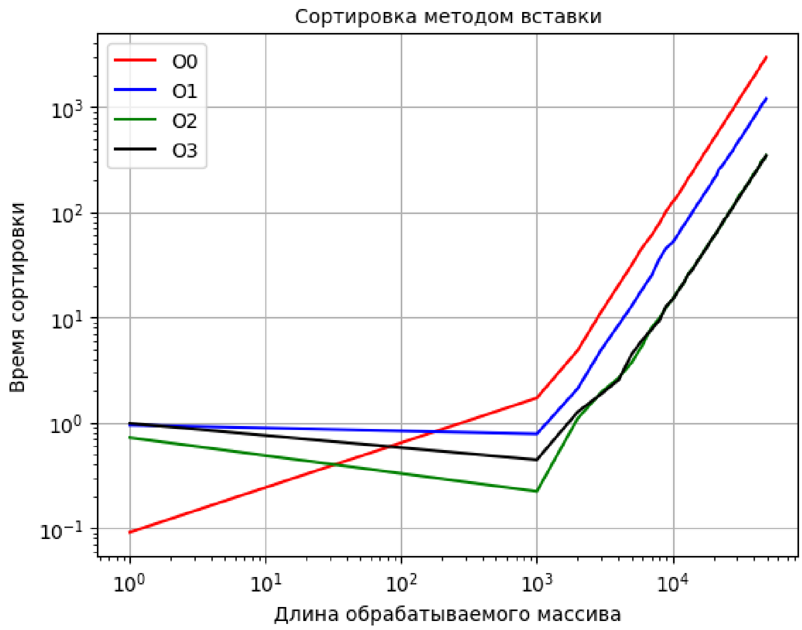


Рис. 14: Метод вставки на O0, O1, O2, O3.

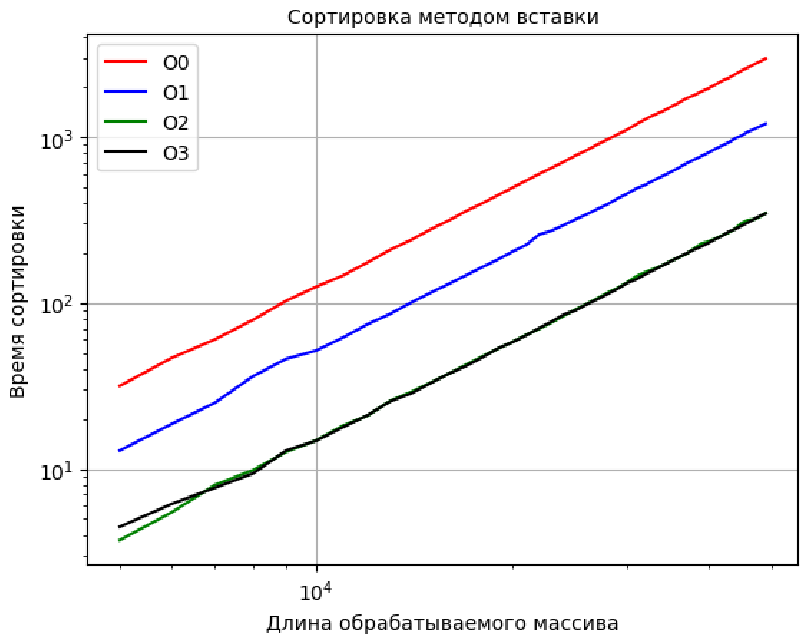


Рис. 15: Метод вставки на O0, O1, O2, O3 (участок графиков).

А у сортировки методов вставки, напротив, оптимизация O2 и O3 ускоряют программу лучше, чем O1.

→Сортировка методом шейкера:

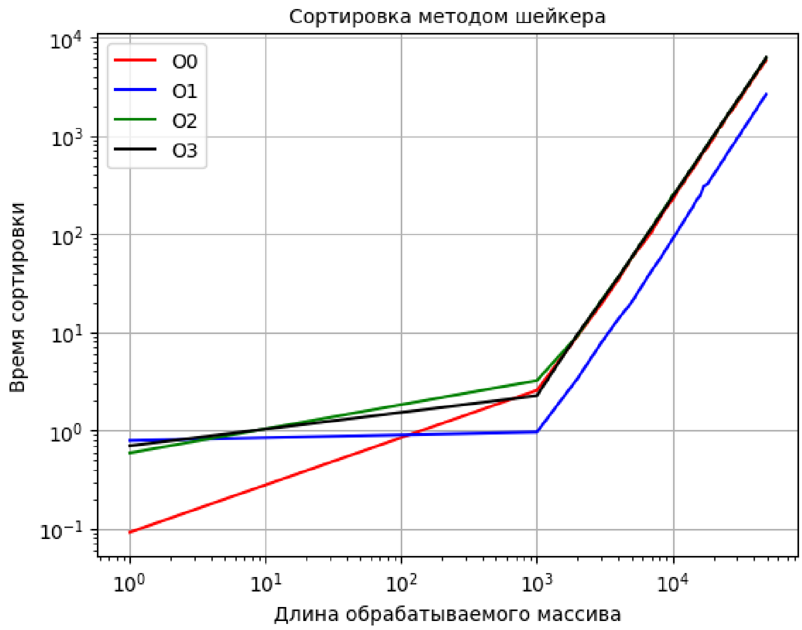


Рис. 16: Метод шейкера на O0, O1, O2, O3.

