**Week 1 Task 1:**

import random

def numbershuf(list1):

for x in list1: #N

p1 = list1.index(x) #N

p2 = random.randint(0, len(list1)-1) #N

print("Old Position of Element = " + str(p1)) #N

print("New Poistion of Element = " + str(p2)) #N

list1[p1], list1[p2] = list1[p2], list1[p1] #N

print("Current state of the Element: ") #N

print(list1) #N

print("") #N

return(list1)

print (numbershuf([5,3,8,9,1,9,2,7]))

# Big O for this piece of code is O(N)

**Week 1 Task 2:**

import math

factlist = []

def trailnum():

number = int(input("What is the number: ")) #N

num = math.factorial(number) #1

numstr = str(num) #1

factlist = list(numstr) #1

count = 0 #1

for i in reversed(factlist): #N

if i == '0': #N

count += 1 #N

elif i != '0': #N

break #N

print (count)

trailnum()

**Week 2 Task 1:**

def PerfSquare(a):

if a >= 0:

b = a\*\*(1/2)

b = b – b%1

b = b(b\*\*2)

return (b)

**Pseudocode**

PerfSquare (a)

If A is less than or equal to 0

B <- A to the power of (1/2)

B <- B minius b mod 1

B <- B (b to the power of 2)

Return B

**Week 3 Task 1:**

Def reverseString(string):

Newstring = string.split()

Print(newstring)

Counter = len(newstring) – 1

While counter != -1:

Print(newstring[counter])

Counter -= 1

reverseString(‘my name is mir’)

**Pseudocode**

reverseString(string)

new string <- create a list of words in the string

print (newstring)

counter <- counter - 1

reverseString ( ‘my name is mir’)

**Week 3 Task 2:**

def isPrime(number):

x = True

for i in range(2,number):

if number % i == 0:

x = False

print(x)

isPrime(135)

**Pseudocode**

isPrime(number)

x <- True

for i in range between 2 and number

if number mod i = 0

x <- False

print ( x )

isPrime ( 135)

**Week 3 Task 3:**

def removeVowels(word):

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

for i in word:

if i in vowels:

word = word.replace(i, "")

print('word: ',word)

removeVowels('beautiful')

**Pseudocode**

removeVowels(word)

vowels <- [‘a’,’e’,’i’,’o’,’u’]

for i in word

if i in vowels

word <- word replace I

print (‘word’:, word)

removeVowels(‘beautiful’)

**Week 4:**

def binarySearch(alist, Min, Max):

first = 0

last = len(alist)-1

found = False

while first<=last and not found:

midpoint = (first + last)//2

if (alist[midpoint] >= Min) and (alist[midpoint] <= Max):

found = True

else:

if Min < alist[midpoint]:

last = midpoint-1

else:

first = midpoint+1

return found

Numberlist = [2,3,5,7,9,13]

print(binarySearch(Numberlist, 3,5))

print(binarySearch(Numberlist, 14,20))

**Pseudocode**

binarySearch with parameters alist, min, max

first <- 0

last <- length (of alist) -1

found <- False

while loop first <= last and not found

midpoint <- (first + last)//2 do

if (alist[midpoint]>=min) and (alist[midpoint]<= max) do

found <- true

else do

if min < alist[midpoint] do

last <- midpoint – 1

else do

first <- midpoint + 1

return found

**Week 5 Task 2:**

def remove(self, n):

if n.prev != None:

n.prev.next = n.next

else:

self.head = n.next

if n.next != None:

n.next.prev = n.prev

else:

self.tail = n.prev

**Week 6:**

def in\_order(tree):

stack = []

empty = 0

while empty == 0:

if tree != none:

stack.append(tree)

tree = tree.left

else :

if (len(stack)>0):

tree = stack.pop()

print(tree.value)

tree = tree.right

else :

empty = 1

**Week 7 :**

class graph:

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

self.adjacency = {} #

def insertvertex (self, v):

self.adjacency[v] = []

def insertedge(self, v, e):

if v in self.adjacency:

self.adjacency[v].append(e)

self.adjacency[e].append(v)

def printadjacency (self):

print (self.adjacency)

def dfs (self, v):

self.stack = []

self.visited = []

self.stack.append (v)

while self.stack != []:

value = self.stack.pop()

if value not in self.visited:

self.visited.append(value)

for e in self.adjacency[value]:

self.stack.append(e)

f = open("dfs.txt", "w")

f.write("dfs traversal: %s " % self.visited )

f.close()

return self.visited

def bfs (self,v):

self.queue = []

self.visited = []

self.queue.insert(0,v)

while self.queue != []:

value = self.queue.pop ()

if value not in self.visited:

self.visited.append(value)

for e in self.adjacency(value):

self.queue.insert(0,e)

f = open("bfs.txt", "w")

f.write("bfs traversal: %s " % self.visited )

f.close()

return self.visited

g = graph()

g.insertvertex(1)

g.insertvertex(2)

g.insertvertex(3)

g.insertvertex(4)

g.insertedge(1,2)

g.insertedge(2,3)

g.insertedge(3,4)

**Pseudocode**

class graph

init function

adjacency <- EMPTY DICTIONARY

insertvertex function with v as a parameter

adjacency with v in list <- empty list

insert edge function with e, v and self as parameters

if v is in adjacency do

add to adjacency vertex <- vertex

add to adjacency edge <- edge

printadjacency function with self as a parameter

print (adjacency)