1] $T(n) = 64\,T(n/8) - n^2 \log_2 n$

2] $T(n) = \sqrt{2}\,T(n/2) + \log_2 n$

1] $a = 64 \quad b = 8 \quad f(n) = -n^2 \log_2 n$

$= n^2 \log_2 \frac{1}{n}$

CANNOT BE SOLVED

2] $T(n) = \sqrt{2}\,T(n/2) + \log_2 n$

$a = \sqrt{2} \quad b = 2 \quad f(n) = \log_2 n$

$\therefore n^{\log_b a} = n^{\log_2 \sqrt{2}} = n^{0.5}$

$\therefore f(n) < n^{\log_b a}$

$T(n) = O(n^{\log_b a}) = O(n^{\log_b a - \epsilon}) = O($

$\boxed{\text{Amortized ANALYSIS}}$ → Related sequence ca

```
K=1;
func(int n)
{  if (K is even)
       O(n); //expensive
   else
       O(1); //cheap
   K=K+1;
}
```

2 call

$O(1)$ K=1 func(10

$O(n)$ K=2 func(10

For m calls Normally $O(m*n)$

For Amortized $O(m/2\ n) + O(m/$

$\Rightarrow \frac{\frac{m}{2} O(1) + \frac{m}{2} O(n)}{m} = \frac{O(1) + O(}{2}$

Scanned by CamScanner

Amortized Analysis ~~is~~ – Spreading out big cost over a period of time.

It is applied on Datastructures that support many operations. Asymptotic analysis gives worst case analysis of each operation without taking effect of one operation on another, whereas amortized analysis focusses on sequence of operation & interplay between operations & thus it yields an analysis which is precise and microlevel analysis.

. This analysis is applied where an occasional operation is very slow (expensive) but other operations are faster (cheap).

Amortized Analysis is an upper bound & is average performance of each operation in the worst case. This analysis ~~comp~~ concerned with overall cost of sequence of operations doesn't say anything about cost of specific operation in that sequence.

This analysis is not asymptotic average time analysis because here probability distribution of input data is not assumed.

Aggregate Analysis.

It is simplest way of analysis.

Compute the total cost for sequence of n operations i.e. $T(n)$

Avg cost of Amortized cost is given as :-

$$(\hat{c}) = \frac{T(n)}{n}$$

The limitation of this method is that s

shot on moto G5s Plus

Scanned by CamScanner

amortized cost is assigned for different of operations, whereas other two methods give precise analysis i.e. accounting m & potential method.

Multipop(Stack s, int k)
{ while (not StackEmpty(S) & k≠0)

$x = pop(s);$
// do something with x.
k--;
}

For Multipop Asymptotic Analysis ⇒ $T(n) = O($

For sequence of n operations complexity turns out to be $O(n^2)$
This is not a precise value because no. of times pop is called is equal to no. of times items are pushed into stack.
Let there be n operations in sequence
Let l be no. of multipop operations

∴ $(n-l)$ no. of push & pop operations

$$\text{Avg cost} = \frac{2\,O(n)}{n} = 2 \simeq O(1)$$

For n no of operations total cost
$O(n) + O(n) = 2\,O(n)$
∵ (At most n elements can be popped)

Time Reqd to perform sequence of data structure operation is avg over all the operations performed.
Avg cost of an operation is small even though That operation might be expensive.

Ex. 2. For Aggregate Analysis

Consider a binary counter of 8 bits
Analyse Amortized cost of sequence of n operations where counter is set to 0.

```
increment (a, k)
{
    i = 0 ;
    while (i < k && a[i] ≠ 0)
    {
        a[i] = 0;
        i++ ;
    }
    if (i < k)
        a[i] = 1 ;
}
```

0010

~~0100~~ 0011

0100

$T(n) = O(k)$

Scanned by CamScanner

Binary Counter

Analyse Amortised cost of n incre operations.

a[0] flips $n/2^0$ times
a[1] flips $n/2^1$ times
a[2] flips $n/2^2$ times

a[i] flips $n/2^i$ times

For Asymptotic Analysis,

$$T(n) = O(nk)$$

No of operations — No of bits

For Amortized Analysis

$$\text{Total Cost} = \sum_{i=0}^{\lfloor \log_2 n - 1 \rfloor} n/2^i$$

$$= n \sum_{i=0}^{\log_2 n - 1} 1/2^i$$

$$= 2n$$

$$\cong O(n)$$

$$\text{Aggregate cost} = \frac{\text{Total Cost}}{n}$$

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

Scanned by CamScanner

Imp Dynamic Table

This table grows as we insert the element into the array. If there are no empty cells left at the end of table then new table of double size is created & all the data from old table is copied to new table & newly inserted element is appendended at end.

```
insertTable (T, x)
{
    if (T.size == 0)
    {
        Create Table of size 1 &
        T[0] = x
        exit
    }
    ~~else if (Loadfactor =~~
    {
    Load factor = T.num / T.size
    if (Loadfactor == 100%)
    {
        Allocate newTable of size 2*T.size
        Copy all elements of Table T
        to new Table
        T = newTable
        T.size = newTable.size
    }
        T[T.num] = x ;
        T.num++ ;
    }
}
```

