Consider a sequence of operations $O_1, O_2, \ldots, O_m$ on a given data structure $D$ where we are interested to bound the total cost $T$ of the $m$ operations. This can be done by bounding the worst case cost of any operation, say times $m$. 

But it is possible that we do not encounter the cost times very frequently, i.e., we may be able to get a superior bound.

Example 1: Suppose $D$ is a stack and $O_i \in \{\text{push, pop, empty stack}\}$.

Cost of push, pop in $O(1)$ and cost of empty stack = # elements on stack.
Worst case cost \( \text{Empty Stack} \in O(m) \)

\[
= \text{Total cost for a sequence of } m \text{ operations in } O(m \cdot m) = O(m^2)
\]

**Note:** For special cases like the stack not growing beyond say constant size, total cost \( \leq O(c \cdot m) = O(m) \)

**Observation:** The worst case bound \( O(m) \) cannot happen very often.

In particular, if we encounter a cost \( K \) for empty stack \( \Rightarrow \) there were at least \( K \) push operations after the previous empty stack call.

Push, Pop, push for \( E \) at push point \( 3.5 \)

\[
\geq K \text{ push operations}
\]

\( i.e. \) the average cost: \( O(k) \times O(1) + O(k) \), \( ~ O(1) \)

\( \frac{1}{K} \)
\# pops \leq \# push ops

Every stack can be written as a sequence of pop ops:

```
Push pop push . . ES
```

Averaging arguments is known as amortized analysis.

A general technique for amortized analysis:

We define a potential function \( \phi : D \to \mathbb{R} \)

The amortized cost of a single operation:

\[ \text{actual cost} + \Delta \phi \; \text{change in potential} \]

\( \phi \) is potential after

\( \phi \) is potential before

\( \phi_1 \) is initial potential \( \phi_0 \)

\[ \text{Total amortized cost} = w_1 + (\phi_1 - \phi_0) + w_2 + (\phi_2 - \phi_1) + \ldots + w_m + \phi_f - \phi_i \]

\[ = \sum w_i + \phi_f - \phi_0 \]
Total amortized cost: Total actual cost + $\phi_f - \phi_o$.

In particular, if $\phi_f - \phi_o > 0$

$\Rightarrow$ Amortized cost > Actual cost

Example: Stacks

$\phi(\text{stack}) = \# \text{ elements in stack}$

Amortized cost of push: $1 + 1 = 2$

Amortized cost of pop: $1 + (-1) = 0$

Amortized cost of empty stack = 0

Total amortized cost of $m$ ops on stack

= Worst-case amortized cost of a single op $\times m$

= $2 \times m = O(m)$

Note: $\phi_f - \phi_o \geq 0$  ($\phi_o = 0$  $\phi_f - \phi_o > 0$)

Suppose we did not begin with an empty stack
Ex 2: Counters: A counter starting from 0 that can count till $2^n$ can be represented by $n$ bit.

3 bit counter:

- The cost of incrementing a counter: # bit flips

Total cost for the counter to go from 0 to $2^n \leq n \cdot 2^n$

Observation: The $i$th bit flips $\frac{2^n}{2^i}$ times, $0 \leq i \leq n-1$

Total # bit flipped: $\sum_{i=0}^{n-1} \frac{2^n}{2^i} \leq [2] \cdot 2^n$
\( \Phi \) (counter) = \# bits with value 1

Amortized cost of incrementing the counter

\( = \# \) bits flipped + \( \left( \# 1s \mod \text{ cand}(i+1) \right) - \# 1s \mod \text{ cand } i \)

when going from \( i \rightarrow i+1 \)

\[ \begin{array}{cccccccc}
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{array} \]

For a block of \( k \) 1's actual cost: \( k+1 \)

change in potential: \[ \frac{-(k-1)}{2} \]

Amortized cost of increment: \( 2 \)

Alternative for BST (without deletion)

Operation: search, insert

Semidynamic dictionary
Arrays:

Given $n$ elements, we store them in a set of arrays that can accommodate $2^i$ elements and we have no more than one for each $i$.

$n = 6 ightarrow A_2 : 2^2 \{0, 3, 7, 9\} \\
\quad \rightarrow A_1 : 2^1 \{1, 8\} \\
\quad A_0 : 0$

Within each array, we keep the elements sorted, but there is no relation between the elements of two distinct arrays.

Search?

Insert?

- Logarithmic search once for each of the log arrays

- $= O(\log^2 n)$