Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

# ОТЧЕТ
# ПО ЛАБОРАТОРНОЙ РАБОТЕ №10
## дисциплины «Алгоритмизация»

Выполнил:
Епифанов Алексей Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. Написал программу сравнения функций сортировки кучей: стандартной, улучшенной и использующей мин-кучу, а не макс.

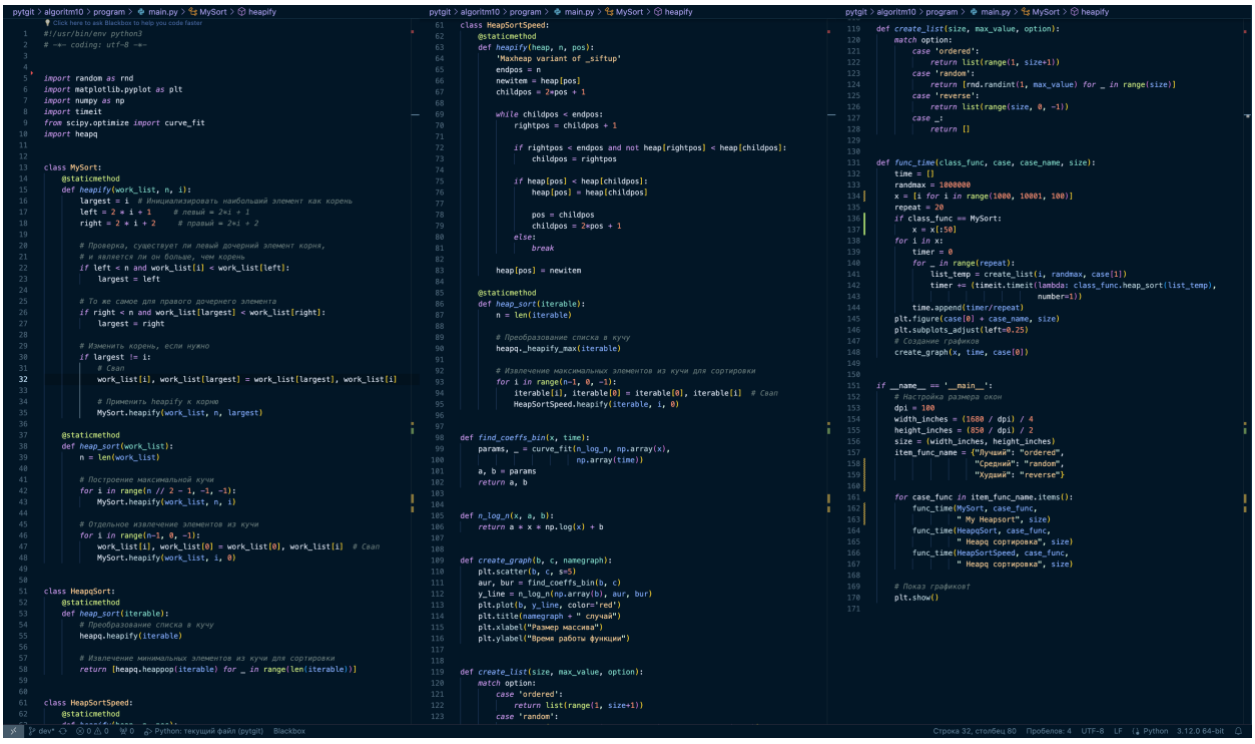

Рисунок 1. Код программы

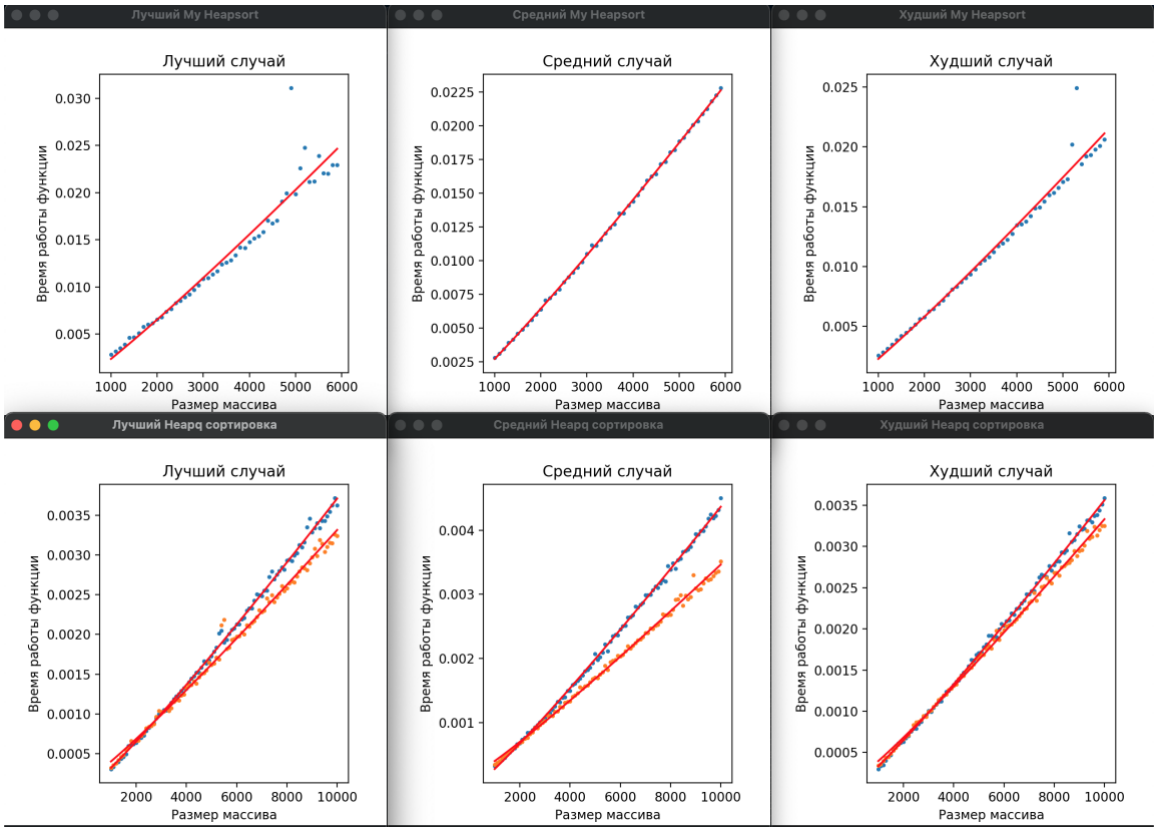


Рисунок 2. Результат работы программы

2. Сравнение с другими сортировками:

| Алгоритм | Лучший случай | В среднем | Худший случай |
|---|---|---|---|
| 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) |

3. Применение в реальной жизни:

Heap Sort может быть предпочтительным выбором, когда требуется эффективная сортировка больших массивов данных, так как его пространственная сложность = O(1), а значит при любых размерах входных данных алгоритм будет требовать минимального выделения пространства для него, размер которого никак не зависит от размера входного массива. Также алгоритм применим когда важна стабильность процесса сортировки в плане производительности, так как он гарантирует время сортировки O(n log n) независимо от начального порядка элементов.

4. Оценка времени и пространственной сложности:

Heapsort имеет временную сложность O(n log n)

Построение максимальной кучи:

```
for i in range(n // 2 - 1, -1, -1):
    heapify(work_list, n, i)
```

heapifi() Выполняется за O(log n) так как внутри содержит операции, выполняющиеся за O(1), и может вызвать себя максимум log n раз, потому что высота дерева log n.

Цикл выполняется примерно n/2 раз, но в сумме построение максимальной кучи выполняется за O(n).

Отдельное извлечение элементов из кучи:

```
for i in range(n-1, 0, -1):
    work_list[i], work_list[0] = work_list[0], work_list[i]  # Свап
    heapify(work_list, i, 0)
```

heapifi работает n раз, каждый раз выполняется за O(log n), а значит в сумме получается O(n log n)

Конечное время получается: O(n + n log n) что равно O(n log n).

