**Question 1.**

import random

def shuffle(x):

listNum = list(str(x))# covenerts the input into a list

length = len(str(x))# stores the length of the list

for i in range(0,length):

newNum = random.choice(listNum) # chooses a random number from the list and stores it

newNum2 = random.choice(listNum)

a, b = listNum.index(newNum), listNum.index(newNum2) #selects the random numbers in the list and represnts them as 'a' and 'b'

listNum[b], listNum[a] = listNum[a], listNum[b] # will swap position 'a' with position 'b'

return listNum

print(shuffle(123456789))

#Runtime O(n)

'''

User inputs a list of number

for loop, will run the length amount of 'x'

first pick 1 random number from the list then pick second random number from list

asigns first number and second number postion to a variable, a and b

swaps postion a with postion b

this will loop as long as the length of 'x'

'''

**Question 2.**

def factorial(x):

factorial = 1

trail = 0

for i in range(2,x + 1): #Will calculate the factorial by looping x number of time, x is the user input

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

print(factorial(60))

#runtime O(n^2)

'''

factorial is equal to 1

the loop will start at 2 and run what ever x is +1

for every time it loops factorial will be 'factorial' times 'i'

while loop will check if 'i' is greater than 0

if it is it will check if i divided by 5 has a remainder

if it has no remainder trail will increase by 1

then i gets divided by 5 and loops again until 'i' is no longer greater than 0

returns the trailing value

'''

**Question 3.**

FUNCTION ADDITION (B,C)

for i in range of length B

for n in range of length B[0]

Answer1[i][n] <- b[i][n] + C[i][n]

return to function Multiplication(B,C,Answer1)

FUNCTION MULTIPLICATION(B,C, ANSWER1)

for i in range of length B

for n in range of length C[0]

for x in range of length C

Answer2[i][n] <- Answer2[i][n] + B[i][x] \* B[x][n]

for i in range of length answer1

for n in range of length Answer1[0]

Asnwer3[i][n] <- Answer[i][n] \* 2

return to function subtract(Answer2,Answer3)

FUNCTION SUBTRACT(ANSWER2,ANSWER3)

for i in range of length Answer2

for n in range of length Answer2[0]

Answer[i][n] <- Answer2[i][n] - Answer3[i][n]

return Answer

**Question 4.**

Question 1 = O(n)

Question 2 = O(N^2)

**Question 5.**

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))

print(square(998001))

'''

user inputs a number

the number get stored into a separate variable

while sqr \* sqr is greater than x then -1 from sqr

if sqr \* sqr is not equal to x then it’s not a perfect square and returns the value to the other function

this function will -1 until sqr is equal to x

if its equal to x then

check if it’s less than or equal to 2 if it is then its 4

else returns a perfect square less than its parameter.

'''

'''

FUNCTION CHECK(X,SQR)

if sqr\*sqr not = x then

return check(x-1,sqr)

else

if sqr <= 2 then

return(4)

else:

return(sqr\*sqr)

FUNCTION SQUARE(X)

sqr <- x

while sqr\*sqr > x

sqr <- sqr - 1

if sqr\*sqr is not = x then

return to functionn check(x,sqr)

else

if sqr <= 2 then

return(4)

'''

**Question 5.**

'''

FUNCTION CHECK(X,SQR)

if sqr\*sqr not = x then

return(check(x-1,sqr))

else

if sqr <= 2 then

return(4)

else:

return(sqr\*sqr)

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)

'''

**Question 6.**

def reverser(data,i):#reverse recursive function

x = ""

if len(data)>i+1:

x = reverser(data,i+1)

return x+" "+data[i]

def sentence(string):# sentence input and reverse output

word = string.split(' ')# breaks up the string

return(reverser(word,0)) #returns teh splited string and 0 to 'reverser'

print(sentence("my name is salah"))

print(sentence("how about this?"))

print(sentence("this should work!"))

'''

User input a sentence, any sentence

the sentence gets split into separate words into a list

function returns the list of strings

if the length of the sting is greater than i +1 then

'''

'''

Pseudocode:

FUNCTION REVERSER(DATA,I)

x <- empty string

if length of data > i+1 then

x <- reverser(data,i+1)

return x data[i]

function SENTENCE(STRING)

word <- split string

return to function reverser(word,0)

'''

**Question 7.**

# x = the testing value to see if its prime

# i = the number you going to divide x by to see if its prime

def prime (x, i = 2):

if x <= 1:

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

if 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")

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")

print(prime(67))

print(prime(9))

print(prime(13))

'''

user inputs a number to test and another number which is less than x but greater than 2 to divide by x to see if its prime

if x is less than 1 or equal to 1 then it’s not prime

this while loop will only run if x is greater than i

within the while loop the if statement will check if x divided by i has a remainder if it doesn’t then it’s not prime

else we call the function again and change the value of i =+1

this will keep looping until x is no longer greater than i

'''

**Question 7.**

'''

Pseudocode:

FUNCTION PRIME (X, I <- 2)

if x <= 1 then

return False

if x > I then

if x MOD i = 0 then

return False

else

return ro function prime(x, i+1)

else

return True

'''

**Question 8.**

def removeVowel(s,done=""):

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

if len(s)> 0:

if s[0] in vowels:

return removeVowel(s[1:],done) # skips the letter

else:

return removeVowel(s[1:],done+s[0]) # adds the letter to done

else:

return done

print(removeVowel("supercalifragilisticexpialidocious"))

print(removeVowel("coventry university"))

print(removeVowel("google"))

