1. Implement a Binary Search Tree as a template class. Your binary search tree must support the following operations. **Element** represents the type of the data component of each node and **Key** represents the type of the key value in the **Element**. You should assume that the **Element** type includes a member function **Key()** that returns the key value.

   ```cpp
   bool Empty(); // return true if empty; false o/w
   Element *Search(Key k); // locate first element with key k
   void Insert(Element *x); // insert x into the tree
   void Delete(Key k); // delete x from the tree
   
   Element *Maximum(); // return the maximum element
   Element *Minimum(); // return the minimum element
   
   Element *Successor(Key k); // return element’s successor
   Element *Predecessor(Key k); // return element’s predecessor
   
   void InOrder(); // print elements using an inorder traversal
   void PreOrder(); // print elements using a preorder traversal
   void PostOrder(); // print elements using a postorder traversal
   ```

2. A list can be sorted by inserting the elements into a binary search tree and then extracting them in an in-order traversal.

   (a) What is the worst case running time of this algorithm?
   (b) Implement this sorting algorithm and compare it to the other sorting algorithms you implemented.

3. Consider a binary search tree T whose keys are distinct. Prove that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x. (Recall that every node is its own ancestor.)

4. Let T be a binary search tree with distinct keys, let x be a leaf node, and let y be its parent. Prove that y.key is either the smallest key in T larger than x.key or the largest key in T smaller than x.key.