These are some review questions to test your understanding of the material. Some of these questions may appear on an exam.

1 Binary Search Trees

Please see the definition of `BinaryNode` on page 4 and of `BinarySearchTree` on page 3.

1.1 Define *binary search tree.*

1.2 Write valid `C++` code to complete the definition of the `find` member function of the `BinarySearchTree` class. The function takes two arguments:

`const Comparable & x` the element to search for

`BinaryNode<Comparable> * t` pointer to the node at which to start the search.

If `x` is found in the tree, a pointer to its node is returned; otherwise return `NULL` (see definition of the `BinarySearchTree` class on page 3).

```
template <class Comparable>
BinaryNode<Comparable> *
BinarySearchTree<Comparable>::
find( const Comparable & x, BinaryNode<Comparable> *t ) const
{
   // write the body of the method
}
```

1.3 Describe how deletion is performed in a binary search tree. Mention all relevant cases.

1.4 Explain: the number of comparisons needed to build a binary search tree of n elements is $O(n^2)$ worst case and $O(n \lg n)$ best case. Describe the best and worst cases.

1.5 For the following list of `C++` keyword strings:
`break, operator, if, typedef, else, case, while, do, return, unsigned, for, true, double, void`

1. Draw the binary search tree that results when the words are inserted into an initially empty tree in the order given. Show the tree after each insertion. Assume that `operator<` for strings means "alphabetical order."
2. Show the sequences of keywords under pre-order, post-order, and in-order traversals of the tree.

1.6 Given the following binary search tree:

```
          50
          /\
         /  \
        /    \
       /      \
     15        62
     /\        /\
    /  \      /  \
   5    20  58    91
  / \    \   \
 /   \    \   \
3     8    37  60
           /
          /
         24
```

Draw the binary search trees that result after each of the following operations or sequences of operations are performed. Each problem below is separate and begins with the tree shown above.

1. Insert 7
2. Insert 7, 1, 55, 29, and 19, in that order
3. Delete 8
4. Delete 8, 37, and 62, in that order
5. Insert 7, Delete 8, Insert 59, Delete 60, Insert 92, Delete 50, in that order

1.7 Suppose you have the following traversal sequence from a binary search tree:

pre-order: `L H D A F G K W R Q P Z`

Draw the tree.

1.8 Give an example of a binary search tree situation in which the immediate predecessor of some node is not in that node's left subtree.

1.9 Give an example of a binary search tree situation in which the immediate successor of some node is not in that node's right subtree.

1.10 Write pseudocode for finding the immediate predecessor of a node in a binary search tree.

1.11 Write pseudocode for finding the immediate successor of a node in a binary search tree.

1.12 Prove that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

Definition of `BinarySearchTree` Class

This definition is directly from the text.

```
template <class Comparable>
class BinarySearchTree
{
public:
  explicit BinarySearchTree( const Comparable & notFound );
  BinarySearchTree( const BinarySearchTree & rhs );
  ~BinarySearchTree( );

  const Comparable & findMin( ) const;
  const Comparable & findMax( ) const;
  const Comparable & find( const Comparable & x ) const;
  bool isEmpty( ) const;
  void printTree( ) const;

  void makeEmpty( );
  void insert( const Comparable & x );
  void remove( const Comparable & x );

  const BinarySearchTree & operator=( const BinarySearchTree & rhs );

private:
  BinaryNode<Comparable> *root;
  const Comparable ITEM_NOT_FOUND;

  const Comparable & elementAt( BinaryNode<Comparable> *t ) const;

  void insert( const Comparable & x, BinaryNode<Comparable> * & t ) const;
  //            void remove( const Comparable & x, BinaryNode<Comparable> * & t ) const;
  void remove( const Comparable & x, BinaryNode<Comparable> * & t );
  BinaryNode<Comparable> * findMin( BinaryNode<Comparable> *t ) const;
  BinaryNode<Comparable> * findMax( BinaryNode<Comparable> *t ) const;
  BinaryNode<Comparable> * find( const Comparable & x, BinaryNode<Comparable> *t ) const;
  void makeEmpty( BinaryNode<Comparable> * & t ) const;
  void printTree( BinaryNode<Comparable> *t ) const;
  BinaryNode<Comparable> * clone( BinaryNode<Comparable> *t ) const;
};
```

Definition of `Binarynode` Class

This definition is directly from the text.

```
template <class Comparable>
class BinaryNode
{
  Comparable element;
  BinaryNode *left;
  BinaryNode *right;

  BinaryNode( const Comparable & theElement,
              BinaryNode *lt, BinaryNode *rt )
    : element( theElement ), left( lt ), right( rt ) { }
  friend class BinarySearchTree<Comparable>;
};
```