Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

В лабораторной работе предлагается изучить альтернативные первой лабораторной сортировки, которые обладают меньшей асимтотической сложносью и сравнить их с результатами предыдущей лабораторной работы. Используя предыдущий код посерийного выполнения алгоритма сортировки и измерения времени требуется реализовать метод быстрой сортировки.

Описание метода/модели.

Быстрая сортировка

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

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.

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.

Схема Хаора:

Данная схема использует два индекса (один в начале массива, другой — в конце), которые приближаются друг к другу, пока не найдётся пара элементов, где один больше опорного и расположен перед ним, а второй — меньше и расположен после. Эти элементы меняются местами. Обмен происходит до тех пор, пока индексы не пересекутся. Алгоритм возвращает последний индекс. Схема Хоара эффективнее схемы Ломуто, так как происходит в среднем в три раза меньше обменов (swap) элементов и разбиение эффективнее, даже когда все элементы равны. Подобно разбиению Ломуто,

данная схема также показывает эффективность в O(n2), когда входной массив уже отсортирован. Сортировка с использованием данной схемы нестабильна. Следует заметить, что конечная позиция опорного элемента необязательно совпадает с возвращённым индексом.

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

Блок кода отвечающий за реализацию сортировки:

```
def quicksort(nums, depth=0):
    global recursion_calls, max_recursion_depth, swap_operations
    recursion_calls += 1
    max_recursion_depth = max(max_recursion_depth, depth)
    if len(nums) <= 1:
        return nums
    else:
        q = choice(nums)
        l_nums = [n for n in nums if n < q]
        e_nums = [q] * nums.count(q)
        b_nums = [n for n in nums if n > q]
        swap_operations += len(l_nums) + len(b_nums)
        return quicksort(l_nums, depth + 1) + e_nums + quicksort(b_nums, depth + 1)
```

Результаты работы программы выводится в results.txt файл в формате {размерность массива: время затраченное на сортировку: количество вызовов рекурсий: глубина рекурсии: количество перестановок}. Пример приведён ниже для сортировки массива размерностью 1000 элементов.

```
  Размерность: 1000 Время: 0.001377 Количество вызовов рекурсии: 1319 Глубина рекурсии: 20 Количество обменов: 11348
```

Полный файл с данными:

Размерность: 1000 Время: 0.001377 Количество вызовов рекурсии: 1319 Глубина рекурсии: 20 Колич
11348
Размерность: 1000 Время: 0.001385 Количество вызовов рекурсии: 1357 Глубина рекурсии: 20 Колич
11067
Размерность: 1000 Время: 0.001609 Количество вызовов рекурсии: 1333 Глубина рекурсии: 20 Колич
11472
Размерность: 1000 Время: 0.002016 Количество вызовов рекурсии: 1349 Глубина рекурсии: 18 Колич
10341
Размерность: 1000 Время: 0.001824 Количество вызовов рекурсии: 1331 Глубина рекурсии: 17 Колич
9900
Размерность: 1000 Время: 0.001479 Количество вызовов рекурсии: 1319 Глубина рекурсии: 20 Колич
10355
Размерность: 1000 Время: 0.001991 Количество вызовов рекурсии: 1331 Глубина рекурсии: 23 Колич
11861
Размерность: 1000 Время: 0.001990 Количество вызовов рекурсии: 1341 Глубина рекурсии: 21 Колич
10570
Размерность: 1000 Время: 0.001253 Количество вызовов рекурсии: 1309 Глубина рекурсии: 21 Колич
10933
Размерность: 1000 Время: 0.001535 Количество вызовов рекурсии: 1337 Глубина рекурсии: 20 Колич
11347
Размерность: 1000 Время: 0.001554 Количество вызовов рекурсии: 1327 Глубина рекурсии: 19 Колич
10382
Размерность: 1000 Время: 0.001350 Количество вызовов рекурсии: 1337 Глубина рекурсии: 19 Колич
10751
Размерность: 1000 Время: 0.001230 Количество вызовов рекурсии: 1317 Глубина рекурсии: 23 Колич
10382
Размерность: 1000 Время: 0.001258 Количество вызовов рекурсии: 1333 Глубина рекурсии: 24 Колич
10754
Размерность: 1000 Время: 0.001359 Количество вызовов рекурсии: 1317 Глубина рекурсии: 20 Колич
11589
Размерность: 1000 Время: 0.001286 Количество вызовов рекурсии: 1331 Глубина рекурсии: 19 Колич
10621
Размерность: 1000 Время: 0.001860 Количество вызовов рекурсии: 1333 Глубина рекурсии: 23 Колич
11433
Размерность: 1000 Время: 0.001787 Количество вызовов рекурсии: 1323 Глубина рекурсии: 23 Колич
10690
Размерность: 1000 Время: 0.001254 Количество вызовов рекурсии: 1347 Глубина рекурсии: 20 Колич
10387
Размерность: 1000 Время: 0.001747 Количество вызовов рекурсии: 1341 Глубина рекурсии: 22 Колич
12481

