*NAME: OLUWAGBEMINIYI REST JOHNSON*

**PROJECT 1 REPORT**

ITCS 8114 – ALGORITHMS AND DATA STRUCTURES

*DR DEWAN AHMED*

*UNIVERSITY OF NORTH CAROLINA - CHARLOTTE*

1. **Insertion Sort**

* Data Structure Used: Insertion Sort operates on an array data structure. It iteratively takes an element from the array and inserts it into the correct position in an already sorted part of the array.
* Complexity Analysis:
  + Best Case: O(n) when the array is already sorted.
  + Average Case: O(n2)
  + Worst Case: O(n2 )when the array is sorted in reverse order.
* General Results: Insertion Sort is efficient for small datasets and nearly sorted arrays. However, its efficiency decreases significantly as the size of the array increases.

1. **Merge Sort**

* Data Structure Used: Merge Sort utilizes an array for its operations. It divides the array into halves, recursively sorts the halves, and then merges the sorted halves back together.
* Complexity Analysis:
  + Best, Average, and Worst Case: O(nlogn).
* General Results: Merge Sort is a stable sort with consistent O(nlogn) performance but requires additional memory for temporary storage, making it less memory efficient for large datasets.

1. **Heap Sort**

* Data Structure Used: Heap Sort is implemented using a heap data structure, often represented as a binary tree within an array. The heap is built incrementally by inserting one item at a time.
* Complexity Analysis:
  + Best, Average, and Worst Case: O(nlogn).
* General Results: Heap Sort is efficient for large datasets due to its O(nlogn) performance and in-place sorting capability. However, it is not a stable sort and may have slightly slower constant factors compared to other O(nlogn) sorting algorithms due to the overhead of maintaining the heap structure.

1. **In-Place Quick Sort**

* Data Structure Used: Quick Sort operates on an array, partitioning the array around a pivot element and recursively sorting the partitions.
* Complexity Analysis:
  + Best and Average Case: O(nlogn).
  + Worst Case: O(n2) particularly when the smallest or largest element is always chosen as the pivot.
* General Results: Quick Sort is highly efficient for large datasets with O(nlogn) average complexity. Choosing a good pivot is crucial to avoid the worst-case scenario. It’s an in-place, but not a stable sort.

1. **Modified Quicksort Using Median of Three**

* Data Structure Used: This version of Quick Sort also operates on an array but improves pivot selection by using the median of three strategy (picking the median of the first, middle, and last elements).
* Complexity Analysis:
  + Best General Results and Average Case: O(nlogn).
  + Worst Case: O(n2), but much less likely than the basic Quick Sort due to improved pivot selection.
* : The median of three modifications to Quick Sort generally improves performance by reducing the likelihood of encountering the worst-case scenario. It maintains the in-place sorting advantage and is particularly effective on diverse datasets.

**Conclusion**

Sorting algorithms vary widely in their efficiency, memory usage, and suitability for different types of data. While Insertion Sort and Heap Sort offer benefits for specific scenarios, like small datasets or memory-efficient sorting, Merge Sort and Quick Sort (including its median of three variant) provide more consistently efficient O(nlogn) performance for larger datasets. The choice of sorting algorithm depends on the specific requirements, including dataset size, memory constraints, and whether stability is a concern.

**Normal Runtime Plot**

**A graph with a line

Description automatically generated**

**Special Case 1: Array Already Sorted**

**A graph with a line

Description automatically generated**

Excluding the median of three quick sort, all other algorithms appear to be relatively fast.

**Special Case 2: Array Is in Reverse Order**

**A graph with a line

Description automatically generated**

Performance is similar to that of the normal plot. However, there is a larger disparity between the runtime of insert sort and median of three quick sort.