ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Реализация алгоритмов сортировки»

Выполнил работу

Волков Владимир

Академическая группа

J3112

Принято

Практик Дунаев Максим

Санкт-Петербург

2024

1. Цель работы:  
   Изучить и реализовать три алгоритма сортировки: Cocktail Shaker Sort, HeapSort и Counting Sort. Исследовать их эффективность на практике, проанализировать временную и пространственную сложность.

Задачи:

1. Описать теоретические аспекты алгоритмов.
2. Реализовать алгоритмы и протестировать их на лучших, худших и средних случаях.
3. Построить линейные графики времени выполнения для разных объемов данных.
4. Построить box plot графики для анализа времени выполнения.
5. Проанализировать результаты и сделать выводы.
6. **Ассимтотика и память:**

**Cocktail Shaker Sort:**

**Временная сложность:**

* **Лучший случай:**
  + O(n) — Проходит один цикл от начала до конца и обратно.
* **Худший случай:**
  + O(n^2) — Для каждого элемента выполняется сравнение и, возможно, обмен, проходя n/2 итераций.
* **Средний случай**:
  + O(n^2) — В среднем требуется такое же количество сравнений и обменов.

**Пространственная сложность:**

* O(1) — Алгоритм использует константное количество дополнительной памяти, так как выполняется сортировка на месте. Используются лишь переменные start, end, и временная переменная для std::swap.

**Heap Sort:**

**Временная сложность:**

* **Построение кучи**:
  + Выполняется O(n), так как мы вызываем heapify для всех узлов начиная с последнего родителя до корня.
* **Удаление элементов и восстановление кучи**:
  + Каждый вызов heapify требует O(log n) операций, и он выполняется n раз для всех элементов.
  + Итоговая сложность: O(n\*log n)
* **Общий случай (всегда)**:
  + O(n\*log n).

**Пространственная сложность:**

* Алгоритм выполняется **на месте**, поэтому использует O(1) дополнительной памяти (без учёта входного массива). Рекурсивные вызовы heapify используют стек вызовов, но его глубина ограничена O(log n).