Тестируемый блок с неблагополучными вариантами:

```
    def quicksort(nums, depth=0):
        global recursion_calls, max_recursion_depth, swap_operations
        recursion_calls += 1
        max_recursion_depth = max(max_recursion_depth, depth)
        if len(nums) <= 1:
            return nums
        else:
            q = choice(nums)
            l_nums = [n for n in nums if n < q]
            e_nums = [q] * nums.count(q)
            b_nums = [n for n in nums if n > q]
            swap_operations += len(l_nums) + len(b_nums)
            return quicksort(l_nums, depth + 1) + e_nums + quicksort(b_nums, depth + 1)
    def quicksort_deterministic(nums, depth=0):
        global recursion_calls, max_recursion_depth, swap_operations
        recursion_calls += 1
        max_recursion_depth = max(max_recursion_depth, depth)
        if len(nums) <= 1:
            return nums
        else:
            q = nums[len(nums) // 2]
            l_nums = [n for n in nums if n < q]
            e_nums = [q] * nums.count(q)
            b_nums = [n for n in nums if n > q]
            swap_operations += len(l_nums) + len(b_nums)
            return quicksort_deterministic(l_nums, depth + 1) + e_nums +
quicksort_deterministic(b_nums, depth + 1)
    def test_negative_case(test_type, nums, sort_function, results_file):
        global recursion_calls, max_recursion_depth, swap_operations
        recursion_calls = 0
        max_recursion_depth = 0
        swap_operations = 0
        start_time = time.time()
        sorted_nums = sort_function(nums)
        end_time = time.time()
        with open(results_file, "a", encoding="UTF-8") as file:
            file.write(
                f"Тест: {test_type} "
                f"Размер массива: {len(nums)} "
                f"Время выполнения: {end_time - start_time:.6f} "
```

Результаты тестирования выводятся в testing.txt. Ниже приведены полученные данные.

Тест: Отсортированный массив Размер массива: 1000 Время выполнения: 0.001250 Рекурсии: 1323 Максимальная глубина: 18 Операции замены: 10561

Тест: Отсортированный массив Размер массива: 2000 Время выполнения: 0.002741 Рекурсии: 2673 Максимальная глубина: 23 Операции замены: 25621

Тест: Отсортированный массив Размер массива: 4000 Время выполнения: 0.007238 Рекурсии: 5347 Максимальная глубина: 26 Операции замены: 54858

Тест: Отсортированный массив Размер массива: 8000 Время выполнения: 0.012913 Рекурсии: 10679 Максимальная глубина: 30 Операции замены: 117153

Тест: Отсортированный массив Размер массива: 16000 Время выполнения: 0.029274 Рекурсии: 21341 Максимальная глубина: 33 Операции замены: 263364

Тест: Отсортированный массив Размер массива: 32000 Время выполнения: 0.055486 Рекурсии: 42787 Максимальная глубина: 39 Операции замены: 609607

Тест: Отсортированный массив Размер массива: 64000 Время выполнения: 0.144264 Рекурсии: 85311 Максимальная глубина: 36 Операции замены: 1173411

Тест: Отсортированный массив Размер массива: 128000 Время выполнения: 0.226488 Рекурсии: 170785 Максимальная глубина: 41 Операции замены: 2563570

