

DAA UNIT – 5 (Amortized Analysis) – END-SEM PYQ Answers

► MAY / JUN 2022

Q5) a) Write short notes on the following. [10]

i) Aggregate Analysis

ii) Accounting Method

iii) Potential Function method

iv) Tractable and Non-tractable Problems

i) Aggregate Analysis

Aggregate analysis is a method in **amortized analysis** where we determine an upper bound on the **total running time** of a sequence of operations and divide it by the number of operations to obtain the **amortized cost per operation**.

Key ideas:

- Used when **some operations are cheap** and **some are expensive**, but overall performance is good.
- Instead of analyzing individual operations, we consider the **whole sequence**.

Example:

A dynamic array doubling in size upon overflow:

- Total cost of n insertions = **$O(n)$** even though doubling costs $O(n)$ occasionally.
- Amortized cost = **$O(1)$** per insertion.

ii) Accounting Method (Banker's Method)

The **Accounting Method** assigns an artificial cost (called **amortized cost**) to each operation.

This cost may be:

- **higher** than actual cost (deposit credit), or
- **lower** (withdraw using stored credit).

Idea:

We maintain a **balance** so that expensive operations get paid using credit saved from earlier cheap operations.

Example:

Dynamic array:

- Charge each push: **amortized cost = 3**
- Actual cost = 1, with surplus added as credit.
- During resizing, the saved credit pays for copying elements.

iii) Potential Function Method

Potential method is another amortized analysis technique where we define a **potential function Φ** over the data structure states.

The amortized cost is:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

where

- c_i = actual cost of operation i
- $\Phi(D_i)$ = potential after operation i

Idea:

Potential stores "energy" that pays for future expensive operations.

Example:

Stack operations where potential = number of elements gives $O(1)$ amortized push/pop.

iv) Tractable and Non-tractable Problems

Tractable Problems

Problems solvable in **polynomial time**, i.e., $O(n)$, $O(n^2)$, $O(n^3)$.

- Efficient and practical.
- Belong to class **P**.

Examples: shortest path, sorting, MST.

Non-tractable Problems

Problems requiring **super-polynomial** or **exponential time**, like $O(2^n)$, $O(n!)$.

- Often NP-hard/NP-complete.
- No known polynomial-time algorithms.

Examples: traveling salesman (TSP), knapsack, Hamiltonian cycle.

b) Write short notes on with suitable example of each [8]

i) Randomized algorithm

ii) Approximation algorithm

i) Randomized Algorithm

A randomized algorithm uses **random numbers** during execution, so its behavior and running time may vary.

Two types:

1. Las Vegas Algorithms:

- Always correct, time varies.
- Example: randomized quicksort.

2. Monte Carlo Algorithms:

- Time fixed, output may be incorrect with small probability.
- Example: randomized primality testing.

Example (Randomized Quicksort):

Pivot chosen randomly → expected time $O(n \log n)$.

ii) Approximation Algorithm

Approximation algorithms are used for **NP-hard optimization problems** where exact solutions are costly.

They produce solutions **close to optimal** with a performance guarantee.

Approximation Ratio:

If algorithm returns value A and optimum value = OPT:

$$\rho = \frac{A}{OPT}$$

Example:

Greedy algorithm for Vertex Cover gives 2-approximation

→ at most twice the optimal value.

Q6) a) What is Potential function method of amortized analysis? To illustrate Potential method, find amortized cost of PUSH, POP and MULTIPOP stack operations. [9]

Potential Method

We assign a potential Φ to each state of the data structure.

Amortized cost formula:

$$\hat{c} = c + \Phi_{\text{after}} - \Phi_{\text{before}}$$

Choose potential such that expensive operations have large negative potential change.

Stack Operations (PUSH, POP, MULTIPOP)

Let the potential = number of items in the stack:

$$\Phi = \text{stack size}$$

1. PUSH(x)

- Actual cost = 1
- Φ increases by 1

$$\hat{c} = 1 + (\Phi_{new} - \Phi_{old}) = 1 + 1 = 2$$

2. POP()

- Actual cost = 1
- Φ decreases by 1

$$\hat{c} = 1 + (-1) = 0$$

3. MULTIPOP(k)

MULTIPOP removes up to k items.

If r items are removed:

- Actual cost = r
- Φ decreases by r

$$\hat{c} = r + (-r) = 0$$

Conclusion using potential method:

Amortized cost of PUSH, POP, MULTIPOP is $O(1)$

b) What is embedded algorithm? Explain Embedded system scheduling using power optimized scheduling algorithm. [9]

1. Embedded Algorithm

An embedded algorithm is an algorithm designed to run on an **embedded system**, which is a:

- resource-limited,
- task-specific,
- real-time computing device (e.g., washing machine controller, automobile ECU).

These algorithms must satisfy:

- low memory usage,
- real-time deadlines,
- low power consumption.

2. Embedded System Scheduling

Embedded systems run multiple tasks and must schedule them so that:

- **real-time deadlines are met,**

- **energy consumption is minimized.**

Key Scheduling Approaches:

1. **Rate Monotonic Scheduling (RMS)** – fixed priority.
2. **Earliest Deadline First (EDF)** – dynamic priority.

3. Power-Optimized Scheduling Algorithm

Many embedded processors support **Dynamic Voltage and Frequency Scaling (DVFS)**.

Idea:

- Lower CPU frequency → less power → slower task.
- Higher CPU frequency → more power → faster task.

Power-Optimized Scheduling Steps:

1. Determine task deadlines and worst-case execution times.
2. Use EDF/RMS to decide execution order.
3. For each scheduled task:
 - Lower processor frequency *as long as* the task meets the deadline.
 - Run at minimal power mode.
4. If deadline becomes tight → increase frequency temporarily.

Outcome:

- **Significant reduction in energy consumption**
- **All deadlines still satisfied**

Example:

Task T1 requires 5 ms at full speed, deadline 20 ms.

We can slow CPU to 25% speed and still finish before deadline → saves power.

► MAY/JUNE 2023

Q5) a) What is amortized analysis? Explain aggregate and potential function methods used for amortized analysis with respect to stack operations?[9]

1. What is Amortized Analysis?

Amortized analysis determines the average running time per operation over a sequence of operations, even if some individual operations are very expensive.

It guarantees performance in the worst case over many operations, unlike:

- Worst-case analysis (single operation)

- Average-case (probabilistic)

Used for:

- Dynamic arrays
- Stacks (with MULTIPOP)
- Splay trees, Union–Find, etc.

2. Aggregate Method (with Stack Example)

Idea:

Compute total cost of a sequence of n operations.

Divide by n to get amortized cost per operation.

Stack Operations

Operations: PUSH, POP, MULTIPOP(k).

Let the stack initially be empty.

A sequence might look like:

PUSH, PUSH, PUSH, POP, POP, MULTIPOP

Actual Costs

- PUSH = 1
- POP = 1
- MULTIPOP(k) = min(k , current stack size)

Total cost reasoning

Each element is:

- pushed once
- popped once

So in any sequence of n operations:

- At most n PUSH operations
- At most n POP operations
- MULTIPOP operations pop already-pushed elements \rightarrow no extra cost beyond POPs

Total cost $\leq 2n$

Amortized cost

$$\text{Amortized cost} = \frac{2n}{n} = 2 = O(1)$$

Thus PUSH, POP, MULTIPOP all have $O(1)$ amortized cost.

3. Potential Function Method (with Stack Example)

Idea:

Assign energy value (“potential”) to system state.

Let potential:

$$\Phi = \text{number of elements in stack}$$

Amortized cost:

$$\hat{c} = c + \Delta\Phi = c + (\Phi_{\text{after}} - \Phi_{\text{before}})$$

Apply to Stack Operations

PUSH(x)

- Actual cost = 1
- Potential increases by 1

$$\hat{c} = 1 + 1 = 2$$

POP()

- Actual cost = 1
- Potential decreases by 1

$$\hat{c} = 1 - 1 = 0$$

MULTIPOP(k)

- Removes r elements
- Actual cost = r
- Potential decreases by r

$$\hat{c} = r - r = 0$$

Conclusion

All stack operations have $O(1)$ amortized cost

b) What is potential function method, of amortized analysis? To illustrate potential method, find amortized cost of PUSH, POP and MULTIPOP stack operations. [9]

Potential Method — Definition

- Define a potential function Φ on data structure states.
- Amortized cost:

$$\hat{c} = c + \Phi(D_i) - \Phi(D_{i-1})$$

Potential stores "credits" to pay for expensive operations.

Stack Example

Choose:

$$\Phi = \text{stack size}$$

PUSH:

$$1 + (1) = 2$$

POP:

$$1 + (-1) = 0$$

MULTIPOP(k):

If r pops:

$$r + (-r) = 0$$

Final: PUSH = 2, POP = 0, MULTIPOP = 0 (All $O(1)$)

Q6) a) Write short notes on the following. [10]

i) Aggregate analysis

iii) Potential function method

ii) Accounting Analysis

iv) Tractable and Non-tractable problems

i) Aggregate Analysis

- Find total cost of n operations.
- Divide by n \rightarrow amortized cost.
- Ensures total cost stays within linear or near-linear bounds.
- Example: Dynamic array expansion \rightarrow amortized $O(1)$ insert.

ii) Accounting Analysis (Banker's Method)

- Assign artificial amortized cost to each operation.
- Extra charge stored as *credit* to pay for future expensive ops.
- Ensures credit never goes negative.
- Example: charge 3 for each dynamic array insert \rightarrow extra credit pays for expansion.

iii) Potential Function Method

- Define a potential Φ to measure "stored work".
- Amortized cost = actual cost + change in potential.
- Flexible and powerful for data structures.
- Example: stack potential = number of items $\rightarrow O(1)$ amortized operations.

iv) Tractable vs Non-Tractable Problems

Tractable:

- Solvable in **polynomial time**: $O(n)$, $O(n^2)$, $O(n^3)$
- Efficient and practical
Examples: sorting, shortest path.

Non-Tractable:

- Require **super-polynomial** time: $O(2^n)$, $O(n!)$
- Typically NP-hard/NP-complete
Examples: TSP, 0/1 knapsack, Hamiltonian cycle.

b) Write short notes on with suitable example of each. [8]

i) Randomized algorithm

ii) Approximation algorithm

i) Randomized Algorithm

Algorithms that use random numbers during execution.
Output/time may vary.

Types:

- **Las Vegas**: always correct, time varies
Example: randomized quicksort ($O(n \log n)$ expected)
- **Monte Carlo**: fast but may be incorrect with low probability
Example: randomized primality testing

Benefits:

- Often simpler
- Good expected performance
- Avoids worst-case behaviour

ii) Approximation Algorithm

Used for NP-hard optimization problems where exact solution is too costly.

Features:

- Produces **near-optimal** solution
- Has a **performance guarantee** (approximation ratio)

$$\frac{A}{OPT} \leq \rho$$

Example: Vertex Cover greedy → 2-approximation

Travelling salesman MST-based heuristic → 2-approximation (metric TSP)

► **NOV/DEC 2023**

Q5) a) What is amortized analysis? Explain the aggregate method with example.[9]

What is Amortized Analysis?

Amortized analysis studies the average running time per operation over a sequence of operations, ensuring that the worst-case total time of n operations is still efficient.

Why it is needed:

Some operations are expensive (e.g., array resizing), but occur rarely. Amortized analysis spreads this high cost across many cheap operations.

It does not use probability; it works in worst-case overall.

Aggregate Method (Explanation + Example)

Definition

Aggregate method computes:

1. The total cost of n operations.
2. Divides by $n \rightarrow$ amortized cost per operation.

Example: Stack Operations (PUSH, POP, MULTIPOP)

Consider a stack supporting operations:

- PUSH(x)
- POP()
- MULTIPOP(k) \rightarrow pop up to k items

Actual costs

- PUSH = 1
- POP = 1
- MULTIPOP(k) = number of pops actually performed (\leq stack size)

Total cost of n operations

- Each element is pushed once and popped once.
- MULTIPOP does not add extra costs; it pops elements that were already pushed.

Thus, total cost of n operations $\leq 2n$

(Each item contributes at most 2: one push + one pop)

Amortized cost

$$\text{Amortized cost} = \frac{2n}{n} = 2 = O(1)$$

Conclusion: PUSH, POP, MULTIPOP all have amortized cost $O(1)$ using aggregate method

b) What is Potential function method of amortized analysis? To illustrate Potential method, find amortized cost of PUSH, POP and MULTIPOP stack operations. [9]

Potential Method — Definition

A potential function Φ is assigned to each state of the data structure.

Amortized cost of operation i :

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Purpose

- To store “energy” (credit) in potential for future expensive operations.

Apply Potential Method to Stack Operations

Choose potential

Φ = number of items in the stack

1. PUSH(x)

- Actual cost = 1
- Potential increases by 1

$$\hat{c} = 1 + (1) = 2$$

2. POP()

- Actual cost = 1
- Potential decreases by 1

$$\hat{c} = 1 + (-1) = 0$$

3. MULTIPOP(k)

If r items are popped:

- Actual cost = r
- Potential decreases by r

$$\hat{c} = r + (-r) = 0$$

Final conclusion: PUSH = 2, POP = 0, MULTIPOP = 0 \rightarrow O(1) amortized cost each

Q6) a) What are special needs of embedded algorithm? Which sorting algorithm is best for embedded systems? Why? [6]

Special Needs of Embedded Algorithms

Embedded systems have:

- Limited memory (KBs)

- Low processing power
- Real-time deadlines
- Low energy consumption
- Deterministic behavior (predictable timing)
- Small code size
- Low heat generation

Thus algorithms must be:

- Simple
- Predictable
- Low-overhead
- Non-recursive (preferably)
- Iterative and memory-efficient

Which sorting algorithm is best for embedded systems? Why?

Answer: Insertion Sort (or Selection Sort)

Insertion sort is most commonly used because:

- Runs efficiently for small inputs (common in embedded systems)
- $O(1)$ extra memory
- Predictable worst-case time
- Simple to implement
- Low code size
- No recursion
- Works well when data is almost sorted

Therefore: Insertion Sort is preferred for embedded systems

b) Explain Randomized and Approximate algorithms. [4]

Randomized Algorithm

An algorithm that uses random numbers during execution.

Features:

- Output or time may vary.
- Often avoids worst-case input.

- Good average performance.

Approximation Algorithm

Used for NP-hard optimization problems.

Provides a near-optimal solution with a known performance guarantee.

Example:

- Vertex Cover 2-approximation
- Metric TSP 2-approximation
- Knapsack greedy approximation

c) What is randomized algorithm? Give any example of randomized algorithm? Also explain Random variable, Binomial random variable and-Mathematics for Randomized algorithm. [8]

1. What is a Randomized Algorithm?

A randomized algorithm uses **random bits** during computation.

Types:

- **Las Vegas:** Always correct, time varies
- **Monte Carlo:** Fast, small error probability

Example: Randomized QuickSort

Pivot chosen randomly.

Expected time:

$$O(n \log n)$$

Worst case rarely occurs due to random pivot selection.

2. Random Variable

A random variable (RV) maps outcomes of an experiment to numbers.

Example:

Number of heads in 5 coin tosses.

3. Binomial Random Variable

A binomial RV counts the number of **successes** in n independent Bernoulli trials.

$$X \sim \text{Bin}(n, p)$$

Probability: $P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$

Example:

Toss a coin 10 times ($p = 1/2$).

X = number of heads.

4. Mathematics in Randomized Algorithms

Randomized algorithm analysis uses:

Linearity of Expectation:

$$E[X + Y] = E[X] + E[Y]$$

Expected Running Time:

If cost C_i occurs with probability P_i :

$$E[T] = \sum P_i \cdot C_i$$

Indicator Random Variables:

Used to simplify expectation analysis.

Example (Randomized QuickSort):

Expected comparisons =

$$2(n+1)H_n = O(n \log n)$$

► MAY/JUNE 2024

Q5) a) What are randomized algorithms? Enlist and explain in brief the primary reasons for using randomized algorithms. [8]

What are Randomized Algorithms?

A *randomized algorithm* is an algorithm that uses random numbers or random choices at one or more steps during execution.

Thus, its performance (time) or outcome may vary across executions.

Types:

1. Las Vegas algorithms

- Always produce *correct* results
- Running time varies (e.g., randomized quicksort)

2. Monte Carlo algorithms

- Output may be *incorrect* with small probability
- Fast execution (e.g., randomized primality test)

Primary Reasons for Using Randomized Algorithms

1. Simplicity

Randomization often simplifies logic:

- e.g., randomized quicksort is easier than balanced pivot selection.

2. Avoids Worst-Case Scenarios: Worst-case inputs become extremely unlikely.

- e.g., worst-case quicksort $O(n^2)$ becomes practically impossible.

3. Good Expected Performance: Expected time becomes fast and predictable:

$$E[T] = O(n \log n)$$

4. Helps in Breaking Symmetry: Useful in distributed systems, graph algorithms, and load balancing.

5. Useful for Large and Complex Input Spaces: Random sampling gives efficient approximate solutions.

6. Robustness: Performance stable even against adversarial inputs.

7. Sometimes Only Practical Option: Some complex problems (e.g., high-dimensional geometry, hashing) need randomness for efficiency.

b) What are approximation algorithms? Based on the approximation ratio, classify the approximation algorithms. [9]

What are Approximation Algorithms?

Approximation algorithms are algorithms for **NP-hard optimization problems** that run in polynomial time and produce solutions that are **close to optimal**.

They do **not** give exact solutions but guarantee:

$$\text{Approximation Ratio} = \frac{\text{Algorithm's Output}}{\text{Optimal Output}}$$

(for maximization; reversed for minimization)

Classification Based on Approximation Ratio

1. Constant-Factor Approximation

Guarantee solution within a fixed constant factor of optimum:

$$\frac{A}{OPT} \leq k \text{ or } A \geq \frac{OPT}{k}$$

Examples:

- 2-approximation for Vertex Cover
- 3-approximation for TSP (metric)

2. PTAS (Polynomial Time Approximation Scheme)

- For any $\epsilon > 0$, produces a $(1+\epsilon)$ -approximation
- Running time is polynomial for *fixed* ϵ
- Time may be $n^{1/\epsilon} \rightarrow$ not practical for small ϵ

Examples:

Knapsack PTAS

3. FPTAS (Fully Polynomial Time Approximation Scheme)

- $(1+\epsilon)$ -approximation
- Running time polynomial in both n and $1/\epsilon$

$$T = O\left(\frac{n^3}{\epsilon}\right)$$

Example:

Knapsack FPTAS

4. Logarithmic Approximation

Approximation ratio depends on $\log n$

Example:

Set Cover $\rightarrow O(\log n)$ -approximation

5. Randomized Approximation Algorithms

Use randomness; guarantee expected performance.

Example:

Randomized MAX-SAT $\rightarrow 3/4$ approximation

Q6) a) Explain the methods of amortized analysis. Give suitable example. [8]

1. Aggregate Method

- Compute total cost of n operations
- Divide by $n \rightarrow$ amortized cost

Example: Stack PUSH/POP/MULTIPOP

Total cost $\leq 2n$

Amortized cost = $O(1)$

2. Accounting Method (Banker's Method)

- Assign *amortized* cost to operations
- Save extra credit during cheap operations
- Use credit to pay for expensive operations

Example: Dynamic array insert \rightarrow charge 3 credits per insert \rightarrow saved credit pays for copying during expansion.

