Project presentation

Presented by Minh Chinh

Contens

01. A stack ADT, a concrete data structure for a First In First out (FIFO) queue.

02. Two sorting algorithms.

03. Two network shortest path algorithms.

1. A stack ADT, a concrete data structure for a First In First out (FIFO) queue.

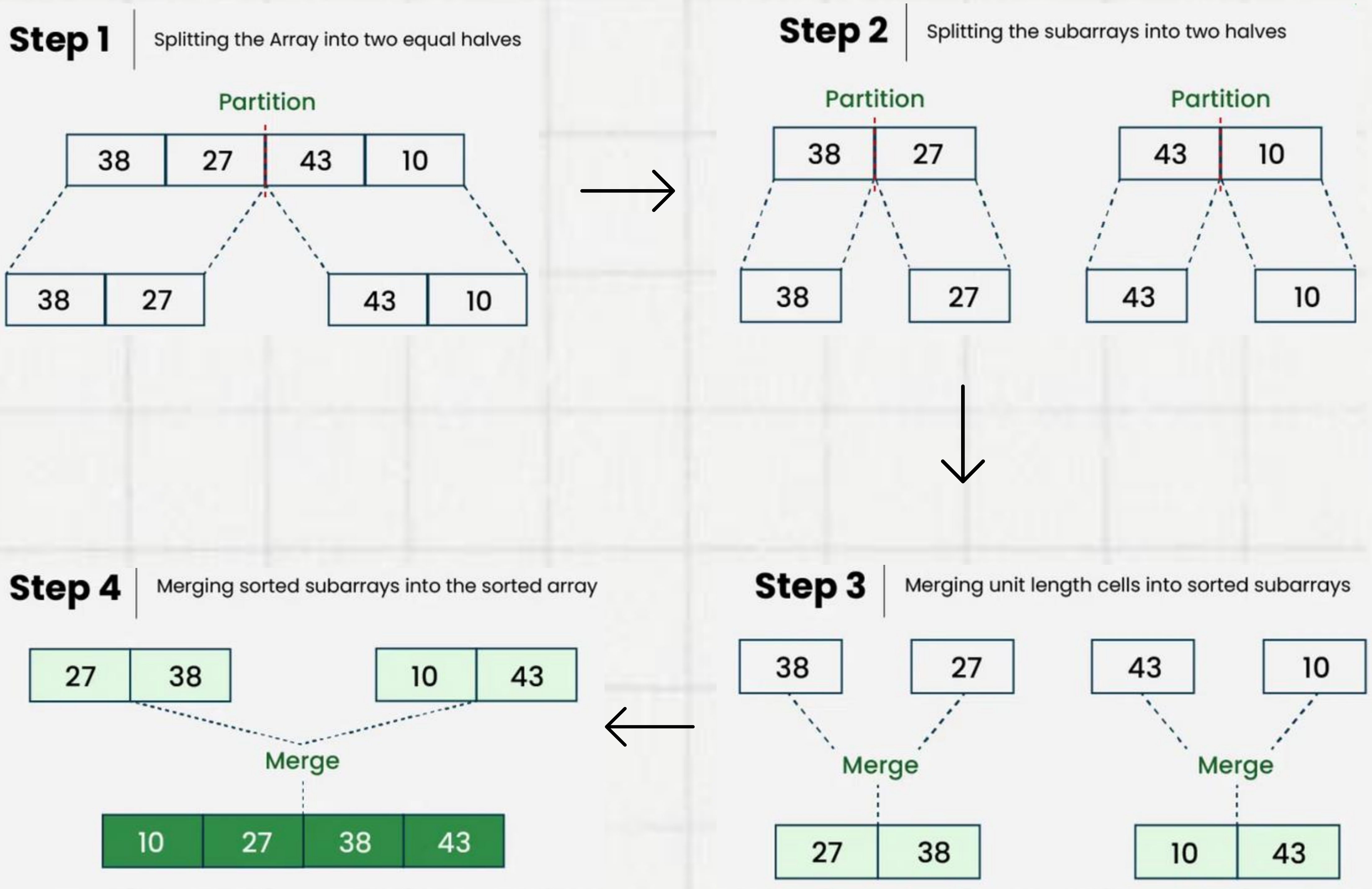
-Stack (LIFO):
Last In, First Out, meaning the most recently added item is removed first.
Example: An undo function, where the most recent change is the first to be undone.
-Queue (FIFO):
First In, First Out, meaning the earliest added item is removed first.
Example: A line at a ticket counter, where the first person in line is served first.
Implementation Differences

-In a stack, you add (push) and remove (pop) elements from the top only.
In a queue, you add elements at the rear and remove them from the front.

