AKTU · 2ND YEAR · SEMESTER 3 · BCS301

Data Structure

Unit 3 — Searching and Sorting

- Searching: Concept of Searching, Sequential search, Index Sequential Search, Binary Search
- Concept of Hashing & Collision resolution Techniques used in Hashing
- Sorting: Insertion Sort, Selection Sort, Bubble Sort, Quick Sort, Merge Sort, Heap Sort and Radix Sort



DECODED BY

Chirag Jain

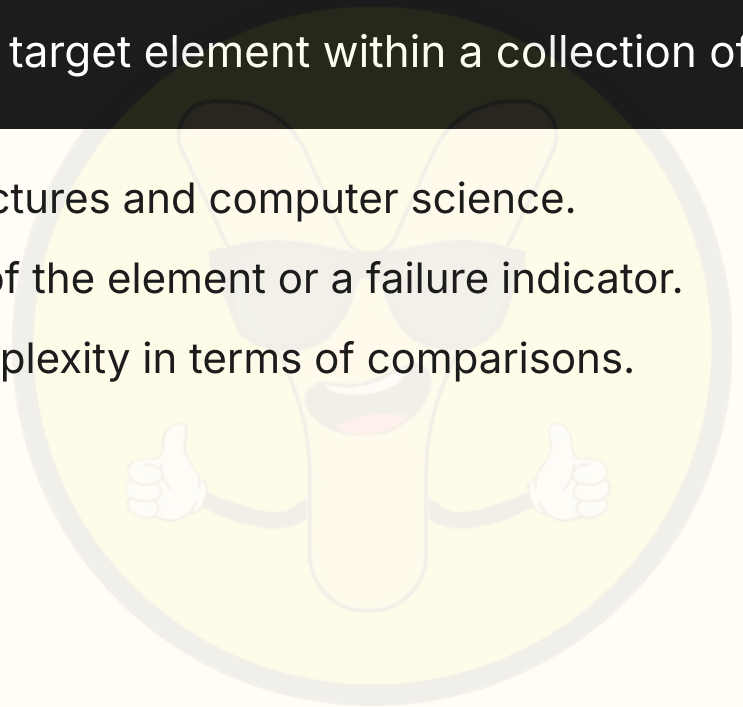
Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Concept of Searching

SEARCHING

The process of locating a specific target element within a collection of data items.

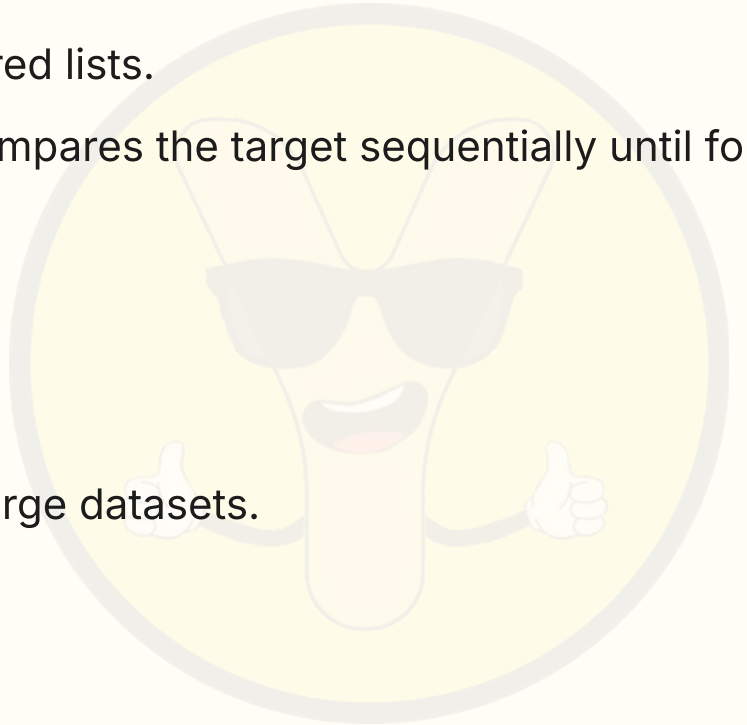
- Fundamental operation in data structures and computer science.
- Returns either the position (index) of the element or a failure indicator.
- Efficiency is measured by time complexity in terms of comparisons.



