What are Algorithms?

Algorithms are a set of instructions or steps designed to perform a specific task or to solve a particular problem. They are essential in computer science for data processing, calculations, and other tasks.

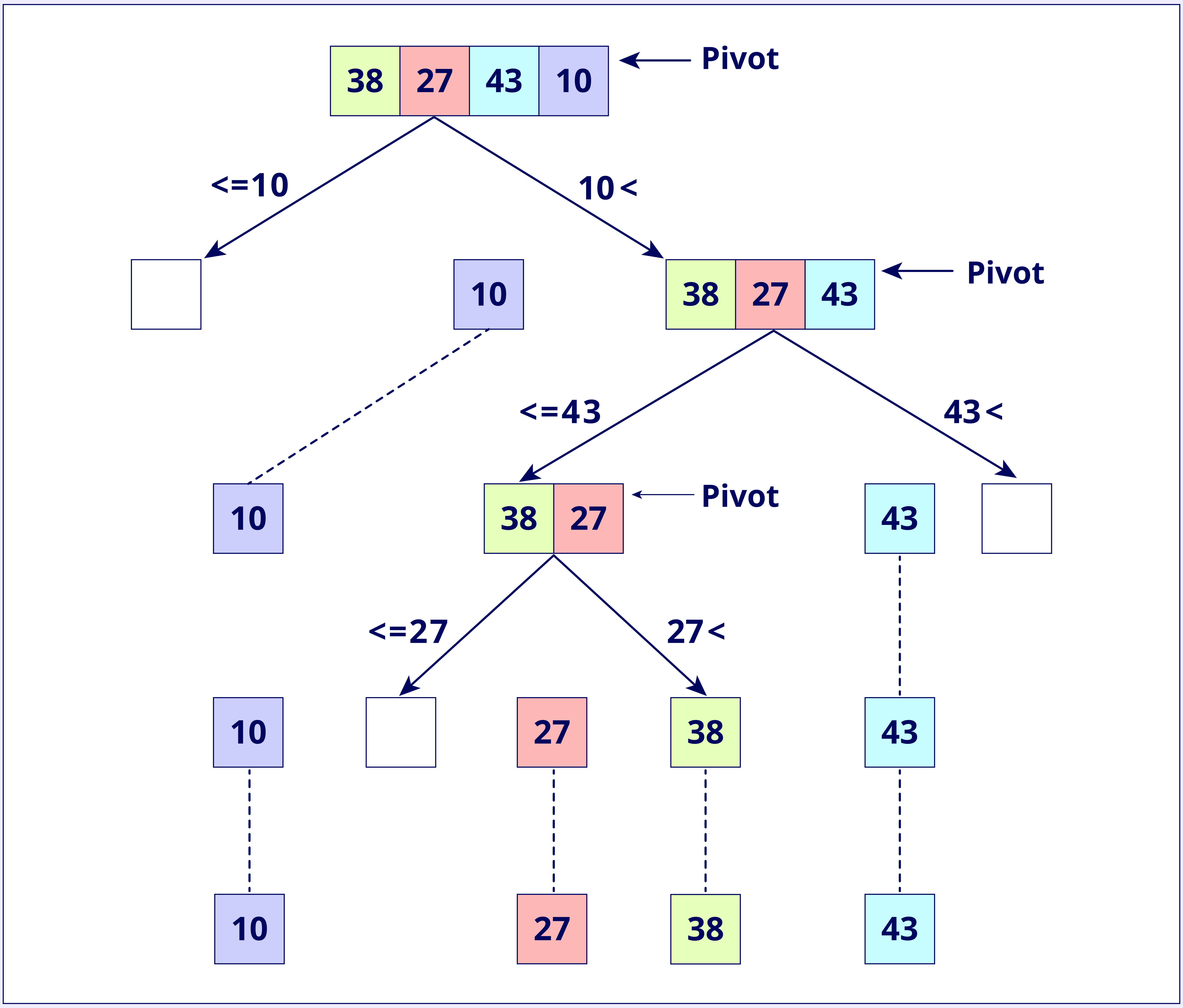
Algorithm Design Techniques (Definition + Applications + Algorithms)

Algorithm Design Technique	Definition	Applications	Examples
Brute Force	Simple and straightforward method of solving problems by trying all possible solutions until the correct one is found.	Finding the maximum element in an array, checking all substrings.	Naive string matching, Bubble sort.
