21. How do you print duplicate characters from a string?

```
NO_OF_CHARS = 256

def fillCharCounts(string, count):

    for i in string:

        count[ord(i)] += 1

    return count

def printDups(string):

    count = [0] * NO_OF_CHARS

    count = fillCharCounts(string,count)

    k = 0

    for i in count:

        if int(i) > 1:

            print chr(k) + ", count = " + str(i)

        k += 1

string = "test string"

print printDups(string)
```

22. How do you check if two strings are anagrams of each other?

```
class Solution:

    def isAnagram(self, a, b):

        if sorted(a) == sorted(b):

            return True

        else:

            return False

if __name__ == '__main__':
    a = "gram"
    b = "arm"
    if(Solution().isAnagram(a, b)):
      print("The two strings are anagram of each other")
    else:
      print("The two strings are not anagram of each other")
```

23. How do you print the first non-repeated character from a string?

```
string = "geeksforgeeks"
index = -1
fnc = ""
for i in string:
    if string.count(i) == 1:
        fnc += i
        break
    else:
        index += 1
if index == 1:
    print("Either all characters are repeating or string is empty")
else:
    print("First non-repeating character is", fnc)
```

24. How can a given string be reversed using recursion?

```
def reverse(string):
    if len(string) == 0:
        return
    temp = string[0]
    reverse(string[1:])
    print(temp, end='')
string = "Geeks for Geeks"
reverse(string)
```

25. How do you check if a string contains only digits?

```
MAX = 10
def isDigit(ch):
    ch = ord(ch)
    if (ch >= ord('0') and ch <= ord('9')):
        return True
    return False
def allDigits(st, le):
    present = [False for i in range(MAX)]
    for i in range(le):
        if (isDigit(st[i])):
            digit = ord(st[i]) - ord('0')
            present[digit] = True
    for i in range(MAX):
        if (present[i] == False):
            return False
    return True
```

```
st = "Geeks12345for69708"
le = len(st)
if (allDigits(st, le)):
    print("Yes")
else:
    print("No")
```

26. How are duplicate characters found in a string?

```
def findDuplicate(str1, N):
    first = 0
    second = 0
    for i in range(N):
        if (first & (1 << (ord(str1[i]) - 97)))
            second = second | (1 << (ord(str1[i]) - 97))
        else:
            first = first | (1 << (ord(str1[i]) - 97))
    for i in range(26):
        if ((first & (1 << i)) and (second & (1 << i))):
            print(chr(i + 97), end = " ")
if __name__ == '__main__':
    str1 = "geeksforgeeks"
    N = len(str1)
    findDuplicate(str1, N)
```

27. How do you count the number of vowels and consonants in a given string?

```
def countCharacterType(str):
    vowels = 0
    consonant = 0
    specialChar = 0
    digit = 0
    for i in range(0, len(str)):
        ch = str[i]
        if ( (ch >= 'a' and ch <= 'z') or
             (ch >= 'A' and ch <= 'Z') ):
            ch = ch.lower()
            if (ch == 'a' or ch == 'e' or ch == 'i'
                        or ch == 'o' or ch == 'u'):
                vowels += 1
            else:
                consonant += 1
        elif (ch >= '0' and ch <= '9'):
            digit += 1
        else:
            specialChar += 1
    print("Vowels:", vowels)
    print("Consonant:", consonant)
    print("Digit:", digit)
    print("Special Character:", specialChar)
str = "geeks for geeks121"
```

```
countCharacterType(str)
```

28. How do you count the occurrence of a given character in a string?

```
def count(s, c) :
    res = 0
    for i in range(len(s)) :
        if (s[i] == c):
            res = res + 1
    return res
str= "geeksforgeeks"
c = 'e'
print(count(str, c))
```

29. How do you find all the permutations of a string?

```
def toString(List):
    return ''.join(List)
def permute(a, l, r):
    if l==r:
        print (toString(a))
    else:
        for i in range(l,r):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l] # backtrack
string = "ABC"
```

```
n = len(string)

a = list(string)

permute(a, 0, n)
```

30. How do you reverse words in a given sentence without using any library method?

```
def reverse_word(s, start, end):

    while start < end:

        s[start], s[end] = s[end], s[start]

        start = start + 1

        end -= 1

s = "i like this program very much"

s = list(s)

start = 0

while True:

    try:

        end = s.index(' ', start)

        reverse_word(s, start, end - 1)

        start = end + 1

    except ValueError:

        reverse_word(s, start, len(s) - 1)

        break

s.reverse()

s = "".join(s)

print(s)
```

31. How do you check if two strings are a rotation of each other?

```
def checkString(s1, s2, indexFound, Size):
    for i in range(Size):
        if(s1[i] != s2[(indexFound + i) % Size]):
            return False
    return True
s1 = "abcd"
s2 = "cdab"
if(len(s1) != len(s2)):
    print("s2 is not a rotation on s1")
else:
    indexes = []  # store occurrences of the first character of s1
    Size = len(s1)
    firstChar = s1[0]
    for i in range(Size):
        if(s2[i] == firstChar):
            indexes.append(i)
    isRotation = m
    for idx in indexes:
        isRotation = checkString(s1, s2, idx, Size)
        if(isRotation):
            break
    if(isRotation):
        print("Strings are rotations of each other")
    else:
        print("Strings are not rotations of each other")
```

32. How do you check if a given string is a palindrome?

```
def isPalindrome(string):
    l = 0
    h = len(string) - 1
    while h > l:
        l+= 1
        h-= 1
        if string[l-1] != string[h + 1]:
            return False
    return True
def isRotationOfPalindrome(string):
    if isPalindrome(string):
        return True
    n = len(string)
    for i in range(n-1):
        string1 = string[i + 1:n]
        string2 = string[0:i + 1]
        string1+=(string2)
        if isPalindrome(string1):
            return True
    return False
print ("1" if isRotationOfPalindrome("aab") == True else "0")
print ("1" if isRotationOfPalindrome("abcde") == True else "0")
print ("1" if isRotationOfPalindrome("aaaad") == True else "0")
```

