Amortized Analysis:

- If there are a series of operations where the cost of a single operation is very large & the rest of the operations cost less, worst case running asymptotic time complexity may not give a tighter bound.

E.g.

A

$= 500 \times 501$

$= ₹\ 2,50,000$

Asymptotic

$500 \rightarrow ₹1$ each

$1 \rightarrow ₹500$

B

$= 500 \times 1 + 1 \times 500$

$= ₹\ 1000$

Amortized

- In amortized analysis, your average or worst case measures is not input dependent.

Augmented Stack

push - O(1)

pop - O(1)

multi-pop(K) = { false, if K > n
true, otherwise

n → no. of items to be popped

pops multiple items

No. of items in stack

pop() = multi-pop(1)

- multi-pop(k) (where k == n)
  ⇒ O(n)

TC for n-operations $O(n) \times n = O(n^2)$

Consider a stack completely filled
1st operation. multipop (n) → TC → O(n)
Now we can't perform multipop (n) again,
we need to do n-pushes first → TC → $n \times O(1) = O(n)$

$\therefore$ TC = $O(n) + O(n) = 2(O(n)) \simeq O(n)$

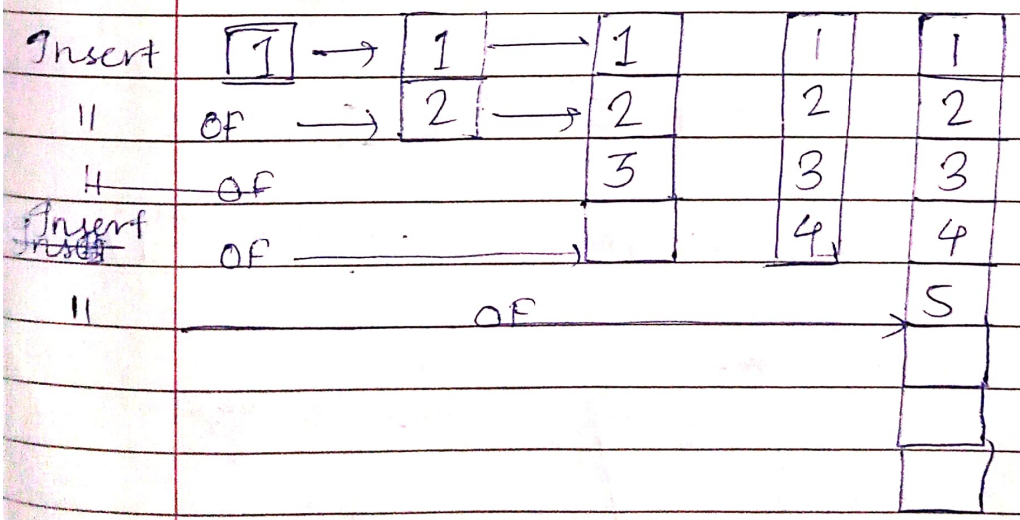
Aggregate method

→ Simplest method in amortized analysis

Amortized cost per operation = $\frac{\text{Total cost for all operations}}{\text{No. of operations}}$

E.g. Dynamic Tables
Everytime there is a overflow the size of the dynamic array doubles.

Insert



PAGE NO.:

Asymtotic TC:

i (No. of insert.)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
size	1	2	4	4	8	8	8	8	16	16	16	16	16	16	16	16
cost	1	2	3	1	5	1	1	1	9	1	1	1	1	1	1	1

first you need to copy 2 previous elements & then enter the new one.

you only need to insert a new-element

Cost of ith insertion = { i, where i-1 is exact power of 2
                       { 1, otherwise

Cost of n-insertions = $\sum_{i=1}^{n}$ cost i $\left[2^0 + 2^1 + \cdots + 2^{\log(n-1)}\right]$

$= n + \sum_{j=0}^{\log(n-1)} 2^j$ $\quad \frac{a(r^n-1)}{r-1}$

$= n + 2(n-1)$ $\quad = 1\left(\frac{2^{\log(n-1)+1}-1}{1}\right)$

$= 3n - 2$ $\quad = 2^{\log_2(n-1) + \log_2 2}$

$= O(n)$ $\quad = 2^{\log_2 2(n-1)}$

$= 2(n-1)$

Cost of single insertion = $\frac{O(n)}{n} = O(1)$

Accounting method

- Assign a random cost / amortized cost $c_i\hat{}$ / to the i^{th} operation, where 1 unit of amount pays for 1 unit of work
- If actual cost of operation (c_i) is less than the allocated amount $(c_i\hat{})$, remaining amount is saved & can be used later for costlier operations