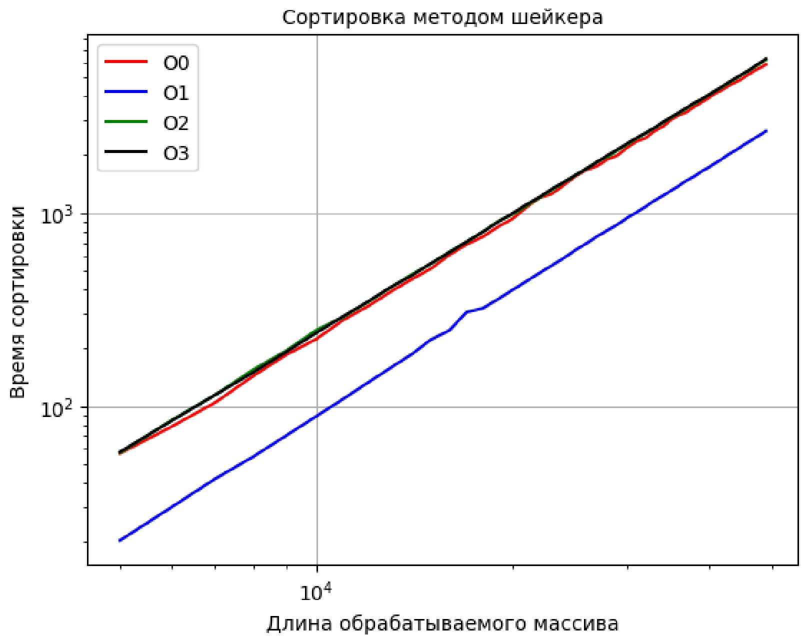


Рис. 17: Метод шейкера на O0, O1, O2, O3 (участок графиков).

А при сортировке методом шейкера работу программы ускоряет оптимизация O1, а при O2 и O3 время работы практически не отличается от времени работы программы без оптимизации.

Задача 2: Настоящие быстрые сортировки

Цель: Написать три разных сортировки с временной сложностью O(N log N). Это может быть куча, Хоар, расческа, слияние или что угодно, что можно нагуглите/найдете в лекциях.

Ход работы: Аналогично задаче 0 мы сначала напишем функции для сортировок, а затем построим графики, чтобы наглядно показать, что их сложность составляет $O(Nlog(N))$. Построим графики в осях $\frac{t}{Nlog(N)}$. Должна получиться константа, но поскольку условия эксперимента не идеальные, то получается не совсем прямая (много маленьких колебаний (гармошка) около одного числа). Поэтому построим в этих же координатах функцию $y = N \cdot log(N)$, отнгосительно этого графика наша зависимость представляет собой константу, близкую к нулю - доказательство того, что зависимоть $O(Nlog(N))$.

→Сортировка методом Хоара:

```
void hoarasort(int* array, int first, int last) {
    int i = first, j = last;
    int p, opora = array[(first + last) / 2];
    do {
        while (array[i] < opora) {
            i++;
        }
        while (array[j] > opora) {
            j--;
        }
        if (i <= j) {
            if (i < j) {
                p = array[i];
                array[i] = array[j];
                array[j] = p;
            }
            i++;
            j--;
        }
    } while (i <= j);

    if (i < last)
    hoarasort(array, i, last);
    if (first < j)
    hoarasort(array, first,j);
}
```

Рис. 18: Функция сортировки методом Хоара.

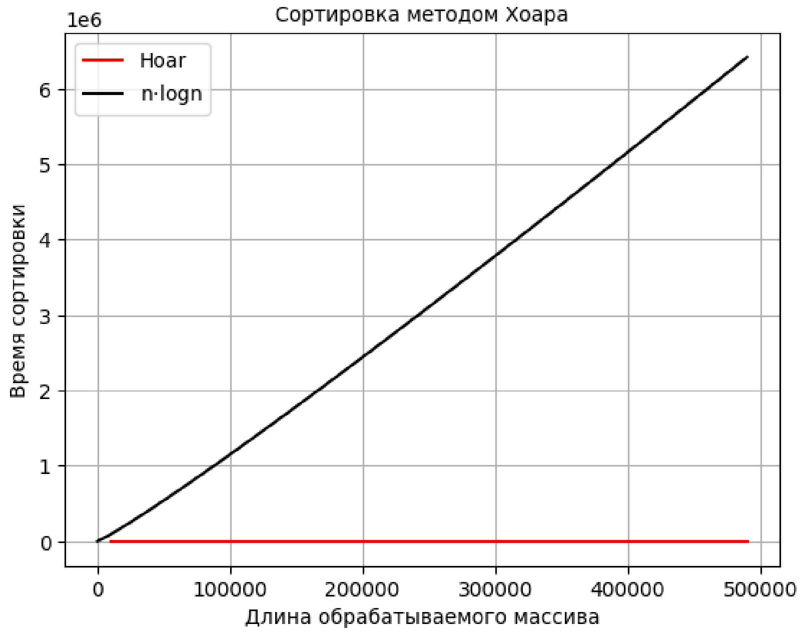


Рис. 19: Зависимость времени работы алгоритма от размера массива.

→Сортировка методом Кучи:

С реализацией данного алгоритма у меня возникли трудности, поэтому я прибегнул к гуглу, за что каюсь (но я разобрался и это круто!). Поэтому данный алгоритм, я считаю, показывать здесь не имеет смысла.

