

Programming, Algorithms and Data Structures (210CT)

COURSEWORK REPORT

Nathan Zengamambu

INTRODUCTION:

The report is a record of all programming code written in accordance to the tasks objectives, alongside the comments for further description and clarification of functionalities as well as side notes entailing encountered drawbacks and applicable (or theoretical) solutions to them.

All live code from each task can be viewed in the git repository (see link in the appendix 2).

CONTENTS:

Question	Page
1.....	2
2.....	3
3.....	4
4.....	5
5.....	6
6.....	12
7.....	13
8.....	14
9.....	15
10.....	19
12.....	21
Appendix.....	21

1. Defining the shuffle function:

```
01  from random import *
02  input1 = [5,3,8,6,1,9,2,7]
03  print(input1)
04
05  def shuff(L):
06      '''Randomises order of elements in the taken array'''
07      for i in range(len(L)):
08          iRandom = randint(0,len(L)-1) # random index in the array
09          temp = L[i] # swapping takes place
10          L[i] = L[iRandom]
11          L[iRandom] = temp
12      return L
13
14  print(shuff(input1))
```

2. Counting the trailing zeros in a factorial number:

```
01 num = int(input("num: "))
02
03 def trailingZero(t):
04     '''Returns number of trailing zeros of answer from a factorial int'''
05     def fact(n):
06         '''Returns factorial number of argument'''
07         if n == 1:
08             return n
09         else:
10             return n * fact(n-1)
11     count = 0
12     factAns = str(fact(t))[:-1] # factorial number, converted to string and sequentially reversed
13     for i in range(len(factAns)): # counts 0s: stops if no more at trailing end
14         if factAns[i] != "0":
15             break
16         else:
17             count+=1
18     return count
19
20 print(trailingZero(num))
```

3. Returning the highest square number less or equal to the function parameter:

```
01  from math import sqrt
02  num = int(input("number: "))
03
04  def highestSquareNumber(n):
05      ''' Returns closest perfect square number '''
06
07      psn = []                                ## generating perfect square numbers
08      for i in range(10000):
09          if sqrt(i) % 1 == 0:
10              psn.append(i)
11
12      for i in range(len(psn)):                ## returns closest perfect square number
13          if n <= psn[i]:
14              return psn[i]
15
16  print( highestSquareNumber(num) )
```

4. Pseudocode for questions 1 and 2:

Question 1:

Pseudocode	Time cost
from random import *	1
input1 = [5,3,8,6,1,9,2,7]	1
print(input1)	
def shuff(L):	
for i in range(len(L)):	n
iRandom =	n
randint(0,len(L)-1)	n
temp = L[i]	n
L[i] = L[iRandom]	n
L[iRandom] = temp	1
return L	
print(shuff(input1))	1
Runtime bound:	$f(n) = 1 + 1 + n + n + n + n + n + n + 1 + 1$ $= 5n + 4$
Complexity:	$O(n)$

Question 2:

Pseudocode	Time cost
num = int(input("num: "))	1
def trailingZero(t):	
def fact(n):	
if n == 1:	1
return n	1
else:	
return n * fact(n-1)	n
count = 0	1
factAns = str(fact(t))[:-1]	1
for i in range(len(factAns)):	n
if factAns[i] != "0":	n
break	1
else:	
count+=1	n
return count	1
print(trailingZero(num))	1
Runtime bound:	$f(n) = 2n + 2n? + 8$ $= 2n + 2mn + 8$ $= 2n + 2n + 8$ $= 4n + 8$
Complexity:	$O(n)$

5. Matrix calculations:

Addition function

Pseudocode

```

ADD_MATRICES(matrix1, matrix2):
    '''Returns a new matrix with each value equal to the sum of each index values of the given
    matrices'''

    newMatrix <- new empty integer array [] of size length of matrix1 and matrix2

    FOR i <- 0 TO matrix1.length:
        IF len(matrix1) != len(matrix2) or len(matrix1[i]) != len(matrix2[i]):
            RETURN NONE
        ELSE:
            FOR j <- 0 to matrix1[i].length:
                newMatrix[i][j] <- matrix1[i][j] + matrix2[i][j]

    RETURN newMatrix

```

Time cost

1

n

n

1

 n^2 n^2 Runtime bound: $2n^2 + 2n + 2$ Complexity: $O(n^2)$

[CONTINUED]

