210CT Coursework

Student Name: Selvachandran Thanushn

Student ID: 6785057

GitHub Repositories: https://github.com/thanushs/210CTcoursework/tree/thanushs-cw



PYTHON CODE

1. **import** random

|  |
| --- |
| (1) |
| (1) |
| (n) |
| (n) |
| (n) |
| (n) |
| (1) |
| (1) |
| (1) |
| (1) |
| (n) |
| (n) |
| (n\*n) |
| (n\*n) |
| (n) |
| (n) |
| (1) |

1. x = 'yes'
2. intList = []
3. **while** x == 'yes':                           #asks the user to input the numbers into the list one by one
4. y = int(input('Enter a number: '))
5. intList.append(y)
6. x = input("Do you want to continue? ")
7. **print**(intList)
8. count = 0
9. maxNumber = (len(intList)-1)
10. minNumber = 0
11. **for** i **in** intList:
12. randomNumber = random.randrange(minNumber, maxNumber)
13. **if** randomNumber == count:   #checks that the elements are not swithing with themselves
14. randomNumber = random.randrange(minNumber, maxNumber)
15. intList[count], intList[randomNumber] = intList[randomNumber], intList[count]
16. count = count + 1
17. **print**(intList)

1+1+n+n+n+n+1+1+1+1+n+n+n^2+n^2+n+n+1 = 2(n^2) + 9n + 7

O(n^2)

The rationale behind the shuffle is a loop that uses the random number function to give the index of two different elements in a list and swithes them. It does this for every number of elements in the list.



PYTHON CODE

|  |
| --- |
| (1) |
| (n) |
| (n) |
| (n) |
| (n) |
| (n) |
| (n\*n) |
| (n\*n\*n)) |
| (n\*n\*n)) |
| (n\*n\*n)) |
| (n\*n) |
| (n\*n) |
| (n\*n\*n) |
| (n\*n\*n\*n) |
| (n\*n\*n\*n) |
| (n) |
| (1) |

1. x = 0
2. **while** x == 0:
3. factorialInput = int(input("Enter a factorial value: "))
4. total = 1
5. noOfZeros = 0
6. count = factorialInput
7. **if** factorialInput > 0:
8. **while** count > 1:   #loop to work out factorial of input
9. total = total \* count \* (count - 1)
10. count = count - 2
11. **print** (total)
12. total = str(total)
13. **for** i **in** total:   #goes through every digit and checks if they are zero and how many there are
14. **if** i == "0":
15. noOfZeros = noOfZeros + 1
16. **print**("The number of zeros in " + str(factorialInput) + " factorial is " + str(noOfZeros))
17. **print**("")

1+n+n+n+n+n+n^2+n^3+n^3+n^3+n^2+n^2+n^3+n^4+n^4+n+1 = 2(n^4)+4(n^3)+3(n^2)+6n+2

O(n^4)

3.

PSEUDOCODE

value ←int

squareNumber ← 1

count ← 1

WHILE (squareNumber <= value) DO

squareNumber ← count ^ 2

count ← count + 1

END WHILE

IF (squareNumber > value) DO

squareNumber ← (count – 2) ^ 2

PRINT( squareNumber)

PYTHON CODE

1. value = int(input("Enter a value: "))
2. squareNumber = 1
3. count = 1
4. **while** squareNumber <= value: #loop to calulate closest square number
5. squareNumber = count \*\* 2
6. count = count + 1

9. **if** squareNumber > value:   #checks if the answer is bigger than out put so it goes bak one number
10. squareNumber = (count - 2) \*\* 2
11. **print**("The highest perfect square number is " + str(squareNumber))

4. For this question, I have written the run time bounds for last week’s task in question 1 and 2.

5.

