Name: Yiling Jiang
Email: yiling124@gmail.com
Problem Set 4 - Part I

Problem 1:
1, Breadth - First Search (BFS):

a , b, c, d , e, f

2, Depth-first search (DFS):

a, b, d, e, c, f

3, Iterative-deepening search (IDS):

a, a, b, a, c, a, b , d

Problem 2:
1, Binary Tree
In best case scenario, the root key equals the given key, and it takes O(1) to find the node and return result. So, in best case: time complexity is O(1).

If this is a balanced tree: In worst case scenario, we need to go through all the nodes and the give key equals to the last key we goes through or the given key does not exist in the tree, in these cases we need to go through n nodes to find the result. So, in worst case: time complexity is O(n).

If this is *not a balanced tree*: In worst case scenario, we need to go through all the nodes just like the case when it's a balanced tree. The give key equals to the last key we goes through, in this case we need to go through n nodes to find the result. So, in worst case: time complexity is O(n).

2,
```
public class Solution{
  public int depth(int key){
    if(root == null){
      throw new IllegalStateException("the tree is empty");
    }

    return depthInTree(key, root);
```

```
 }

 private static int depthInTree(int key, Node root){
  if (key == root.key){
   return 0;
  }
  if (root == null) return -1;

  if (key < root.key) {
   int depthInLeft = depthInTree(key, root.left);
   if (depthInLeft != -1) {
    return depthInLeft + 1;
   }
  }

  if (key > root.key) {
   int depthInRight = depthInTree(key, root.right);
   if (depthInRight != -1) {
    return depthInRight + 1;
   }
  }
  return -1;
 }
}
```

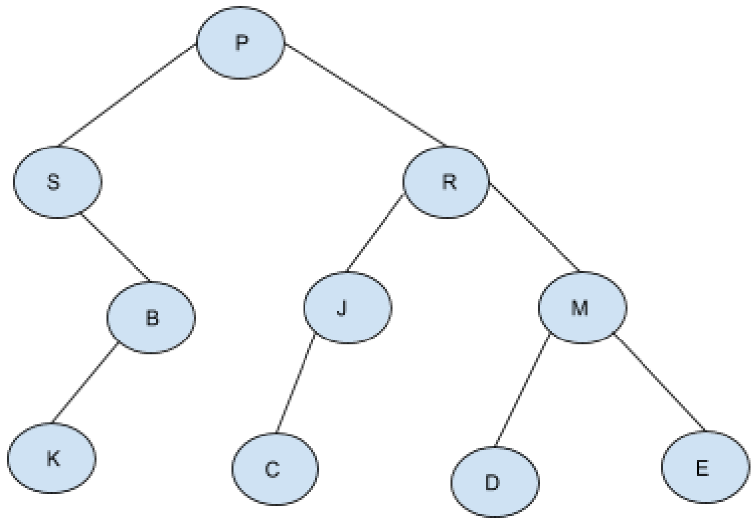
3, Binary Search Tree:
In *best case scenario*, the root key equals the given key, and it takes O(1) to find the node and return result. So, in best case: time complexity is O(1).

If this is a *balanced tree*: Due to the fact that this is a binary search tree, all nodes(keys) on the left subtree are smaller than the root key, and all the nodes(keys) on the right subtree are larger than the root key. Each time by comparing the target key with the current root key, we can exclude half of the tree nodes to decrease the amount of nodes to be searched. Eventually we will narrow down the searching scope to one node. In this case, it will only take O(lg n) to find the target key