Divide and Conquer	Breaking down a problem into smaller subproblems, solving each subproblem independently, and combining their solutions to solve the original problem.	Sorting algorithms, binary search.	Merge sort, Quicksort.
Dynamic Programming	Solving problems by breaking them down into simpler subproblems and storing the results of subproblems to avoid redundant work.	Optimization problems, sequence alignment.	Fibonacci sequence, Knapsack problem.
Greedy Algorithms	Making the locally optimal choice at each step with the hope of finding the global optimum.	Optimization problems, scheduling.	Dijkstra's algorithm, Prim's algorithm.
Backtracking	Building solutions incrementally and abandoning solutions that fail to satisfy the constraints of the problem.	Constraint satisfaction problems, puzzles.	N-Queens problem, Sudoku solver.
Randomized Algorithms	Using random numbers to make decisions during the algorithm’s execution.	Optimization problems, Monte Carlo methods.	Quick sort (randomized), Randomized algorithms for median finding.
Linear Programming	Optimizing a linear objective function subject to linear equality and inequality constraints.	Operations research, economics.	Simplex algorithm, Interior-point methods.

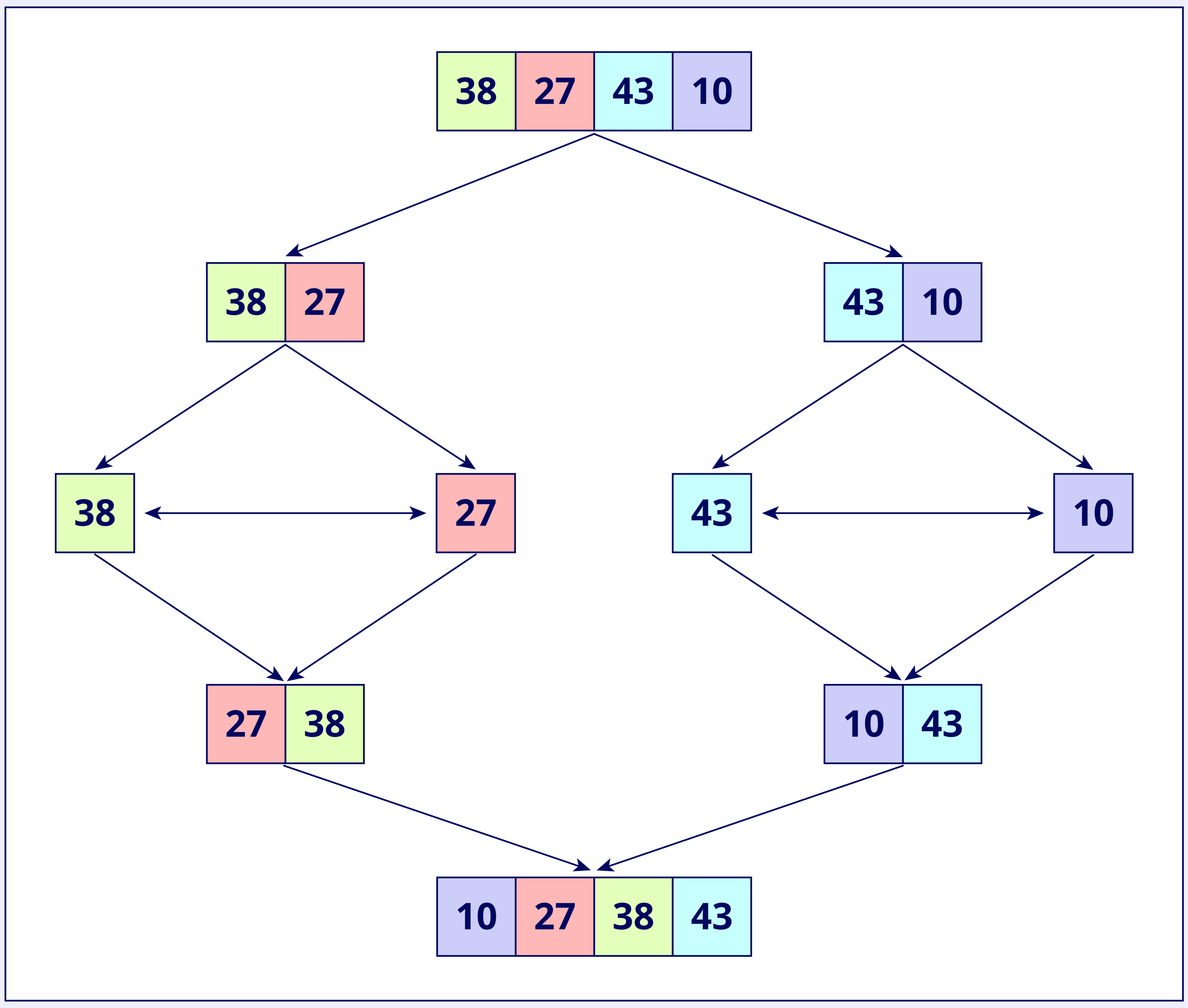
Branch and Bound	Systematically enumerating candidate solutions by means of state-space search.	Optimization problems, combinatorial problems.	Traveling salesman problem, Knapsack problem.
Reduction (Transform and Conquer)	Transforming a problem into a different version or into another problem entirely.	Problem-solving strategies, proving NP-completeness.	Reducing a problem to graph theory, reducing a sorting problem to a selection problem.
