Data Structures & Algorithms Documentation

Tristan Oa Galea
B.S.C. IN COMPUTER SCIENCE

Table of Contents

Foreward

This assignment was programmed using the Python programming language. The documentation for this assignment is individually split up based on each task.

Each task explanation is divided into 3 sections: (i) Task explanation, (ii) Actual source code and (iii) Code testing. For each task, (i) a brief overview of what has been achieved in the task performed is discussed. Following this, (ii) the actual python code is shown. Finally, (iii) the testing of this code is explained with the help of screenshots.

Each task explanation gives a brief overview of what has been achieved in the task performed. A more detailed explanation of sections of code can be found in the comments included in the pasted source code.

Task 1 Explanation:

In this first task, 2 empty lists were created. The user was then asked to enter a value for the desired length of the first list. A function called ‘user_input()’ was created to accept this user input and validate it using try and except statements in the following ways:

- The function ensured that only integer values could be stored. This was done so as to make sure that the user enters valid data. If the user enters any other data type such as a character, he/she is informed and prompted to re-enter a length for the list without causing the program to crash.
- A further validation performed was that of assuring that the user enters a length greater than or equal to 256, as specified.

After the user successfully enters a value which satisfies both validation techniques, another function ‘populate_array()’ is called which populates the list randomly with integers from a range of 0-1024 up to the length entered by the user. The randomly generated list is then displayed to the user.

This same procedure is repeated for array B, this time, with an extra validation method involved. This consists of a while loop which was implemented to ensure that the length of the second list, upon passing both previous validation tests, is not equal to that of the first. This was done to satisfy the question requirements.

After having successfully created, populated and printed out the arrays, the 2 lists are then passed through 2 separate functions to be sorted. The ‘shell_sort()’ function is called to sort ‘array_a’ and the user is informed of the gap size and an updated version of the list based on the sorting of that gap size. The final version of the list, having a gap size of 1 is displayed as the last iteration.

Moving on, the ‘quick_sort()’ function is called to sort ‘array_b’. With every iteration, the user is informed of the splitting factor which is set to be approximately the element in the middle of the list. The lists smaller and greater than the number are also displayed to show the user the new lists

on which the ‘quick_sort()’ function is going to be called again. These 3 qualities are displayed to the user with each iteration in a random manner. This means that the user might encounter the splitting of a particular list at a far later stage in the sorting algorithm (ie a newly split list might be split again and displayed to the user after a couple of different iterations and not exactly after). Although this may be confusing, nothing could be done to ensure that the lists split up one following the other. Finally, the sorted ‘array_b’ is printed out to the user.

Task 1 was successfully implemented and fit all the specified requirements without having any bugs.

Task 1 Code:

```python
# importing the module 'random' to be able to generate random numbers
import random


# A function which validates the input entered by the user such that the user is informed
# and requested to re-enter a value if characters or numbers less than 256 are inputted.
# This function has no parameters passed to it but returns an integer value
def user_input():

    # A while loop which runs indefinitely
    while True:

        # Section of code which handles exceptions
        try:

            # Converts to the integer data type and stores the value entered by the user
            x = int(input("\nEnter the max amount of integers in the list: "))

            # Checks if the value entered fits in a particular range before proceeding
            assert(x >= 256)

            # If all checks are successful, the value entered is returned
            return x

        # Informs the user if anything other than an integer is inputted
        except ValueError:
            print("Invalid input!")

        # Informs the user that a particular range of numbers is valid only
        except AssertionError:
            print("Number must be greater than or equal to 256")
```

```python


# A function which simply populates an array based on the 2 parameters passed.
# The first parameter is the size of the array entered by the user and the second
# is the actual array to be populated. Nothing is returned back
def populate_array(x, array):

    # A for loop which loops through each individual empty element in the array
    # and populates it
    for i in range(x):

        # Generates a random number from 0-1024 each time and stores it in the array
        array.append(random.randint(0, 1024))

```

```
46
47  # The main function for a shell sort which also calls the 'split_list()' function.
48  # This takes 2 parameters which are the array size and the array itself and returns nothing
49  def shell_sort(x, array):
50
51      # The approximate middle position of the list is determined and stored
52      skip = x // 2
53
54      # Loops as long as skip > 0
55      while skip > 0:
56
57          # Loops up to the splitting factor
58          for i in range(skip):
59
60              # Calls the split_list() function with the appropriate parameters passed
61              split_list(array, i, skip, x)
62
63          # Displays the list being sorted in stages whilst halving the skip value each time
64          print(skip, "\t\t\t", array)
65          skip = skip // 2
66
67
68  # The sub-function for the shell sort which compares and sorts the list.
69  # This has 4 parameters passed into it but returns nothing. The first is the array
70  # to be sorted, the second the starting element to be compared, the third the amount
71  # of integers to be skipped each time and the fourth, the length of the list
72  def split_list(array, start, skip, x):
73
```

```
74      # Loops through the list to compare and sort
75      for i in range(start + skip, x, skip):
76
77          # Initialising of 2 new variables. val stores the array value at position i
78          # and j stores the index of the current iteration
79          val = array[i]
80          j = i
81
82          # Loops as long as the value being skipped <= j and the integer stored in val
83          # is less than another integer being compared to
84          while skip <= j and val < array[j-skip]:
85
86              # Shifts the smaller number to take the larger number's place
87              array[j] = array[j-skip]
88              j = j-skip
89
90          # Shifts the larger number in its new appropriate position
91          array[j] = val
92
93
```

```
# The main function for a quick sort which also calls the 'partition()' function.
# This is a recursive function taking 3 parameters which are the array to be sorted,
# the starting position and ending position for the sorting procedure.
# This function does not return anything.
def quick_sort(array, start, end):

    # Statement runs if the end of the list is not yet reached
    if start < end:

        # The value returned from the sub-function being called is stored in
        # the placeholder split
        split = partition(array, start, end)

        # List-related information is printed out to the user in stages
        # as the list gets sorted
        print("\nSplitting Factor:\t\t", array[split])
        print("Smaller or equal to:  \t", array[start:split])
        print("Larger: \t\t\t\t", array[split+1:])

        # The recursive part of the function which proceeds to sorting 2 smaller
        # sections of the list separated by the split variable
        quick_sort(array, start, split-1)
        quick_sort(array, split+1, end)


# The sub-function of the quick sort which switches different numbers in the
# list to be sorted based on the pin. This takes 3 parameters:
# the array to be sorted, the starting and ending point.
# This function returns the value stored in i which determines the split
# for the next iteration
def partition(array, start, end):

    # Sets the pin to be the one approximately in the middle of the list
    pinPosition = (start + end) // 2
    pinValue = array[pinPosition]

    # Swaps the pin with the last element in the list
    array[pinPosition], array[end] = array[end], array[pinPosition]

    # Sets the counter i to 1 less than the first position of the list
    i = start - 1

    # Loops through the list from start to end excluding the pin
    for j in range(start, end):

        # If the pin is greater than an element being compared, the
        # counter variable i is incremented and the values stored in
        # positions i and j are swapped
        if array[j] <= pinValue:
            i += 1
            array[j], array[i] = array[i], array[j]

    # i is incremented
    i += 1

    # The pin is now swapped back in it's appropriate position in the list
    array[i], array[end] = array[end], array[i]

    # The counter value i is returned
    return i
```

```
154
155
156     # The 2 empty lists to be populated and sorted are initialised here
157     array_a = []
158     array_b = []
159
160     print("\t\t\t\t\t\tARRAY A")
161     print("------------------------------------------------------------\n")
162
163     # Placeholder length1 stores the value entered by the user after validation
164     length1 = user_input()
165
166     # The 1st array is populated when populate_array() function is called
167     populate_array(length1, array_a)
168
169     # The 1st array is displayed to the user
170     print("Array A: ", array_a)
171
172     print("\n\t\t\t\t\t\tARRAY B")
173     print("------------------------------------------------------------\n")
174
175     # Placeholder length2 stores the value entered by the user after validation
176     length2 = user_input()
177
178     # Ensures that the 2 lists created are not of equal length
179     while length1 == length2:
180         print("The different arrays cannot be of the same size")
181         length2 = user_input()
```

```
182
183     # The 2nd array is populated when populate_array() function is called
184     populate_array(length2, array_b)
185
186     # The 2nd array is displayed to the user
187     print("Array B: ", array_b)
188
189     print("\n\t\t\t\tSorting array A using Shell Sort")
190     print("-------------------------------------------------------------------------------------------------\n")
191     print("Original Array A: ", array_a)
192     print("\nGap Size |\t Current list")
193
194     # The shell_sort() funtion is called which sorts the list
195     shell_sort(length1, array_a)
196
197     print("\n\t\t\t\tSorting array B using Quick Sort")
198     print("-------------------------------------------------------------------------------------------------\n")
199     print("Original Array B: ", array_b)
200
201     # The quick_sort() function is called which sorts the list
202     quick_sort(array_b, 0, length2-1)
203     print("\nSorted Array B: ", array_b)
```

