

2016/2017

**Programming, Algorithms and Data Structures (210CT)**

Coursework

**MODULE LEADER:** Dr. Diana Hintea

**Student Name: Hannah Rice**

**SID: 6238519**

1. **can confirm that all work submitted is my own: Yes**

**Question 1:**

import random

import sys

def random\_shuffle(A,shuffled,length):

""" Function to randomly shuffle an array of integers """

a = random.choice(A)

while length > 0: # Base Case for recursion

if a in shuffled:

A.remove(a)

random\_shuffle(A,shuffled,length)

return shuffled

else:

shuffled.append(a)

length = length-1

random\_shuffle(A,shuffled,length)

return shuffled

try: # Check for invalid input to avoid errors.

A = ([1,3,4,6,7,2]) # Input

shuffled = []

length = len(A)

random\_shuffle(A,shuffled,length)

except NameError:

print("Invalid input: Must be integers")

sys.exit()

except IndexError:

print("Invalid input: Must only be one of each number")

sys.exit()

print(shuffled)

**Explanation:**

I used a recursive function to randomly choose a value in A and then add that value into new array. I used a while loop to ensure that the new array would be the same length as the input. To ensure that each value in the array would only be used once and to increase the efficiency I decided to remove the value that had been selected from the input once it had been added to the new array. I added except statements to ensure that if invalid input was given there would be no errors. I avoided infinite recursion by with every call decreasing the lengths of the array.

**Question 2:**

import sys

def factorial\_trailing\_zeroes(n):

""" Function first detemines the factorial of input """

""" Then calculates the number of trailing zeroes """

if n > 0: # A factorial number must be greater than 0

factorial = 1

for i in range(1, n+1): # Loop range detemined by input

factorial = (factorial\*i) # Determines the factorial of input

# (1\*1 = 1), (1\*2 = 2), (2\*3 = 6), (6\*4 = 24), etc...

calc = (5\*\*(i-1)) # Determines trailing zeroes

# eg: (5\*\*0, 5\*\*1, 5\*\*2, 5\*\*3, 5\*\*4)

print(calc)

calc = int(calc)

ans = (factorial/calc)

ans = int(ans)

trailing\_zeroes = (ans + 1)

print(str(n) + "!" + " = " + str(factorial))

print("The number of trailing zeroes are " + str(trailing\_zeroes))

else:

print("Input must be greater than zero")

try:

n = (5)

factorial\_trailing\_zeroes(n)

# Expect potential false input

except NameError:

print("Invalid Input: Must be a whole number")

sys.exit()

except TypeError:

print("Invalid Input: Must be an integer")

sys.exit()

**Explanation:**

I used a ‘for loop’ to multiply the numbers lower than the input until the factorial was found. Then I used the same loop to calculate the number of trailing zeroes. Due to the input, only being one number I had to allow for input errors such as negative numbers which would raise an error.

**Question 3:**

**Pseudocode:**

HIGHEST\_PERFECT\_SQUARE (n)

ans <- 0

if n >= 0

while ans\*ans < n

ans <- ans + 1

if ans\*ans = n

return(n)

else

n <- (n-1)

repeat(n)

else:

return n

**Python:**

import sys

def highest\_perfect\_square(enter):

""" Returns the highest perfect square which is less than or equal to input """

factor = 0

if enter >= 0:

while factor\*factor < enter:

factor = factor + 1 # 1\*1, 2\*2, 3\*3, etc

if factor\*factor == enter: # Either the first input or less than n

print(str(enter) + " is a perfect square number")

else:

enter = enter-1 # counts down from n, until perfect square

highest\_perfect\_square(enter)

else:

print("invalid input")

try:

enter = (36)

highest\_perfect\_square(enter)

except NameError:

print("Invalid Input")

sys.exit()

except TypeError:

print("Invalid Input")

sys.exit()

**Explanation:**

I used a while loop to multiply all numbers lower than the given input together to determine square numbers. The input is also decreases with each recursive call so that the next highest square that is now equal to the new input can be found. I used zero as the bases case the answer must be greater than zero.

