EE 602 – Algorithm I **(UH Manoa, Fall 2020)**

Algorithms Assignment 6

October 10, 2020

Instructor: Yingfei Dong *Name: Frendy Lio Can*

Problem 1

A heap is a nearly complete binary tree where all the levels, except for the lowest, are completely full. We can conclude that a heap has a minimum of 2^h and a maximum of $2^{h+1} - 1$ elements, where h represents the height of the tree.

Therefore, $2^h \leq n \leq 2^{h+1} - 1 \Rightarrow h \leq lgn \leq h + 1$. Thus, since the height of a tree is an integer. We can conclude that $h = \lfloor lgn \rfloor$.

Problem 2

A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

Thus, if the root of the subtree does not contain the largest value occurring anywhere in the subtree, the heap property will be violated. Therefore, the root of a subtree will always be the highest value in that tree.

Problem 3

Lets show this by doing a proof by induction.

Base case:
Lets show that it is true for $h = 0$. Since the number of nearly complete binary tree is $\lceil \frac{n}{2} \rceil$. Thus, it is true for $h = 0$.

Inductive Step:
Assume that is true for $h - 1$. Let N_h be the number of nodes of height h induction the $n - node$ tree T_1.
Let T_2 be a tree that is made by removing the leaves of T_1. This implies that
$n_2 = n_1 - \lceil \frac{n_1}{2} \rceil = \lceil \frac{n_1}{2} \rceil$. This implies that the height in $T_2 = h - 1$
Let N_2 denote the number of nodes at height $h - 1$ in T_2. Thus, by induction, we conclude the following:

$N_h = N_2 = \lceil \frac{n_2}{2^h} \rceil = \lceil \frac{\frac{n_1}{2}}{2^h} \rceil = \lceil \frac{n_1}{2^{h+1}} \rceil$.

Therefore, there are at most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes of height h in any n-element heap.

Problem 4

It is $\Theta(n^2)$ because the partition in QUICKSORT will always happen at the end of the array. Thus, this QUICKSORT will always behave as worst-case.

Problem 5

The reason why we analyze the expected running time of a randomized algorithm is because it represents a more typical representation of the time cost for a randomized algorithm.

This can be explained as a randomized algorithm will act different each time it is executed. Thus, instead of calculating the best-case or worst-case running-time, we calculate the average as every time we run the algorithm, the running time will be different.

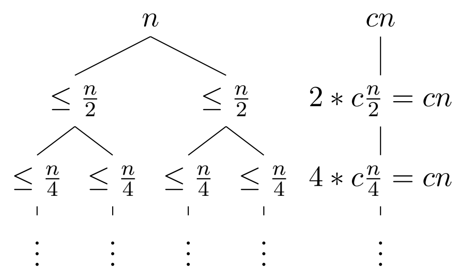
Problem 6

The best-case running time of QUICKSORT happens when the partitions sizes are equal or are within 1 of each other.

The partition sizes are equal when the array has an odd number of elements and the pivot is right in the middle after each partitioning. This implies that each partition has $(n-1)/2$ elements.

The partition sizes are 1 of each other when the array has an even number and one partition has $n/2$ elements while the other has $n/2 - 1$.

For both of these cases, we can conclude that at most $n/2$ elements are in each partition. Therefore we can draw the recursion tree:



We can observed that $f(n) = cn$, $a = 1$ and $b = 2$.

Therefore, in order to calculate the lower bound, we add the cost of each level of the tree by knowing that the tree is completed when the Recursion tree level is $log_b n = lgn$:

$$\begin{aligned}
T(n) &= \sum_{j=0}^{lg \cdot n} f(n) \\
&= \sum_{j=0}^{lg \cdot n} cn \\
&= \sum_{j=1}^{lg \cdot n} cn + cn \\
&\geq nlgn \\
\therefore T(n) &= \Omega(nlgn)
\end{aligned}$$