Task 1 Testing:

Upon running the program, the user is asked to enter the maximum amount of integers in the first list:

```
                        ARRAY A
------------------------------------------------------------


Enter the max amount of integers in the list:
```

Assuming the user misunderstands what is required of him and instead enters:

1) A character
2) Multiple characters
3) A float
4) A number less than 256
5) A negative number
6) 'Enter' key

```
Enter the max amount of integers in the list: c
Invalid input!

Enter the max amount of integers in the list: What should I enter?
Invalid input!

Enter the max amount of integers in the list: 358.2
Invalid input!

Enter the max amount of integers in the list: 200
Number must be greater than or equal to 256

Enter the max amount of integers in the list: -300
Number must be greater than or equal to 256

Enter the max amount of integers in the list:
Invalid input!
```

For each of the above incorrect instances, the program handles the exception correctly and keeps waiting for an appropriate integer value before proceeding.

Considering now that the user understands what is expected of him, the value entered is successfully stored and a list of size as specified by user is generated. The value the user enters is 280 and a list of 280 elements is randomly generated. Below, a fraction of the list created is displayed.

```
Enter the max amount of integers in the list: 280
Array A:  [564, 556, 364, 461, 933, 805, 875, 73, 960, 718, 158, 564, 704, 516, 309, 997, 978, 76, 842,
```

Now that list A has been successfully populated, list B is now left to be created. Like before, the user is asked once again to enter a length for this list.

```
                        ARRAY B
------------------------------------------------------------


Enter the max amount of integers in the list:
```

Assuming the user still does not understand what is expected of him and enters garbage data, the program continues persisting for the correct data as shown.

```
Enter the max amount of integers in the list: f
Invalid input!

Enter the max amount of integers in the list: I am confused!!@#$%^
Invalid input!

Enter the max amount of integers in the list: 150
Number must be greater than or equal to 256

Enter the max amount of integers in the list: -1000
Number must be greater than or equal to 256

Enter the max amount of integers in the list: 3456.3456
Invalid input!
```

Given that the user now enters a value in the range specified (ie a value not less than 256) but equal to the length of the previous list, the program prints on the screen the following:

```
Enter the max amount of integers in the list: 280
The different arrays cannot be of the same size
```

The user finally understands and this time enters the value 300. This data is accepted and the program generates 300 random elements which are stored in list B. This list is then displayed to the user.

```
Enter the max amount of integers in the list: 300
Array B:  [736, 141, 23, 497, 257, 481, 261, 189, 265, 766, 201, 385, 299, 1023, 226, 333, 925, 596, 524, 7, 753, 680, 556, 501,
```

From this point onwards, the program sorts list A using Shell Sort and list B using Quick Sort as can be observed. Due to the large amount of elements in the list, only a fraction of each list being sorted is shown in the screenshots.

```
                Sorting array A using Shell Sort
-------------------------------------------------------------------------------------------

Original Array A:  [564, 556, 364, 461, 933, 805, 875, 73, 960, 718, 158, 564, 704, 516, 309, 997, 978, 76, 842, 879

Gap Size |   Current list
140              [564, 327, 364, 27, 463, 615, 645, 73, 583, 423, 158, 564, 659, 516, 309, 252, 256, 76, 454, 162, 2
70            [144, 148, 150, 27, 463, 615, 585, 73, 348, 34, 158, 205, 349, 340, 309, 252, 198, 76, 454, 162, 289, 3
35            [144, 124, 150, 27, 71, 145, 165, 73, 305, 34, 158, 205, 108, 183, 303, 252, 46, 76, 189, 162, 72, 125,
17            [76, 124, 144, 27, 71, 143, 165, 73, 48, 34, 28, 16, 26, 48, 181, 110, 46, 103, 138, 148, 72, 125, 145,
8             [46, 34, 28, 16, 26, 40, 108, 73, 48, 103, 109, 27, 71, 48, 145, 110, 76, 124, 138, 145, 71, 125, 165,
4             [26, 34, 28, 16, 46, 40, 108, 27, 48, 48, 109, 73, 71, 103, 138, 110, 71, 124, 144, 145, 72, 125, 144,
2             [26, 16, 28, 27, 46, 34, 48, 40, 71, 48, 71, 73, 72, 103, 76, 110, 108, 124, 109, 125, 138, 143, 144, 1
1             [16, 26, 27, 28, 34, 40, 46, 48, 48, 71, 71, 72, 73, 76, 103, 108, 109, 110, 124, 125, 138, 143, 144, 1
```

The screenshot shown above displays all the gap sizes required for the sorting of the list A with the final sorted list as the last line. In the screenshots below we can see parts of the sorting process for list B. Quick Sort, being much more demanding than Shell Sort, requires more steps for the list to be sorted. For this reason, only 3 random samples of this process are shown in the next 3 screenshots:

```
                Sorting array B using Quick Sort
-----------------------------------------------------------------------------------------

Original Array B:  [736, 141, 23, 497, 257, 481, 261, 189, 265, 766, 201, 385, 299, 1023, 226, 333,

Splitting Factor:        897
Smaller or equal to:     [736, 141, 23, 497, 257, 481, 261, 189, 265, 766, 201, 385, 299, 226, 333,
Larger:                  [1023, 937, 939, 994, 1009, 925, 902, 977, 929, 923, 1005, 990, 941, 924, 1

Splitting Factor:        211
Smaller or equal to:     [141, 23, 189, 201, 7, 140, 87, 91, 133, 20, 131, 101, 176, 160, 187, 77, 1
Larger:                  [768, 834, 632, 772, 686, 766, 476, 789, 497, 778, 459, 839, 385, 247, 245,

Splitting Factor:        90
Smaller or equal to:     [23, 7, 87, 20, 77, 47, 27, 50, 48, 75, 63, 41, 15, 69, 68, 42, 48, 48, 42]
Larger:                  [172, 189, 102, 91, 210, 133, 160, 201, 201, 131, 107, 200, 101, 145, 176,

Splitting Factor:        75
Smaller or equal to:     [23, 7, 20, 47, 27, 50, 48, 42, 63, 41, 15, 69, 68, 42, 48, 48]
Larger:                  [87, 77, 90, 172, 189, 102, 91, 210, 133, 160, 201, 201, 131, 107, 200, 101
```

```
Splitting Factor:        510
Smaller or equal to:     [510]
Larger:                  [513, 524, 626, 596, 562, 619, 571, 568, 580, 671, 578, 611, 677, 565, 548, 546, 660, 6

Splitting Factor:        578
Smaller or equal to:     [562, 571, 568, 578, 565, 548, 546, 551, 537, 564, 557, 535, 542, 556, 536, 569, 548]
Larger:                  [603, 619, 593, 641, 611, 684, 677, 594, 596, 680, 612, 670, 612, 664, 601, 657, 586, 6

Splitting Factor:        537
Smaller or equal to:     [535, 536]
Larger:                  [578, 565, 548, 546, 551, 548, 564, 557, 562, 542, 556, 571, 569, 568, 578, 603, 619, 5

Splitting Factor:        535
Smaller or equal to:     []
Larger:                  [536, 537, 578, 565, 548, 546, 551, 548, 564, 557, 562, 542, 556, 571, 569, 568, 578, 6

Splitting Factor:        564
Smaller or equal to:     [548, 546, 551, 548, 557, 562, 542, 556]
Larger:                  [568, 578, 571, 569, 565, 578, 603, 619, 593, 641, 611, 684, 677, 594, 596, 680, 612, 6

Splitting Factor:        548
Smaller or equal to:     [548, 546, 542]
Larger:                  [557, 562, 551, 556, 564, 568, 578, 571, 569, 565, 578, 603, 619, 593, 641, 611, 684, 6
```

```
Splitting Factor:        1009
Smaller or equal to:     [1009]
Larger:                  [1010, 1015, 1020, 1023, 1023, 1021, 1024]

Splitting Factor:        1023
Smaller or equal to:     [1015, 1020, 1023, 1021]
Larger:                  [1024]

Splitting Factor:        1020
Smaller or equal to:     [1015]
Larger:                  [1023, 1021, 1023, 1024]

Splitting Factor:        1023
Smaller or equal to:     [1021]
Larger:                  [1023, 1024]
```

Finally, the sorted list B is displayed upon completion:

```
Sorted Array B:  [7, 15, 20, 23, 27, 41, 42, 42, 47, 48, 48, 48, 50, 63, 68, 69, 75, 77, 87, 90, 91, 96, 101, 102, 107, 113, 125, 131,
```

Task 2 Explanation:

After completing and testing task 1, it was time to move on to task 2. In this task, all the contents of task 1 were imported at the very start so that they could be included rather than having to code everything again. Therefore when running the program, 2 empty arrays are initialized, populated and sorted from the start.

