Amortized Analysis

What is Amortized Analysis ?

- **In amortized analysis, the time required to perform a sequence of operations is averaged over all the operations performed.**

- No involvement of probability
- Average performance on a sequence of operations, even some operation is expensive.
- Guarantee average performance of each operation among the sequence in worst case.

Amortized analysis **is not just an analysis tool, it is also a way of thinking about designing algorithms.**

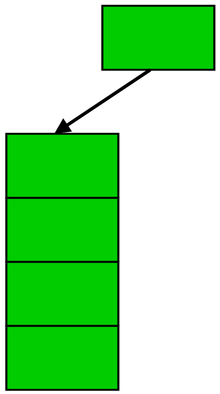
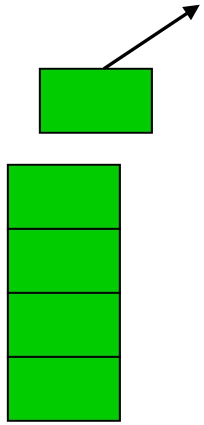
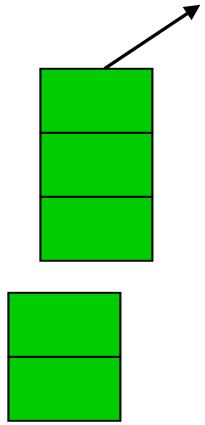
Methods of Amortized Analysis

- **Aggregate Method**: we determine an upper bound *T(n)* on the total sequence of *n* operations. The cost of each will then be *T(n)/n*.
- **Accounting Method:** we overcharge some operations early and use them to as prepaid charge later.
- **Potential Method:** we maintain credit as potential energy associated with the structure as a whole.

1. Aggregate Method

- Show that for all n, a sequence of n operations take worst-case time T(n) in total
- In the worst case, the average cost, or amortized cost , per operation is T(n)/n.
- The amortized cost applies to each operation, even when there are several types of operations in the sequence.

Aggregate Analysis: Stack Example

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
Worst-case cost:	O(1)	O(1)	$O(\min(\lvert S \rvert,k) = O(n)$

Amortized cost: O(1) per operation

……. Aggregate Analysis: Stack Example

- Sequence of n push, pop, Multipop operations
  - Worst-case cost of Multipop is O(n)
  - Have n operations
  - Therefore, worst-case cost of sequence is $O(n^2)$
- Observations
  - Each object can be popped only once per time that it’s pushed
  - Have <= n pushes => <= n pops, including those in Multipop
  - Therefore total cost = O(n)
  - Average over n operations => O(1) per operation on average
- Notice that no probability involved

2. Accounting Method

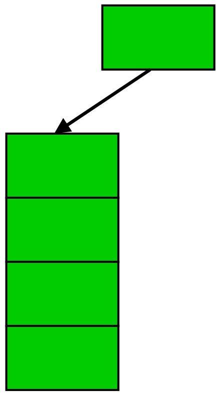
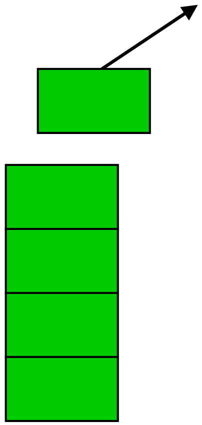
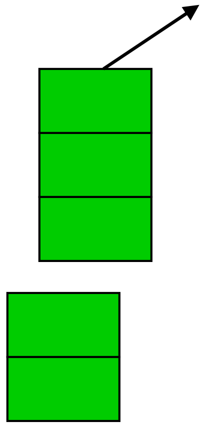
- Charge i th operation a fictitious amortized cost $\hat{c}_i$, where \$1 pays for 1 unit of work (i.e., time).
  - Assign different charges (amortized cost) to different operations
    - Some are charged more than actual cost
    - Some are charged less
- This fee is consumed to perform the operation.
- Any amount not immediately consumed and it is stored in the bank for use by subsequent operations.
- The bank balance (the credit) must not go negative!

We must ensure that for all n.

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

- Thus, the total amortized costs provide an upper bound on the total true costs.

….. Accounting Method: Stack Example

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
•Assigned cost:	2	0	0
•Actual cost:	1	1	min(|S|,k)

Push(S,x) pays for possible later pop of x.

….. Accounting Method: Stack Example

- When pushing an object, pay $2
  - $1 pays for the push
  - $1 is prepayment for it being popped by either pop or Multipop
  - Since each object has $1, which is credit, the credit can never go negative
  - Therefore, total amortized cost = O(n), is an upper bound on total actual cost

….. Accounting Method: Binary Counter

Introduction

- k-bit Binary Counter: A[0..k–1]

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^{i}$$

```
INCREMENT(A)
i ← 0
while i < length[A] and A[i] = 1
    do A[i] ← 0          ▷ reset a bit
       i ← i + 1
if i < length[A]
    then A[i] ← 1        ▷ set a bit
```

….. Accounting Method: Binary Counter

Consider a sequence of n increments. The worst-case time to execute one increment is $\Theta(k)$. Therefore, the worst-case time for n increments is $n \cdot \Theta(k) = \Theta(n \cdot k)$.

