CIS 413/513 Advanced Data Structures  
Winter 2020  

Assignment 1  

due Wednesday, January 22, 2020  

Handwritten solutions are acceptable if very neat, but it is preferable to type up your solution (LaTeX recommended).

1. Consider the array doubling problem covered in class and shown in some of the references. It is shown that the amortized cost of an Insert is 3 if the array-doubling cost charges only for the cost of copying all elements from the original array to the larger one. That is, if an Insert is performed at location \( i \), where \( i = 2^k + 1 \) (for some \( k \)), then the cost is \( i - 1 \) for the copying and then 1 for the write at location \( i \).

On the other hand, we saw in class that the amortized cost is 5 if the array doubling is charged for (i) copying the \( i - 1 \) elements into the left side of the new array (locations 1, \ldots, \( 2^k \)) and (ii) initializing the right side (locations \( 2^k + 1, \ldots, 2^{k+1} \)) to zero. Total cost, including the write of the new element, is \( (i - 1) + (i - 1) + 1 = 2i - 1 = 2^{k+1} + 1 \).

For this problem, we modify the array doubling cost to charge for (i) an initialization of the whole array (set locations 1, \ldots, \( 2^{k+1} \) to zero) and then (ii) copying the smaller array into the larger one (locations 1, \ldots, \( 2^k \)).

For this problem, you should carefully derive the amortized cost of an Insert operation using each of the given techniques.

(a) aggregate (also called brute force)
(b) accounting (aka banker’s, taxation)
(c) potential (aka physicist’s)

Notes:
• The amortized cost should be 7 per insertion (I think!).
• The arrays here are indexed starting at 1, rather than the first location being 0. Sorry.

2. (from Er) A quack is a data structure combining properties of both stacks and queues. It can be viewed as a list of elements written left to right such that three operations are possible:

- QUACKPush\((x)\): add a new item \( x \) to the left end of the list;
- QUACKPop(): remove and return the item on the left end of the list;
- QUACKPull(): remove the item on the right end of the list.

Implement a quack using three stacks and \( O(1) \) additional memory, so that the amortized time for any QUACKPush, QUACKPop, or QUACKPull operation is \( O(1) \). In particular, each element in the quack must be stored in exactly one of the three stacks. Again, you cannot access the component stacks except through the interface functions Push and Pop.
3. (from Er) Suppose you are given a collection of up-trees representing a partition of the set \( \{1, 2, \ldots, n\} \) into disjoint subsets. **You have no idea how these trees were constructed.** You are also given an array \( \text{node}[1 \ldots n] \), where \( \text{node}[i] \) is a pointer to the up-tree node containing element \( i \). You task is to create a new array \( \text{label}[1 \ldots n] \) using the following algorithm \text{LABELEVERYTHING}:

\[
\text{for } i=1 \text{ to } n \\
\quad \text{label}[i] = \text{Find}(\text{node}[i])
\]

Prove that if we implement \text{FIND} using path compression, then all \( n \) operations are executed in \( O(n) \) time.