PSEUDOCODE

matrixB ← list[](2,2) (1)

matrixC ← list[](2,2) (1)

FUNCTION matrixMultiplication (matrixB, matrixC) (1)

x ← LENGTH(row of (matrixB)) (1)

y ← LENGTH(column of (matrixC)) (1)

z ←list[] (x, y) (1)

FOR i IN x DO (n)

matrixA[i] ← SUM matrixB[i] \* matrixC[i] (n)

RETURN matrixA (n)

FUNCTION matrixAddition(matrixB, matrixC) (1)

IF (SIZE of matrixB = SIZE of matrixC) DO (n)

FOR i IN x DO (n\*n)

matrixA[i] ← matrixB[i] + matrixC[i] (n\*n)

ELSE DO (n)

PRINT (“Both matrices are not the same size”) (n)

RETURN matrixA (1)

FUNCTION matrixSubtraction(matrixB, matrixC) (1)

IF (SIZE of matrixB = SIZE of matrixC) DO (n)

FOR i IN x DO (n)

matrixA[i] ← matrixB[i] + matrixC[i] (n)

ELSE DO (n)

PRINT (“Both matrices are not the same size”) (n)

RETURN matrixA (1)

matrixB ← matrixMultiplication(matrixB, matrixC) (1)

matrixC ← (2 \* matrixAddition(matrixB, matrixC)) (1)

matrixFinal ← matrixSubtraction(matrixB, matrixC) (1)

1+1+1+1+1+1+n+n+n+1+n+n^2+n^2+n+n+1+n+n+n+n+n+n+1+1+1+1 = 2(n^2) + 12n + 12

O(n^2)

6.

PSEUDOCODE

intString ← string

finalStringList ← list

newString ← string

newStringList ← list

FOR char IN intString DO

IF char = “ ” DO

APPEND newString TO newStringList

newString ← “ ”

ELSE DO

NEXT char IN newString

IF newString DO

APPEND newString TO newStringList

count ← LENGTH(newStringList) – 1

WHILE count >= 0 DO  
 x ← newStringList[count]

APPPEND x TO finalStringList

count = count – 1

finalString ← JOIN finalStringList

PRINT(finalStringList)

PYTHON CODE

|  |
| --- |
| (1) |
| (1) |
| (1) |
| (1) |
| (n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n) |
| (n) |
| (1) |
| (n) |
| (n) |
| (n) |
| (n) |
| (1) |
| (1) |

1. intString = str(input("Enter the sentence: "))
2. finalStringList = []
3. newString = ""
4. newStringList = []
5. **for** c **in** intString:
6. **if** c == ' ':   #ignores any spaces in the string
7. newStringList.append(newString)
8. newString = ''
9. **else**:
10. newString += c
11. **if** newString:
12. newStringList.append(newString)
13. count = len(newStringList) - 1
14. **while** count >= 0:
15. x = newStringList[count]   #converts the appended string into a new list
16. finalStringList.append(x)
17. count = count - 1
18. finalString = " ".join(finalStringList)  #separates the words
19. **print**(finalString)

1+1+1+1+n+n^2+n^2+n^2+n^2+n^2+n+n+1+n+n+n+n+1+1 = 5(n^2) +7n + 7

O(n^2)

7.

PSEUDOCODE

n ← integer

count ← n – 1

answer ← “ ”

FUNCTION primeCheck (n, count, answer)

WHILE answer = “ ” AND count > 1 DO

IF (n MOD count = 0) DO

answer ← n + “is not a prime number”

RETURN answer

count ← count – 1

IF count = 1 DO

answer ← n + “is a prime number”

RETURN answer

output ←primeCheck(n, count, answer)

PRINT(output)

PYTHON CODE

1. n = int(input("Enter a number: "))
2. count = n - 1
3. answer = ''
4. **def** primeCheck (n, count, answer):
5. **while** answer == '' **and** count > 1:   #loops until answer is outputted or ount is equal to 1
6. **if** (n % count == 0):
7. answer = (str(n) + " is not a prime number")  #checks if n is divisible by any other numbers
8. **return** (answer)
9. count = count - 1
10. **if** count == 1:
11. answer  = (str(n) + " is a prime number")
12. **return**(answer)
14. output = primeCheck(n, count, answer)
15. **print**(output)

8.

