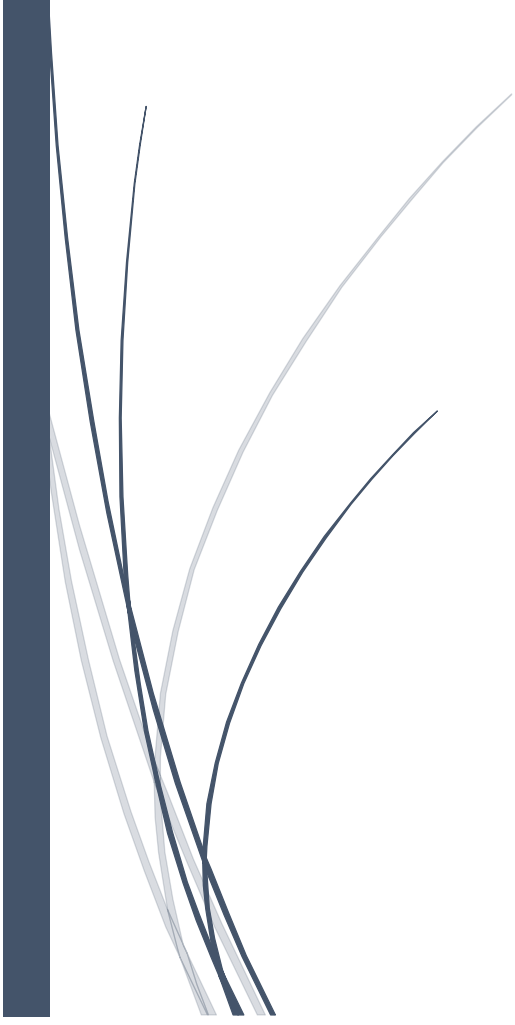
6/1/2021

Data Structures & Algorithms Coursework

ICT 1018 Assignment



Matteo Sammut 0206002L

Table of Contents

The Statement of Completion

- **Question 1 – Attempted and works well**
- **Question 2 – Attempted and works well**
- **Question 3 – Attempted and works well**
- **Question 4 – Attempted and works well**
- **Question 5 – Attempted and works well**
- **Question 6 – Attempted and works well**
- **Question 7 – Attempted and works well**
- **Question 8 – Attempted and works well**
- **Question 9 – Attempted and works well**
- **Question 10 – Attempted and works well**
- **Question 11 – Attempted and works well**
- **Question 12 – Attempted and works well**

Signed Matteo Sammut:

Plagiarism Declaration

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I/ We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Matteo Sammut
Student Name

Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

ICT1018
Course Code

Data Structures & Algorithms Assignment
Title of work submitted

01/06/2021
Date

Question 1

Code

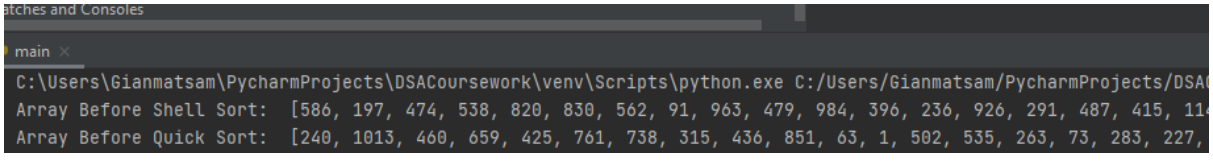
```
import random

asize = random.randint(256, 1024)
bsize = random.randint(256, 1024)
# generate the size of each array

A = []
B = []

while asize == bsize:
    bsize = random.randint(256, 1024)
# if both arrays are the same size array B is reset

for i in range(asize):
    A.append(random.randint(0, 1024))

for i in range(bsize):
    B.append(random.randint(0, 1024))
# both arrays are populated

print("Array Before Shell Sort: ", A)
print("Array Before Quick Sort: ", B)

# shell sort
gap = asize // 2
while gap > 0:
    for index in enumerate(A[gap:]):

        if A[index[0]] > A[gap + index[0]]:
            hold = A[index[0]]
            A[index[0]] = A[gap + index[0]]
            A[gap + index[0]] = hold
    gap = gap // 2
# insertion sort
for index in enumerate(A[0:]):
    temp = index[1]
    aIndex = index[0] - 1
    aValue = index[0]
    while aIndex >= 0 and temp < A[aIndex]:
        A[aValue] = A[aIndex]
        aIndex -= 1
        aValue -= 1
    A[aIndex + 1] = temp


# quick sort
def pivotpos(B, first, last):
    # choose pivot using first middle last
    pvot = last - 1
    # sort first last and middle
    if B[last] < B[pvot]:
        hold = B[last]
        B[last] = B[pvot]
```

```
        B[pvot] = hold
    elif B[pvot] > B[last]:
        hold = B[pvot]
        B[pvot] = B[last]
        B[last] = hold
    if B[pvot] < B[first]:
        hold = B[pvot]
        B[pvot] = B[first]
        B[first] = hold

    # swap pivot with last value
    hold = B[pvot]
    B[pvot] = B[last]
    B[last] = hold
    # set i and j to starting positions
    left = first
    # -1 because otherwise right would be the pivot
    right = last - 1
    while True:
        # i moving until it finds a value greater than the pivot
        while B[left] < B[last]:
            left += 1
        while B[right] >= B[last]:
            right -= 1
        # when i passes j, swap i with the pivot and stop
        if left > right:
            hold = B[left]
            B[left] = B[last]
            B[last] = hold
            break
        # swapping i and j
        hold = B[left]
        B[left] = B[right]
        B[right] = hold
    return left

def quickSort(B, first, last):
    if last < first+1:
        return B
    middle = pivotpos(B, first, last)
    # rearrange
    quickSort(B, first, middle-1)
    quickSort(B, middle+1, last)


quickSort(B, 0, bsize-1)
print("Array After Shell Sort: ", A)
print("Array After Quick Sort: ", B)
```

Explanation

Creating the Arrays

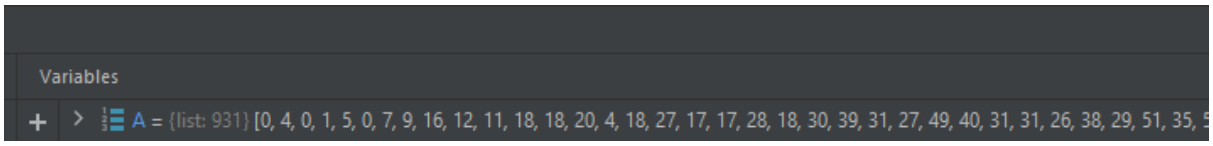
For the creation process two random numbers are generated, these are to be the sizes of the arrays. If both sizes are equal the second is regenerated until they are unequal. It then loops for the size given and populates an array with random numbers and prints the unsorted arrays.



Shell Sort

Comparing each element to every other element is very inefficient, the shell sort attempt to remedy this situation by taking elements from the extremities.

A gap is taken which is to be half of the array size. It will then compare the element at position 'i' with the element at position 'i+gap' and swaps them depending on their size, it loops until it reaches the element at the end of the list at which point the gap is divided in two. When the gap hits zero the shell sort is completed.



As one can notice this doesn't mean that the sort is completed, it just means that the elements are roughly at their final positions. Therefore, we must utilise a sorting algorithm to finish the sort, namely the insertion sort. When the insertion sort encounters an element, whose value is less than the one before it, it will move it backwards until it is in a position where it is greater than the value before it. This sort works well because the amount it has to travel backwards is reduced because the elements are roughly where they need to be.

Quick Sort

Quick sort works by instead of comparing every element we take an element which is roughly in the middle to be our pivot. In this case a method called "first middle last" was used in which the first, middle and last elements are taken and sorted in ascending order with the element in the middle of the sorted mini list being taken as the pivot. The pivot is then swapped with the last element and the sorting begins.

```
# choose pivot using first middle last
pvot = last - 1
# sort first last and middle
if B[last] < B[pvot]:
    hold = B[last]
    B[last] = B[pvot]
    B[pvot] = hold
elif B[pvot] > B[last]:
    hold = B[pvot]
    B[pvot] = B[last]
    B[last] = hold
if B[pvot] < B[first]:
    hold = B[pvot]
    B[pvot] = B[first]
    B[first] = hold

# swap pivot with last value
hold = B[pvot]
B[pvot] = B[last]
B[last] = hold
```

We make two pointers, one beginning at the start of the list and the other at the end of the list. The right pointer moves left until it finds a value smaller than the pivot, then the left pointer moves right till it encounters a value greater than the pivot at which point the elements are swapped. If at some point the pointers overtake each other, the right pointer swaps its element with the pivot and the pivot is returned.

We now divide our array in two an array before the pivot (including it) and one after the pivot. Quick sort is now performed on these two lists, this recursion will continue until we come to arrays with less than two elements.

Testing

Testing was performed with smaller arrays of about 20 elements. The debugger was used to follow the program to see if it met the specifications. At the end of testing the full array size was implemented and the output was checked to see if it was sorted.

Question 2

Code

```
i = 0
j = 0

C = []

# merge sort
while i < asize and j < bsize:
    if A[i] < B[j]:
        C.append(A[i])
        i += 1
    else:
        C.append(B[j])
        j += 1

# remaining elements are added
while i < asize:
    C.append(A[i])
    i += 1
while j < bsize:
    C.append(B[j])
    j += 1

print("Array of both Arrays merged in linear time: ", C)

print("Array of both Arrays merged in linear time: ", C)
```

Explanation

This function takes the two arrays from the previous list and sorts them into one array by using the merge sort.

Merge Sort

A pointer is placed at the beginning of each array, both elements are incremented each time appending the smaller of the elements to the list. At the end the remaining elements are added from the larger of the two arrays.

```
Array Before Shell Sort:  [113, 635, 705, 803, 931, 317, 648, 504, 492, 248, 426, 817, 372, 371, 820, 918
Array Before Quick Sort:  [868, 652, 596, 981, 413, 318, 537, 440, 32, 155, 1004, 929, 743, 126, 64, 189,
Array After Shell Sort:  [2, 3, 3, 6, 6, 8, 11, 13, 13, 14, 15, 17, 19, 20, 21, 23, 23, 28, 33, 34, 34, 3
Array After Quick Sort:  [2, 4, 4, 10, 12, 13, 13, 15, 15, 16, 16, 16, 17, 19, 20, 21, 22, 28, 29, 31, 32
Array of both Arrays merged in linear time:  [2, 2, 3, 3, 4, 4, 6, 6, 8, 10, 11, 12, 13, 13, 13, 13, 14,
```

Due to both arrays being already previously sorted only one passthrough is required and therefor it is completed in linear time O(n).

Testing

Testing was performed with smaller arrays of about 20 elements. The debugger was used to follow the program to see if it met the specifications. At the end of testing the full array size was implemented and the output was checked to see if it was sorted.

Question 3

Code

```
import random

aSize = random.randint(8, 16)

A = []
Extreme = []

for i in range(aSize):
    A.append(random.randint(0, 128))
# random array is generated

print("Initial array:", A)

n = len(A)
for i in enumerate(A):
    if 0 < i[0] < n - 1 and (A[i[0] - 1] < A[i[0]] > A[i[0] + 1] or A[i[0]
- 1] > A[i[0]] < A[i[0] + 1]):
        Extreme.append(A[i[0]])
# each element is checked to see if it meets the condition to be and
extreme point
# and if so it is appended to the extreme array

if len(Extreme) > 0:
    print("The extreme points are:")
    print(Extreme)
else:
    print("SORTED")
# Extreme points are printed
# if there aren't any SORTED is printed

# Yes, because the list will be linearly ascending and descending
# (note that one cannot dictate if the list is ascending or descending)
```

Explanation

A random array is generated like was done in Question 1. Each element of this randomly generated array is check against the provided condition (0 < i < n –1and either A[i − 1] < A[i] > A[i + 1] or A[i − 1] > A[i] < A[i + 1]), if this condition is met then the element is added to the 'Extreme' array. If the 'Extreme' array has no elements then "SORTED" is printed, otherwise the elements of the array are printed.

```
C:\Users\Gianmatsam\PycharmProjects\DSACoursework\venv
Initial array: [75, 86, 44, 42, 53, 128, 35, 8]
The extreme points are:
[86, 42, 128]
```

Figure 1: Extreme points are printed

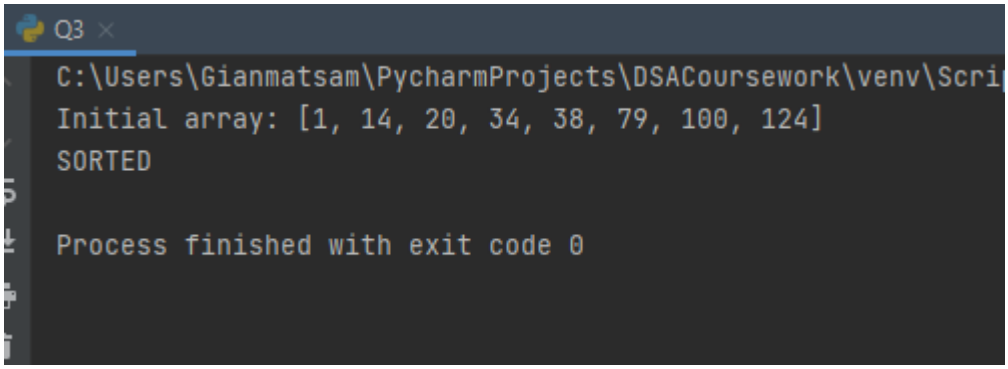


Figure 2:Sorted List

In response to the question "Do you agree that an array has no extreme points if and only if it is sorted?".

Yes, because the list will be linearly ascending and descending however it should be noted that the user cannot dictate if the array will be ascending or ascending.

Testing

The elements of the array were initially hard coded to make bug testing, after which the random array generator was added.

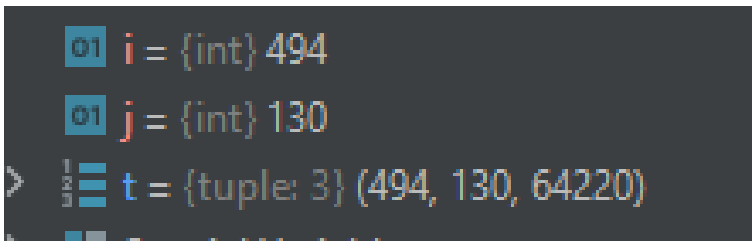
Question 4

Code

```
import random

asize = random.randint(64, 128)

A = []

for i in range(asize):
    A.append(random.randint(0, 1024))
# an array is made randomly similar to previous questions

hash = []
ans = []

print(A)

for i in A:
    for j in A:
        if i != j:
            t = (i, j, i*j)
            hash.append(t)
# each element is paired with every other element along with their product
(element 1, element 2 , product)

for i in hash:
    for j in hash:
        if i[2] == j[2]:
        # checks if two tuples have the same product
            if i[0] != i[1] and j[0] != j[1] and i[0] != j[1] and j[0] !=
i[1] and i[0] != j[0] and i[1] != j[1]:
            # checks that both tuples have unique elements so a!=b!=c!=d
                holdAB = (i[0], i[1])
                holdCD = (j[0], j[1])
                holdANS = (holdAB, holdCD)
                ans.append(holdANS)
                # the tuples are tupled and added to the answer list

print("Two pairs of integers: \n")
print(ans)
```

Explanation

A random array is generated. An array of tuples is made, each tuple consists of two elements and their product (element 1, element 2, product) every element is tupled with every other element and appended to the array.



It then loops through the array of tuples and checks if there are two tuples with matching products and have no elements alike. These matching tuples are tupled and added to the ‘ans’ array in the form ((element 1, element 2), (element 3, element 4)).

It does everything in big O(n^2). It also has repeated tuples with different position, for example ((8,161), (56,23)) and ((8,161), (23, 56)) are both answers.

```
C:\Users\Glanmatsam\PycharmProjects\DSACourseWork\venv\Scripts\python.exe C:/Users/Glanmatsam/PycharmProjects/
[553, 8, 56, 854, 293, 968, 505, 712, 952, 558, 212, 901, 161, 969, 1003, 730, 753, 234, 223, 526, 43, 984, 23
Two pairs of integers:

[((8, 161), (56, 23)), ((8, 161), (23, 56)), ((8, 294), (56, 42)), ((8, 294), (42, 56)), ((8, 189), (36, 42)),

Process finished with exit code 0
```

Testing

It was tested by doing trial and error with a pre-set array, this was then replaced with the randomly generated array and the output was repeatedly checked to make sure the correct output was being given.

Question 5

Code

```
RPN = []
Stack = []

sPOS = 0
hold = ""
# declaration of variables

print("Enter the reverse polish notation to calculate it\n")
temp = input("Note: put commas between the numbers/symbols example
3,10,5,+,*,\n")
for letters in temp:
    if letters == ',':
        RPN.append(hold)
        hold = ""
    else:
        hold += letters
print(RPN)
# tokenisation of input

if hold != "":
    RPN.append(hold)


for i in enumerate(RPN):
    if i[1] == '*':
        Stack[sPOS-2] = int(Stack[sPOS-2])*int(Stack[sPOS-1])
        Stack.pop(sPOS-1)
        sPOS -= 1
    elif i[1] == '/':
        Stack[sPOS-1] = int(Stack[sPOS-2])/int(Stack[sPOS-1])
        sPOS -= 1
    elif i[1] == '-':
        Stack[sPOS-1] = int(Stack[sPOS-2])-int(Stack[sPOS-1])
        sPOS -= 1
    elif i[1] == '+':
        Stack[sPOS-2] = int(Stack[sPOS-2])+int(Stack[sPOS-1])
        Stack.pop(sPOS - 1)
        sPOS -= 1
    else:
        Stack.append(i[1])
        sPOS += 1
    # arithmetic  operations
    print("Contents Currently of the Stack :\n")
    print(Stack)
print(ans)
```

Explanation

The program starts by asking the user for input, the input should be given as a line with a comma after each character (for example: 3,10,5, +, *,). This input is then tokenised ['3', '5', '10', '+', '*'] in preparation for the stack process.

The stack has two modes, push and pop, the program is going to read the tokenised list and if it encounters a number it is going to push its contents onto the stack and if it is an arithmetic symbol it is going to pop the two top elements of the stack.

Dry Run

Our input is ['3', '5', '10', '+', '*']

First character is ‘3’ so push it onto the stack:

Stack: [3]

Second is ‘5’ so do the same:

Stack: [3],[5]

Third is ‘10’ so do the same:

Stack: [3],[5],[10]

Fourth is ‘+’ so perform addition on the last two elements.

Stack: [3],[15]

Fifth is ‘*’ so perform multiplication on the last two elements.

Stack: [45]

```
Note: put commas between the numbers/symbols example 3,10,5,+,*,
3,10,5,+,*,
['3', '10', '5', '+', '*']
Contents Currently of the Stack :

['3']
Contents Currently of the Stack :

['3', '10']
Contents Currently of the Stack :

['3', '10', '5']
Contents Currently of the Stack :

['3', 15]
Contents Currently of the Stack :

[45]
```

Testing

Dry runs were first performed to see what the answer should come this was compared to the output of the algorithm for discrepencies.

Question 6

Code

```
def prime(num):
    if num == 0 or num == 1:
        return False
    elif num == 2:
        return True
    elif (num % 2) == 0:
        return False
    elif num == 3:
        return True
    # program checks to see if the number is 0, 1, 2 ,3 or even
    # and returns an appropriate result

    current = 3
    while current < num:
        if (num % current) == 0:
            return False
        else:
            current += 2
    return True


# checks if num is prime

def sieveOfEra(num):
    if num == 0 or num == 1:
        return False
    sieve = []
    start = 2

    while start <= num:
        sieve.append(start)
        start += 1
    # create an array with all the values until the provided number
    element = 0
    p = sieve[element]

    while True:
        iterate = p
        count = 1

        while iterate <= num:
            count += 1
            iterate = count * p
            # iterates through all the multiples
            if iterate in sieve:
                sieve.remove(iterate)
                # removes multiples

        element += 1
        if element >= len(sieve):
            return sieve
            # if the end of the elements is reached then the program stops

        p = sieve[element]
        # the sieve moves to the next element that was not marked to see if
the pother elements are multiples of it
```

```
# Sieve of Eratosthenes


num = int(input("Please enter a positive integer to check if it is
prime\n"))
while num < 0:
    num = int(input("Error integer must be positive please enter again\n"))
check = prime(num)

if check == True:
    print("The number is prime\n")
else:
    print("The number is not prime\n")

primeNums = sieveOfEra(num)

print("Prime numbers until that value \n")
print(primeNums)
```

Explanation

The user is asked to enter a positive integer and if these specifications are not met, he is asked to reinput the integer.

```
Please enter a positive integer to check if it is prime
-1
Error integer must be positive please enter again
-50
Error integer must be positive please enter again
7
The number is prime
```

Check if Prime

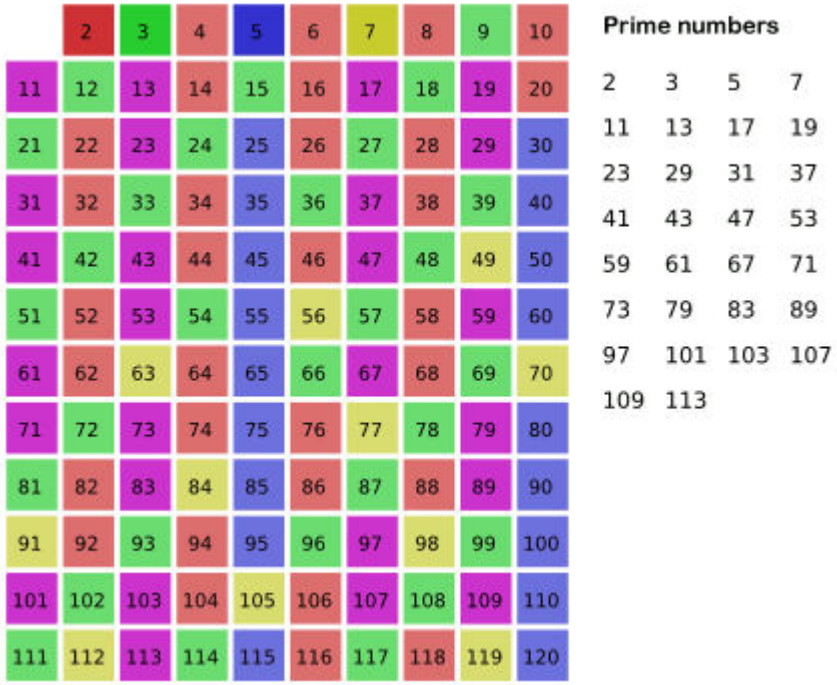
The function fist checks if the number is 0, 1, 2 ,3 or even and if it is an appropriate Boolean is returned for each case. Otherwise starting from 3 it will loop through all odd numbers until the number to see if the provided number is a multiple of one of them.

```
Please enter a integier to check if it is prime
9
The number is not prime
```

```
7
The number is prime
```

Sieve of Eratosthenes

The sieve works by firstly making an array and filling it with all the incremental values until the provided number. It then starts from the smallest ‘unmarked’ element and removes all of its multiples from the list, the program then moves to the next element and keeps on going till it runs out of unmarked elements.



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

2	3	5	7
11	13	17	19
23	29	31	37
41	43	47	53
59	61	67	71
73	79	83	89
97	101	103	107
109	113		

Figure 3: Note the unmarked elements are highlighted in purple

```
Please enter a positive integer to check if it is prime
121
The number is not prime

Prime numbers until that value

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113]

Process finished with exit code 0
```

Testing

Testing consisted of doing a dry run and checking the debugger to see if the correct steps were taken. This was followed by checking the output with a correct answer (Figure 3).

Question 7

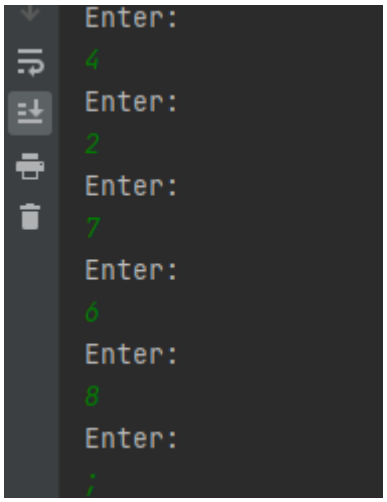
Code

```
class BST:
    def __init__(self, value):
        self.leftChild = None
        self.parent = value
        self.rightChild = None
        # definition of the BST type

    def addTree(self, value):
        if value <= self.parent:
            # checks where the new node should go
            if self.leftChild:
                # if there is already a node go even deeper
                return self.leftChild.addTree(value)
            else:
                # if there isn't a node set it to the value
                self.leftChild = BST(value)
                return
        else:
            # if the node isn't on the left it goes on the right
            if self.rightChild:
                # if there is already a node go even deeper
                return self.rightChild.addTree(value)
            else:
                # if there isn't a node set it to the value
                self.rightChild = BST(value)
                return

    def printTree(self):
        if self.leftChild:
            self.leftChild.printTree()
        print(self.parent)
        if self.rightChild:
            self.rightChild.printTree()
    # tree is printed in order


root = BST(0)
first = True
print("Enter the integers\n")
print("To exit entering input ;\n")
while True:
    temp = input("Enter:\n")
    if temp == ';':
        break
    else:
        temp = int(temp)
        if first == True:
            # if root isn't set it will set it
            first = False
            root = BST(temp)
        else:
            root.addTree(temp)

root.printTree()
```

Explanation

Input

The user Is to enter values sequentially and should enter a semicolon to stop entering. On receiving the first input the program sets the root, any additional inputs are appended to the tree by the AddTree function.



Definition

A binary search tree is a structure where every node can point to not more than 2 other nodes and every node has to be connected to the structure. In this regard the binary search tree is implemented very similarly to a linked list except instead of pointing to one node it has the ability to point to two.

Adding to Tree

The program first checks if the value is lass or greater than the parent node, values smaller or equal to the parent go on the left of the tree and greater values go on the right. If there is already a value on the left or right node then the program goes deeper on the appropriate node, where the left or right node will be the parent in this deeper section. If there isn't a node then the node is set to the inputted value.

Printing the tree

The tree prints inorder, it first goes as deep as it can on the left subtree, then it prints the root and then the right subtree. It works by firstly checking if there is a left node, if there is it goes into it, it then prints the parent value and then checks if there is a right node and if so, goes into it.

Testing

The program was followed using the debugger and the output was compared to a correct answer.

Question 8

Code

```
while True:
    n = float(input("please enter a positive integer to find its square
root\n"))
    if n > 0:
        break
    print("Error: negative integer entered or zero")
    # asks the user for a positive integer and if this isn't met he is
asked to reenter

x = n/2
# starts with n being half of x
# this is because square roots (except 1 and 2) are all less than half
their squares
i = 0
while True:
    x = (1/2)*(x+(n/x))
    # Newton-Raphson formula
    if x*x == n:
        # if the current approximation square equals the original value
then the square root has been achieved
        print('The square root is : ', x)
        break
    if i == 20:
        print('iteration limit reached (20 iterations)')
        print('The approximated square root is : ', x)
        break
    i += 1
# every iteration gives a more accurate approximation
```

Explanation

The user is asked to input a positive integer and as asked to re-enter if the value is not a positive integer. The program starts by taking the first approximation to be half of the original value, this is because all square roots of numbers greater than 4 are less than half the original number. The program iterates for a maximum of twenty times, each time the Newton-Raphson formula ((1/2) *(x+(n/x))) is performed on the current approximation. If the current approximation squared is equal to the original value, then the square root is printed. If the maximum number of iterations (20) is reached then the current approximation is printed.

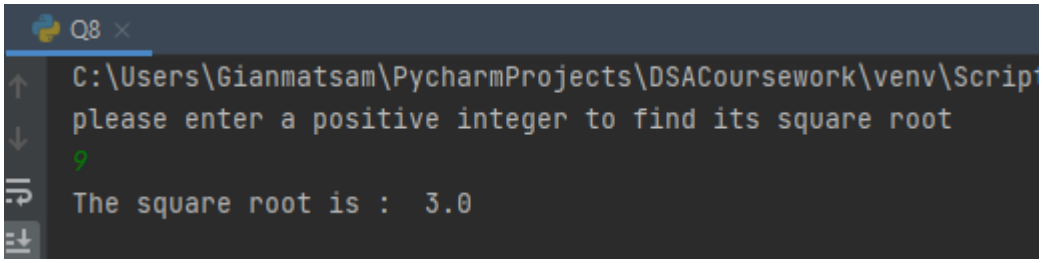


Figure 4: Square root of 9 was correctly found

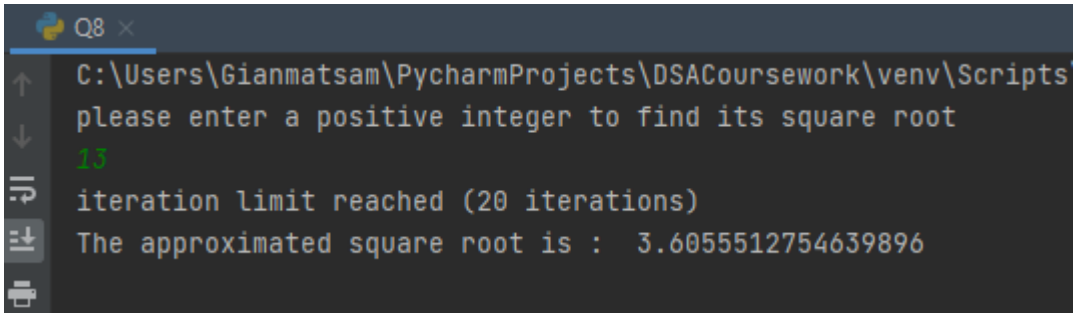


Figure 5: an accurate approximation of 13s square root

Testing

The programs output was compared to the square root of the number to see if an accurate square root was being outputted.

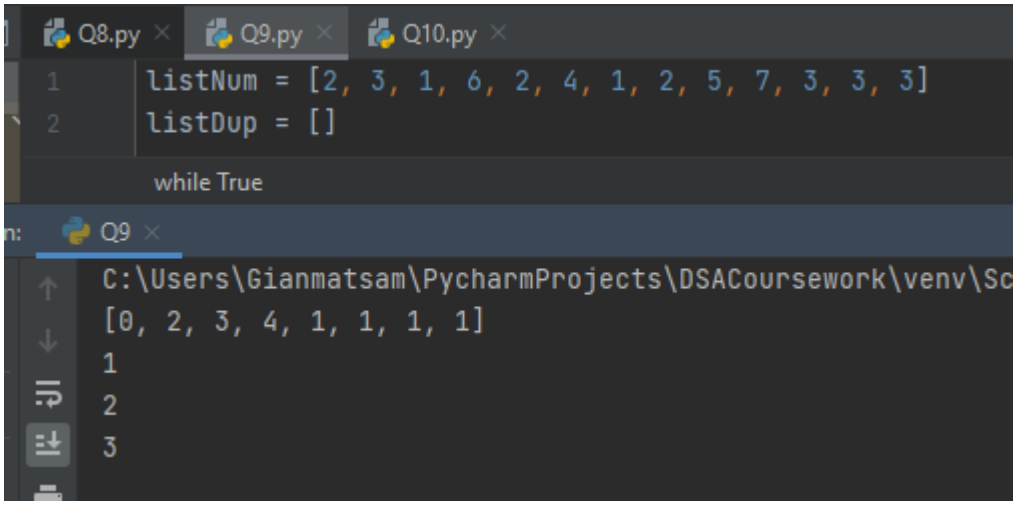
Question 9

Code

```
listNum = [2, 3, 1, 6, 2, 4, 1, 2, 5, 7, 3, 3, 3]
listDup = []
# pre-set array made on purpose with duplicate values

largest = -999
smallest = 999

for num in listNum:
    if num > largest:
        largest = num
    if num < smallest:
        smallest = num
    # the largest and smallest values in the array are found

if smallest > 0:
    i = 0
    smallest = 0
else:
    i = smallest
# sets i and smallest to 0

while True:
    listDup.append(0)
    if i >= largest:
        break
    i += 1
    # blank array of the size of the largest value is made

for num in enumerate(listNum):
    listDup.insert(num[1] - smallest, listDup[num[1] - smallest] + 1)
    listDup.pop((num[1] - smallest) + 1)
    # uses the value in the pre-set array as the index to be incremented in
the new duplicate array

print(listDup)

for num in enumerate(listDup):
    if listDup[num[0]] > 1:
        print(num[0] + smallest)
# if an element has a value greater than 1 that means it has appeared more
than 1 time and
# so should be printed
```

Explanation

The program does one pass through the entire list and finds the smallest and largest value in the pre-set array, this is used to make a blank array with the number of spaces being that of the largest value. The program makes another pass through, it takes the value in the pre-set array as an index in the new blank array and increments it. Finally, it loops through the new array and prints the index of any element that has a value greater than one (therefor it has appeared more than one time). Like this the program has a big O notation of O(n).



From the array above ([0, 2, 3, 4, 1, 1, 1, 1]) we can see that 0 has appeared 0 times, 1 has appeared 2 times, 2 has appeared 3 times, 3 appears 4 times and 4,5,6 and 7 appear all 1 time. Since 1, 2 and 3 appear more than 1 time they are printed.

Testing

The output was checked to see if the correct index was being incremented and if the correct duplicate values were being printed.

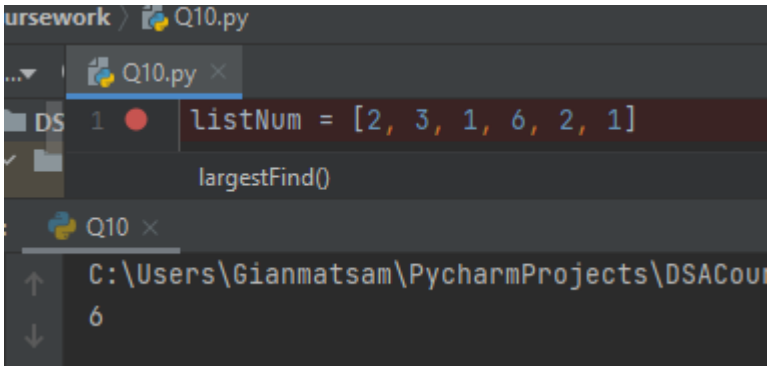
Question 10

Code

```
listNum = [2, 3, 1, 6, 2, 1]

largest = -999
pointer = 0

def largestFind(large, listNum, pointer):
    if large < listNum[pointer]:
        large = listNum[pointer]
    # if the current number is larger than the previous larger than the
previous than it is set as the new largest

    if pointer == (len(listNum) - 1):
        # if the end of the array is reached then return the current
largest value
        return large
    else:
        pointer += 1
        return largestFind(large, listNum, pointer)

hold = largestFind(largest, listNum, pointer)
print(hold)
# print the largest number
```

Explanation

The program calls the function passing the pre-set array. Starting from the first element it checks if the value of the current value is greater than the largest found value, if it is it replaces the previous value as the largest. If the end of the list is reached then the largest value is returned, otherwise the index is incremented and the function is called pointing to the next index. Program terminates by outputting the returned value (which should be the largest).



Testing

Pre-Set array was passed and the output was checked to see if the largest value was being returned.

Question 11

Code

```
import math

cos = 0
while True:
    equation = input("Please enter the equation in the form 'sin63' or
'cos4'\n")
    if equation[0:3] == "sin":
        print("Sin value entered\n")
        break
    elif equation[0:3] == "cos":
        print("Cos value entered\n")
        cos = 1
        break
    # checks to see if sine or cosine was entered
    else:
        print("Error neither sine or cosine entered please re enter in the
correct format\n")
    # if the user entered neither sine or cosine, the equation is re-
entered

x = int(equation[3:]) #x value
counter = 0
ans = 0
n = int(input("For how many times do you wish the program to loop through
(n times) \n"))

while counter != n:
    if cos == 0:
        ans += (((-
1)**counter)/math.factorial(2*counter+1))*(x**(2*counter+1))
        # this formula is utilised if the user entered sin
    else:
        ans += (((-1) ** counter) / math.factorial(2 * counter)) * (x ** (2
* counter))
        # this formula is utilised if the user entered cos
    counter += 1
# the program loops for n times

print(ans)
```

Explanation

The program asks the user to input an equation with sin or cos (ex sin2 or cos45) if the user inputs neither then the program asks the user to re-enter the equation. The user is then asked how many terms of the Maclaurin he wishes, the more terms will result in a more accurate approximation. Depending if the user entered sin or cos dictates which one of two formulas is used, each time the formula is used this is a new term being added to the answer.

```
Please enter the equation in the form 'sin63' or 'cos4'
cos28
Cos value entered

For how many times do you wish the program to loop through (n times)
100
-0.9626050889492841

Process finished with exit code 0
```

Figure 6: approximation of cos28 at 100 terms

Figure 7: approximation according to google calculator (quite similar)

Testing

The output was repeatedly compared to correct values to see if accurate approximation were being given.

Question 12

Code

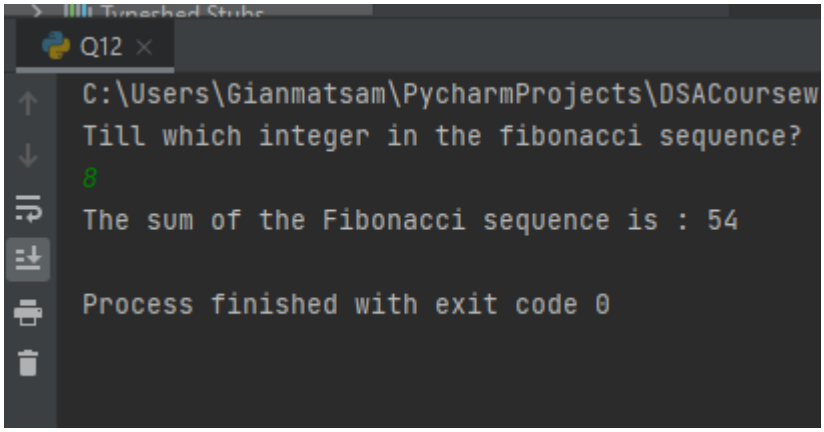
```
while True:
    n = int(input("Till which integer in the fibonacci sequence? \n"))
    if n > 2:
        break
    print("Error integer must be greater than 2 in a fibonacci sequence")
    # the user is asked for the number of terms he wishes and is asked to
re-enter if the integer is less than 3

def fibonacci(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
    # function that returns the fibonacci number at n terms

count = 0
sum = 0
limit = n
while True:
    if count == n:
        break
    sum += fibonacci(limit)
    limit -= 1
    count += 1
# loops adding the fibonacci number each time to the sum

print("The sum of the Fibonacci sequence is :", sum)
# sum is printed and the program terminates
```

Explanation

The user is asked till what terms of the Fibonacci sequence he wishes to be summed, if he enters less than 3, he is asked again to input the term limit. The program has a utilises a function Fibonacci(n), n being the term wished to be found, which returns the Fibonacci number at term ‘n’. The program loops, calling the function to return all the terms from 1 to ‘n’ to be summed up.

Example:

8 terms = 21+13+8+5+3+2+1+1 = 54



Testing

Output was compared to the actual summation of the Fibonacci sequence for discrepancies.