Asymptotic Time Complexity = $O(n^2)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Size	1	2	4	4	8	8	8	8	16	16	16	16	16	16	16	16	32
Cost	1	2	3	1	5	1	1	1	9	1	1	1	1	1	1	1	17
Insert cost	~~1~~	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	[illegible]
Instruction	0	1	2	0	0	0	0	0	8	0	0	0	0	0	0	0	[illegible]

$c_i = i$ if $(i-1)$ is perfect pow[er]
$= 1$ else

$d_i = i-1$ if $(i-1)$ is perfect pow[er]
$= 0$ else

$$\sum_{i=1}^{n} cost_i = \sum_{i=1}^{n} cost_i + \sum_{i=0}^{\log_2 n-1} 2^i$$

$$= n + 2n$$

$$= 3n$$

$$\simeq O(n)$$

Aggregate cost $= \frac{3n}{n} = 3 \simeq O($

Accounting Method

Here we maintain an account wi[th] underlined data structure. Initia[lly] it will have zero balance (zero credits). We assign different charges [to] different operation with some operat[ions] charge more or less than their actual cost. The cost assigned by us is calle[d] amortized operation (cost). When the amorti[zed] cost is greater than actual cost of operation then there will be a balance (credit). That credit we can use later to pay for operations having amortized cost lesser than actual cost.

Let c_i be the actual time of operatio[n]

Let $\hat{c_i}$ be amortized cost of i^{th} ope[ration]

If $\hat{c_i} > c_i$ then there will be a credit

shot on moto G5S Plus

Scanned by CamScanner

$$\boxed{\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i}$$

$\therefore$ Credit is always positive

Low cost operations are charged little more than their actual cost & surplus is deposited in bank account whereas high cost operations are charged less than their true cost & the deficit is paid by savings in the bank account.

Note:- The charge through each operation must be safe large enough that the balance in the bank account always remains positive but small enough so that no operation can charge significantly more than its actual cost.

- In stack push operation takes 2 units in amortized analysis so out of which 1 unit is used for actual push operation & 1 unit goes as credit & is used in pop & multipop later.

- For Binary counter we are assigning 2 unit cost to each bit flipping when it is flipping from 0 to 1.