PSEUDOCODE

intString ← string

FUNCTION vowels (intString)

IF LENGTH(intString) = 0 DO

RETURN intString

ELSEIF (intString[0] in “aeiouAEIOU”) DO

RETURN vowels(intString[1:])

RETURN (intString[0] + vowels(intString[1:]))

PRINT vowels(intString)

PYTHONCODE

1. intString = str(input("Enter a string: "))
2. **def** vowels(intString):
3. **if** len(intString) == 0:   #ends the function once the string is empty
4. **return** intString
5. **elif** (intString[0] **in** "aeiouAEIOU"):
6. **return** vowels(intString[1:])  #returns everything exept the first element
7. **return** (intString[0] + vowels(intString[1:]))  #brings it all together
8. **print**(vowels(intString))

9.

PSEUDOCODE

x ← “yes”

intList ← list[]

count ← 0

low ← 10

high ← 14

answer ← “False”

WHILE x = “yes” DO

a ← integer

APPEND a TO intList

x ← string

SORT intList IN ascending order

WHILE LENGTH(intList) > 0 AND answer = “False” DO

IF (LENGTH(intList) MOD 2) = 0 DO

count ← LENGTH(intList)

ELSE DO

count ← LENGTH(intList) + 1

count ← INT((count/2) – 1)

IF intList[count] >= low and intList[count] <= high DO

answer ← “True”

ELIF intList[count] < low DO

DELETE intList[:count]

ELSE DO

DELETE intList[count:]

PRINT answer

PYTHON CODE

|  |
| --- |
| (1) |
| (1) |
| (1) |
| (1) |
| (1) |
| (1) |
| (n) |
| (n) |
| (n) |
| (n) |
| (1) |
| (1) |
| (n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (n\*n) |
| (1) |

1. x = "yes"
2. intList = []
3. count = 0
4. low = 10
5. high = 14
6. answer = "False"
7. **while** x == "yes":
8. a = int(input("Enter a number: "))
9. intList.append(a)
10. x = input("Do you want to continue? ")
11. intList.sort(key=int)
12. **print**(intList)
13. **while** len(intList) > 0 **and** answer == "False":
14. **if** (len(intList) % 2) == 0:   #checks the midpoint value in the list
15. count = len(intList)
16. **else**:
17. count = len(intList) + 1
18. count = int((count / 2) - 1)
19. **if** intList[count] >= low **and** intList[count] <= high:  #checks where the midpoint lies
20. answer = "True"
21. **elif** intList[count] < low:
22. **del** intList[:count]
23. **else**:
24. **del** intList[count:]
25. **print**(answer)

1+1+1+1+1+1+n+n+n+n+1+1+n+n^2+n^2+n^2+n^2+n+n^2+n^2+n^2+n^2+n^2+n^2+1 = 10(n^2) + 6n + 9

O(n^2)

10.

PYTHON CODE

1. numberList = []
2. count = 0
3. finalNumberList = []
4. finalNumberList2 = []
5. x = 'yes'
6. **while** x == 'yes':
7. numberList.append(int(input("Enter an integer: ")))
8. x = str(input("Do you want to add another number? "))
9. **print**(numberList)
10. **for** i **in** range(len(numberList)):   #repeats for length of list
11. finalNumberList.append(numberList[i])
12. **if** i ==len(numberList)- 1 **or** numberList[i] > numberList[i+1] : #check for where the value of i is
13. **if** len(finalNumberList) > len(finalNumberList2):  #compares the two lists for their lengths
14. finalNumberList2 = finalNumberList
15. finalNumberList=[]
16. **else**:
17. finalNumberList2 = finalNumberList2
18. **print**(finalNumberList, finalNumberList2)
19. **print**(finalNumberList2)

11.

PYTHON CODE

1. **class** Node(object):
2. def \_\_init\_\_(self, value):
3. self.value=value
4. self.next=None
5. self.prev=None
7. **class** List(object):
8. def \_\_init\_\_(self):
9. self.head=None
10. self.tail=None
11. def insert(self,n,x):
12. **if** n!=None:
13. x.next=n.next
14. n.next=x
15. x.prev=n
16. **if** x.next!=None:
17. x.next.prev=x
18. **if** self.head==None:
19. self.head=self.tail=x
20. x.prev=x.next=None
21. elif self.tail==n:
22. self.tail=x
23. def display(self):
24. values=[]
25. n=self.head
26. **while** n!=None:
27. values.append(str(n.value))
28. n=n.next
29. print "List: ",",".join(values)
30. def **delete**(self, n):
31. **if** n.prev != 0:   #if the node is empty then it moves on to the next node
32. n.prev.next = n.next
33. **else**:
34. self.head = n.head
35. **if** n.next != 0:   #replaces the node with the next one if it is larger
36. n.next.prev = n.prev
37. **else**:
38. self.tail = n.prev
40. **if** \_\_name\_\_ == '\_\_main\_\_':
41. l=List()
42. l.insert(None, Node(4))
43. l.insert(l.head,Node(6))
44. l.insert(l.head,Node(8))
45. l.display()

12.