Пространственная сложность данного алгоритма равна O(1), так как функция не требует дополнительной памяти и работает с данным списком(за исключением локальных переменных, которые не влияют на сложность, так как не меняются в зависимости от размера списка).

5. Написал программу по заданию: Даны массивы A[1…n] и B[1…n]. Мы хотим вывести все n2 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(n2logn) и использует O(n2) памяти. Приведите алгоритм с таким же временем работы, который использует линейную память.

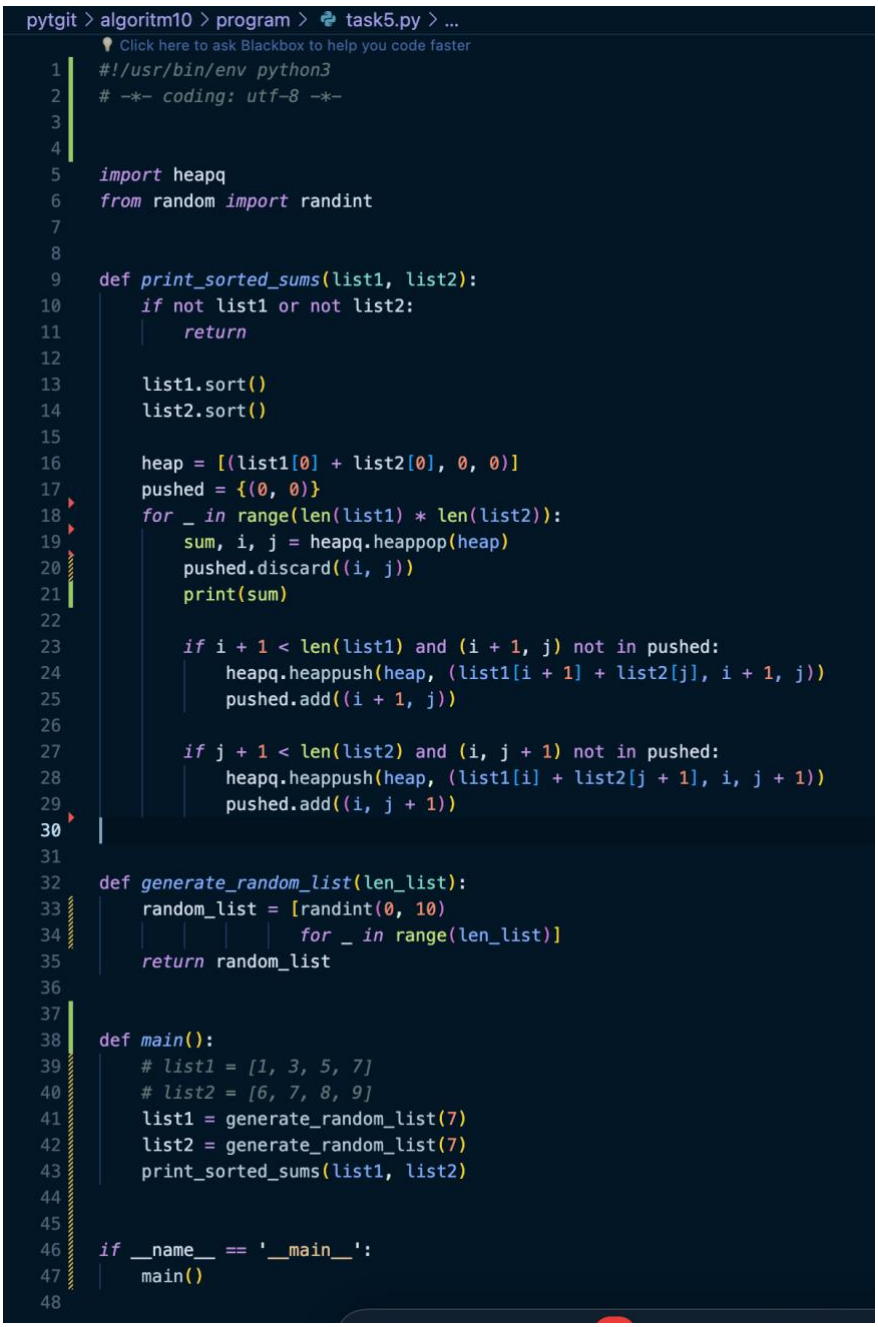


Рисунок 3. Код программы

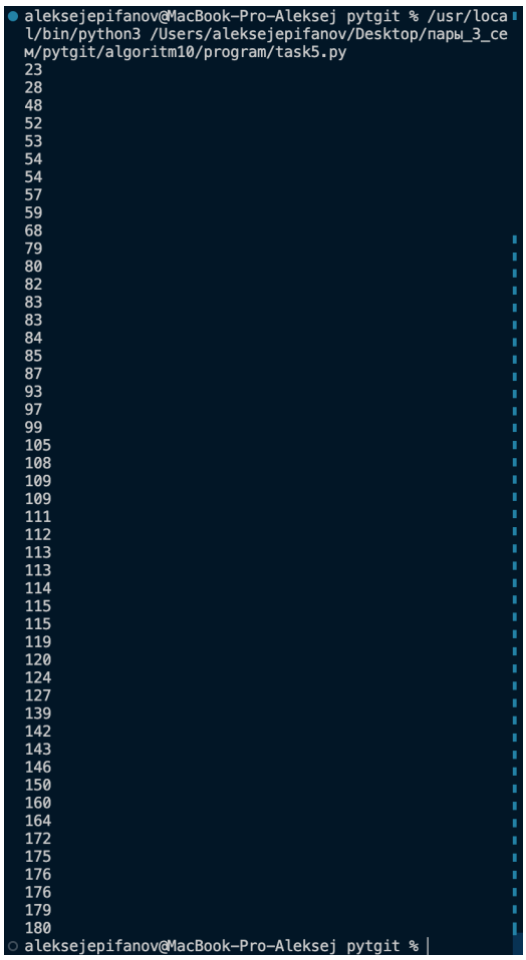


Рисунок 4. Результат работы программы

Данная программа работает за время $O(n^2 \log n)$ и пространственную сложность $O(n)$, так как сначала выполняется сортировка двух списков на месте за время $O(n \log n)$, затем создается мин-куча из кортежей (сумма, индекс элемента из первого списка, индекс элемента из второго списка) и список для отслеживания уже помещенных в кучу сумм. В списке для отслеживания всегда столько же элементов, сколько и в куче.

Далее в кучу последовательно добавятся все n*n пар элементов, каждое добавление будет выполняться за время $O(\log n)$, а также удалятся из кучи все эти элементы, каждое удаление также будет выполняться за время $O(\log n)$.

Суммарное время выполнения функции будет $O(2n \log n + n^2 \log n)$, что тоже самое, что и $O(n^2 \log n)$.

