Exercise 1:

If we insert the entries (3, C), (1, A), (2, B), (5, E), and (4, D), in this order, into an initially empty binary search tree (BST), what will it look like?

Answer

![Binary Tree](image)

Figure 1: Binary Tree

Exercise 2:

We defined a binary search tree (BST) so that keys equal to a node’s key can be in either the left or right subtree of that node. Suppose we change the definition so that we restrict equal keys to the right subtree. What must a subtree of a binary search tree containing only equal keys look like in this case?

Answer

A subtree of equal keys must be a single chain of right children.
Exercise 3:

Insert, into an empty binary search tree, entries with keys 30, 40, 24, 58, 48, 26, 11, 13 (in this order). Draw the tree after each insertion.

Exercise 4:

How many different binary search trees can store the keys 1, 2, 3?

Answer

5 (2 w/ 1 as root, 1 w/ 2 as root, 2 w/ 3 as root).

Exercise 5:

One of your classmates claims that the order in which a fixed set of entries is inserted into a binary search tree does not matter, i.e., the same tree results every time. Give a small example that proves he is wrong.

Answer

There are several solutions. One is to draw the binary search tree created by the input sequence: 9, 5, 12, 7, 13. Now draw the tree created when you switch the 5 and the 7 in the input sequence: 9, 7, 12, 5, 13.

Exercise 6:

There are a huge variety of sorting algorithms which we could use, and it is sometimes hard to know which one one should use in any given situation.

To assist in the decision making process we can look at the real world run times of various different algorithms.

On elearning you will find implementations for a number of sorting algorithms, as well as code which will time how long it will take for the various implementations to run on some randomly generated lists.

Following this structure, implement the Bubble Sort, algorithm in the provided space and graph the results of the program, allowing a comparison of all the implemented algorithms.

Question: Despite the fact that all of the algorithms are either $O(n \log(n))$ or $O(n^2)$, they all show different real-world run times. In fact, for small lists, selection sort seems to perform better than more complicated sorts like merge sort and heap sort. Why might this be?
Exercise 7: (INFO1105 and INFO1905)

Write pseudocode for a method that performs a range query for a binary search tree. That is, the method should visit all items that have a search key in a given range of values (such as all values between 100 and 1000).

Answer

```java
public rangeQuery(node, low, high)
{
    if (node is null)
        return;
    // If the current node exceeds our search, look only at the left child.
    if (node > high) {
        rangeQuery(node.left, low, high);
    }
    // In range. Visit node and the query left and right subtrees.
    else if (node <= high && node >= low) {
        System.out.println(node);
        rangeQuery(node.left, low, high);
        rangeQuery(node.right, low, high);
    }
    // If the node is less than our search, look only at the right child.
    else if (node < low) {
        rangeQuery(node.right, low, high);
    }
}
```

Exercise 8: (INFO1905)

Duplicates in an ADT could mean either identical items, or more subtly, items that have identical search keys but with difference in other fields. If duplicates are allowed in a binary search tree, it is important to have a convention that determines the relationship between the duplicates. Items that duplicate the root of a tree should either all be in the left subtree or all be in the right subtree, and, of course, this property must hold for every subtree.

1. Why is this convention critical to the effective use of the binary search tree?

2. You can delete an item from a binary search tree by replacing it with the item whose search key either immediately follows or immediately precedes the search key of the item to be deleted. If duplicates are allowed, however, the choice between inorder successor and inorder predecessor is no longer arbitrary. How does the convention of putting duplicates in either the left or right subtree affect this choice?
Answer

Without a convention for placing duplicates in the left or right subtree, searching for duplicates would require that both subtrees be searched. If duplicates are placed in the left subtree, then a deleted internal node must be replaced by its inorder successor, and vice versa for the right subtree.

Exercise 9: (INFO1905)

Design a variation of algorithm TreeSearch for performing the operation getAll(k), which returns all the entries whose keys equal \( k \) in a binary search tree \( T \), and show that it runs in time \( O(h + s) \), where \( h \) is the height of \( T \) and \( s \) is the size of the collection returned.

Note: Assume duplicates are placed in the RIGHT subtree.

Algorithm 1 Recursive search in a binary search tree

1: procedure TREESEARCH(Tree \( T \), Key \( k \), node \( v \))
2: if \( T.isExternal(v) \) then
3: return \( v \)
4: end if
5: if \( k < \) key \((v)\) then
6: return TREESEARCH \((T, k, T.left(v))\)
7: else if \( k > \) key \((v)\) then
8: return TREESEARCH \((T, k, T.right(v))\)
9: end if
10: return \( v \)
11: end procedure

Answer

Note that after finding \( k \), if it occurs again, it will be in the left most internal node of the right subtree.

Algorithm findAllElements(k, v, c):
Input: The search key \( k \), a node of the binary search tree \( v \) and a collection \( c \)
Output: An iterable collection containing the found entries

if \( v \) is an external node then
    return \( c \)
if \( k = \) key \((v)\) then
    c.addLast \((v)\)
    return findAllElements \((k, T.right(v), c)\)
else if \( k < \) key \((v)\) then
    return findAllElements \((k, T.left(v), c)\)
else
    // {we know \( k > \) key \((v)\)}
    return findAllElements \((k, T.right(v), c)\)
Exercise 10: (INFO1105)

Implement in Java the method described in Question 7 that performs a range query.

Exercise 11: (INFO1905)

How many differently shaped, \( n \)-node binary trees are possible?

How many differently shaped, \( n \)-node binary search trees are possible

**Hint:** Write recursive definitions.

**Answer**

The number of structurally different binary trees (i.e. Binary Search Trees) is defined by the Catalan Formula.

\[
\frac{(2n)!}{(n+1)!n!}
\]  

(1)

To extend this to general binary trees, consider all permutations of the values stored in the nodes. The number of permutations of \( n \) elements is \( n! \), thus for a binary tree the solution is:

\[
\frac{(2n)!}{(n+1)!}
\]

(2)

We can define a simple recursive function to count how many structurally different trees there are:

```java
public static int countTrees(int n) {
    // There is only one possible tree with one node.
    if (n == 1)
        return 1;

    int sum = 0;
    int left, right, root = 1;

    // The root node always removes one node. Try all possible distributions
    // between the left and the right child nodes.
    for (left = n-1, right = 0; left != 0; left --, right ++) {
        int ltree = countTrees(left);
        int rtree = countTrees(right);

        // The number of trees with the nodes distributed this way is
        // ltree*rTree.
        sum += ltree * rTree;
    }

    return sum;
}
```
Exercise 12: (INFO1905)

A binary search tree with a given set of data items can have several different structures that conform to the definition of a binary search tree. If you are given a list of data items, does at least one binary search tree whose preorder traversal matches the order of the items on your list always exist? Is there ever more than one binary search tree that has the given preorder traversal.