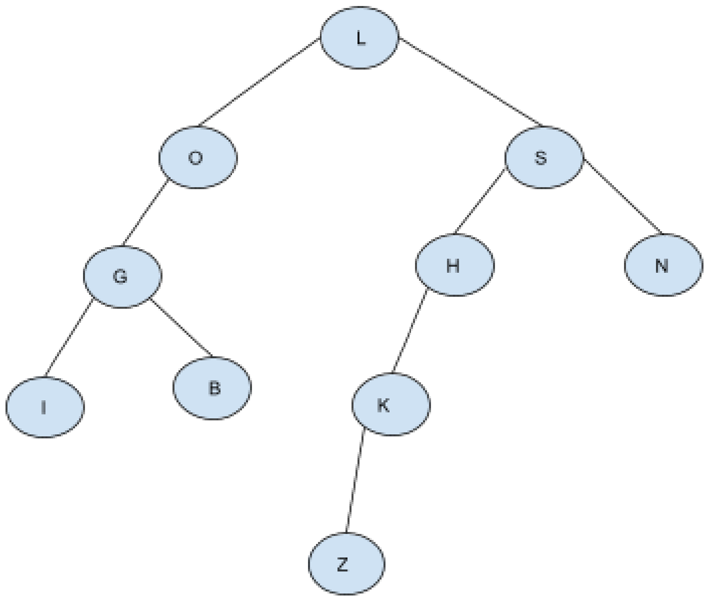
If this is *not a balanced tree*: In worst case scenario, on every level of the tree there is one and only one single node. In other words, the tree's height is equal to n. In worst case, we need to go through all the nodes and the give key equals to the last key we goes through, in this case we need to go through n nodes to find the result. So, in worst case: time complexity is O(n).

Problem 3: Tree traversal puzzles

1,
When a binary tree of characters (which is *not* a binary *search* tree) is listed in inorder, the result is SKBPCJRDME. Preorder traversal gives PSBKRJCMDE. Construct the tree.

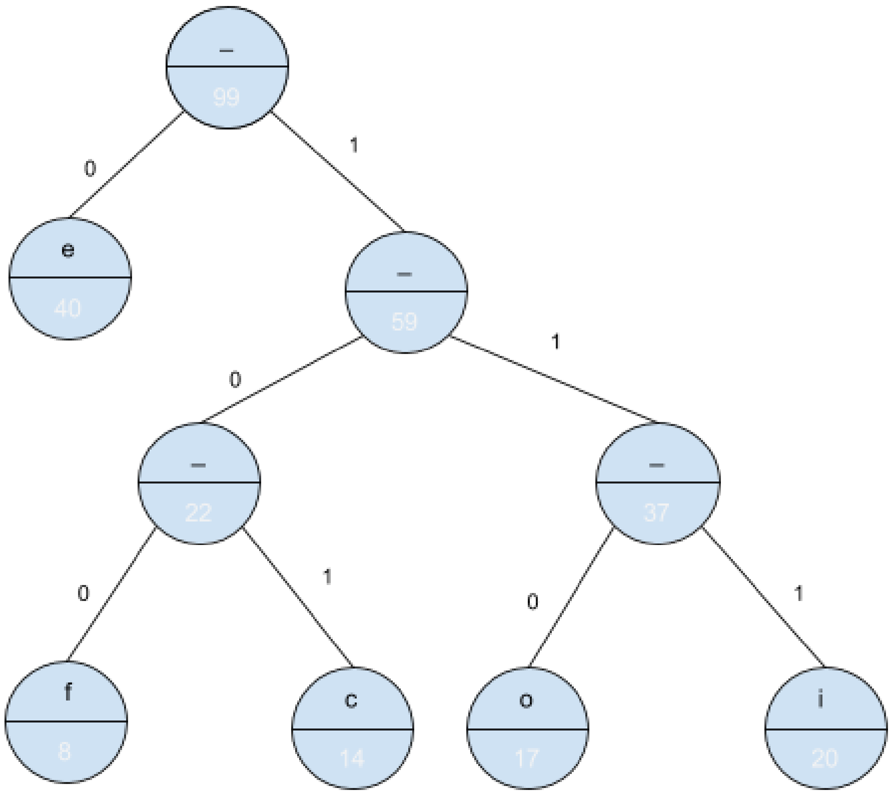


2,
When a binary tree of characters (which is *not* a binary *search* tree) is listed in postorder, the result is IBGOZKHNSL. Preorder gives LOGIBSHKZN. Construct the tree. (There is more than one possible answer in this case.)



