

Week 1 – Question 1

```
import random

array = [1,2,3,4,5,6]

shuffled = []

indexes = []

def shuffling():

    while len(indexes) < len(array): #O(N)    #Loops while lent of indexes is smaller than len of array

        a = random.randint(0,(len(array)-1))#creates a random integer that will be an index

        while a in indexes: #O(N*N)

            a = random.randint(0,(len(array)-1))#If that integer is already in indexes it creates a new one

        else:

            indexes.append(a)

    for i in indexes:

        shuffled.append(array[i])#loops for indexes and adds values to array according to the index

shuffling()

print("Shuffled List: ")

print(shuffled)

#Big O Notation = O(N*N)
```

Week 1 – Question 2

```
def trailing():

    number = int(input("Enter a number: ")) #(1)

    trailing = 0                #(1)

    for x in range (5,number+1):    #(N)

        fact = int(x)            #(N)

        while fact:              #(N*N)

            if fact % 5 == 0:      #(N*N)

                trailing+=1        #(N*N)

                fact = fact / 5     #(N*N)

            else:

                break

    print ("There are ", trailing, "zeros") #(1)

trailing()

#Program has a Big O Notation of O(N*N)
```

Week 2 – Question 1

```
def HPerfSquare(num1):

    if num1 >= 0:                #Checks if number is >= 0

        PerfSquare = num1**(1/2)    #If it is per square is input square rooted

        PerfSquare = PerfSquare - (PerfSquare%1) #Gets rid of the decimal

        PerfSquare = PerfSquare ** 2    #squares the number to make the biggest perf sq

        print(PerfSquare)            #prints it

        return PerfSquare            #Returns it

    elif num1 < 0:                #If number is -ve it just prints

        print ("Integer has to be positive")

HPerfSquare(50)
```

Week 2 – Question 2

Check the tasks in week 1 for the Big O notation (source code comments states it)

Week 2 – Question 3

NEEDS DOING

Week 3 – Question 1

```
def reverse():

    x = input ("Enter a sentence to reverse: ") #Gets a string to reverse          O(1)

    strList = x.split()    #Splits the string into words into a list              O(1)

    count = -1             #Counter...                                          O(1)

    revList = []           #Reversed list                                       O(1)

    for i in strList:      #Loops through the non-reversed list                 O(N)

        revList.append(strList[count]) #Appends the words from the list (from the back)  O(N)

        count -= 1         #Decreases the count so next value from the left is checked  O(N)

    print (' '.join(revList)) #Converts the reversed list into a string and prints it    O(N)

reverse()
```

#The Big O Notation for this algorithm is O(N) due to the for loop involved.

Pseudo Code –

```
FUNCTION reverse

    x <- USER INPUT

    NEW LIST strList <- LIST of characters in string x

    Counter <- -1

    NEW LIST revList <- EMPTY LIST

    FOR LOOP through strList

        APPEND strList[Counter] to revList

        Counter <- +1

    Convert revList to a string

    PRINT the string
```

Week 3 – Question 2

```
def prime(n,counter=2):    #Counter starts at 2 because if it started at 1 all numbers would be prime

    if n == 0:             # Base Case

        return True

    elif n == 1:           #If n is 1, return True because 1 is a prime number

        return True

    elif n == counter:     #If counter gets to n number is a prime as n%n and n%1 = Prime

        return True

    elif (n%counter) == 0: #If any counter value gets a 0 after doing modulo number is not a prime

        return False

    else:

        return prime(n,counter+1) #Recursion - Adds to the counter each time
```

Pseudo Code –

```
FUNCTION prime with Parameters and counter <- 2

    IF n = 0 DO

        RETURN True

    ELSE IF n = 1 DO

        RETURN True

    ELSE IF n = counter DO

        RETURN True

    ELSE IF (n MOD counter) = 0 DO

        RETURN False

    ELSE DO

        RETURN FUNCTION PRIME CALL with parameters n and counter <- +1
```