Minimum Spannin Trees	A subset of edges in a weighted graph that connects all the vertices without any cycles and with the minimum possible total edge weight.	Network design, circuit design.	Kruskal's algorithm, Prim's algorithm.

Classification by Design Approach (Definition + Illustration)

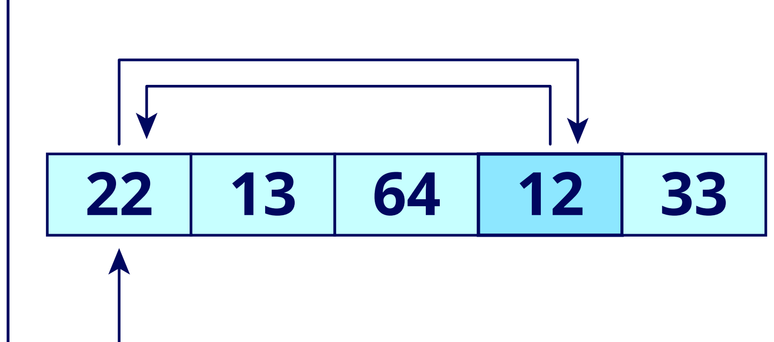
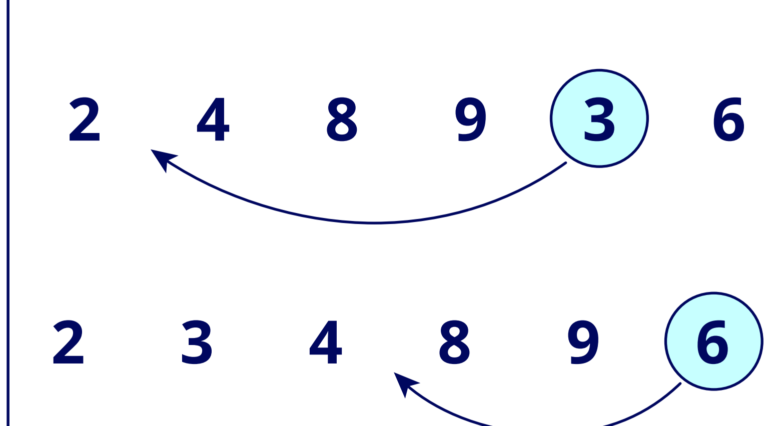
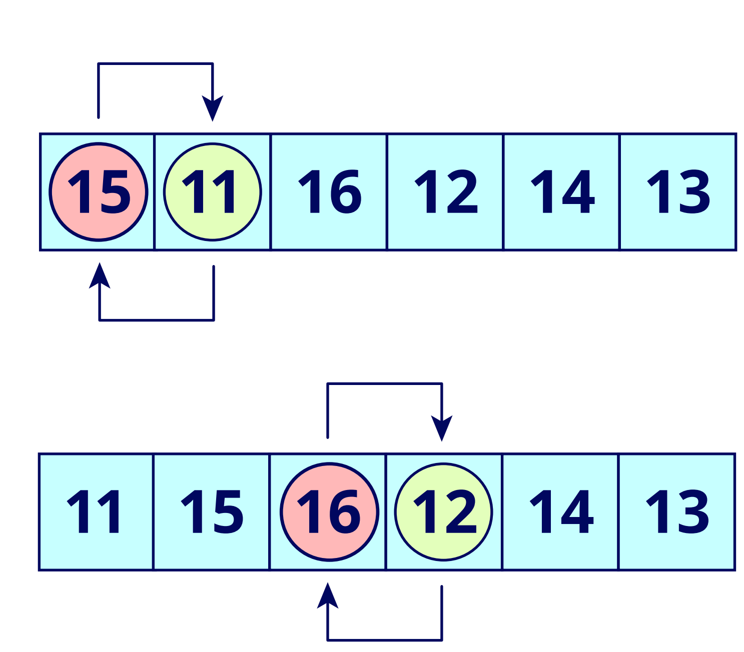
- **Top-Down Approach**
  - **Definition:** Breaking down a system into smaller subsystems to understand its compositional subcomponents.