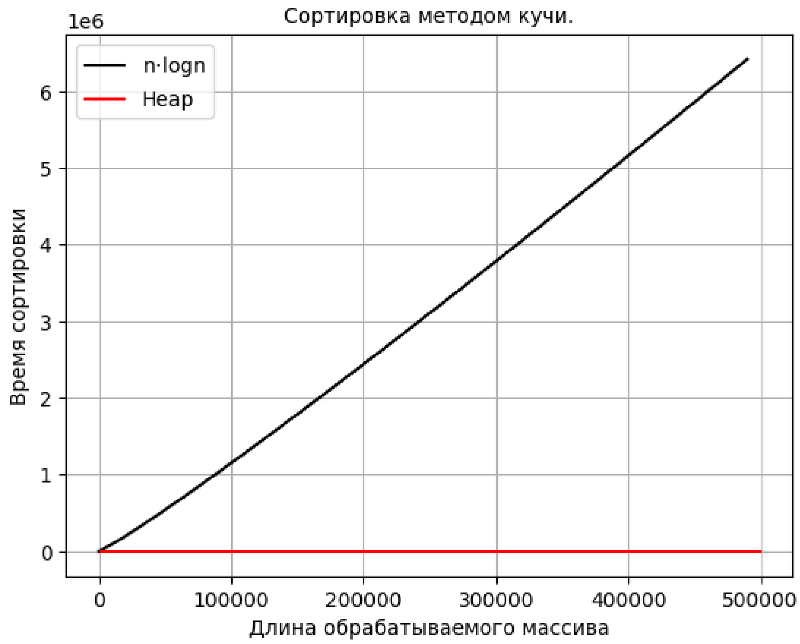


Рис. 20: Зависимость времени работы алгоритма от размера массива.

→Сортировка методом расчёстки:
Так же возникли трудности, из-за чего я обратился к гуглу.

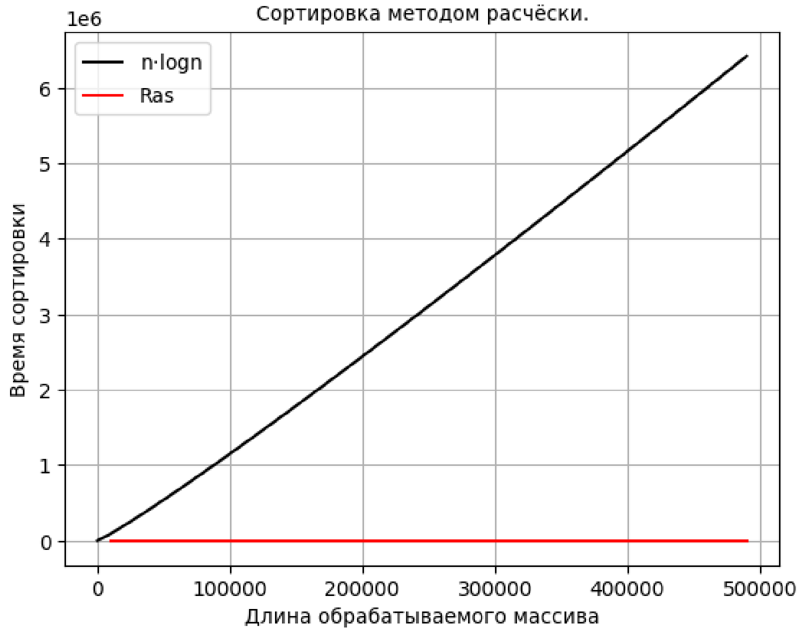


Рис. 21: Зависимость времени работы алгоритма от размера массива.

Построим все графики в одной системе координат без функции $y = N \cdot log(N)$. Видим, что начиная с некоторой длины массива график представляет собой константу (ну или почти).