Тест: Массив с одинаковыми элементами Размер массива: 1000 Время выполнения: 0.000069 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 2000 Время выполнения: 0.000112 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 4000 Время выполнения: 0.000194 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 8000 Время выполнения: 0.000409 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 16000 Время выполнения: 0.000807 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 32000 Время выполнения: 0.001595 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 64000 Время выполнения: 0.003153 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

Тест: Массив с одинаковыми элементами Размер массива: 128000 Время выполнения: 0.006202 Рекурсии: 3 Максимальная глубина: 1 Операции замены: 0

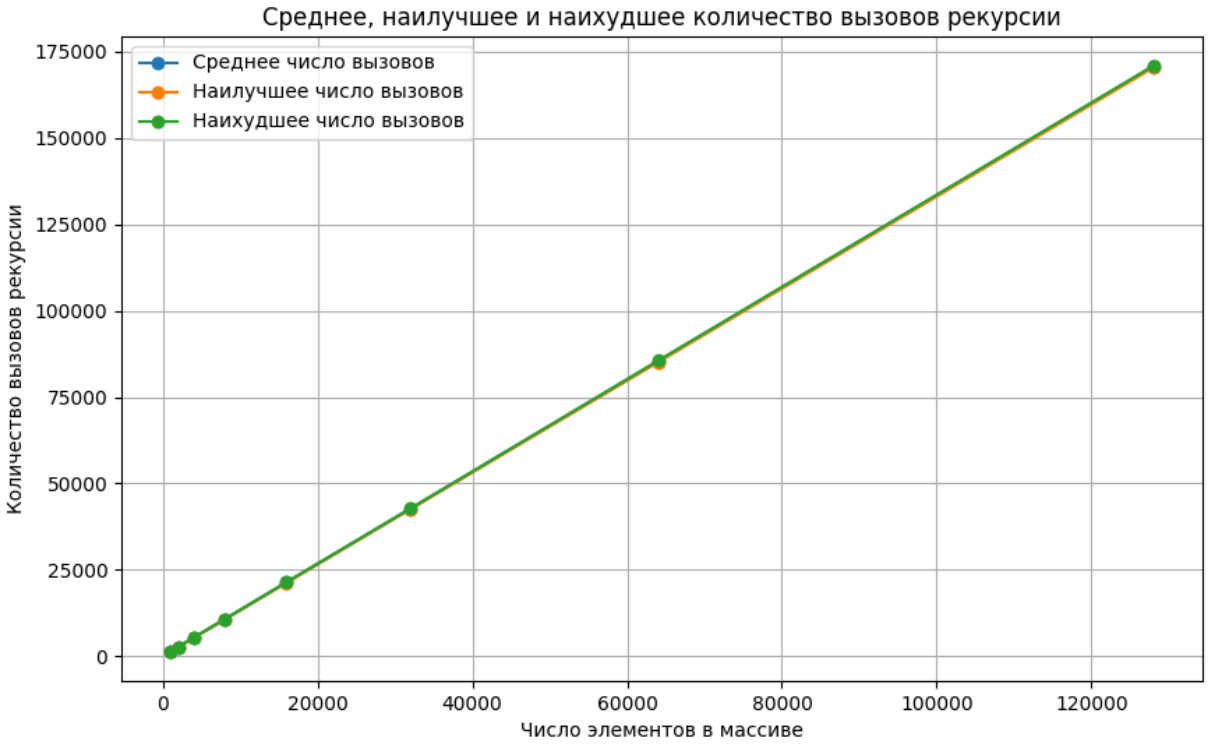
Тест: Обратно отсортированный массив (Средний как pivot) Размер массива: 1000 Время выполнения: 0.001377 Рекурсии: 1329 Максимальная глубина: 21 Операции замены: 11321