Week 3 – Question 3

```
def main():

    string = list(input("Please input a string: "))    #Gets the String from user

    counter = 0                                     #Sets the counter to 0

    print(vowelSearch(string,counter))              #Calls the removing function

def vowelSearch(string,counter):

    vowels = ['a','e','i','o','u']    #Array of vowels

    while counter < len(string):    #Base Case

        for i in vowels:            #Iterates through the vowels list

            if i in string:        #checks if currently checked vowel is in string

                string.remove(i)    #If it is , then it removes that vowel from the string

                vowelSearch(string, counter) #RECURSION!! - checks for the same vowel again

            else:

                counter += 1        #If it doesnt find the vowel again, it goes to the next vowel

    #Changes the list of characters into a single string

    finalString = ""

    for i in string:

        finalString += i

    return finalString            #Returns the result

main()
```

Pseudo Code-

FUNCTION main

 string <- USER INPUT

 convert string to list

 counter <- 0

 PRINT FUNCTION CALL vowelSearch with parameters string and counter

FUNCTION vowelSearch with parameters string and counter

 vowels (LIST) <- 'a','e','i','o','u'

 WHILE LOOP counter < Length of string

 FOR LOOP (i) loops through vowels list

 IF i is in vowels DO

 REMOVE i from string

 FUNCTION CALL with parameters string and counter

 ELSE DO

 counter <- +1

 finalString <- empty string

 FOR LOOP (i) loops through string

 finalString <- +i

 RETURN finalString

Week 4 – Question 1

def binarySearch(val1, val2, List): #Function takes v(value) and a list to iterate

 first = 0 #First Value

 last = len(List)-1 #Last Value

 found = False #Found Boolean

 while first <= last and not found: #Iterates while first value <= last value, and not found is True

 mid = int((first+last)/2) #Finds the midpoint

 if (List[mid] <= val2) and (List[mid] >= val1) :#Checks if midvalue is a val between given range

 found = True #Changes found to True when the value is found

 else:

 if (val1 < List[mid]): #If value is less than midvalue it

 last = mid-1 #Puts the last value to be the mid value (cuts list)

 else:

 first = mid+1 #Otherwise it makes the mid value the first value (cuts)

 if found == True: #Print statements to inform the user of the result

 print ("A number between the range has been found")

 else:

 print ("A number between the range was not found")

binarySearch(-1,0, [1,2,3,4,5,8,59])

Week 5 – Question 1

```
def longestSequence(sequence, pointer, SequenceList, final):

    for i in range(0, len(sequence)):

        if i == 0:                #always adds the first digit in the list

            pointer.append(i)

        elif i == (len(sequence)-1):    #stops iteration when the loop finishes the list

            pointer.append((len(sequence))-1)

        elif sequence[i] > sequence[i-1]:    #if the digit before previous is smaller dont append check next

            pass

        elif sequence[i] <= sequence[i-1]:    #if the digit is bigger before previous, append

            pointer.append(i-1)

            pointer.append(i)

    for x in range(0, len(pointer), 2):    #Iterates through the pointer list

        seq = [pointer[x], pointer[x+1]]    #Puts two pointers into a seperate list to use as coordinates

        SequenceList.append(seq)

    for i in SequenceList:                #Iterates through SequenceList

        finseq = []                        #List for each sequence

        for x in range (i[0], i[1]+1):    #Iterates through sequence list between the two indexes given in SequenceList

            finseq.append(sequence[x])    #Appends each number two the finseq value holder

        final.append(finseq)              #Appends the final sequence to final list

    print ("Pointer :", pointer)

    print ("SequenceList:", SequenceList)

    print('Longest subsequence = ', max(final, key = len)) #Finds the longest sequence using max and key len.

longestSequence([1,2,3,3,4,5,6,7,8,2,3,2,3,4,5,6,7,8,9,10], [], [], [])

#sequence = List of values to check

#pointer = Start and end of each sequence

#SequenceList = Coordinate list, basically pointer list but with start and end of sequences seperated

#final = List of lists of sequences
```