33. How is a binary search tree implemented?

```
def search(root,key):
    if root is None or root.val == key:
        return root
    if root.val < key:
        return search(root.right,key)
    return search(root.left,key)
```

34. How do you perform preorder traversal in a given binary tree?

```
class Node():
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def getPreIndex():
    return constructTreeUtil.preIndex
 def incrementPreIndex():
    constructTreeUtil.preIndex += 1
def constructTreeUtil(pre, low, high):
    if(low > high):
        return None
    root = Node(pre[getPreIndex()])
    incrementPreIndex()
    if low == high:
        return root
    r_root = -1
    for i in range(low, high+1):
        if (pre[i] > root.data):
            r_root = i
            break
     if r_root == -1:
        r_root = getPreIndex() + (high - low)
    root.left = constructTreeUtil(pre, getPreIndex(), r_root-1)
    root.right = constructTreeUtil(pre, r_root, high)
def constructTree(pre):
    size = len(pre)
    constructTreeUtil.preIndex = 0
    return constructTreeUtil(pre, 0, size-1)
def printInorder(root):
    if root is None:
        return
    printInorder(root.left)
    print (root.data,end=' ')
    printInorder(root.right)
pre = [10, 5, 1, 7, 40, 50]
root = constructTree(pre)
print ("Inorder traversal of the constructed tree:")
printInorder(root)
```

35. How do you traverse a given binary tree in preorder without recursion?

```
class newNode():
    def __init__(self, key):
        self.key = key
        self.child =[]
def traverse_tree(root):
    nodes=[]
    nodes.append(root)
    while (len(nodes)):
        curr = nodes[0]
        nodes.pop(0)
        print(curr.key,end=" "
        for it in range(len(curr.child)-1,-1,-1):
            nodes.insert(0,curr.child[it])
if __name__ == '__main__':
    root = newNode('A')
    (root.child).append(newNode('B'))
    (root.child).append(newNode('F'))
    (root.child).append(newNode('D'))
    (root.child).append(newNode('E'))
    (root.child[0].child).append(newNode('K'))
    (root.child[0].child).append(newNode('J'))
    (root.child[2].child).append(newNode('G'))
    (root.child[3].child).append(newNode('C'))
    (root.child[3].child).append(newNode('H'))
    (root.child[3].child).append(newNode('I'))
    (root.child[0].child[0].child).append(newNode('N'))
    (root.child[0].child[0].child).append(newNode('M'))
    (root.child[3].child[0].child).append(newNode('O'))
    (root.child[3].child[2].child).append(newNode('L'))
    traverse_tree(root)
```

36. How do you perform an inorder traversal in a given binary tree?

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def inOrder(root):
    current = root
    stack = [] # initialize stack
    while True:
        if current is not None:
            stack.append(current)
            current = current.left
        elif(stack):
            current = stack.pop()
            print(current.data, end=" ") # Python 3 printing
            current = current.right
        else:
            break
    print()
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
inOrder(root)
```

37. How do you print all nodes of a given binary tree using inorder traversal without recursion?

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def inOrder(root):
    current = root
    stack = []
    while True:
        if current is not None:
            stack.append(current)
            current = current.left
        elif(stack):
            current = stack.pop()
            print(current.data, end=" ") # Python 3 printing
            current = current.right
        else:
            break
    print()
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
inOrder(root)
```

38. How do you implement a postorder traversal algorithm?

```
INT_MIN = -2**31
INT_MAX = 2**31
def findPostOrderUtil(pre, n, minval,
                      maxval, preIndex):
    if (preIndex[0] == n):
        return
    if (pre[preIndex[0]] < minval or
            pre[preIndex[0]] > maxval):
        return
     val = pre[preIndex[0]]
    preIndex[0] += 1
    findPostOrderUtil(pre, n, minval,
                  val, preIndex)
     findPostOrderUtil(pre, n, val,
                  maxval, preIndex)
    print(val, end=" ")
def findPostOrder(pre, n):
    preIndex = [0]
    findPostOrderUtil(pre, n, INT_MIN,
                  INT_MAX, preIndex)
if __name__ == '__main__':
    pre = [40, 30, 35, 80, 100]
    n = len(pre)
    findPostOrder(pre, n)
```

39. How do you traverse a binary tree in postorder traversal without recursion? How are all leaves of a binary search tree printed?

```
class newNode:
     def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def postorder(head):
    temp = head
    visited = set()
    while (temp and temp not in visited):
        if (temp.left and temp.left not in visited):
            temp = temp.left
        elif (temp.right and temp.right not in visited):
            temp = temp.right
        else:
            print(temp.data, end = " ")
            visited.add(temp)
            temp = head
if __name__ == '__main__':
    root = newNode(8)
    root.left = newNode(3)
    root.right = newNode(10)
    root.left.left = newNode(1)
    root.left.right = newNode(6)
    root.left.right.left = newNode(4)
    root.left.right.right = newNode(7)
    root.right.right = newNode(14)
    root.right.right.left = newNode(13)
    postorder(root)
```

40. How do you count the number of leaf nodes in a given binary tree? How do you perform a binary search in a given array?

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
def getLeafCount(node):
    if node is None:
        return 0
    if(node.left is None and node.right is None):
        return 1
    else:
        return getLeafCount(node.left) + getLeafCount(node.right)
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print ("Leaf count of the tree is %d" %(getLeafCount(root)))
```