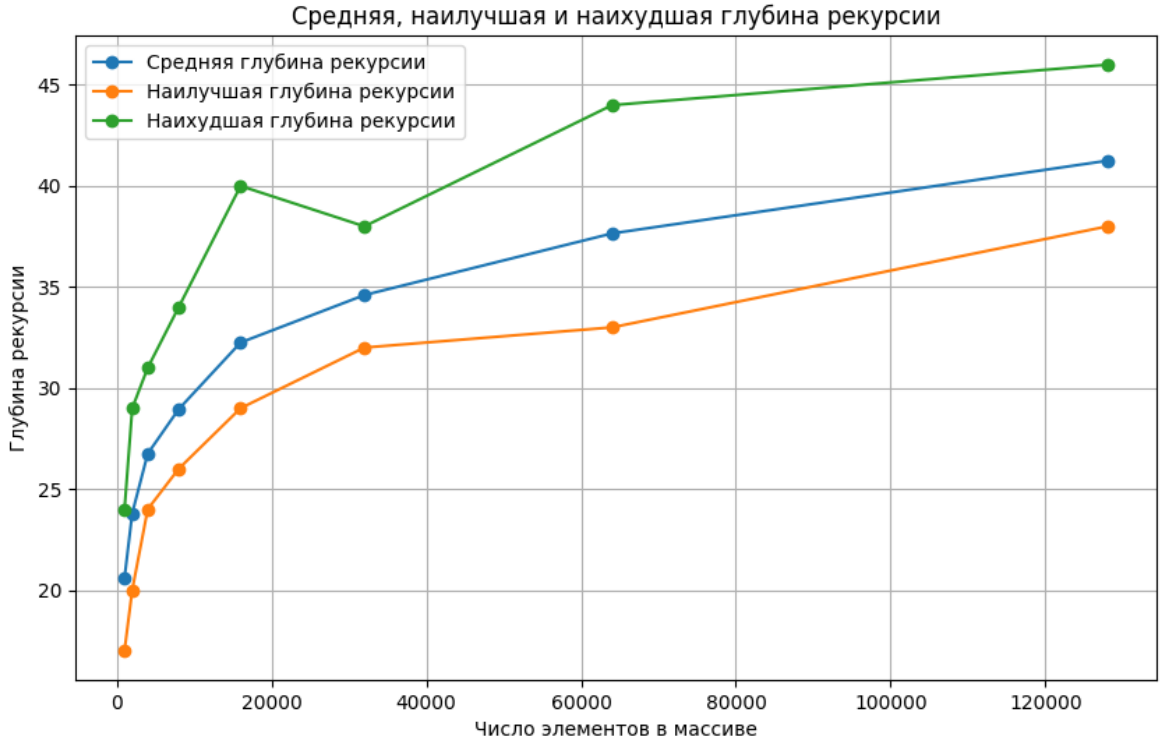
Тест: Обратно отсортированный массив (Средний как pivot) Размер массива: 2000 Время выполнения: 0.003488 Рекурсии: 2699 Максимальная глубина: 26 Операции замены: 28760

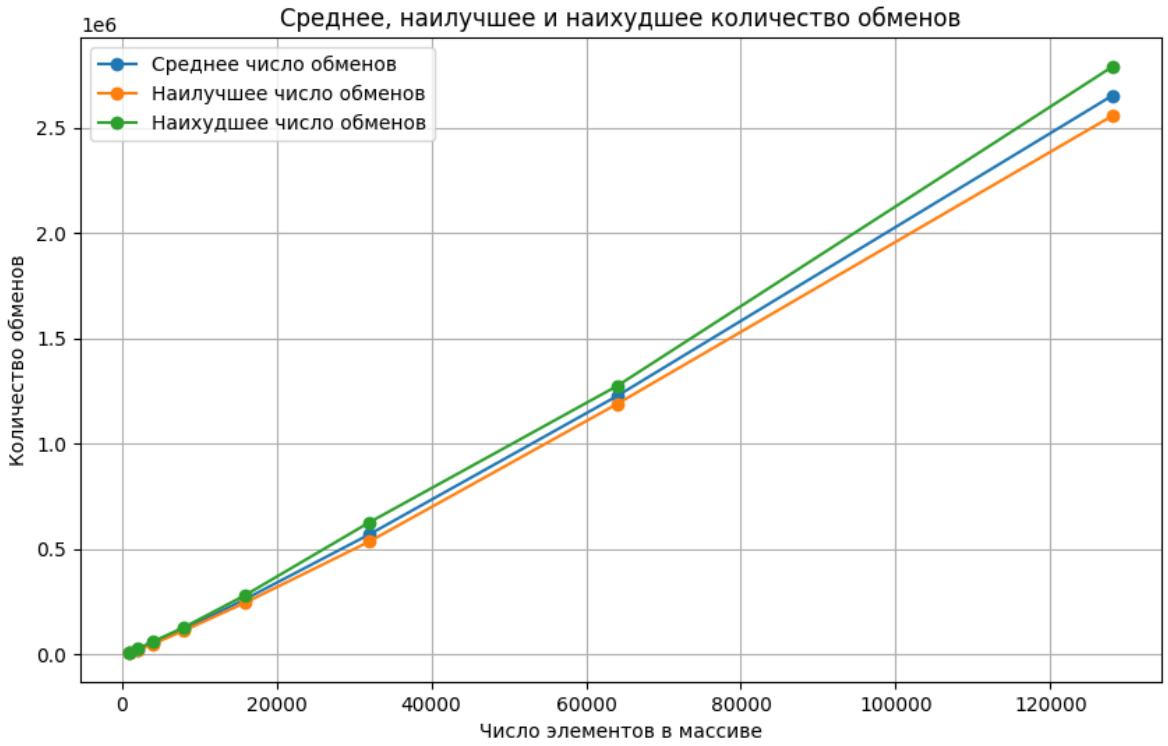
Тест: Обратно отсортированный массив (Средний как pivot) Размер массива: 4000 Время выполнения: 0.006110 Рекурсии: 5361 Максимальная глубина: 27 Операции замены: 58707

