ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Алгоритмы сортировки»

Выполнил работу

Бессонов Александр

Академическая группа №J3112

Принято

Преподаватель практики, Дунаев Максим

Санкт-Петербург

2024

# 1. Подсчет памяти для 3-х алгоритмов сортировки:

**1. Comb Sort:**

* Пространственная сложность: O(1).
* Алгоритм сортирует массив "на месте" и не использует дополнительную память. Алгоритм Comb Sort использует одну переменную для хранения текущего значения gap (расстояния между элементами), а также булевую переменную swapped для контроля завершения итераций. Сортировка выполняется "на месте" (in-place), без создания дополнительных массивов или структур данных.

Описание работы:  
Comb Sort работает путем уменьшения промежутков между сравниваемыми элементами (gap) в массиве. На каждой итерации массив модифицируется "внутри себя", что позволяет сохранить постоянное потребление памяти независимо от размера входных данных.

**2. Heap Sort:**

* Пространственная сложность: O(1).
* Алгоритм Heap Sort выполняет сортировку "на месте" за счет модификации исходного массива. Он преобразует массив в бинарную кучу (heap) и постепенно извлекает элементы, заменяя их в массиве.

Описание работы:  
Heap Sort организует элементы массива в структуру кучи (heap) и последовательно извлекает наибольший элемент, перемещая его в конец массива. Поскольку работа выполняется на одном массиве, дополнительных массивов не создается.

3. Bucket Sort:

* Пространственная сложность: O(n + k), где:  
   - n — количество элементов в массиве.  
   - k — количество "корзин" (buckets).
* Для создания корзин требуется O(k) памяти. Каждый элемент из исходного массива добавляется в одну из корзин, что требует дополнительного места для хранения этих элементов (всего O(n)). После сортировки содержимого каждой корзины элементы объединяются в результирующий массив.

Описание работы:  
Bucket Sort распределяет элементы массива по k корзинам на основе их значений. Затем каждая корзина сортируется отдельно, используя другой алгоритм (например, встроенный QuickSort). Алгоритм эффективен при равномерном распределении данных, так как каждую корзину сортировать проще, чем весь массив.

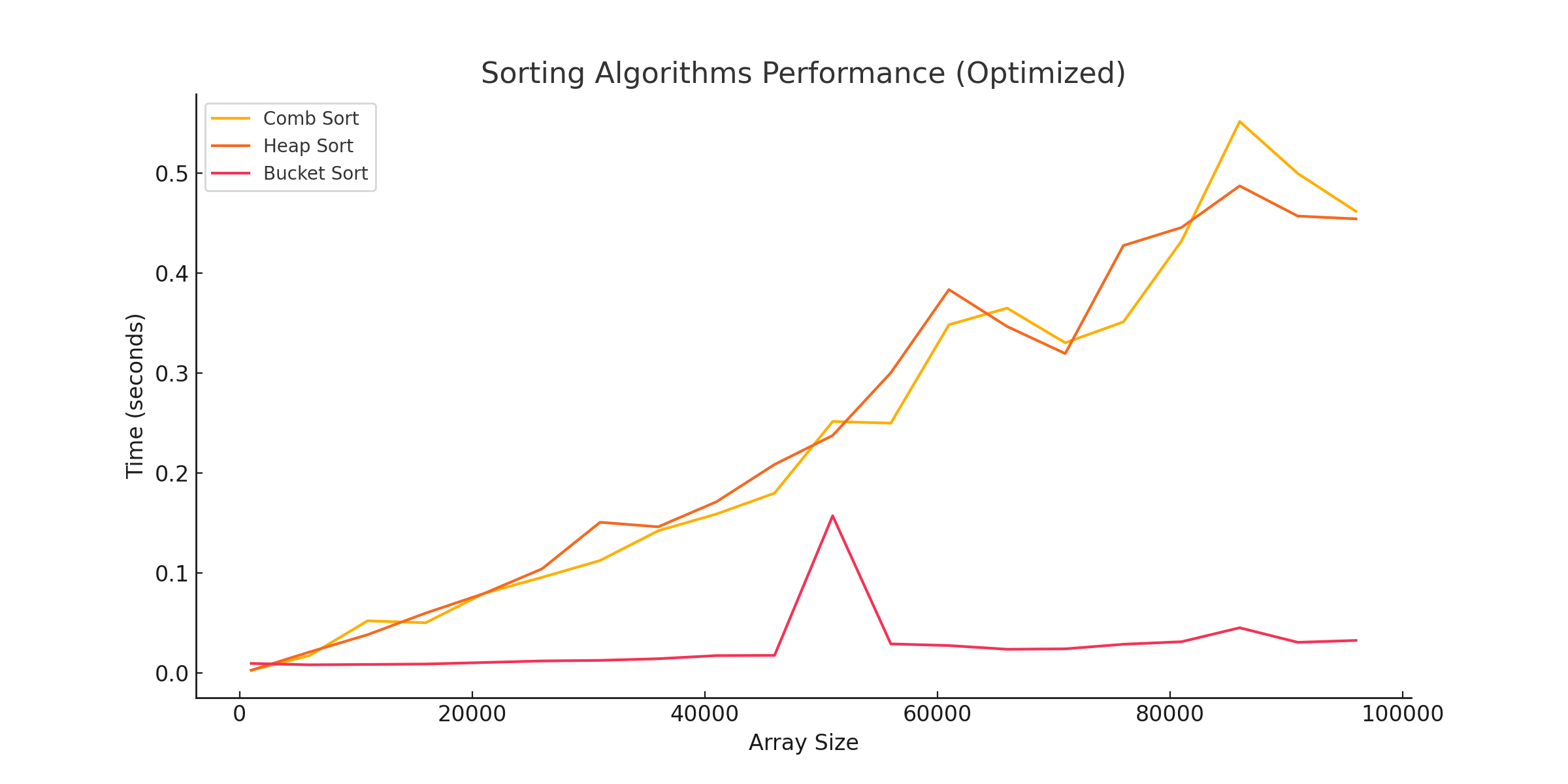
# 2. Асимптотика алгоритмов:

1. Comb Sort:  
 - Лучший случай: O(n log n).  
 - Средний случай: O(n^2).  
 - Худший случай: O(n^2).  
  
2. Heap Sort:  
 - Лучший случай: O(n log n).  
 - Средний случай: O(n log n).  
 - Худший случай: O(n log n).  
  
3. Bucket Sort:  
 - Лучший случай: O(n + k), где k — число корзин.  
 - Средний случай: O(n + k).  
 - Худший случай: O(n^2) (если все элементы попадают в одну корзину).

# 3. Линейный график работы алгоритмов:

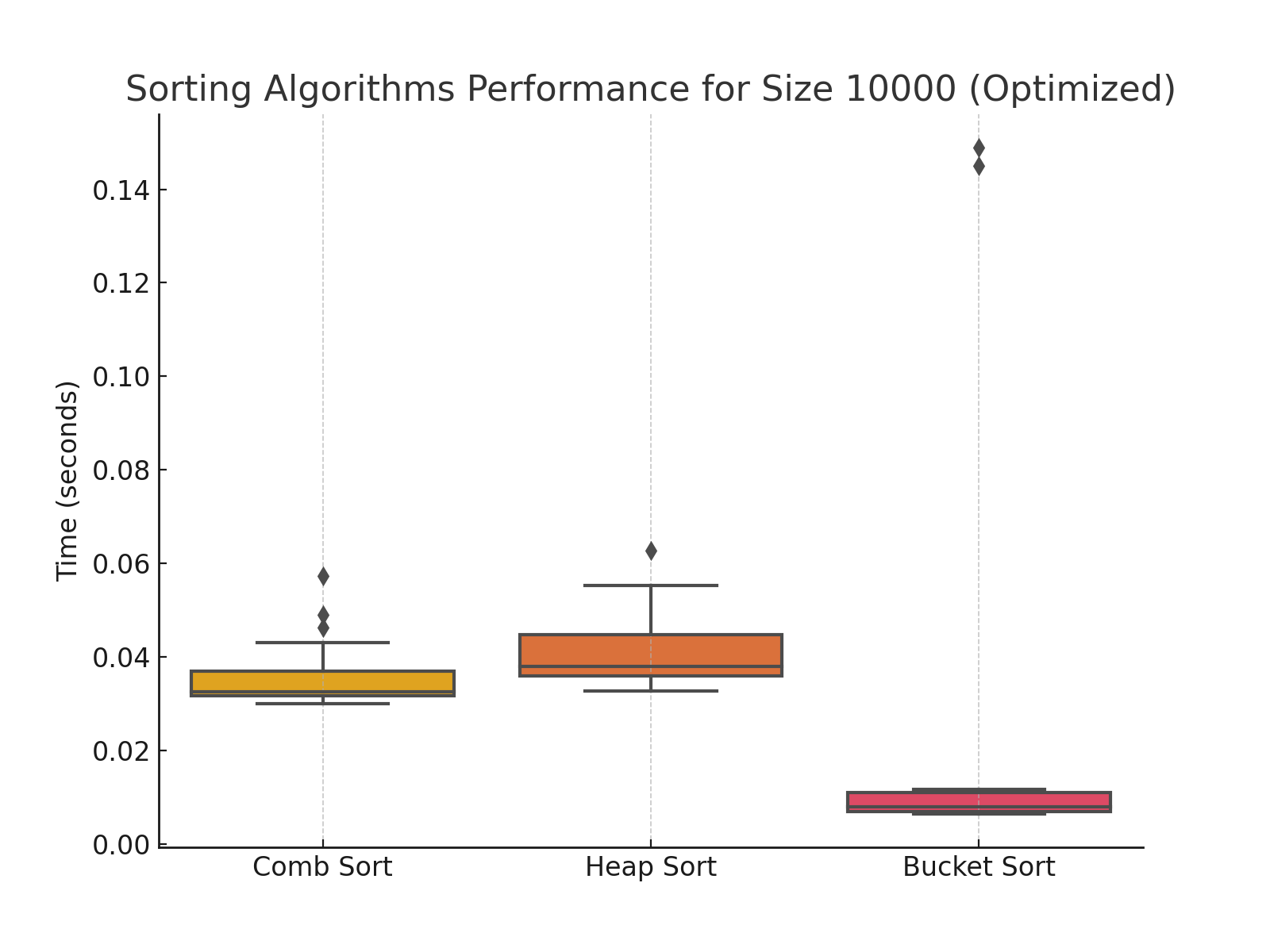
График ниже показывает время работы Comb Sort, Heap Sort, и Bucket Sort на массивах размером от 1000 до 100000 с шагом 5000.