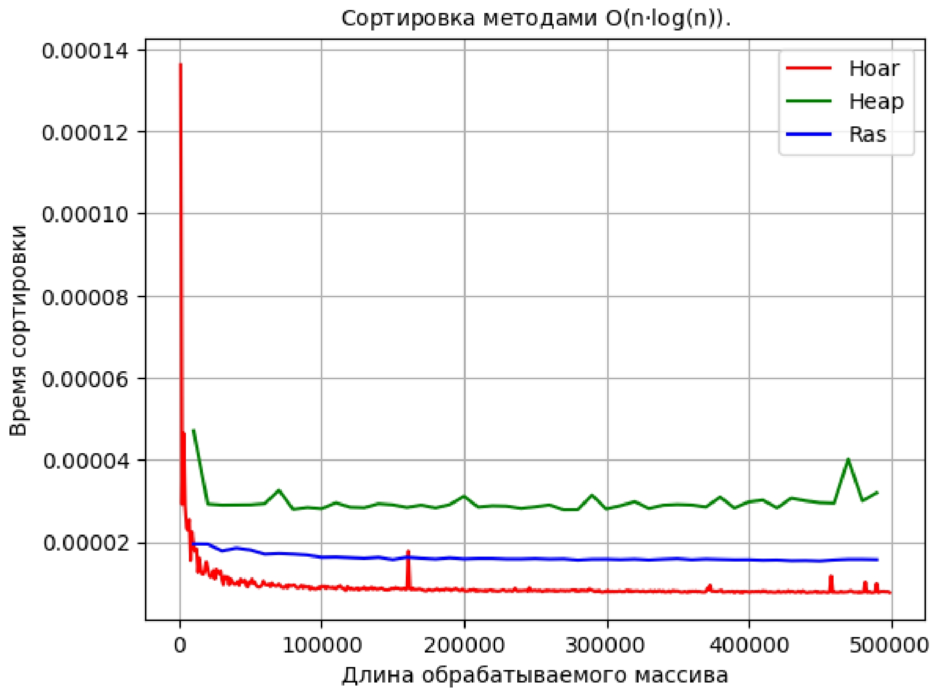
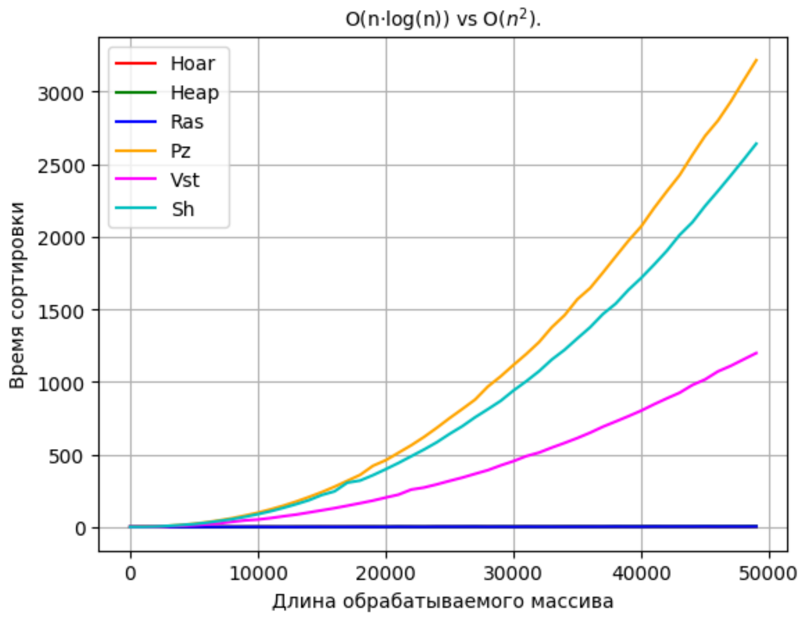


Рис. 22: Сравнение алгоритмов.

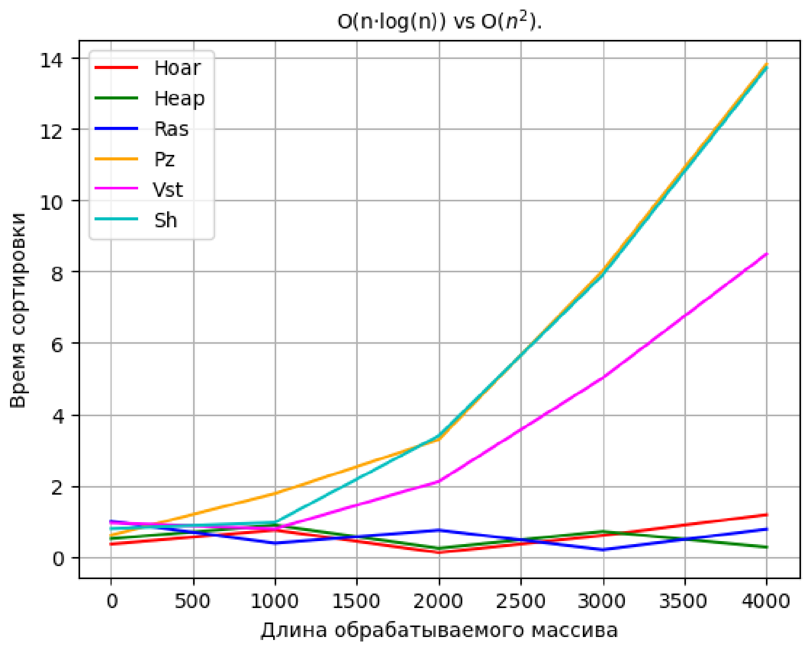
Задача 3: $O(n^2)$ vs $O(nlog(n))$

Цель: Нарисовать кривые из пунктов 0 и 2 на одном графике. Графики строим по данным, которые получили с помощью одной оптимизации.

Ход работы: Построим график из предыдущих пунктов в одной системе координат для одинаковой оптимизации Пользуемся оптимизацией O1 - она, как мне кажется, наиболее эффективна в общем случае. Поскольку мы уже доказали сложность данных функций, то сейчас построим графики в обычных координатных осях.



Рассмотрим ближе графики при малых массивах:



Видим, что при малых длинах обрабатываемых массивов особо не играет роль, какой алгоритм совершает сортировку. Но при больших длинах обрабатываемого массива время работы быстрых алгоритмов сильно меньше, нежели у пузырька, шейкера и вставки.

Задача 4: Зависимость от начальных данных

Цель: Сравнить время работы алгоритмов на отсортированном массиве, массиве произвольных чисел и массиве, отсортированном в обратную сторону. То есть лучший, средний и худший случай. Для каждого из шести реализованных алгоритмов построить по три кривых – лучший, средний и худший случай. Подумайть, как их сгруппировать, чтобы наиболее наглядно продемонстрировать разницу – в этом пункте может получиться несколько картинок.

