# IGNOU | BCA

2023-2024

FREE

All In One

# SOLVED ASSIGNMENT

## BCS-042

## Introduction to Algorithm Design

Prepared by

ISM

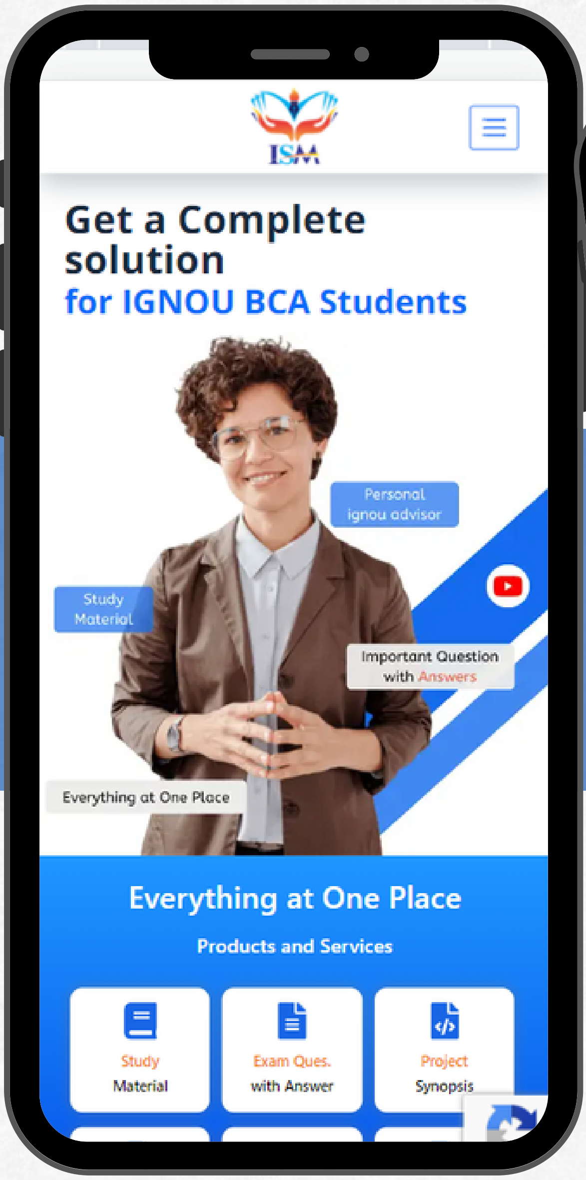
730-339-3555

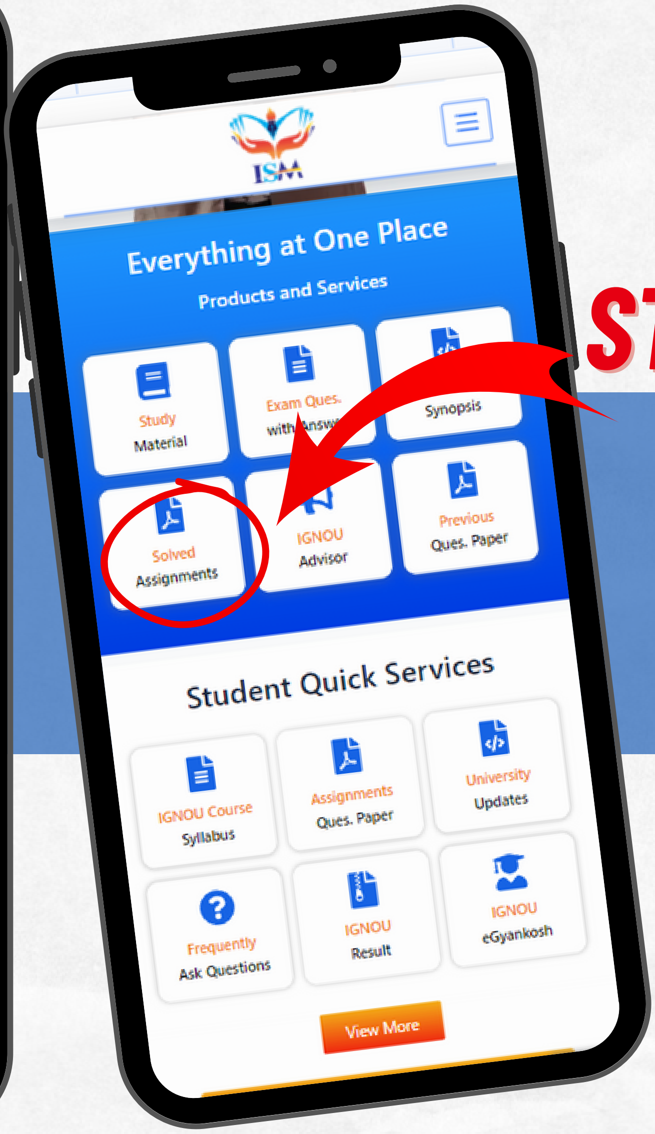
www.ignoustudymentor.com

@ism_ignoustudymentor_

# STEP 1

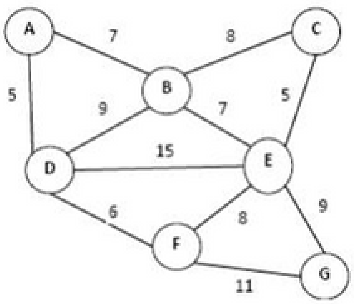
www.ignoustudymentor.com




# STEP 2

**Ques.6 Write Kruskal's algorithm for finding minimum cost spanning tree using greedy approach and apply to the following graph and show step by step results**



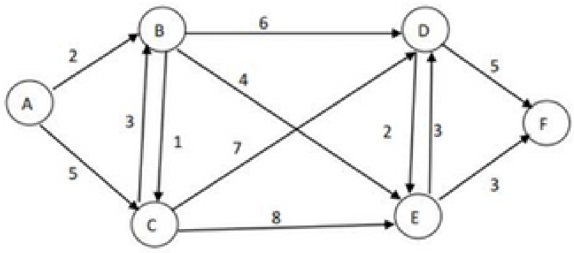

**Ans.** Kruskal's algorithm is a greedy algorithm used to find the minimum cost spanning tree of a connected, undirected graph. It operates by iteratively adding edges to the spanning tree while ensuring that no cycles are formed. Here's how Kruskal's algorithm works:

1. Create a list of all the edges in the graph, sorted in ascending order of their weights.
2. Initialize an empty set to represent the minimum cost spanning tree.
3. Iterate through the sorted edges: a. If adding the current edge to the spanning tree does not form a cycle, add it to the spanning tree set.

b. Otherwise, skip the edge.

4. Continue this process until the spanning tree set has V- 1 edges, where V is the number of vertices in the graph.

**Ques.7 Whatis edge relaxation technique in shortest path algorithm? Write and apply Bellman Ford's algorithm to find the shortest path from a node A to all the remaining nodes in the following graph:**




**Ans.** Edge relaxation is a fundamental technique used in shortest path algorithms to continuously update the minimum distance estimate from a source node to all other nodes in a graph. It involves examining each edge of the graph and updating the distance to the destination node if a shorter path is found. The process iterates until no further updates can be made.

The Bellman-Ford algorithm is a single-source shortest path algorithm that utilizes edge relaxation to find the shortest paths from a given source node to all other nodes in a weighted graph, even in the presence of negative edge weights. Here's how the Bellman-Ford algorithm works:

1. Initialize the distance from the source node to itself as 0, and set the distance to all other nodes as infinity.

2. Repeat the following steps for a total of V- 1 iterations, where V is the number of nodes in the graph: a.Iterate through all edges (u, v) in the graph. b. For each edge (u, v), perform edge relaxation: If distance[u] + weight(u, v) < distance[v], update distance[v] with the new distance distance[u] + weight(u, v).

3.After V- 1 iterations, all shortest paths are guaranteed to be found.