In this task, only one function was declared. This is called the 'merge()' function which initializes an 'array_c' with exactly the length of 'array_a' and 'array_b'. For this to be done, all the elements in list C has to be set to 'NONE' initially.

Moving on, lists A and B would be compared when passed as the 1st and 2nd parameters to the 'merge()' function. The smaller element out of the elements being compared would be added to position 'c' in the 'array_c'. The counters of list C and of the list whose value has been added to list C are then incremented by 1 to point to the next position in both lists. This process would repeat until the end of both lists A and B is reached.

This function was originally coded as is and the list was printed out at the end of the merging. However, when inspecting the new merged list C, it was evident that the last few elements were still set as 'None'. This implied that not all the elements from either list A or B had been added. To fix this, 2 separate while loops, one for A and another for B were added to append any remaining elements in the previous lists to the new one. 'array_c' would then be printed out after these 2 loops as the last line in this function.

When calling the 'merge()' function, the 2 lists passed and their respective lengths had to include a 'task_1.' prefix to indicate that the lists and variables passed were stored in the external 'task_1.py' file.

Task 2 was successfully implemented and fit all the specified requirements without having any bugs.

Task 2 Code:

```
# The previous task_1 python file used for task 1 is now imported
# Any functions previously defined in the task_1 python file can now be
# accessed from this file
# Also, any previous operations performed like declaration of arrays and
# the actual sorting of these arrays will still be done in this file
# giving the illusion that such operations where actually coded in this file
import task_1


# The function used to merge array_a and array_b into a newly created array_c
# This function takes 4 parameters with the first 2 being the 2 arrays to be
# merged and the last 2 being their respective sizes
# Nothing is returned back as a result of calling this function
def merge(array1, array2, size1, size2):

    # The new array C is created with the sizes of both arrays A and B
    # together and each entry of the array is initialised to 0
    array_c = [None] * (size1 + size2)

    # Counter variables for arrays A, B & C respectively are set to 0
    a = 0
    b = 0
    c = 0

    # A while loop which runs until the end of both lists A & B is reached
    while a < size1 and b < size2:

        # If the an element stored in A is less than that stored in B,
        # this is stored in C and both counters for a and c are incremented
        if array1[a] < array2[b]:
            array_c[c] = array1[a]
            c += 1
            a += 1

        # else, the element at B is stored in C and both counters for
        # b and c are incremented
        else:
            array_c[c] = array2[b]
            c += 1
            b += 1

```

```
    # Any remaining elements of A are stored in C
    while a < size1:
        array_c[c] = array1[a]
        c += 1
        a += 1

    # Any remaining elements of B are stored in C
    while b < size2:
        array_c[c] = array2[b]
        c += 1
        b += 1

    # The merged array C is then printed
    print("\n\nMerged Array C: \t", array_c)


# The merge() function is called with the 4 appropriate parameters passed
# Each parameter contains a "task_1." prefix which causes the program
# to refer to the 'task_1' python file and obtain the arrays and lengths
# required from that file
merge(task_1.array_a, task_1.array_b, task_1.length1, task_1.length2)
```

Task 2 Testing:

When ‘task_2’ is run, the contents of ‘task_1’ are evaluated first due to the ‘import’ preprocessor command. Thus the user has to enter valid values for the lengths of lists A and B respectively. The user input entered below shows incorrect followed by valid input:

```
                        ARRAY A
------------------------------------------------------------


Enter the max amount of integers in the list: 345.6
Invalid input!

Enter the max amount of integers in the list: 300
Array A:  [864, 493, 182, 643, 486, 235, 190, 641, 798, 869, 480, 564, 166, 241, 344, 417, 868, 144, 69, 461, 839,

                        ARRAY B
------------------------------------------------------------


Enter the max amount of integers in the list: twenty
Invalid input!

Enter the max amount of integers in the list: 300
The different arrays cannot be of the same size

Enter the max amount of integers in the list: 290
Array B:  [241, 645, 983, 357, 757, 1019, 612, 778, 48, 514, 1, 138, 331, 447, 260, 379, 284, 202, 1006, 53, 358,
```

‘array_a’ was assigned a length of 300 elements whilst that of ‘array_b’ was 290. Parts of the 2 randomly generated lists can also be seen in the above screen dump. These lists were then sorted as before and the final merged ‘array_c’ is displayed to the user:

```
               Sorting array A using Shell Sort
------------------------------------------------------------------------------------------

Original Array A:  [864, 493, 182, 643, 486, 235, 190, 641, 798, 869, 480, 564, 166, 241, 344, 417, 868, 144, 69, 461,

Gap Size |   Current list
150              [136, 493, 182, 424, 445, 69, 190, 641, 62, 143, 480, 60, 166, 241, 41, 359, 508, 45, 69, 114, 284, 4
75           [121, 231, 182, 23, 307, 40, 49, 528, 62, 143, 76, 60, 166, 96, 41, 359, 131, 45, 69, 114, 246, 308, 114,
37           [75, 136, 182, 23, 307, 40, 49, 111, 62, 143, 76, 34, 99, 6, 41, 16, 131, 45, 69, 102, 47, 263, 114, 56,
18           [54, 96, 47, 23, 114, 40, 20, 7, 17, 20, 36, 34, 60, 6, 41, 16, 64, 45, 69, 102, 114, 140, 227, 56, 49, 5
9            [20, 36, 34, 23, 6, 40, 16, 7, 17, 54, 43, 47, 60, 16, 41, 20, 50, 45, 69, 76, 86, 99, 114, 56, 34, 58, 6
4            [6, 16, 16, 7, 17, 36, 34, 20, 20, 40, 41, 23, 34, 45, 43, 47, 50, 49, 61, 56, 60, 54, 64, 62, 69, 58, 69
2            [6, 7, 16, 16, 17, 20, 20, 23, 34, 36, 34, 40, 41, 45, 43, 47, 50, 49, 60, 54, 61, 56, 64, 58, 69, 62, 69
1            [6, 7, 16, 16, 17, 20, 20, 23, 34, 34, 36, 40, 41, 43, 45, 47, 49, 50, 54, 56, 58, 60, 61, 62, 64, 69, 69
```

```
Original Array B:  [241, 645, 983, 357, 757, 1019, 612, 778, 48, 514, 1, 138, 331, 447, 260, 379, 284, 202,

Splitting Factor:        750
Smaller or equal to:     [241, 645, 357, 612, 48, 514, 1, 138, 331, 447, 260, 379, 284, 202, 53, 358, 501,
Larger:                  [981, 892, 847, 842, 987, 809, 1009, 847, 906, 815, 834, 908, 925, 777, 894, 987,

Splitting Factor:        261
Smaller or equal to:     [241, 48, 1, 138, 260, 202, 53, 216, 89, 103, 234, 259, 92, 22, 154, 209, 109, 1,
Larger:                  [589, 584, 681, 588, 485, 549, 465, 652, 299, 645, 372, 657, 526, 643, 402, 602, 5

Splitting Factor:        50
Smaller or equal to:     [48, 1, 22, 1, 1, 33, 46, 36, 41, 18, 28, 31, 43]
Larger:                  [154, 209, 109, 138, 166, 140, 170, 83, 228, 241, 189, 163, 119, 228, 261, 180, 11

Splitting Factor:        46
Smaller or equal to:     [1, 22, 1, 1, 33, 43, 36, 41, 18, 28, 31]
Larger:                  [48, 50, 154, 209, 109, 138, 166, 140, 170, 83, 228, 241, 189, 163, 119, 228, 261,
```

```
Splitting Factor:        1006
Smaller or equal to:     []
Larger:                  [1017, 1019, 1009]

Splitting Factor:        1019
Smaller or equal to:     [1017, 1009]
Larger:                  []

Splitting Factor:        1017
Smaller or equal to:     [1009]
Larger:                  [1019]

Sorted Array B:  [1, 1, 1, 18, 22, 28, 31, 33, 36, 41, 43, 46, 48, 50, 53, 55, 64, 67, 70, 80, 80, 82, 83,
```

```
Merged Array C:      [1, 1, 1, 6, 7, 16, 16, 17, 18, 20, 20, 22, 23, 28, 31, 33, 34, 34, 36, 36, 40, 41, 41, 43, 43,
```

Task 3 Explanation:

In this task, it was required to create a function which identifies the extreme points of a given array.

