Exercises

1. (10 points) Draw a binary search tree after inserting the following integer values, in order: 15, 3, 19, 12, 8, 7, 2. Show the state of the tree after deleting 8 and 15.

Solution: Here are the two trees

2. (10 points) Add a constructor to the BSTree class to turn a sorted array of $N$ values into a BST that is as balanced as possible in $O(N)$ time. “Balanced as possible” means that at each internal node in the tree, the number of nodes in the left subtree and the number of nodes in the right subtree differ by at most 1. This is a stronger property than the AVL tree property. (Hint: think about which value should be stored at the root.) Your function should calculate the heights of the nodes. You do not need to prove that the function requires $O(N)$ time.

Solution: Here it is. The constructor acts as a driver for the recursive function SortedArrayToTree, which does most of the work.

```cpp
template <class Comparable >
BSTree<Comparable>::BSTree( const vector<Comparable >& sorted, bool balancing )
{
    root_ = SortedArrayToTree( sorted, 0, sorted.size()-1 );
    balanced_ = balancing;
}
```
template <class Comparable >
BSNode<Comparable> *
BSTree<Comparable>::SortedArrayToTree( const vector<Comparable >& sorted,
  int low, int high )
{
  if( low>high )
    return 0;
  else {
    int mid = (low+high)/2;
    BSNode<Comparable>* t =
      new BSNode<Comparable>( sorted[mid],
      SortedArrayToTree( sorted, low, mid-1 ),
      SortedArrayToTree( sorted, mid+1, high ) );
    Calculate_Height( t );
    return t;
  }
}