Data Structures and Algorithms I

Coursework 2021-2022

Statement of Completion:

All questions were attempted and work well.

Signature:

Plagiarism Declaration Form:

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).

Matthias Bartolo
Student Name

MBar
Signature

Student Name

Signature

Student Name

Signature

Student Name

Signature

ICT 1018
Course Code

Data Structures and Algorithms I Coursework
Title of work submitted

14/05/2022
Date

Question 1:

Code:

```
import random;
#Declaring arrays
arrayA=[];
arrayB=[];

#Populating both arrays with random values
for i in range(0,258):
    arrayA.append(random.randint(0,1024));

for i in range(0,257):
    arrayB.append(random.randint(0,1024));
```

```
def ShellSort(mylist,n):
    k=int(n/2);#Where k is gap
    #Looping until gap>0
    while k>0:
        #Looping from gap to end
        for i in range(k,n):
            nextPos=i;#Position of element to insert
            nextVal=mylist[i];#Value of element
            while nextPos>=k and mylist[nextPos-k]>nextVal:
                #Shifting elements at nextPos-gap to nextPos
                temp=mylist[nextPos];
                mylist[nextPos]=mylist[nextPos-k];
                mylist[nextPos-k]=temp;
                #Decrementing nextPos by gap
                nextPos=nextPos-k;
        if k==2:#if gap= 2 then set gap to 1
            k=1;
        else:#Dividing gap by 2
            k=int(k/2);
```

```
def QuickSort(mylist,first,last):
    #Recursive Case where first pointer is smaller than last pointer
    if first<last:
        #Finding pivot element such that elements are smaller on the left and greater on the right
        pivotPos=partition(mylist,first,last);
        #Sorting left sublist
        QuickSort(mylist,first,pivotPos-1);
        #Sorting right sublist
        QuickSort(mylist,pivotPos+1,last);
```

```
def partition(mylist,first,last):
    #Finding the size of array
    n=last-first;
```

```
    #Finding the best pivot
    tempa=[];

    if n>10:
        #For larger lists if statement is used
        #Populating tempa with random integers
        for i in range(0,int(n/5)):
            m=random.randint(first,last);
            tempa.append(mylist[m]);
        #Sorting array
        ShellSort(tempa,len(tempa));
        #Taking median as the pivot
        pivot=tempa[int(len(tempa)/2)];
        #Finding pivot position in original array
        pivotpos=mylist.index(pivot);

        #Setting pivot value last position by swapping
        mylist[pivotpos]=mylist[last];
        mylist[last]=pivot;
    else:
        #For smaller lists else statement is used
        pivot=mylist[last];

    #Pointer pointing to the position of the last element which is smaller than the pivot
    psmall=first-1;

    #Traversing through list
    for pelement in range(first,last):
        #If element is smaller than pivot
        if mylist[pelement]<pivot:
            #Incrementing pointer of the smaller element
            psmall=psmall+1;
            #Swapping the smaller element with the current element
            tempv=mylist[pelement];
            mylist[pelement]=mylist[psmall];
            mylist[psmall]=tempv;

    #Putting the pivot in it's right position by swapping
    tempv=mylist[last];
    mylist[last]=mylist[psmall+1];
    mylist[psmall+1]=tempv;
    #Returning the pivot position
    return psmall+1;
```

```
print(arrayA);
ShellSort(arrayA,len(arrayA));
print();
print(arrayA);
```

```
print(arrayB);
QuickSort(arrayB,0,len(arrayB)-1);
print();
print(arrayB);
```

Note that:

The Quick Sort was inspired from the notes and the following link: link

Testing:

Testing the Shell Sort with a random populated array of 258 elements:

```
print(arrayA);
ShellSort(arrayA,len(arrayA));
print();
print(arrayA);
```

```
[65, 369, 549, 544, 671, 145, 429, 310, 37, 530, 282, 54, 803, 530, 264, 713, 420, 620, 124, 193, 597, 992, 285, 460, 613, 96,
271, 239, 605, 477, 340, 725, 670, 682, 517, 699, 334, 250, 151, 556, 460, 707, 910, 379, 956, 106, 661, 811, 925, 183, 766, 7,
996, 113, 70, 715, 706, 258, 709, 695, 433, 643, 22, 697, 936, 266, 678, 763, 587, 753, 624, 623, 924, 571, 254, 13, 166, 335,
829, 421, 548, 783, 833, 344, 83, 167, 767, 951, 972, 87, 289, 673, 574, 1003, 782, 343, 325, 885, 1020, 287, 936, 144, 1011, 9
36, 909, 431, 843, 241, 563, 646, 612, 148, 735, 552, 983, 710, 402, 347, 276, 270, 159, 786, 579, 80, 627, 898, 416, 258, 151,
302, 318, 597, 632, 859, 890, 406, 691, 742, 788, 209, 67, 436, 961, 430, 989, 888, 434, 869, 1015, 105, 968, 880, 50, 490, 89
3, 172, 665, 220, 875, 237, 617, 450, 878, 415, 2, 650, 505, 19, 364, 907, 304, 325, 938, 326, 848, 893, 724, 292, 688, 417, 73
9, 542, 395, 796, 482, 845, 152, 529, 682, 663, 993, 437, 911, 165, 847, 38, 253, 446, 129, 222, 242, 2, 202, 511, 124, 755, 35
6, 666, 320, 624, 958, 882, 329, 660, 609, 205, 649, 528, 703, 890, 763, 65, 567, 624, 65, 675, 501, 935, 691, 31, 420, 148, 15
2, 568, 325, 848, 747, 957, 699, 843, 801, 731, 569, 336, 198, 236, 524, 976, 677, 509, 1006, 47, 995, 970, 385, 852, 643, 353]

[2, 2, 7, 13, 19, 22, 31, 37, 38, 47, 50, 54, 65, 65, 65, 67, 70, 80, 83, 87, 96, 105, 106, 113, 124, 124, 129, 144, 145, 148,
148, 151, 151, 152, 152, 159, 165, 166, 167, 172, 183, 193, 198, 202, 205, 209, 220, 222, 236, 237, 239, 241, 242, 250, 253, 25
4, 258, 258, 264, 266, 270, 271, 276, 282, 285, 287, 289, 292, 302, 304, 310, 318, 320, 325, 325, 325, 326, 329, 334, 335, 336,
340, 343, 344, 347, 353, 356, 364, 369, 379, 385, 395, 402, 406, 415, 416, 417, 420, 420, 421, 429, 430, 431, 433, 434, 436, 43
7, 446, 450, 460, 460, 477, 482, 490, 501, 505, 509, 511, 517, 524, 528, 529, 530, 530, 542, 544, 548, 549, 552, 556, 563, 567,
568, 569, 571, 574, 579, 587, 597, 597, 605, 609, 612, 613, 617, 620, 623, 624, 624, 624, 627, 632, 643, 643, 646, 649, 650, 66
0, 661, 663, 665, 666, 670, 671, 673, 675, 677, 678, 682, 682, 688, 691, 691, 695, 697, 699, 699, 703, 706, 707, 709, 710, 713,
715, 724, 725, 731, 735, 739, 742, 747, 753, 755, 763, 763, 766, 767, 782, 783, 786, 788, 796, 801, 803, 811, 829, 833, 843, 84
3, 845, 847, 848, 848, 852, 859, 869, 875, 878, 880, 882, 885, 888, 890, 890, 893, 893, 898, 907, 909, 910, 911, 924, 925, 935,
936, 936, 936, 938, 951, 956, 957, 958, 961, 968, 970, 972, 976, 983, 989, 992, 993, 995, 996, 1003, 1006, 1011, 1015, 1020]
```

Testing the Quick Sort with a random populated array of 257 elements:

```
print(arrayB);
QuickSort(arrayB,0,len(arrayB)-1);
print();
print(arrayB);
```

```
[91, 478, 4, 1014, 122, 791, 229, 684, 426, 836, 0, 971, 114, 135, 872, 322, 256, 535, 829, 127, 673, 376, 335, 541, 227, 219,
281, 418, 337, 105, 574, 676, 223, 987, 93, 572, 132, 843, 706, 189, 8, 136, 131, 124, 257, 571, 394, 106, 708, 65, 157, 230, 8
50, 84, 753, 67, 943, 804, 401, 506, 310, 824, 288, 480, 870, 114, 563, 531, 735, 848, 94, 332, 373, 776, 273, 231, 544, 16, 86
3, 973, 353, 118, 542, 507, 182, 679, 109, 38, 699, 463, 746, 360, 143, 360, 727, 953, 389, 767, 689, 551, 784, 362, 905, 773,
584, 443, 556, 448, 277, 840, 277, 619, 123, 585, 676, 477, 399, 319, 461, 358, 967, 952, 786, 481, 123, 259, 142, 477, 159, 80
5, 817, 446, 62, 119, 807, 836, 58, 267, 127, 466, 1004, 834, 701, 64, 230, 110, 885, 803, 590, 176, 844, 743, 538, 409, 968, 8
21, 933, 920, 181, 470, 160, 655, 817, 136, 475, 673, 304, 332, 551, 800, 760, 631, 304, 36, 138, 312, 566, 313, 51, 563, 30,
8, 980, 855, 159, 585, 681, 736, 610, 49, 977, 25, 137, 786, 330, 732, 217, 573, 590, 673, 449, 90, 719, 376, 451, 621, 205, 91
9, 84, 922, 784, 342, 2, 444, 12, 39, 694, 312, 481, 305, 1018, 726, 553, 130, 416, 752, 767, 620, 677, 557, 465, 807, 873, 59
3, 914, 147, 690, 337, 308, 930, 222, 810, 138, 672, 235, 873, 685, 99, 468, 551, 978, 605, 840, 180, 922, 129, 972]

[0, 2, 4, 8, 8, 12, 16, 25, 30, 36, 38, 39, 49, 51, 58, 62, 64, 65, 67, 84, 84, 90, 91, 93, 94, 99, 105, 106, 109, 110, 114, 11
4, 118, 119, 122, 123, 123, 124, 127, 127, 129, 130, 131, 132, 135, 136, 136, 137, 138, 138, 142, 143, 147, 157, 159, 159, 160,
176, 180, 181, 182, 189, 205, 217, 219, 222, 223, 227, 229, 230, 230, 231, 235, 256, 257, 259, 267, 273, 277, 277, 281, 288, 30
4, 304, 305, 308, 310, 312, 312, 313, 319, 322, 330, 332, 332, 335, 337, 337, 342, 353, 358, 360, 360, 362, 373, 376, 376, 389,
394, 399, 401, 409, 416, 418, 426, 443, 444, 446, 448, 449, 451, 461, 463, 465, 466, 468, 470, 475, 477, 477, 478, 480, 481, 48
1, 506, 507, 531, 535, 538, 541, 542, 544, 551, 551, 551, 553, 556, 557, 563, 563, 566, 571, 572, 573, 574, 584, 585, 585, 590,
590, 593, 605, 610, 619, 620, 621, 631, 655, 672, 673, 673, 673, 676, 676, 677, 679, 681, 684, 685, 689, 690, 694, 699, 701, 70
6, 708, 719, 726, 727, 732, 735, 736, 743, 746, 752, 753, 760, 767, 767, 773, 776, 784, 784, 786, 786, 791, 800, 803, 804, 805,
807, 807, 810, 817, 817, 821, 824, 829, 834, 836, 836, 840, 840, 843, 844, 848, 850, 855, 863, 870, 872, 873, 873, 885, 905, 91
4, 919, 920, 922, 922, 930, 933, 943, 952, 953, 967, 968, 971, 972, 973, 977, 978, 980, 987, 1004, 1014, 1018]
```