To start off, an empty ‘arrayA’ was initialized. The user was then asked to enter the length the list would have as well as the min and max values that could be present when calling the ‘user_input()’ function. This function consisted of a modified version of the one previously defined in task_1. It consisted of a try-except statement which would ensure that the user enters only integer values. If not, a warning message is displayed allowing the user to enter the value again. Upon successfully entering the first value, a counter ‘i’ initially set to 1 would be incremented and the if statement which runs only when ‘I’ is 2 is run. The user is now asked to enter the min value present in the list. When appropriate data is entered, ‘i’ is incremented once again and the max value present in the list is requested from the user.

These values are then returned back and stored in placeholders to be passed as parameters (including the actual ‘arrayA’ as the 4th argument) to another function which populates the list with randomly generated integers. This is the ‘generate_list()’ function which works on a similar concept to that of task_1 with the only difference of having variables entered by the user determine the min and max values in the list.

After the list is generated, it is displayed to the user. Following this, the newly created ‘extreme()’ function (with the array to be tested and its length as parameters) is called which simply determines the extreme points of the list.

In this function, an array ‘points’, to store the extreme points of ‘arrayA’ is defined. Then a for loop, which loops through all the possible extreme points (excluding the first and the last elements) checks for these points and appends them to the ‘points’ list. Following this is another if statement, which lets the user know that a list is sorted if no extreme points are found. If not, the extreme points found are displayed to the user.

To test if a sorted array has any extreme points, a ‘quick_sort()’ function was created, identical to the one in task 1. The ‘arrayA’ list was then sorted using this function and displayed to the user in ascending order. This newly sorted array was then passed through the ‘extreme()’ function and as expected, the array had no extreme points. This was because the elements were now one after the other in ascending order and neither

$$arr[i-1] < arr[i] > arr[i+1] \text{ nor } arr[i-1] > arr[i] < arr[i+1]$$

Task 3 was successfully implemented and fit all the specified requirements without having any bugs.

Task 3 Code:

```python
# importing the module 'random' to be able to generate random numbers
import random


# A function which validates the input entered by the user such that the user is informed
# and requested to re-enter a value if characters are inputted instead of numbers.
# This function has no parameters passed to it but returns 3 integer values
def user_input():

    # A counter variable is initialised to 1
    i = 1

    # A while loop which runs indefinitely
    while True:

        # Section of code which handles exceptions
        try:

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 2
            if i is 1:
                x = int(input("\nEnter the max amount of integers in the list: "))
                i += 1
                continue

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 3
            elif i is 2:
                y = int(input("\nEnter the min value an integer present in the list can have: "))
                i += 1
                continue

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 4
            elif i is 3:
                z = int(input("\nEnter the max value an integer present in the list can have: "))
                i += 1
                continue

            # The user is informed in this part that all inputs were correct and these
            # values are then returned back
            elif i is 4:
                print("All inputs were successfully stored and the list is shown below:\n")
                return x, y, z

        # Informs the user if anything other than an integer is inputted
        except ValueError:
            print("Invalid input!")

```

```
50
51  # A function which based on the 3 parameters passed (size of list, min value in list and
52  # max value in list) which were previously entered by the user, generates a list of randoms
53  # These numbers are then stored to the 'arr' array passed as the 4th parameter
54  def generate_list(x, y, z, arr):
55
56      # A for loop which loops through each individual empty element in the array
57      # and populates it
58      for i in range(x):
59
60          # Generates a random number from y-z each time and stores it in the array
61          arr.append(random.randint(y, z))
62
63
```

```
64  # The main function for a quick sort which also calls the 'partition()' function.
65  # This is a recursive function taking 3 parameters which are the array to be sorted,
66  # the starting position and ending position for the sorting procedure.
67  # This function does not return anything.
68  def quick_sort(array, start, end):
69
70      # Statement runs if the end of the list is not yet reached
71      if start < end:
72
73          # The value returned from the sub-function being called is stored in
74          # the placeholder split
75          split = partition(array, start, end)
76
77          # The recursive part of the function which proceeds to sorting 2 smaller
78          # sections of the list separated by the split variable
79          quick_sort(array, start, split-1)
80          quick_sort(array, split+1, end)
81
82
```

```
83  # The sub-function of the quick sort which switches different numbers in the
84  # list to be sorted based on the pin. This takes 3 parameters:
85  # the array to be sorted, the starting and ending point.
86  # This function returns the value stored in i which determines the split
87  # for the next iteration
88  def partition(array, start, end):
89
90      # Sets the pin to be the one approximately in the middle of the list
91      pinPosition = (start + end) // 2
92      pinValue = array[pinPosition]
93
94      # Swaps the pin with the last element in the list
95      array[pinPosition], array[end] = array[end], array[pinPosition]
96
97      # Sets the counter i to 1 less than the first position of the list
98      i = start - 1
99
```

```
    # Loops through the list from start to end excluding the pin
    for j in range(start, end):

        # If the pin is greater than an element being compared, the
        # counter variable i is incremented and the values stored in
        # positions i and j are swapped
        if array[j] <= pinValue:
            i += 1
            array[j], array[i] = array[i], array[j]

    # i is incremented
    i += 1

    # The pin is now swapped back in it's appropriate position in the list
    array[i], array[end] = array[end], array[i]

    # The counter value i is returned
    return i


```

```
# A function which finds the extreme points in an array
# It has 2 parameters passed to it (the array whose extreme points are to be found
# and the size of the array passed) but returns nothing back
def extreme(arr, x):

    # An empty list to store the extreme points is created
    points = []

    # The statement which loops through all the possible extreme points
    # (from the second to the element before the last) and adds any extreme
    # points found to the newly created array
    for i in range(1, x-1):
        if (arr[i-1] < arr[i] > arr[i+1]) or (arr[i-1] > arr[i] < arr[i+1]):
            points.append(arr[i])

    # This statement checks if any extreme points have been found
    # If none are found, the user is informed that the list is sorted
    # If not, the extreme points found are printed out to the user
    if len(points) == 0:
        print("\nSORTED")
        print("This array had no extreme points")
    else:
        print("\nThe extreme points are: ")
        print(points)


```

```
# An empty array to be randomly populated is created
arrayA = []

print("This program will print the extreme points of a given array.")

# The input entered by the user for the size, min and max values of the list
# are validated in the user_input() function. These are then
# returned and stored in the variables a, b & c respectively
a, b, c = user_input()

# The function generate_list() is then called and the placeholders a, b, c are
# passed as parameters to this function
generate_list(a, b, c, arrayA)

# The generated list is displayed to the user
print("ARRAY:", arrayA)

# The extreme() function is called with the 'arrayA' list passed as the first
# parameter and its size as the second parameter
extreme(arrayA, a)

# The quick_sort() function is called which sorts the list to show that a
# sorted list has no extreme points
# This has the 'arrayA' list passed as the 1st parameter, 0 as the 2nd and
# the length of the array - 1 as the 3rd
quick_sort(arrayA, 0, a-1)
print("\nSorted Array: ", arrayA)

# The extreme() method is called once again to prove that a sorted list
# contains no extreme points
extreme(arrayA, a)
```

Task 3 Testing:

Upon running the program, the user is requested to enter the length of the list he/she wants to create. Consider 2 invalid input attempts are entered before a correct value is entered and stored.

```
This program will print the extreme points of a given array.

Enter the max amount of integers in the list: ninety
Invalid input!

Enter the max amount of integers in the list: 67.5
Invalid input!

Enter the max amount of integers in the list: 30

Enter the min value an integer present in the list can have:
```

