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»

Выполнила:
Беседина Инга Олеговна
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

1. **Реализация алгоритма Heap Sort:** Спроектируйте и реализуйте алгоритм сортировки кучей (Heap Sort) на любом удобном для вас языке программирования. После этого протестируйте вашу реализацию на различных видах входных данных, таких как отсортированный массив, массив в обратном порядке и случайный массив. Оцените эффективность алгоритма в каждом случае.

- Лучший случай: в лучшем случае алгоритм heap sort имеет время выполнения O(nlogn), что делает его эффективным даже при уже отсортированных данных.

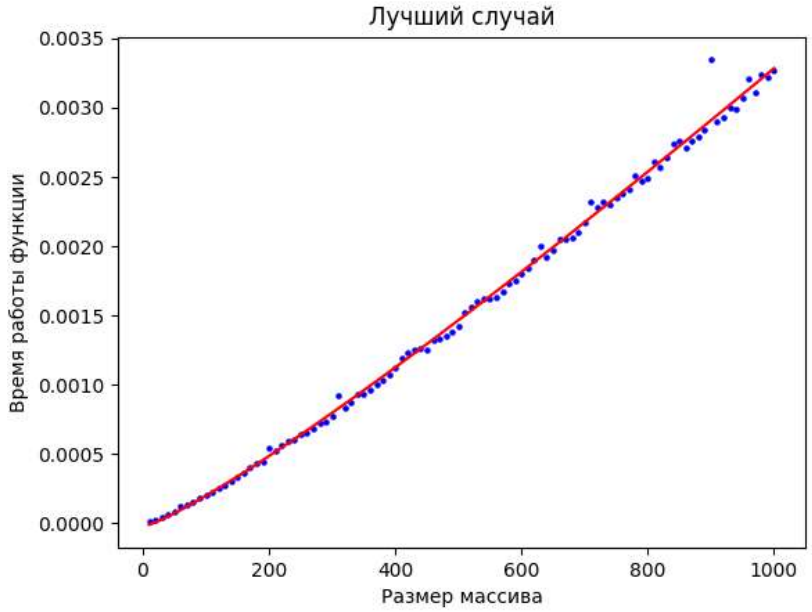

Рисунок 1. Зависимость времени выполнения алгоритма от количества элементов в списке

- Средний случай: в среднем случае алгоритм heap sort также имеет время выполнения O(nlogn), что делает его хорошим выбором для средних случаев распределения элементов.

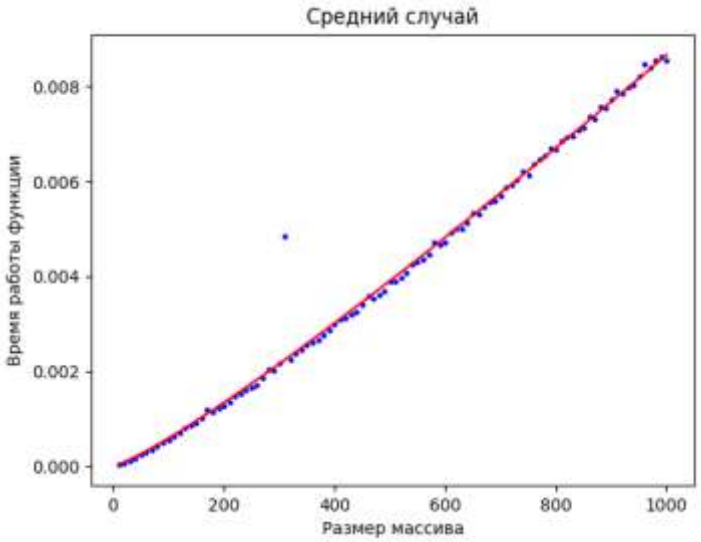

Рисунок 2. Зависимость времени выполнения алгоритма от количества элементов в списке

- Худший случай: в худшем случае алгоритм heap sort также имеет время выполнения O(nlogn), что означает, что он эффективен даже в случае, когда данные находятся в обратном порядке или уже отсортированы в обратном порядке.