**Counting Sort:**

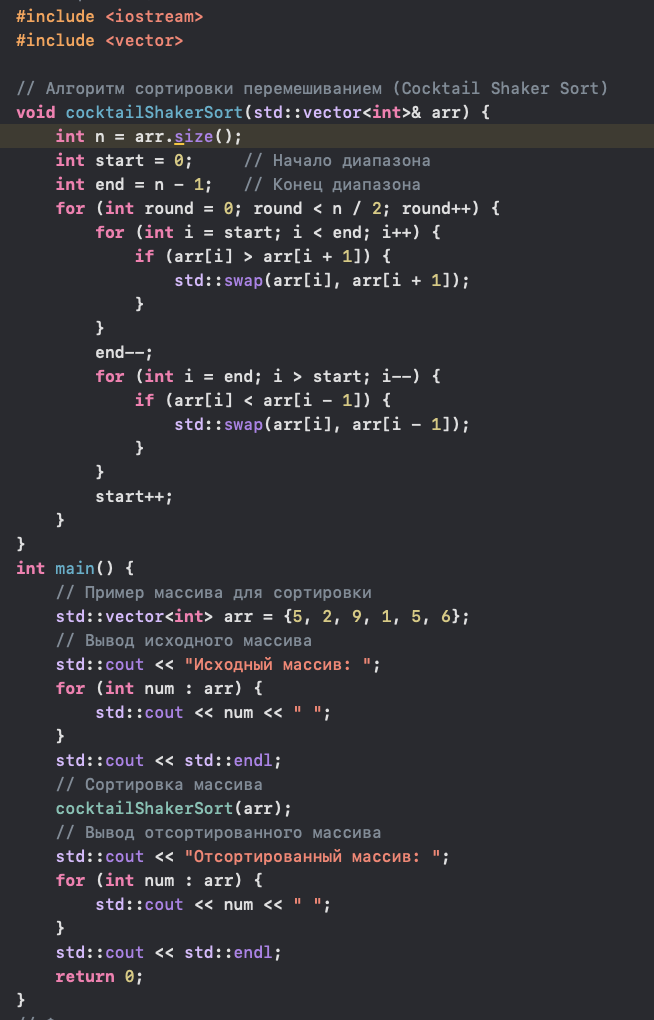
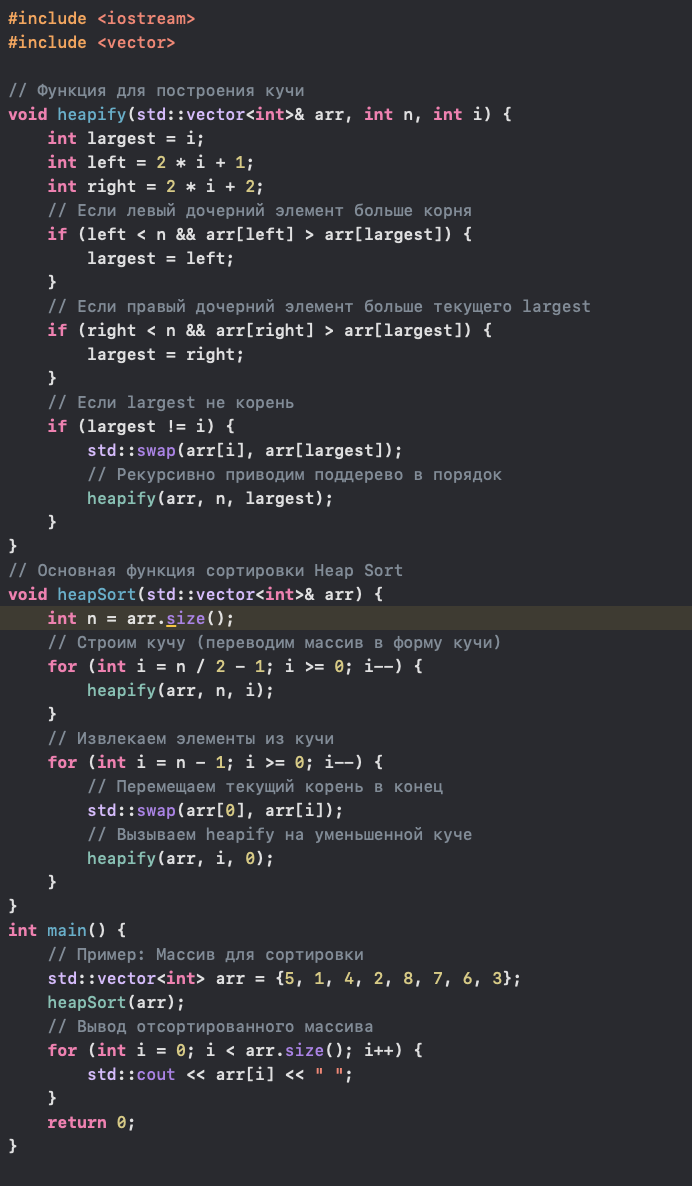
**Временная сложность:**

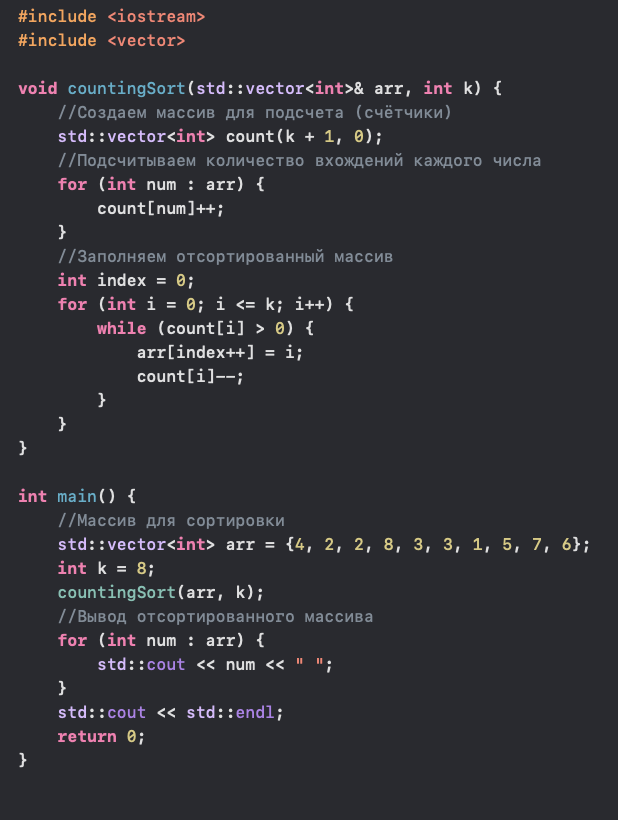
* **Подсчёт вхождений (проход по массиву)**:
  + Операция проходит по каждому элементу массива n, поэтому сложность O(n).
* **Заполнение отсортированного массива (по массиву счётчиков)**:
  + Проходим по диапазону чисел от 0 до k, и для каждого числа, в зависимости от его количества, записываем его в отсортированный массив. Это также требует O(n+k), так как для каждого числа из диапазона можно несколько раз записывать его в массив.
* **Общая временная сложность**:
  + O(n+k), где n — размер массива, а k — максимальное значение в массиве.