3. Potential Function Method

- Define potential $\Phi =$ “stored energy”
- Amortized cost:

$$\hat{c} = c + \Phi_{\text{after}} - \Phi_{\text{before}}$$

Example (Stack):

Potential = stack size

- PUSH → amortized cost 2
- POP → amortized cost 0
- MULTIPOP → amortized cost 0

All are $O(1)$ amortized.

b) Suppose you are working on an embedded system for a medical device that monitors patient vital signs. The device continuously collects data from various sensors and needs to process and display this information in real-time. The data includes timestamps, temperature readings, heart rate, and blood pressure measurements. Suggest suitable sorting algorithm for this scenario. Clearly justify your answer with respect to key factors.[9]

Scenario Requirements

The device collects vital signs continuously:

- timestamps
- temperature
- heart rate
- blood pressure

Needs:

- **Real-time processing**
- **Low memory usage**
- **Low energy consumption**
- **Predictable timing (deterministic)**
- **High reliability**
- **Small code footprint**

Best Sorting Algorithm: INSERTION SORT

Reasons (Key Factors):

1. Real-Time Constraints

Insertion sort is incremental:

- Can sort streaming data piece-by-piece
- No need to re-sort entire dataset
- Perfect for time-series sensor data

2. Low Memory Usage

Insertion sort uses:

$O(1)$ extra space

Ideal for embedded devices with KB-level RAM.

3. Predictable Worst-Case Time

Embedded systems need deterministic timing.

Quicksort has unpredictable worst-case.

Insertion sort's worst-case = $O(n^2)$, but:

- Data size is small
- Inputs are nearly sorted (sensor data arrives in order)

So actual time is close to $O(n)$.

4. Very Small Code Size

Fits easily into ROM/Flash memory of microcontrollers.

5. Works Best for Nearly Sorted Data

Vital-sign data arrives **chronologically**, so list is almost sorted.

Insertion sort becomes extremely fast for such inputs.

Therefore: Insertion Sort is the most suitable sorting algorithm for embedded medical devices

► NOV/DEC 2024

Q5) a) What are the advantages and disadvantages of : [8]

i) Aggregate Analysis

ii) Accounting Method

i) Aggregate Analysis

Advantages

1. **Very simple and intuitive**
– Just compute total cost over n operations.
2. **Gives tight amortized bounds**
– Works well when total number of expensive operations is bounded.
3. **Does not require potential functions or credit system**
– Straightforward for students and practical for simple structures.
4. **Best for uniform sequences**
– Especially when operations behave predictably (e.g., stack, dynamic array expansion).

Disadvantages

1. **Cannot analyze individual operations**
– Only analyzes *whole sequences*, not per-operation behavior.
2. **Fails when operations have highly varying costs**
– Example: binary counter with many different-cost increments is harder to analyze.
3. **Not suitable when potential must change dynamically**
– Cannot capture stored work or prepayment.
4. **Does not support different operation costs simultaneously**
– Cannot assign different amortized costs as accounting method can.

ii) Accounting Method**Advantages**

1. **Flexible**
– You can assign different amortized costs for different operations.
2. **Good for systems with occasional expensive operations**
– Credit stored by cheap operations pays for costly ones.
3. **Can handle varying operation types**
– E.g., PUSH, POP, MULTIPOP elegantly.
4. **More intuitive than potential method**
– Physically storing credits helps understand amortization.

Disadvantages

1. **Choosing amortized cost is tricky**
– If chosen incorrectly, the credit may become negative, giving an invalid analysis.
2. **Less powerful than potential method**
– Some problems (e.g., splay trees) require potential-based reasoning.
3. **Not suitable for deeply nested structures**
– Hard to assign consistent credits.
4. **Credits must remain non-negative**
– Adds mathematical restrictions compared to aggregate method.

b) What are approximation algorithms? Based on the approximation ratio, classify the approximation algorithms. [9]

What Are Approximation Algorithms?

