2016/2017

Programming, Algorithms and Data Structures (210CT)

Coursework

MODULE LEADER: Dr. Diana Hintea

Student Name: Myles Rankin
SID: 6328197
I can confirm that all work submitted is my own: Yes

This coursework is testing your ability to design algorithms, write pseudocode, describe their efficiency, implement various data structures and use the programming language of your choice for the implementation.

For any programming implementation that is not Pseudocode, I have chosen Python 3.5.x
GitHub 210CT Repository: *https://github.com/mylesrankin/210CT-Coursework*

Week 1

Task 1 - Function for shuffling an array randomly

Implementation (Python 3.5.x):

```
import random as r

def shuffleArray(array):
    newIndex = r.sample(range(len(array)), len(array)) # Creates new indexes using sample
    shuffled = [] # Create a new array for shuffled result
    for j in newIndex:
        shuffled.append(array[j]) # Inserts into new indexes
    return(shuffled)

# Example run
# shuffleArray([5,3,8,6,1,9,2,7])
```

In this implementation I used a function that requires an array of numbers as an input, then a random sample from the random library in Python is used to make a new set of indexes. The new set of indexes are shuffled randomly from the old then for each item in the list, they are put into a shuffled array in the order of the new indexes.

Task 2 - Function for count the number of trailing zeros in a factorial number

Implementation (Python 3.5.x):

```
import math

def countZeros(i): # Enter i as factorial i.e. 5
    j = math.factorial(i)
    j = str(j) # turn the factorial number into a string so it can be iterated through
    count = 0
    zero = True # Create a bool to set when a zero is found and when not a zero
    for i in reversed(j): # Reverse string
        if i != "0": # Checks for zeros until a whole number is found
            zero = False # Set if non zero is found
        else:
            if zero == True: # Step up zero counter
                count +=1 # Raise zero counter if zero is detected
    return count # Return number of trailing zeros from counter
```

Week 2

Task 1 - Pseudocode for a function that returns the highest square less/equal to a positive parameter, and then implement in chosen language

Pseudocode:

```
HIGHEST-PERFECTSQUARE(a) # a gets input of positive number
       IF a > 0 # check if above zero (non-negative)
              b <- (a)^(1/2) # root input
              b <- b-(b%1) # take away remainder
       ELSE
              RETURN "Enter a number greater than zero."  # return error if -ve value
       RETURN(b*b) # square b to get highest perfect square
```

Implementation (Python 3.5.x):

```python
def highestSquare(a):
    if a > 0:
        b = a**(1/2)  # root input
        b = b-(b%1)   # take away remainder
        return int(b*b) # square b to get highest perfect square
    else:
        return("Enter a number that is greater than zero.")
```

Task 2 - Describe run-time bounds of previous weeks algorithms using Big-O notation

Week 1 task 1 with run-time bounds - Implementation (Python 3.5.x):

```python
import random as r

def shuffle(numbers):
    newIndex = r.sample(range(len(numbers)), len(numbers))    1
    shuffled = []                                             1
    for j in newIndex:                                        n
        shuffled.append(numbers[j])                           n
    return shuffled                                           1
                                                              =2n+3,
                                                              therefore O(n)
                                                              complexity
```

Week 1 task 2 with run-time bounds - Implementation (Python 3.5.x):

```
import math

def countZeros(i):
    j = math.factorial(i)                 n
    j = str(j)                            1
    count = 0                             1
    zero = True                           1
    for i in reversed(j):                 n
        if i != "0":                      n
            zero = False                  n
        else:                             n
            if zero == True:              n
                count +=1                 n
    return count                          1
                                          =8n+4, therefore O(n)
                                          complexity
```

