Amortized Analysis: 3 Methods

- **Aggregate**
  - n ops take time T(n)
  - Avg cost per op = T(n)/n (Amortized Cost)
  - Imprecise - no separate cost for each item

- **Accounting**
  - (Book-Keeping scheme for counting ops)
  - Charge each op an amortized cost
  - Store in “bank” amount not used
  - Later ops can use “banked” work

- **Potential**
  - “Stored work” viewed as “potential energy”
  - Most flexible & powerful point of view

Stack Example

- Consider a stack data structure with 3 ops
  - PUSH(S,x) --- O(1)
  - POP(S) --- O(1)
  - MULTIPOP(S,k) --- Min(s,k)

Stack S

```
MULTIPOP(S,k)
1 while not STACK-EMPTY(S) and k ≠ 0
2 do POP(S)
3 k ← k - 1
```

Stack Example (cont.)

Non-Tight Bound

Assume O(n) ops. Then the worst case is a MultiPop of n objects.
Hence, T(n) = O(n^2).

Ergo, T(n)/n = O(n) cost per op.

Aggregate Method

- If the stack S is initially empty, then an object can be Popped only after it has been pushed. So if there are n ops, then
  \[ \# Pops \leq \# Pushes \leq n \]

- Moreover, # Pops counts all Pops, even those executed by all MultiPop ops. So the total work T(n) for n ops is at most n+n = O(n). Hence, the amortized cost per operation is
  \[ T(n)/n = O(1) \]
Incrementing a Counter: Aggregate Analysis

We have a k-bit register which counts up from 0.

Value of counter = \[ \sum_{i=0}^{k-1} A[i] 2^i \]

There are two types of bit flip ops:
- Reset to 0
- Set to 1

Cursory Analysis: Non-Tight Bound

- The computation work for a single bit = \( O(k) \), since worst case is \( k-1 \) resets and 1 set.
- Hence, work for n ops = \( O(nk) \)
- Ergo, Amortized cost = \( T(n)/n = O(k) \)

Better Analysis: Tighter Bound

Observation: If register is initially set to 0, then after a seq of n ops, bit \( A[i] \) flips exactly:

\[
\begin{cases} 
\left\lfloor \frac{n}{2^i} \right\rfloor \text{ times, if } i \leq \lfloor \log n \rfloor \\
0 \text{ times, if } i > \lfloor \log n \rfloor 
\end{cases}
\]

Hence, \( T(n) = \sum_{i=0}^{\lfloor \log n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\lfloor \log n \rfloor} 2^{-i} = 2n \)

So, \( T(n) = O(n) \Rightarrow T(n)/n = O(1) \)
Accounting Method

Assign Amortized Costs in such a way that you overcharge on some ops, and undercharge on others.

**Overcharge** = Actual Cost + Credit
**Undercharge** = Actual Cost – Credit
**Amortized Cost** = Actual Cost ± Credit

But always,

Total Actual Cost ≤ Total Amortized Cost

---

Stack Example: Accounting Method

Actual Costs:
- Push 1
- Pop 1
- MultiPop Min(k, s)

Amortized Costs:
- Push 2
- Pop 0
- MultiPop 0

---

Incrementing Counter: Accounting Method

Amortized Costs:
- Set Bit to 1 --- $2
- Reset Bit to 0 -- $0

Note. A bit cannot be reset unless it has been set sometime in the past. So if we pay $2 for each Set (which is $1 too much), then there will be $1 left to pay for a Reset, should it occur. Therefore, the total available credit will always be ≥ 0

Actual Cost ≤ Total Amortized Cost = 2#Sets ≤ 2n = O(n)
The Potential Method

To be Completed

Example: A Dynamic Table

TABLE-INSERT(T, x)
1 if size[T] = 0
2 then allocate table[T] with 1 slot
3 size[T] ← 1
4 if num[T] = size[T]
5 then allocate new-table with 2 \cdot size[T] slots
6 insert all items in table[T] into new-table
7 free table[T]
8 table[T] ← new-table
9 size[T] ← 2 \cdot size[T]
10 insert x into table[T]
11 num[T] ← num[T] + 1

Example: A Dynamic Table (Cont.)