Algorithms Homework 2

Liam Dillingham

October 2, 2018

1 Question 6-1

We can build a heap by repeatedly calling MAX-HEAP-INSERT to insert the elements into the heap. Consider the following variation on the BUILD-MAX-HEAP procedure:

1.1 Do the procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP' always create the same heap when run on the same input array? Prove that they do, or provide a counterexample.

I started out just trying a few small examples to see if I could discover a counter example. And I did.
Suppose we have an array $< 1, 2, 3, 4, 5 >$
By using our new algorithm, BUILD-MAX-HEAP', we obtain the results $< 5, 4, 2, 1, 3 >$ Yet using the original algorithm, BUILD-MAX-HEAP, we obtain $< 5, 4, 3, 1, 2 >$
Thus the two do not produce the same heap.

1.2 Show that in the worst case, BUILD-MAX-HEAP requires $\Theta(nlg(n))$ time to build an n-element heap

Note that within BUILD-MAX-HEAP calls MAX-HEAPIFY $\lfloor \frac{A.length}{2} \rfloor$ times.
$\frac{1}{2}$ is a constant, so we can remove that. Thus MAX-HEAPIFY is called $\Theta(n)$ times.

Suppose that for a given subtree i, all of its child subtrees are not max heaps. That is, if we are at the root, then we will have to compare and exchange values for the entire height of the tree. Recall that the height of a tree is equal to $\lfloor lg(n) \rfloor$. Thus, worst case, MAX-HEAPIFY will run $\Theta(lg(n))$ times.

Therefore, since BUILD-MAX-HEAP calls MAX-HEAPIFY $\Theta(n)$ times, and worst-case MAX-HEAPIFY runs at time $\Theta(lg(n))$, then the worst-case performance for BUILD-MAX-HEAP is $\Theta(nlg(n))$

2 Question 7.2-2

What is the running time of QUICKSORT when all elements of array A have the same value?

We know that the PARTITION function takes $\Theta(n)$ because it completes n comparisons, so we know it takes at least $\Omega(n)$ time. Note that due to the nature of the comparison, $A[j] \leq x$ (From the book's pseudocode), this condition will always be true. Since this condition is always true, i is always incremented, and thus $A[r-1]$ and $A[r]$ will be swapped. Since the last and next to last elements are always swapped, each subsequent call to QUICKSORT recieves a subproblem of size $n-1$, and therfore, the runtime is $\Theta(n^2)$

3 Question 7.2-3

Show that the running time of QUICKSORT is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

Since $A[r]$ is the least element in the array, each call will produce a subproblem of 0 on the left, and $n-1$ on the right. Since each call to the partition does n comparisons, each partition call costs $\Theta(n)$ time. Also, note that the partition produces a subproblem of size $n-1$. Thus, QUICKSORT is called n times, and since PARTITION takes $\Theta(n)$ time, then the run time in the worst case is $\Theta(n)$.