Polytechnic Tutoring Center

Final Exam Review - CS 1134 Answer Key, Spring 2020

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

1 Fill in the runtimes for the following table.

Data Structure	Insert	Search	Delete
Array	**O(n)**	**O(n)**	**O(n)**
Stack	**O(1)**	**O(n)**	**O(1)**
Queue	**O(1)**	**O(n)**	**O(1)**
Binary Search Tree	**O(h)**	**O(h)**	**O(h)**
Binary Heap	**O(logn)**	**O(n)**	**Min/Max: O(logn)**
Hashtable	**O(1)**	**O(1)**	**O(1)**

2 The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10. Using the hash function h(k) = k mod 10, draw the hash table using:

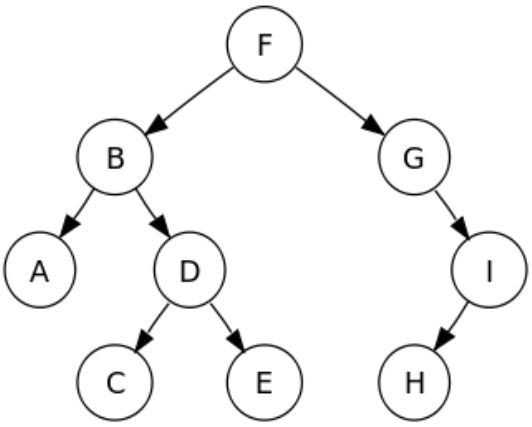
1.Chanining Approach

2. Open Addressing with linear Probing

0	
1	
2	12, 2
3	13, 3, 23
4	
5	5, 15
6	
7	
8	18
9	

0	
1	
2	12
3	13
4	2
5	3
6	23
7	5
8	18
9	15

3 Given the following tree write the preorder, postorder, and inorder traversal of nodes.



Preorder : F, B, A, D, C, E, G, I, H

Inorder : A, B, C, D, E, F, G, H, I

Postorder : A, C, E, D, B, H, I, G, F

1. What is the runtime of the traversals?

O(n)

2. Draw out the BST based on the following information:

Preorder: 15 8 11 9 20 17 16

Inorder: 8 9 11 15 16 17 20

```
      15
    /    \
   8      20
    \     /
    11   17
   /    /
  9    16
```

4 Write code to Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists

```
def mergeTwoLists(l1,l2):
    if l1 == None and l2 == None:
        return None
    elif l1 != None and l2 == None:
        return l1
    elif l2 != None and l1 == None:
        return l2
    else:
        if l1.val < l2.val:
```

```
            l1.next = self.mergeTwoLists(l1.next, l2)
            return l1
        elif l1.val > l2.val:
            l2.next = self.mergeTwoLists(l1,l2.next)
            return l2
        else:
            l2.next = self.mergeTwoLists(l1.next,l2.next)
            l1.next = l2
            return l1
```

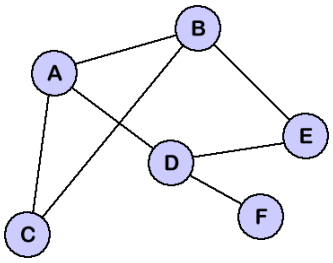
5 Given a non-empty binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

Symmetric tree examples:

Code:

```
def is_symmetric(root):
    return helperMirror(root, root)
def helperMirror(l, r):
    if l == r == None:
        return True
    if l == None or r == None:
        return False
    return l.val == r.val and helperMirror(l.right, r.left) and
helperMirror(l.left, r.right)
```