WRONG! In fact, the worst-case cost for n increments is only $\Theta(n) \ll \Theta(n \cdot k)$.

Let's see why.

Note: You'd be correct if you'd said $O(n \cdot k)$. But, it's an overestimate.

….. Accounting Method: Binary Counter

Total cost of n operations

A[0] flipped every op n

A[1] flipped every 2 ops $n/2$

A[2] flipped every 4 ops $n/2^2$

A[3] flipped every 8 ops $n/2^3$

… … … … …

A[i] flipped every 2^i ops $n/2^i$

Ctr	A[4]	A[3]	A[2]	A[1]	A[0]	*Cost*
0	0	0	0	0	**0**	***0***
1	0	0	0	**0**	**1**	***1***
2	0	0	0	1	**0**	***3***
3	0	0	**0**	**1**	**1**	***4***
4	0	0	1	0	**0**	***7***
5	0	0	1	**0**	**1**	***8***
6	0	0	1	1	**0**	***10***
7	0	**0**	**1**	**1**	**1**	***11***
8	0	1	0	0	**0**	***15***
9	0	1	0	**0**	**1**	***16***
10	0	1	0	1	**0**	***18***
11	0	1	**0**	**1**	**1**	***19***

….. Accounting Method: Binary Counter

Cost of n increments

$$= \sum_{i=1}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor$$

$$< n \sum_{i=1}^{\infty} \frac{1}{2^i} = 2n$$

$$= \Theta(n)$$

Thus, the average cost of each increment operation is $\Theta(n)/n = \Theta(1)$.

….. Accounting Method: Binary Counter

Charge an amortized cost of \$2 every time a bit is set from 0 to 1

- \$1 pays for the actual bit setting.
- \$1 is stored for later re-setting (from 1 to 0).

At any point, every 1 bit in the counter has \$1 on it… that pays for resetting it. (reset is "*free*")

Example:

0 0 0 $1^{\$1}$ 0 $1^{\$1}$ 0

0 0 0 $1^{\$1}$ 0 $1^{\$1}$ $1^{\$1}$ **Cost = \$2**

0 0 0 $1^{\$1}$ $1^{\$1}$ 0 0 **Cost = \$2**

Amortized Analysis

….. Accounting Method: Binary Counter

```
INCREMENT(A)
1.  i ← 0
2.  while i < length[A] and A[i] = 1
3.       do  A[i] ← 0          ⊳ reset a bit
4.             i ← i + 1
5.  if  i < length[A]
6.       then  A[i] ← 1        ⊳ set a bit
```

- When Incrementing,
  - Amortized cost for line 3 = \$0
  - Amortized cost for line 6 = \$2

- Amortized cost for INCREMENT(A) = \$2
- Amortized cost for n INCREMENT(A) = \$2n =O(n)

3. Potential Method

IDEA: View the bank account as the potential energy (as in physics) of the dynamic set.

FRAMEWORK:

- Start with an initial data structure D_0.
- Operation i transforms D_{i-1} to D_i.
- The cost of operation i is c_i.
- Define a ***potential function*** $\Phi : \{D_i\} \to \mathrm{R}$, such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all i.
- The ***amortized cost*** $\hat{c}_i$ with respect to Φ is defined to be $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

….. Potential Method

- Like the accounting method, but think of the credit as ***potential*** stored with the ***entire data structure***.
  - Accounting method stores credit with specific objects while potential method stores potential in the data structure as a whole.
  - Can release potential to pay for future operations
- Most flexible of the amortized analysis methods.

….. Potential Method

$$\underbrace{\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})}_{}$$

$$\underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\textit{potential difference } \Delta\Phi_i}$$

- If $\Delta\Phi_i > 0$, then $\hat{c}_i > c_i$. Operation i stores work in the data structure for later use.
- If $\Delta\Phi_i < 0$, then $\hat{c}_i < c_i$. The data structure delivers up stored work to help pay for operation i.

Amortized Analysis

….. Potential Method

The total amortized cost of *n* operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1}) \right)$$

Summing both sides telescopically.

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

$$\geq \sum_{i=1}^{n} c_i \qquad \text{since } \Phi(D_n) \geq 0 \text{ and } \Phi(D_0) = 0.$$

….. Potential Method: Stack Example

Define: $\phi(D_i)$ = #items in stack Thus, $\phi(D_0)=0$.

Plug in for operations:

Push: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
$= 1 + j - (j-1)$
$= 2$

Pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
$= 1 + (j-1) - j$
$= 0$

Multi-pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
$= k' + (j-k') - j$ $k'=\min(|S|,k)$
$= 0$