Unit 1

Analysis of Non-Recursive Algorithms

- **Steps:**
  - Identify the basic operations
  - Count the number of basic operations
  - Express the count as a function of input size
- **Example:** Linear search

Understanding Non-Recursive Algorithms

- **Definition:**
  - Algorithms that do not call themselves during their execution.
- **Examples:**
  - Iterative processes like looping through an array.
  - Algorithms like linear search, bubble sort, insertion sort, etc.

Slide 6b: Steps to Analyze Non-Recursive Algorithms

- **Identify the Basic Operations:**
  - Determine the most significant operation that contributes to the running time (e.g., comparisons, assignments).
- **Count the Number of Basic Operations:**
  - Establish how many times the basic operation is executed based on the input size.
- **Express the Count as a Function of Input Size:**
  - Create a mathematical expression that represents the total number of operations in terms of the input size n.

Example - Linear Search

- **Problem Statement:**
  - Given an array of n elements, find a target element.
- **Algorithm:**
  - Start from the first element.
  - Compare the target element with each element in the array.
  - If the target is found, return its position.
  - If the target is not found after checking all elements, return -1.

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1
```

Time Complexity Analysis of Linear Search

- **Identify Basic Operation:**
  - Comparison of elements (arr[i] == target)
- **Count Basic Operations:**
  - In the worst case, all n elements are compared.
- **Express as a Function of Input Size:**
  - Worst-case time complexity: $T(n)=n$
  - Therefore, $T(n)$ is $O(n)$.

Analysis of Recursive Algorithms

- **Steps:**
  - Define the recurrence relation
  - Solve the recurrence relation
  - Use the solution to express time complexity
- **Example:** Binary search

Understanding Recursive Algorithms

- **Definition:**
  - Algorithms that call themselves with a subset of the original problem.
- **Examples:**
  - Divide and conquer algorithms like merge sort, quicksort.
  - Recursive algorithms like factorial calculation, Fibonacci sequence.

Steps to Analyze Recursive Algorithms

- **Define the Recurrence Relation:**
  - Establish a mathematical relation that describes the time complexity in terms of the input size.
- **Solve the Recurrence Relation:**
  - Use methods such as substitution, recursion tree, or the Master theorem to solve the recurrence.
- **Express Time Complexity:**
  - Derive the overall time complexity from the solution of the recurrence relation.

Binary Search

- **Problem Statement:**
  - Given a sorted array of nnn elements, find the target element.
- **Algorithm:**
  - Compare the target with the middle element.
  - If equal, return the position.
  - If less, repeat the search on the left subarray.
  - If more, repeat the search on the right subarray.

```python
def binary_search(arr, low, high, target):
    if high >= low:
        mid = (high + low) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] > target:
            return binary_search(arr, low, mid - 1, target)
        else:
            return binary_search(arr, mid + 1, high, target)
    else:
        return -1
```

- **Define the Recurrence Relation:**
  - Each call reduces the problem size by half.
  - $T(n) = T(n/2) + O(1)$
- **Solve the Recurrence Relation:**
  - Using the Master theorem: $T(n) = T(n/2) + O(1)$
  - Here, $a = 1$, $b = 2$, and $f(n) = O(1)$.
  - $f(n) = O(n^c)$ where $c = 0$.
  - Since $f(n)$ is $O(n^{\log_b a}) = O(n^0) = O(1)$, we use Case 2 of the Master theorem: $T(n) = O(\log n)$.
- **Express Time Complexity:**
  - Therefore, $T(n) = O(\log n)$.

Understanding Amortized Analysis

- **Definition:**
  - Amortized analysis provides the average time per operation over a sequence of operations, ensuring that occasional expensive operations are accounted for over time.
- **Why Amortized Analysis?**
  - To give a more accurate overall performance metric than worst-case or average-case analysis for each operation.

Types of Amortized Analysis

- **Aggregate Method:**
  - Calculate the total cost of nnn operations and divide by nnn to find the average cost per operation.
- **Accounting Method:**
  - Assign different costs to operations, charging more than the actual cost for cheaper operations to cover the cost of expensive ones.
- **Potential Method:**
  - Use a potential function to represent the pre-paid work stored in the data structure, which can be used to pay for future operations.

Dynamic Array Resizing (Aggregate Method):

- **Problem Statement:**
  - Implement a dynamic array that resizes itself when full.
- **Operations:**
  1. Insertion
  2. Resizing (doubling the array size)
- **Analysis:**
  - **Total Cost:**
    - Insertions: n operations
    - Resizings: $\log n$ resizing operations (since array size doubles each time)
  - **Cost Breakdown:**
    - Each insertion typically costs $O(1)$.
    - Each resizing operation costs $O(n)$ but occurs less frequently.
  - **Amortized Cost:**
    - Total insertion cost: $O(n)$
    - Total resizing cost: $O(n)$ (since each element is copied once for each doubling)
    - Total cost: $O(n) + O(n) = O(2n) = O(n)$
    - Amortized cost per operation: $\frac{O(n)}{n} = O(1)$.

Writing Characteristic Polynomial Equations

- **Purpose:**
  - To solve linear recurrence relations.
- **Example:**
  - Recurrence relation: $T(n) = 2T(n/2) + n$
  - Characteristic equation: $x^2 - 2x + 1 = 0$

Solving Recurrence Equations

- **Definition:**
- A recurrence equation defines a sequence where each term is a function of preceding terms.
- To find a closed-form expression for the sequence.

- **Why Solve Recurrence Equations?**
- To analyze the performance of recursive algorithms.

Solving Recurrence Equations Methods:

- Substitution method
- Master theorem
- Iteration method

Proof Techniques: By Contradiction

- **Steps:**
  - Assume the opposite of what you want to prove.
  - Show that this assumption leads to a contradiction.
- **Example:**
  - Proving the irrationality of √2

Proof Techniques: By Mathematical Induction

- **Steps:**
  - Base case: Prove the statement for the initial value.
  - Induction step: Assume the statement for k, prove for k+1.
- **Example:**
  - Proving the formula for the sum of the first n natural numbers