**Ques.8 Write Quick Sort algorithm to sort the following list of integer numbers. Show all the intermediate steps**
**15, 12, 18, 5, 6, 8, 22, 3, 25, 30, 35, 8, 32**
**Also compute the worst case time complexity of the algorithm.**

**Ans.** The Quick Sort algorithm to sort the given list of integer numbers:

```
 plaintext function quickSort(arr, low, high):
 if low < high:
pivotIndex = partition(arr, low, high)
quickSort(arr, low, pivotIndex- 1)
quickSort(arr, pivotIndex + 1, high)

function partition(arr, low, high):
pivot = arr[high]
i = low- 1
for j = low to high- 1:
if arr[j] <= pivot:
 i = i +1
```

swap(arr[i], arr[j])
swap(arr[i + 1], arr[high])
return i + 1

Apply the Quick Sort algorithm to sort the given list step by step:
Given list: 15, 12, 18, 5, 6, 8, 22, 3, 25, 30, 35, 8, 32

1. Applying `quickSort(arr, 0, 12)`:

- Select pivot: 32
- Partitioning: 15, 12, 18, 5, 6, 8, 22, 3, 25, 30, 35, 8, 32
 i
 j
Swap 18 and 8: 15, 12, 8, 5, 6, 18, 22, 3, 25, 30, 35, 8, 32
i
j
Swap 6 and 22:
15, 12, 8, 5, 6, 18, 22, 3, 25, 30, 35, 8, 32
i
j
Swap 5 and 18:
15, 12, 8, 5, 6, 18, 22, 3, 25, 30, 35, 8, 32
i
j

Swap 8 and 25:
15, 12, 8, 5, 6, 18, 22, 3, 25, 30, 35, 8, 32
i
j
- Swap pivot 32 with the element at index i + 1:
15, 12, 8, 5, 6, 18, 22, 3, 8, 30, 35, 25, 32
i+1 pivot

2. Applying `quickSort(arr, 0, 7)`:
- Select pivot: 3
- Partitioning: 15, 12, 8, 5, 6, 8, 22, 3, 8, 30, 35, 25, 32
i
j
- Swap 8 and 8 (no change): 15, 12, 8, 5, 6, 8, 22, 3, 8, 30, 35, 25, 32
i
j
- Swap 5 and 8: 15, 12, 8, 5, 6, 8, 22, 3, 8, 30, 35, 25, 32
i
j
- Swap 3 and 8: 15, 12, 8, 5, 6, 8, 22, 8, 3, 30, 35, 25, 32
i
j
- Swap pivot 3 with the element at index i + 1: 3, 12, 8, 5, 6, 8, 22, 15, 8, 30, 35, 25, 32
i+1
pivot

3. Applying `quickSort(arr, 9, 12)`:
- Select pivot: 32
- Partitioning: 3, 12, 8, 5, 6, 8, 22, 15, 8, 30, 35, 25, 32
i
- Swap pivot 32 with the element at index i + 1: 3, 12, 8, 5, 6, 8, 22, 15, 8, 30, 35, 25, 32
i+1
pivot

4. Applying `quickSort(arr, 0, 3)`:
- Select pivot: 6
- Partitioning: 3, 12, 8, 5, 6, 8, 22, 15, 8, 30, 35, 25, 32
i
j
- Swap 5 and 12: 3, 5, 8, 12, 6, 8, 22, 15, 8, 30, 35, 25, 32
i
j
- Swap 6 and 8: 3, 5, 6, 12, 8, 8, 22, 15, 8, 30, 35, 25, 32
i
j
- Swap pivot 6 with the element at index i + 1: 3, 5, 6, 12, 8, 8, 22, 15, 8, 30, 35, 25, 32
i+1
pivot

5. Applying `quickSort(arr, 5, 6)`:- This subarray is already sorted, so no changes are needed.

6. Applying `quickSort(arr, 8, 12)`:- This subarray is already sorted, so no changes are needed.

7. Applying `quickSort(arr, 9, 12)`:- This subarray is already sorted, so no changes are needed.

The sorted list is: 3, 5, 6, 8, 8, 8, 12, 15, 22, 25, 30, 32, 35

The worst-case time complexity of the Quick Sort algorithm is O(n^2), which occurs when the pivot selection consistently divides the array into two unbalanced subarrays. However, on average, the Quick Sort algorithm has an expected time complexity of O(n log n), making it a very efficient sorting algorithm.