1. How do you check if two strings are a rotation of each other?
   Solution:

```
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 = []
        Size = len(s1)
        firstChar = s1[0]
        for i in range(Size):
                if(s2[i] == firstChar):
                        indexes.append(i)
        isRotation = False
        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")
```

2. How do you check if a given string is a palindrome?
   Solution:

```
def isPalindrome(str):
        for i in range(0, int(len(str)/2)):
                if str[i] != str[len(str)-i-1]:
                        return False
        return True
s = "malayalam"
ans = isPalindrome(s)
```

```
if (ans):
        print("Yes")
else:
        print("No")
```

3. How is a binary search tree implemented?
   Solution :

```
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)
```

4. How do you perform preorder traversal in a given binary tree?
   Solution:

```
class Node:
        def __init__(self, key):
                self.left = None
                self.right = None
                self.val = key
def printPreorder(root):
        if root:
        print(root.val),
                printPreorder(root.left)
                printPreorder(root.right)
if __name__ == "__main__":
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print "Preorder traversal of binary tree is"
printPreorder(root)
```

5. How do you traverse a given binary tree in preorder without recursion?
   Solution:

```
class Node:
        def __init__(self, data):
                self.data = data
                self.left = None
                self.right = None
def iterativePreorder(root):
        if root is None:
                return
        nodeStack = []
        nodeStack.append(root)
        while(len(nodeStack) > 0):
                node = nodeStack.pop()
                print (node.data, end=" ")
                if node.right is not None:
                        nodeStack.append(node.right)
                if node.left is not None:
                        nodeStack.append(node.left)
root = Node(10)
root.left = Node(8)
root.right = Node(2)
root.left.left = Node(3)
root.left.right = Node(5)
root.right.left = Node(2)
iterativePreorder(root)
```

6. How do you perform an inorder traversal in a given binary tree?
   Solution:

```
class Solution(object):
    def inorderTraversal(self, root):
        res = []
         if root:
        self.inorderTraversal(root.left)
        res.append(root.val)
```

```
            self.inorderTraversal(root.right)
        return res
```

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

```
class Node:
    def __init__(self, data=None, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right
def inorder(root):
    if root is None:
        return
    inorder(root.left)
    print(root.data, end=' ')
    inorder(root.right)
if __name__ == '__main__':
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.right.left = Node(5)
    root.right.right = Node(6)
    root.right.left.left = Node(7)
    root.right.left.right = Node(8)
    inorder(root)
```

8. How do you implement a postorder traversal algorithm?
   Solution:

```
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)
```

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

```
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)
```

10. 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?
    Solution:

```
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)))
```

11. How is a bubble sort algorithm implemented?
    Solution:

```
def bubbleSort(arr):
        n = len(arr)
        swapped = False
        for i in range(n-1):
                for j in range(0, n-i-1):
                        if arr[j] > arr[j + 1]:
                                swapped = True
                                arr[j], arr[j + 1] = arr[j + 1], arr[j]

                if not swapped:
                        return
```

```
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)
print("Sorted array is:")
for i in range(len(arr)):
        print("% d" % arr[i], end=" ")
```

12. How is an iterative quicksort algorithm implemented? How do you implement an insertion sort algorithm?
    Solution:

```
from collections import deque
def swap (A, i, j):
    temp = A[i]
    A[i] = A[j]
    A[j] = temp
def partition(a, start, end):
    pivot = a[end]
    pIndex = start
    for i in range(start, end):
        if a[i] <= pivot:
            swap(a, i, pIndex)
            pIndex = pIndex + 1
    swap(a, pIndex, end)
    return pIndex
def iterativeQuicksort(a):
    stack = deque()
    start = 0
    end = len(a) - 1
    stack.append((start, end))
    while stack:
        start, end = stack.pop()
        pivot = partition(a, start, end)
        if pivot - 1 > start:
            stack.append((start, pivot - 1))
        if pivot + 1 < end:
            stack.append((pivot + 1, end))
if __name__ == '__main__':
    a = [9, -3, 5, 2, 6, 8, -6, 1, 3]
    iterativeQuicksort(a)
     print(a)
```

13.How do you implement a bucket sort algorithm?

