**210CT- Programming, Algorithms and Data Structures**

1. **Random shuffle an array of integers**

**Introduction:**

For this task, I had to write a function which would show an array of integers being shuffled randomly using Python. To do this, I had to first create a suitable variable name because this would help me to understand what code I am functioning. I then needed to import a randomlist because this would help the array to shuffle the integers from the list. Once I had these two lines of code I was then able to figure out the last part of the code which was to make sure that the length in the list would always equal to 0 because this would make sure that

**Code:**

import random

def randomlist(List):

answer= []

while len(List) > 0:

num = random.randrange(len(List)) answer.append(List.pop(num))

return answer # returns a new list

**Explanation:**

This code is rationale because with the import random the function would be unsure of what to do because nothing has been defined for the function to start. Also by creating the random list it will be able to shuffle the array once the code is written for the list to be randomly shuffled.

1. **Number of trailing zeros**

**Introduction:**

For this task, I had to write code in python that would count the number of trailing zeros. To do this I first I had to create a suitable variable name because this would help me to understand what code I am writing and then after this I had to generate a list of numbers that I would use so that I could count the trailing zeros because this was what the task was. I then needed to add a while loop because I had to make sure that any of the numbers entered was greater than 0 or else the code would not work and once this was done I made the numbers divisible by 5 because I thought that it would be a good number to use when writing this code. Once this was all done I added a return function which would return all of the code above when I came to test the code once it was finished.

**Code:**

def Zeros(n):

fives = 0

for number in range(2, n+1):

while number > 0:

if number % 5 ==0:

fives += 1

number = number/5

else:

break

return fives # returns the function

1. **Highest perfect square**

**Introduction:**

For this task, I had to write pseudocode and code for a function that would return the highest perfect square. To do this I first had to give the code a variable name which would help with the rest of the code to understand what I am doing. I then needed to implement the import math function because this would allow the code to understand that it would have to do maths so that it can work out what the highest perfect square number would be for any integer that I enter once the code has finished.

**Code:**

import math

def Perfect\_Square(p):

interg=0

while interg\*interg < p:

interg = interg+1

if interg \* interg != p:

return "false"

else:

return p

**pseudocode:**

Perfect\_Square(p)

interg=0

WHILE interg\*interg < p:

interg = interg+1

IF (interg \* interg != p)

return string "false"

ELSE

return p

1. **Run time bonds using Big O nation**

Looking back from last week’s task, the run time bond using the big O nation is O(n) because for both the tasks there was only one function when n was called which was in the trailing zeros code and also n was only added to 1 meaning that it is only 0(n).

For the matrices question the big O nation is 0(n^3) because the function n is called multiple times during the pseudocode.

1. **Addition, subtraction and multiplication of two matrices**

**Introduction:**

For this task, I had to write pseudocode that would work to create two matrices which would add, subtract and multiple. To do this I first tried to work out the code so that it would be easier to translate into pseudocode.

**Pseudocode:**

1. **Reverse words in a sentence function**

**Introduction:**

For this task, I had to first create code that would function properly to make a set sentence be reversed once it was printed in python IDLE. To do this I first defined a variable name which would help execute the code and then after this I had then started to work on the code. I had to use my variable name that I defined in the () because this would be the base of the code that would run to make the sentence be reversed. I then needed to make sure that the phrase length of the sentence was equals to -1 because this would make sure that the code was reverse the sentence that was entered in the code. After the code was written I added a print statement because it was printing out a reverse sentence.

**Code:**

def sentence(phrase):

phraseLen = len(phrase)

if phraseLen == 1:

return phrase

return [phrase[-1]] + sentence(phrase[:-1])

phraseList = ["This","is","awesome"]

print(sentence(phraseList))

**Pseudocode:**

Sentence(Phrase)

PhraseLen <-- LENGTH OF Phrase

IF(Phrase)=1

return Phrase

return(Pharse[1]+ Sentence(Phrase[:-1])

1. **Recursive function to check if a number n is prime**

**Introduction:**

For this task, I had to write code and pseudocode for a function to check if a number n is prime or not. To do this I first created variables that would be used within the code to make sure that the code can function well. I then needed to write math functions in python to make sure that the number n would be checked to see if it was a prime number or not and to do this, I first needed to make sure that the first variable was equals to -1 and also greater than 2 because this would help the code to see whether it’s a prime number or not. I also decided to add print statements to show if the number was a prime number or not.