Sequential Search

Also known as linear search, it is the simplest searching algorithm.

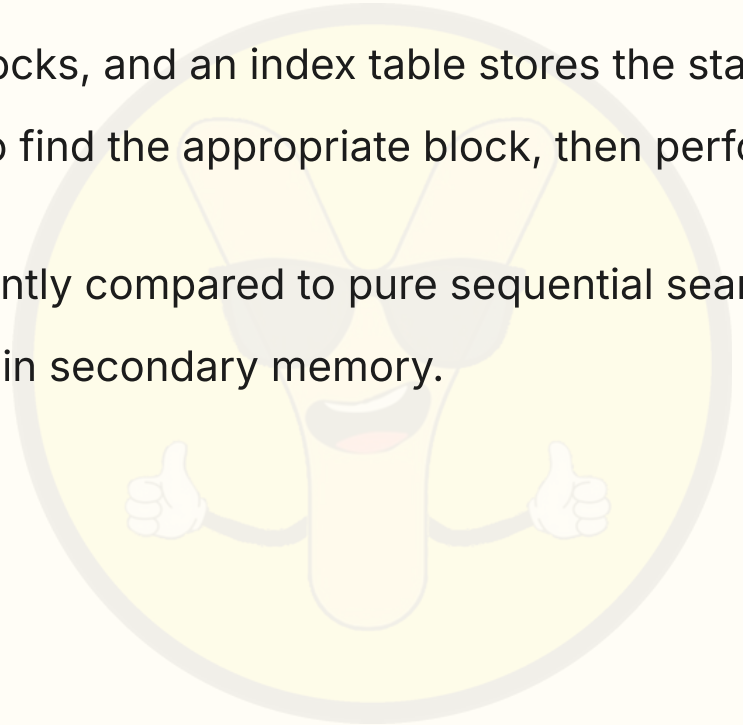
- Works on both ordered and unordered lists.
- Starts from the first element and compares the target sequentially until found or list ends.
- Complexity
  - Best Case: $O(1)$
  - Worst Case: $O(n)$
  - Average Case: $O(n)$
- Disadvantage: Extremely slow for large datasets.



Indexed Sequential Search

An improvement over sequential search using an index table.

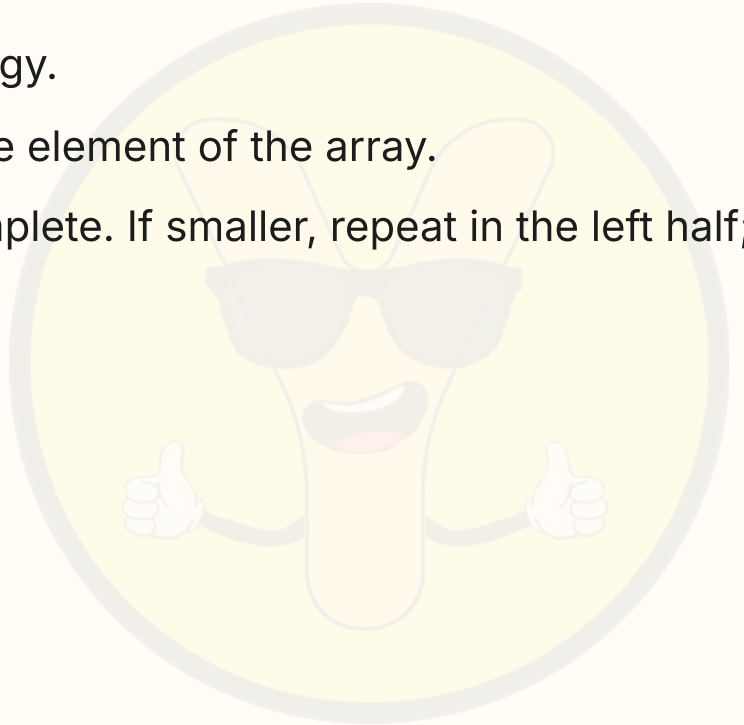
- The main data file is divided into blocks, and an index table stores the starting key of each block.
- Search first scans the index table to find the appropriate block, then performs a sequential search within that block.
- Reduces the search space significantly compared to pure sequential search.
- Ideal for medium-sized data stored in secondary memory.