When a valid value is entered, this is stored and the program asks for the minimum value possible to be present in the list to be generated. Consider the user enters a further 2 invalid inputs before entering a 3rd successful one.

```
Enter the min value an integer present in the list can have: minimum$$^
Invalid input!

Enter the min value an integer present in the list can have: 1.000
Invalid input!

Enter the min value an integer present in the list can have: 1

Enter the max value an integer present in the list can have:
```

Moving on, the value ‘1’ is stored as the minimum possible value for the list to be generated. Now, the user is asked one last time to enter a max value to be present in the list. Consider the user hits ‘Enter’ without entering anything on the 1st attempt followed by a 2nd successful attempt. This leads to the list being generated based on the input fed by the user as shown:

```
Enter the max value an integer present in the list can have:
Invalid input!

Enter the max value an integer present in the list can have: 50
All inputs were successfully stored and the list is shown below:

ARRAY: [45, 12, 31, 45, 16, 23, 49, 42, 22, 44, 18, 23, 24, 50, 3, 34, 49, 19, 33, 50, 21, 7, 38, 37, 27, 38, 18, 34, 24, 10]
```

From now on, the program runs till the end without requiring any further user input. Following the printed generated array is another array which displays the extreme points found.

```
ARRAY: [45, 12, 31, 45, 16, 23, 49, 42, 22, 44, 18, 23, 24, 50, 3, 34, 49, 19, 33, 50, 21, 7, 38, 37, 27, 38, 18, 34, 24, 10]

The extreme points are:
[12, 45, 16, 49, 22, 44, 18, 50, 3, 49, 19, 50, 7, 38, 27, 38, 18, 34]
```

As explained before, to fully understand if a sorted array has any extreme points or not, the array was then passed through a quick_sort()' function and the sorted result was displayed. This newly sorted array was then passed through the 'extreme()' function once again and as expected, no extreme points were found.

```
Sorted Array:  [3, 7, 10, 12, 16, 18, 18, 19, 21, 22, 23, 23, 24, 24, 27, 31, 33, 34, 34, 37, 38, 38, 42, 44, 45, 45, 49, 49, 50, 50]

SORTED
This array had no extreme points

Process finished with exit code 0
```

Task 4 Explanation:

In this task, a program was written to identify all 2-pairs of integers that have the same product given that the 4 multiplied are all different.

Initially, an empty ‘numbers’ list was instantiated. This would be populated based on a list size entered by the user. The user input for the list size was validated using the ‘user_input()’ function. Just like task 1, this consisted of a try-except statement which would ensure that the user enters an integer value only. The only difference from the one implemented in task 1 is that the user has to enter a value greater or equal to 4. This was done to ensure that the list consisted of at least 4 elements, which was the minimum amount of elements that could be compared.

Moving on, the ‘user_input()’ function returns the valid data entered and passes it as a parameter to the ‘populate_array()’ function, along with the list to be populated. Once again, this function is identical to the one implemented in task 1 with the only difference that numbers are generated from 1-1024 instead of 0-1024. Then, the list is generated and printed out to the user.

The final function implemented in this task, which also happens to be the function which identifies all the 2-pairs of integers having the same product, is then called with the list size as the 1st parameter and the list as the 2nd.

In this function, a counter variable h is instantiated to 0 at the start. This would increment each time a 2-pair of integers is found. This was added because whenever an array passed had 0 pairs, nothing would be printed out on the screen. In this manner, the user would be informed at the end the amount of 2-pairs found, even when none are found.

Implemented also in this function are 4 nested for loops which loop through each individual element in the list. At the core of these for loops is an if statement which actually identifies the 2-pairs of integers while at the same time making sure that arr[i] ≠ arr[j] ≠ arr[k] ≠ arr[l]. If any are found, these are printed out and the counter variable is incremented by 1 each time.

Finally, the user is informed of the total repeated pairs found at the end of the function and the program finishes execution.

One evident bug that showed up when testing this program was the fact that a 2-pair found is usually printed out 8 times. Once in the ‘a X b = c X d’ formation and another time in a swapped commutative form ‘a X b = d X c’. This process is then repeated for a and b aswell. Then, the program usually considers another case in which ‘a’ and ‘b’ are swapped with ‘c’ and ‘d’. This eventually leads to another 4 cases due to the commutative swapping already explained. This meant that the total amount of 2-pairs found was actually 8 times the actual amount, if these irrelevant pairs were omitted. So for 3 real 2-pairs found, the program would register 24 instead, by swapping the first and last 2 elements and also by swapping elements ‘a’ and ‘b’ with ‘c’ and ‘d’. Given the time allotted for completing this assignment, this was considered as a minor bug and was not further investigated to be fixed.

Apart from that, task 4 was successfully implemented and fit all the other specified requirements.

Task 4 Code:

```
# importing the module 'random' to be able to generate random numbers
import random


# A function which validates the input entered by the user such that the user is informed
# and requested to re-enter a value if characters or a number less than 4 is inputted.
# This function has no parameters passed to it but returns an integer value
def user_input():

    # A while loop which runs indefinitely
    while True:

        # Section of code which handles exceptions
        try:

            # Converts to the integer data type and stores the value entered by the user
            x = int(input("\nEnter the max amount of integers in the list: "))

            # Checks if the value entered fits in a particular range before proceeding
            assert(x >= 4)

            # If all checks are successful, the value entered is returned
            return x

        # Informs the user if anything other than an integer is inputted
        except ValueError:
            print("Invalid input!")

        # Informs the user that a particular range of numbers is valid only
        except AssertionError:
            print("Number must be greater than or equal to 4")


# A function which simply populates an array based on the 2 parameters passed.
# The first parameter is the size of the array entered by the user and the second
# is the actual array to be populated. Nothing is returned back
def populate_array(x, arr):

    # A for loop which loops through each individual empty element in the array
    # and populates it
    for i in range(x):

        # Generates a random number from 1-1024 each time and stores it in the array
        arr.append(random.randint(1, 1024))


```

```
# A function which determines all possible 2-pairs of integers having the same product
# The first parameter is the size of the array entered by the user and the second
# is the actual array to be populated. Nothing is returned back
def pairs(x, arr):

    # A counter variable which is initialised to 0
    h = 0

    # 4 for loops which loop through all the numbers in the list for 4 times, each time
    # searching for all possible 2-pair configurations
    # Any configurations found are printed out and the counter variable 'h' is incremented
    for i in range(x):
        for j in range(x):
            for k in range(x):
                for l in range(x):

                    # An if statement which checks for all possible 2-pairs of integers having the same
                    # product while ensuring that in the pairs found, arr[i] ≠ arr[j] ≠ arr[k] ≠ arr[l]
                    if ((arr[i] * arr[j] == arr[k] * arr[l]) and (arr[i] != arr[j]) and (arr[i] != arr[k]) and
                            (arr[i] != arr[l]) and (arr[j] != arr[k]) and (arr[j] != arr[l]) and (arr[k] != arr[l])):
                        print(arr[i], "X", arr[j], "=", arr[k], "X", arr[l])
                        h += 1

    # The user is informed of the amount of 2-pairs found
    print("\nA total of", h, "2-pairs of integers were found.")