PYTHON CODE

1. **class** BinTreeNode(object):
3. **def** \_\_init\_\_(self, value):
4. self.value=value
5. self.left=None
6. self.right=None


10. **def** tree\_insert( tree, item):
11. **if** tree==None:
12. tree=BinTreeNode(item)
13. **else**:
14. **if**(item < tree.value):
15. **if**(tree.left==None):
16. tree.left=BinTreeNode(item)
17. **else**:
18. tree\_insert(tree.left,item)
19. **else**:
20. **if**(tree.right==None):
21. tree.right=BinTreeNode(item)
22. **else**:
23. tree\_insert(tree.right,item)
24. **return** tree
26. **def** postorder(tree):
27. **if**(tree.left!=None):
28. postorder(tree.left)
29. **if**(tree.right!=None):
30. postorder(tree.right)
31. **print** (tree).value

34. **def** in\_order(tree):
35. x = []
36. **while**(tree != null):   #if the tree node is empty then it moves on the left hand side
37. **if** (tree != null):
38. x.push(tree)
39. tree = tree.left
40. **else**:
41. tree = x.pop()   #otherwise it moves the right and deletes that value in the new list
42. visit(tree)
43. tree = tree.right
45. **if** \_\_name\_\_ == '\_\_main\_\_':
47. t=tree\_insert(None,6);
48. tree\_insert(t,10)
49. tree\_insert(t,5)
50. tree\_insert(t,2)
51. tree\_insert(t,3)
52. tree\_insert(t,4)
53. tree\_insert(t,11)
54. in\_order(t)

13.

PSEUDOCODE

originalVertex ← []

vertices ← []

vertex ← character

FUNCTION newVertex (vertex)

IF vertex != originalVertex DO

vertices ← vertex

RETURN True

ELSE DO

RETURN False

FUNCTION newEdge (edge, originalVertex)

IF edge NOT IN originalVertex DO

APPEND edge TO vertices.neighbour

ELSE DO

RETURN FALSE

FUNCTION printGraph (vertices, edge)

FOR i IN vertices DO

PRINT(vertices[i], edge)

CALL printGraph(vertices, edge)

For the python code, I had attempted to do the code but I kept on getting more and more errors and I couldn’t solve it in the time given. Here is what I could do and hope that it still counts for some marks. Thanks

PYTHON CODE

1. edgeList = ['AB', 'AE', 'BF', 'CG', 'DE', 'DH', 'EH', 'FG', 'FI', 'FJ']
2. **class** vertex:
3. **def** init(self, n):
4. self.name = n
5. self.neighbours = list()
6. **def** addNeighbour(self, v):
7. **if** v **not** **in** self.neighbours:
8. self.neighbours.append(v)
9. self.neighbours.sort()
11. **class** graph:
12. vertices = {}
14. **def** newVertex(self, vertex):
15. **if** isinstance(vertex, Vertex) **and** vertex.name **not** **in** self.vertices:
16. self.vertices[vertex.name] = vertex
17. **return** True
18. **else**:
19. **return** False
21. **def** newEdge(self, u, v):
22. **if** u **in** self.vertices **and** v **in** self.verties:
23. **for** key, value **in** self.vertices.items():
24. **if** key == u:
25. value.newNeighbour(v)
26. **if** key == v:
27. value.newNeighbour(u)
28. **return** True
29. **else**:
30. **return** False
32. **def** printGraph(self):
33. **for** key **in** sorted(list(self.vertices.keys())):
34. **print**(key + str(self.vertices[key].neighbours))

37. graph.printGraph()