02. Merge sort

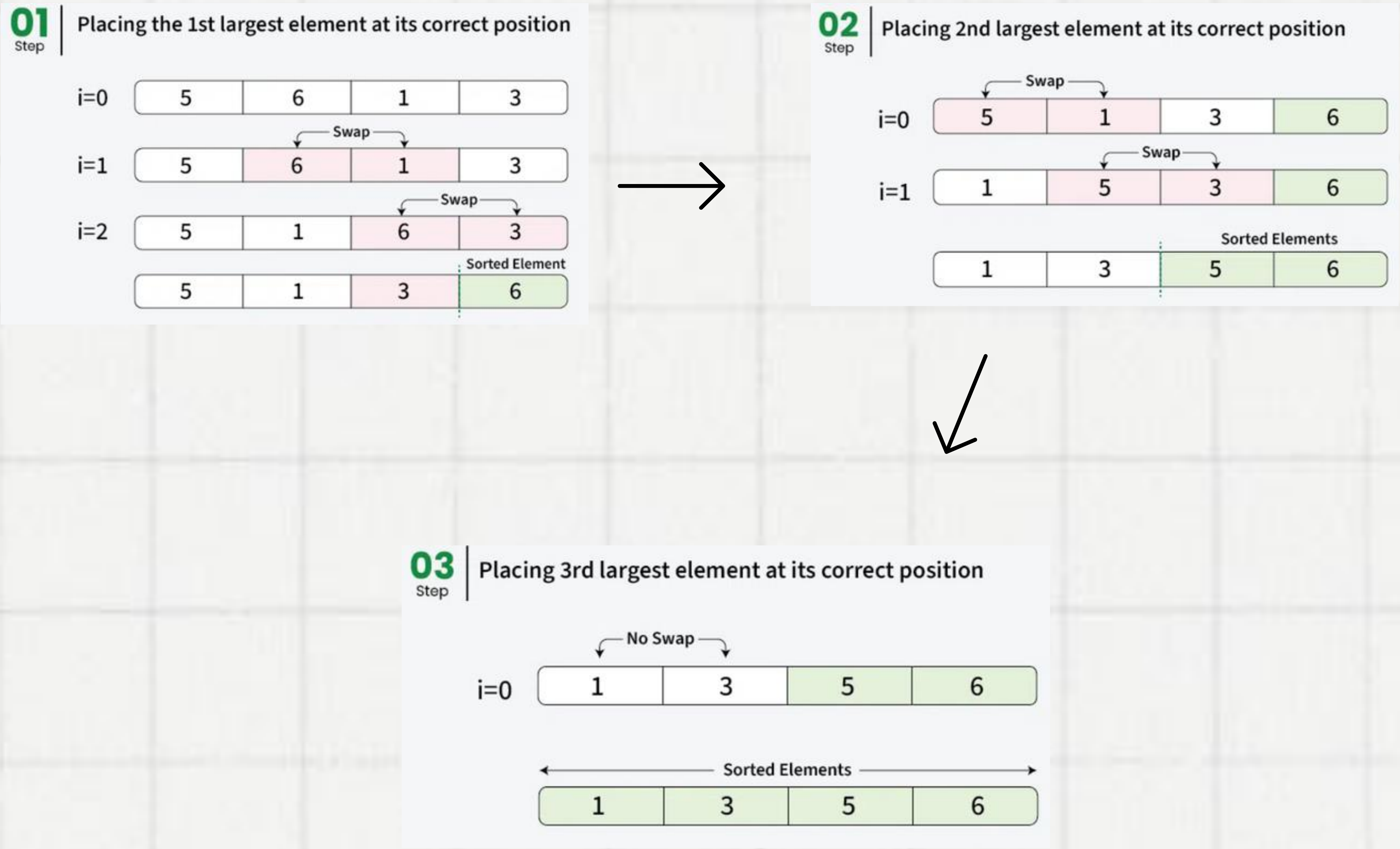
-Merge sort is a sorting algorithm that follows the divide-and-conquer approach. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

-In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



03.Bubble Sort

-Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity are quite high.



04. Dijkstra's Algorithm

-Dijkstra's Algorithm is a graph-based algorithm used to find the shortest path between a starting node and all other nodes in a weighted graph. It guarantees that, when it terminates, the shortest path from the source node to every other node is known. The algorithm works for graphs with non-negative weights and is widely used in applications like GPS navigation, network routing protocols, and many other optimization scenarios.

-Purpose

Dijkstra's Algorithm is primarily used for single-source shortest path problems:

- To find the shortest path from a source node to a destination node.
- To find the shortest path from a source node to all other nodes in the graph.

-Steps of Dijkstra's Algorithm

1. Initialization:
    - Set the distance to the starting node (source) as 0 and to all other nodes as infinity (∞).
    - Mark all nodes as unvisited. Create a set of unvisited nodes.
2. Select Node:
    - Pick the unvisited node with the smallest known distance (initially, this is the source node). Let's call this node current.
3. Update Neighbors:
    - For each unvisited neighbor of the current node, calculate the tentative distance through the current node.
    - If the tentative distance is smaller than the previously recorded distance for that neighbor, update it with the smaller distance.
4. Mark Node as Visited:
    - Once the neighbors have been updated, mark the current node as visited. A visited node will not be checked again.
5. Repeat:
    - Repeat steps 2-4 until all nodes have been visited or until the smallest tentative distance among the unvisited nodes is infinity. If the smallest distance is infinity, the remaining unvisited nodes are not connected to the source, and the algorithm can stop.

05.Prim-Jarnik Algorithm

The Prim-Jarnik algorithm, more commonly referred to as Prim’s Algorithm, is a classic greedy algorithm in graph theory used to find a minimum spanning tree (MST) for a connected, undirected graph. A minimum spanning tree is a subgraph that connects all the vertices of the original graph without any cycles and with the minimum possible total edge weight.

Purpose

The goal of Prim's algorithm is to construct a spanning tree that covers all vertices in a graph while minimizing the sum of the edge weights. It is used when we need to connect multiple points in the most cost-effective way, such as in network design, like wiring buildings or connecting nodes in a computer network.

Steps of Prim’s Algorithm

1. Initialization:
    - Start with an arbitrary node (vertex) and treat it as part of the growing spanning tree.
    - Mark this vertex as visited.
2. Edge Selection:
    - From the set of all edges that connect the visited nodes to unvisited nodes, pick the edge with the smallest weight.
3. Expand the Tree:
    - Add the edge and the connected unvisited vertex to the tree.
    - Mark this newly connected vertex as visited.
4. Repeat:
    - Repeat the process of selecting the smallest edge that connects the tree to a new unvisited node until all vertices are included in the tree.

Thank you very much!