```

```
# An array is initialised as empty
numbers = []

# Placeholder 'a' stores the value entered by the user after validation
a = user_input()

# The size entered by the user and stored in placeholder 'a' together
# with the newly created array are passed as parameters to the function
# populate_array() which then fills the list with random integers
populate_array(a, numbers)

# The list is then printed out
print("Generated List:\n", numbers)

# The function pairs() is then called with the appropriate parameters passed
pairs(a, numbers)
```

Task 4 Testing:

The user is first asked to enter a length for the list. Suppose the user enters 4 incorrect values followed by a valid one.

```
Enter the max amount of integers in the list: dfksjdfnjk
Invalid input!

Enter the max amount of integers in the list: -30
Number must be greater than or equal to 4

Enter the max amount of integers in the list: 30.5
Invalid input!

Enter the max amount of integers in the list: 3
Number must be greater than or equal to 4

Enter the max amount of integers in the list: 30
```

Upon entering 30, the list created is populated with 30 random elements from 1 to 1024. This list is then displayed to the user and checks for all possible 2-pairs of integers are performed. The result of the checks performed are shown below:

```
Enter the max amount of integers in the list: 30
Generated List:
 [255, 904, 451, 307, 699, 701, 251, 1010, 594, 918, 581, 282, 692, 692, 486, 427, 286, 490, 732, 715, 943, 985, 912, 819, 728, 283, 227, 486, 485, 939]

A total of 0 2-pairs of integers were found.

Process finished with exit code 0
```

In this case, 0 2-pairs of integers were found in this randomly generated list containing 30 elements. Consider now generating another list of 30 random elements.

```
Enter the max amount of integers in the list: 30
Generated List:
 [196, 662, 822, 43, 101, 372, 275, 830, 463, 850, 72, 773, 880, 395, 237, 728, 636, 961, 683, 647, 885, 626, 199, 207, 225, 509, 272, 871, 808, 170]
850 X 72 = 225 X 272
850 X 72 = 272 X 225
72 X 850 = 225 X 272
72 X 850 = 272 X 225
225 X 272 = 850 X 72
225 X 272 = 72 X 850
272 X 225 = 850 X 72
272 X 225 = 72 X 850

A total of 8 2-pairs of integers were found.
```

This time round, the program claims to have found 8 2-pairs of integers having the same product. However, as already discussed, the actual amount of distinct 2 pairs found is 1 because of a bug. From the printed 2-pairs, one can notice that the 7 other copies are the same as the first case.

Task 5 Explanation:

In task 5, a program that uses an ADT Stack to evaluate arithmetic expressions in RPN format was created. It contains the following arithmetic operations: +, -, x, and /.

A stack class was created which defined basic stack operations. This included the actual list being implemented as a stack defined in the constructor function, a push function, a pop function and a function which returns the string representation of the elements in the stack.

Next, the 'operator' module was imported to be able to use operator functions (operator.add, operator.sub, operator.mul & operator.truediv). These were placed in a dictionary and the operator symbols were assigned to them.

Moving on, an 'rpn()' function was defined to be able compute mathematical functions in rpn notation. In this function, a stack object called 'storage' was created. An array elements was initialized to store the separate formula elements. These were then traversed one after the other, pushing the integers onto the stack each time one is encountered. If an operator is encountered, the 2 last pushed elements are popped off and the expression is evaluated, pushing the answer back onto the stack.

The user is informed at every step of the process of what is going on. When all the expression is evaluated, the answer is displayed to the user.

Task 5 was successfully implemented and fit all the specified requirements without having any bugs.

Task 5 Code:

```
# This module is imported to be able to make use of specific
# operator functions listed in the operators dictionary below
import operator


# A class which defines basic stack operations including an
# additional __str__() function to be able to print out the stack
class Stack:

    # A constructor function which defines the stack being used
    def __init__(self):
        self.items = []

    # A function which allows items to be pushed onto the stack
    def push(self, info):
        self.items.append(info)

    # A function which returns the string representation of the
    # object created
    def __str__(self):
        return str(self.items)


# An 'operators' dictionary which associates actual operators to
# binary operator functions imported from the operator module
operators = {"+":  operator.add,
             "-":  operator.sub,
             "*":  operator.mul,
             "/":  operator.truediv}
```

```
# The function which does the evaluation of the RPN arithmetic
# It takes in a formula as the only argument and returns back a
# popped variable. This is a recursive function
def rpn(formula):

    # An instance 'storage' of the 'Stack()' class is created
    # to be able to make use of 'Stack()' functions
    storage = Stack()

    # The formula to be evaluated is split up
    elements = formula.split()

    # A for loop which loops through the split up elements
    for element in elements:

        # The arithmetic operator going to affect any values on the stack
        print("\nCurrent element:\t\t", element)

        # The stack contents are displayed before the operation
        print("Stack contents BEFORE:", storage)

        # If an element present is found in the operators dictionary,
        # '+'/'-'/'*'/'/' is performed on the last 2 elements pushed
        # onto the stack. These are then popped off the stack, evaluated
        # and the result is then popped back onto the stack
        if element in operators:

            # The last 2 elements present in the stack are popped
            # and stored in variables
            num2 = storage.pop()
            num1 = storage.pop()

            # The result is then computed
            result = operators[element](num1, num2)

            # the result is then pushed back onto the stack
            storage.push(result)
        else:

            # else, the element found is an integer and not an operator,
            # so it is pushed onto the stack
            storage.push(int(element))

        # The stack contents are displayed after the operation
        print("Stack contents AFTER: ", storage)

    # The last value on the stack is popped off at the end
    return storage.pop()


# Driver code which tests the rpn calculator
print("\n\n\nThe final answer is :", rpn("15 6 8 * 2 + +"))
```

Task 5 Testing:

The following expression was to be evaluated: (6 x 8) + (15 + 2)

This was converted into RPN form and included in the driver code of task 5.

```
86      # Driver code which tests the rpn calculator
87      print("\n\n\nThe final answer is :", rpn("15 6 8 * 2 + +"))
```

When the user runs the program, the expression is evaluated and onscreen updates are given to the user to inform him of the calculation's current status.

```
Current element:         15
Stack contents BEFORE: []
Stack contents AFTER:  [15]

Current element:         6
Stack contents BEFORE: [15]
Stack contents AFTER:  [15, 6]

Current element:         8
Stack contents BEFORE: [15, 6]
Stack contents AFTER:  [15, 6, 8]

Current element:         *
Stack contents BEFORE: [15, 6, 8]
Stack contents AFTER:  [15, 48]

Current element:         2
Stack contents BEFORE: [15, 48]
Stack contents AFTER:  [15, 48, 2]

Current element:         +
Stack contents BEFORE: [15, 48, 2]
Stack contents AFTER:  [15, 50]

Current element:         +
Stack contents BEFORE: [15, 50]
Stack contents AFTER:  [65]
```

The final answer is displayed in the end:

```
The final answer is : 65
```

Task 6 Explanation:

In this task, a program was written which check if a given number is prime. Following this, the Sieve of Eratosthenes Algorithm was implemented.

A function was created to determine if an entered number was prime. In this function, the number being tested was passed as the only parameter and an if statement was implemented to check if the number was 1. If so, False is returned as 1 is not prime. If not, the else branch contains a for loop which loops from 2 up to the number, checking if the remainder of anything divided by the number gives a 0. If so a direct divisor and the number is not prime, thus returning false once again. But if all the numbers are traversed and not a single direct divisor is found, then True is returned because the number is prime.

The Sieve of Eratosthenes Algorithm was implemented next. This function consisted of an empty list which would be filled with numbers from 2 up till the number being entered by the user to be tested as the last element. Then, a for loop which traverses from 2 up till the square root of the number entered, loops through all the numbers while eliminating any non-prime numbers and their factors from the list. The remaining numbers in the list (ie prime factors) are then shown to the user.

An optimization made to the Sieve of Eratosthenes function was that when checking for non-prime numbers to be removed from the list, the list was traversed from 2 up till the square root of the number entered rather than looping up till the number entered. This was done because if a number found is not prime, then all it's factors are not prime by default. Thus, the loop went on till the square root of the number entered and to compensate for this change, all remaining factors of numbers removed from the list were also removed rather than having to traverse all the list and simply removing a tested number.

Task 6 was successfully implemented and fit all the specified requirements without having any bugs.

Task 6 Code:

```
# Importing the module 'math' to be able to use mathematical functions
import math

# A function which checks if a number is prime and returns True or False
# The only parameter is the actual number being checked
def check_if_prime(num):

    # This section of code checks if the number is one, if not divides
    # the number each time by a set of numbers starting from 2 until a
    # the remainder from a division is found to be 0
    # If neither checks are true, then number is prime and user is informed
    if num == 1:
        return False
    else:
        for i in range(2, num):
            if (num % i) == 0:
                return False
        return True


# A function which implements the sieve of eratosthenes algorithm
# The number up till which the algorithm will check for prime numbers
# is passed as a parameter and the list of prime numbers evaluated
# from this algorithm is returned back as a list
def sieve_of_eratosthenes(num):

    # Initialisation of empty list
    table = []

    # All the numbers starting from the first prime up till the final
    # number to be checked are appended to the list
    for n in range(2, num+1):
        table.append(n)

    # The limit for the next loop is set as the square root of the
    # last integer to be evaluated
    limit = int(math.sqrt(num)) + 1

    # This section iterates through all the numbers in the table list
    # and if an integer present there is found to not be prime, it is
    # removed along with all of its factors
    for n in range(2, limit):
        if n in table:
            for m in range(n+n, num+1, n):
                if m in table:
                    table.remove(m)
        n += 1

    # The remaining numbers in the table list are returned
    return table
```

```
51
52
53      print("This program will check if a number is prime.")
54
55      # The number to be checked if prime is entered, converted to an int and stored
56      check = int(input("Please enter the number to be checked: \n"))
57
58      # The check_if_prime() function is called and a 'True' or 'False" are returned
59      print(check_if_prime(check))
60
61      print("\n-----------------------------------------------------\n")
62
63      print("Now, the Sieve of Eratosthenes Algorithm will be implemented.")
64
65      # The last number to be evaluated in the sieve algorithm is inputted,
66      # converted to an int and stored
67      last = int(input("Enter the last number to be checked if it is prime: "))
68
69      print("\nThe following numbers are all prime: ")
70
71      # The sieve_of_eratosthenes() function is called and a list is returned
72      print(sieve_of_eratosthenes(last))
73
```