Task 3 - Psuedocode of functions to add, subtract and multiply two matrices

Pseudocode

```
ADD_MATRIX(M1,M2)
    RESULT <- EMPTY MATRIX # Imagine matrix as 2d array i.e. [[1,2][3,4]]   1
    FOR I IN (0 TO LEN(M1)-1)                                               n
        FOR J IN (0 TO LEN(I)-1)                                            n*n
            RESULT[I][J] <- M1[I][J] + M2[I][J]                             n*n
    RETURN RESULT                                                           1

SUB_MATRIX(M1,M2)
    RESULT <- EMPTY MATRIX # Imagine matrix as 2d array i.e. [[1,2][3,4]]   1
    FOR I IN (0 TO LEN(M1)-1)                                               n
        FOR J IN (0 TO LEN(I)-1)                                            n*n
            RESULT[I][J] <- M1[I][J] - M2[I][J]                             n*n
    RETURN RESULT                                                           1

MULT_MATRIX(M1,M2)
    RESULT <- EMPTY MATRIX                                                  1
    FOR I IN (0 TO LEN(M1)-1 # Iterates through rows of M1                  n
        FOR J IN (0 TO LEN(M2[0])) # Iterates through columns of M2         n*n
              FOR K IN (0 TO LEN(M2)) # Iterates through rows of M2         n*n*n
                  RESULT[I][J] <- M1[I][K] * M2[K][J] # row * column        n*n*n
    RETURN RESULT                                                           1

A <- INPUT(MATRIX_A)                                                        1
B <- INPUT(MATRIX_B)                                                        1
C <- INPUT(MATRIX_C)                                                        1

# A <- B*C+2*(B+C) to S1 = B*C, S2 = B+C, S3 = S2*2, R = S1+S3
# Split the expression into segments
S1 <- MULT_MATRIX(B,C)                                                      1
S2 <- ADD(B,C)                                                              1
S3 <- ADD(S2,S2)                                                            1
RESULT <- ADD(S1,S3)                                                        1

PRINT(RESULT)                                                               1
```

© Myles Rankin 2016 - Coventry University

Run-time complexities in Big-O notation for each matrix function:

- ADD_MATRIX - $2n^2 + n + 2 = O(n^2)$ *Complexity*
- SUB_MATRIX - $2n^2 + n + 2 = O(n^2)$ *Complexity*
- MULT_MATRIX - $2n^3 + n^2 + n + 2 = O(n^3)$ *Complexity*
- Overall run-time will be in the order of $O(n^3)$

Week 3

Task 1 - Write psuedocode for a function that reverses the words in a sentence

Pseudocode

```
REVERSEWORDS(sentence)
    sentence <- SPLIT(sentence)               1
    result <- NEW ARRAY                       1
    step <- -1                                1
    FOR I IN SENTENCE                         n
        APPEND TO RESULT(sentence[step])      n
        step <- step - 1                      n
    RETURN JOIN(result)                       1
                                              =3n+3, therefore O(n) complexity
```

Implementation (Python 3.5.x):

```
def reverseWords(sentence):
    sentence = sentence.split()     # split list into words
    result = []     # create list for the reversed result
    step = -1       # start from -1 on the step as this is the end of the list
    for i in sentence: # loop through all letters
        result.append(sentence[step]) # append letters starting from the end
        step -= 1  # move step value down to select the next from the end of list

    return ' '.join(result) # join array together and return
```

Task 2 - Write psuedocode and implemented code for a recursive function to check if a number is prime

Pseudocode:

```
PRIMECHECK(n,d<-2)
    IF n = 1
        RETURN True
    ELSE IF n = d
        RETURN True
    ELSE IF (n MOD d) = 0
        return False
    ELSE
        RETURN PRIMECHECK(N,D+1)
```

© Myles Rankin 2016 - Coventry University

Implementation (Python 3.5.x):

```
import sys
sys.setrecursionlimit = 2000 # Python recursion limit very low, be wary of stack overflows
def primeCheck(n,d=2): # n is number to check, d is set default to 2 no need to input
    if n == 1: # 1 is always a prime number
        return True
    elif n == d: # if step value equals d then all numbers to n have been checked
        return True
    elif (n%d) == 0: # n%d=0 implies n is div by another number so not prime
        return False
    else:
        return primeCheck(n,d+1) # Recursive 'step up' d value to n


# Example of checking first 50 prime numbers
for i in range(1,50):
    print(i)
    print(primeCheck(i))
```

Task 3 - Write psuedocode and implemented code for a recursive function that removes all vowels from a given string

Pseudocode:

```
REMOVEVOWELS(n,d<-0)
    IF d = LENGTH(n)
        RETURN n
    ELSE IF LOWERCASE(n[0]) IN ["a","e","i","o","u"]
        RETURN REMOVEVOWELS(n[1:],d)
    ELSE
        RETURN REMOVEVOWELS(n[1:]+n[0],d+1)
```

Implementation (Python 3.5.x):

```
def removeVowels(n,d=0): # n is string input,d=0 is stepper value which is left as default 0
    if d == (len(n)): # checks if stepper value is equal to list size
        return n
    elif n[0].lower() in ["a","e","i","o","u"]:
# ^ Check if current position in list is a vowel
        return removeVowels(n[1:],d) # return list apart from position 0
    else:
        return removeVowels((n[1:]+n[0]),d+1)
# ^ return list but with n[0] at end so i.e. input vowels = owelsv

# Example of removing vowels from a string
print(removeVowels("Ellie"))
```