High-level language code (Python):

```
09 def addM(matrix1, matrix2):
10     '''Returns a new matrix with each value equal to the sum of each index values of the given matrices'''
11     newMatrix = [[0 for i in range(len(matrix1[x]))] for x in range(len(matrix1))]
12     # 0s added to each index of inner list; inner list added to each index of main list
13     # indicates empty spaces to new array, equal to size length of one of given matrices
14
15     for i in range(len(matrix1)):
16         if len(matrix1) != len(matrix2) or len(matrix1[i]) != len(matrix2[i]): # checks if given matrices
are of the same size length for addition to take place
17             return None
18         else:
19             for j in range(len(matrix1[i])):
20                 newMatrix[i][j] = matrix1[i][j] + matrix2[i][j] # gives sum of values of both given
matrices by index
21     return newMatrix
```

[CONTINUED]

Subtraction function:**Pseudocode****Time cost**

```

SUBTRACT_MATRICES(matrix1, matrix2):
    '''Returns a new matrix with each value equal to the subtraction of each index values of the
    first matrix by that of the second matrix'''

    newMatrix <- new empty integer array [] of size length of matrix1 and matrix2

    FOR i <- 0 TO matrix1.length:
        IF len(matrix1) != len(matrix2) or len(matrix1[i]) != len(matrix2[i]):
            RETURN NONE
        ELSE:
            FOR j <- 0 to matrix1[i].length:
                newMatrix[i][j] <- matrix1[i][j] - matrix2[i][j]
    RETURN newMatrix

```

1

n

n

1

 n^2 n^2

1

Runtime bound: $2n^2 + 2n + 2$ **Complexity:** $O(n^2)$ **[CONTINUED]**

High-level language code (Python):

```
09 def subM(matrix1, matrix2):
10     '''Returns a new matrix with each value equal to the subtraction of each index values of the first
    matrix by that of the second matrix'''
11     newMatrix = [[0 for i in range(len(matrix1[x]))] for x in range(len(matrix1))]
12
13     for i in range(len(matrix1)):
14         if len(matrix1) != len(matrix2) or len(matrix1[i]) != len(matrix2[i]):
15             return None
16         else:
17             for j in range(len(matrix1[i])):
18                 newMatrix[i][j] = matrix1[i][j] - matrix2[i][j] # subtracts each value in given matrices
19     return newMatrix
```

[CONTINUED]

Multiplication function:**Pseudocode**

```

MULTIPLY_MATRICES(matrix1, matrix2):
    '''Returns a new matrix with each value equal to the multiplication of given matrices'''
    newMatrix <- new empty integer array [] equal size length of matrix1

    IF matrix2 is a list type variable:
        FOR i <- 0 TO matrix1.length:
            FOR j <- 0 to matrix1[i].length:
                FOR k <- 0 to matrix2.length-1:
                    newMatrix[i][j] <- newMatrix[i][j] + matrix1[i][k] * matrix2[k][j]

    ESLE IF matrix2 is a integer type variable:
        number <- matrix2
        FOR i <- 0 TO matrix1.length:
            FOR j <- 0 to matrix1[i].length:
                newMatrix[i][j] <- matrix1[i][j] * number

    RETURN newMatrix

```

Time cost

1

1

n

 n^2 n^3 n^3

1

n

n

 n^2 n^2

1

Runtime bound: $2n^3 + 3n^2 + 3n + 4$ **Complexity:** $O(n^3)$ **[CONTINUED]**

High-level language code (Python):

```
09 def multiM(matrix1, matrix2):
10     '''Returns a new matrix with each value equal to the multiplication of given matrices'''
11
12     newMatrix = [[0 for i in range(len(matrix1[x]))] for x in range(len(matrix1))]
13     # determines that the dimension product of new matrix will be height (i) of matrix1 and width (j) of
    matrix2
14
15     if type(matrix2) == list: # multiplies first matrix by second matrix, if second argument is a matrix
16         for i in range(len(matrix1)):
17             for j in range(len(matrix1[0])):
18                 for k in range(len(matrix2)-1):
19                     newMatrix[i][j] += matrix1[i][k] * matrix2[k][j] # multiplies each index value in given
    matrix by that of second matrix
20
21     elif type(matrix2) == int: # multiplies first matrix by given number, if second argument is an integer
22         number = matrix2 # to identify second argument as number
23         for i in range(len(matrix1)):
24             for j in range(len(matrix1[i])):
25                 newMatrix[i][j] = matrix1[i][j] * number
```

6. Function reversing the words in a given sentence:

Pseudocode	Time cost
String word <- "This is awesome"	1
WORD_REVERSE(word):	
word <- String [] word	1
reversedWord <- New String Array []	1
For i <- a.length-1 Downto 0 {	n
Append a[i] to reversedWord	n
}	
reversedWord <- joined String	1
reversedWord	1
RETURN reversedWord	
	1
PRINT WORD_REVERSE(word)	
Runtime bound:	$f(n) = 2n + 6$
Complexity:	$O(n)$