  - **Illustration:** Recursively solving subproblems (e.g., Quicksort).

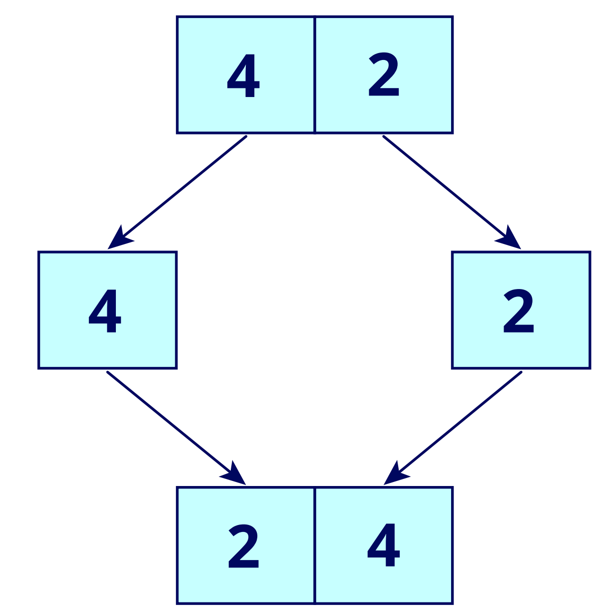
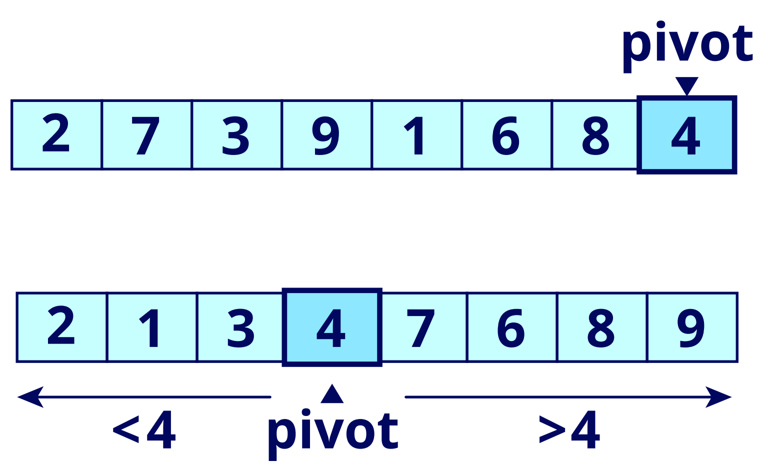
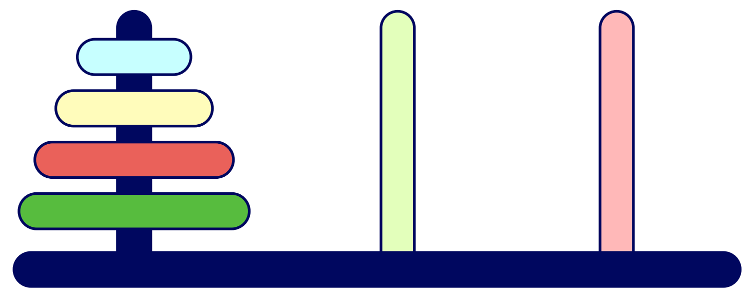
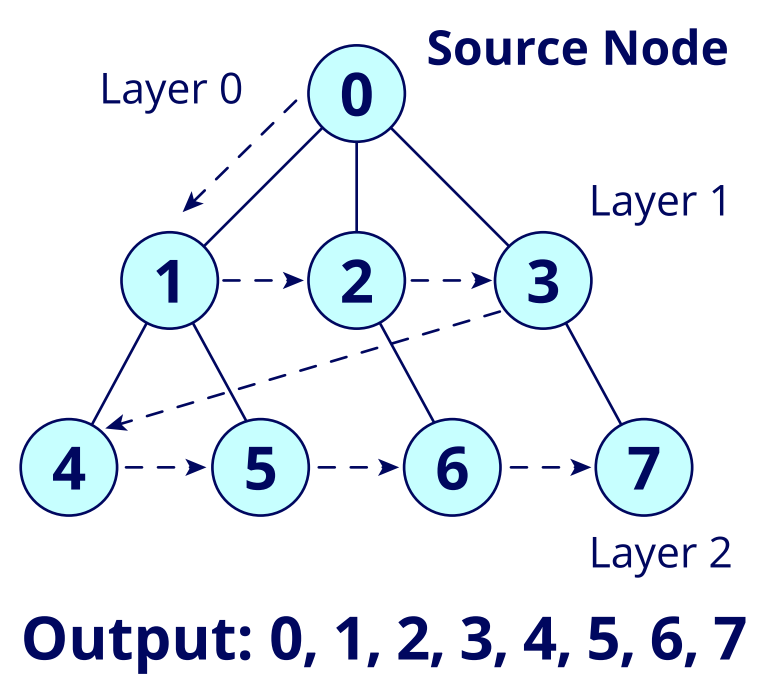
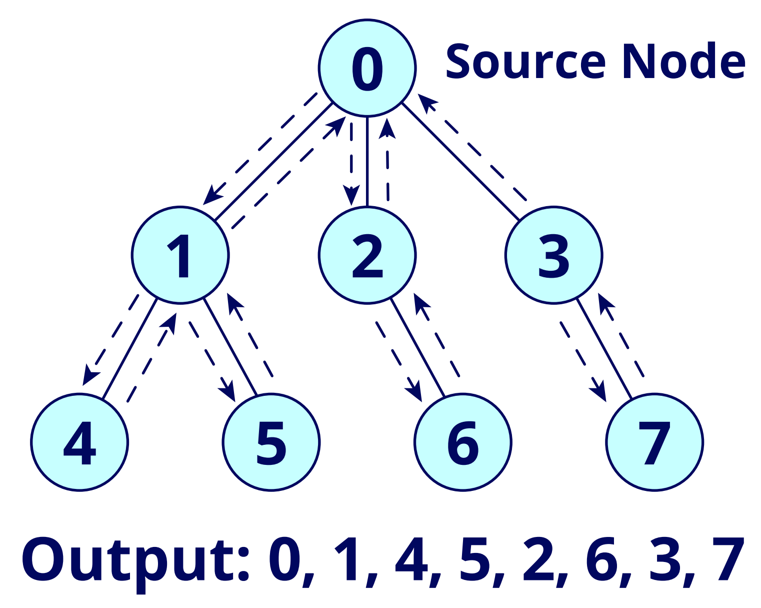
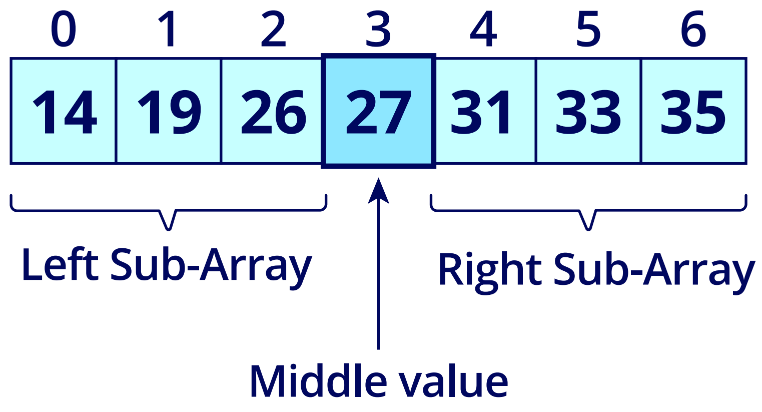


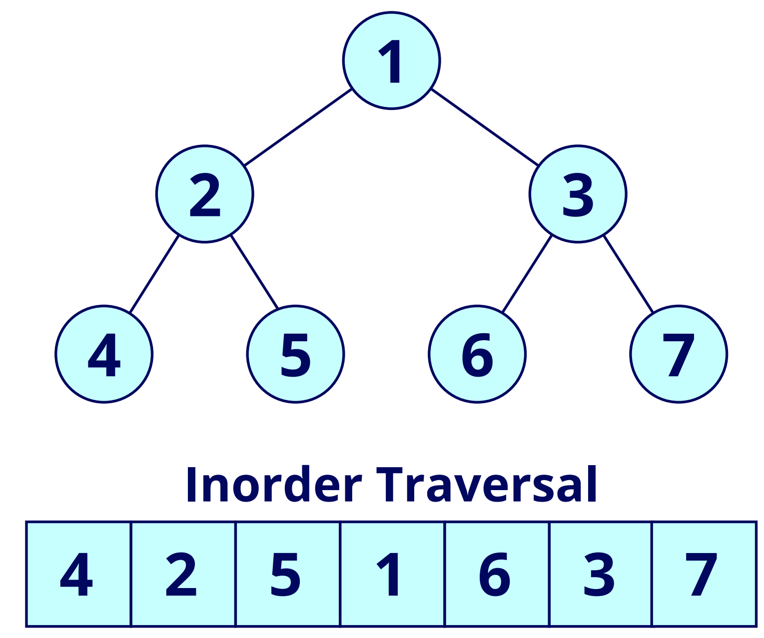
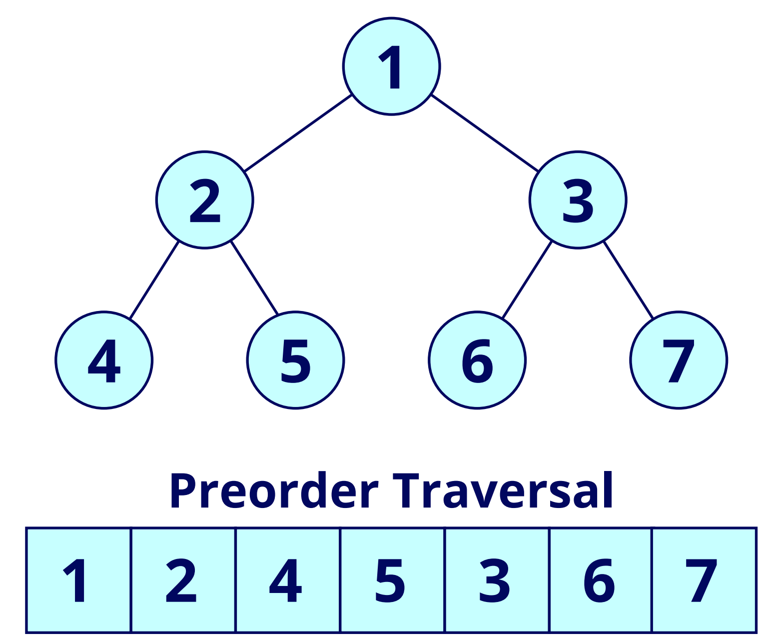
- **Bottom-Up Approach**
  - **Definition:** Building a system from the smallest subsystems up to the overall system.
