Sorting Algorithms  
(Insertion,Bubble,Selection)

1. **Write the application of sorting.**

**Ans:**

**Quick Access:** Find k-th smallest/largest elements in O(1) time after sorting.

**Searching:** Essential for binary/ternary search algorithms.

**Data Management:** Simplifies searching, retrieval, and analysis.

**Database Optimization:** Enhances query performance with sorted data.

**Machine Learning:** Prepares data for model training.

**Data Analysis:** Identifies patterns, trends, and outliers.

**Operating Systems:** Used in task scheduling, memory management, and file organization.

1. **Write the advantage of sorting.**

**Ans:**

**Efficiency:** Faster searching, retrieval, and analysis.

**Improved Performance:** Enhances algorithm operations.

**Simplified Analysis:** Easier pattern and trend identification.

**Reduced Memory:** Eliminates duplicates, saving memory.

**Better Visualization:** Sorted data improves chart and graph clarity.

1. **Compare the time complexities of insertion sort,bubble sort & selection sort.**

**Ans:**

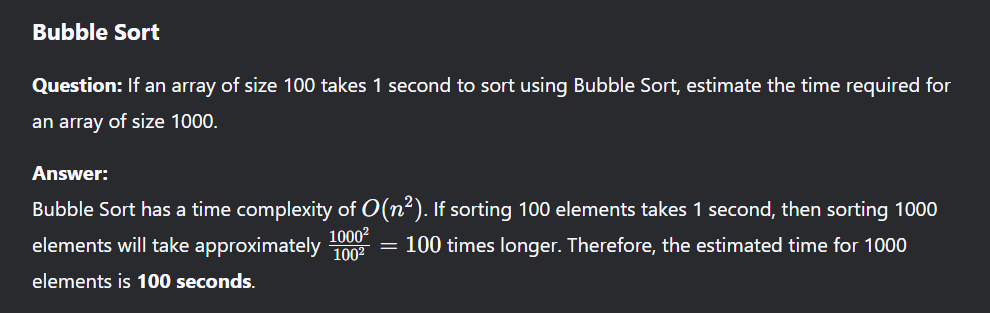
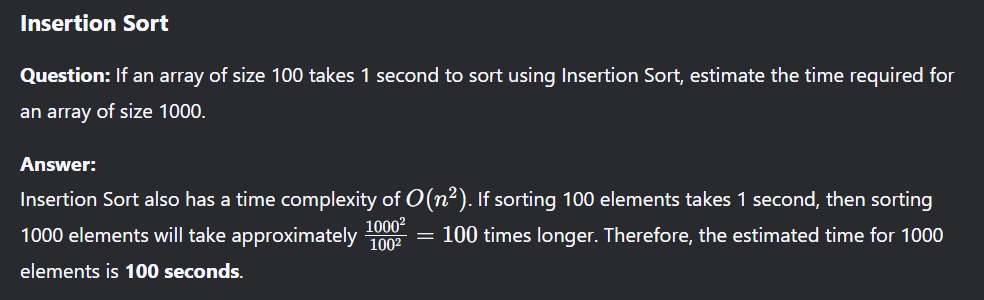
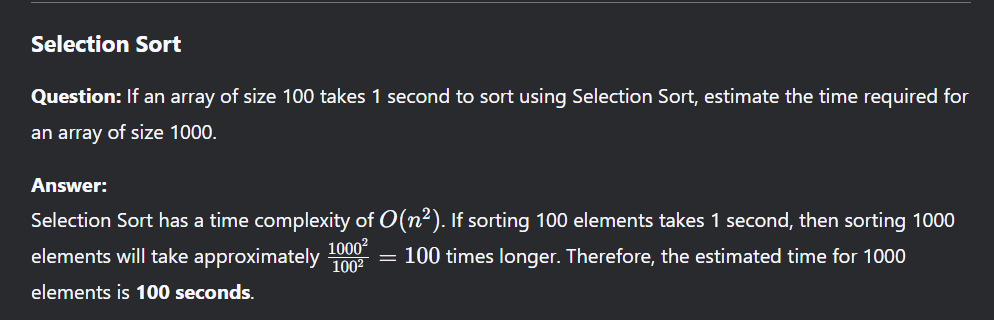
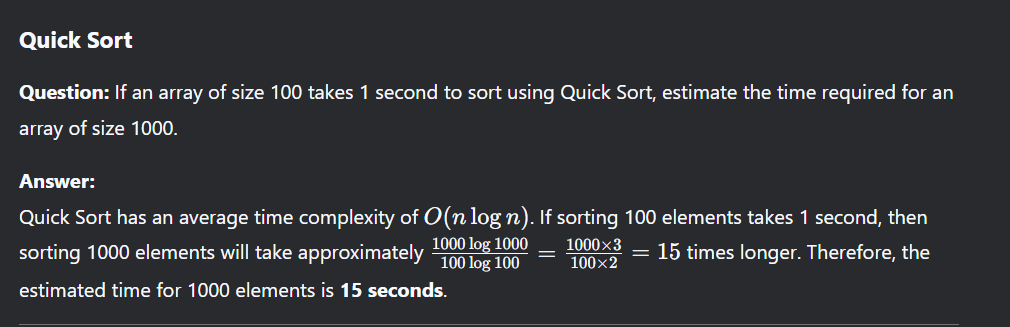
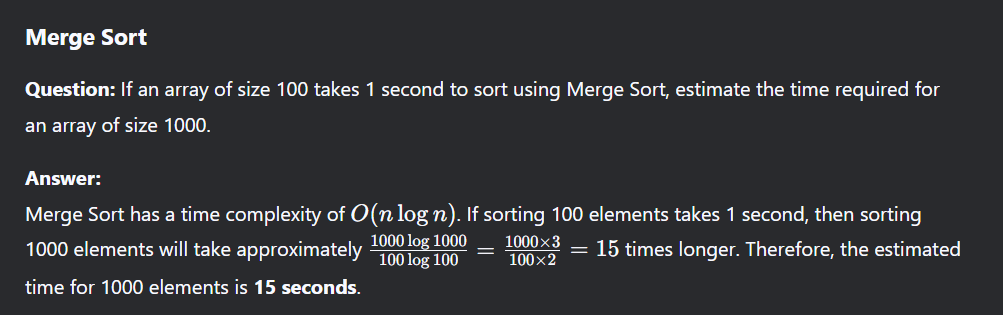
|  |  |  |  |
| --- | --- | --- | --- |
| Sorting Algorithm | Best Case | Average Case | Worst Case |
| Insertion Sort | **O(n)** | **O(n2)** | **O(n2)** |
| Bubble Sort | **O(n)** | **O(n2)** | **O(n2)** |
| Selection Sort | **O(n2)** | **O(n2)** | **O(n2)** |