Week 5 – Question 2

```
def node_delete(self, n):

    if n.prev != 0:

        n.prev.next = n.next

    else:

        self.head = n.next

    if n.next != 0:

        n.next.prev = n.prev

    else:

        self.tail = n.prev
```

COMMENT IT

Week 6 – Question 1

```
def in_order(tree):  
    stack = []  
  
    finished = False  
  
    while (finished == False): #Keeps the loop going until the else statement  
        if tree != None:      #If tree isnt empty  
            stack.append(tree) #append the root node to the stack  
            tree = tree.left   #pointer goes to the left value  
        else:  
            if (len(stack) > 0):#if the length of the stack is more than 0  
                tree = stack.pop()#pop the most recent value from stack  
                print(tree.value)#and print it  
                tree = tree.right#pointer goes to the right value  
            else:  
                finished = True#if stack is empty, finish the loop
```

Week 7 – Question 1 & 2

```
class graph:  
    def __init__(self):  
        self.dictionary = {} #Creates a dictionary  
  
    def addVertex(self,vertex):  
        if vertex not in self.dictionary: #If input vertex isnt in dictionary it adds it  
            self.dictionary[vertex] = []  
        else:  
            pass #If it is it does nothing and ignores it  
  
    def addEdge(self,vertex,edge): #Adds an edge to the graph using 2 points  
        self.dictionary[vertex].append(edge)#Adds an edge to a vertex  
        self.dictionary[edge].append(vertex)#Adds the vertex to the edge value (has to work both ways for AL)  
  
    def printDict(self):  
        for key in self.dictionary: #Function just prints out the dictionary one value under another  
            print(key, ': ', self.dictionary[key])  
  
    def DFS(self, vertex):  
        self.visited = [] #List storing all the values that have been visited  
        self.stack = [] #Creates a stack for backtracking and moving between nodes  
        self.stack.append(vertex) #Adds the starting vertex to the stack  
        while self.stack != []: #While the stack isnt empty...  
            u = self.stack.pop() #pops the value and puts it into value holder u  
            if u not in self.visited: #if u isnt already in the visited list...  
                self.visited.append(u) #it appends u to that list  
                for edge in self.dictionary[u]:#it also loops through all of the edges of that vertex  
                    self.stack.append(edge) #Pushes those edges onto the stack  
  
        BFS_Text = open("DfsOutput.txt", "w")  
        BFS_Text.write("DFS traversal: %s " % self.visited)  
        BFS_Text.close()  
  
    def BFS(self,vertex):  
        self.q = [] #Creates a list q  
        self.visited = [] #List of already visited nodes  
        self.q.insert(0, vertex) #adds the starting point to the queue  
        while self.q != []: #While the queue (q) isnt empty...  
            u = self.q.pop() #it pops the value from the queue and holds it in u
```

```

        if u not in self.visited:      #if u isnt already in the visited list...

            self.visited.append(u)      #it appends u to visited

            for edge in self.dictionary[u]: #loops through the edges of vertex u

                self.q.insert(0,edge)    #inserts them into the queue

BFS_Text = open("BfsOutput.txt", "w")

BFS_Text.write("BFS traversal: %s " % self.visited)

BFS_Text.close()

```

```

if __name__ == '__main__':

```

```

    g = graph()

```

```

## 7 : 6          9

```

```

## 6 :   7   8   9

```

```

## 8 : 6          9

```

```

## 9 : 6   7   8

```

```

    g.addVertex(7)

```

```

    g.addVertex(6)

```

```

    g.addVertex(8)

```

```

    g.addVertex(9)

```

```

    g.addEdge(6,7)

```

```

    g.addEdge(6,8)

```

```

    g.addEdge(6,9)

```

```

    g.addEdge(7,9)

```

```

    g.addEdge(9,8)

```

```

    g.printDict()

```

```

    g.DFS(7)

```

```

    g.BFS(7)

```

#FOR BFS I have used a list instead of an actual queue to save the amount of code written as a list can be used

#as a queue if you insert values at index 0 and pop values from the end

#Therefore the first value is always the end of the queue and last value is the start of the queue