Approximation algorithms are polynomial-time algorithms for **NP-hard optimization problems**, where finding the exact optimal solution is too expensive.

They return solutions **close to optimal** with a mathematically proven guarantee.

Approximation Ratio:

For minimization:

$$\frac{A}{OPT} \leq \rho$$

For maximization:

$$\frac{OPT}{A} \leq \rho$$

Classification Based on Approximation Ratio**1. Constant-Factor Approximation**

Achieves a fixed approximation ratio $\rho = c$.

Examples:

- Vertex Cover (2-approx)
- Metric TSP (2-approx)

2. Logarithmic Approximation

Approximation ratio:

$$O(\log n)$$

Example:

- Set Cover (ln n approximation)

3. Polynomial-Time Approximation Scheme (PTAS)

For any $\epsilon > 0$, produces a $(1 + \epsilon)$ -approximation.

Running time polynomial for fixed ϵ .

Example: Knapsack PTAS

4. Fully Polynomial-Time Approximation Scheme (FPTAS)

Running time polynomial in n and $1/\epsilon$.

More efficient than PTAS.

Example: Knapsack FPTAS

5. Randomized Approximation Algorithms

Uses randomness to achieve expected approximation guarantee.

Example: MAX-SAT (3/4-approximation expected)

Conclusion:

Approximation algorithms are classified as Constant, Logarithmic, PTAS, FPTAS, and Randomized.

Q6) a) Why potential function method cannot be used for analysing binary counter? Explain [8]**Binary Counter Problem**

A binary counter supports INCREMENT operation:

Example:

01111 → 10000 (many bits flip)

We want to analyze amortized number of bit flips per increment.

Why Potential Function Method Fails? (Key Reasons)**1. Potential must be non-negative**

Binary counter naturally wants a potential:

$$\Phi = \text{number of 1's}$$

But:

- After increment, number of 1's may drop massively
- Example: 11111 → 00000
 Φ drops from 5 to 0
 This creates large negative $\Delta\Phi$, often larger magnitude than the actual cost.

This violates condition:

$$\Phi \geq 0 \text{ always}$$

2. Cannot capture expensive ripple carries

When many consecutive 1s turn to 0s, large cost occurs:

- Actual cost may be k flips
- Potential drop may exceed actual cost
 This breaks the amortized cost formula:

$$\hat{c} = c + \Delta\Phi$$

3. Potential Method Works Only for "Stored Effort"

Binary counter flips many bits because earlier operations did not store enough potential.

No single potential function satisfies:

- Non-negative
- Captures heavy ripple flips
- Gives $O(1)$ amortized cost

Thus, the standard potential method fails.

Correct Method

Binary counter amortization is usually solved by aggregate method, not potential method.

b) Comment on the following statements : [9]

i) “The knapsack problem is NP-hard”

ii) “Boolean Satisfiability Problem (SAT) is NP-complete”

iii) “Minimum spanning tree is tractable problem”

i) “The Knapsack Problem is NP-hard” — Comment

0/1 Knapsack is NP-hard because:

- No polynomial-time algorithm is known
- Believed impossible unless $P = NP$
- Exponential algorithms exist only
- Many NP-hard problems reduce to knapsack

Thus, it is computationally intractable for large inputs.

ii) “Boolean Satisfiability Problem (SAT) is NP-complete” — Comment

SAT is:

- In NP (solution can be verified in polynomial time)
- NP-hard (every NP problem reduces to SAT)

SAT was the **first NP-complete problem** (Cook–Levin theorem).

If SAT were solved in polynomial time, **all NP problems become tractable**.

iii) “Minimum Spanning Tree is a tractable problem” — Comment

MST is **tractable** because:

- Solvable in polynomial time:
 - Kruskal: $O(E \log V)$
 - Prim: $O(E + V \log V)$
- Efficient even for very large graphs
- No exponential complexity

Thus $MST \in P$, so it is tractable.

NOTE: Please verify all answers before referring.