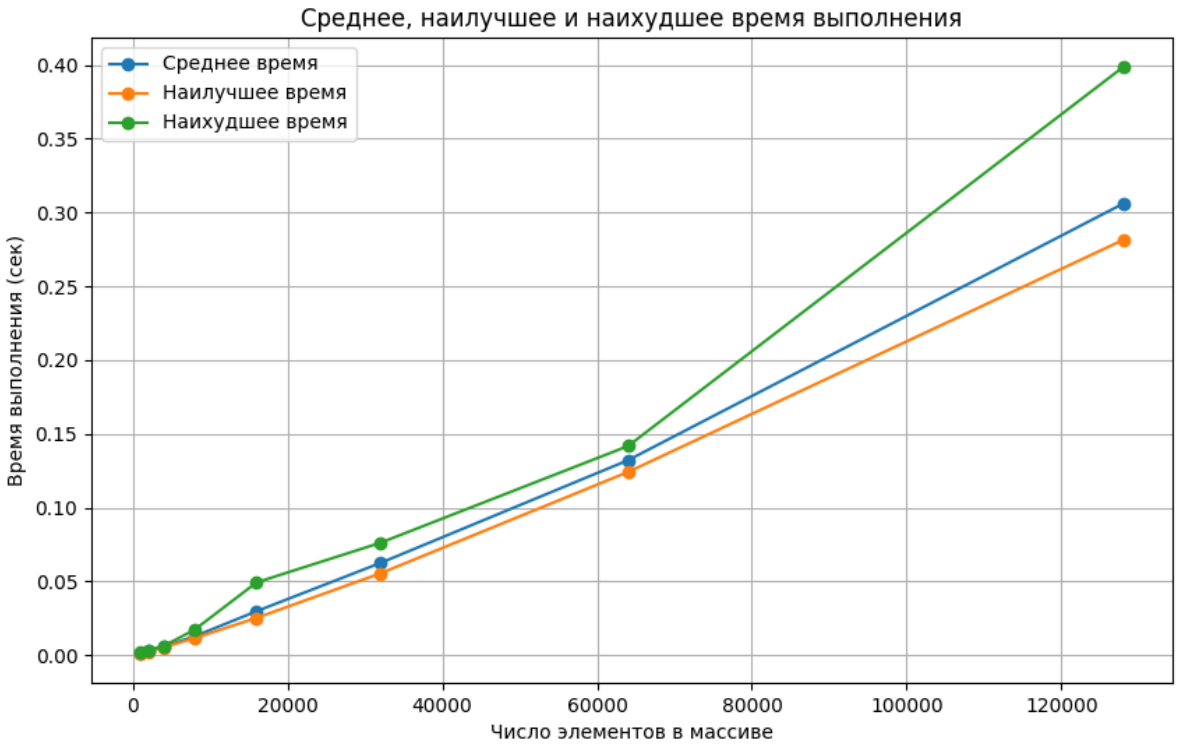
Тест: Обратно отсортированный массив (Средний как pivot) Размер массива: 8000 Время выполнения: 0.013179 Рекурсии: 10691 Максимальная глубина: 32 Операции замены: 119888