**Code:**

def primeNumber(n, val = None):

if val is None:

val = n-1

while val >=2:

if n % val == 0:

print("Not a prime number")

return False

else:

return primeNumber(n, val-1)

else:

print("This is a prime number")

return True

**pseudocode:**

1. **Recursive function that removes vowels**

**Introduction:**

In this task, I had to write both pseudocode and code for a function which would remove vowels from it. The sentence was given to me in the coursework brief and I needed to write the code that would make sure that all of the vowels was removed. I first started off by creating an appropriate variable name because this would be used in the code, so by naming it appropriately helped me to understand more of the code that I was writing. After this I needed to function to other variables where one would define what vowels were located in the sentence and the other variable would allow the sentence to be entered. I then wrote a short if statement which would help to remove the vowels in the sentence and also, I added a return statement because this would return the sentence with the vowels being removed.

**Code:**

def deleteVowels (word):

vowels = ["a","u", "e", "i"]

words = []

for i in word:

if i.lower() not in vowels:

words.append(i)

return "".join(words)

**Pseudocode:**

deleteVowels(word)

vowels<-- List()

NoVowels<-- List()

For I in Text

IF(I Not in Vowels)

NoVowels Append I

Return NoVowels

1. **Binary search algorithm**

**Code:**

def Binary\_Search(List, high, low):

firstLow = 0

lastHigh = len(List)-1

answer = False

while firstLow <= lastHigh and not answer:

middleValue = (firstLow + lastHigh) // 2

if List[middleValue] > high and List[middleValue] < low:

answer = True

else:

if high < List[middleValue]:

lastHigh = middleValue - 1

else:

firstLow = middleValue + 1

return answer

print(Binary\_Search([100,101,102,103,104,105,106,107,108,109,110,111],111,112))

**Pseudocode:**

Binary\_Search(LIST, HIGH, LOW)

FIRSTLOW = 0

LASTHIGH <--LENGTH(LIST)-1

ANSWER <-- FALSE

WHILE FIRSTLOW <= LASTHIGH AND NOT ANSWER

MIDDLEVALUE <-- FIRSTLOW + LASTHIGH DIV 2

IF(LIST(MIDDLEVALUE> HIGH AND LIST MIDDLEVALUE < LOW)

ANSWER <-- TRUE

ELSE:

IF(HIGH LIST(MIDDLEVALUE)

The big 0 nation for this is 0(n)

1. **Ascending order**

**Code:**

def LargestSequence(sequence):

finalList = []

tempList = []

num = 0

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

if i == 0:

tempList.append(sequence[i])

else:

if sequence[i] >= sequence[i-1]:

tempList.append(sequence[i])

if sequence[i] < sequence[i-1]:

finalList.append(tempList)

tempList = []

tempList.append(sequence[i])

finalList.append(tempList)

print(finalList)

for i in range(len(finalList)):

if i == 0:

num = len(finalList[i])

sequence = finalList[i]

else:

if len(finalList[i]) > num:

num = len(finalList[i])

sequence = finalList[i]

return("This is the largest sequence " + str(sequence) + str(num))

sequence = [1,2,3,4,1,5,1,6,7]

print(LargestSequence(sequence))

1. **Linked list delete node function**

**Code:**

def Delete(L, N):

if N.prev != None:

N.prev.next = N.next

else:

L.head = N.next

if N.next != None:

N.next.prev = N.prev

else:

L.tail = N.prev

The code above was implemented by me, I didn’t put the whole entire code because the question only asked for the delete function that I made.

1. **Inorder function for Tree\_Sort algorithm**

**Code:**

def in\_order(tree):

existingNode = tree

newList = []

mention = o

while not mention:

if(tree.left!=None):

in\_order(tree.left)

print (tree.value)

if(tree.right!=None):

in\_order(tree.right)

else mention = true

1. **Graph data structure**

**Code:**

class Node:

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

self.node = node

self.adjacent = {}

def add\_Adjacent(self,adjacentList):

self.adjacent[adjacentList]= adjacentList

class UnweightedGraph:

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

self.NewNodes = {}

def addNodes(self, total):

node = node(total)

self.nodes[total] = node

print("Node Added" + str(total))

def addEdges(self, total, adjacentL):

self.nodes[total].add\_Adjacent(self.nodes[adjacentL])

self.nodes[adjacentL].add\_Adjacent(self.nodes[total])

def PrintGraph(self):

List = []

for value in self.NewNodes:

List.append(value)

for value in List:

print(str(value))

graph = UnweightedGraph()

graph.addNodes(1)

graph.addNodes(2)

graph.addNodes(3)

graph.addNodes(4)

graph.addEdge(1, 2)

graph.addEdge(2, 3)

graph.addEdge(3, 4)

graph.PrintGraph()