Task 6 Testing:

When running task 6, the user is asked to enter a number to be tested if it is prime. Consider testing 2 non-prime numbers and 2 prime numbers

```
This program will check if a number is prime.
Please enter the number to be checked:
8
False
```

```
Please enter the number to be checked:
100
False
```

```
Please enter the number to be checked:
17
True
```

```
Please enter the number to be checked:
31
True
```

Moving on, the user is now asked to enter the last number to be evaluated in the Sieve of Eratosthenes Algorithm. If 100 is entered, the following output is given:

```
Now, the Sieve of Eratosthenes Algorithm will be implemented.
Enter the last number to be checked if it is prime: 100

The following numbers are all prime:
[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]

Process finished with exit code 0
```

If 50 is now entered:

```
Now, the Sieve of Eratosthenes Algorithm will be implemented.
Enter the last number to be checked if it is prime: 50

The following numbers are all prime:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Task 7 Explanation:

In this task, a Binary Search Tree was implemented. This was done with the help of 2 classes, a ‘Node’ class and a ‘Tree’ class.

In the ‘Node’ class, a constructor function was initialized which specifies all the possible nodes a binary search tree can have. These are the root node, left node and right node.

Moving on, the next function is the ‘inserting()’ function which is a function called from the Tree class to be able to insert an integer in the correct position in the BST. This function checks if the value being entered is less than the root and if so proceeds to checking if a left node exists, if so this function is called recursively but if not, the value being tested is set as the left child. However, if the value entered is greater than the root, another check is performed to check if a right node exists. If so, the function is called recursively, but if not, the value is set as the right child. If the value being tested is neither greater nor smaller than the value being compared too, then the user is informed that the value being entered is already present in the BST.

The last function implemented in this class simply prints out the elements stored in the BST using inorder traversal. This method was chosen over the pre and post order traversals as it is the easiest to understand and follow of the three.

In the Tree class, a constructor function is initialized which sets the root node to None at the start.

The next function called ‘insert()’ checks if the root node exists. If so, the ‘inserting()’ function from the Node class is called to be able to insert this node in the appropriate position. If not, then it means that there is no root node and so, the value entered is set as the root node.

The final function in the Tree class is the ‘print_tree()’ function. This simply calls the ‘in_order()’ function from the Node class.

Task 7 was successfully implemented and fit all the specified requirements without having any bugs.

Task 7 Code:

```
# A class Node for the node objects to be stored in the tree
# This consists of 3 functions: a constructor, an inserting()
# function and a in_order() function
class Node:

    # A constructor taking 2 parameters which are 'self' and also
    # 'num" which is set to 'None' by default
    # All possible nodes in a BST are enlisted
    def __init__(self, num=None):
        self.num = num
        self.left = None
        self.right = None

    # A recursive function which is originally called from the Tree class
    # This simply checks if a value to be entered is less than the root
    # node and if so, calls the function recursively if there already
    # exists a left node, but if not, simply creates one and assigns
    # that value to it. If this value happens to be greater than the root
    # node, a similar procedure is run for the right node instead
    # If the value is neither larger nor smaller than a particular node,
    # then the user is informed that the value already exists in the BST
    def inserting(self, val):
        if self.num > val:
            if self.left:
                self.left.inserting(val)
                print()
            else:
                self.left = Node(val)
                print(val, "is set as a left child")
        elif self.num < val:
            if self.right:
                self.right.inserting(val)
            else:
                self.right = Node(val)
                print(val, "is set as a right child")
        else:
            print("Value already exists in the Binary Search Tree")

    # A recursive function which is originally called from the Tree
    # class. This simply prints out each individual BST number in
    # inorder traversal
    def in_order(self):
        if self:
            if self.left:
                self.left.in_order()
            print(str(self.num))
            if self.right:
                self.right.in_order()


```

```
# A class Tree which can be explained as the main interface for
# the user. This class then makes use of the Node class
class Tree:

    # A default constructor which initialises the root node to None
    def __init__(self):
        self.root = None

    # A function which checks if the Root node exists. If so, the
    # inserting() function from the class Node is called and if
    # not, the value entered is initialised as the Root Node
    def insert(self, val):
        print("\nInserting", val, "into the BST")
        if self.root:
            self.root.inserting(val)
        else:
            self.root = Node(val)
            print(val, "is the Root Node")

    # A function which calls the in_order() function from the Node
    # class to be able to print the elements in the BST
    def print_tree(self):
        print("\nPrinting the existing Binary Search Tree using IN ORDER Traversal:")
        self.root.in_order()


print("\t\tWelcome to this program which implements a BINARY SEARCH TREE")
print("-------------------------------------------------------------------------------")

# Creating an instance 'values' of the object 'Tree' to be able to make use of
# certain functions
values = Tree()

# Inserting values into the newly created Binary Search Tree
values.insert(5)
values.insert(9)
values.insert(2)
values.insert(15)
values.insert(4)
values.insert(1)
values.insert(6)
values.insert(15)

print("\nAll values were added to the Binary Search Tree (except for repeated ones)")
print("-------------------------------------------------------------------------------")

# Printing the values using in order traversal
values.print_tree()
```

Task 7 Testing:

To test this program, the following code was inserted:

```
# Creating an instance 'values' of the object 'Tree' to be able to make use of
# certain functions
values = Tree()

# Inserting values into the newly created Binary Search Tree
values.insert(5)
values.insert(9)
values.insert(2)
values.insert(15)
values.insert(4)
values.insert(1)
values.insert(6)
values.insert(15)

print("\nAll values were added to the Binary Search Tree (except for repeated ones)")
print("--------------------------------------------------------------------------------")

# Printing the values using in order traversal
values.print_tree()
```

First, an instance of the object Tree was created to be able to make use of certain functions available in this class. Next, some values were inserted into the binary search tree. Note that the number ‘15’ was inserted twice to test the BST. Finally, the ‘print_tree()’ function was called to print out the tree using the in order traversal method. The program was then compiled and the following output was displayed to inform the user on what is going on:

```
        Welcome to this program which implements a BINARY SEARCH TREE
--------------------------------------------------------------------------------

Inserting 5 into the BST
5 is the Root Node

Inserting 9 into the BST
9 is set as a right child

Inserting 2 into the BST
2 is set as a left child

Inserting 15 into the BST
15 is set as a right child

Inserting 4 into the BST
4 is set as a right child


Inserting 1 into the BST
1 is set as a left child


Inserting 6 into the BST
6 is set as a left child

Inserting 15 into the BST
Value already exists in the Binary Search Tree
```

As expected, the unique values were added to the tree. When trying to add 15 a second time, the user is informed that it already exists in the tree and so this value is ignored. Next, a message is printed out to inform the user that the values have been successfully added and finally, the contents of the tree are printed out.