  - **Illustration:** Combining solutions to subproblems to solve the main problem (e.g., Merge sort).



Discussion on Important Algorithms

Algorithm	Definition	Complexity	Why	When	
Selection Sort	Repeatedly finding the minimum element from the unsorted part and putting it at the beginning.	O(n^2)	Simple to understand and implement; useful for small datasets or when memory usage is a concern.	Suitable for small arrays or when the cost of swapping elements is low.	
Insertion Sort	Building a sorted array one element at a time by inserting elements into their correct position.	O(n^2)	Efficient for small dataset or nearly sorted data; in-place and stable sort.	Ideal for small arrays or arrays that are already partially sorted.	
Bubble Sort	Repeatedly swapping adjacent elements if they are in the wrong order.	O(n^2)	Easy to implement and understand; can be optimized to stop early if the array is already sorted.	Rarely used in practice, mainly for educational purposes or very small datasets.	

Merge Sort	Dividing the array into halves, sorting each half, and merging the sorted halves.	O(n log n)	Efficient and stable sort with a predictable O(n log n) time complexity.	Suitable for large datasets, especially when stability is required or when data cannot fit into memory.	
Quicksort	Partitioning the array into two parts, sorting each part, and combining them.	O(n log n) on average	Highly efficient for large datasets; average-case time complexity is O(n log n); in-place sort.	Preferred for general-purpose sorting, particularly when space is limited, and average performance is critical.	
Tower of Hanoi	Moving disks from one rod to another, following specific rules.	O(2^n)	Classic problem for understanding recursion and algorithmic problem-solving.	Primarily used for educational purposes to teach recursion.	
Breadth-First Search (BFS)	Exploring all neighbors at the present depth before moving on to nodes at the next depth level.	O(V + E)	Useful for finding the shortest path in unweighted graphs; explores all neighbors before moving on to the next level.	Ideal for finding the shortest path or level-order traversal in graphs or trees.	
Depth-First Search (DFS)	Exploring as far as possible along each branch before backtracking.	O(V + E)	Explores as deep as possible before backtracking; useful for pathfinding and topological sorting.	Suitable for scenarios where you need to explore all possible paths, such as maze solving or topological sorting.	
Binary Search	Repeatedly dividing the search interval in half and eliminating the half that does not contain the target value.	O(log n)	Extremely efficient for searching in sorted arrays; O(log n) time complexity.	Ideal for searching in large, sorted datasets where quick lookup times are essential.	

Inorder	Traverse the left subtree, visit the root node, then traverse the right subtree.	O(n)	Provides nodes in non-decreasing order for binary search trees; useful for retrieving sorted data.	Useful for binary search trees and for applications where nodes need to be processed in sorted order.	
Preorder	Visit the root node, then traverse the left subtree, followed by the right subtree.	O(n)	Used to create a copy of the tree or to get a prefix expression of an expression tree.	Useful for prefix notation expressions and when you need to process the root node before the subtrees.	
Postorder	Traverse the left subtree, then the right subtree, and visit the root node last.	O(n)	Used to delete or free nodes in a tree; useful for postfix expression of an expression tree.	Useful for deleting trees and for applications where nodes need to be processed after their subtrees.