Homework 5 Solutions

CSC 140 – Advanced Algorithm Design and Analysis

If you find any errors in the solution writeup, please let me know. Ask in class or come to office hours if you need any further help understanding the problems and their solutions.

17.1-3) See book solution at `https://mitpress.mit.edu/books/introduction-algorithms-third-edition`

17.2-2) Consider the operations after the one that costs 2^k upto and including the one that costs 2^{k+1}. There are 2^k of them. For each, we'll put 2 into the account, so over the 2^k operations, 2^{k+1} goes into the account. We'll also pay immediately (not out of the account) for each of the first $2^k - 1$ operations, meaning they each cost 3 (1 for the work and 2 to put into the account). The final operation of this sequence costs $2^{k+1} + 2$ (2^{k+1} for the actual work and 2 to put into the account). We'll pay for this final operation 2^{k+1} out of the account, leaving only the cost of 2. Thus each operation costs no more than 3 (amortized), or $\Theta(1)$ per operation.

17.3-2) We need a potential function Φ that is never negative, goes up by a constant amount for each non-power-of-two cost operation, and goes down by a lot on the power-of-two cost operations to counter the actual cost of those. If we define $\Phi(i) = 2i$ as the potential after the i-th operation, then we get the gradual increase, but it does not go down to compensate for the expensive operations. What I really want is my potential to collapse to 0 after an expensive operation.

$$\Phi(i) = 2i - 2 \cdot 2^{\lfloor \lg i \rfloor}$$

Plot: `http://www.wolframalpha.com/input/?i=plot+2k+-+2*2%5Efloor(log_2+k),+k%3D0+to+1024`

This potential function goes up by 2 for each i that is not a power of 2, and is 0 when i is a power-of-two. Thus for non power-of-two i the cost is:

$$\hat{c} = c + \Delta\Phi = 1 + 2 = 3$$

and for power-of-two i (where $i = 2^k$ for some k) the cost is:

$$\hat{c} = c + \Delta\Phi = i + [2i - 2i] - [2(i-1) - 2 \cdot 2^{k-1}] = i + 0 - 2i + 2 + i = 2$$

So, amortized, each operation is $\Theta(1)$.

17.3-6) A stack reverses the order of things, so if you push 1,2,3 onto stack S2, and then pop them off and push them onto stack S1, then the top of S1 is 1. Let S1 and S2 be initially empty stacks, representing and empty queue. Convince yourself that the following obeys the queue discipline.

```
ENQUEUE(x):         DEQUEUE():                          EMPTY():
  S2.push(x)          if ( S1.empty() )                   return S1.empty() and S2.empty()
                        while ( not S2.empty() )
                          S1.push(S2.pop())
                      return S1.pop()
```

The expensive operation here is when S1 is empty, meaning the next item to dequeue is at the bottom of the S2 stack, and dequeue is called. We want this event to correspond with a large decrease in the potential function. This suggests defining $\Phi(Q) = |S2|$. It goes up slowly with each ENQUEUE and then when S2 gets emptied all the work of emptying it gets paid from a collapse in Φ. In the following analysis, we'll count calls to push and pop as units of work.

Cost of ENQUEUE: $\hat{c} = c + \Delta\Phi = 1 + 1 = 2$. Cost of cheap DEQUEUE: $\hat{c} = c + \Delta\Phi = 1 + 0 = 1$. Cost of expensive DEQUEUE (where $|S1| = 0$ and $|S2| = n$ before dequeue): $\hat{c} = c + \Delta\Phi = (2n+1) + (0-n) = n+1$.

