Amortized Analysis

Data Structure

where operations may take different amounts of time, depending on the State of the data Structure.

Only care about total time taken by a sequence of n operations (worst case)

Example: binary counter.

Stores a natural number in binary. 

Ops:
reset – set counter to 0
increment – add one to counter
display – output current contents

Assume:
reset
n increments (for some n)
display
list of bits
need k bits, where
k = \lceil \log n \rceil + 1
Increment takes
\( \Theta(k) \) worst-case time.
\( \Theta(lg n) \) (per increment)
worst-case

Total time for \( n \) increments
is \( O(n \lg n) \).

Tight? No
Worst case happens rarely in the sequence.

Suppose \( i \) carries:
\[
...01\ldots11111
\]
\[
10\ldots00000
\]

\[ \frac{n}{2^i} \] multiples of \( 2^i \)
from 1 to \( n \)
\[ \frac{n}{2^i} \] many odd multiples of \( 2^i \) in \( \{1, \ldots n\} \).

\[ \Theta\left(\frac{n}{2^i}\right) \] \( i \) carries takes \( \Theta(i) \) time.

Total time for \( n \) increments
\[ T(n) = \Theta \left( \sum_{i=0}^{\infty} i \cdot \frac{n}{2^i} \right) = \]
$T(n) = \Theta \left( n \sum_{i=1}^{k} i \right) = \Theta \left( n \sum_{i=1}^{\infty} \frac{i}{2^i} \right) = \Theta(n)$ converges

upper bound $S(n)$ is obvious, so $T(n) = \Theta(n)$ on average, $O(1)$ time per increment.

$O(1)$ amortized time per increment.

An operator (or set of possible operations) takes amortized time $t$ if any sequence of $n$ such operations takes $\leq tn$ total time (in worst case) increment takes $O(1)$ amortized time decrement?
counter (initially 0)
a sequence of
n ops (increment/decrement)
amortized time per op?

\[ 2^i \text{ many increments where} \]
\[ \frac{n}{4} \leq 2^i \leq \frac{n}{2} \]
\[ i \geq \lg n - 2 \]
from now on, alternate
decrement/increment

Total time for this particular sequence is
\[ \geq \frac{n}{2} (\lg n - 2) = \Omega(n \lg n) \]

\[ \therefore \Omega(\lg n) \text{ amortized} \]
time per op
no better than
non-amortized time.

Better implementation
of increment/decrement
counter:
\[ b_{k-1} b_{k-2} \ldots b_1 b_0 \]
each \( b_i \in \{0, 1\}^3 \)

represents each
number in the range
\[ 0 - 2^k - 1 \]
\[ n = \frac{2^k - 1}{2^3} \cdot b_i \]
Alternative:
store
\[ b_{k-1} \ldots b_0 \]
where each \( b_i \in \{ -1, 0, 1 \} \)
represents the number
\[ n = \sum_{i=0}^{k-1} 2^i b_i. \]

\( n \) no longer has a unique representation.

\[ 5 = 101 \]
\[ = 2 \cdot (-1) \cdot (-1) \]
\[ = \boxed{021} \]
increment: only carry if sum of digits is \( > 2 \)

\[ \begin{array}{c}
021 \\
+ \quad 022
\end{array} \]
\[ \begin{array}{c}
\hline
111
\end{array} \]

after a carry

\[ \begin{array}{c}
16 \quad b
\end{array} \]
\[ + \quad c \]
\[ \begin{array}{c}
d = 1
\end{array} \]

3 methods for amortized analysis:
- aggregate
- accounting
- potential
Example: Stack with multipop:

ops:

$O(1)$—push 1 item onto stack

$O(k)$—pop $k$ items off stack

Start with empty stack, any sequence of $n$ ops.

Aggregate method:

$\leq n$ pushes, so $\leq n$

different items total:

an item is pushed initially, sits there
for a while, then is popped (maybe).

For an item:

1 unit of time to push it

1 unit of time for that item
during a pop

$\leq 2$ units of time spent processing
any single item.

$\leq n$ items processed

$\leq 2 \cdot n = O(n)$
So $O(1)$ amortized time per operation.

Example: dynamic table resizing.