1. [5 points] Draw the result of applying the TreeDelete operation to the node labeled 25 in the following binary search tree:

```
  50
   / \
  25 55
    /
   10
   /  \
  4   15
   /
  40
   /
  48
      /
     32
      /
     30
```

2. [14 points] Given a sorted array $A$ of $n$ numbers, describe an algorithm to build a balanced binary search tree (i.e., a tree of height $\Theta(\log n)$) for elements in $A$. Analyze and provide the asymptotic time complexity of your algorithm.

3. [6 points] Assume that $T$ is a binary search tree with $n$ nodes and height $h$. Each node $x$ of $T$ stores a real number $x.Key$. Give the worst-case time complexity of the following algorithm Func1($T.root$). You need to justify your answer.

```
Func1(x) /* x is a tree node */
1  if (x == NIL) return (0);
2      s1 $\leftarrow$ Func1(x.Left()); /* x.Left() returns the left child of tree node x */
3      if (s1 < 100) then
4            s2 $\leftarrow$ Func1(x.Right()); /* x.Right() returns the right child of tree node x */
5        end
6      else
7          s2 $\leftarrow$ 0;
8      end
9  s $\leftarrow$ s1 + s2 + x.Key(); /* x.Key() returns the key value stored at tree node x */
10 return (s)
```
4. [5 points] Color the nodes of the following binary search tree so that it is a red-black tree:

```
  25
 /   \
20    35
 /   /   \
10   28   40
    /   /
   32  30
```

5. [10 points] Draw the result of applying RBTreeInsert to insert 48 in the following red-black tree. (Black nodes have double circles. Red nodes have dashed circles.) Indicate which nodes are red or black in your drawing. Be sure that the resulting tree has all the properties of a red-black tree. Show your work, i.e., show the effects of the different algorithm steps on the tree.

6. [15 points] Consider the following algorithm:

```plaintext
procedure Alg-X(A, p, q)
/* A[p, . . . , q] is an array of size q − p + 1 */
1 if q < p then
2     return (NIL) ;
3 end
4 new pointer root;
5 r = ⌊(q−p)/2⌋;
6 value = Selection (A, p, q, r) ;
7 Partition(A, p, q, value) ;
8 root.key = value;
9 root.left = Alg-X(A, p, r + p + 1);
10 root.right = Alg-X(A, r + p + 1, q);
11 return (root);
```

The procedure Selection (A, p, q, r) select the element with rank r (in increasing order) from the array A[p, . . . , q]. For example, Selection (A, p, q, 1) returns the smallest element from the array A[p, . . . , q]. It takes Θ(q − p + 1) worst-case running time. The procedure Partition(A, p, q, value) partitions the array A[p, . . . , q] so that the first part contains elements with value smaller than value, while the second part contains elements with value larger than value. (Same as the Partition procedure we used in QuickSort, but with value being the pivot value). It worst case running time is Θ(q − p + 1).
(a) Explain what procedure \( \text{Alg-X}(A, 1, n) \) does (its output).

(b) Give the time complexity analysis of the procedure \( \text{Alg-X}(A, 1, n) \) (first, give a recurrence, and then provide an asymptotic bound).

7. [10 points] Let \( T \) be a binary search tree with \( n \) nodes and height \( h \), where each node of \( T \) stores a “key” value (which is a real number). Now, we would like to augment the tree \( T \) so that each node \( x \in T \) stores

\[
x.\text{sum} = \text{the total sum of all keys stored in the subtree rooted at } x \text{ (including } x)\.
\]

Describe an algorithm that compute this augmentation to each tree node, and analyze the asymptotic time complexity of your algorithm.

8. [10 points] Let \( T \) be a binary search tree with \( n \) nodes and height \( \Theta(\log n) \), where each node \( x \) of \( T \) stores a \( x.\text{key} \) value (which is a real number), as well as an augmented value \( x.\text{sum} \) as introduced above in Problem 7.

Describe an algorithm \( \text{QueryPartialSum}(T, k_1, k_2) \) which answers the following query: Given two ranks \( k_1 \) and \( k_2 \), return the total sum of all keys in \( T \) whose rank falls in \([k_1, k_2]\).

For example, \( \text{QueryPartialSum}(T, 1, n) \) should return the sum of all keys stored in \( T \), and \( \text{QueryPartialSum}(T, 1, k) \) returns the sum of smallest \( k \) keys stored in \( T \).

You are NOT allowed to use extra data structure to aid you (say, build another array of size \( O(n) \)). You can only use the current tree structure – but you may augment it if you need to (although there are solutions without needing to do that).

Give the time complexity analysis of your algorithm. Slower but correct algorithm will receive partial credits.