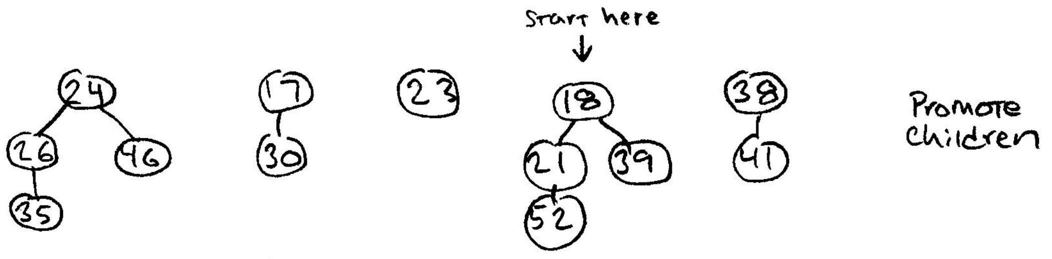
Notice that the $\Delta\Phi$ of $-n$ was insufficient to cover the cost of the $2n+1$ pushes and pops. This is easily fixed. If we redefine $\Phi(Q) = 2 \times |S2|$, then we instead have

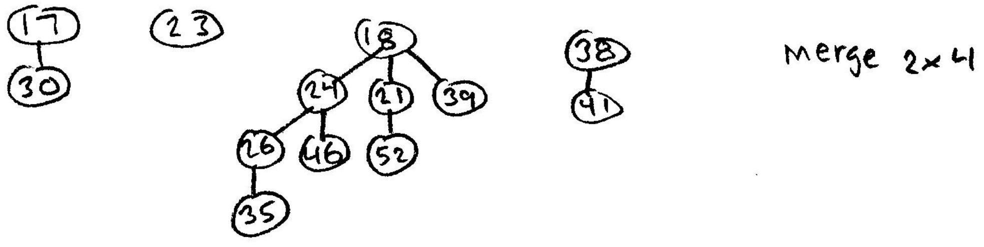
Cost of ENQUEUE: $\hat{c} = c + \Delta\Phi = 1 + 2 = 3$. Cost of cheap DEQUEUE: $\hat{c} = c + \Delta\Phi = 1 + 0 = 1$. Cost of expensive DEQUEUE (where $|S1| = 0$ and $|S2| = n$ before dequeue): $\hat{c} = c + \Delta\Phi = (2n+1) + (0-2n) = 1$. In all these cases the amortized cost is $\Theta(1)$.

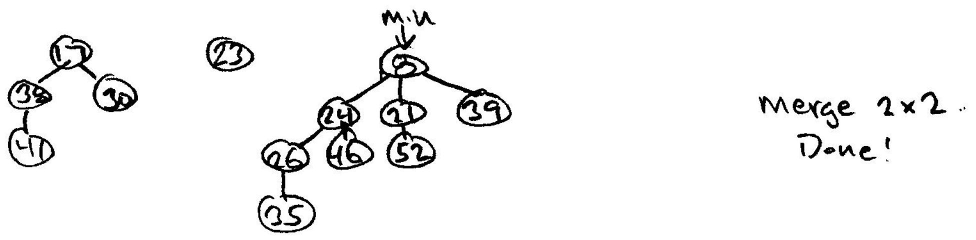
21.3-1) I simulated the algorithms from the book by drawing the trees after each step. Then to verify my answer, I coded the data-structure, algorithms and main program. My program output the following table.

```
element   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
parent    8  16   4  16   8   8   8  16  16  16  12  16  16  16  16  16
rank      0   1   0   2   0   1   0   3   0   1   0   2   0   1   0   4
```

2) If FIB-HEAP-EXTRACT-MIN were called on the heap at Figure 19.5a, what would the resulting data structure be? Do this carefully, following the algorithms and figures of the chapter. You can ignore all marking, that's only used by algorithms we did not discuss in class.







The CONSOLIDATE algorithm isn't completely clear on some points, so I had to scrutinize Figure 19.4 to do it the same way the authors did. If you have to do this on an exam or quiz you should follow the same conventions. Promoting children maintains their order. The visitation of nodes is right-to-left beginning just to the right of the old min. When combining two trees, the new child is to be the leftmost child. One point is clear, but takes some figuring out: when two trees are combined, the combined tree is located where its root already was.