Source Code

1.

```
import random
#Initialise two empty lists.
list1=[]
list2=[]

#Input random numbers using randint into the lists.
for x in range(0,150):
    y=random.randint(0,1024)
    list1.append(y)

for x in range(0,140):
    y=random.randint(0,1024)
    list2.append(y)

#Creation of Shell Short algorithm.
def shellSort(array, n):

    interval = n//2
    while interval > 0:
        for i in range(interval, n):
            temp = array[i]
            j = i
            while j >= interval and array[j - interval] > temp:
                array[j] = array[j - interval]
                j -= interval

            array[j] = temp
        interval //= 2
```

```
#Creation of Quick Sort algorithm.
def partition(arr, low, high):
    i = (low-1)
    pivot = arr[high]

    for j in range(low, high):

        if arr[j] <= pivot:

            i = i+1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)

def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:

        pi = partition(arr, low, high)
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

#Inputting the random generated numbers list to the Shell Sort and Quick Sort algorithm.
shellSort(list1,len(list1))
quickSort(list2,0,len(list2)-1)

#Printing the sorted lists.
print("Shell sort: ", list1)
print("Quick sort: ", list2)
```

2.

```
#Length of lists
len1=len(list1)
len2=len(list2)

final=[]
a=0
b=0

#Checks that the current iteration is not larger than the length of list1 and list2.
while (a < len1) and (b <len2):
   #Checks which element is smaller and appends it to the a new list, the smaller elment is incremented.
   if(list1[a]<list2[b]):
      final.append(list1[a])
      a=a+1
   else:
      final.append(list2[b])
      b=b+1

#Prints merged lists.
print("Merged lists: ",final)
```

3.

```
#List of numbers.
list1=[0,5,3,6,8,7,15,9]
#Empty list to put extreme points.
list2=[]

size=len(list1)
```

```
#Loops in list of numbers
for x, item in enumerate(list1):
    #Checks if x is in the last position of the list, if yes it breaks the loop.
    if(x==size-1):
        break
    #Used to skip the first elment of the list.
    elif(x>0):
        #Checks if current item is an extreme point.
        if(list1[x-1] > item < list1[x+1]):
            #Extreme point appended.
            list2.append(item)
        elif(list1[x-1] < item > list1[x+1]):
            #Extreme point appended.
            list2.append(item)
        else:
            continue
    else:
        continue
#Prints extreme point.
print("Extreme points:" ,list2)
```

4.

```
import random
def distinct(list):
    #Loops four times to compare each possibility
    for i in range(len(list)):
        for j in range(len(list)):
            for k in range(len(list)):
                for l in range(len(list)):
                    #Checks  a * b = c * d and a ≠ b ≠ c ≠ d
                    if (list[i] != list[j] and list[i] != list[k] and
```

```
                    list[i] != list[l] and list[j] != list[k] and
                    list[j] != list[l] and list[k] != list[l] and
                    list[i] * list[j] == list[k]  * list[l]):

                print(list[i], " x ", list[j], " =", list[k], " x ", list[l])

A=[]
#Inputting random numbers in a list.
for x in range(50):
   A.append(random.randint(0, 1025))
#Prints the list.
print(A)
#Prints all possibilties.
distinct(A)

5.
#List in RPN
list=[5,4,"-",9,"*"]
#Empty stack
stack=[]

#Temp variables
A=0
B=0
C=0

#For loop to loop through  RPN list
for x in list:
   #Checks if element is *, if it is, it pops the last two elements and the result is appended to the new stack.
   if(x=='*'):
      A=stack.pop()
```

```
        B=stack.pop()
        C=B*A
        stack.append(C)
        continue
    #Checks if element is /, if it is, it pops the last two elements and the result is appended to the new stack.
    elif(x=='/'):
        A=stack.pop()
        B=stack.pop()
        C=B/A
        stack.append(C)
        continue
    #Checks if element is +, if it is, it pops the last two elements and the result is appended to the new stack.
    elif(x=='+'):
        A=stack.pop()
        B=stack.pop()
        C=B+A
        stack.append(C)
        continue
    #Checks if element is -, if it is, it pops the last two elements and the result is appended to the new stack.
    elif(x=='-'):
        A=stack.pop()
        B=stack.pop()
        C=B-A
        stack.append(C)
        continue
    #If the element is not any of the above, it appends to the new stack with out any calculations.
    else:
        stack.append(x)
        continue
```

```
#Prints result
print(“Result: ”, stack)
```

6.

```
num=15

halfNum=num/2

#Function checks if number is prime
if num>1:
    #Loops through half of the number, example if 50 is given it loops through 1-25
    for i in range(2,int(halfNum+1)):
        #If it finds a number that when divided by the input gives a modulus=0 it means the number is not prime.
        if(num%i)==0:
            print("Number given is not prime")
            break
    else:
        #If it loops through and does not find any modulus=0 it means the number is prime
        print("Number is prime")

#Sieve of Eratosthenes.
prime = [True for i in range(num+1)]
p = 2
print(“All prime numbers under the number given: ”)
while (p * p <= num):
    if (prime[p] == True):
        for i in range(p * p, num+1, p):
            prime[i] = False
    p += 1
```

```
for p in range(2, num+1):
    if prime[p]:
        print(p)
```