$\hat{c}_{0\to 1} = 2 \Rightarrow O(1)$

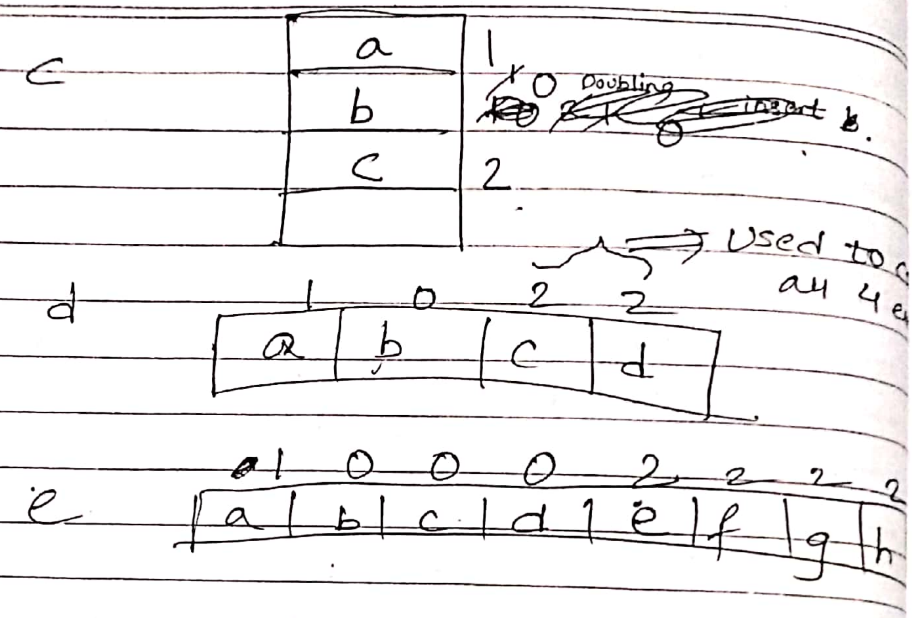
$\hat{c}_{1\to 0} = 0 \Rightarrow O(1)$

- Dynamic Table :-

Insert a $\boxed{a}$ $2(3-1(\text{For insert})) = 2$

Insert b | a | 1 |
| b | 2 |

120



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1
$size_i$	1	2	4	4	8	8	8	8	16	16	16	16	16	16	16	1
Actual cost	1	2	3	1	5	1	1	1	9	1	1	1	1	1	1	1
$cost_i$	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Balance	2	3	3	5	3	5	7	9	3	5	7	9	11	13	15	1

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

$\hat{c}_i > c_i$ Surplus Amt - Credit

$\hat{c}_i < c_i$ ... Deficit paid by credit

$\boxed{\text{Potential Method}}$

$\phi(D_i) \rightarrow$ Change in structural parameter of D after i^{th} operation

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) \quad \text{—(1}$$

Actual cost

Change in potential

shot on moto G5S Plus

Scanned by CamScanner

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} [\phi(D_i) - \phi(D_{i-1})]$$

$$\therefore \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0)$$

Stack

↓

Structural parameter. no of elements.

$$\hat{c}_{push} = c_{push} + [\phi(D_i) - \phi(D_{i-1})]$$

Assume before i^{th} push 'x'

$$\therefore \hat{c}_{push} = 1 + [(x+1) - x]$$

$$= 1 + x + 1 - x = 2 = O(1)$$

$$\hat{c}_{push} = c_{pop} + [\phi(D_i) - \phi(D_{i-1})]$$

$$= c_{pop} + ((x-1) - x)$$

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

$$\hat{c}_{multipop} = c_{multipop} + [\phi(D_i) - \phi(i-1)]$$

$$= k + [(x-k) - x]$$

$$= 0 = O(1)$$

Counter :-

To replace '0'

No. of bit changes = $(\underbrace{x}_{\text{Total no of 1's}} - \underbrace{t}_{\text{Total 1's before '0'}} + 1)$

Structural parameter is no of 1's in pa

shot on moto G5S Plus

Scanned by CamScanner

Let x be no of 1's before i^{th} ope
let t be no of 1's before last
So after i^{th} operation,

$(x - t + 1)$ no of 1's in coun

$$\hat{C}_{increment} = C_{increment} + [\phi(D_i) -$$
$$= (t+1) + [(x-t+1) - x]$$
$$= 2 = O(1)$$

Dynamic Table

Amortized cost = Actual cost + Change in potential

$$\phi(T) = 2 * Num(T) - size(T)$$

Initially No of Element = 0
Size of Table = 0
Therefore $\phi(T) = 0$

There are 2 possible cases :-

Case 1 :- i^{th} insertion does not trigger an expansion

$$\hat{C}_{insertion} = C_{insertion} + (\phi_i - \phi_{i-1})$$

$$\phi_i = 2N_i - S_i$$
$$\phi_{i-1} = 2(N_{i-1}) - (S_{i-1})$$
$$= 2(N_i - 1) - (S_i - 1) \rightarrow S_i$$
$$= 2(N_i - 1) - (S_i)$$
$$= 1 + (\cancel{2N_i} - \cancel{S_i} - \cancel{2N_i} + 2$$
$$= 3$$

shot on moto G5S Plus

Scanned by CamScanner

Case 2 :- ith instruction triggers an expansion

$$\hat{C}_{insertion} = C_{insertion} + (\phi_i - \phi_{i-1})$$

$$\phi_i = 2N_i - S_i$$

$$\phi_{i-1} = 2(N_{i-1}) - S_{i-1}$$

$$= 2(N_i - 1) - \frac{S_i}{2}$$

$N_i \Rightarrow$ No of elements in Table

$$\hat{C}_{insertion} = N_i + \left(2N_i - S_i - 2N_i + 2 + \frac{S_i}{2}\right)$$

$$= N_i - \frac{S_i}{2} + 2$$

But $N_i = \frac{S_i}{2} + 1 \Rightarrow$ No of elements to be copied suppose on doubling size $= 16$ for 9^{th} element insertion No of elements to insert $= \frac{16}{2} + 1 = 8 + 1 = 9$ elements

$$\hat{C}_{insertion} = 3 = O(1)$$