AMORTISED TIME COMPLEXITY

Analysis of Amortized Time Complexity for Push Operation

Amortised cost = $\frac{total\ cost\ for\ all\ operations}{number\ of\ operations}$

The amortized cost is calculated as the total cost for all operations divided by the number of operations. In this case, the logic is that if no overflow occurs, the element is inserted as usual. However, if an overflow occurs, a new array of double the size is created, and make a copy of elements from the old array, and the new element is inserted thereby deleting the old array.

Initially, let the size of the array be 1. The first element can be inserted in O(1) time. However, when adding the second element, an overflow occurs and as a result, doubling the array and inserting causes O(2) time. Now, when adding the third element, a stack overflow happens again; the size becomes 4 (double), and inserting an element happens in O(3) time.

Therefore, the cost of the i^{th} iteration is equal to i if (i-1) power of 2. If it is not a power of 2, then the cost of the i^{th} iteration becomes 1.

Let i represent the iteration number, or it can be considered as the number of elements in the array. The cost incurred during the i^{th} iteration is determined by the following expression:

$$\text{Cost(i)} = \begin{cases} i, & if\ i-1\ is\ power\ of\ 2 \\ 1, & otherwise \end{cases}$$

Let K be the number of overflows. Then, we have:

⇨ $i-1 = 2^K$

⇨ $K = log_2(i-1)$

i^{th} operation	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Capacity of the Array	1	2	4	4	8	8	8	8	16	16	16	16	16	16	16	16	32
Cost	1	$1+2^0$	$1+2^1$	1	$1+2^2$	1	1	1	$1+2^3$	1	1	1	1	1	1	1	$1+2^4$

Cost of n iterations = $n + \sum_{i=0}^{[log_2(n-1)]} 2^i$ [.] indicates Greatest Integer function

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

$= 3n - 3$

⇨ O(3n-3) ~ O(n);

So, the number of push operations is n, and the amortized time is $\frac{O(N)}{N}$, which simplifies to O(1), a constant time for each push operation.

Analysis of Amortized Time Complexity for Pop Operation

When analyzing the pop operation, we observe that if the number of elements becomes equal to the (capacity/4 + 1), we create a new array with half the capacity and copy all elements, excluding the last one, which takes O(n-1) time. Otherwise, it's just removing an element, which takes O(1) time.

$$\text{Cost(i)} = \begin{cases} i-1 & if\ i-1\ is\ equal\ to\ \frac{capcity}{4} \\ 1, & otherwise \end{cases}$$

i^{th} operation	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Capacity of the Array	1	2	4	8	8	16	16	16	16	32	32	32	32	32	32	32	32	32
Cost	1	2^0	2^1	1	2^2	1	1	1	2^3	1	1	1	1	1	1	1	1	1

Therefore the number of multi pop operations will be K = $\lfloor \log_2(N-1) - 1 \rfloor$

Here n is the current size of the array which is being filled

Cost of n iterations = N-K + $\sum_{i=0}^{K} 2^i$

= N-K+N-2

= 2N-2+K

⇨ O(2N-2+K) ~ O(N)

This results in an amortized time of O(2N-2+K), which is equivalent to O(N), where N is the number of elements in the current array. Therefore, the amortized time for each pop operation is O (1), which is a constant.

Therefore the amortized time is $\frac{\mathrm{O(N)}}{N}$ = O(1) is constant.