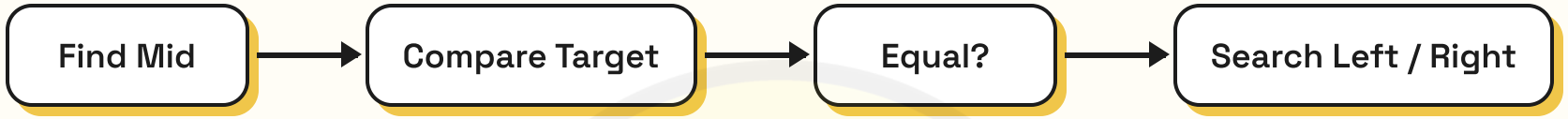
Binary Search

A fast search algorithm that requires the input list to be sorted.

- Uses the Divide and Conquer strategy.
- Compares the target with the middle element of the array.
- If the target matches, search is complete. If smaller, repeat in the left half; if larger, repeat in the right half.
- Complexity
  - Best Case: $O(1)$
  - Worst Case: $O(\log n)$
  - Space Complexity: $O(1)$



Binary Search Flow



Step-by-step division of search space in Binary Search.

Concept of Hashing

HASHING

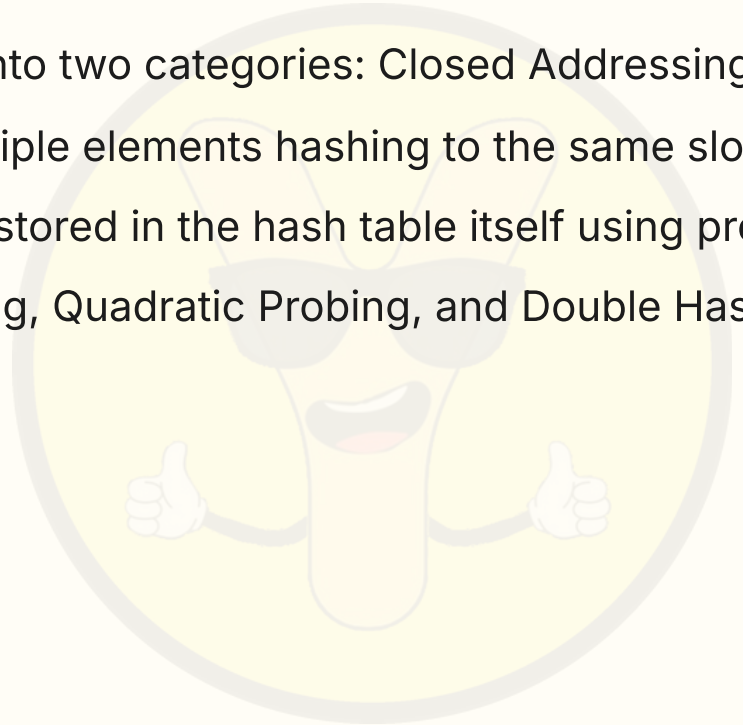
A technique that maps keys to specific values in a hash table using a mathematical formula called a hash function.

- Provides near constant time $O(1)$ performance for search, insert, and delete operations.
- Hash Function: $h(k) = k \pmod{m}$ where k is the key and m is the table size.
- A good hash function distributes keys uniformly to minimize collisions.

Collision Resolution Techniques

Collisions occur when two different keys map to the same hash table index.

- Techniques are broadly classified into two categories: Closed Addressing and Open Addressing.
- Closed Addressing (Chaining): Multiple elements hashing to the same slot are stored in a linked list.
- Open Addressing: All elements are stored in the hash table itself using probing methods.
- Probing types include Linear Probing, Quadratic Probing, and Double Hashing.