High-level language code (Python):

```

01 word = "This is awesome"
02
03 def wordReverse(w):
04     '''Returns the string argument reversed by
    word'''
05     w = w.split(' ')
06     reversedWord = []
07     ## w = w[::-1]
08     for i in range(len(w)-1, -1, -1):
09         reversedWord.append(w[i])
10     reversedWord = ' '.join(reversedWord)
11     return reversedWord
12
13 print(wordReverse(word))

```

7. Checking if a given integer is a prime number:

Pseudocode	Time cost
number <- any given integer value	1
IS_PRIME(num, test <- number)	
If num ≤ 2 OR test = 1 {	1
Return TRUE	1
} Else If num mod test = 0 {	1
Return IS_PRIME(num, test-1)	n
}	
Return FALSE	1
}	
PRINT IS_PRIME(number)	1
Runtime bound:	f(n) = n + 6
Complexity:	O(n)

High-level language code (Python):

```

01 number = int(input("number: "))
02
03 def isPrime(num, test = number-1):
04     '''Determines whether an integer value is
    prime'''
05     try:
06         if num <= 2 or test == 1:
07             return True
08         elif num % test != 0:
09             return isPrime(num, test-1)
10         return False
11     except RecursionError:
12         return False
13
14 print( "%d is a prime number: %s" % (number,
    isPrime(number)) )

```

8. Removing vowels from a given string:

Pseudocode	Time cost
word <- "beautiful"	1
REMOVE_VOWEL(wrd)	
wrd <- String [] wrd	1
For i <- 0 to length(wrd):	n
If w[i] = "a" OR	
w[i] = "e" OR	
w[i] = "i" OR	
w[i] = "o" OR	
w[i] = "u":	n
Remove wrd[i]	n
wrd = String wrd	1
Return wrd	1
PRINT REMOVE_VOWEL(word)	1
Runtime bound:	$f(n) = 3n + 5$
Complexity:	$O(n)$

High-level language code (Python):

```

01 word = "beautiful"
02
03 def removeVowel(wrd):
04     wrd = list(wrd)
05     for i in range(len(wrd)):
06         if wrd[i] == "a" or wrd[i] == "e" or
wrd[i] == "i" or wrd[i] == "o" or wrd[i] == "u":
07             wrd[i] = ''
08     wrd = ''.join(wrd)
09     return wrd
10
11 print(removeVowel(word))

```

9. Adapting the binary search (searching by interval):

```

01 number1 = int(input("1st number: "))
02 number2 = int(input("2nd number: "))
03 List = [4, 19, 23, 36, 40, 43, 61, 64, 78, 95]
04
05 def binarySearch(num1, num2, array):
06     '''performs binary search to identify if there is
07         a number in the List within a given interval'''
08     mid = len(array)//2
09     try:
10         if num1 <= num2:
11             if array[mid-1] >= num1 and array[mid-1] <= num2:
12                 return True
13             elif array[mid-1] > num2:                ##if the value is larger than pivot (to
the right of the array)
14                 return binarySearch(num1, num2, array[:mid-1])    ##calls itself with the array halved to
the right side of the pivot
15             elif array[mid-1] < num1:                ##if the value is smaller than pivot
(to the left of the array)
16                 return binarySearch(num1, num2, array[mid:])    ##calls itself with the array halved to
the left side of the pivot
17             return False
18         else:
19             return "ERROR! Lower value > upper value"
20     except IndexError or RecursionError:            ##signifying the two common errors
during runtime
21         return False
22
23 print("Is there an integer between %d and %d in the list? Answer: %s" % (number1, number2,
binarySearch(number1, number2, List)))

```

[CONTINUED]

Table 1:

Pseudocode	Time cost
number1 <- String input converted to Int	1
number2 <- String input converted to Int	1
List <- [4, 19, 23, 36, 40, 43, 61, 64, 78, 95]	1
binarySearch(num1, num2, array) {	
midpoint <- len(array)//2	1
If num1 <= num2 {	1
If array[midpoint] ≥ num2 AND array[midpoint] ≤ num2 {	1
Return TRUE	1
}	
ElseIf midpoint > num2 {	1
Return binarySearch(num1, num2, right half of array)	n
}	
ElseIf midpoint < num2 {	1
Return binarySearch(num1, num2, left half of array)	n
Return FALSE	1
}	
Else {	
Return "Error! Lower value > upper value"	1
}	
}	
Runtime bound:	= 2n + 11
Complexity:	O(n)

[CONTINUED]

Table2 – After considering how recursion affects runtime bound:

Pseudocode	Time cost	
number1 <- String input converted to Int	1	1
number2 <- String input converted to Int	1	1
List <- [4, 19, 23, 36, 40, 43, 61, 64, 78, 95]	1	1
binarySearch(num1, num2, array) {		
midpoint <- len(array)//2	n	n
If num1 <= num2 {	n	n
If array[midpoint] ≥ num2 AND array[midpoint] ≤ num2 {	n	n
Return TRUE	1	1
}		
ElseIf midpoint > num2 {	n	n
Return binarySearch(num1, num2, right half of array)	n	n
}		
ElseIf midpoint < num2 {	n	n
Return binarySearch(num1, num2, left half of array)	n	n
Return FALSE	1	1
}		
Else {		
Return "Error! Lower value > upper value"	1	1
}		
}		
Runtime bound:		$f(n) = 7n + 6$
Complexity:		$O(n)$

[CONTINUED]

Note 1:

If I specified the except statement as:

```
20         except IndexError and RecursionError:
```

then the program, when executed, would have first encountered an error that would be both an IndexError and RecursionError, by first identifying the IndexError and rendering it mute without the user's awareness and then proceeding to the identifying recursion error as it would also be the case.

As a solution I substituted to above statement to:

```
20         except IndexError or RecursionError:
```

The solution is mainly based on a test run of the program and entering a negative number for the lower range bound. The attempt was carried out with the except statement written before the amendment, as shown in the first instance, and when running the program, the stack report had returned an IndexError regardless of the written except statement. This led to the second instance as the solution, which indicates that the program would the code would cause either error instead of both, thus finally displaying the expected result.

Note 2:

Initial analysis concluded that the time taken for some of the code is constant, yet when the first condition (base case) is not met and the program executes the code in lines 14 or 16, the function calls itself thus triggering a recursion. Due to this, the program re-executes the code before the call until the base case is met, which arguably therefore alters the time complexity of the said code to **$O(\log n)$** . Nevertheless, after much consideration, the worst-case scenario is taken in account so the complexity of the program algorithm would result to $O(\log n)$ regardless.

10. Extracting the sub-sequence of maximum length (ascending order):

```
01     array = [71, 41, 15, 68, 49, 9, 26, 46, 15, 53, 96, 23, 54, 17, 11, 5]
02
03     print(array, end="\n\n")
04
05     def subSeq(list1):
06
07         '''Finds and returns ordered sub-sequence within a given list'''
08
09         newList = [] # new list for ordered sub-sequence
10
11         while len(list1) >= 2:
12             try:
13                 lowest = list1[0] # lowest value is first value in given list by default (temp)
14                 if lowest > list1[1]:
15                     lowest = list1[1] # value in second index of the list is now the lowest
16                     list1 = list1[1:] # list sequence now starts from point of new lowest value
17             else:
18                 for i in range(1, len(list1)):
19                     if lowest > list1[i]: # checks for smaller integers than lowest value in given
list
20                         lowest = list1[i] # value in index i is now the lowest value
21                         newList.append(lowest) # adds new lowest value to new list
22                         list1 = list1[i+1:] # list now starts from index adjacent to that of lowest
value
23                 for j in range(len(newList[:i])): # code in for-loop deletes every value before
last value of new list (lowest) if a smaller value is found
24                     if newList[-1] < newList[j]:
25                         del newList[:i]
26                         break
27                 break
28             if len(list1) == 2:
29                 if newList[-1] < list1[-2] and newList[-1] < list1[-1]:
30                     # checks if last value in new list is smaller than last two elements in given list;
31                     # if otherwise, following statements are disregards them + returns new list
```

```
32         if list1[0] < list1[1]:           # orders remaining values if size length of list is
two
33             newList.append(list1[0])
34             newList.append(list1[1])
35         else:
36             newList.append(list1[1])
37         break
38     except IndexError:
39         break
40
41     return newList
42
43     print(subSeq(array))
```

Note:

I could have adjusted the program to run through both the main and new list to check if there is a smaller value in the main list than the last value added to the new list, but bigger than the second to last value and onwards (in the new list). This would have made the function find a more closely consecutive sub-sequence. This could not have been fully accomplished due to great difficulty and time-consuming attempts.

12. Rendering IN_ORDER function as iterative from recursive:

```
34 def in_order(tree):
35     currentNode = tree
36     stack = []
37
38     while currentNode:
39
40         if currentNode.left != None:
41             stack.append(currentNode.value)
42             currentNode = currentNode.left
43
44         print( currentNode.value )
45
46         if currentNode.right != None:
47             print(stack[-1])
48             stack.pop()
49             currentNode = currentNode.right
50
51
52     ##if currentNode.right != None:
53     ##    stack.pop()
54
55     if stack == []:
56         print("break")
57         break
```

Appendix:

1. Binary Search tree template - <http://pastebin.com/LXdWF0KW>
2. Live code (Git repository) – <https://github.com/Nathan-Zenga/210CT-Coursework-tasks>