**Пространственная сложность:**

* Мы используем дополнительный массив count длиной k+1для хранения количества каждого числа, что требует O(k) дополнительной памяти.
* Итоговая пространственная сложность:
  + O(n+k), так как мы используем массив для хранения результатов и счётчиков.

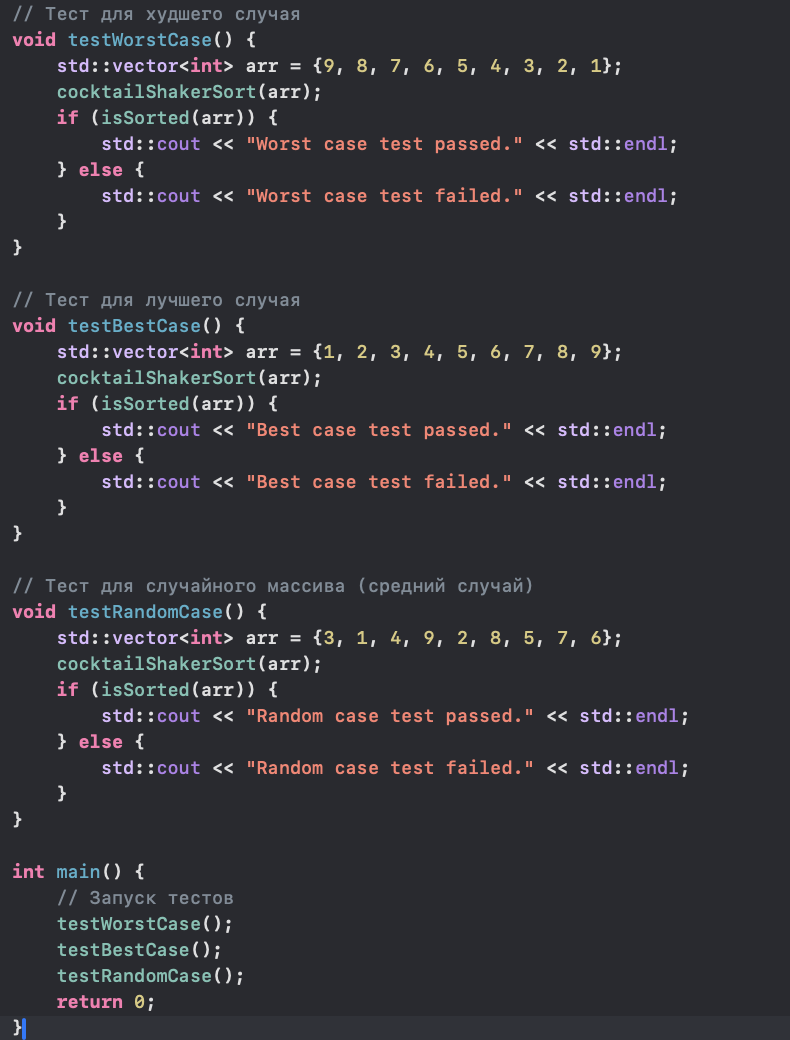