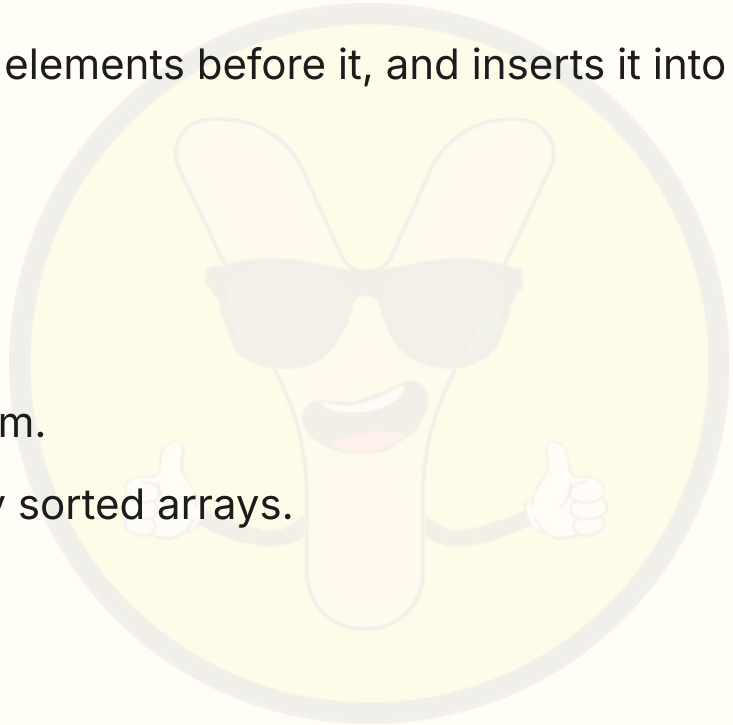
Comparison of Collision Resolution

Technique	Mechanism	Advantage	Disadvantage
Chaining	Linked list at each slot	No overflow issues	Extra memory for pointers
Linear Probing	Check $(h(k) + i) \pmod m$	Easy to implement	Primary clustering
Quadratic Probing	Check $(h(k) + i^2) \pmod m$	Reduces primary clustering	Secondary clustering
Double Hashing	Use second hash function	Minimal clustering	Computationally expensive

Insertion Sort

Builds the final sorted array one item at a time.

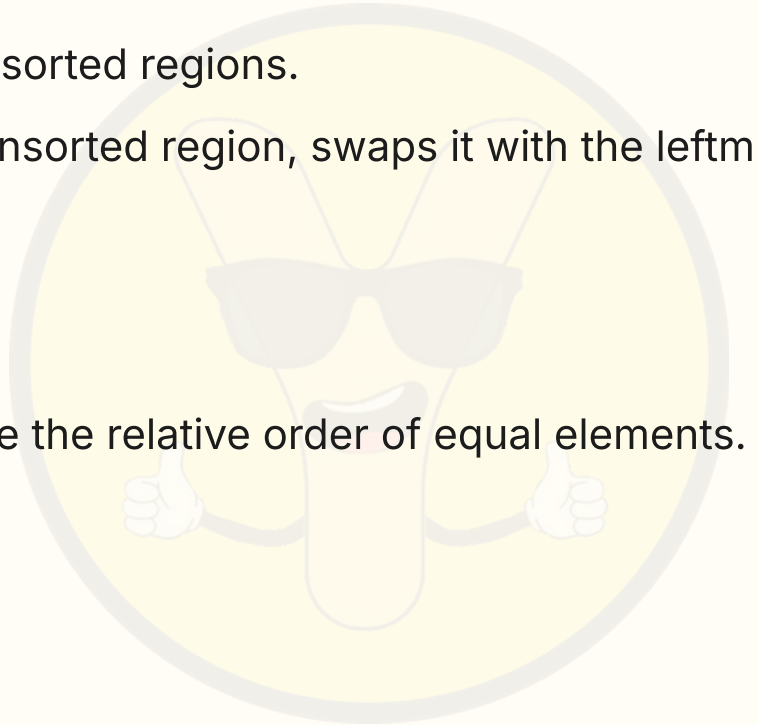
- Takes an element, compares it with elements before it, and inserts it into its correct sorted position.
- Complexity
  - Best Case: $O(n)$ (already sorted)
  - Worst Case: $O(n^2)$
  - Space Complexity: $O(1)$
- Adaptive and stable sorting algorithm.
- Efficient for small datasets or nearly sorted arrays.



Selection Sort

Repeatedly selects the minimum element from the unsorted region.

