Solutions for Midterm Practice Questions

Problem 1 Solution: In the first step, the list is partitioned with element 3 as the pivot element. The result is one list A with 1, 2, 0.5 and another list B with 9.5, 4.5, 6, 21, 3. The two sublists are sorted by recursive calls to quicksort. List A gives sublist C with 0.5 and sublist D with 2, 1. Sublist C contains just one element and it is already sorted. Sublist D gives sublist E with 1 and sublist F with 2, which are sorted. List B gives list G with 3, 4.5, 6 and H with 21, 9.5. Then G breaks into 3 and 4.5, 6. 4.5, 6 breaks into 4.5 and 6. List H breaks into 9.5 and 21. So the final result is 0.5, 1, 2, 3, 4.5, 6, 9.5, 21.

Problem 2 Solution: $n^{\log \log n} = o(n^{\log n})$ is TRUE. Take limits. This implies that also $n^{\log \log n} = O(n^{\log n})$ is TRUE.

Problem 3 Solution: There are $N + 1$ possible answers to this problem. Every decision tree that solves the problem must have at least $N + 1$ leaves. This gives a $\log N$ lower bound. Applying also a binary search on the the input $0, \ldots, N$ (ask if the number is less than $N/2$ and according to YES or NO answer, ask for $N/4$ or $3N/4$ respectively etc.) gives a $\log N$ upper bound.

Problem 4 Solution: The recurrence for the worst case running time $T(n)$ of merge sort under the assumption that merging two sorted arrays takes constant time is:

$$T(n) = 2T(n/2) + c$$

Using Master’s Theorem, we conclude that $T(n)$ is $O(n)$.

Problem 5 Solution: The recurrence of the described procedure is:

$$T(n) = T(n/3) + T(2n/3) + \Theta(n)$$

For simplicity consider:

$$T(n) = T(n/3) + T(2n/3) + n$$
You can solve the recurrence by applying Master’s Theorem. First, use 
\[ T(n) \leq 2T(2n/3) + n. \] Using master theorem, we get 
\[ T(n) = n^{\log_{3/2}2}. \]

We also introduce another solving method which gives a better bound 
in this case: Recursion trees.

\[ T(n) \] can be expanded to an equivalent tree representing the recurrence. 
The \( n \) term is put as the root. Its left subtree is \( T(n/3) \) and its right subtree 
is \( T(2n/3) \). You can carry on the same process by expanding the subtrees. 
So, the left subnode becomes \( n/3 \) and the right subnode becomes \( 2n/3 \) in 
the second level of recursion. Continue expanding each node in the tree 
according to the recurrence.

Adding the values across the levels of the recursion tree, we get a value 
of \( n \) for every level. The longest path from the root to the leaf is 
\( n \rightarrow (2/3)n \rightarrow (2/3)^2n \rightarrow \ldots \rightarrow 1. \) Since \( (2/3)^k n = 1 \) when \( k = \log_{3/2}n \), the 
height of the tree is \( \log_{3/2}n \). Adding up all levels of the tree, the solution is 
at most \( n \log_{3/2}n = O(n \log n) \).

**Problem 6 Solution:** Start from the root. If the value of the root is 
greater than \( x \) then continue recursively to the left child of the root. If the 
root equals \( x \) then print the root element and also its left subtree (following 
one of the known traversals). If the root is smaller than \( x \), then do what 
you did for the case that root equals \( x \) and also continue recursively to its 
right child.

The complexity is \( O(N) \) where \( N \) is the number of nodes in the AVL 
tree. In the worst case (all nodes are less than \( x \)), the algorithm has to 
traverse every node of the tree.