1. **Красивый код**

** **

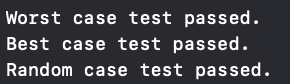
****

1. **Юнит-тесты**

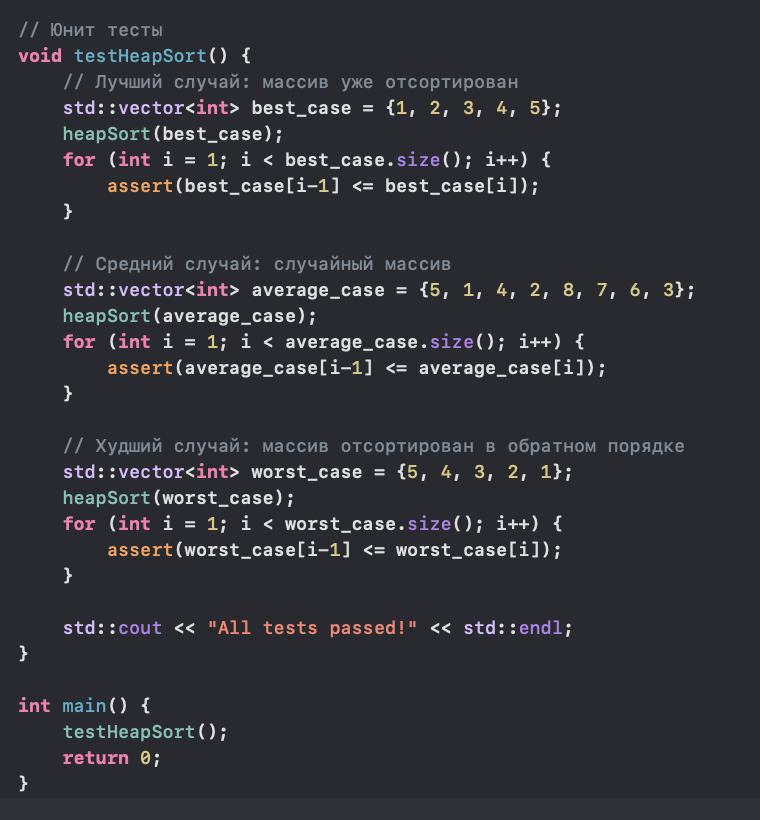
**Тесты для Cocktail Shaker Sort**

****

**Результат:**

****

**Юнит-тесты для HeapSort**

****

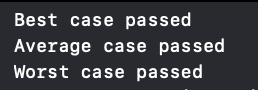