Augmented Stack:

	$c_i\hat{}$	c_i
push()	2	O(1)
pop()	0	O(1)
multipop()	0	O(1)

Created a stack & pushed 4 elements

2	-1
9	-1
6	-1
3	-1

balance = (2-1) + (2-1) + (2-1) + (2-1)

= 4 units

- The bank balance must never be negative

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c_i}$$, for all values of n.

actual cost incurred

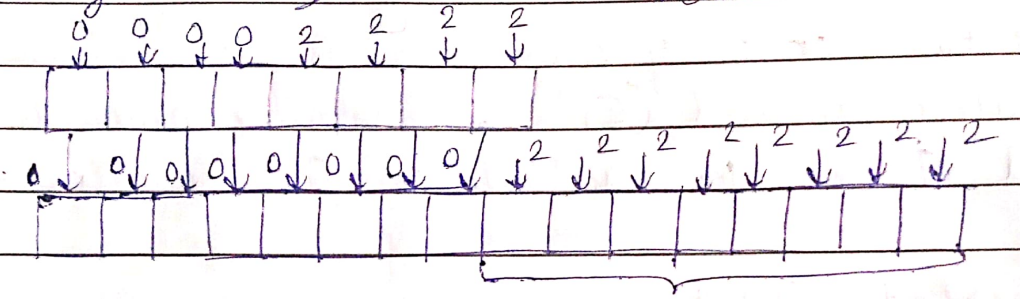
Amortized cost assigned

Dynamic Tables

Required = 1
c_i = 3 units

Original size = 4

balance = 8 units



8 units balance is used to copy the 8 elements

16 units of time would be used for 17th insertion

TC of n insertions = $3 \times n = O(3n) = O(n)$

Potential Method

→ At any given point, there is a potential in the data structure (DS)

→ Initial potential of the DS = D_0

→ ~~Act~~ Actual cost of i^{th} operation $\Rightarrow c_i$

Amortized - " - " ——— " ——— $\Rightarrow \hat{c_i}$

→ Define a potential func^n $\Phi : \{D_i\} \rightarrow R$

$\Phi(D_0) = 0$ & $\Phi(D_i) \geq 0$, for all i

Amortized cost = Actual cost + Change in potential

$$\hat{c_i} = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\Delta\Phi_i}$$

If $\Delta\Phi_i > 0$

$\hat{c_i} > c_i$ → ~~undercharged~~ overcharged

else $\hat{c_i} < c_i$ → undercharged

Augmented Stack:

$\Phi(D_i)$ = No. of elements in stack

push = $1 + (x+1) - x$

$\hat{c_i} = 2$

~~pop~~ pop : $1 + (n-1) - n = 0$

$$\boxed{\hat{c_i} = 0}$$

Multipop (k): $\hat{c_i} = k + (n-k) - n = 0$

$$\boxed{\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})}$$

Dynamic Table / Table ~~doubl~~ doubled

$$c_i = \begin{cases} i, \text{ where } i-1 \text{ is an exact power of 2} - C1 \\ 1, \text{ otherwise} - C2 \end{cases}$$

$$\boxed{\phi(D_i) = 2i - 2^{\lceil \log_2 i \rceil}} \qquad \boxed{\hat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})}$$

C1

$\hat{c_i} = i + 2i - 2^{\lceil \log_2 i \rceil} - (2(i-1) - 2^{\lceil \log_2 (i-1) \rceil})$

$= i + 2 - 2^{\lceil \log_2 i \rceil} + 2^{\lceil \log_2 (i-1) \rceil}$

e.g. $i = 9$, $i-1 = 8$

$= 2^{\lceil \log_2 9 \rceil} = 2^{\lceil 3 \ldots \rceil} = 2^4 = 16$

$= 2(i-1)$

$= 2^{\lceil \log_2 (i-1) \rceil} = 2^{\lceil \log_2 8 \rceil} = 2^3$

$= (i+1)$

C2

$\hat{c_i} = 1 + 2i - 2^{\lceil \log_2 i \rceil} - (2(i-1) - 2^{\lceil \log_2 (i-1) \rceil})$

$= 1 + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log (i-1) \rceil}$

= when $(i-1)$ is not an exact power of 2

$$\boxed{2^{\lceil \log i \rceil} = 2^{\lceil \log (i-1) \rceil}}$$

$\hat{c_i} = 3$

$= i + 2 - 2(i-1) + (i-1)$

$= i + 2 - (i-1)$

$= 3$

$\boxed{C_i\hat{} = 3}$

Potential method cannot be used for all cases, even though it gives a tighter bound for many cases