**Question 4:**

random\_shuffle = O(n)/O(n2)orO(n^n)

factorial\_trailing\_zeroes = O(n^n) nested loops are exponentially expensive

**Question 5:**

**Pseudocode:**

MATRIX\_ADDITION(B,C)

for i in range(len(B)) n

for j in range(len(B[i])) n\*n

A[i][j] <- B[i][j] + C[i][j] n\*n

MATRIX\_SUBTRACTION(B,C)

for i in range(len(B)) n

for j in range(len(B[i])) n\*n

A[i][j] <- B[i][j] – C[i][j] n\*n

MATRIX\_MULTIPLICATION(B,C)

for i in range(len(B)) n

for j in range(len(B[i])) n\*n

A[i][j] <- B[i][j] \* C[i][j] n\*n

**Python:**

import sys

answer = [[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]] # Empty matrix to store answer

# Matricies to store resuts of calculations

calc\_first\_half = [[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]

calc\_second\_half = [[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]]

def matrix\_addition(calc\_first\_half, B, C):

""" Add all values in matrix B and C, store in calc\_first\_half """

for i in range(len(B)): # Adds each element in B with C

for j in range(len(B[0])):

calc\_first\_half[i][j] = B[i][j] + C[i][j] # Stores result

print("")

print("B + C = ")

print(calc\_first\_half)

matrix\_multiplication\_second(calc\_first\_half,B,C)

def matrix\_subtraction(answer, calc\_first\_half, calc\_second\_half):

""" Subtracts all values in calc\_first\_half with calc\_second\_half """

for i in range(len(B)):

for j in range(len(B[0])):

answer[i][j] = calc\_first\_half[i][j] - calc\_second\_half[i][j]

# Stores result in "answer" which is the result of the given calculation

def matrix\_multiplication(calc\_second\_half, B, C):

""" Multiplies all B values with C values """

for i in range(len(B)):

for j in range(len(B[0])):

calc\_second\_half[i][j] = B[i][j] \* C[i][j]

print("")

print("B \* C = ")

print(calc\_second\_half)

def matrix\_multiplication\_second(calc\_first\_half, B, C):

""" Multiplies B + C by 2 """

for i in range(len(B)):

for j in range(len(B[0])):

calc\_first\_half[i][j] = calc\_first\_half[i][j] \* 2

print("")

print("2 \* (B+C) = ")

print(calc\_first\_half)

# Matricies for calculations

try:

B = [[1,2,3,4], [5,6,7,8], [9,10,1,2]],[3,4,5,6]]

C = [[5,8,2,4], [6,1,8,2], [2,3,4,9], [10,7,9,5]]

if len(B) == len(C):

print("A = B\*C - 2\*(B+C) ")

matrix\_addition(calc\_first\_half, B, C)

matrix\_multiplication(calc\_first\_half, B, C)

matrix\_subtraction(answer, calc\_first\_half, calc\_second\_half)

print("")

print("Answer :")

print(answer)

else:

print("The matricies must be the same size")

except NameError:

print("Invalid Value in Matrix")

sys.exit()

**Explanation:**

I gave three empty matrices, one to store the answer and the other two to store the answer to each half of the given calculation. First to be calculated is (B+C) which uses function matrix\_addition. The result is stored in one of the empty matrices. Next matrix\_multiplication\_second which performs 2\*(B+C) using the result from the previous function. Then I moved to the second half of the calculation matrix\_multiplication which computes B\*C. Finally, now that both parts of the calculation are done and we have the result of both stored in two separate matrices we can then subtract them to provide the complete answer in matrix A.

To perform calculations with the matrices I used two ‘for loops’ to iterate over each element in the matrices and multiply, add, or subtract each element of the matrices and then be stored in a new matrix. I also added an ‘if’ statement at the start to ensure that both matrixes are the same size. I decided to print the operations and results of each step of the computation simply to explain the code clearer.

**Question 6:**

**Pseudocode:**

REVERSE\_SENTENCE(S)

X <- Length(S)

If X <= 0

return S

else

X <- X-1

S <- S

reverse\_sentence(S[1:]) to S[0:]

return S

S <- (“This”, “is”, “awesome”)

reverse\_sentence(S)

**Python:**

import sys

def reverse\_sentence(s):

""" Takes s as input then reverses the string """

sLength = len(s)

if sLength <= 0: # Stops infinite recursion

# The function stops once the sentence in reversed

return s

else:

sLength = sLength-1

s = s

reverse\_sentence(s[1:]) + s[0:] # Recalls the function with new '0'

position = s[0] # Saves position 0 value

# S[0] changes each itteration

print(position)

return s # Continues through the input

sentence = ("This is awesome") # Original input

if type(sentence) != str:

print("Input must be a string")

sys.exit()

s = sentence.split() # Split so that can index each word in sentence

print(sentence)

print("")

print("REVERSED: ")

print("")

reverse\_sentence(s)

**Explanation:**

First, I had to split the string so that each word could be accessed individually. Then I created a copy of the sentence so that I could keep the original to print later. Then the function is recalled and the last word in the string is now the first. The length is always decreasing to avoid infinite recursion.