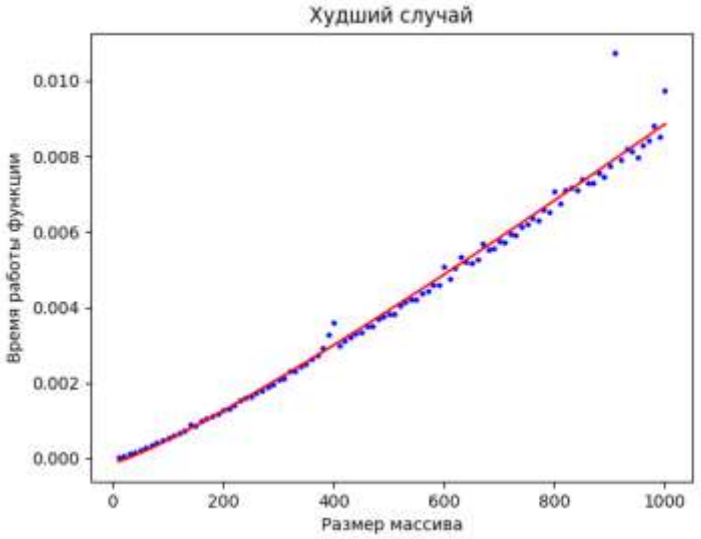

Рисунок 3. Зависимость времени выполнения алгоритма от количества элементов в списке

2. **Оптимизация алгоритма:** Попробуйте оптимизировать алгоритм Heap Sort. Исследуйте возможности улучшения

производительности, например, путем использования оптимизированных структур данных или алгоритмических подходов. Сравните вашу оптимизированную реализацию с базовой версией.

Использовать min-heap для сортировки списка с помощью heapsort:

```
def min_heap_sort(iterable):
    h = []
    for value in iterable:
        heapq.heappush(h, value)

    return [heapq.heappop(h) for i in range(len(h))]
```

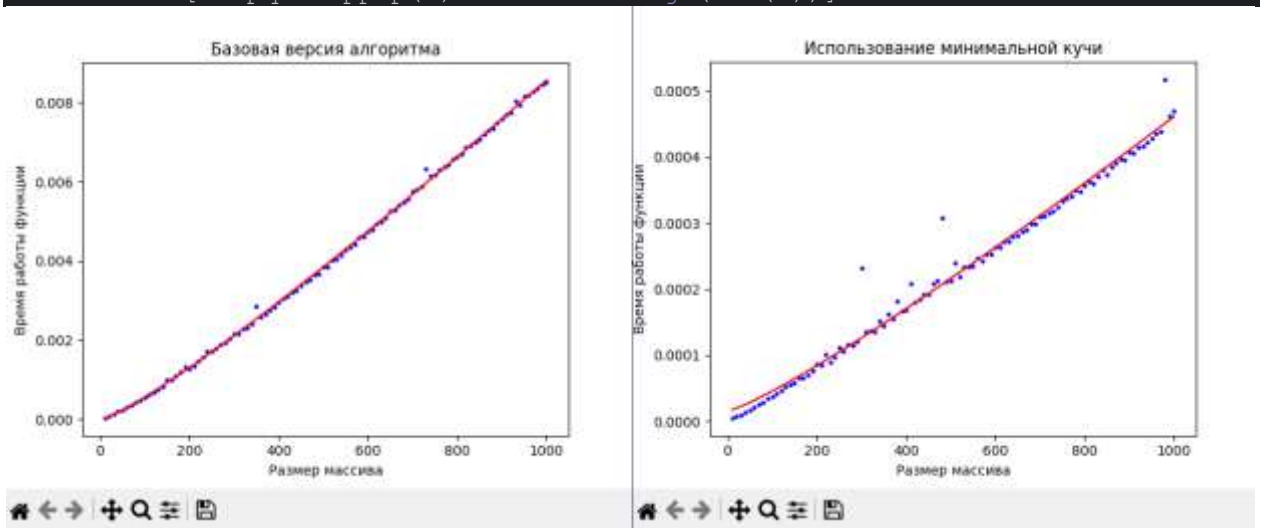

Рисунок 4. Сравнение оптимизированной реализации с базовой версией

3. **Сравнение с другими сортировками:** Сравните производительность Heap Sort с другими известными алгоритмами сортировки, такими как Quick Sort и Merge Sort.

1. Время выполнения:

- Quick sort: в среднем случае время выполнения составляет O(nlogn), но в худшем случае может быть квадратичным ($O(n^2)$).

- Merge sort: время выполнения всегда составляет O(nlogn), что делает его очень эффективным.

