Name: Piyusha Rajendra Supe. BE Computer
Roll No. 23C0315 - B.

Design and Analysis of Algorithms -

Assignment - DAA.

Unit V -

Q1] Define amortized analysis. How does it differ from worst case and average case analysis?

→ ∘ Amortized analysis measures the average cost per operation over a sequence of operations, even if some operations are expensive.
∘ It guarantees an upper bound on the overall time per operation rather than on individual operations.
∘ Commonly used in dynamic data structures like dynamic arrays, splay trees, hash tables with rehashing.

∘ Difference from worst case analysis -
- Worst case looks at maximum cost of a single operation.
- amortized analysis spreads occasional high costs over many cheap operations to get a tighter bound.

∘ Difference from average-case analysis -
- Average case uses a probability distribution of inputs to compute expected costs.

- ammortized analysis makes no assumptions about input probability ; it analyzes the algorithm's behaviour over sequences of operations.
- Hence, amortized analysis ≠ average case analysis its a deterministic guarantee on long run performance not a probabilistic one.

Q2] What is the aggregate method in amortized analysis? Give a basic example.

→ Aggregate method -

- It is simplest technique for amortized analysis
- We take a sequence of n operations and compute the total actual cost for performing all these operations.
- Then we divide this total cost by 'n' to obtain a uniform cost per operation called the ammortized cost.
- This shows that even if some operations are expensive, the average per operation over the entire sequence stays low.
- The aggregate method gives a single bound for all operations in the sequence without assigning different credits to different operations.

- Example -

- Suppose we insert 'n' elements into a dynamic array that doubles its size when full.

- Most insertions take O(1) time, occasionally a resize takes O(n)
- Total cost for n insertions ≈ O(n) (because resizes happen at 1, 2, 4, 8... positions)
- Ammortized cost per insertion = O(n)/n = O(1)
- This shows each insertion has an amortized time of O(1) even though some single insertions take O(n) during resizing.

93] Evaluate the effectiveness of power optimized scheduling algorithms in embedded systems. What metrics would you use?

→ - Power optimized scheduling algorithms in embedded systems aim to reduce energy consumption while still meeting timing and performance constraints.
- Their effectiveness is judged by how well they balance power savings and real time deadlines.

- Metrics used include -

1. Total energy consumed during execution.
2. Average and peak power usage.
3. Execution time/ deadline miss rate to ensure real time compliance
4. Throughput (tasks completed per unit time)
5. CPU utilization and idle time (how efficiently low power states are used.)
6. Energy delay product (EDP) to combine energy, speed.
7. Battery life extension as a practical outcome.

- Thermal impact / temperature rise for reliability.
- Quality of service (QoS) maintained under power saving modes.

- A scheduler is effective if it minimizes power and energy while maintaining deadlines and acceptable performance.

UNIT - 06 -

Q4] Is multithreading parallel or concurrent? Analyze any multithreaded algorithm with example.

→ - Multithreading means multiple threads of execution within a single process.
- Concurrency vs. Parallelism -
  - on a single core CPU threads are interleaved (concurrent but not parallel).
  - on a multicore CPU, threads can run truly at the same time (parallel)
  - So multithreading always provides concurrency; true parallelism only if hardware supports it.

- Example - Multithreaded Merge sort.
  - Split array into two halves
  - Launch one thread to sort the left half, another to sort the right half.
  - On a multicore, system both sorts actually run in parallel and finish later.
  - On a single core system OS switches between threads giving concurrency but not speed up

- Thus we can say that multithreading expresses tasks as independent flows; hardware decides parallel vs concurrent execution.

Q5] Define naïve string matching algorithm. Apply the same to solve T = 1011101110 for P = 111. Find all the valid shift.

→ Naive string matching - Slide the pattern P along the text T one position at a time; at each shift compare P with the substring of T of the same length. If every character matches at a shift, that shift is a valid match.

- Time complexity worst case - $O(|T| \cdot |P|)$.

Given T = 1011101110, P = 111. No. of possible shifts = $|T| - |P| + 1 = 10 - 3 + 1 = 8$ (shifts 0 through 7, 0 based).

- Checking each shift -
  - Shift 0: T[0..02] = 101 → compare with 111 → no.
  - Shift 1: T[1..3] = 011 → 111 → no.
  - Shift 2: T[2..4] = 111 → 111 → match.
  - Shift 3: T[3..5] = 110 → 111 → no.
  - Shift 4: T[4..6] = 101 → 111 → no
  - Shift 5: T[5..7] = 011 → 111 → no
  - Shift 6: T[6..8] = 111 → 111 → match.
  - Shift 7: T[7..9] = 110 → 111 → no.

- All valid shifts - 0-based shifts : 2, 6.
- Hence, pattern 111 occurs in 1011101110 at shifts 2 and 6.

Q6] Define rabin Karp algorithm? given the string T = "abcabcabc" and pattern p = "abc", show how Rabin Karp algorithm works by computing the hash values.

→ ∘ Rabin Karp algorithm - a string search algorithm that uses hashing to find any one of a set of pattern strings in a text.

- It computes a hash value for pattern and for each substring of text of same length.
- If hash values match, then it compares the actual strings to confirm.
- Average complexity is O(n+m), worst case O(nm) when many hash collisions occur.

Given - Text T = "abcabcabc", Pattern P = "abc".

Step 1: Convert each letter to number (a=1, b=2, c=3).
Hash = sum of values., so for p = "abc". = 1+2+3 = 6

Step 2: Compute hash for each substring of length 3.
S1 = "abc": 1+2+3 = 6 → match.
S2 = "bca": 2+3+1 = 6 → hash match, but string is not equal to pattern
S3 = "cab": 3+1+2 = 6 → no match.
S4 = "abc": 1+2+3 = 6 → match.
S5 = "bca": 2+3+1 = 6 → no match
S6 = "cab": 3+1+2 = 6 → no match.
S7 = "abc": 1+2+3 = 6 → match.

Step 3: Positions 0, 3, 6 contain "abc", substrings 1, 4, 7 are a match.