



# ALGORITHMS AND COMPLEXITY ANALYSIS

Sorting Algorithms

03



School of Computing

# BULLET POINT SUMMARY

## SORTING

### ALGORITHMS

Sorting algorithms like **Bubble Sort** and **Selection Sort** play a crucial role in arranging data in a specific order, either ascending or descending, to enable efficient data processing and search operations.

### BUBBLE SORT

**Bubble Sort** is a simple comparison-based algorithm that repeatedly compares adjacent elements, swapping them if they are in the wrong order, with a worst-case time complexity of  $O(n^2)$  and a best-case time complexity of  $O(n)$  if the list is already sorted.

### SELECTION SORT

**Selection Sort** repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first unsorted element, also with a time complexity of  $O(n^2)$  in both the worst and best cases.

Both **Bubble Sort** and **Selection Sort** are **in-place algorithms**, meaning they do not require additional memory, but they are inefficient for large datasets.

**Bubble Sort** is ideal for small datasets or scenarios where simplicity is key, but it is not suitable for large datasets due to its  $O(n^2)$  time complexity.

**Selection Sort** is also best suited for small datasets, and it is more efficient than **Bubble Sort** in terms of swaps, as it performs fewer swaps during sorting.

**Linear Search** and **Binary Search** are foundational search algorithms used to find specific elements in a dataset, with **Linear Search** checking each element sequentially and **Binary Search** dividing the search space in half for sorted data.

## HASH TABLES

**Hash Tables** store data in key-value pairs, using a hash function to provide  $O(1)$  average time complexity for searching, insertion, and deletion, and are ideal for large datasets where constant-time operations are required.

### HASH FUNCTIONS

**Hash Functions** play a critical role in hash tables, ensuring that keys are mapped uniformly across the table to avoid clustering and improve performance.

### COLLISION HANDLING

**Collision handling** techniques like **chaining** and **open addressing** are used in hash tables to manage cases where multiple keys hash to the same index.

**Hash Tables** are widely used in databases, compilers, and networking applications, providing fast access to data with minimal performance degradation.

# REFLECTION QUESTIONS AND CALL TO ACTION

## REFLECTION QUESTIONS:

How do **Bubble Sort** and **Selection Sort** compare in terms of memory usage and algorithm complexity for small datasets?

What is the importance of **collision handling** in **hash tables**, and which technique would be more suitable for a high-load system?

How do **Bubble Sort** and **Selection Sort** fail to scale for large datasets, and what would you recommend as alternatives?

When implementing a **hash table**, what characteristics should the **hash function** have to ensure optimal performance?

Given the time complexities of **Bubble Sort** and **Selection Sort**, in what practical situations might they still be preferable over more complex sorting algorithms?

## CALL TO ACTION:

Implement both **Bubble Sort** and **Selection Sort** algorithms and test them with datasets of various sizes to observe their performance. Also, try creating a **hash table** with an appropriate **hash function** and test it with different datasets.

## QUESTIONS:

How would you optimise **Bubble Sort** for slightly better performance, especially for nearly sorted lists?

What scenario would lead you to use **Selection Sort** despite its inefficiency in the worst case?

In the context of sorting algorithms, what are the advantages of **in-place sorting** as seen in both **Bubble Sort** and **Selection Sort**?

What are the trade-offs between the performance of **Bubble Sort** and **Selection Sort** when handling large datasets?

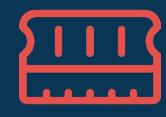
How does **Binary Search** compare to **Linear Search** when used on unsorted datasets?

# SKILLS AND COMPETENCIES



## UNDERSTANDING OF SORTING ALGORITHMS

**Understanding of sorting algorithms** (Bubble Sort, Selection Sort, etc.).



## EFFICIENT MEMORY USAGE

**Efficient memory usage** through in-place sorting algorithms.

**Algorithm optimisation** for better performance with large datasets.

**Comparative analysis** of search algorithms (Linear Search, Binary Search) on sorted and unsorted data.



## TIME COMPLEXITY ANALYSIS

**Time complexity analysis** for **Bubble Sort** and **Selection Sort**.



## IMPLEMENTATION

**Implementation** of collision handling techniques in **hash tables**.

**Designing and testing efficient hash functions** for **hash table operations**.

**Problem-solving skills** in selecting the appropriate algorithm based on dataset characteristics.