Question 1 (Computing Runtime, 30 points). Consider the following sorting algorithm:

RecursiveSort(A, i, j) \ \ sorts the elements of A between indices i and j
    if j=i+1
        if A[i]>A[j]
            swap(A[i],A[j])
        else
            Set k = ceiling(2*(j-i+1)/3)-1
            RecursiveSort(A, i, i+k)
            RecursiveSort(A, j-k, j)
            RecursiveSort(A, i, i+k)

(a) Give a recurrence for the runtime of this algorithm on an array of length n (when i = 1 and j = n).

This algorithm does \(O(1)\) work in order to split the problem into three recursive calls of approximately \(2/3\) the size of the original (because \(k\) the length of the sub-interval being sorted is roughly \(2(i - j)/3\)). So, we have the recurrence

\[ T(n) = 3T(2n/3 + O(1)) + O(1). \]

(b) What is the asymptotic runtime of this algorithm (as a big-\(\Theta\))?

Using the Master Theorem with \(a = 3, b = 3/2\) and \(d = 0\), we find that we are in the case where \(a > b^d\) and thus the asymptotic runtime is \(\Theta(n^{\log_{3/2}(3)})\).
**Question 2** (Longest Root to Leaf Path, 35 points). Let $T$ be a balanced binary tree on $n$ vertices (so $n$ is one less than a power of 2) with (possibly negative) edge weights. Give an $O(n)$ time divide and conquer algorithm for computing the length of the longest root-to-leaf path in $T$.

We note that the longest path passes through either the left subtree or the right subtree, in which case it is the length of the appropriate edge from the root plus the length of the longest path to the appropriate subtree. We have the algorithm:

```
LongestPath(T, l):
    If |T| = 1
        Return 0
    Set LeftDist = $l(root, root.left) + LongestPath(LeftSubtree(T), l)$
    Set RightDist = $l(root, root.right) + LongestPath(RightSubtree(T), l)$
    Return max(LeftDist, RightDist)
```

To see that this works, note that the only path has length 0 if the tree has size 1 (i.e. is a single node). Otherwise LeftDist computes the length of the longest path through the left subtree, and RightDist the length of the longest path through the right subtree. To analyze the runtime, note that this algorithm does $O(1)$ work in addition to making two recursive calls to problems of half the size, so the runtime is given by the recurrence:

$$T(n) = 2T(n/2) + O(1).$$

By the Master Theorem, the final runtime is $O(n)$. 
**Question 3** (Shortest Paths From Path Lengths, 35 points). *Suppose that you are given a graph \( G \) with (possibly negative) edge weights. Additionally, you are given a vertex, \( s \) and the lengths, \( d(w) \) of the shortest path from \( s \) to each other vertex \( w \) (you may assume that shortest paths actually exist). Give a linear time algorithm that given another vertex \( t \) returns the short path from \( s \) to \( t \) (the actual path and not just its length). You may assume that the shortest path is unique.*

We note that in the shortest path to \( t \), the vertex immediately before \( t \) must be a \( u \) so that \( d(t) = d(u) + \ell(u, t) \). In fact, if the path is unique, it is the only such \( u \). If \( t = s \), we return the trivial path. Otherwise, we find the unique \( u \) with the above property and return the shortest path to \( u \) followed by \( t \). Pseudocode is as follows:

\[
\text{ShortestPath}(G, s, t, d):
\]
\[
\text{If } t=s
\]
\[
\text{Return } [s]
\]
\[
\text{For each edge } (u, t) \text{ in } E:
\]
\[
\text{If } d(t) = d(u) + \ell(u, t)
\]
\[
\text{Return ShortestPath}(G, s, u, d) \text{ with } t \text{ appended to the end}
\]

Note that each recursive call to ShortestPath above does \( O(1) \) work in addition to the loop over edges. This loop does \( O(1) \) work for each edge leading into \( t \). Since ShortestPath gets called recursively at most once for each vertex, this happens at most once per edge of \( G \). Thus, the total runtime is at most \( O(|V| + |E|) \).