7.

```
#Class which defines one node in the binary seacrh tree.
class Node:
    def __init__(self,d):
        self.value = d
        self.left = None
        self.right = None

#Function to insert a node, it compares the value of the node inputted to the tree to give it the right position.
    def insert(self, d):
        if self.value:
            if d < self.value:
                if self.left is None:
                    self.left = Node(d)
                else:
                    self.left.insert(d)
            elif d > self.value:
                if self.right is None:
                    self.right = Node(d)
                else:
                    self.right.insert(d)
        else:
            self.value = d

#Function which prints the Inorder traversal.
    def printInorder(self):
        if self.left:
```

```
            self.left.printInorder()
        print(self.value)
        if self.right:
            self.right.printInorder()

#Function which prints the Preorder traversal.
    def printPreorder(self):
        print(self.value)
        if self.left:
            self.left.printPreorder()
        if self.right:
            self.right.printPreorder()

#Function which prints the Postorder traversal.
    def printPostorder(self):
        if self.left:
            self.left.printPostorder()
        if self.right:
            self.right.printPostorder()
        print(self.value)

#Initialises the root node
root = Node(27)
#Initialises other nodes
root.insert(14)
root.insert(35)
root.insert(31)
root.insert(10)
root.insert(19)

#Prints the different traversal.
```

```
print("Inorder: ")
root.printInorder()
print("Preorder: ")
root.printPreorder()
print("Postorder: ")
root.printPostorder()
```

8.

```
#Function which returns an approximation of the sqaure root of a number using the Newton-Raphson Method.
#In the function it loops for 10 times.
def newtonsMethod(num, reps=10):
    cons= float(num)
    for i in range(reps):
        num = (num+(cons/num))/2
    return num


print(newtonsMethod(2))
```

9.

```
#List of numbers.
nums=[1,2,3,4,8,3,1,5,9]
#Temp List.
tempList=[]
#Flag.
bool=False


#Loops in list of numbers.
for i in range(0,len(nums)):
    #Loops in .
    for j in range(0,len(tempList)):
        #Checks if the current value in list of numbers if found in temp list, if found that means its repeated, so it prints the value.
```

```
        if(nums[i]==tempList[j]):
            print(nums[i])
            bool=True
            break
    #If value is not found to be repeated it is appended to temp list to be used in more itterations.
    if(bool==False):
        tempList.append(nums[i])
```

10.

```
#List of numbers.
nums=[1,2,3,4,8,3,1,5,9]
#Temp List.
tempList=[]
#Flag.
bool=False

#Loops in list of numbers.
for i in range(0,len(nums)):
    #Loops in .
    for j in range(0,len(tempList)):
        #Checks if the current value in list of numbers if found in temp list, if found that means its repeated, so it prints the value.
        if(nums[i]==tempList[j]):
            print(nums[i])
            bool=True
            break
    #If value is not found to be repeated it is appended to temp list to be used in more itterations.
    if(bool==False):
        tempList.append(nums[i])
```

11.

```
import math
```

```
#Function which finds the cosin value of a taylor series.
def cos(num, terms):
    cosApprox=0
    #Function loops to have a more accurate approximation.
    for x in range(terms):
        #Cosin function of the taylor series.
        cosApprox += ((-1)**x)*((num**(2*x))/(math.factorial(2*x)))
    return cosApprox

#Function which finds the sin value of a taylor series.
def sin(num, terms):
    sinApprox=0
    #Function loops to have a more accurate approximation.
    for x in range(terms):
        #Sin function of the taylor series.
        sinApprox += ((-1)**x)*((num**((2*x)+1)))/(math.factorial((2*x)+1))

#Value given in radians.
radians1 = math.radians(45)
radians2 = math.radians(20)

#Prints values
print(cos(radians1,5))
print(sin(radians2,5))
```

12.

```
#Function which gives the fibonacci sequence.
def fibonacciSeq(num):
    if num<0:
        print("Input too small")
```

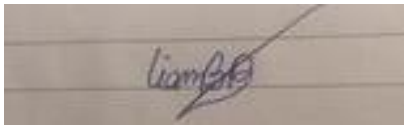
```
    elif num==0:
        return 0
    elif num<2:
        return 1
    else:
        return fibonacciSeq(num-1) + fibonacciSeq(num-2)

    total=0
    for i in range(0,num):
        total=total+fibonacciSeq(i)
    print(total)

print(fibonacciSeq(13))
```

State of Completion

- Question 1: Attempted and works well.
- Question 2: Attempted and works well.
- Question 3: Attempted and works well.
- Question 4: Attempted and works however it prints repeated pairs with different positions
- Question 5: Attempted and works well.
- Question 6: Attempted and works well.
- Question 7: Attempted and works well.
- Question 8: Attempted and works well.
- Question 9: Attempted and works well.
- Question 10: Attempted and works well.
- Question 11: Attempted and works well.
- Question 12: Attempted and works well.



