Lecture 6 exercise solutions

1.

ins, ins (expand, size 2), ins (expand, size 4), ins, ins (expand, size 8), ins, three dels, five ins (after this the table is full containing 8 elements), six dels, del (contract, size 4), three ins, ins (expand, size 8), three ins (again, the table is full containing 8 elements). All in all four expansions, one contraction.

CLRS4 16.1-2 (CLRS3 17.1-2)

Here is the sequence of operations that would cost $\Theta(nk)$: DECREMENT, INCREMENT, DECREMENT, INCREMENT, … The counter would flip from k zeros to k ones or the other way around on each operation in the sequence.

CLRS4 16.2-3 (CLRS3 17.2-3)

Here is the pseudocode (we assume a counter is an array of size k with a special field *len* initialized to 0). Note the lines 6–8 in **Increment** which are added to the example from the text-book and the lecture.

```
Increment(A)
01 i = 0
02 while i < k and A[i] = 1
03    A[i] = 0
04    i = i + 1
05 if i < k then
06    A[i] = 1
07    if A.len <= i then A.len = i + 1
08 else A.len = 0


Reset(A)
01 for i = 0 to A.len-1
02    A[i] = 0
03 A.len = 0
```

Here is a simple analysis using the accounting method. Let's pay \$3 for **Increment**. Then we use one dollar to set a bit to to 1 in line 6 and we place one dollar on the account of that bit to pay for checking and flipping the bit back to 0 in line 3 of a future call to **Increment**. Finally, if *A.len* is incremented in line 7, we put the third dollar on the account of the i-th bit, otherwise we throw it away. The "third dollar" is used in line 2 of **Reset** to set the corresponding bit to zero. We can then do **Reset** for free and any sequence of n **Increment**'s and **Reset**'s will cost at most $3n$, $O(1)$ per operation.

CLRS4 16.3-5 (CLRS3 17.3-6)

The queue is stored in the two stacks T and H.

```
Enqueue(a)
01 T.Push(a)


Dequeue()
01 if H.Empty()
02    while not T.Empty() do H.Push(T.Pop())
03 return H.Pop()
```

If we put two dollars into the account of an element when pushing it into T in line 1 of **Enqueue**, we can use those two dollars to pop and push that element in line 2 of **Dequeue**. Thus, the amortized cost of **Enqueue** is 3, and the amortized cost of **Dequeue** is 1 (popping from H in line 3), i.e., both operations have $O(1)$ amortized cost.

3.

1. The problem is that the pseudocode will remove too many elements if there are duplicates of x. We could fix the pseudocode by adding a loop after the main loop and before the last line:

for $i = j$ **to** floor($S.num$ / 2) **do** $S[i] = x$

Alternatively, after x = **Median**(S), we could call the **Partition** algorithm from the **Quicksort** using x as the pivot. Then, we would just remove the second half of the array.

2. We can demonstrate the constant amortized cost through the accounting method by charging k+1 dollars for an insertion, such that k dollars are always in the account of each of the inserted elements. Then, deleting 1/k-th of the elements can be completely paid by taking one dollar from the accounts of all elements in the structure (leaving $k - 1$ dollars). The extra (k-1)*n/k = $n - n/k$ dollars of the removed elements are equal to the number of elements remaining in the structure. Thus, these dollars are enough to give one dollar to each of the remaining elements, restoring all their accounts to k.

CLRS4 16.4-4 (CLRS3 17.4-3)

Note that when $\alpha_{i-1} > ½$, the potential function is exactly the same as described in the textbook. If $\alpha_{i-1} \leq ½$, then $2num < size$ and $\Phi = |2num - size| = size - 2num$. Note also that $num_i = num_{i-1}$-1, because it is a deletion.

Case 1: $1/3 < \alpha_{i-1} \leq ½$: no contraction, which means that $size_i = size_{i-1}$.

$\hat{c}_i = 1 + \Phi_i - \Phi_{i-1} = 1 + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1}) = 1 - 2(num_{i-1} - 1) + 2num_{i-1} = 3.$

Case 2: $\alpha_{i-1} = 1/3$: contraction, which means that $size_{i-1} = 3num_{i-1}$ and $size_i = (2/3)size_{i-1} = 2num_{i-1}$. Finally, the real cost $c_i = num_{i-1}$.

$\hat{c}_i = num_{i-1} + \Phi_i - \Phi_{i-1} = num_{i-1} + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1}) = num_{i-1} + 2num_{i-1} - 2(num_{i-1} - 1) - 3num_{i-1} + 2num_{i-1} = 2.$

In both cases the cost is bounded above by a constant. Case 1, shows that 2 is added to the potential on each deletion, similarly to how 2 is added to the potential on each insertion when $\alpha > ½$. In this way, the potential builds up to $size$/3 during $size$/6 deletions.