Question 2:

Code:

```
def merge(mylist1,mylist2):
    #Calculating size of new list
    size=len(mylist1)+len(mylist2);
    #Setting indexes of both lists to starting position i.e pos 0
    count1=0;
    count2=0;
    #Declaring new list
    mylist3=[];
    #Looping through all elements until count1 is smaller than length of mylist1
    #and count2 is smaller than length of mylist2
    while count1<len(mylist1) and count2<len(mylist2):
            #If element in mylist1 is larger than element in mylist2
            if mylist1[count1]>mylist2[count2]:
                #Add element in mylist2 to new list
                mylist3.append(mylist2[count2]);
                #Incrementing count2
                count2=count2+1;
            else:
                #Add element in mylist1 to new list
                mylist3.append(mylist1[count1]);
                #Incrementing count1
                count1=count1+1;

    #Placing the remaining elements to mylist3:
    #Note that only one of the following loops will be executed depending which
    #counter has not reached the end of the list respectively.

    #Adding the remaining elements of mylist1 to mylist3
    #if the count1 is smaller than length of mylist1
    while count1<len(mylist1):
            mylist3.append(mylist1[count1]);
            count1=count1+1;

    #Adding the remaining elements of mylist2 to mylist3
    #if the count2 is smaller than length of mylist2
    while count2<len(mylist2):
            mylist3.append(mylist2[count2]);
            count2=count2+1;

    #Returning new list
    return mylist3;
```

```
arrayC=merge(arrayA,arrayB);
print(arrayC)
```

Testing:

Testing Question 2, with arrayA and arrayB from Question1:

```
arrayC=merge(arrayA,arrayB);
print(arrayC)
```

```
[0, 2, 2, 2, 4, 7, 8, 8, 12, 13, 16, 19, 22, 25, 30, 31, 36, 37, 38, 38, 39, 47, 49, 50, 51, 54, 58, 62, 64, 65, 65, 65, 65, 6
7, 67, 70, 80, 83, 84, 84, 87, 90, 91, 93, 94, 96, 99, 105, 105, 106, 106, 109, 110, 113, 114, 114, 118, 119, 122, 123, 123, 12
4, 124, 124, 127, 127, 129, 129, 130, 131, 132, 135, 136, 136, 137, 138, 138, 142, 143, 144, 145, 147, 148, 148, 151, 151, 152,
152, 157, 159, 159, 159, 160, 165, 166, 167, 172, 176, 180, 181, 182, 183, 189, 193, 198, 202, 205, 205, 209, 217, 219, 220, 22
2, 222, 223, 227, 229, 230, 230, 231, 235, 236, 237, 239, 241, 242, 250, 253, 254, 256, 257, 258, 258, 259, 264, 266, 267, 270,
271, 273, 276, 277, 277, 281, 282, 285, 287, 288, 289, 292, 302, 304, 304, 304, 305, 308, 310, 310, 312, 312, 313, 318, 319, 32
0, 322, 325, 325, 325, 326, 329, 330, 332, 332, 334, 335, 335, 336, 337, 337, 340, 342, 343, 344, 347, 353, 353, 356, 358, 360,
360, 362, 364, 369, 373, 376, 376, 379, 385, 389, 394, 395, 399, 401, 402, 406, 409, 415, 416, 416, 417, 418, 420, 420, 421, 42
6, 429, 430, 431, 433, 434, 436, 437, 443, 444, 446, 446, 448, 449, 450, 451, 460, 460, 461, 463, 465, 466, 468, 470, 475, 477,
477, 477, 478, 480, 481, 481, 482, 490, 501, 505, 506, 507, 509, 511, 517, 524, 528, 529, 530, 530, 531, 535, 538, 541, 542, 54
2, 544, 544, 548, 549, 551, 551, 551, 552, 553, 556, 556, 557, 563, 563, 563, 566, 567, 568, 569, 571, 571, 572, 573, 574, 574,
579, 584, 585, 585, 587, 590, 590, 593, 597, 597, 605, 605, 609, 610, 612, 613, 617, 619, 620, 620, 621, 623, 624, 624, 624, 62
7, 631, 632, 643, 643, 646, 649, 650, 655, 660, 661, 663, 665, 666, 670, 671, 672, 673, 673, 673, 673, 675, 676, 676, 677, 677,
678, 679, 681, 682, 682, 684, 685, 688, 689, 690, 691, 691, 694, 695, 697, 699, 699, 699, 701, 703, 706, 706, 707, 708, 709, 71
0, 713, 715, 719, 724, 725, 726, 727, 731, 732, 735, 735, 736, 739, 742, 743, 746, 747, 752, 753, 753, 755, 760, 763, 763, 766,
767, 767, 767, 773, 776, 782, 783, 784, 784, 786, 786, 786, 788, 791, 796, 800, 801, 803, 803, 804, 805, 807, 807, 810, 811, 81
7, 817, 821, 824, 829, 829, 833, 834, 836, 836, 840, 840, 843, 843, 843, 844, 845, 847, 848, 848, 848, 850, 852, 855, 859, 863,
869, 870, 872, 873, 873, 875, 878, 880, 882, 885, 885, 888, 890, 890, 893, 893, 898, 905, 907, 909, 910, 911, 914, 919, 920, 92
2, 922, 924, 925, 930, 933, 935, 936, 936, 936, 938, 943, 951, 952, 953, 956, 957, 958, 961, 967, 968, 968, 970, 971, 972, 972,
973, 976, 977, 978, 980, 983, 987, 989, 992, 993, 995, 996, 1003, 1004, 1006, 1011, 1014, 1015, 1018, 1020]
```

Question 3:

Code:

```
def extreme(array):
    check =0;#Boolean variable to check if array is sorted
    #Traversing through array
    for i in range(1,len(array)-1):
        #Checking extreme points and setting boolean variable to 1 to indicate not sorted
        if(array[i-1]<array[i])and(array[i]>array[i+1]):
            print(array[i]);
            check=1;
        if(array[i-1]>array[i])and(array[i]<array[i+1]):
            print(array[i]);
            check=1;

    #If condition to check if list is sorted based on boolean variable
    if check==0:
        print("SORTED")
```

```
#Figure i)
mylist= [1,2,3,4,5,6,7];
extreme(mylist);
```

```
#Figure ii)
mylist= [7,6,5,4,3,2,1];
extreme(mylist);
```

```
#Figure iii)
mylist= [5,7,7,7,5];
extreme(mylist);
```

```
#Figure iv)
mylist= [0,5,3,6,8,7,15,9];
extreme(mylist);
```

Testing & Final answer:

Do you agree that an array has no extreme points if and only if it is sorted? Explain your answer.

No, there are certain exceptions to this rule.
When an array is sorted the element prior is smaller than the current element and the subsequent element is larger than the current element, as can be seen in Figure i).

This also applies if the prior element is larger than the current element and the subsequent element is smaller than the current element, as can be seen in Figure ii).

The only exception to this rule, pertains to the data at the bounds i.e., first element and last element, as can be seen in Figure iii).

Figure iv) prints the extreme points in the array

```
#Figure i)
mylist= [1,2,3,4,5,6,7];
extreme(mylist);
```

```
SORTED
```

```
#Figure ii)
mylist= [7,6,5,4,3,2,1];
extreme(mylist);
```

```
SORTED
```

```
#Figure iii)
mylist= [5,7,7,7,5];
extreme(mylist);
```

```
SORTED
```

```
#Figure iv)
mylist= [0,5,3,6,8,7,15,9];
extreme(mylist);
```

```
5
3
8
7
15
```

Question 4:

Code:

```
import random;
#n is size of list
n=50;
#Declaring empty list
myList=[];

#Populating list with random values
for i in range(0,n):
    myList.append(random.randint(1,1024));

#Declaring empty list to hold unique values
uniquelist=[];
#Looping through list and appending unique values to uniquelist
for i in range(0,len(myList)):
    if myList[i] not in uniquelist:
        uniquelist.append(myList[i]);

#Setting myList to uniquelist and changing the size
myList=uniquelist;
n=len(myList);
print(myList);

#Creating relevant arrays to be used
pair1=[];
pair2=[];
product=[];

#Finding all product combinations for each pair
#and storing them respectively
#pair1 to hold the first number
#pair2 to hold the second number
#and product array to hold the product of both
for i in range(0,n):
    for j in range(i+1,n):
        if myList[i]!=myList[j]:
            pair1.append(myList[i]);
            pair2.append(myList[j]);
            product.append(myList[i]*myList[j])


#Matching the products of the first pair's with the second pair's
#If they match and the pairs are disctint, the 2 pairs are printed
for i in range(0,len(pair1)):
    for j in range(i+1,len(pair1)):
        if product[i]==product[j] and i!=j:
            if pair1[i]!=pair1[j] and pair2[i]!=pair2[j] and pair2[i]!=pair1[j]:
```

```
            print("(("+str(pair1[i])+","+str(pair2[i])+"),("+str(pair1[j])+","+str(pair2[j])+"))
="+str(product[i]))
```

