

## Порядок выполнения работы:

1. Написал программу (heap\_sort.py), Проведено исследование, в рамках которого был реализован алгоритм сортировки кучи, а также измерено время работы данного алгоритма в случае, когда на вход поступают: отсортированный массив, массив, обратный отсортированному, и массив с произвольными значениями.:

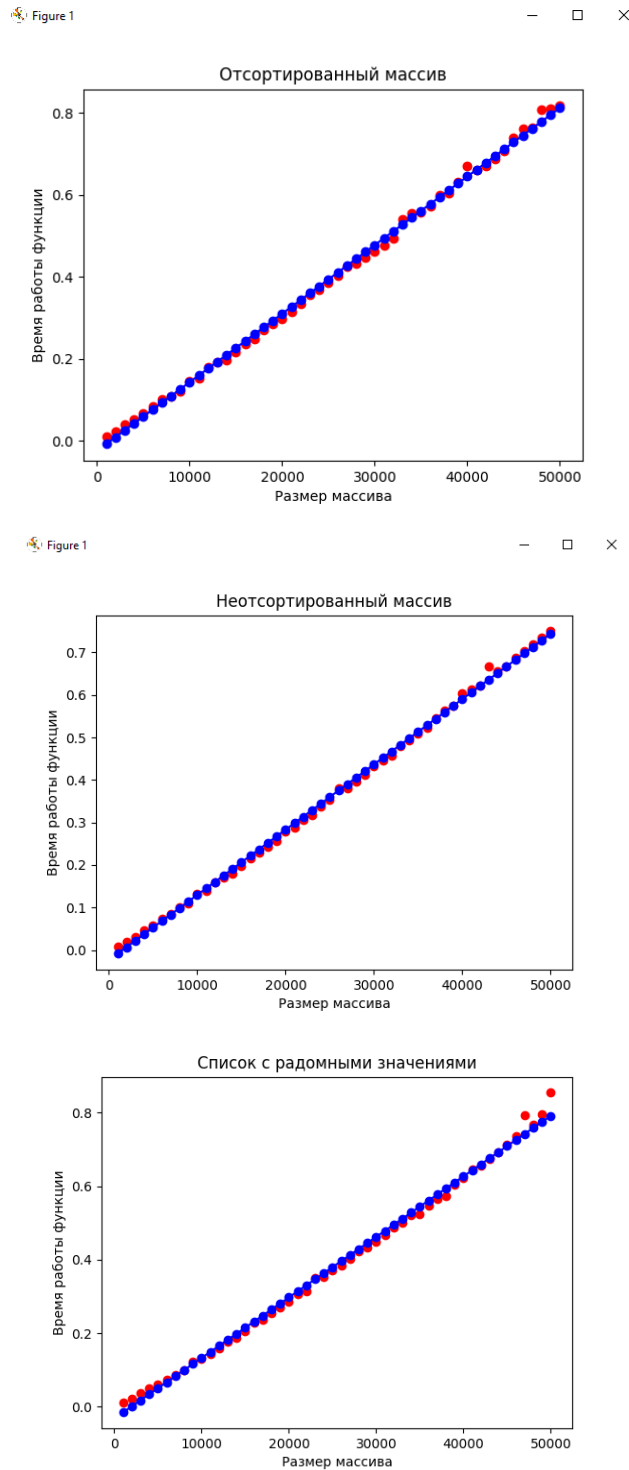


Рисунок 1 – Графики зависимости длины массива от времени

## 2. Сравнение с другими сортировками

Случай	Лучший	Средний	Худший
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Алгоритм сортировки кучей (Heap Sort) имеет временную сложность  $O(n \log n)$  в лучшем, среднем и худшем случае, что делает его эффективным для сортировки больших объемов данных. Однако он не является стабильным, что может быть недостатком в определенных сценариях.

Алгоритм сортировки слиянием (Merge Sort) также обладает временной сложностью  $O(n \log n)$  в лучшем, среднем и худшем случае, что делает его эффективным для сортировки больших объемов данных. Его преимуществом является стабильность, что означает, что одинаковые элементы не меняют свой порядок относительно друг друга. Однако, для слияния массивов этот алгоритм требует дополнительной памяти, что может быть недостатком при работе с большими объемами данных.

Алгоритм быстрой сортировки (Quick Sort) имеет временную сложность  $O(n \log n)$  в среднем и в лучшем случае, но в худшем случае его временная сложность составляет  $O(n^2)$ . Quick Sort обычно эффективен на практике и широко используется из-за своей высокой производительности в среднем случае. Однако в худшем случае, когда выбор опорного элемента не оптимален, производительность алгоритма может снизиться до квадратичной сложности.

3. Написал программу (heap\_sort\_upgrade.py), исследование включало реализацию алгоритма сортировки кучи и измерение времени работы в случае, когда на вход поступают: отсортированный список, список, который обратный отсортированному, и список с случайными значениями.:

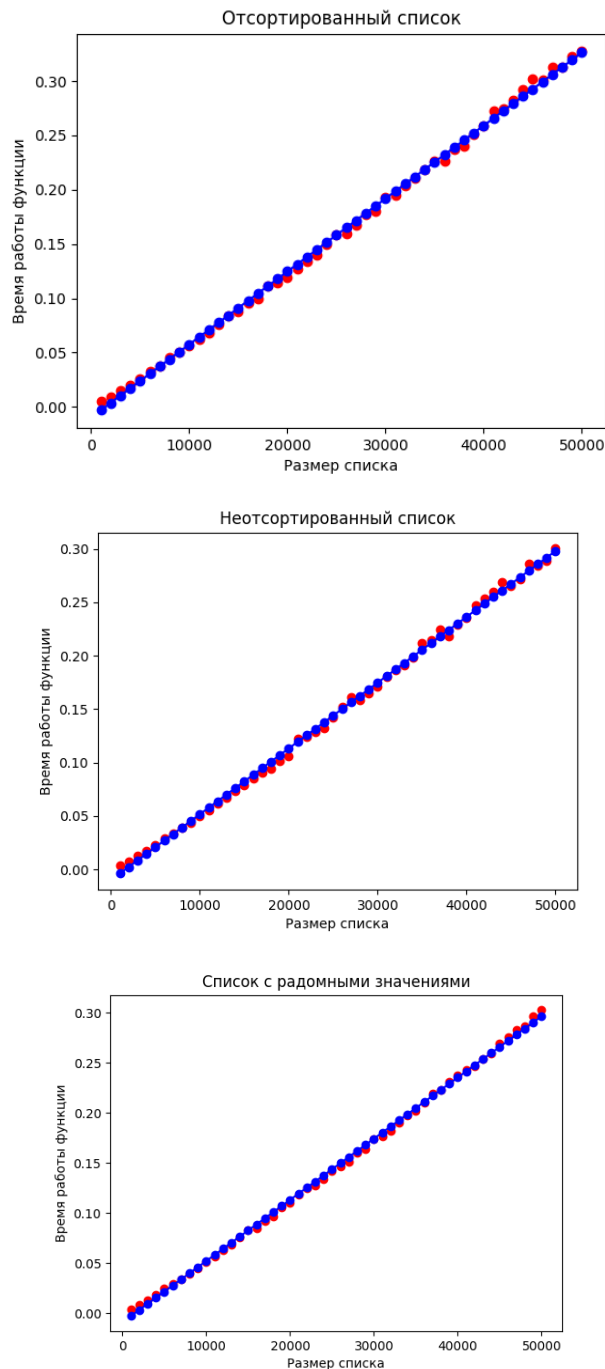


Рисунок 2 – Графики зависимости длины списка от времени

#### 4. Применение в реальной жизни

Алгоритм сортировки кучей может быть использован в реальных сценариях, таких как оптимизация работы баз данных, обработка событий и другие вычислительные задачи. В определенных ситуациях он может быть предпочтительным выбором благодаря гарантированному времени выполнения в худшем случае. Например, в базах данных, где необходим быстрый доступ к отсортированным данным, сортировка кучей может быть

полезна из-за своей эффективности и предсказуемости времени выполнения. Также в сценариях, где требуется сортировка больших объемов данных с важным гарантированным временем выполнения, алгоритм сортировки кучей может быть предпочтительным выбором.

## 5. Анализ сложности

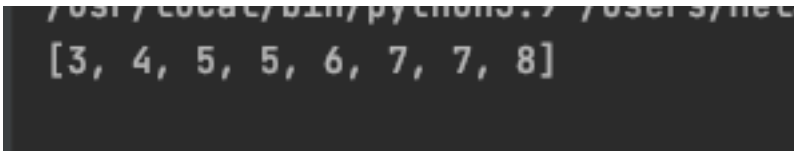
Алгоритм сортировки кучей имеет временную сложность  $O(n \log n)$  в худшем, лучшем и среднем случае. Пространственная сложность составляет  $O(1)$ , что означает, что дополнительная память, используемая для сортировки, не зависит от размера входных данных.

С увеличением размера входных данных время выполнения алгоритма Heap Sort увеличивается логарифмически, что делает его эффективным для больших объемов данных. Однако, по сравнению с другими алгоритмами сортировки, такими как Quick Sort или Merge Sort, в некоторых случаях Heap Sort может быть менее эффективным из-за постоянных операций с памятью и более сложной реализации.

Таким образом, алгоритм сортировки кучей может быть более эффективным в случаях, когда требуется гарантированное время выполнения в худшем случае и при работе с большими объемами данных, но может быть менее эффективным по сравнению с другими алгоритмами сортировки в некоторых сценариях, где важна простота реализации и минимальное использование памяти.

6. Даны массивы  $A[1 \dots n]$  и  $B[1 \dots n]$ . Мы хотим вывести все  $n^2$  сумм вида  $A[i] + B[j]$  в возрастающем порядке. Наивный способ — создать массив, содержащий все такие суммы, и отсортировать его. Соответствующий алгоритм имеет время работы  $O(n^2 \log n)$  и использует  $O(n^2)$  памяти. Приведите алгоритм с таким же временем работы, который использует линейную память.

Написал программу (task6.py), которая решает данную задачу:



```
7031 / code42 / bin / python3.7 / user3 / hec  
[3, 4, 5, 5, 6, 7, 7, 8]
```

Рисунок 3 – Результат выполнения программы task6.py

Код программы:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
def linear_memory_sum(A, B):  
    n = len(A)  
    A.sort()  
    B.sort()  
    min_heap = []  
  
    min_heap.append((A[0] + B[0], 0, 0))  
    result = []  
  
    for _ in range(n * n - 1):  
        a_sum, i, j = min_heap.pop(0)  
        result.append(a_sum)  
  
        if j < n - 1:  
            min_heap.append((A[i] + B[j+1], i, j+1))  
        if j == 0 and i < n - 1:  
            min_heap.append((A[i+1] + B[j], i+1, j))  
  
        min_heap.sort()  
  
    return result  
  
def main():  
    A = [3, 1, 2]  
    B = [2, 4, 6]  
    result = linear_memory_sum(A, B)  
    print(result)  
  
if __name__ == "__main__":  
    main()
```

Вывод: в результате лабораторной работы было изучено использование алгоритма сортировки кучей, и было установлено, что этот алгоритм работает быстрее с списками данных, чем с массивами.