**R-10.6**

In book page 296, the Lemma 10.4 states that two characters that have smallest two weights are associated with nodes that have the maximum depth. Thus, similarly if all eight characters have the same frequency then they shall have the same depth either, which gives a complete binary tree since the number of characters is even.

**C-10.5**

Each time we give a quarter means two less dimes and one less nickel, or 5 less nickels or 25 less pennies. Same for dimes would save number of nickels and pennies, and nickels would save the number of pennies. Thus, if possible, always give as many quarters as possible, then dimes, nickels, and pennies.

**A-10.1**

Suppose the position X is ordered and x0 is the left-most painting. We can start from x0 and each time we found an un-guarded painting we place one guard to the right of that painting with a distance of 1. This ensures the current painting will be protected and let the guard covers largest range. We only need to scan though the set X once.

**R-11.1**

From Master Method,

1. T(n)=2T(n/2)+log n:

nlog2(2- ε) = n. log n is smaller than O(n^logb­a), Thus, from case 1 we can say T(n) is Θ(n) and also O(n).

1. T(n)=8T(n/2)+n^2:

n^logb­a=n^3.f(n)=n^2 is smaller than O(n^3). Thus, from case 1 T(n) is Θ(n^3) and also O(n^3).

1. T(n)=16T(n/2)+(nlogn)^4:

n^logb­a=n^4.f(n) is within case 2. Thus, from case 2 T(n) is Θ(n^4(logn)^4) and also O(n^4(logn)^4).

1. T(n)=7T(n/3)+n:

n^logb­a=n^(log37).f(n) is within case 1. Thus, T(n) is Θ(n^(log37)) and also O(n^(log37)).

1. T(n)=9T(n/3)+n3logn:

n^logb­a=n2.f(n) is within case 3. Thus, T(n) is Θ(f(n)), which is Θ(n3logn), and also O(n3logn).

**C-11.3**

n^logb­a = n ^(log23), which is case 1. f(n) is O(n^(log23-ε). For ε = 1, T(n) is Θ(n^(log3/23).

**A-11.1**

We can divide set S of n points into pairs, each pair is 2 elements which make a total of n/2 groups. Then we compare in each group and get max and min for n/2 time. Move all max numbers to a array and min numbers to another, then find max/min in those with (n/2)\*2 time. As a result we used 3n/2 running time.

**R-12.9**

The number of widgets n is the weight of the sack. Each bid can be seeing as an item of weight ki and value di. It’s obvious that this is a knapsack problem. Then if the bidders cannot accept any partial lots then this is a 0-1 problem. Else if they can accept that then this is a fractional problem.

A-12.3

This is a similar problem to LCS with dynamic-programming. Define a array W, where W[i] is true if for a substring Sj from S[j] to S[i], valid[Sj] returns true. Initially W should all be false. As we move i from 0 to n, within each move we check 0 to i to see if there is a valid word. After we scan through the string, we can get valid separated words between each true in W. Note that this may be confusing if we scan through characters like TODO, SUBWAY, which can be individual words to, do, sub, way or to-do, subway. I just have to believe a trained spy would avoid usage of such words.