Testing:

Testing Question 4 with a random populated array of 50 elements, the following pairs were outputted:

```
[794, 507, 890, 414, 958, 38, 817, 917, 905, 480, 428, 573, 797, 262, 108, 463, 787, 357, 923, 737, 523, 137, 557, 86, 384, 36
2, 560, 220, 957, 853, 297, 124, 864, 964, 473, 744, 727, 574, 389, 825, 224, 611, 110, 813, 273, 10, 333, 848, 992, 588]
((905,224),(362,560)) =202720
((108,992),(124,864)) =107136
```

```
[2, 782, 960, 290, 784, 923, 639, 977, 621, 763, 525, 670, 166, 117, 77, 501, 327, 195, 182, 936, 347, 529, 882, 146, 590, 541,
303, 350, 953, 818, 975, 1011, 345, 778, 556, 85, 68, 340, 417, 705, 490, 831, 912, 520, 1006, 798, 247, 319, 109]
((621,490),(882,345)) =304290
((621,520),(936,345)) =322920
((525,182),(195,490)) =95550
((195,340),(975,68)) =66300
((936,490),(882,520)) =458640
```

Question 5:

Code:

```
#word in RPN which is to be evaluated
word=input("Enter Arithmetic expression: ")+" "
#Declaration of stack and token
stack=[];
token="";

#Looping through all characters in word
for i in range(0,len(word)):
        #' ' is used as a Delimiter
        if word[i]==' ':
            #Executing respective if statements based on token contents
            if token=='x'or token=='*':
                temp1=int(stack.pop());
                temp2=int(stack.pop());
                stack.append(temp2*temp1);
                print(stack);
            elif token=='+':
                temp1=int(stack.pop());
                temp2=int(stack.pop());
                stack.append(temp2+temp1);
                print(stack);
            elif token=='/':
                temp1=int(stack.pop());
                temp2=int(stack.pop());
                stack.append(temp2/temp1);
                print(stack);
            elif token=='-':
                temp1=int(stack.pop());
                temp2=int(stack.pop());
                stack.append(temp2-temp1);
                print(stack);
            else:
                #Since the token is not an operator it will resort to add the operand
                #to the stack
                stack.append(token);
                print(stack);
            #Resetting token
            token="";
        else:
            #Adding current character to the token until a ' ' is found
            token=token+word[i];
```

Testing:

Testing Question 5 with the input: "4 2 3 5 1 - + * + ", and showing contents of stack at each step:

```
Enter Arithmetic expression: 4 2 3 5 1 - + * +
['4']
['4', '2']
['4', '2', '3']
['4', '2', '3', '5']
['4', '2', '3', '5', '1']
['4', '2', '3', 4]
['4', '2', 7]
['4', 14]
[18]
```

Testing Question 5 with the input: "1 2 + 3 4 + x ", and showing contents of stack at each step:

```
Enter Arithmetic expression: 1 2 + 3 4 + x
['1']
['1', '2']
[3]
[3, '3']
[3, '3', '4']
[3, 7]
[21]
```

Question 6:

Code Part 1) Checks if number is Prime:

```
#Method that checks if number is prime
def Prime(number):
    #1 is not a prime number, thus, false is returned
    if number<=1:
        return False;
    #Dividing the number by all the other numbers which are smaller
    for i in range(2,number-1):
        #If remainder is 0 then number is not prime thus, false is returned
        if((number%i)==0):
            return False;
    #Number is prime thus, true is returned
    return True;


#Testing:
number = int(input("Input number\n"));
check=Prime(number);

#Relevant print statements based on boolean function return
if(check==True):
    print("Number is Prime number");
else:
    print("Number is not a Prime number");
```

Testing Part 1) Checks if number is Prime:

- Testing whether 7 is a prime number:

```
Input number
7
Number is Prime number
```

- Testing whether 30 is a prime number:

```
Input number
30
Number is not a Prime number
```

- Testing whether 1 is a prime number:

```
Input number
1
Number is not a Prime number
```

- Testing whether -5 is a prime number:

```
Input number
-5
Number is not a Prime number
```

Code Part 2) Sieve of Eratosthenes:

```
import math;

#n is the maximum value to check
n=102;
#Error Checking
if n>0:
        #Creating a list with odd numbers and including number 2
        myList=list(range(3,n,2));
        myList.insert(0,2);

        #i is being used as a divisor. Looping from 3 since, all the even values were removed
        #till math.sqrt(n)+1. It is looping till math.sqrt(n)+1 since,
        #all the factors preceding the square root would be removed
        for i in range(3,int(math.sqrt(n)+1)):
        #j is being used as an index to myList
        for j in range(i,len(myList)):
        #if myList[j] has a remainder then it is set to 0 to indicate absence
         if myList[j]%i==0:
                myList[j]=0;


        #Removing 0's from list
        myList=[i for i in myList if i!=0] ;
        print(myList);
elif n>0 and n<2:
        myList=[];
        print(myList);
else:
        print("n must be larger than 0");
```

Note that:

The Sieve of Eratosthenes algorithm was inspired from the following link: link

Optimizations:

1. Prior to applying the Sieve of Eratosthenes algorithm, as an optimization all the even numbers were removed except for 2.
2. The outer for loop will loop from index 3 until the (square root of n)+1 , since all the factors would have been removed till the (square root of n)+1, and thus, removing excess iterations.

Testing Part 2) Sieve of Eratosthenes:

Testing Sieve of Eratosthenes with 102 elements:

```
[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]
```

Testing Sieve of Eratosthenes with 302 elements:

```
[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, 127,
131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 26
9, 271, 277, 281, 283, 293]
```

Question 7:

Code:

```
#Creating an object to fit the specifications of a BST where:
#1.node is the integer
#2.left is the pointer pointing to the left subtree
#3.right is the pointer pointing to the right subtree
class element(object):
    node=None;
    left = None;
    right = None;


#Method to add an item to the BST
def AddItem(value,tree,pointer,position):
    #Checking if list is empty
    if not tree:
        #Adding the root:
        newnode=element();
        newnode.node=value;
        tree.append(newnode);
        return;

    #if value is smaller or equal to current node
    if value<=tree[pointer].node:
        #if the left pointer is null it will attempt to add the
        #new node to the list (Base Case)
        if tree[pointer].left==None:
            tree[pointer].left=position;
            newnode=element();
            newnode.node=value;
            tree.append(newnode);

        #else it will traverse the left subtree (Recursive Case)
        else:
            AddItem(value,tree,tree[pointer].left,position);

    #if value is larger than current node
    elif value>tree[pointer].node:
        #if the right pointer is null it will attempt to add the
        #new node to the list (Base Case)
        if tree[pointer].right==None:
            tree[pointer].right=position;
            newnode=element();
            newnode.node=value;
            tree.append(newnode);

        #else it will traverse the right subtree (Recursive Case)
        else:
```

```
            AddItem(value,tree,tree[pointer].right,position);
```

```
#Method to Display tree
def DisplayTree(tree,pointer):
        #Printing current node information
        output="\nPointer: "+str(pointer)+"\t value: "+str(tree[pointer].node);
        if tree[pointer].left==None:
            output=output+"\t left value: None";
        else:
            output=output+"\t left value: "+str(tree[tree[pointer].left].node)+"\t";

        if tree[pointer].right==None:
            output=output+"\t right value: None";
        else:
            output=output+"\t right value: "+str(tree[tree[pointer].right].node);
        print(output);

        #Traversing left subtree
        if tree[pointer].left!=None:
            DisplayTree(tree,tree[pointer].left);

        #Traversing right subtree
        if tree[pointer].right!=None:
            DisplayTree(tree,tree[pointer].right);
```

```
#Asking the user how many values will be inputted
iterations=int(input("Enter how many values will be inputted: "));
uinput=0;
#Declaring empty tree structure
tree=[];

#User inputting integers and adding said values to tree
for i in range(0,iterations):
        uinput=int(input("Enter number: "+str(i+1)));
        AddItem(uinput,tree,0,i);#pointer set to 0 since always need to start traversing from root

#Displaying tree
DisplayTree(tree,0);
```

Testing:

Testing Question 7 with the following inputs:

```
Enter how many values will be inputted: 7
Enter number: 15
Enter number: 21
Enter number: 34
Enter number: 42
Enter number: 53
Enter number: 67
Enter number: 78
Pointer: 0      value: 5        left value: 1           right value: 7

Pointer: 1      value: 1        left value: None        right value: 4

Pointer: 2      value: 4        left value: 2           right value: None

Pointer: 3      value: 2        left value: None        right value: 3

Pointer: 4      value: 3        left value: None        right value: None

Pointer: 5      value: 7        left value: None        right value: 8

Pointer: 6      value: 8        left value: None        right value: None
```

Testing Question 7 with the following inputs including double values:

```
Enter how many values will be inputted: 5
Enter number: 120
Enter number: 221
Enter number: 31
Enter number: 44
Enter number: 520

Pointer: 0      value: 20       left value: 1           right value: 21

Pointer: 2      value: 1        left value: None        right value: 4

Pointer: 3      value: 4        left value: None        right value: 20

Pointer: 4      value: 20       left value: None        right value: None

Pointer: 1      value: 21       left value: None        right value: None
```

Testing Question 7 with the negative inputs:

```
Enter how many values will be inputted: 4
Enter number: 17
Enter number: 26
Enter number: 38
Enter number: 4-5

Pointer: 0      value: 7        left value: 6           right value: 8

Pointer: 1      value: 6        left value: -5          right value: None

Pointer: 3      value: -5       left value: None        right value: None

Pointer: 2      value: 8        left value: None        right value: None
```