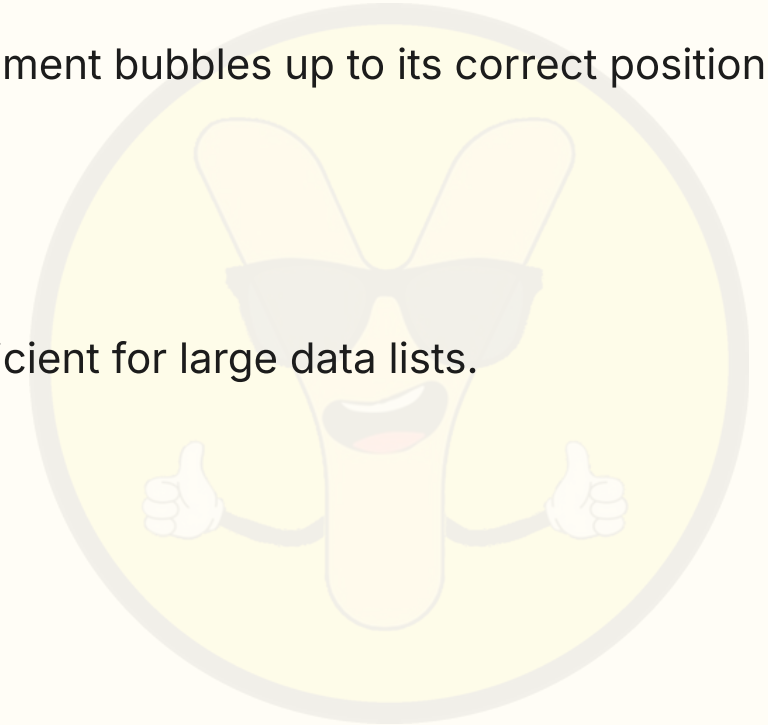
- Divides the array into sorted and unsorted regions.
- Finds the minimum element in the unsorted region, swaps it with the leftmost unsorted element, and expands the sorted region.
- Complexity
  - Best, Worst, Average Case: $O(n^2)$
- Unstable sort because it can change the relative order of equal elements.



Bubble Sort

Repeatedly swaps adjacent elements if they are in the wrong order.

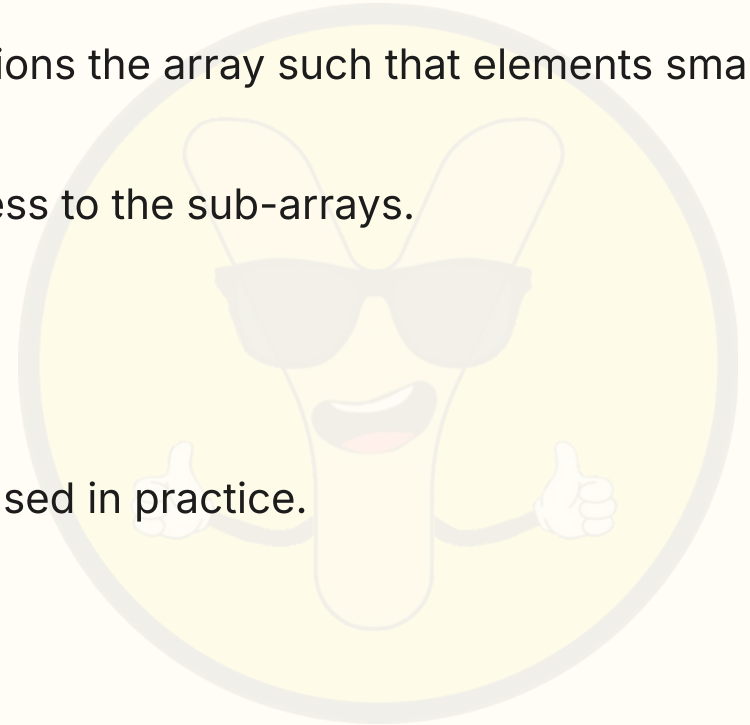
- After each full pass, the largest element bubbles up to its correct position at the end of the array.
- Complexity
  - Best Case: $O(n)$ (with optimized flag)
  - Worst Case: $O(n^2)$
- Simple to implement but very inefficient for large data lists.
- Stable sorting algorithm.



