Problem 1:

Draw the final binary search tree that is constructed by the following sequence of operations: 
Add(20), Add(5), Add(15), Add(30), Add(40), Add(25) Add(10), Add(12), Add(8), Add(28), Delete(20), 
Add(20).
Problem 2

Suppose that we modify the definition of the binary search tree so that, at each node \( N \) in addition to the value, it shows the size of each subtree under \( N \). The picture above shows an example.

A. Describe how the “Add” method must be modified to maintain this size field accurately. The procedure should continue to work with a running time proportional to the depth of the node being added.

**Answer:** When a node is added, trace your way back up the tree from the new leaf to the root, and add 1 to each count field.

If the tree is implemented without “parent” pointers, when a node is added, do a second search down from the root, and add 1 to the count field of every node you pass.

B. Describe how the “Delete” method must be modified to maintain this size field accurately. The procedure should continue to work with a running time proportional to the depth of the deepest of the nodes that are involved.

**Answer:** When all the dust has settled, trace your way back up the tree from the lowest node involved, and subtract 1 to each count field.

If the tree is implemented without “parent” pointers, when a node is added, do a second search down from the root, and subtract 1 to the count field of every node you pass.

C. Show how you can use this to implement the method \( \text{nth}(N) \) which returns the \( N \)th smallest element in the tree. Assume 0-based counting. For instance, in the above tree, \( T.\text{nth}(0) \) should return 1; \( T.\text{nth}(3) \) should return 6.
Node nth(N) {
    if (this is a leaf)
        if (N==0) return this;
        else return "No such node"
    else {
        if (left.size > N)
            return left.nth(N);
        else return right.nth(N-left.size);
    }
}

Problem 3

Suppose that you have a sorted array of integers of length N. Describe how you can construct a binary search tree of height log(N).

Answer: Let A be the array. The recursive routine BuildBST1(L,U) builds a balanced binary search tree for the part of the array from L to U by putting the middle element at the root, and then recursively calling BuildBST1 for the first half and for the second half.

```cpp
int A[]; // The array A is a global variable

BuildBST() {
    BuildBST1(0,A.length);
}

BuildBST1(L,U) {
    if (U < L) return null;
    M = (L+U)/2;
    create a node N labelled A[M];
    N.left = BuildBST1(L,M-1);
    N.right = BuildBST1(M+1,U)
    return N;
}
```
Honors Problem

Using the data structure in problem 2 above, describe how you can write a function “numBetween(L,U)” which gives the number of elements in the tree greater than L and less than U, and which runs in time proportional to the height of the tree. For instance, in the tree above, numBetween(6,17) should return 4, corresponding to the values 7, 10, 14, 15.

Answer:

0. If L==U return 0.
1. Find the path P from the root to L. \\
2. Find the path Q from the root to U. \\
3. Let W be the least common ancestor of L and U; that is, the lowest node that is on both P and Q.

   Note that one of the following is true:
   A. L is a descendent of the left child F of W and U is a descendent of the right child G of W.
   B. L is a descendent of the left child F of W and U==W.
   C. L == W and R is a descendent of the right child G of W.

4. sum = 0;
5. N = L;
6. while (N != W) {
   if (N is the left child of N.parent)
      sum = sum+ N.right.count + 1;
   N = N.parent;
  }
7. N = U;
8. while (N != W) {
   if (N is the right child of N.parent)
      sum = sum+ N.left.count + 1;
   N = N.parent;
  }
9. return sum-1; // W has been overcounted.