- Heap sort: время выполнения также составляет O(nlogn) в худшем, лучшем и среднем случаях.

2. Пространственная сложность:

- Quick sort: Требует O(logn) дополнительной памяти для рекурсивных вызовов.

- Merge sort: Требует O(n) дополнительной памяти для временного массива при слиянии.

- Heap sort: Требует только O(1) дополнительной памяти.

Heap sort может быть полезным в ряде практических сценариев благодаря своим характеристикам:

4. **Применение в реальной жизни:** Рассмотрите практические применения Heap Sort в реальных сценариях. Например, как этот алгоритм может быть использован для оптимизации работы баз данных, событийной обработки или других вычислительных задач. Объясните, почему Heap Sort может быть предпочтительным выбором в некоторых ситуациях.

1. Событийная обработка: в системах обработки событий, где необходимо обрабатывать события в порядке их приоритета, heap sort может быть использован для эффективной сортировки событий по их приоритету. Например, в системе управления задачами или планирования ресурсов, где необходимо обрабатывать задачи в порядке их важности или срочности.

2. Оптимизация работы баз данных: в базах данных, где необходимо отсортировать большие объемы данных, heap sort может быть предпочтительным выбором из-за его стабильной временной сложности O(nlogn) в худшем случае. Это может быть полезно, например, при сортировке результатов запросов или индексации данных.

3. Планирование задач: в системах управления ресурсами и планирования задач, heap sort может быть использован для эффективного планирования и распределения ресурсов в соответствии с их приоритетами.

4. Приоритетные очереди: Heap sort может быть использован для реализации приоритетных очередей, где элементы извлекаются в порядке их приоритета.

Heap sort предпочтителен в таких ситуациях из-за своей стабильной временной сложности в худшем случае, отсутствия необходимости

дополнительной памяти (O(1) дополнительной памяти), а также возможности эффективно работать с большими объемами данных.

5. **Анализ сложности:** Проведите анализ времени выполнения и пространственной сложности алгоритма Heap Sort. Исследуйте, как эти характеристики зависят от размера входных данных. Сделайте выводы о том, в каких случаях Heap Sort может быть более или менее эффективным по сравнению с другими алгоритмами сортировки.

Время выполнения и пространственная сложность алгоритма Heap Sort зависят от размера входных данных.

1. Время выполнения: в среднем и в худшем случае время выполнения Heap Sort составляет O(nlogn), где n - количество элементов в массиве. Это делает его эффективным для сортировки больших объемов данных, так как время выполнения остается стабильным при увеличении размера входных данных.

2. Пространственная сложность: пространственная сложность алгоритма Heap Sort составляет O(1), то есть он требует постоянного количества дополнительной памяти. Это делает его привлекательным для ситуаций, где доступная память ограничена или когда требуется минимизировать использование памяти.

6. 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 $O(n^2 \log n)$ и использует $O(n^2)$ памяти. Приведите алгоритм с таким же временем работы, который использует линейную память.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import heapq
import numpy as np
```

```python
def sort_sums(list1, list2):
    sum_el = []

    list1.sort()
    list2.sort()

    heap = [(list1[0] + list2[0], 0, 0)]
    pushed = {(0, 0)}
    for _ in range(len(list1) * len(list2)):
        s, i, j = heapq.heappop(heap)
        pushed.discard((i, j))
        sum_el.append(s)

        if i + 1 < len(list1) and (i + 1, j) not in pushed:
            heapq.heappush(heap, (list1[i + 1] + list2[j], i + 1, j))
            pushed.add((i + 1, j))

        if j + 1 < len(list2) and (i, j + 1) not in pushed:
            heapq.heappush(heap, (list1[i] + list2[j + 1], i, j + 1))
            pushed.add((i, j + 1))

    return sum_el


def main():
    list1 = np.array(np.random.randint(-100, 100, 5))
    list2 = np.array(np.random.randint(-100, 100, 5))
    print(sort_sums(list1, list2))


if __name__ == '__main__':
    main()
```

```
[-73, -40, -31, -10, -5, -4, 2, 23, 28, 28, 38, 57, 59, 61, 64, 73, 97, 99, 115, 120, 125, 136, 141, 158, 174]
```

Рисунок 5. Результат работы алгоритма

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.

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 $O(n)$.