Solution:

```
def insertionSort(b):
        for i in range(1, len(b)):
                up = b[i]
                j = i - 1
                while j >= 0 and b[j] > up:
                        b[j + 1] = b[j]
                        j -= 1
                b[j + 1] = up
        return b
def bucketSort(x):
        arr = []
        slot_num = 10
        for i in range(slot_num):
                arr.append([])
        for j in x:
                index_b = int(slot_num * j)
                arr[index_b].append(j)
        for i in range(slot_num):
                arr[i] = insertionSort(arr[i])
        k = 0
        for i in range(slot_num):
                for j in range(len(arr[i])):
                        x[k] = arr[i][j]
                        k += 1
        return x

x = [0.897, 0.565, 0.656,0.1234, 0.665, 0.3434]
print("Sorted Array is")
print(bucketSort(x))
```

14.How is a radix sort algorithm implemented?

Solution:

```
def countingSort(arr, exp1):
        n = len(arr)
        output = [0] * (n)
```

```
        count = [0] * (10)
        for i in range(0, n):
                index = arr[i] // exp1
                count[index % 10] += 1
        for i in range(1, 10):
                count[i] += count[i - 1]
        i = n - 1
        while i >= 0:
                index = arr[i] // exp1
                output[count[index % 10] - 1] = arr[i]
                count[index % 10] -= 1
                i -= 1
        i = 0
        for i in range(0, len(arr)):
                arr[i] = output[i]
def radixSort(arr):
        max1 = max(arr)
        exp = 1
        while max1 / exp >= 1:
                countingSort(arr, exp)
                exp *= 10
arr = [170, 45, 75, 90, 802, 24, 2, 66]
radixSort(arr)
for i in range(len(arr)):
        print(arr[i],end=" ")
```

15. How do you check if two rectangles overlap with each other?

Solution:

```
class Point:
        def __init__(self, x, y):
                self.x = x
                self.y = y
def do_overlap(l1, r1, l2, r2):
        if l1.x == r1.x or l1.y == r1.y or r2.x == l2.x or l2.y == r2.y:
                return False
        if l1.x > r2.x or l2.x > r1.x:
                return False
        if r1.y > l2.y or r2.y > l1.y:
                return False
```

```
        return True
if __name__ == "__main__":
        l1 = Point(0, 10)
        r1 = Point(10, 0)
        l2 = Point(5, 5)
        r2 = Point(15, 0)
        if(do_overlap(l1, r1, l2, r2)):
                print("Rectangles Overlap")
        else:
                print("Rectangles Don't Overlap")
```

16. How do you design a vending machine?
    Solution:

17. How can you find the first non-repeated character in a word?
    Solution:

```
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)
```

18. How can you remove duplicates from arrays?
    Solution:

```
def Remove(duplicate):
        final_list = []
        for num in duplicate:
```

```
            if num not in final_list:
                final_list.append(num)
    return final_list
duplicate = [2, 4, 10, 20, 5, 2, 20, 4]
print(Remove(duplicate))
```

19. How can we check if a number is a prime number?
    Solution:

```
from math import *
is_prime = [True for i in range(10**6 + 1)]
primes =[]
def SieveOfEratosthenes(n):
    p = 2
    while (p * p <= n):
        if (is_prime[p] == True):
            for i in range(p * p, n + 1, p):
                is_prime[i] = False
        p += 1
    for i in range(2, n + 1):
        if is_prime[i]:
            primes.append(i)
def power_of_prime(n):
    for i in primes:
        if n % i == 0:
            c = 0
            while n % i == 0:
                n//= i
                c += 1
            if n == 1:
                return (i, c)
            else:
                return (-1, 1)

if __name__ == "__main__":
    n = 49
    SieveOfEratosthenes(int(sqrt(n))+1)
    num, power = power_of_prime(n)
    if num > 1:
        print(num, "^", power)
```

```
        else:
            print(-1)
```

20. How can you check if strings contain only digits?

    Solution:

```
a = "\u0030"
b = "\u00B2"
c = "10km2"
print(a.isnumeric())
print(b.isnumeric())
```