Certainly! Here's a comprehensive breakdown of the topics you've mentioned for your upcoming Advanced Data Structures and Algorithms exam at Dhaka University, Institute of IT, along with some sample questions to aid your preparation.

**1. Recurrence Relation Tree**

**Sub-topics:**

* **Definition**: Representation of recursive algorithms through a tree structure.
* **Node Structure**: Each node represents a recursive call.
* **Leaf Nodes**: Base cases of the recursion.
* **Cost Calculation**: Determining the cost at each level and summing them up.
* **Height of the Tree**: Depth of recursion.
* **Total Cost**: Sum of costs from all levels.

**Sample Question:**

* **Given the recurrence relation T(n) = 2T(n/2) + n, draw the recurrence tree and calculate the total cost.**

**2. Substitution Method**

**Sub-topics:**

* **Guessing the Form**: Proposing a solution for the recurrence.
* **Inductive Proof**: Verifying the guessed solution using mathematical induction.
* **Base Case**: Establishing the validity for small inputs.
* **Inductive Step**: Proving the validity for larger inputs assuming it's true for smaller ones.

**Sample Question:**

* **Use the substitution method to solve the recurrence T(n) = 3T(n/4) + n².**([GeeksforGeeks](https://www.geeksforgeeks.org/recurrence-relations-a-complete-guide/?utm_source=chatgpt.com" \o "Recurrence Relations | A Complete Guide | GeeksforGeeks))

**3. Master's Theorem**

**Sub-topics:**

* **Standard Form**: T(n) = aT(n/b) + f(n).
* **Case Analysis**:
  + Case 1: If log\_b(a) > d, then T(n) = O(n^log\_b(a)).
  + Case 2: If log\_b(a) = d, then T(n) = O(n^d log n).
  + Case 3: If log\_b(a) < d, then T(n) = O(n^d).
* **Conditions**: Understanding the constraints and applicability.([TutorialsPoint](https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_masters_theorem.htm?utm_source=chatgpt.com" \o "Master’s Theorem))

**Sample Question:**

* **Apply the Master's Theorem to determine the time complexity of T(n) = 4T(n/2) + n³.**

**4. Greedy Algorithms**

**Sub-topics:**

* **Greedy Choice Property**: Making the locally optimal choice at each stage.
* **Optimal Substructure**: A problem has an optimal solution that can be constructed efficiently from optimal solutions of its subproblems.
* **Algorithm Design**: Constructing algorithms that follow the greedy approach.
* **Proof of Correctness**: Verifying that the greedy approach leads to an optimal solution.
* **Examples**:
  + Fractional Knapsack Problem
  + Activity Selection Problem
  + Huffman Coding
  + Dijkstra's Shortest Path Algorithm
  + Prim's and Kruskal's Algorithms for Minimum Spanning Tree([GeeksforGeeks](https://www.geeksforgeeks.org/greedy-algorithms-/?utm_source=chatgpt.com" \o "Greedy Algorithms - GeeksforGeeks), [GeeksforGeeks](https://www.geeksforgeeks.org/greedy-algorithms/?utm_source=chatgpt.com))

**Sample Question:**

* **Solve the Activity Selection Problem using a greedy algorithm.**

**5. Asymptotic Notations**

**Sub-topics:**

* **Big O Notation (O)**: Upper bound of the running time.
* **Big Omega Notation (Ω)**: Lower bound of the running time.
* **Big Theta Notation (Θ)**: Tight bound of the running time.
* **Little o Notation (o)**: Upper bound that is not tight.
* **Little omega Notation (ω)**: Lower bound that is not tight.
* **Best, Worst, and Average Case Analysis**: Evaluating algorithms under different scenarios.

**Sample Question:**

* **Determine the asymptotic complexity of the following code segment:**

for i in range(n):

for j in range(i, n):

print(i, j)

**6. Sorting Algorithms**

**Sub-topics:**

* **Comparison-Based Sorting**:
  + Bubble Sort
  + Selection Sort
  + Insertion Sort
  + Merge Sort
  + Quick Sort
  + Heap Sort
* **Non-Comparison-Based Sorting**:
  + Counting Sort
  + Radix Sort
  + Bucket Sort
* **Stable vs Unstable Sorting**: Understanding the difference and its implications.
* **Time and Space Complexity**: Analyzing the efficiency of sorting algorithms.
* **Applications**: Real-world scenarios where sorting algorithms are applied.([GeeksforGeeks](https://www.geeksforgeeks.org/classification-of-sorting-algorithms/?utm_source=chatgpt.com" \o "Classification of Sorting Algorithms | GeeksforGeeks), [GeeksforGeeks](https://www.geeksforgeeks.org/sorting-algorithms/?utm_source=chatgpt.com))

**Sample Question:**

* **Compare and contrast Merge Sort and Quick Sort in terms of time complexity, space complexity, and stability.**

Feel free to reach out if you need further clarification or additional practice problems. Best of luck with your exam!