# Fancy Stack

In this chapter, we discuss the complexity of various operations of a stack which supports a multipop operation.

We are all aware of the push and pop operations offered by a stack data structure. Both push and pop are trivial and take O(1) or constant time. Now imagine the stack also offers a multipop operation, which allows us to pop $k$ elements at once, assuming the stack holds atleast k elements. If we conduct $n$ operations on the stack, what is the average time complexity of each of the $n$ operations?

## Naive Analysis

Consider a sequence of $n$ pop, push, and multipop operations. The maximum number of elements that can accumulate in the stack is also $n$ since the sequence can consist of all push operations. In the worst case, the multipop operation would take O(n) time to pop all the n elements of a stack. So, in general, the time complexity for execution of $n$ operations in the worst case will be:

$$n * O(n) = O(n^2)$$

Since the $n$ operations can consist of all multipop operations, each multipop takes O(n) time in the worst case. Though this analysis is correct, the bound isn't tight enough.

## Aggregate Analysis

Intuitively, if at most $n$ elements have been pushed onto the stack then it is possible to pop them back out only once. The multipop operation internally makes repeated calls to pop operation, but pop can only be called at most $n$ times in a sequence of $n$ operations. Thus the **total** cost of n operations can be n, at most. Note that we say there can be at most $n$ pop operations. However,

since we start with an empty stack any legal sequence of $n$ operations can consist of at most $n/2$ pop operations, as an equal number of push operations need to be executed to make the stack non-empty. Since each individual operation has cost O(1) whether it be push or pop, n operations will have cost O(n). The amortized cost for each operation then becomes:

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

### Accounting Analysis

We'll use the accounting method to analyze the fancy stack. The actual costs for the three operations are:

**Push : 1**
**Pop : 1**
**MultiPop : k - assuming stack has atleast k elements.**

As a first step, we'll need to assign amortized costs to each stack operation. Let's choose the following amortized costs for the stack operations:

**Push : 2**
**Pop : 0**
**MultiPop : 0**

Now imagine that whenever we push an item on the stack, we pay 1 rupee as the cost for the push operation and we place an additional rupee as credit with the object we are pushing on the stack. When the time comes to pop the object, the cost will be paid by the extra rupee that we stored alongside the object when it was pushed onto the stack.

Now it's easy to reason that if we have a sequence of $n$ pop, push, or multipop operations, the amortized cost will be **2*n = O(n)** in the worst case. The amortized cost is an upperbound on the actual cost, and we can thus declare that the cost of n operations will be O(n).

### Changing Cost Assignment

A reader may wonder if the amortized cost assignments can be changed. For instance, will the following values work?

**Push : 0**
**Pop : 2**
**MultiPop : 0**

While undertaking accounting method of analysis, it is imperative that ***the total amortized cost for any sequence or subsequence of operations shouldn't be negative or less than the actual cost***. If the sequence only consists of push operations, then the total amortized cost will turn out to be less than the actual cost, thus we can't use the suggested assignments.

Q

If we introduced a multipush(k) operation for our fancy stack, how would that affect the cost of stack operations?

Check Answers