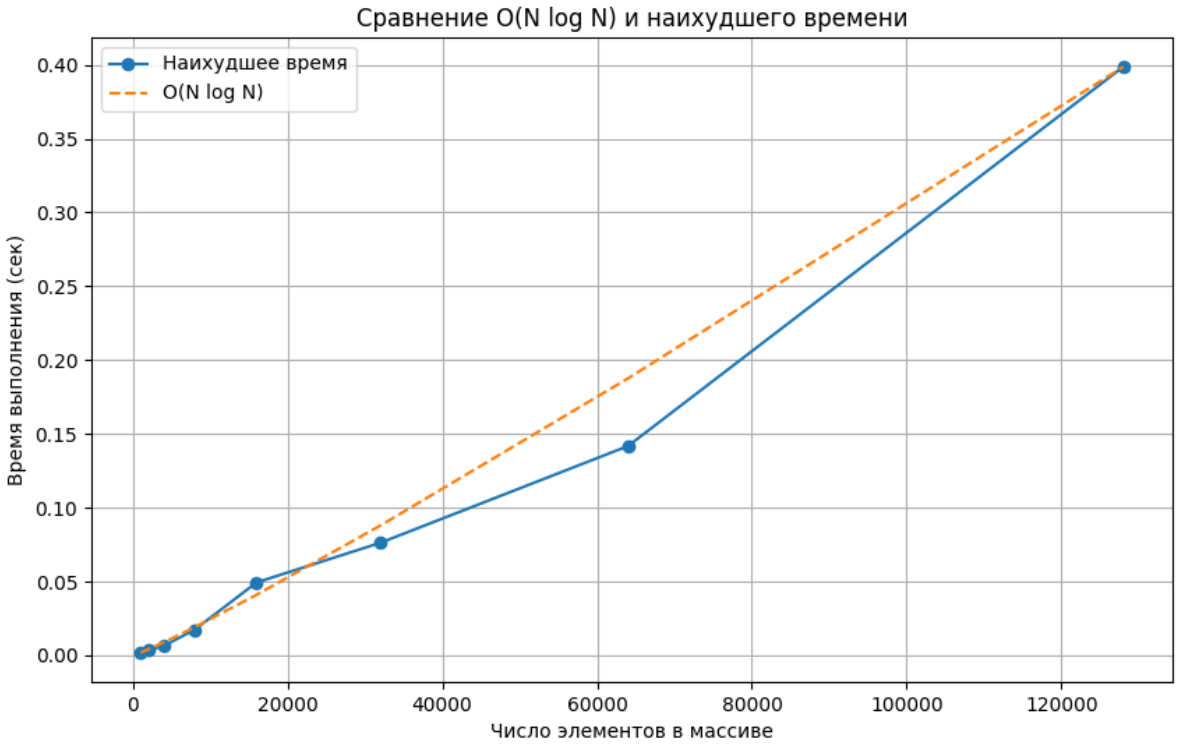
Полученный фал с данными после 20 итераций каждой из серий размерностей (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000) импортируется в Ру скрипт для анализа и построения графиков зависимостей полученных результатов. Графики приведены ниже.






Среднее, наилучшее и наихудшее количество обменов
1e6
Среднее число обменов
Наилучшее число обменов
Наихудшее число обменов
Количество обменов
0.0
0.5
1.0
1.5
2.0
2.5
0
20000
40000
60000
80000
100000
120000
Число элементов в массиве


Среднее, наилучшее и наихудшее время выполнения
Среднее время
Наилучшее время
Наихудшее время
Время выполнения (сек)
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0
20000
40000
60000
80000
100000
120000
Число элементов в массиве



Заключение.

По проделанной работе можно понять, что работа быстрой сортировки сильно упрощает и ускоряет сортировку крупных массивов данных. Данный алгоритм подходит для быстрой сортировки массивов неизвестной размерности, так как его среднее время работы O(n log(n)), а худшее O(n^2). Таким образом можно сказать, что в среднем быстрая сортировка показывает неплохой результат и подходит для больших размерностей массивов.