Question 8:

Code:

```
import math

#Declaring counter to hold the number of iterations
counter=0;
#n is number
n=100;
#root being used for checking
root=n;
#approx used to hold the final approximation
approx=0;

#Looping until the root != the square root of n
while root!=math.sqrt(n) :
    if root!=math.sqrt(n) :
        approx=root;#Setting the approx to root
    #Performing calculation method
    root=(root+n/root)/2;
    #Incrementing counter
    counter=counter+1;

print("Approximation: "+str(approx));
print("Iterations: "+str(counter));
```

Note that:

The Newton Raphson Square Root Method was inspired from the following link: link

Testing:

Testing Question 8 with n as 100:

```
Approximation: 10.000000000139897
Iterations: 8
```

Testing Question 8 with n as 777:

```
Approximation: 27.874719729532714
Iterations: 10
```

Question 9:

Code:

```
#Declaring an empty array and filling said array with random values
array=[];
for i in range(0,100):
    array.append(random.randint(0,1024));
print("Unsorted array");
print(array);

#Sorting the array via QuickSort from Question 1) since worst case is nlog(n)
QuickSort(array,0,len(array)-1)
print("\nSorted array");
print(array)

counter=0;#Variable used to avoid printing repeated number twice

#Finding the repeated integers
#Looping through all the elements in the array
for i in range(0,len(array)-1):
    #Checking whether the current element and the next element are the same,
    #and that the number was not already printed
    if array[i]==array[i+1] and counter==0:
        print(str(array[i])+" is a repeated integer")
        #Incrementing counter
        counter=counter+1;
    else:
        #Setting counter to 0
        counter=0;
```

Note that:

Before checking for repeated integers, first Quick Sort was used to sort the array, to make the process of finding repeated integers easier, then the adjacent elements were compared with each other, to check if they were repeated (the latter required the use of a single for loop). This method requires n space and has a Best Case: of O(n log(n)) and a Worst Case: of O(n^2). This method was chosen, instead of opting to utilise 2 for loops (to traverse the list multiple times to find the repetitions) which have a time complexity of O(n^2) to present the same functionality.

Testing:

Testing Question 9 with a random populated array of 100 elements:

```
Unsorted array
[760, 637, 501, 414, 390, 614, 459, 176, 714, 954, 570, 508, 309, 672, 295, 358, 520, 924, 645, 1015, 251, 871, 488, 124, 594,
206, 448, 870, 19, 877, 318, 335, 375, 761, 784, 105, 912, 893, 61, 761, 123, 807, 77, 879, 536, 134, 218, 986, 939, 452, 191,
10, 30, 141, 580, 11, 644, 827, 339, 911, 566, 68, 539, 818, 481, 733, 110, 64, 721, 929, 81, 534, 488, 1001, 589, 439, 528, 89
7, 751, 222, 447, 630, 210, 860, 30, 805, 451, 614, 91, 464, 643, 605, 348, 199, 931, 88, 615, 1024, 153, 429]

Sorted array
[10, 11, 19, 30, 30, 61, 64, 68, 77, 81, 88, 91, 105, 110, 123, 124, 134, 141, 153, 176, 191, 199, 206, 210, 218, 222, 251, 29
5, 309, 318, 335, 339, 348, 358, 375, 390, 414, 429, 439, 447, 448, 451, 452, 459, 464, 481, 488, 488, 501, 508, 520, 528, 534,
536, 539, 566, 570, 580, 589, 594, 605, 614, 614, 615, 630, 637, 643, 644, 645, 672, 714, 721, 733, 751, 760, 761, 761, 784, 80
5, 807, 818, 827, 860, 870, 871, 877, 879, 893, 897, 911, 912, 924, 929, 931, 939, 954, 986, 1001, 1015, 1024]
30 is a repeated integer
488 is a repeated integer
614 is a repeated integer
761 is a repeated integer
```

```
Unsorted array
[361, 522, 989, 84, 87, 722, 747, 100, 37, 220, 375, 268, 73, 287, 761, 800, 331, 606, 667, 665, 528, 177, 989, 858, 394, 985,
234, 111, 971, 944, 850, 282, 734, 440, 389, 70, 645, 322, 199, 332, 165, 15, 627, 661, 877, 17, 239, 606, 50, 941, 373, 687, 8
88, 680, 842, 943, 338, 62, 105, 378, 1020, 969, 416, 814, 903, 298, 433, 302, 291, 511, 238, 640, 1022, 773, 222, 449, 362, 18
6, 391, 962, 44, 967, 673, 385, 66, 176, 815, 425, 952, 693, 513, 120, 923, 327, 409, 355, 1018, 572, 365, 129]

Sorted array
[15, 17, 37, 44, 50, 62, 66, 70, 73, 84, 87, 100, 105, 111, 120, 129, 165, 176, 177, 186, 199, 220, 222, 234, 238, 239, 268, 28
2, 287, 291, 298, 302, 322, 327, 331, 332, 338, 355, 361, 362, 365, 373, 375, 378, 385, 389, 391, 394, 409, 416, 425, 433, 440,
449, 511, 513, 522, 528, 572, 606, 606, 627, 640, 645, 661, 665, 667, 673, 680, 687, 693, 722, 734, 747, 761, 773, 800, 814, 81
5, 842, 850, 858, 877, 888, 903, 923, 941, 943, 944, 952, 962, 967, 969, 971, 985, 989, 989, 1018, 1020, 1022]
606 is a repeated integer
989 is a repeated integer
```