Ход работы: Для начала построим графики для каждого алгоритма - при отсортированном массиве, массиве из произвольных чисел и отсортированном массиве в обратном порядке. Для создания отсортированных массивов будем создавтаь массивы из рандомных чисел и сортировать одним из методов - их у нас предостаточно. Так же пользуемся оптимизацией O1.

→Сортировка методом пузырька (в обычных координатных осях):

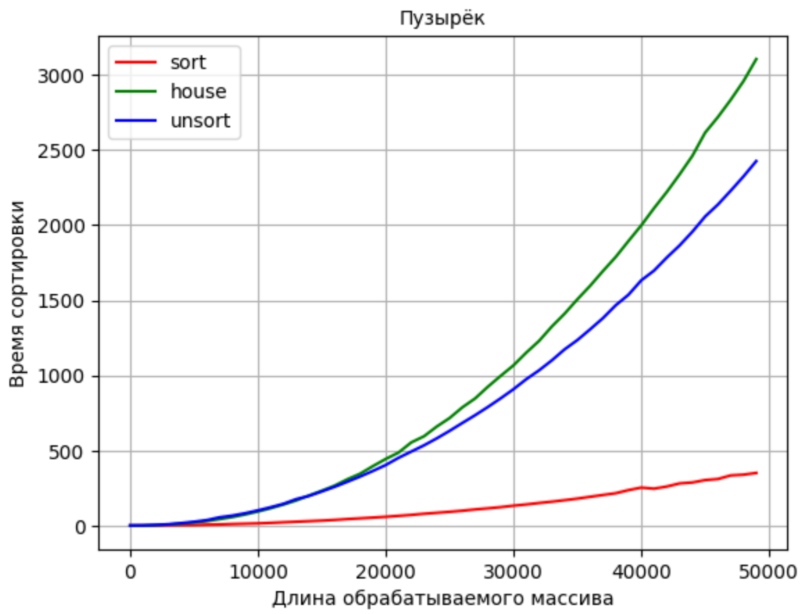


Рис. 23: Сравнение времени работы алгоритма в лучшем, среднем и худшем случаях.

При отсортированном массиве (*sort*) время сортировки сильно меньше, чем при отсортированном массиве в обратном порядке (*unsort*), для обработки массива из случайных чисел (*house*) время работы программы наибольшее.

→Сортировка методом вставки (в обычных координатных осях):

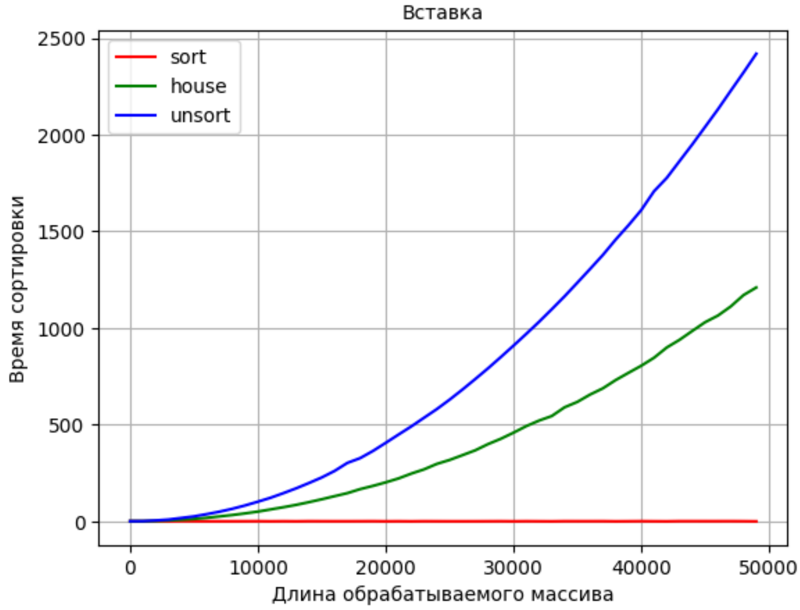


Рис. 24: Сравнение времени работы алгоритма в лучшем, среднем и худшем случаях.

При данной сортировке время работы для массива из случайных чисел (*house*) меньше, чем для обработки массива, отсортированном в обратном порядке (*unsort*). При отсортированном массиве (*sort*) время сортировки сильно меньше, чем для остальных.

→Сортировка методом шейкера (в обычных координатных осях):

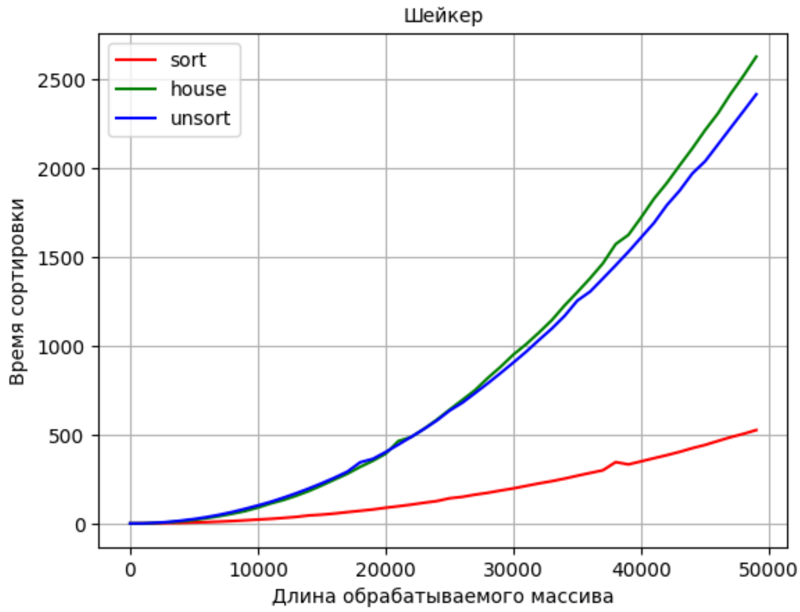


Рис. 25: Сравнение времени работы алгоритма в лучшем, среднем и худшем случаях.