Problem 4: Huffman encoding

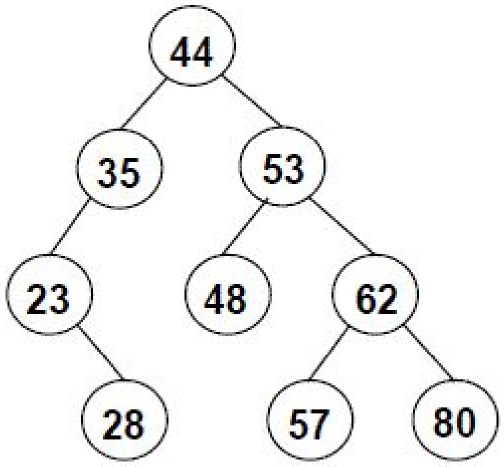
1,



2, Using the Huffman tree from part 1, encoding of the string *office* :
1101001001111010

Problem 5: Binary search trees

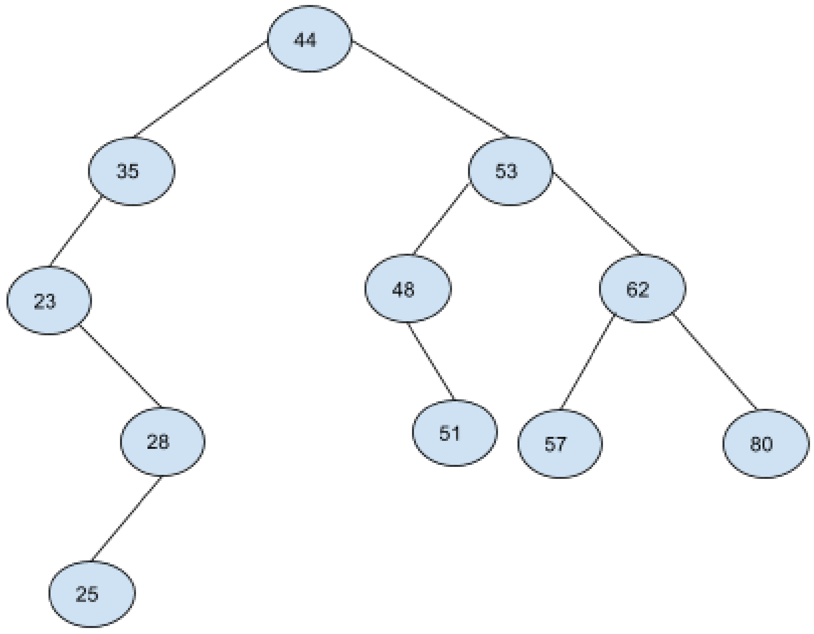
Consider the following binary search tree, in which the nodes have the specified integers as keys:



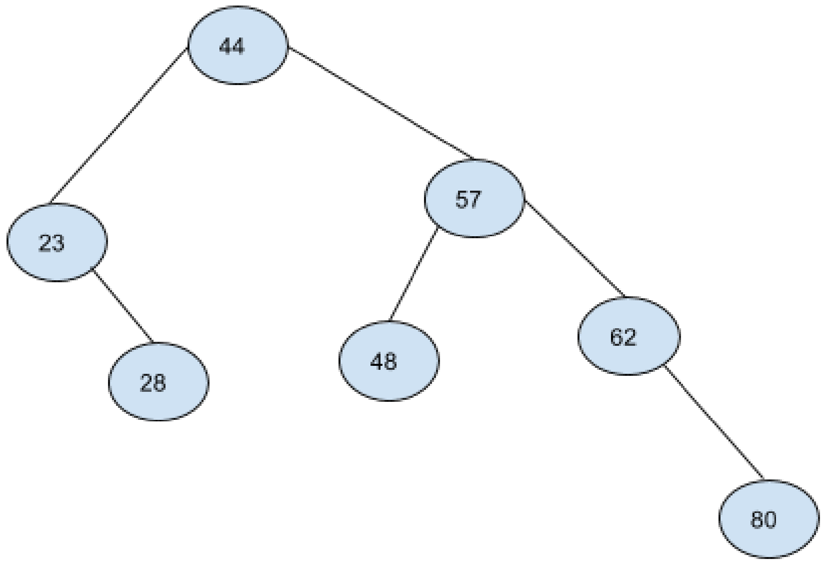
1, Preorder traversal:
44 - 35 - 23 - 28 - 53 - 48 - 62 - 57 - 80