Question 10:

Code:

```
#Parameters include List, largest element and counter to hold the index of the
current element
def Largest(mylist,max,counter):
    #Base Case
    if counter==-1 :
        return max;
    #Recursive Case
    else :
        #if the max is smaller than current element
        #then max is set to be the current element
        if max<mylist[counter]:
            max=mylist[counter];
        #Calling recursive case and decrementing counter(index)
        return Largest(mylist,max,counter-1);
```

```
mylist=[];
#Populating list with random values
for i in range(0,100):
    mylist.append(random.randint(1,1024));
print(mylist);

print("\nThe largest number in list is:")
print(Largest(mylist,0,len(mylist)-1));
```

Testing:

Testing Question 10 with a random populated array of 100 elements:

```
[731, 514, 279, 141, 577, 35, 351, 432, 784, 232, 916, 423, 256, 121, 385, 292, 948, 941, 300, 915, 793, 967, 537, 714, 696, 14
2, 616, 216, 167, 171, 586, 242, 89, 879, 508, 882, 70, 409, 137, 401, 584, 456, 512, 659, 255, 607, 612, 293, 1005, 821, 320,
636, 548, 453, 872, 936, 619, 535, 602, 904, 1013, 707, 578, 425, 452, 368, 525, 989, 873, 137, 102, 426, 203, 12, 767, 986, 27
4, 623, 71, 312, 31, 980, 767, 495, 906, 839, 540, 332, 843, 292, 331, 878, 855, 893, 152, 308, 150, 94, 29, 732]

The largest number in list is:
1013
```

```
[177, 292, 839, 289, 307, 763, 979, 475, 211, 961, 687, 142, 804, 917, 331, 819, 495, 193, 1009, 803, 49, 273, 415, 121, 821, 9
19, 378, 82, 653, 294, 648, 121, 78, 588, 640, 591, 480, 605, 1010, 136, 552, 431, 350, 919, 490, 403, 563, 92, 771, 556, 526,
58, 92, 498, 214, 853, 740, 334, 151, 565, 4, 996, 630, 117, 630, 859, 983, 451, 1020, 435, 492, 311, 367, 390, 344, 405, 541,
19, 470, 57, 256, 989, 708, 588, 294, 992, 607, 273, 603, 569, 772, 624, 750, 77, 660, 459, 553, 384, 158, 705]

The largest number in list is:
1020
```

Question 11:

Code:

```
#Method factorial which will be used in calculation
def factorial(n):
    #Base Case
    if n==0:
        return 1;
    #Recursive Case
    else :
        return n*factorial(n-1);
```

```
#n is the number of iterations
n=7;
#Input needs to be in radians
x=1.5707;
#Declaring variable ans
ans=0;

for i in range(0,n):
    #Performing Calculation
    numerator=(x**(2*i+1));
    denominator=factorial(2*i+1);
    ans=ans+((-1)**i)*(numerator/denominator);

print("The input is "+str(x));
print("The approximation is "+str(ans));
```

Note that:

1. The Maclaurin of Sine Formula was inspired from the following link: link
2. The input x needs to be in radians

Testing:

Testing Question 11 with input x as 1.5707:

```
The input is 1.5707
The approximation is 0.9999999960227458
```

Testing Question 11 with input x as 3.1414:

```
The input is 3.1414
The approximation is 0.00021377680845532845
```

Question 12:

Code:

```
def Fibonacci(n):
    #Declaring and initialising variables:

    returnsum=0;#Variable to hold the sum of n numbers
    sum=0;#Temporary variable to perform addition of num1 and num2
    num1=1;#num1 to hold the first term in sequence
    num2=1;#num2 to hold the second term in sequence
    count=0;#Counter to be used for looping

    #Looping if count<n
    while count<n :
        #Adding the contents of num1 to returnsum
        returnsum=returnsum+num1;

        #Finding the next term in the Fibonacci Sequence and incrementing count
        sum=num1+num2;
        num1=num2;
        num2=sum;
        count=count+1;

    #Returning the sum of n numbers
    return returnsum;

count=int(input("Please input n\n"))
print("\nThe sum of the first "+str(count)+" numbers in the Fibonacci sequence is
"+str(Fibonacci(count)));
```

Testing:

Testing Question 12 with input n as 5:

```
Please input n
5

The sum of the first 5 numbers in the Fibonacci sequence is 12
```

Testing Question 12 with input n as 7:

```
Please input n
7

The sum of the first 7 numbers in the Fibonacci sequence is 33
```