Project 1: Comparison-Based Sorting Algorithms

By: Bashar Shabani

Algorithms Implemented:

- Insertion Sort
- Merge Sort
- Heap Sort: Vector-based, and insert one item at a time
- In-place Quicksort: Any random item or the first, or the last item of your input can be the pivot.
- Modified Quicksort: Use median-of-three as pivot. For a small sub-problem of size ≤ 15, you must use insertion sort.

Time Complexity:

- Insertion Sort: $O(n^2)$
- Merge Sort: $O(n \log n)$
- Heap Sort: $O(n \log n)$
- In-place Quicksort: $O(n \log n)$
- Modified Quicksort: $O(n \log n)$

Observations on Random Input:

- Insertion Sort performed well on smaller inputs but became too slow beyond 10,000 elements, so it was excluded from larger sizes.
- Merge Sort remained consistently fast and scaled efficiently across all input sizes.
- Heap Sort slowed down significantly as input size increased, due to inserting elements one at a time into the heap.
- Both quicksort implementations performed best overall. The modified version was slightly faster, likely due to better pivot selection and using insertion sort on smaller subarrays.

Special Cases Observations:

Algorithm	**Time (seconds)**
Insertion Sort	0.0016
Merge Sort	0.0210
Heap Sort	5.5111

- Insertion Sort was the fastest, which makes sense for already sorted data.
- Merge Sort stayed efficient with no major change.
- Heap Sort was still slow because of how it builds the heap step by step.

Conclusion:

- Modified QuickSort gave the best overall performance across all input sizes.
- Merge Sort was consistent and scaled well.
- Heap Sort didn't perform well here because of the one-at-a-time insert approach.
- Insertion Sort only worked well for small inputs or sorted data.

Comparison Charts: