**Amortized Analysis**

Appropriate when the worst-case cost of an operation can occur only after many inexpensive operations.

Calculate the worst-case total cost of a sequence of operations to get a bound for the average cost of an operation over the sequence.

**Methods:**
- Aggregate Method
- Accounting Method
- Potential Method

---

**Amortized Analysis (cont)**

**Aggregate Method**

Example: Incrementing a k-bit Counter  $A[0:k-1]$

Time required is proportional to number of bits reset.

In $n$ operations,

- $A[0]$ is reset at most $\left\lceil \frac{n}{2} \right\rceil$ times
- $A[1]$ is reset at most $\left\lceil \frac{n}{4} \right\rceil$ times
- ... 
- $A[k-1]$ is reset at most $\left\lceil \frac{n}{2^{k-1}} \right\rceil$ times

Worst-case for $n$ increments is $\Theta(n+k)$.

---

**Amortized Analysis (cont)**

**Accounting Method**

Overcharge some operations to pay for others.

Need to ensure credit balance is not negative.

Example: Incrementing a k-bit Counter  $A[0:k-1]$

Setting a bit has cost 2. (pay ahead for resetting it)
Resetting a bit has cost 0. (was prepaid)

Each increment sets at most one bit.
The cost of $n$ increments is $O(n)$ on an initially 0 counter. $O(n+k)$ if counter can be non-zero initially.
Amortized Analysis (cont)

**Potential Method**

Each operation pays for increasing the potential of the data structure.

$D_i$ = data structure after $i^{th}$ operation

$\Phi(D_i) =$ potential of $D_i$

amortized cost is $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

real cost

$\sum_{i=0}^{n} \hat{c}_i = \sum_{i=0}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=0}^{n} c_i + \Phi(D_n) - \Phi(D_0)$

Amortized Analysis (cont)

Example 2: Widget Distribution

$k$ widgets are trucked to warehouse every night

Initially empty

In daytime $V$ vans can take widgets to stores

One widget per van per day.

Amortized Analysis (cont)

Example 2: Widget Distribution

Maximum number of truck and van trips per day: $V + 1$

Over $n$ days maximum number of trips is: $n(V + 1)$?

**Aggregate Method**

Each widget accounts for $\frac{1}{k} + 1$ trips.

Over $n$ days $kn$ widgets leave factory.

Total # of trips is $kn\left(\frac{1}{k} + 1\right) = n(k + 1)$
Amortized Analysis (cont)
Example 2: Widget Distribution

**Accounting Method**

Each truck trip is charged $k+1$ trips. Van trips are free. In $n$ days there are $n$ truck trips. Cost is $n(k+1)$

**Potential Method**

$D_i$ is the warehouse at the end of day $i$

$\Phi(D_i) = \#\text{ of widgets in } D_i$

$c_i$ is the number of trips on day $i$

---

Amortized Analysis Of Fibonacci Heaps

**Potential Method**

$\Phi(H) = a(t(H) + 2m(H))$

Largest constant in any of the operations

$\#\text{ of trees in } H$

$\#\text{ of marked nodes in } H$

Summed over all Fibonacci Heaps

Everywhere!!

---

Amortized Analysis Of Fibonacci Heaps (cont)

$H = (\min[H], n[H])$

$t[H] = \#\text{ of trees in } H = 5$

$m[H] = \#\text{ of marked nodes in } H = 3$

$\Phi(H) = a(5+2(3)) = 11a$
### Amortized Analysis Of Fibonacci Heaps (cont)

\[ \Phi(H) = a(t(H) + 2m(H)) \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Actual Cost</th>
<th>( \Delta ) Potential</th>
<th>Amortized Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make-Heap()</td>
<td>( \Theta(1) )</td>
<td>0</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>Minimum(H)</td>
<td>( \Theta(1) )</td>
<td>0</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>Insert(H, x)</td>
<td>( \Theta(1) )</td>
<td>( a )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>Union (H, H_x)</td>
<td>( \Theta(1) )</td>
<td>0</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>Delete - Min(H)</td>
<td>( O(D(n) + t(H)) )</td>
<td>( a(D(n) - t(H) + 1) )</td>
<td>( \Theta(D(n)) = \Theta(\log n) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a[2m(H) + t(H)] )</td>
<td></td>
</tr>
<tr>
<td>Decrease(H, x, k)</td>
<td>( O(c) )</td>
<td>( a(4 + c) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( d[(H) + c + 2m(H) - (c - 1) + 1 - t(H) + 2m(H)] )</td>
<td></td>
</tr>
<tr>
<td>Delete(H, x)</td>
<td></td>
<td></td>
<td>( \Theta(\log n) )</td>
</tr>
</tbody>
</table>

### Amortized Analysis Of Fibonacci Heaps (cont)

Without Decrease or Delete all trees will be Binomial Trees.

Each Insert operations pays \( a \) extra to cover the cost of being linked into a new tree in `ConsolidateHeap`.

Decrease or Delete operations can make long skinny trees.

When a child is removed and its parent is marked this adds \( 2a \) to the potential to cover the cost of

1. traversing the node again in another Decrease or Delete
2. the extra tree created when it is traversed the second time.