Assignment 12 Solutions

Section 8.1:

16. Construct a 5-ary tree where an internal vertex represents a person that sent out five letters and a leaf represents a person that a received but did not send out a letter. So we know that there are 10,000 internal vertices. By theorem 4, there are a total of 5*10000+1= 50,001 vertices and (5-1)*10000+1=40,001 leaves. So assuming that the person that started the chain letter does not receive one, 0,000 people receive the letter and 40,001 people do not send it out.

    We shall prove this statement directly using a counting argument for the tree height, $h>=1$. Consider a full m-ary balanced tree with height h. Since it is balanced, every leaf is at height h or $h-1$, and every node that is not a leaf has m children. Consider the nodes at level $h-1$. They are either leaves or internal nodes. Since the tree is full and balanced, there must be m^{h-1} nodes with height h-1, some of which are leaves and others internal nodes. For each internal node with height h-1, there must be m leaves. Since the tree has height h, there must be at least m leaves with height h. Thus the total number of leaves must be strictly larger than the number of nodes with height h-1, i.e. the number of leaves must be more than m^{h-1}.
    An alternate argument is that from the corollary in the book, a full balanced m-ary tree has height h=ceiling($\log_m l$). So (h-1)< $\log_m l$ else if (h-1)=$\log_m l$, then h-1=ceiling($\log_m l$). Thus (h-1)< $\log_m l$ and raising both sides to m, we get $m^{h-1} < l$.

24.

Section 8.2:

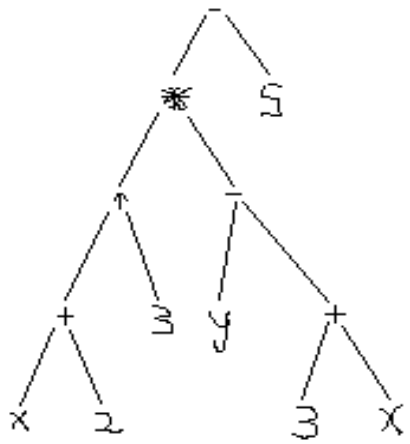
6. To find a lighter counterfeit coin among four coins, it would take no more than 2 weighings. Split the four coins into two pairs and weigh the first pair against the second. The lighter pair contains the counterfeit coin. Now with a second weighing one could find the lighter of the two coins from the lighter pair.

10. To find a counterfeit coin among four coins, if one exists, it would take no more than 3 weighings. Denote the four coins A,B,C,D. Weigh A against B. If they are not the same weight, record the heavier of the two and then weigh A against C. If they are the same weight, and A was heavier than B, then B is counterfeit and lighter, and if B was heavier than A, then B is counterfeit and heavier. If A and C are not the same weight, then A is counterfeit and since C is real, one may infer whether A is lighter or heavier. Now if A and B weigh the same, switch the roles of A and C and the roles of B and D in the above procedure. If C and D weigh the same there is no counterfeit coin and so we have obtained all the information in three weighings.

Section 8.3:

8. Preorder traversal: a, b, d, e, i, j, m, n, o, c, f, g, h, k, l, p
    Inorder traversal: d, b, i, e, m, j, n, o, a, f, c, g, k, h, p, l
    Postorder traversal: d, i, m, n, o, j, e, b, f, g, k, p, l, h, c, a

16.



30. Base case: If there is just one node, the statement is trivial.
IH: Assume that for a preorder list with n-1 entries and the number of children of each node given, the tree is completely specified.
IS: Given a preorder list with n entries, and the number of children of each node specified, find where the first leaf in the list and find a path pack to the root where each successive node is the first child of its parent. Remove the first leaf and apply the induction hypothesis.

32.Carrying out preorder traversals for each of the trees, we visit the vertices in the following order: a, b, c, d, e, f, g, h.

Section 8.4:

2.Applying bubble sort to the list *d,f,k,m,a,b*
First Pass: {*d,f,k,m,a,b*} -> {*d,f,k,a,m,b*} -> {*d,f,k,a,b,m*}
Second Pass: {*d,f,k,a,b,m*} -> {*d,f,a,k,b,m*} -> {*d,f,a,b,k,m*}
Third Pass: {d,f,a,b,k,m} -> {d,a,f,b,k,m} -> {d,a,b,f,k,m}
Fourth Pass: {*d,a,b,f,k,m*} -> {*a,d,b,f,k,m*} -> {*a,b,d,f,k,m*}

4.{4, 3, 2, 5, 1, 8, 7, 6} (L1) -> {4, 3, 2, 5}, {1, 8, 7, 6}
{4, 3, 2, 5} -> {4, 3}, {2, 5}
{1, 8, 7, 6} -> {1, 8}, {7, 6}
{4, 3} -> {4}, {3}
{2, 5} -> {2}, {5}
{1, 8} -> {1}, {8}
{7, 6} -> {7}, {6}

Now merge the lists.

