CSC 1200—Algorithms and Data Structures
Midterm 1 Review Answer Sketches

1. Solve the following recurrences:
   (a) \( T(n) = 4 T\left(\frac{n}{2}\right) + n \log n \)

   **Answer:**
   Recursive part: \( n^{\log_4 4} = n^1 = n \)
   Non-recursive part: \( f(n) = n \log n \)
   Ratio of recursive/non-recursive (ignoring logs): \( \frac{n^2}{n} = n \)
   Therefore, case (a). \( T(n) = n^{\log_4 4} = n \)

   (b) \( T(n) = 3 T\left(\frac{n}{3}\right) + n \)

   **Answer:**
   Recursive part: \( n^{\log_3 3} = n \)
   Non-recursive part: \( n \)
   Ratio: \( \frac{n}{n} = 1 \). Therefore, case (b) applies. \( T(n) = n \log n \)

2. Which of the following is true:
   (a) \( n \log n = \Omega(n^{1.1}) \)

   **Answer:** That's almost like saying \( n \log n \geq n^{1.1} \), or (dividing by \( n \)), we would be claiming that \( \log n \geq n^{0.1} \), which is FALSE.

   (b) \( n^{1.1} = \Omega(n \log n) \)

   **Answer:** TRUE.

3. What are the worst case complexities of the following operations:
   (a) Deletion from a binary search tree with \( n \) nodes.

   **Answer:** \( \Theta(\log n) \)

   (b) Find the maximum value in a max-heap with \( n \) values.

   **Answer:** \( \Theta(1) \)

   (c) InsertionSort of an array of \( n \) elements

   **Answer:** \( \Theta(n^2) \)

   (d) Using Merge to combine two sorted arrays of length \( \frac{n}{2} \) each into one sorted array of \( n \) elements.

   **Answer:** \( \Theta(n) \)

   (e) The time taken for the Divide step in the MergeSort algorithm.

   **Answer:** \( \Theta(1) \)
4. Insert the following values into an initially empty red-black tree: 30, 45, 50, 15, 10, 40.

Answer:
5. Assume the tree you get in question 4 is a binary search tree. Delete the value 30 from it and draw the resulting tree.

   Answer:

6. Given a value $N$ you want to calculate the value of $3^N$. Show that you can do this in $\Theta(\log N)$ steps. You may assume that $N$ is a power of 2.

   Answer: We know that when $N=1$, $3^N=3^1=3$. To calculate $3^N$, we first recursively calculate the value of $\frac{N}{3^2}$. Then, we square this value ($\frac{N}{3^2} \cdot \frac{N}{3^2}$) to get $3^N$. This algorithm's recurrence equation is $T(N)=T\left(\frac{N}{2}\right)+\Theta(1)$. We can solve this using the master method to give $T(n)=\Theta(\log n)$.

7. Suppose that we are given an array $A[1..N]$ of unsorted integers.

   (a) Give a technique to find if there are any duplicates in the array. The algorithm should run faster than $\Theta(n^2)$ in the worst case.

   Answer: First, sort the array using MergeSort. Second, scan the array to see if there are any two adjacent values that are the same. If there are two adjacent elements that are the same, we have found duplicates, otherwise there are none in $A$. The first step takes $\Theta(n \log n)$ time. The second step takes $\Theta(n)$ time in the worst case. Thus, the overall time of the algorithm is $\Theta(n \log n)$.

   (b) Give a method to find an element that occurs most frequently in this array.

   Answer: Again, first sort the array. After the sorting the array, keep track of "runs" in the array (a "run" is a sequence of values in the array that are all the same). Output the largest "run". The total time of the algorithm is again $\Theta(n \log n)$. 

8. Use heapsort the sort the numbers given in question 4.
   \textbf{Answer:} (I've written this answer assuming an array representation of a heap, but you could just as easily do it using the tree representation.)
   After Build-Heap, the array looks like:

   \[
   50 \ 45 \ 40 \ 15 \ 10 \ 30
   \]
   After the first extract max (remove 50), we get:
   \[
   45 \ 30 \ 40 \ 15 \ 10
   \]
   After the second extract max (remove 45), we get:
   \[
   40 \ 30 \ 10 \ 15
   \]
   After the third extract max (remove 40), we get:
   \[
   30 \ 15 \ 10
   \]
   After the fourth extract max (remove 30), we get:
   \[
   15 \ 10
   \]
   After the fifth extract max (remove 15), we get:
   \[
   10
   \]
   After the last extract max (remove 10), we have an empty heap, and we stop.
   Listing the extracted values in reverse order we get: 10 15 30 40 45 50.

9. You are given two lists of numbers \(S\) and \(T\) (each of length \(n\)), and a number \(x\).
   Describe a \(\Theta(n \log n)\) algorithm that determines whether or not there is one number in \(S\) and one number in \(T\) whose sum is exactly \(x\).
   \textbf{Answer:} First, sort array \(T\) using MergeSort. Second, take each value \(a\) in the array \(S\) and use binary search to check if \(x-a\) is in array \(T\). (If \(a\) is in \(S\) and \(x-a\) is in \(T\), their sum would be \(a+x-a=x\).) The first step takes \(\Theta(n \log n)\). In the second step, we perform one binary search for each value in \(S\). Each binary search takes \(\Theta(\log n)\) time in the worst case, for a total of \(\Theta(n \log n)\) time. It follows that the overall time of the algorithm is \(\Theta(n \log n)\).