© Myles Rankin 2016 - Coventry University

Week 4

Task 1 - Adaption of binary search to output a value within an interval if found and give time complexity using Big-O notation (Pseudocode and Implemented code)

Pseudocode:

```
BINARYSEARCH(array,lower,upper)
    start <- 0
   end <- LENGTH(array)
    found <- FALSE
    WHILE (start <= end) AND (found = FALSE)
        midVal = FLOORDIVIDE((start+end),2)
        IF (array[midVal] <= UPPER) AND (array[midVal] >= lower)
            found <- TRUE
            RETURN True
        ELSE
            IF lower < array[midVal]
                end <- midVal - 1
            ELSE
                start <- midVal + 1
    IF found = FALSE
        RETURN False
```

Implementation (Python 3.5.x):

```python
def binarySearch(l,lower,upper):        # assumes input list (l) is sorted in order
    start,end = 0,len(l)-1                      # start and end values of the list
    found = False                               # set found as false
    while start <= end and not found:
# ^ whilst start is less than end, and found isn't true loop through binary search
        midVal = (start+end)//2                 # Finds midpoint with a floor div (//)
        if (l[midVal] <= upper) and (l[midVal] >= lower) :
# ^ Checks value is between the interval provided
            found = True
# ^ set found as True because value is within interval
            return True
        else:
            if (lower < l[midVal]):
# ^ if the middle value is less than lower interval
                end = midVal-1
# ^...change the end interval to one less than midval,reducing the binary search half
            else:
                start = midVal+1
# ^ same as above but for the lower half is being removed
    if found == False:
# ^ when found is false return False (value not found in interval)
        return False

# example binarySearch([2,3,5,7,9,13],10,14) will yield True
```

Time complexity for **binarySearch()** - $o(\log n)$ due to the ‘divide & conquer’ halving of the input.

© Myles Rankin 2016 - Coventry University

Week 5

Task 1 - Write a function to extract subsequences of maximum length which are in ascending order

Implementation (Python 3.5.x):

```
class stack(): # create a standard stack class
    def __init__(self):
        self.items = []
        self.all = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def returnAll(self):
        return self.items
    def size(self):
        return len(self.items)

def selectHighest(array): # function to select highest sub sequence in 2d array
    arrayLen = 0 # set array length 0 as default
    highestSequences = [] # create a new array to handle highest sequences
    for i in array: # for each item in input array
        if len(i) > arrayLen:
            arrayLen = len(i) # sets array length if higher than current
            highestSequences = [] # flushes current highestSequences
            highestSequences.append(i) # appends new highest sequence
        elif len(i) == arrayLen: # if two sub-s are the same size
            highestSequences.append(i) # just appends not flush if size is equal
    return(highestSequences) # returns the highest sequences


def sequences(l):
    a = []
    s = stack() # create new stack
    for i in range(0,len(l)):
        if i == (len(l)-1): # flushes stack to result when at end of sequence list
            s.push(l[i]) # push current item to stack
            a.append(s.returnAll()) # append stack to result array (a)
        elif l[i] < l[i+1]: # if current item less than next then push to stack
            s.push(l[i]) # push item to stack
        elif l[i] > l[i+1]: # if item is more than next
            s.push(l[i]) # push item to stack
            a.append(s.returnAll()) # append stack to result array
            s = stack() # empty stack for next iteration
    return(a) # return resultant array of seperated sequences

# Seperate sub-sequences
# sequences([1,2,3,4,1,2,3,6,5,6,7,8])

# Seperate sub-sequences and return highest sequence(s)
# selectHighest(sequences([1,2,3,1,2,3,6,5,6,7,8]))
```

Task 2 - Based on the code in class, implement the double linked list delete function

Implementation (Python 3.5.x):

```
def remove(self,n): # Added remove function
      if n.prev !=None:
            n.prev.next = n.next # change so pointer skips removed element*
      else:
            self.head = n.next
      if n.next !=None:
            n.next.
            prev = n.prev
      else:
            self.tail = n.prev
```

Week 6

Task 1 - Implement TREE_SORT algorithm and make sure INDORDER function is implemented iteratively

