Amortized Analysis – Summary

1. Introduction

Amortized Analysis is a method used to analyze the **average running time of a sequence of operations** on a data structure over time, rather than the worst-case time of a single operation. It ensures that even if some operations are expensive, the **average cost per operation is small**.

It is different from average-case analysis, which depends on input distribution. Amortized analysis gives guarantees regardless of input.

2. Techniques of Amortized Analysis

a) Aggregate Method

- Calculate total cost of n operations.
- Divide by n to get **amortized cost per operation**.
- Example: In a dynamic array, occasional resizing is expensive, but the total cost divided over all operations remains low.

b) Accounting Method

- Assign an **amortized cost** to operations.
- Overcharge cheap operations to pay for expensive ones.
- Example: In a stack with multiple pop operations, extra credit from push operations can pay for costly pops.

c) Potential Method

- Define a **potential function** representing the state of the data structure.
- Amortized cost = actual cost + change in potential.
- Example: In a binary counter, flipping many bits is expensive, but the potential function ensures the average cost per increment is small.

3. When to Use Amortized Analysis

- Data structures with occasional expensive operations:
- Dynamic arrays (resize)
- Stack with multipop
- Binary counters
- Fibonacci heaps
- Helps guarantee performance over time, not just per operation.

4. Summary Table

Method	Idea	Example	Amortized Cost
Aggregate	Average total cost	Dynamic array push	O(1)
Accounting	Prepay expensive operations	Stack multipop	O(1)
Potential	Use potential function	Binary counter	O(1)

Key Points: - Amortized analysis = **average cost per operation over a sequence**. - Provides **guaranteed bounds**, independent of input distribution. - Three main methods: **Aggregate, Accounting, Potential**. - Very useful in **data structure design and analysis**.