First set of merges:
{7}, {6} -> {6, 7}
{1}, {8} -> {1, 8}
{2}, {5} -> {2, 5}
{4}, {3} -> {3, 4}

Second set of merges (Each step shown in order (List_1, List_2, Merged_List)):
{6, 7}, {1, 8}, {} -> {6, 7}, {8}, {1} -> {7}, {8}, {1, 6} -> {}, {8}, {1, 6, 7} -> {}, {}, {1, 6, 7, 8}
{2, 5}, {3, 4}, {} -> {5}, {3, 4}, {2} -> {5}, {4}, {2, 3} -> {5}, {}, {2, 3, 4} -> {}, {}, {2, 3, 4, 5}

Final set of merges (Each step shown in order (List_1, List_2, Merged_List)):
{1, 6, 7, 8}, {2, 3, 4, 5}, {} -> {6, 7, 8}, {2, 3, 4, 5}, {1} -> {6, 7, 8}, {3, 4, 5}, {1, 2} ->
{6, 7, 8}, {4, 5}, {1, 2, 3} -> {6, 7, 8}, {5}, {1, 2, 3, 4} -> {6, 7, 8}, {}, {1, 2, 3, 4, 5} ->
{}, {}, {1, 2, 3, 4, 5, 6, 7, 8}

6.a. This is the worst case since the smallest element alternates between the two lists each time through the loop. Thus the algorithm proceeds till the first list is empty taking 9 comparisons to merge the lists.

b. Note that every element in the first list is smaller than the smallest element in the second list. Thus after 5 comparisons the first list is empty and so we need 5 comparisons to merge these lists.

8. For each part of this problem, we can use a decision tree to determine the minimum number of comparisons. Given two sorted lists with m elements and n elements respectively, then after merging, the number of possible orderings of the final elements can be obtained by taking the m+n total slots available and choosing m of them for the elements of the first list. Thus the decision tree has C(m+n, m) leaves and the height of the decision tree gives the number of comparisons in the worst case, so the $\lceil \log_2 C(m+n, n) \rceil$ gives a lower bound on the number of comparisons needed.

   a. M=1, n= 4: $\lceil \log_2 C(5, 1) \rceil$ is 3, so at least 3 comparisons are needed. To achieve this bound, let the lists be represented by {A} and {B, C, D, E}. Compare A with C. If A is less than C, compare it to B and decide its position. If A is larger than C, compare it to D. If it is less than D, it belongs between C and D. If it is larger than D, compare it to E and decide its position. Note that this takes at most 3 comparisons and the approach we are using is a binary search approach.

   b. M=2, n=4: $\lceil \log_2 C(6, 2) \rceil$ is 4, so at least 4 comparisons are needed. However, there is no algorithm that can do the merge in 4 comparisons. To show this, we will consider the possible cases for the first comparison and then examine the size of the remaining decision tree in each case. Let the first list be {a,b} and the second be {c,d,e,f}. If we compare a to c first, then it could be that a>c. In this case, the problem becomes that of merging a list with 2 elements and one with 3 elements, which takes at least $\lceil \log_2 C(5, 2) \rceil = 4$, so 5 comparisons are needed in total. If we compare a to d first, then it could be that a<d. In this case, a may lie between c and d, with a total of 4 possible ending arrangements, or a may lie below c, with a total of 5 possible arrangements. Hence with a<d, there are 9 total arrangements left, so at

least 4 more comparisons are needed for a total of 5. If we compare a to e first and a<e, then reasoning as in the previous case there are 3+4+5=12 remaining possibilities for 4 more comparisons and a total of 5. Finally, if we compare a to f and a<f, then there are 2+3+4+5=14 possible arrangements and once again we'll have a total of 5 comparisons. Since the algorithm in the book uses 5 comparisons to merge these lists, 5 is the best possible.

c. $M=3, n=4: \lceil \log_2 C(7, 3) \rceil = 6$, so at least 6 comparisons are needed, and this number is achieved by the algorithm in the book.

d. $M=4, n=4: \lceil \log_2 C(8, 4) \rceil = 7$, so at least 7 comparisons are needed, and this number is achieved by the algorithm in the book.