При данной сортировке время работы для массива из случайных чисел (*house*) и для массива, отсортированном в обратном порядке (*unsort*), практически одинаковое. При отсортированном массиве (*sort*) время сортировки меньше, чем для остальных.

→Сортировка методом Хоара (в осях $\frac{t}{Nlog(N)}$):

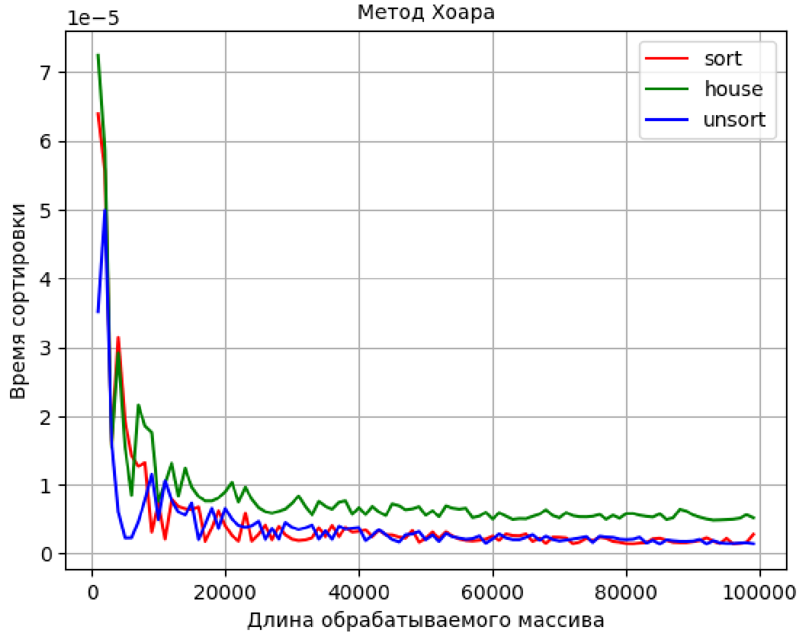


Рис. 26: Сравнение времени работы алгоритма в лучшем, среднем и худшем случаях.

При данной сортировке время работы для массива из случайных чисел (*house*) больше, чем для обработки отсортированного массива (*sort*) и массива, отсортированного в обратном порядке (*unsort*) (анализ при больших N).

→Сортировка методом Кучи (в осях $\frac{t}{Nlog(N)}$):

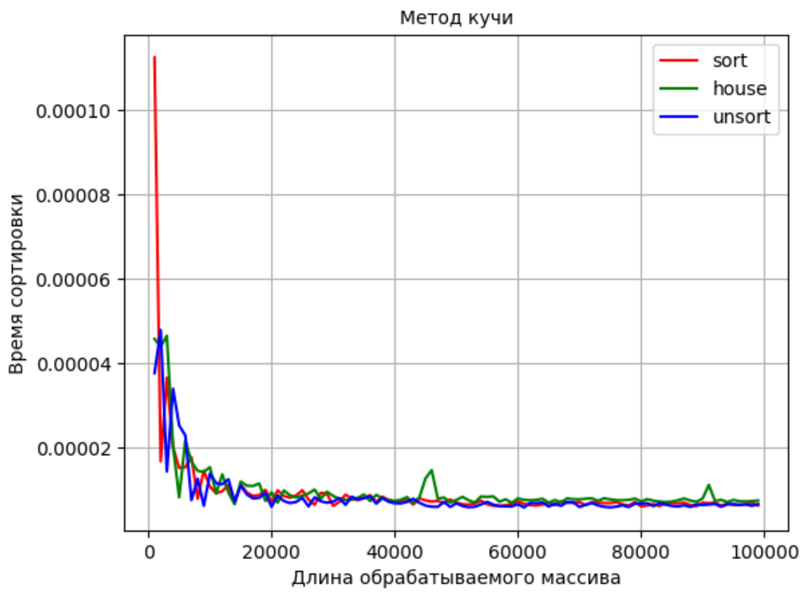


Рис. 27: Сравнение времени работы алгоритма в лучшем, среднем и худшем случаях.

При данной сортировке время работы для все случаем отличается не сильно, однако есть варианты, когда время обработки массива из случайных чисел (*house*) немного больше (мгновенное возрастание графика) (анализ при больших N).

