**Question 1.**

import random

def randomNum(x):

listNum = list(str(x))

d = len(str(x))

for i in range(0,d):

newNum = random.choice(listNum)

newNum2 = random.choice(listNum)

a, b = listNum.index(newNum), listNum.index(newNum2)

listNum[b], listNum[a] = listNum[a], listNum[b]

return listNum

randomNum(123456789)

**Question 2.**

def factorial(x):

factorial = 1

trail = 0

for i in range(2,x + 1):

factorial = factorial\*i

while i > 0: #this loop will check if the remainder = 0

if i % 5 ==0:

trail += 1

i = i/5

else:

break

return(print(" the trailing 0s for: ",factorial, " is " ,trail))factorial(250)

**Question 3.**

**Question 4.**

Question 1 = O(n)

Question 2 = O(N^2)

**Question 5.**

**Pseudocode:**

Function SQUARE(X)

sqr <- x

while sqr\*sqr > x

sqr <- sqr - 1

if sqr\*sqr is not = x then

return(check(x,sqr))

else:

if sqr <= 2:

return(4)

function CHECK(X,SQR)

if sqr\*sqr is not = x then

return(check(x-1,sqr))

else

if sqr <= 2 then

return(4)

else

return(sqr\*sqr)

**Question 5.**

**Code:**

def check(x,sqr):

if sqr\*sqr != x:

return(check(x-1,sqr))

else:

if sqr <= 2:

return(4)

else:

return(sqr\*sqr)

def square(x):

sqr = x

while sqr\*sqr > x:

sqr = sqr - 1

if sqr\*sqr != x: # this if statement was nested inside the while

return(check(x,sqr))

else:

if sqr <= 2:

return(4)

else:

return((sqr-1)\*(sqr-1))

**Question 6.**

**Pseudocode:**

function REVERSER(DATA,I)

x <- place holder

if length of data > i+1 then

x <- reverser(data,i+1)

return x+" "+data[i]

function SENTENCE(STRING)

word <- string.split(' ')

return(reverser(word,0))

**Code:**

def reverser(data,i):

x = ""

if len(data)>i+1:

x = reverser(data,i+1)

return x+" "+data[i]

def sentence(string):

word = string.split(' ')

return(reverser(word,0))

**Question 7.**

**Pseudocode:**

Function PRIME (X, I <- 2):

if x <= 1 then

return("prime numbers start at 2, this is not a prime number")

while x > I

if (x % i) == 0 then

return(x," is not a prime number")

break

else

return (prime(x, i+1)

else

return(x," is a prime number")

**Code:**

def prime (x, i = 2):

if x <= 1:

return("prime numbers start at 2, this is not a prime number")

while x > i: # this will only run only if x is greater than i

if (x % i) == 0: # if x divided by i has no remainder its not prime

return(x," is not a prime number")

break

else:

return (prime(x, i+1)) # if it does have a remainder it will loop back and divide it by i +1 which is every number less than x

else:

return(x," is a prime number")

**Question 8.**

**Pseudocode:**

FUNCTION REMOVEVOWEL(S,DONE <- empty string)

vowels <- ['a', 'e', 'i', 'o', 'u']

if length of s is > 0 then

if s[0] is in vowels then

return removeVowel(s[1:],done)

else

return removeVowel(s[1:],done+s[0])

else

return done

**Code:**

def removeVowel(s,done=""):

vowels = ['a', 'e', 'i', 'o', 'u']

if len(s)> 0:

if s[0] in vowels:

return removeVowel(s[1:],done)

else:

return removeVowel(s[1:],done+s[0])

else:

return done

**Question 9.**

**Pseudocode:**

Function BSEARCH (SLIST, LOW, HIGH):

FIRST <- 0

last <- length slist - 1

while first <= last

mid <- (first + last) DIV 2

if slist[mid] in range to low and high then

return True

else

if slist[mid] not in range to low and high then:

last <- mid -1

return False

**Code:**

def Bsearch (slist, low, high):

first = 0

last = len(slist)-1

while first <= last:

mid = (first + last)//2 )

if slist[mid] in range (low, high):

return True

else:

if slist[mid] not in range (low, high):

last = mid -1

return False

Bsearch([5,6,7,8,11,15,20,22,23,24,25,27,28,29],8,17)

**Time complexity**

O(log n)

**Question 10.**

**Code:**

def sequence(n):

tempList = []

subSq = []

for i in range(len(n)-1):

if n[i] < n[i + 1]:

tempList.append(n[i])

else:

tempList.append(n[i])

subSq +=[tempList]

tempList = []

tempList.append(n[i+1])

subSq +=[tempList]

return(max(subSq, key=len)) #returns the largest sub sequence within the sublist

sequence([1,5,1,6,7,1,2,3,4,5,6,7,8])

**Question 11.**

**Code:**

**Question 12.**

**Code:**

class BinTreeNode(object):

def \_\_init\_\_(self, value):

self.value=value

self.left=None

self.right=None

def tree\_insert( tree, item):

if tree==None:

tree=BinTreeNode(item)

else:

if(item < tree.value):

if(tree.left==None):

tree.left=BinTreeNode(item)

else:

tree\_insert(tree.left,item)

else:

if(tree.right==None):

tree.right=BinTreeNode(item)

else:

tree\_insert(tree.right,item)

return(tree)

def postorder(tree):

if(tree.left!=None):

postorder(tree.left)

if(tree.right!=None):

postorder(tree.right)

print(tree.value)

def in\_order(tree): # in order none recursive

node = tree

stack = []

treeNode = []

check = False

while check == False:

length = len(stack)

if node != None:

stack.append(node)

node = node.left

else:

if length >0:

node = stack.pop()

treeNode.append(node.value)

node = node.right

else:

check = True

print(treeNode)

if \_\_name\_\_ == '\_\_main\_\_'

t=tree\_insert(None,6);

tree\_insert(t,10)

tree\_insert(t,5)

tree\_insert(t,2)

tree\_insert(t,3)

tree\_insert(t,4)

tree\_insert(t,11)

in\_order(t)