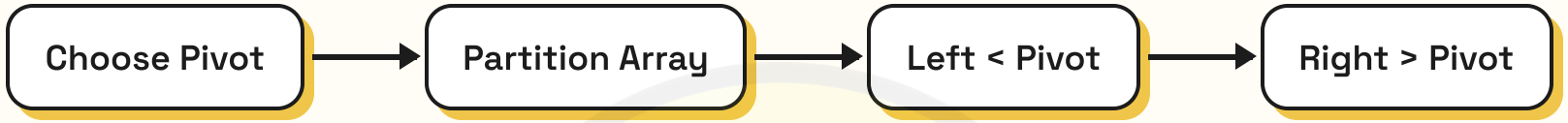
Quick Sort

A highly efficient, divide and conquer sorting algorithm.

- Chooses a pivot element and partitions the array such that elements smaller than the pivot are on the left, and larger are on the right.
- Recursively applies the same process to the sub-arrays.
- Complexity
  - Best/Average Case: $O(n \log n)$
  - Worst Case: $O(n^2)$ (poor pivot choice)
- In-place sorting algorithm, widely used in practice.



Quick Sort Partitioning

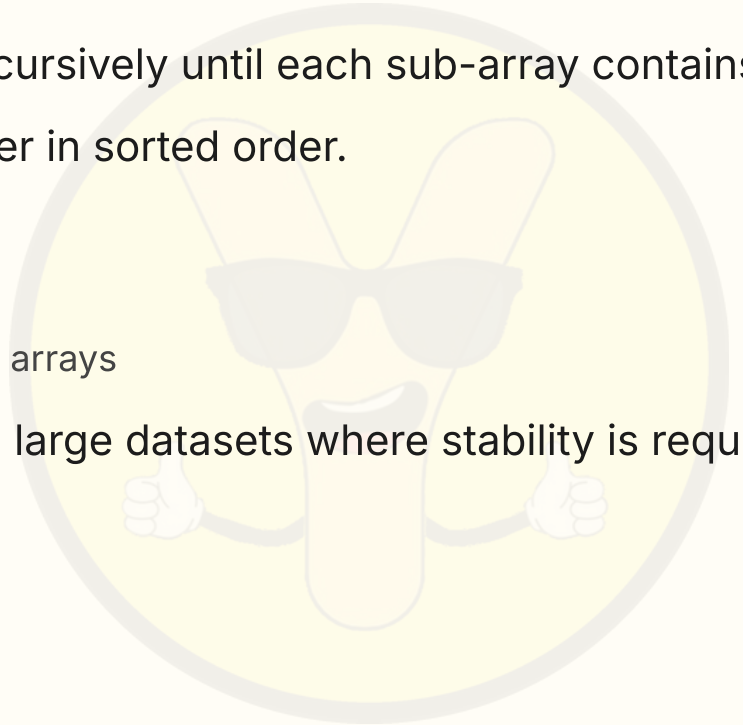


Core partitioning step in Quick Sort.

Merge Sort

A stable divide and conquer algorithm that splits and merges arrays.

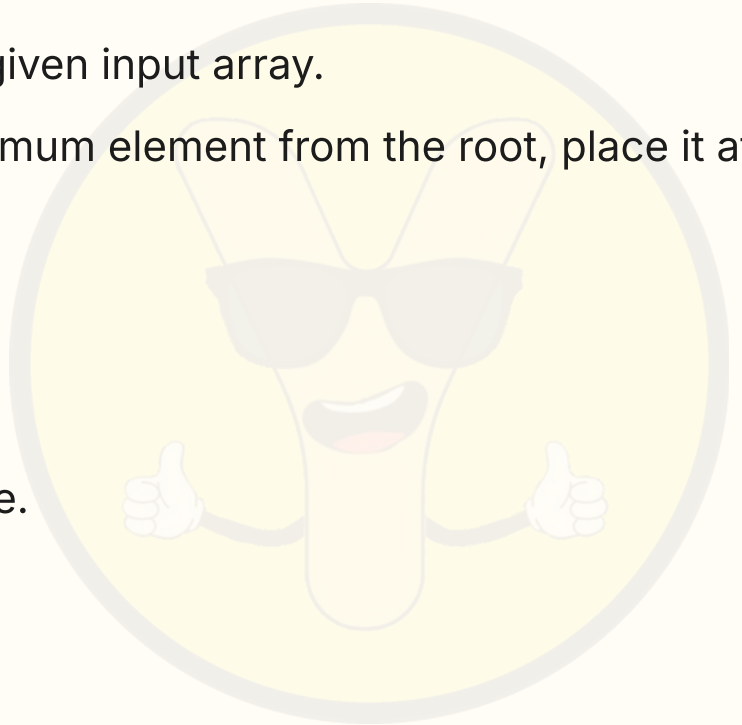
- Divides the array into two halves recursively until each sub-array contains a single element.
- Merges the sub-arrays back together in sorted order.
- Complexity
  - Best, Average, Worst Case: $O(n \log n)$
  - Space Complexity: $O(n)$ for temporary arrays
- Preferred for sorting linked lists and large datasets where stability is required.