→Сортировка методом расчёски (в осях $\frac{t}{Nlog(N)}$):

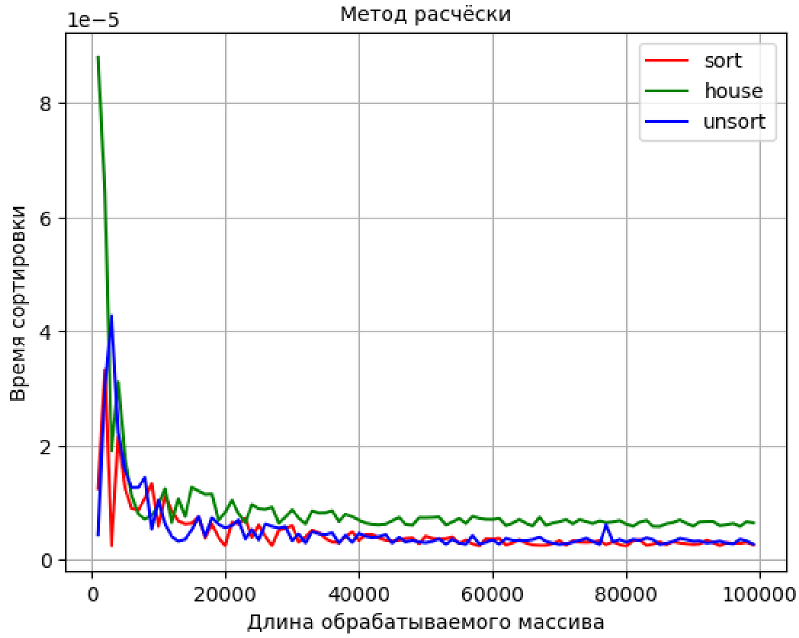


Рис. 28: Сравнение времени работы алгоритма в лучшем, среднем и худшем случаях.

Случай, аналогичный сортировке методом Хоара: время работы для массива из случайных чисел (*house*) больше, чем для обработки отсортированного массива (*sort*) и массива, отсортированного в обратном порядке (*unsort*) (анализ при больших N).

Теперь построим графики на одной координатной плоскости для наглядного сравнения разных методов. Чтобы не городить всё на одном графике, построим три для разных массивов.

→Сортировка отсортированного массива (в обычных координатных осях):

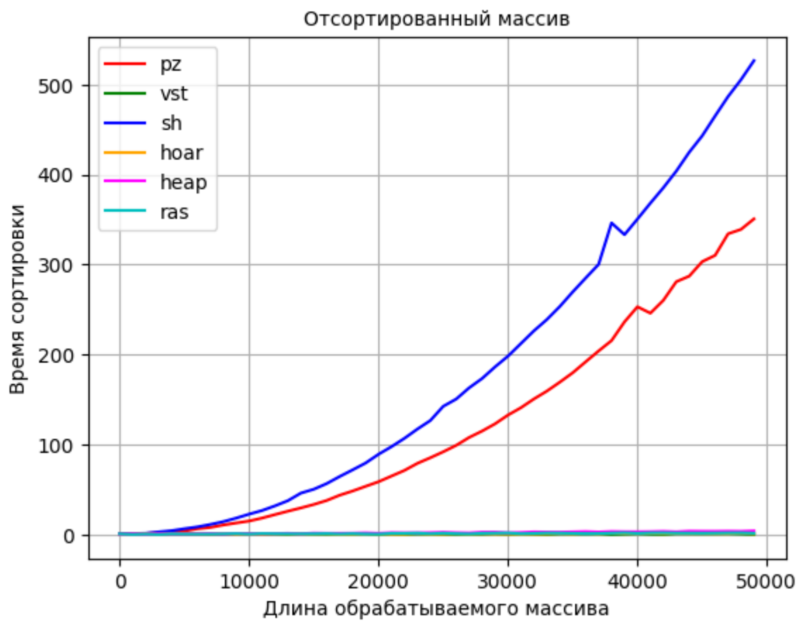


Рис. 29: Сравнение алгоритмов для обработки отсортированного массива.

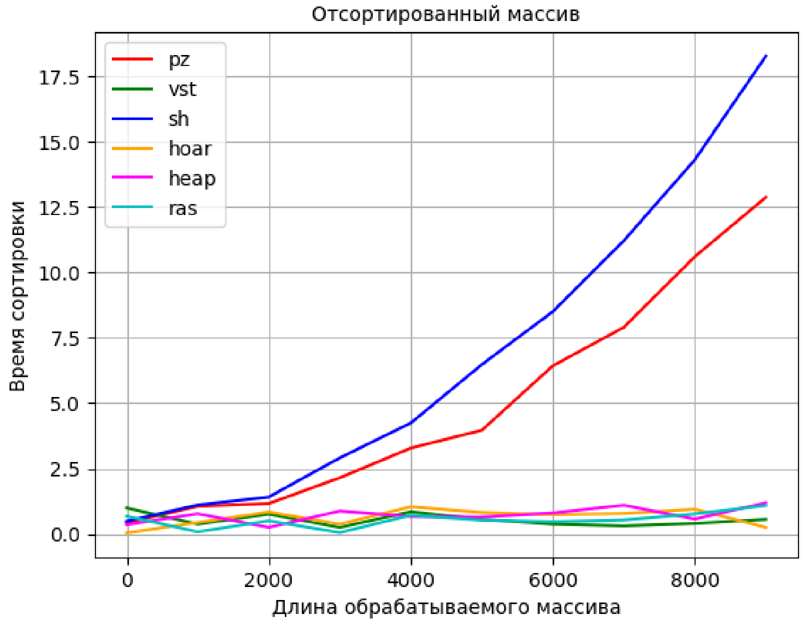


Рис. 30: Сравнение алгоритмов для обработки отсортированного массива (другой масштаб).

