Problem Stack with Backup

- New operation Backup (S) occurs after every k operations, and writes a copy of the stack.
- Assume stack height is always ≤ k.

- Use same accounting method as for Stack with Multipop, but just add 1 more unit of amortized time to all operations other than Backup.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Real-time</th>
<th>Amortized time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push</td>
<td>1</td>
<td>2 + 1</td>
</tr>
<tr>
<td>Pop</td>
<td>1</td>
<td>0 + 1</td>
</tr>
<tr>
<td>Multipop</td>
<td>( \min 3, n^3 )</td>
<td>0 + 1</td>
</tr>
<tr>
<td>Backup</td>
<td>n ≤ k</td>
<td>0</td>
</tr>
</tbody>
</table>

- Since the stack accumulates 3 extra units of credit before every Backup operation, we can always pay for Backup with stored credit.
Exercise: Simulating a queue with two stacks

To simulate the operations Create, Put, and Get on a queue Q, we use two stacks, Front [Q] and Rear [Q], as follows:

Queue Q

Front [Q]            Rear [Q]
0         ...         0
first element    last element

Our implementation is below.

\[
\text{function Create () begin} \\
Q := Memory () \\
\text{Front [Q] := Stack ()} \\
\text{Rear [Q] := Stack ()} \\
\text{return Q} \\
\text{end}
\]

\[
\text{procedure Put (x, Q) begin} \\
\text{Push (x, Rear [Q])} \\
\text{end}
\]

\[
\text{function Get (Q) begin} \\
\text{if Empty (Front [Q]) then} \\
\text{while not Empty (Rear [Q]) do} \\
\text{Push (Pop (Rear [Q]), Front [Q])} \\
\text{return Pop (Front [Q])} \\
\text{end}
\]
Exercise cont’d

For the analysis, let us measure the actual time by the number of Pushes and Pops, and take as our potential function

\[ \Phi_Q = 2 \cdot \text{Size}(\text{Rear}[Q]). \]

Then

\[ a_{\text{Put}} = t_{\text{Put}} + \Delta \Phi_{\text{Put}} \]
\[ = 1 + 2 \]
\[ = O(1). \]

Suppose \( x \) elements are moved from \( \text{Rear}[Q] \) to \( \text{Front}[Q] \) by a Get. Then

\[ a_{\text{Get}} = t_{\text{Get}} + \Delta \Phi_{\text{Get}} \]
\[ = (1 + 2x) - 2x \]
\[ \uparrow \quad \uparrow \]
\[ \text{final Pop} \quad \text{Push, Pop in loop} \]
\[ = O(1). \]

Thus we can simulate a queue with two stacks in \( O(1) \) amortized time.
Exercise: Stack with Multi-Push.

Adding an operation

\[ \text{Multi-Push} \left( S, A, i \right) \]

which is equivalent to

\[ \text{Push} \left( S, AC(1) \right) \]
\[ \text{Push} \left( S, AC(2) \right) \]
\[ \vdots \]
\[ \text{Push} \left( S, AC(i) \right) \]

would spoil the \( O(1) \) amortized time bound on stack operations:

\[
\begin{align*}
\text{Multi-Push} \left( S, A, m \right) & \quad \text{1} \\
\text{Multi-Pop} \left( S, m \right) & \quad \text{1} \\
\text{Multi-Push} \left( S, A, m \right) & \quad \text{2} \\
\text{Multi-Pop} \left( S, m \right) & \quad \text{2} \\
\vdots & \\
\text{Multi-Push} \left( S, A, m \right) & \quad \text{m} \\
\text{Multi-Pop} \left( S, m \right) & \quad \text{m}
\end{align*}
\]

takes \( \Theta(m^2) = \omega(m) \) time. (Note that the time for \( m \) operations must be \( \omega(m) \) for a counterexample.)
Problem (Deleting the larger half)

Implement the following operations on a set $S$ of numbers,

- Insert ($x, S$) : Add $x$ to $S$;
- Delete Larger Half ($S$) : Delete the largest $\frac{|S|+1}{2}$ elements from $S$;

So both operations take $O(1)$ amortized time.

Solution sketch

We implement these operations as follows:

- Insert ($x, S$) : Put $x$ onto a singly-linked, unordered list $L$.
- Delete Larger Half ($S$) : Compute the median element $x$ of $S$.

Every element $y$ in $S$, with $y \leq x$, delete from $L$.

For our amortized analysis we use the charging method:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Actual Time</th>
<th>Amortized Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Delete Larger Half</td>
<td>$2n$</td>
<td>0</td>
</tr>
</tbody>
</table>

We take the actual time for Delete Larger Half, which (i) computes the median and then (ii) does a scan to delete elements, to be $2n$.

An insert takes 1 unit of actual time, but receives 5 units of amortized time; we store the 4 units of credit on the inserted element.
Solution cont'd

For Delete Larger Half, let us assume every element has 4 units of credit on it before the operation.
To execute Delete Larger Half, we use 2 units from every element. (So now every element has 2 units remaining on it.)

Take the remaining 2 units from every deleted element and place those units on the elements not deleted.

Now every element left in S has 4 units of credit again (as there are at least as many elements deleted as not deleted). So the credit assumption is maintained.