Пространственная сложность по итогу будет $O(4(n-1))$, что тоже самое что и $O(n)$.

Это из-за того, что в куче всегда хранится не более чем 2(n-1) элементов, и столько же в списке, так как добавляются только те пары, сумма которых потенциально может быть больше извлеченной, то есть при извлечении из мин-кучи  суммы пары i, j в кучу добавляются суммы пар i+1, j и i, j+1, если они есть и еще не помещены в кучу.

А так как i ограничена длинной первого списка, а j – второго, то размер кучи не превзойдет суммы длин обоих списков.

В ходе выполнения лабораторной работы было исследовано 3 разных алгоритма сортировки кучей. Первый, и самый худший написан сразу после прочтения механизма работы данной сортировки, второй алгоритм использует встроенную в python библиотеку heapq, которая реализует мин-кучу; он уже работает намного быстрее (примерно в 10 раз). Третий же алгоритм использует некоторую часть кода из библиотеки heapq, но немного переделанную; это привело к тому, что алгоритм при маленьких размерах входного списка (до 4000) уступает второму алгоритму, но далее в каждом случае он обгоняет его. Из полученных результатов можно сделать следующий вывод: для эффективной реализации алгоритма сортировки кучей лучше всего воспользоваться или функциями из библиотеки heapq, или воспользоваться кодом этой библиотеки в качестве основы для собственной реализации.