Amortized Analysis

→ in amortized analysis, we average the time required to perform a sequence of data structure operations over all the operations performed.

→ Amortized analysis helps to show that the average cost of an operation is small, if we average over a sequence of operations, even though a single operation within the sequence might to expensive.

→ Amortized analysis is different than average-case analysis

→ Amortized analysis guarantees the average performance of each operation in the worst case.

it has three most common techniques

1. Aggregate analysis
2. The accounting method
3. The potential method.

By performing amortized analysis we get insights into a particular Data structure which helps to optimize the ~~deg~~ design

1] Aggregate analysis :-

→ In this method we show

for all n

a sequence of n operations takes worst-case time $T(n)$ in total.

→ In the worst case, the average cost or amortized cost per operation is $\frac{T(n)}{n}$

→ this amortized cost applies to each operation, even when there are several types of operations in the sequence

Stack operations :-

Fundamental operations of stack takes $O(1)$ time

Push (S, x) pushes object x onto stack S.

Pop (S) pops the top of stack S & returns the popped object.

Calling pop on empty stack returns an Error.

each operation runs in O(1) time,
let's consider the cost of each to be 1.

total cost of a sequence of n Push & Pop operations is therefore n, and the actual running time for n operations is therefore $\Theta(n)$.

Lets take into consideration new operation

Mutipop (S, k) → k is positive

→ it removes the k top objects of stack S,

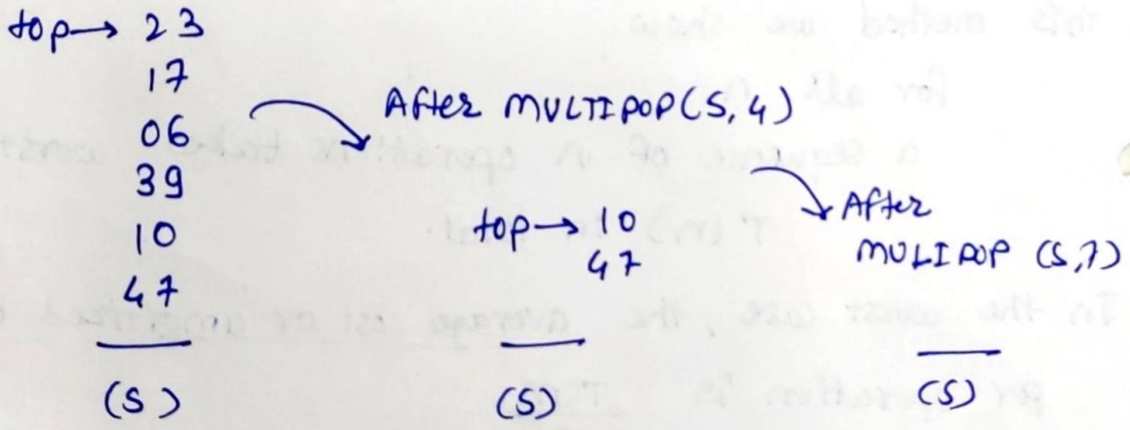
→ pops the entire stack if the stack contains fewer than k objects.

→ if k is not positive then multipop operation will not make any change in stack.

Consider the following pseudo code.

```
MULTIPOP (S, K)
1: while not STACK-EMPTY (S) and k>0
2:   POP (S)
3:   k = k-1
```

Example:



running time of MULTIPOP (S, k) on a stack of s objects.?

→ actual running time is linear in the number of pop operation actually executed.

→ thus MULTIPOP is analyzed in terms of the abstract costs of 1 each for PUSH & POP.

→ The number of iterations of the while loop is the number min (S, k) of objects popped off the stack.

iteration of the loop makes one call to POP

The total cost of MULTIPOP is min (s,k) and actual running time is a linear function of this cost.

→ for sequence of n PUSH, POP and MULTIPOP operations on an initially empty stack.

Worst-case MULTIPOP is O(n) ∴ size of stack is n.

Worst case time of any stack operation is O(n) and hence a sequence of n operations costs $O(n^2)$ since, O(n) MULTIPOP operations costing O(n) each we can have.

But this is not right

Using aggregate analysis

Better upper bound analysis can be obtained by considering the entire sequence of n operations.

→ though MULTIPOP operation can be expensive any sequence of n PUSH, POP & MULTIPOP operations on an initially empty stack is at most O(n)

→ Using aggregate analysis the number of times that POP can be called on a nonempty stack, including calls within MULTIPOP, is at most the number of PUSH operations, it is at most n.

→ for any value of n, any sequence of n PUSH
n POP
n MULTIPOP

takes a total of O(n) time.

The average cost of an operation is

$$\frac{O(n)}{n} = O(1)$$

Accounting method

In the accounting method

→ we assign differing charges to different operations

→ some operations charged more or less than they actually cost.

→ the amount charged for an operation ~~it~~ is amortized cost.

→ when operation's amortized cost exceeds its actual cost we assign the difference to specific objects in the data structure as credit

→ Credit can help pay for later operations whose amortized cost is less than their actual cost.