**Name: Mohammad Zain Abbas**

**Reg #: 6865**

**DE-36 (CE), Syndicate: A**

**LAB 2 JOURNEL**

**Equipment Used:** Notebook Computer, Python IDLE 3.6

**Lab Tasks:**

**Q1: Write a simple calculator program. Follow the steps below:**

1. Create a class named ‘Complex’ that must have the following attributes:

* Variables named ‘Real’ and ‘Imaginary’.
* Methods named Magnitude() and Orientation().

Take a complex number from user in main and print it’s magnitude and orientation.

**SOLUTION CODE:**

import math

class Complex\_Numbers:

def \_\_init\_\_(self,x,y):

self.Real=x

self.Imaginary=y

def Magnitude(self):

Complex\_Number=(self.Real\*\*2) + (self.Imaginary\*\*2)

return math.sqrt(Complex\_Number)

def Orientation(self):

angle=self.Imaginary/self.Real

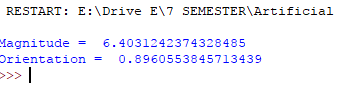
return math.atan(angle)

umer\_object=Complex\_Numbers(4,5)

print(umer\_object.Magnitude())

print(umer\_object.Orientation())

**OUTPUT:**



Q2: Use the previously created ‘Complex’ class and compute an 8-point Discrete Fourier Transform(DFT) of the following real-valued discrete sequence:

x = [2,2,2,2,2,2,2,2,2]

After computing DFT, Print the magnitude and phase. DFT can be computed as:

Where M is the length of input sequence, k ranges from 0 to N-1 and N = 8 here

**SOLUTION CODE:**

import math

class Complex\_Numbers:

def \_\_init\_\_(self,x,y):

self.Real=x

self.Imaginary=y

def Magnitude(self):

Complex\_Number=(self.Real\*\*2) + (self.Imaginary\*\*2)

return math.sqrt(Complex\_Number)

def Orientation(self):

angle=self.Imaginary/self.Real

return math.atan(angle)

##umer\_object=Complex\_Numbers(4,5)

##print(umer\_object.Magnitude())

##print(umer\_object.Orientation())

class DFT(Complex\_Numbers):

def \_\_init\_\_(self,x,y):

self.Real=x;

self.Imaginary=y;

umer\_object=DFT(0,0)

N=8

x=[2,2,2,2,2,2,2,2,2];

X=[];

for k in range(N-1):

umer\_object.Real=0;

umer\_object.Imaginay=0;

for j in range(len(x)):

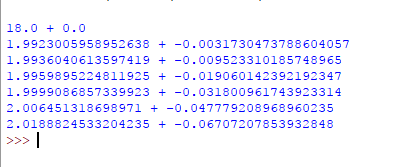
umer\_object.Real=umer\_object.Real+x[j]\*math.cos(2\*3.14\*k\*j/N)

umer\_object.Imaginary=umer\_object.Imaginary+x[j]\*math.sin(2\*3.14\*k\*j/N)

X.append(umer\_object)

print(X[k].Real,'+',X[k].Imaginary)

**OUTPUT:**



**Q3:** Create the binary search tree and search for the node ‘30’.

**SOLUTION CODE:**

class Node:

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

self.left = None;

self.right = None;

self.data = value;

class Tree:

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

self.root = None;

def InsertNode(self, node, value):

if(node==None):

self.root = Node(value);

else:

if(value<node.data):

if(node.left==None):

node.left = Node(value)

else:

self.InsertNode(node.left, value);

else:

if(node.right==None):

node.right = Node(value)

else:

self.InsertNode(node.right, value);

def Inorder(self, node):

if(node!=None):

self.Inorder(node.left)

print(node.data)

self.Inorder(node.right)

def search(self, node, data):

if node is None or node.data == data:

return node.data

if node.data < data:

print(node.data)

return self.search(node.right, data)

else:

print(node.data)

return self.search(node.left, data)

testTree = Tree()

testTree.InsertNode(testTree.root, 200)

testTree.InsertNode(testTree.root, 300)

testTree.InsertNode(testTree.root, 100)

testTree.InsertNode(testTree.root, 30)

print(testTree.search(testTree.root,30))

testTree.Inorder(testTree.root)

**OUTPUT:**