### График №1 – Время выполнения алгоритмов

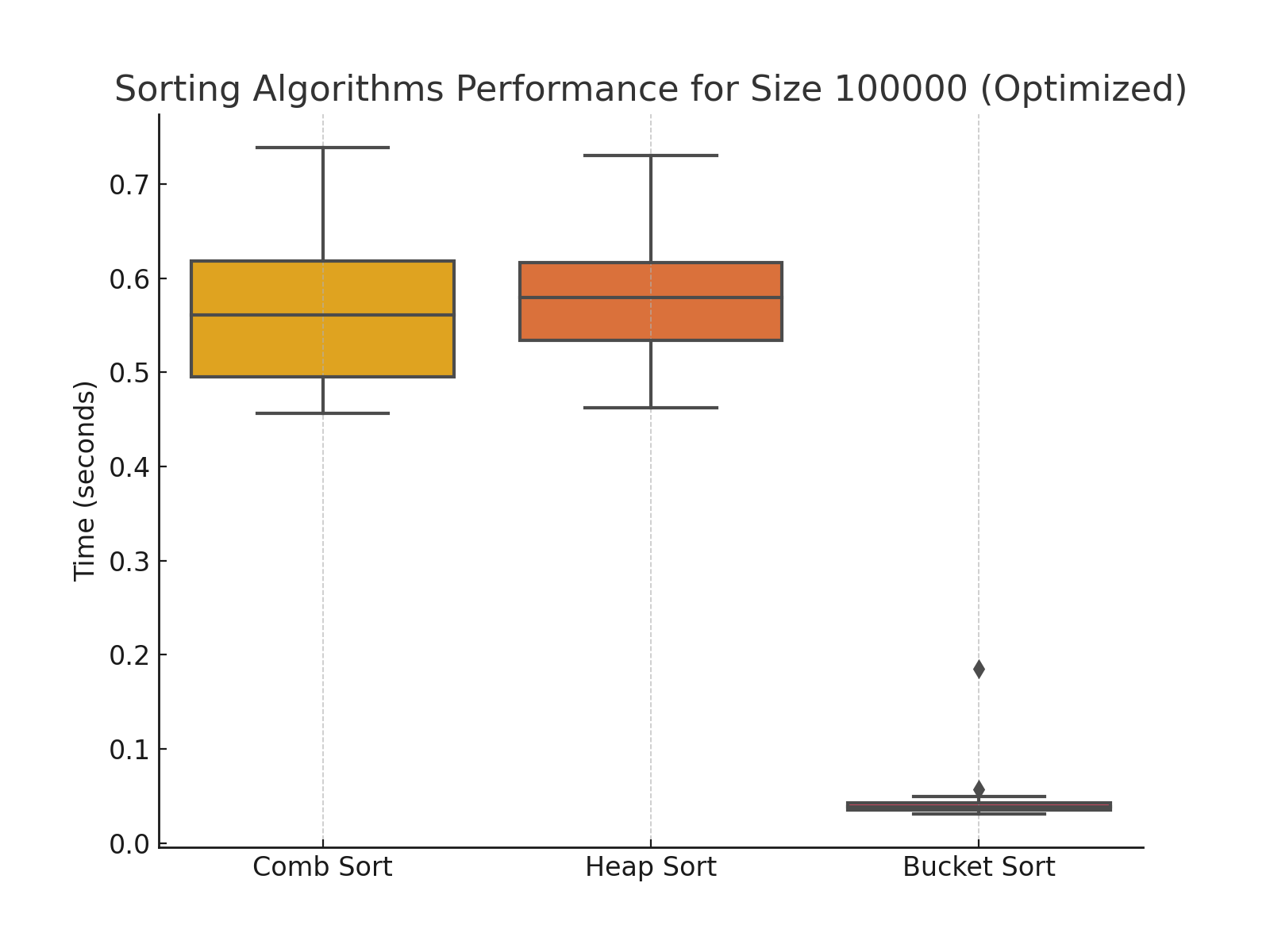


# 4. Box plot графики для времени работы алгоритмов:

### График №2 – Box plot для массивов размером 10^4



### График №3 – Box plot для массивов размером 10^5



# 5. Выводы:

**1. Comb Sort:**

* Подходит для небольших массивов.
* Эффективность падает для больших массивов из-за квадратичной сложности.

**2. Heap Sort:**

* Подходит для любых массивов благодаря стабильной асимптотике.
* Стабильная производительность независимо от распределения данных.

**3. Bucket Sort:**

* Эффективен для массивов с равномерно распределенными данными.
* При наличии выбросов эффективность снижается, так как все элементы могут попасть в одну корзину.