Implementation (Python 3.5.x):

```
class BinTreeNode(object):

    def __init__(self, value):
        self.value=value
        self.left=None
        self.right=None



def tree_insert( tree, item):
    if tree==None:
        tree=BinTreeNode(item)
    else:
        if(item < tree.value):
            if(tree.left==None):
                tree.left=BinTreeNode(item)
            else:
                tree_insert(tree.left,item)
        else:
            if(tree.right==None):
                tree.right=BinTreeNode(item)
            else:
                tree_insert(tree.right,item)
    return tree

def postorder(tree):
    if(tree.left!=None):
        postorder(tree.left)
    if(tree.right!=None):
        postorder(tree.right)
    print(tree.value)
```

```
def in_order(tree):
    stack = [] # create a stack
    finished = False
    while(finished == False): # while ordering is not finished
        if tree != None: # if the tree value is a number
            stack.append(tree) # add to the stack
            tree = tree.left # move pointer to the left
        else:
            if(len(stack) > 0): # if stack is not empty
                tree = stack.pop() # pop value from stack
                print(tree.value) # print tree value from stack
                tree = tree.right # move pointer to the right
            else:
                finished = True # set finished to true when finished ordering

if __name__ == '__main__': # initialise a tree

  t=tree_insert(None,6); # tree here and below insert nodes
  tree_insert(t,10)
  tree_insert(t,5)
  tree_insert(t,2)
  tree_insert(t,3)
  tree_insert(t,4)
  tree_insert(t,11)
  in_order(t)
```

Week 7

Task 1 - Write Pseudocode for an unweighted graph data structure, with add node and edge functions.

Pseudocode - Using an adjacency list approach

```
CLASS GRAPH()
    INITIALISE CLASS()
        graphDictionary <- NEW DICTIONARY

    ADDVERTEX(LABEL)
        IF LABEL IS NOT IN graphDictionary
            graphDictionary[LABEL] = NEW LIST
        ELSE
            RETURN ERROR "Vertex already exists"

    ADDEDGE(VERTEX1,VERTEX2)
        graphDictionary[VERTEX1].APPEND(VERTEX2)
        graphDictionary[VERTEX2].APPEND(VERTEX1)

    DISPLAYADJENCYLIST()
        FOR EACH KEY AND VALUE IN graphDictionary
            PRINT KEY + " | " VALUE
```

© Myles Rankin 2016 - Coventry University

Task 1&2 - Implemented graph structure in Python3.5.x with DFS and BFS traversals.

Implementation (Python 3.5.x):

```
class Stack:
     def __init__(self):
         self.items = []

     def isEmpty(self):
         return self.items == []

     def push(self, item):
         self.items.append(item)

     def pop(self):
         return self.items.pop()

     def peek(self):
         return self.items[len(self.items)-1]

     def size(self):
         return len(self.items)

class Queue:
     def __init__(self):
          self.items = []

     def isEmpty(self):
          return self.items == []

     def enqueue(self, item):
          self.items.insert(0,item)

     def dequeue(self):
          return self.items.pop()

     def size(self):
          return len(self.items)

class Graph:
     def __init__(self):
        self.graphDict = {}

     def addVertex(self,label):
          if label not in self.graphDict:
               self.graphDict[label] = []
               return("Vertex created")
          else:
                 return("Vertex exists")

     def addEdge(self,vertex1,vertex2):
          self.graphDict[vertex1].append(vertex2)
          self.graphDict[vertex2].append(vertex1)
          return("An edge has been created between",vertex1," and ",vertex2)

     def displayGraph(self):
          return self.graphDict

     def displayAdjacencyList(self):
          for k,v in self.graphDict.items():
               print(k , "|", v)

     def depthFirstSearch(self,startNode):
          s = Stack()
          visited = []
          s.push(startNode)
          while s.isEmpty() == False:
               u = s.pop()
               if u not in visited:
                    visited.append(u)
                    for e in self.graphDict[u]:
                         s.push(e)
          f = open("dfs.txt","w")
          f.write(str(visited))
          f.close
          return visited

     def breadthFirstSearch(self,startVertex):
          q = Queue()
          visited = []
          q.enqueue(startVertex)
          while q.isEmpty() == False:
               u = q.dequeue()
               if u not in visited:
                    visited.append(u)
               for edge in self.graphDict[u]:
                    if edge not in visited:
                         q.enqueue(edge)
          f = open("bfs.txt","w")
          f.write(str(visited))
          f.close
          return visited



if __name__ == "__main__":
    g = Graph()
    g.addVertex("a")
    g.addVertex("b")
    g.addVertex("c")
    g.addVertex("d")
    g.addVertex("e")
    g.addEdge("a","b")
    g.addEdge("a","e")
    g.addEdge("a","c")
    g.addEdge("b","e")
    g.addEdge("b","d")
    g.displayAdjacencyList()
    print(g.depthFirstSearch("a"))
    print(g.breadthFirstSearch("a"))
```

© Myles Rankin 2016 - Coventry University