2, Postorder traversal:
28 - 23 - 35 - 48 - 57 - 80 - 62 - 53 - 44

3, 25 is inserted, followed by 51



4, 53 is deleted and then 35 is deleted



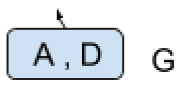
5,

A **binary tree** is **balanced** if for any two leaves the difference of the depth is at most 1.
Based on the definition of balanced tree, this original tree is balanced, as the difference of depth between any two leaves is equal or smaller than 1.

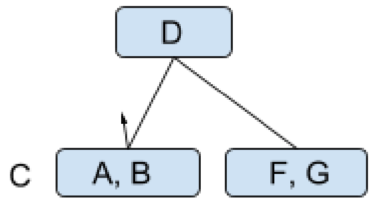
Problem 6:

1, 2- 3 tree :

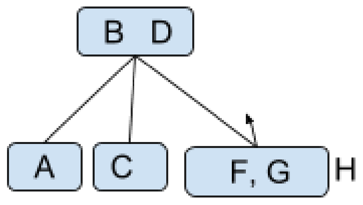
1:



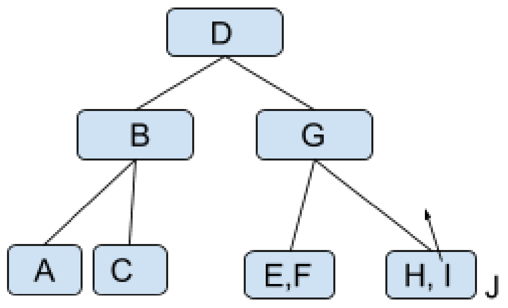
2 :



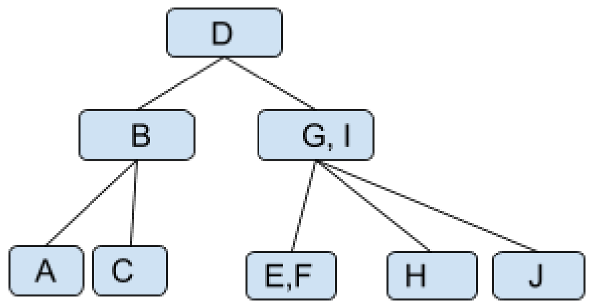
3 :



4 :

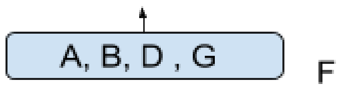


5 :

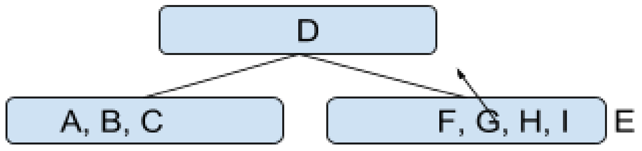


2, B - tree :

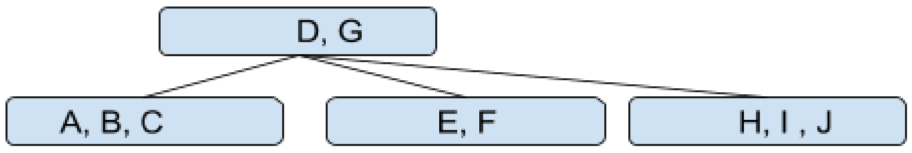
1:



2:



3:


D, G
A, B, C
E, F
H, I , J