Explanation

- Question 1: The program was tested by inputting list with random numbers into the sorting algorithm, and tested multiple times to check for any errors.
- Question 2: The program was tested by inputting lists with different sizes to check if the function works.
- Question 3: The program was tested by inputting lists with different sizes and also adding negative numbers.
- Question 4: The program was tested by inputting lists with random numbers between 0 and 1025.
- Question 5: The program was tested by using various RPN expressions and it was assumed that each expression given was correct.
- Question 6: The program was tested by inputting prime and non-prime numbers in the function and checked if the output was correct.
- Question 7: The program was tested by inputting various numbers in the tree and checked if the output of the different traversals was correct.
- Question 8: The program was tested by using different numbers and checking with online sources if the output was correct.
- Question 9: The program was tested by giving the function different lists and checking if the output was correct based on the list given.
- Question 10: The program was tested by using different lists and also included negative numbers, the output than was checked with the list given to see if it was correct.
- Question 11: The program was tested by giving different values in radians and checked if the output is correct by calculating the answer.

- Question 12: The program was tested by using different numbers and checked that the output was correct when comparing the input.

Sample Screen Dumps

*Arrays sizes in some questions are reduced in the sample screen dump so the working does not take a lot of pages to show the inputs and outputs.

- Question 1: Input: list1=[375, 169, 1001, 3, 23, 465, 655, 498]

  lists2=[614, 526, 420, 921, 584]

  Output: Shell sort: [3, 23, 169, 375, 465, 498, 655, 1001]

  Quick sort: [420, 526, 584, 614, 921]

- Question 2: Input: list1 = [3, 23, 169, 375, 465, 498, 655, 1001]

  list2 = [420, 526, 584, 614, 921]

  Output: Merged lists: [3, 23, 169, 375, 420, 465, 498, 526, 584, 614, 655, 921]

- Question 3: Input: list1=[0,5,3,6,8,7,15,9]

  Output: Extreme points: [5, 3, 8, 7, 15]

- Question 4: Input: list= [7, 40, 39, 10, 12, 31, 28, 19, 2, 36]

  Output: 7 x 40 = 10 x 28
  7 x 40 = 28 x 10
  40 x 7 = 10 x 28
  40 x 7 = 28 x 10
  10 x 28 = 7 x 40
  10 x 28 = 40 x 7
  28 x 10 = 7 x 40
  28 x 10 = 40 x 7

  *As explained in the state of completion I did not manage to print the possibilities once instead four times with different positions.

- Question 5: Input: list=[5,4,”-”,9,”*”]

  Output: Result: [9]

- Question 6: Input: 17

  Output: Number is prime
  All prime numbers under the number given:
  2
  3
  5
  7
  11
  13
  17

- Question 7: Input: root = Node(27)
  root.insert(14)
  root.insert(35)
  root.insert(31)
  root.insert(10)
  root.insert(19)
  Output: Inorder:
  10
  14
  19
  27
  31
  35
  Preorder:
  27
  14
  10
  19
  35
  31
  Postorder:
  10
  19
  14
  31
  35
  27

- Question 8: Input: print(newtonsMethod(2))

  Output: 1.4142135622373095

- Question 9: Input: nums=[1,2,3,4,8,3,1,5,9]

  Output: 3
  1

- Question 10: Input: nums=[1,6,2,6,7,19,54,21,-12]

  Output: 54

- Question 11: Input: radians1 = math.radians(45)

  radians2 = math.radians(80)

  Output: 0.7071068056832942

  None

- Question 12: Input: print(fibonacciSeq(10))

  Output: 55

Sources

- Question 1: https://www.geeksforgeeks.org/python-program-for-quicksort/

  https://www.programiz.com/dsa/shell-sort

- Question 6: https://www.geeksforgeeks.org/sieve-of-eratosthenes/
- Question 7: https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/
- Question 8: https://pythonnumericalmethods.berkeley.edu/notebooks/chapter19.04-Newton-Raphson-Method.html
- Question 11: https://pythonforundergradengineers.com/creating-taylor-series-functions-with-python.html
- Question 12: https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

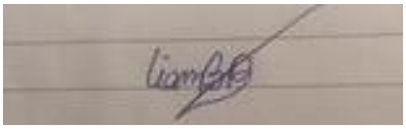
I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).



Liam Bugeja Douglas

Student Name

Signature

ICT1018

Course Code

Data Structures and Algorithms 1 CourseWork

Title of work submitted

29/05/2021

Date