Heap Sort

Uses a binary heap data structure to sort elements.

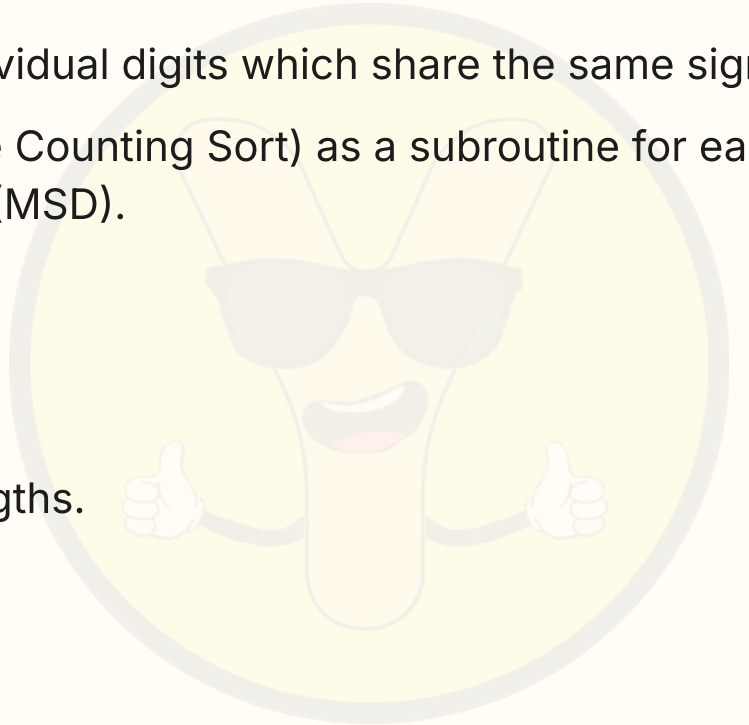
- Step 1: Build a Max-Heap from the given input array.
- Step 2: Repeatedly extract the maximum element from the root, place it at the end, and heapify the remaining tree.
- Complexity
  - Best, Average, Worst Case: $O(n \log n)$
  - Space Complexity: $O(1)$
- Not stable, but efficient and in-place.



Radix Sort

A non-comparative integer sorting algorithm.

- Sorts data by grouping keys by individual digits which share the same significant position.
- Uses a stable sorting algorithm (like Counting Sort) as a subroutine for each digit pass, from least significant digit (LSD) to most significant digit (MSD).
- Complexity
  - Time Complexity: $O(d(n+b))$
  - where d is digit length and b is base.
- Fast for numbers with fixed key lengths.



Sorting Algorithms Complexity Summary

Algorithm	Best Case	Worst Case	Stable?
Bubble Sort	O(n)	O(n^2)	Yes
Insertion Sort	O(n)	O(n^2)	Yes
Selection Sort	O(n^2)	O(n^2)	No
Quick Sort	O(n log n)	O(n^2)	No
Merge Sort	O(n log n)	O(n log n)	Yes
Heap Sort	O(n log n)	O(n log n)	No

Exam corner

- **Q1** Compare Sequential Search and Binary Search with suitable examples and time complexities.
- **Q2** What is hashing? Explain collision and discuss any two collision resolution techniques in detail.
- **Q3** Write the algorithm for Quick Sort and trace it with an unsorted array.
- **Q4** Explain Merge Sort using divide and conquer. Derive its time complexity.
- **Q5** Differentiate between internal and external sorting. Explain Heap Sort step by step.
- **Q6** Why is Insertion Sort preferred over Selection Sort for nearly sorted arrays?
- **Q7** Explain Radix Sort with an example of multi-digit numbers.

Unit 3 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 4 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