→Сортировка массива случайных данных (в обычных координатных осях):

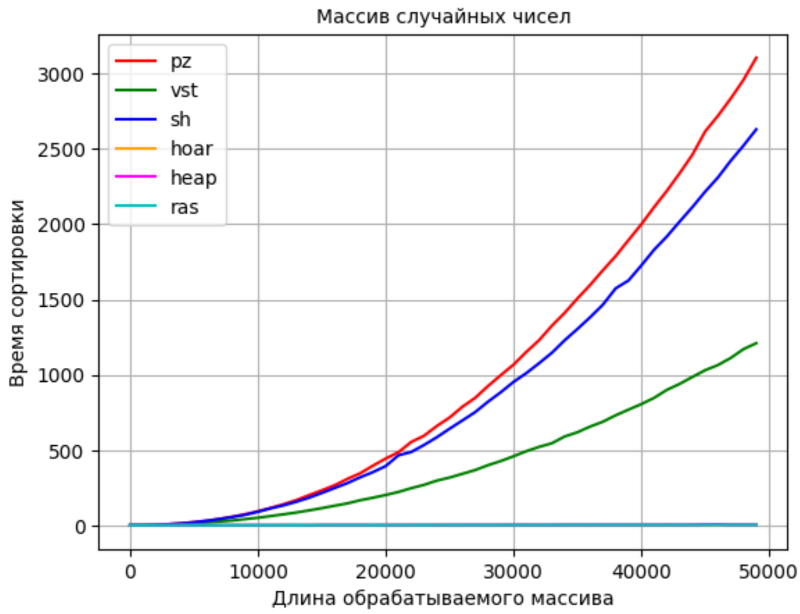


Рис. 31: Сравнение алгоритмов для обработки массива случайных данных.

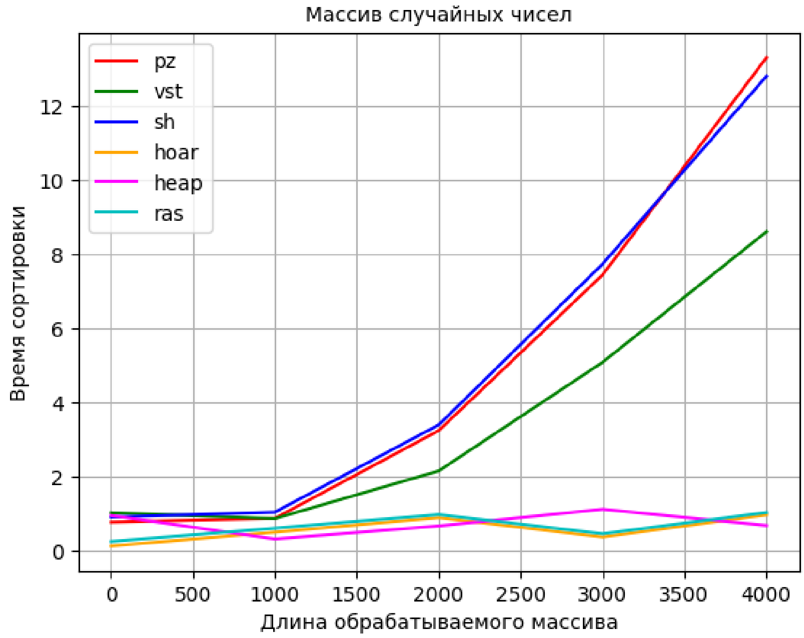


Рис. 32: Сравнение алгоритмов для обработки массива случайных данных (другой масштаб).

→Сортировка отсортированного массива в обратном порядке (в обычных координатных осях):

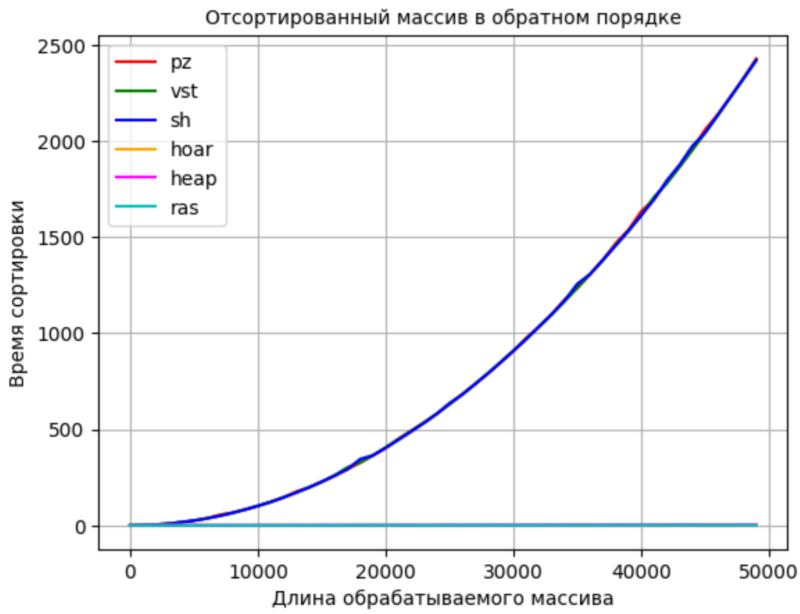


Рис. 33: Сравнение алгоритмов для обработки массива случайных данных.

Рис. 34: Сравнение алгоритмов для обработки массива случайных данных (другой масштаб).