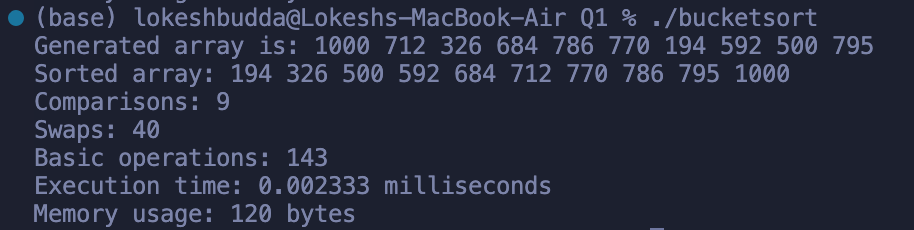
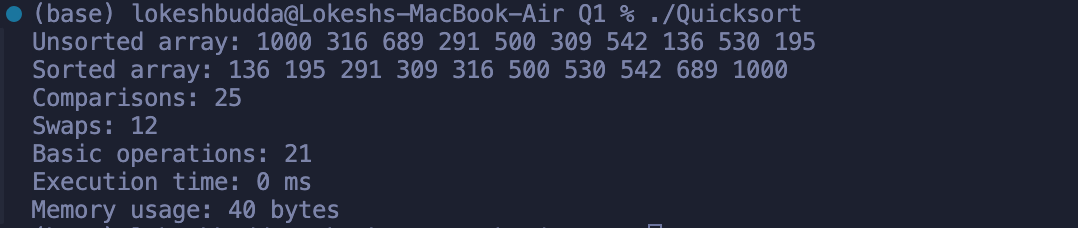
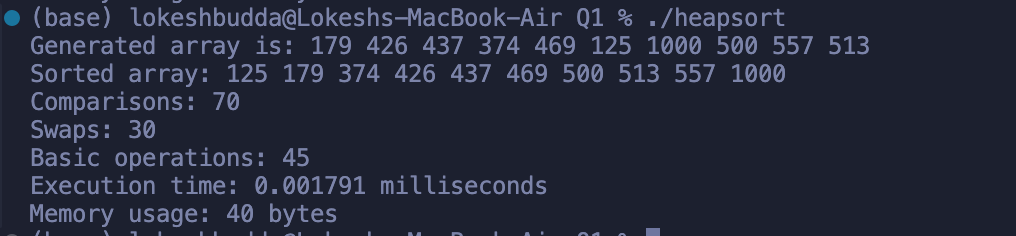
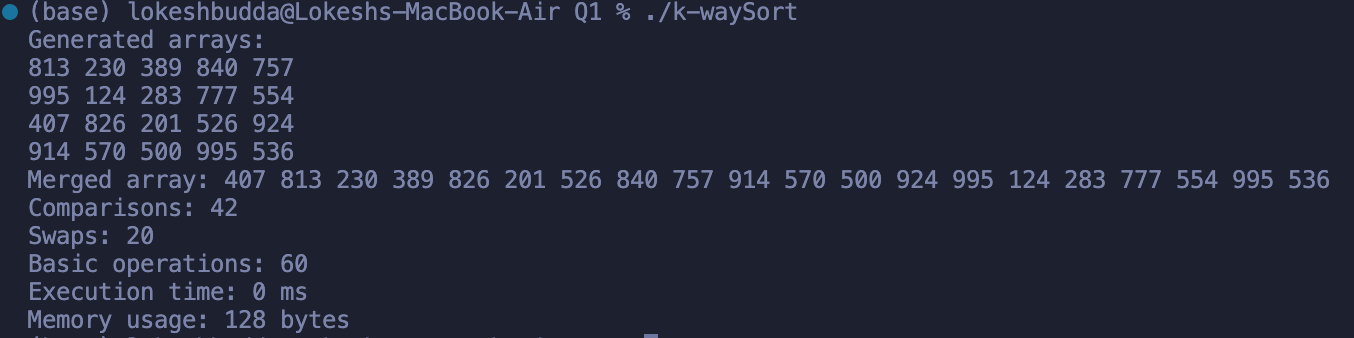
Sorting Algorithms

# Bucket sort Output:

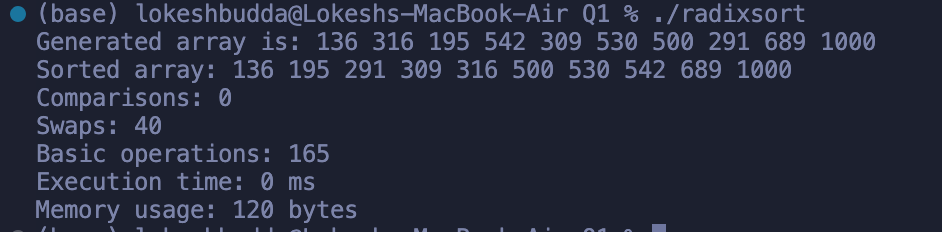
In-place Quick sort Output:

Heap sort Output:

K-way Merge sort Output:



# Radix sort Output:



# In-Place Quick Sort

Quick Sort is a divide-and-conquer algorithm that picks a pivot element and partitions the array into two halves, ensuring elements less than the pivot are on the left and elements greater are on the right. This process is repeated recursively on both halves.  
  
In-place Quick Sort uses a random pivot, minimizing the chance of encountering the worst case.  
  
Best Suited For: Large datasets where average-case performance is crucial and additional memory is limited.

# k-Way Merge Sort (k=4)

k-Way Merge Sort is an extension of the traditional merge sort. Instead of dividing the array into two, it divides it into k parts (in this case, k=4). After sorting each part individually, the algorithm merges them in k-way merges until the entire array is sorted.  
  
Best Suited For: External sorting scenarios where k-way merging can minimize I/O operations, especially with large datasets.

# Heap Sort

Heap Sort uses a binary heap data structure to sort elements. It builds a max-heap from the input data, then repeatedly extracts the maximum element from the heap and adjusts the heap until all elements are sorted.  
  
Best Suited For: Situations requiring an in-place sort with O(n log n) time complexity, particularly when the dataset is large.

# Bucket Sort

Bucket Sort distributes elements into several 'buckets' and sorts each bucket individually, often using another sorting algorithm. After sorting, it concatenates the buckets to get the sorted array.  
  
Best Suited For: Datasets with a uniform distribution of values, particularly when the input range is known and limited.

# Radix Sort

Radix Sort processes digits of numbers one at a time, starting from the least significant digit to the most significant. It uses a stable sorting algorithm (like counting sort) for sorting digits.  
  
Best Suited For: Large datasets of numbers or strings where the keys have a uniform length.

# Time Complexities

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Best Case | Average Case | Worst Case |
| In-Place Quick Sort | O(n log n) | O(n log n) | O(n^2) |
| k-Way Merge Sort (k=4) | O(n log n) | O(n log n) | O(n log n) |
| Heap Sort | O(n log n) | O(n log n) | O(n log n) |
| Bucket Sort | O(n+k) | O(n+k) | O(n^2) |
| Radix Sort | O(nk) | O(nk) | O(nk) |

# Sources:

Bucket sort:

* <https://www.geeksforgeeks.org/bucket-sort-2/> : Used to gain better understanding of the concept

Heap sort:

* <https://www.geeksforgeeks.org/heap-sort/> : Used to Visually understand the concept.
* <https://www.programiz.com/dsa/heap-sort> : Used to gain a better grasp on the algorithm

Quick sort:

* <https://www.geeksforgeeks.org/quick-sort-algorithm/> : Used to Visually understand the concept.
* <https://www.geeksforgeeks.org/in-place-algorithm/> : Understanding in-place algorithms

Radix sort:

* <https://www.geeksforgeeks.org/radix-sort/> : Used to gain a better grasp on the algorithm
* <https://www.w3schools.com/dsa/dsa_algo_radixsort.php> Used to Visually understand the concept.

Random number generator and Execution time handling:

* <https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/> : Used for random function
* <https://www.geeksforgeeks.org/file-handling-c-classes/> : File handling for Input test cases.
* <https://www.geeksforgeeks.org/measure-execution-time-function-cpp/> : Measuring the execution time