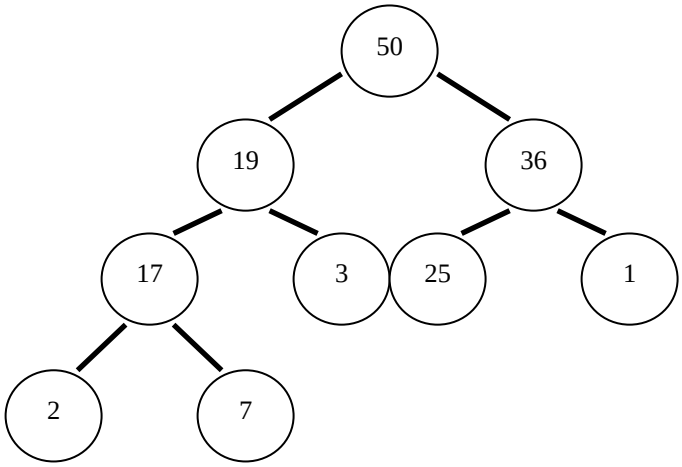
6 Write the BFS and DFS going alphabetically when given multiple options.



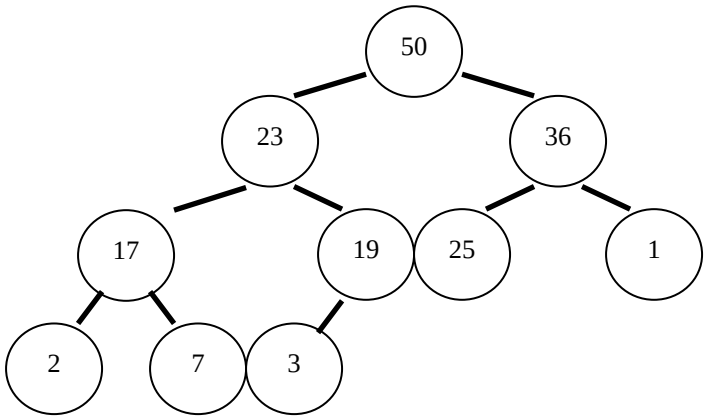
BFS: A, B, C, D, E, F

DFS: A, B, E, D, F, C

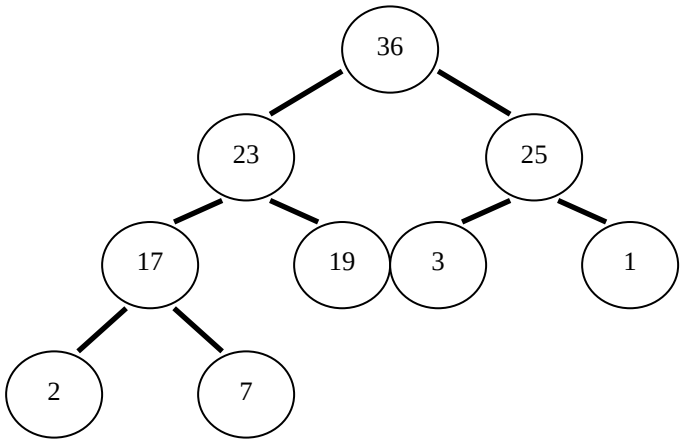
7 Draw the corresponding max-heap from the given list: [50,19,36,17,3,25,1,2,7]



8 Using the above heap draw what happens step by step when you insert 23.



9 Using the above heap draw what happens step by step when you delete max.



10 Write a post-order traversal generator for a tree data structure class.

Code:

```
def post_order(node):
   if node.left:
        yield from post_order(node.left)
   if node.right:
        yield from post_order(node.right)
   yield node.data
```

11 Given a binary search tree and the lowest and highest boundaries as L and R, trim the tree so that all its values lies in the interval [L, R] (R >= L). You might need to change the root of the tree, so the result should return the new root of the trimmed binary search tree.

Code:

```
def trimBST(root,L,R):
    if root == None: return root

        else:

            curr_r = root.val <= R and root.val >= L

            if root.val >= R:

                root.left = trimBST(root.left,L,R)

                root.right = None

            elif root.val <= L:

                root.left = None

                root.right = trimBST(root.right,L,R)

            else:

                root.left = trimBST(root.left,L,R)

                root.right = trimBST(root.right,L,R)

            if curr_r:

                return root

            else:

                if root.left == None and root.right == None:
```

```
        return None
    else:
        if root.left == None:
            return root.right
        return root.left
```