**Результат:**

****

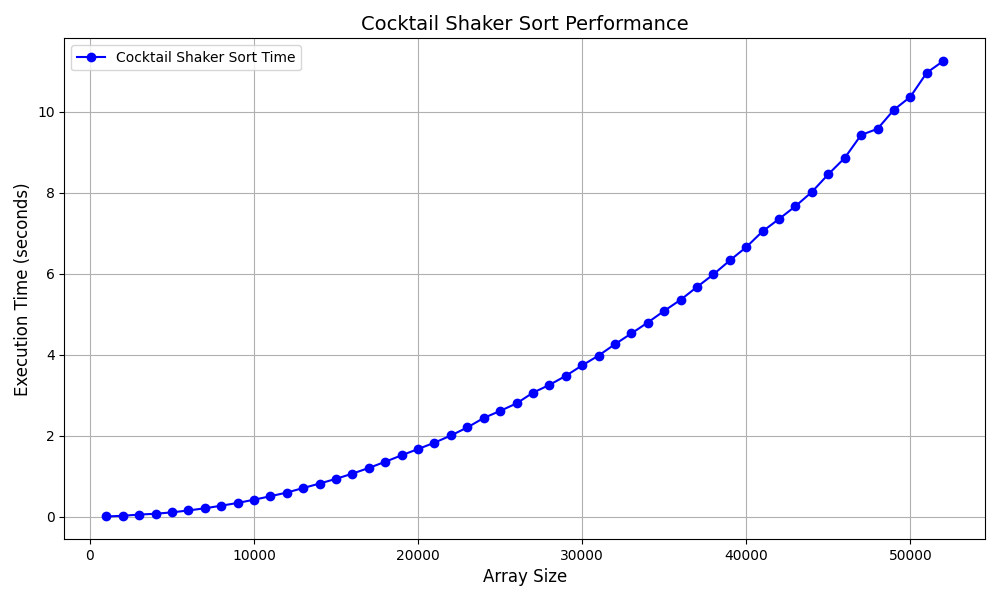
**Юнит-тесты для Counting Sort**

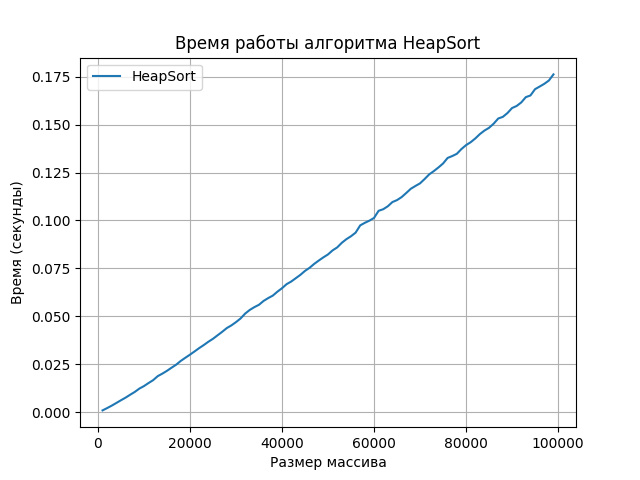
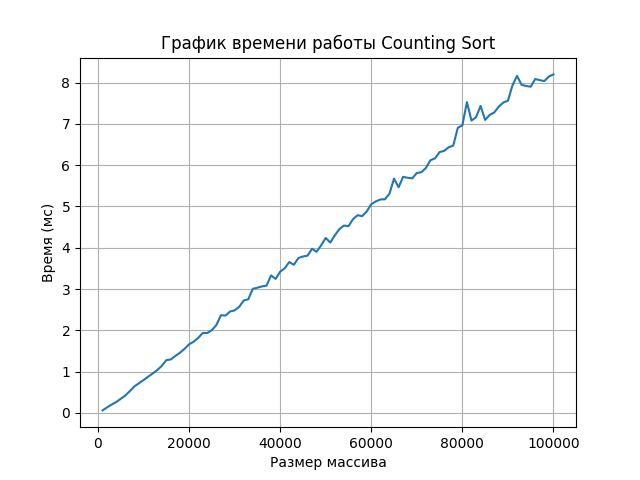
****

**Результат:**

****

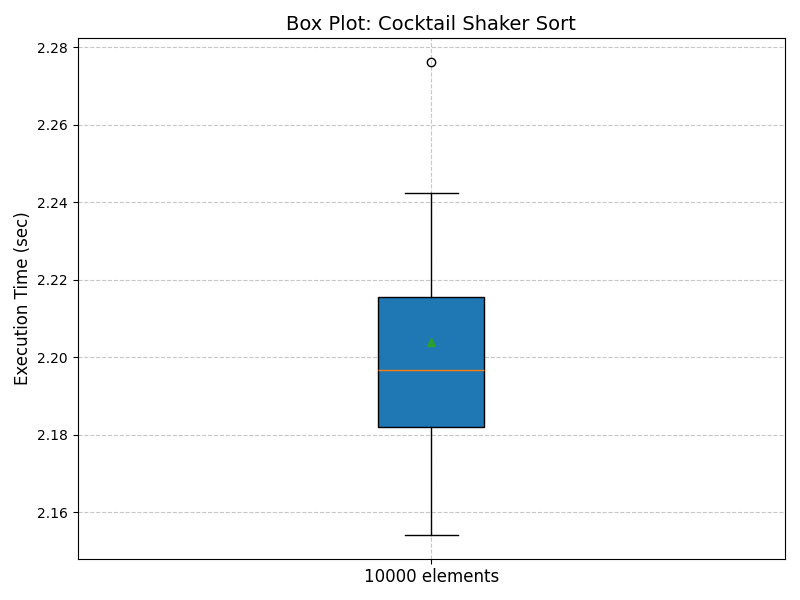
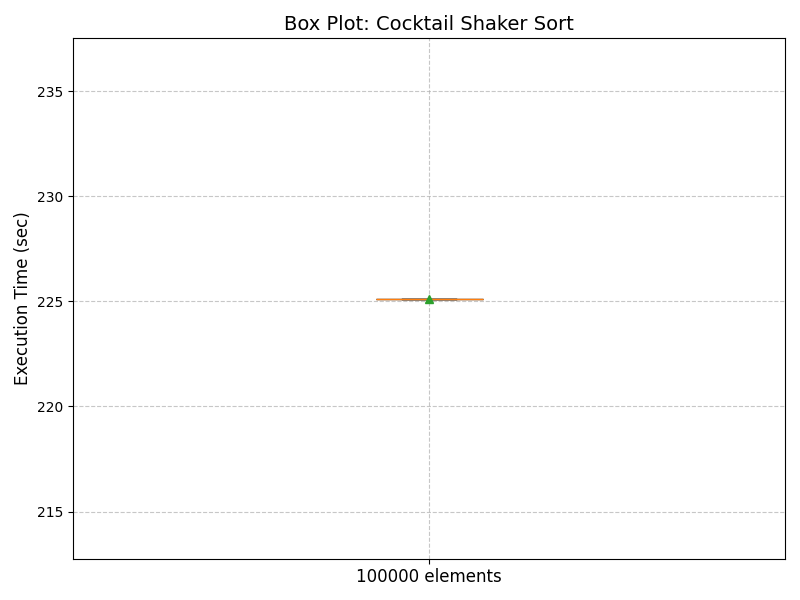
1. **Линейные графики**