* Insertion Sort & Bubble Sort are efficient (**O(n)**) for already sorted data.
* Sometimes insertion Sort is more efficient than Bubble Sort in large dataset(in Best Cases)
* All three have **O(n2)** in average & worst cases,making them slower for large datasets.

1. **Given an unsorted array of size n, how many swaps are performed in Bubble Sort,Insertion Sort,Selection Sort,Merge Sort,Quick Sort in the worst case?**

**Ans:**

|  |  |  |
| --- | --- | --- |
| **Sorting Algorithm** | **Worst Case Swaps** | **Explanation** |
| **Bubble Sort** |  | Reverse order causes maximum swaps. |
| **Insertion Sort** |  | Reverse order causes maximum swaps. |
| **Selection Sort** | **n - 1** | Always swaps the smallest element with the first unsorted element. |
| **Quick Sort** |  | Unbalanced partitions (pivot is smallest/largest). |
| **Merge Sort** | **0 (no swaps)** | Uses merge operation with **O(n)** space and **nlogn** comparisons. |

1. 
2. 
3. 
4. 
5. 
6. **Which is more efficient?(Bubble Sort,Insertion Sort,Selection Sort)**

**Ans:**

* **Bubble Sort** → O(n²) (Worst & Average), O(n) (Best case, already sorted)
* **Selection Sort** → O(n²) (Worst, Average & Best)
* **Insertion Sort** → O(n²) (Worst & Average), O(n) (Best case, nearly sorted)

Insertion Sort is generally more efficient than Bubble Sort and Selection Sort, especially when the input is nearly sorted. It has a best-case time complexity of **O(n)**, whereas Bubble Sort and Selection Sort always take **O(n²)** time.

1. **Given an array [5, 3, 8, 4, 2], show the step-by-step execution of (Insertion Sort,Bubble Sort,Selection Sort)**

**Divide & Conquer**

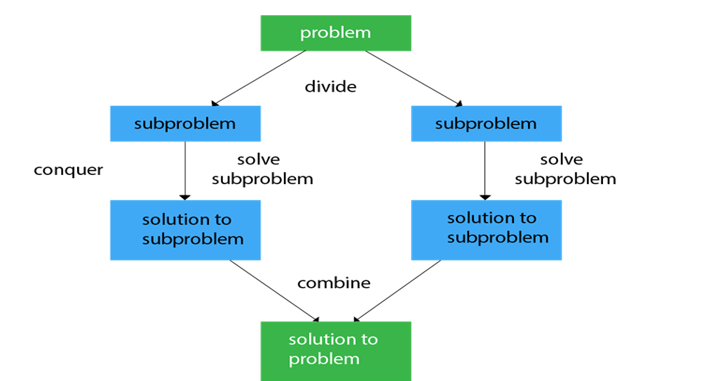
**(Merge Sort & Quick Sort)**

1. **Explain the three steps of the Divide & Connquer technique with an example**

**Ans:**

* **Divide:** Split the problem into two or more smaller subproblems.
* **Conquer:** Solve each of the subproblems.
* **Combine:** Combine the solutions to the subproblems to solve the original problem.

**Example:** Quick Sort,Merge Sort



1. **Key characteristics of Divide & Conquer.**

**Ans:**

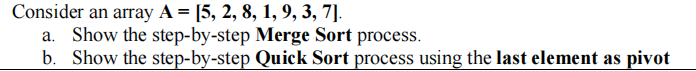
* **Recursion:** Divide and conquer typically uses recursion to solve sub-problems.
* **Divide and Combine Operations:** The problem is split into sub-problems, and the sub-problem results are merged to solve the original problem.
* **Efficiency:** Many divide and conquer algorithms, especially when used in sorting or searching, can achieve optimal time complexity.

1. **How does parallelism improve Divide and Conquer algorithms?**

**Ans:**

* **1.Faster Execution –** Solves subproblems simultaneously, reducing time.
* **2.Better Resource Utilization –** Uses multiple processors efficiently.
* **3. Improved Scalability –** Handles large problems effectively.

**Example:** In Merge Sort, both halves are sorted at the same time, speeding up the process.

1. 
2. **Compare Merge Sort & Quick Sort in terms of their time complexity & space complexity.**