Data Structures Lab Submission

Name: Chris Mugabo
Lab Topic: Sorting Algorithms Analysis

---

1. Relationship Between Array Size and Sorting Algorithm

- **Small Arrays**: Use **Insertion Sort**. It's simple and efficient for small datasets due to low overhead.
- **Large Arrays**: Use **Quick Sort** or **Merge Sort**. They have $\mathrm{O(nlogn)}$ time complexity, making them scalable for large datasets.
- **Partially Sorted Arrays**: Use **Insertion Sort**. It performs well ($\mathrm{O(n)}$) when the array is already partially ordered.

---

2. When to Avoid Certain Algorithms

- **Avoid Insertion Sort** for large datasets ($\mathrm{O(n2)}$ time complexity.
- **Avoid Quick Sort** for small or nearly sorted arrays (poor pivot selection can lead to $\mathrm{O(n2)}$
- **Avoid Merge Sort** in memory-constrained environments (requires $O(n)$ additional space).

Thought Process

Choose a sorting algorithm based on:

1. **Dataset Size**: Small → Insertion Sort; Large → Quick Sort/Merge Sort.
2. **Data Nature**: Partially sorted → Insertion Sort; Random → Quick Sort.
3. **Memory Constraints**: Limited memory → Avoid Merge Sort.
4. **Stability**: If stability is needed, use Merge Sort or Insertion Sort.

Summary Table

Algorithm	Best Use Case	Avoid When
Insertion Sort	Small or nearly sorted arrays	Large datasets
Quick Sort	Large, random datasets	Small or nearly sorted arrays
Merge Sort	Large datasets, stable sorting needed	Memory-constrained environments