'''

input a string

if the length of string is greater than 0 then checkS

if the first index of the list is in the list vowels

if not skip the letter

if it is add the letter to done which is an empty string

now check the second letter until the length of s is no longer greater than 0

which then will return done which is the string without any vowels

'''

**Question 8.**

'''

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 to function removeVowel(s[1:],done)

else

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

else

return done

'''

**Question 9.**

def Bsearch (slist, low, high):

listFirst = 0 # first value

listLast = len(slist)-1 # -1 from the list length since its starts counting at 0

while listFirst < listLast:

mid = (listFirst + listLast)//2 #calculates middle value

if slist[mid] in range (low,high): # checks if the middle value is in range

return True

else:

if slist[mid] < low:

listFirst = mid +1

else:

listLast = mid -1

return False

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

print(Bsearch([5,6,7,8,11,15,20,22,23,24,25,27,28,29],30,40))

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

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

# Time complexity O(log n)

'''

user input a list of sorted numbers

first pointer will be at 0

last pointer will be at the length of the sting - 1 since it starts at 0

while 'first' is less than or equal to 'last' run the while loop

middle pointer is 'listFirst' + 'listLast' divided by 2

if the middle value is in range of the low and high value return true since theres a value

else if middle value is greater than the low value then 'listLast' equal middle value -1

else 'listfirst' equal middle value +1

'''

**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

first <- mid + 1

else

last <- mid - 1

return FALSE

'''

**Question 10.**

def sequence(n):

tempList = []

subSq = []

for i in range(len(n)-1): # 'i' represents the index of the list

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

tempList.append(n[i])

else:

tempList.append(n[i])

subSq +=[tempList] # appends the temp list to another list to create a sublist

tempList = []

tempList.append(n[i+1])

subSq +=[tempList]

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

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

'''

Input the 'n' sequence list

for loop run and look at the index instead of the number

if the first index is smaller than the next index

append that number to the the templist

carry on until the index is no longer less than the next index

append that list into the 'sub sequence' list to create a sub list

clear the the temp list and the loop will carry on

loop exits

add the last the value to the list

and append the last sequence to the sub list

now return the max sub sequence in the sublist

'''

**Question 11.**

**# got source code from moodle provided by lecturer**

class Node(object):

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

self.value=value

self.next=None

self.prev=None

class List(object):

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

self.head=None

self.tail=None

def insert(self,n,x):

#Not actually perfect: how do we prepend to an existing list?

if n!=None:

x.next=n.next

n.next=x

x.prev=n

if x.next!=None:

x.next.prev=x

if self.head==None:

self.head=self.tail=x

x.prev=x.next=None

elif self.tail==n:

self.tail=x

def remove(self,n): #n = node #removed function which was added by me

if n.prev != None:

n.prev.next = n.next # previouse node of n will skip n and go to the next node

else:

self.head = n.next # reached the head

if n.next != None:

n.next.prev = n.prev # next node of n will skip n and go to the previouse one

else:

self.tail = n.prev # read the tail which is the end

'''

the way you can remove and element from a linked list is by

changing the link from the previouse item to point to the next item,

so the one after that item.

if the the previouse node of the node you want to delete is not nothing

then

so the node you want to delete will be skipped and will be deleted by python using

automatic garbage collector

else

the head will be skipped

'''

def display(self):

values=[]

n=self.head

while n!=None:

values.append(str(n.value))

n=n.next

print ("List: ",",".join(values))

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

l=List()

A = Node(4)

B = Node(6)

C = Node(8)

l.insert(None,(A))

l.insert(l.head,(B))

l.insert(l.head,(C))

l.display()

l.remove(A)

l.display()

**Question 12.**

**# got source code from moodle provided by lecturer**

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 # traverses the node to the left subtree**

**else:**

**if length >0:**

**node = stack.pop()**

**treeNode.append(node.value)#adds the node to the list**

**node = node.right# traverses the node to the right subtree**

**else:**

**check = True**

**print(treeNode)**

'''

stored the 'tree' into a variable node

empty stack and empty list

check is set to false

while check is false carry on looping

length will be the legth of the stack

if node is not none then append node to the satck

traverse the node to the left substree

else

if the length is greater than 0

pop the stack and this will be the new node

adds the new node to the list

then traverses the node to the right subtree

else

check is true

print the list

'''

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)

from pprint import pprint

class Vertex:

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

self.name = n

self.neighbors = []

def addNeighbor(self, v):

if v not in self.neighbors:

self.neighbors.append(v)

self.neightbors = sorted(self.neighbors)

class Graph:

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

self.vertices = {}

self.graph = {}

def addVertex(self, vertex): #adds the vertex if the vertex is not there

if vertex.name not in self.vertices:

self.vertices[vertex.name] = vertex

return True

else:

return False

def addEdge(self, vertexFrom, vertexTo): # adds edges to the vertex

if vertexFrom in self.vertices:

for key, value in self.vertices.items():

if key == vertexFrom:

value.addNeighbor(vertexTo)

if vertexTo in self.vertices:

for key, value in self.vertices.items():

if key == vertexTo:

value.addNeighbor(vertexFrom)

return True

else:

return False

def adjacencyList(self):

for key in sorted(self.vertices.keys()):

self.graph[key] = set(self.vertices[key].neighbors)

def Print(self):

pprint(self.graph)