**Question 7:**

**Pseudocode:**

PRIME(n,i)

If n > 1 and i > 1

ans <- n % i

if ans != 0

i <- i – 1

prime(n, i)

return i

elif ans = 0

not a prime number

else

prime number

n <- 3

i <- n – 1

prime(n,i)

**Python:**

import sys

def prime(enter, x):

""" Calculates if the input is a prime number or not """

if enter > 1 and x > 1: # prime number must be greater than 1. BASE CASE

result = enter % x # enter / 1 but disregards the remainder

if result != 0:

x = x-1 # checks n / every number below n until reach 0

prime(enter,x) # recalls the function

return x

elif result == 0:

print(str(enter) + " is not a prime number")

else:

print(str(enter) + " is a prime number")

try:

enter = 3

if enter > 1:

x = enter - 1

prime(enter,x)

else:

print("Input must be greater than 1")

except NameError:

print("Input must be an integer")

sys.exit()

except TypeError:

print("Input must be an integer")

sys.exit()

**Explanation:**

The base case is the fact that a prime number must be greater than one. The function decreases ‘x’ each time it is called allowing us to test if input multiplied by any number below input results in a prime. As long as x is greater than zero because then we know that it is not a prime number. I also accounted for input error for negative numbers and none integer types, to avoid errors.

**Question 8:**

**Pseudocode:**

REMOVE\_VOWELS(S,V,x,new)

If x > 0

x <- (x-1)

if V[x] in S

a <- V[x]

new <- new(remove(a))

remove\_vowels(S,V,x,new)

return x

else

remove\_vowels(S,V,x,new)

else

return x

S <- (“input”)

V <- (“a”,”e”,”i”,”o”,”u”)

x <- Length(V)

new <- S

output(S)

remove\_vowels(S,V,x,new)

**Python:**

import sys

def remove\_vowels(word, vowels, position, word\_copy):

"""Takes input and returns the input without the vowels"""

if position > 0: # Base case

position = position-1 # Itterates through vowels, also moves towards base case

if vowels[position] in word: # Each call checks if that vowel is in the input

a = vowels[position]

word\_copy = word\_copy.replace(a,"") # Takes out the vowel

print(word\_copy)

remove\_vowels(word,vowels,position,word\_copy) # Recalls the function

return position

else:

remove\_vowels(word,vowels,position,word\_copy) # Moves to next vowel

return position

else:

return position

word = ("beautiful")

if type(word) != str:

print("Input must be a string")

sys.exit()

vowels = ("a","e","i","o","u","A","E","I","O","U")

position = len(vowels)

word\_copy = word

print(word) # If there are no vowels, the input word will show

remove\_vowels(word,vowels,position,word\_copy)

**Explanation:**

I stored all vowels in a tuple so that each value could be indexed. Then I recalled the function to search for that value(vowel) in the word. If a vowel is found, then it is replaced from the input with an empty space. The new word without the already found vowel is now returned and this continues until all vowels have been searched for. If no vowels are found, then the original input is printed. I included the uppercase of each vowel as well in case of uppercase input.

**Question 9:**

def binary\_search(entry): # Divide and Conquer

""" Search through input for values within the given high & low parameters """

length = len(entry)

middle = length/2

middle = int(middle)

if entry[middle] == low or entry[middle] == high:

print("TRUE")

elif middle < 1: # Value not in range

print("FALSE")

else:

if entry[middle] > low and entry[middle] < high:

print("TRUE")

elif low < entry[middle]:

entry = entry[:middle] # Disregard first half of list

binary\_search(entry)

return entry

elif high > entry[middle]:

entry = entry[middle:] # Disregard second half of list

binary\_search(entry)

return entry

else:

print("FALSE")

try:

entry = [2,3,5,7,9,13]

low = 10

high = 14

if low < high:

binary\_search(entry)

else:

print("Low parameter must be less than High paramater")

except NameError:

print("Input must be all integers")

**Explanation:**

Divide the input in half to determine which half of the input list to disregard. Half of the list without is removed and then the function is recalled and the same technique starts again until a value between the given parameters is found, or until there is no values left and the result is false. In order for this to work just like the binary search algorithm the given input must be already sorted.