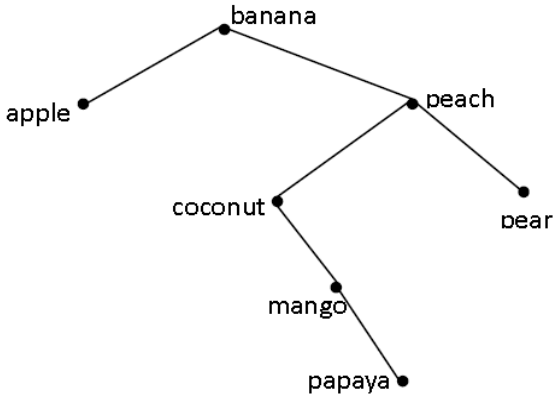
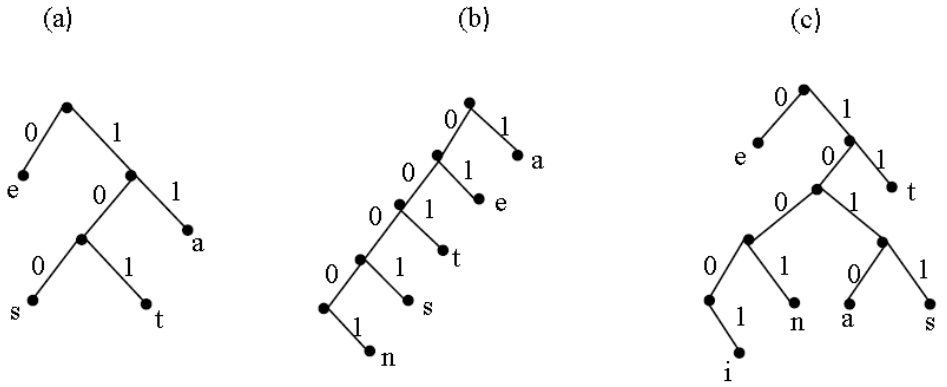
Sample Solutions on HW13 (8 exercises in total)

Sec. 11.2 1, 20, 23

1 We first insert *banana* into the empty tree, giving us the tree with just a root, labeled *banana*. Next we inset *peach*, which, being greater than *banana* in alphabetical order, becomes the right child of the root. We continue in this manner, letting each new word find its place by coming down the tree, branching either right or left until it encounters a free position. The final tree is as shown.



20 The constructions are straightforward.



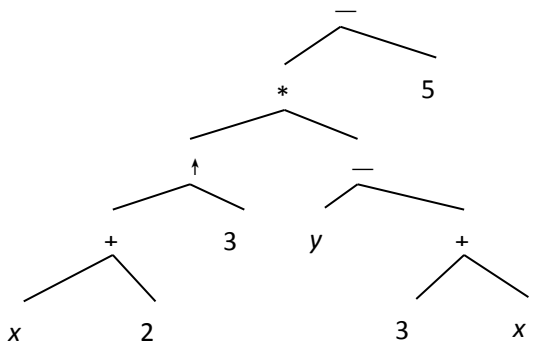
23 We follow Algorithm 2. Since *b* and *c* are the symbols of least weight, they are combined into a subtree, which we will call T_1 for discussion purpose, of weight 0.10+0.15=0.25, with the largest weight symbol, *c*, on the left. Now the two trees of smallest weight are the single symbol *a* and either T_1 or the single symbol *d* (both have weight 0.25). We break the tie arbitrarily in favor of T_1, and so we get a tree T_2 with left subtree T_1 and right subtree *a*. (If we had broken the tie in the other way, our final answer would have been different, but it would have been just as correct, and the average number of bits to encode a character would be the same.) The next step is to combine *e* and *d* into a subtree T_3 of weight 0.55. And the final step is to combine T_2

and T_3. The result is shown as below. We see by looking at the tree that a is encoded by 11, b by 101, c by 100, d by 01, and e by 00. To compute the average number of bits required to encode a character, we multiply the number of bits for each letter by the weight of that letter and add. Since a takes 2 bits and has weight 0.20, it contributes 0.40 to the sum. Similarly b contributes $3 \cdot 0.10 = 0.30$. In all we get $2 \cdot 0.20 + 3 \cdot 0.10 + 3 \cdot 0.15 + 2 \cdot 0.25 + 2 \cdot 0.30 = 2.25$. Thus on the average, 2.25 bits are needed per character. Note that this is an appropriately weighted average, weighted by the frequencies with which the letters occur.

Sec. 11.3 8, 16

8 In preorder, the root comes first, then list its subtrees, in preorder, from left to right. The answer is *a, b, d, e, i, j, m, n, o, c, f, g, h, k, l, p.*

16(a) We build the tree from the top down while analyzing the expression by identifying the outermost operation at each stage. The outmost operation in this expression is the final subtraction. Therefore the tree has – at its root, with the two operands as the subtrees at the root. The right operand is clearly 5, so the right child of the root is 5. The left operand is the result of a multiplication, so the left subtree has * as its root. We continue recursively in this way until the entire tree is constructed.



16(b) We can read off the answer from the picture we have just drawn simply by listing the vertices of the tree in preorder: First list the root, then the left subtree in preorder, then the right subtree in preorder. Therefore the answer is $-*\uparrow+x23-y+3x5$.

16(c) We can read off the answer from the picture we have just drawn simply by listing the vertices of the tree in postorder: $x2+3\uparrow y3x+-*5-$.

16(d) The infix expression is just the given expression, fully parenthesized: ((((*x*+2)↑3)*(*y*-(3+*x*)))-5). This corresponds to traversing the tree in inorder, putting in a left parenthesis whenever we go down to a left child and putting in a right parenthesis whenever we come up from a right child.