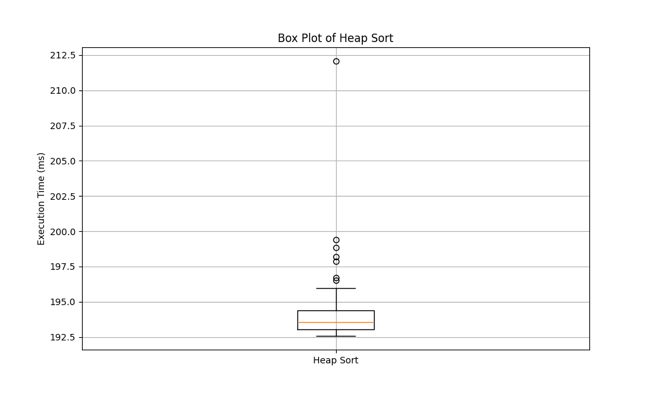
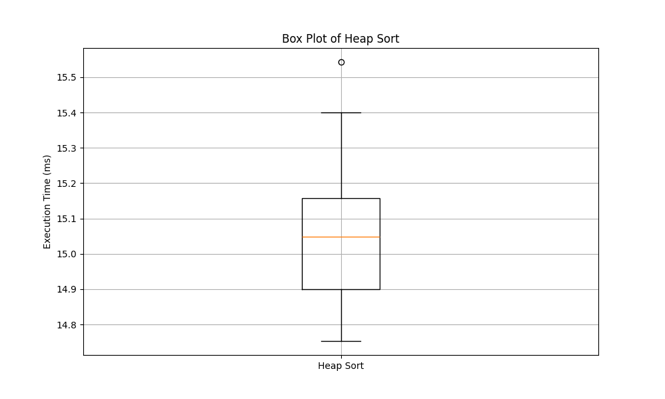
****

1. **Box Plot**

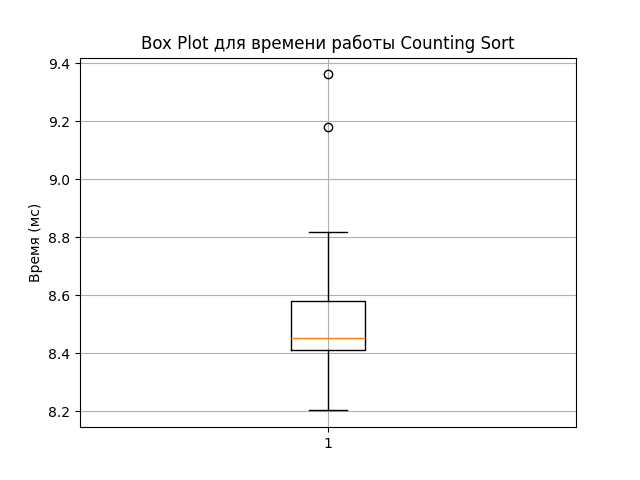
**Cocktail Shaker Sort**



**Heap Sort**



**Counting Sort**



1. **Вывод**

В ходе выполнения работы были реализованы три алгоритма сортировки: Heap Sort, Counting Sort и Cocktail Shaker Sort. Проведённые экспериментальные замеры подтвердили теоретические оценки временной и пространственной сложности. Cocktail Shaker Sort не является эффективным для больших массивов, поскольку в худшем случае его временная сложность составляет O(n^2). Это более подходящий алгоритм для небольших массивов или когда стабильность и простота реализации важнее скорости. Heap Sort является эффективным и работает за O(n\*log n) в любом случае. Однако, несмотря на хорошую временную сложность, он не стабилен (не сохраняет порядок одинаковых элементов). Counting Sort — это алгоритм с линейной сложностью, но только если диапазон значений k не слишком велик по сравнению с размером массива. Если k велико, алгоритм может стать неэффективным по памяти и времени. Это идеальный выбор для сортировки числовых данных в ограниченном диапазоне, например, для целых чисел от 0 до k.