```
All values were added to the Binary Search Tree (except for repeated ones)
--------------------------------------------------------------------------

Printing the existing Binary Search Tree using IN ORDER Traversal:
1
2
4
5
6
9
15

Process finished with exit code 0
```

Task 8 Explanation:

In task 8, a program that finds an approximation to the square root of a given number n using the Newton-Raphson method was written.

In this task, a ‘user_input()’ function was written to be able to validate and store input entered by the user. It was decided that the user had to enter 3 different values to be able to work out a square root approximation. These were:

1) The number n whose square root was to be found
2) An approximate answer that the user thinks is correct
3) The number of approximations required to reach the desired answer.

These were then inserted in a try-except statement in this function which made sure that values entered for 1 and 2 were float and the value entered for 3 was an integer. It also made sure that the value entered for 1 was greater or equal to 0.

Moving on, a function for f(x) and another for its derivative were written. These were then called from the next function called ‘next_approx()’ which calculated the next approximation for the square root of the number n. This function consisted of the formula used for the Newton Raphson method.

Finally, a for loop was implemented at the end to call the ‘next_approx()’ function for so many times based on the previous user input, each time obtaining a more accurate approximation of the square root function.

Task 8 was successfully implemented and fit all the specified requirements without having any bugs.

Task 8 Code:

```
# A function which validates the input entered by the user such that the user is informed
# and requested to re-enter a value if characters or numbers less than 0 are inputted.
# This function has no parameters passed to it but returns 2 float and 1 int values
def user_input():

    # A while loop which runs indefinitely
    while True:

        # Section of code which handles exceptions
        try:

            # Converts to the float data type and stores the value entered by the user
            a = float(input("\nEnter a value to approximate its square root: "))

            # Checks if the value entered fits in a particular range before proceeding
            assert (a >= 0)

            # Converts to the float data type and stores the value entered by the user
            b = float(input("\nEnter an approximate answer you think is correct: "))

            # Converts to the integer data type and stores the value entered by the user
            c = int(input("\nEnter the number of approximations required: "))

            # If all checks are successful, the values entered are returned
            return a, b, c

        # Informs the user if anything other than a number is inputted
        except ValueError:
            print("Invalid input!")

        # Informs the user that a particular range of numbers is valid only
        except AssertionError:
            print("Number must be greater than or equal to 0")


# A function taking only 1 parameter which substitutes the values of 'a' and 'n'
# based on what the user entered and returns a float value back
def function(a):
    f1 = (a ** 2) - n
    return f1


# A function which is simply the derivative of the above function
# It takes the same parameter as the previous one and also returns a float
def derivative(a):
    f2 = 2 * a
    return f2
```

```
48
49
50      # The function which actually computes a Newton Raphson Approximation
51      # based on values returned from the 'function()' and 'derivative()' methods
52      # This also has the parameter 'a' passed in and returns a float value
53      def next_approx(a):
54          a = a - (function(a) / derivative(a))
55          return a
56
57
58      print("This program will find the approximation to the square root of a given number 'n'.")
59
60      # The inputs entered by the user are validated in the user_input() function
61      # These are then returned and stored in the variables a, b & c respectively
62      n, x, m = user_input()
63
64      # The Newton-Raphson Method is iterated for 'm' times and
65      # the user is informed on the updated value each time
66      for i in range(m):
67          x = next_approx(x)
68          print("\nAfter approximation #", i+1, "the answer is:", x)
```

Task 8 Testing:

When running this task, the user was asked to enter the 3 values discussed. Consider the user entered an invalid number on the first attempt followed by 3 valid ones. The output is shown below:

```
This program will find the approximation to the square root of a given number 'n'.

Enter a value to approximate its square root: -2
Number must be greater than or equal to 0

Enter a value to approximate its square root: 2

Enter an approximate answer you think is correct: 3

Enter the number of approximations required: 6
```

The program then computes the answers based on the values entered and the user is informed at every step of the way as shown:

```
After approximation # 1 the answer is: 1.8333333333333333

After approximation # 2 the answer is: 1.4621212121212122

After approximation # 3 the answer is: 1.4149984298948028

After approximation # 4 the answer is: 1.4142137800471977

After approximation # 5 the answer is: 1.4142135623731118

After approximation # 6 the answer is: 1.414213562373095

Process finished with exit code 0
```

These values were compared to the values on a calculator and were found to be more accurate as the program computes more decimal points.

Task 9 Explanation:

In this task, a program was written to identify all the numbers repeated more than once in a given array of integers. This was done in the fastest, most practical and memory efficient way as possible.

To start off, a ‘user_input()’ function identical to the one in task 3 was coded. This asked the user to enter the maximum amount of integers he/she wanted in a list to be created, followed by the possible min and max values present in the list. These values were validated in this function to ensure that the user enters correct information.

The next function to be created was the ‘generate_list()’ function which is also similar to that created in task 3. It simply populates an empty list initialized at the start of the program based on what the user entered. The only difference is that these numbers generated are returned back and eventually printed out to the user.

The overall most efficient sorting algorithm out of all the sorting algorithms existing is the quick sort algorithm. This was implemented just like in task 1 and the generated numbers were then sorted through this function.

Finally a ‘repeat()’ function was created which identified the repeated numbers and printed them out to the user. This was done by first creating an array ‘repeated’ which consisted of only 1 element at the start (“REPEATED NUMBERS: “). This array was created to store any repeated elements found in the tested list. The reason why this string of text was inserted in the ‘repeated’ list will be explained shortly.

Next, a for loop was implemented which appended any repeated numbers to the list. To avoid having more than one instance of the same number show up in the list ‘repeated’, the code was optimized in the following manner:

A number in numbers[i] would be added to the list if and only if the number in numbers[i+1] is the same & the number to be added is not equal to the last number added to the ‘repeated’ list. In this manner, only one instance of a repeated number would be present in the ‘repeated’ list.

The reason why strings were added at the start of the function to the ‘repeated’ list was so that the if statement could work at the very start, when there were still no numbers in the ‘repeated’ list. When checking for repeated numbers, the if statement also ensured that no numbers identical to the one going to be added were available in ‘repeated[-1]’. This statement always worked when the list was not empty as last element in the ‘repeated’ list could be accessed. However, if the list was initialized to empty, the program would give an error and not run because there would be nothing else to compare to. Thus, adding this string of characters made the if statement valid and the program work.

Task 9 was successfully implemented and fit all the specified requirements without having any bugs.

Task 9 Code:

```
# importing the module 'random' to be able to generate random numbers
import random


# A function which validates the input entered by the user such that the user is informed
# and requested to re-enter a value if characters are inputted instead of numbers.
# This function has no parameters passed to it but returns 3 integer values
def user_input():

    # A counter variable is initialised to 1
    i = 1

    # A while loop which runs indefinitely
    while True:

        # Section of code which handles exceptions
        try:

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 2
            if i is 1:
                x = int(input("\nEnter the max amount of integers in the list: "))
                i += 1
                continue

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 3
            elif i is 2:
                y = int(input("\nEnter the min value an integer present in the list can have: "))
                i += 1
                continue

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 4
            elif i is 3:
                z = int(input("\nEnter the max value an integer present in the list can have: "))
                i += 1
                continue

            # The user is informed in this part that all inputs were correct and these
            # values are then returned back
            elif i is 4:
                print("All inputs were successfully stored and the list is shown below:\n")
                return x, y, z

        # Informs the user if anything other than an integer is inputted
        except ValueError:
            print("Invalid input!")


```

```
# A function which based on the 3 parameters passed (size of list, min value in list and
# max value in list) which were previously entered by the user, generates a list of randoms
# These numbers are then stored in an array which is passed as the final parameter and
# returned back at the end of the function
def generate_list(x, y, z, numbers):

    # A for loop which loops through each individual empty element in the array
    # and populates it
    for i in range(x):

        # Generates a random number from y-z each time and stores it in the array
        numbers.append(random.randint(y, z))

    # The generated list is returned
    return numbers


# The main function for a quick sort which also calls the 'partition()' function.
# This is a recursive function taking 3 parameters which are the array to be sorted,
# the starting position and ending position for the sorting procedure.
# This function does not return anything.
def quick_sort(array, start, end):

    # Statement runs if the end of the list is not yet reached
    if start < end:

        # The value returned from the sub-function being called is stored in
        # the placeholder split
        split = partition(array, start, end)

        # The recursive part of the function which proceeds to sorting 2 smaller
        # sections of the list separated by the split variable
        quick_sort(array, start, split-1)
        quick_sort(array, split+1, end)


# The sub-function of the quick sort which switches different numbers in the
# list to be sorted based on the pin. This takes 3 parameters:
# the array to be sorted, the starting and ending point.
# This function returns the value stored in i which determines the split
# for the next iteration
def partition(array, start, end):

    # Sets the pin to be the one approximately in the middle of the list
    pinPosition = (start + end) // 2
    pinValue = array[pinPosition]

    # Swaps the pin with the last element in the list
    array[pinPosition], array[end] = array[end], array[pinPosition]

    # Sets the counter i to 1 less than the first position of the list
    i = start - 1

```

```
    # Loops through the list from start to end excluding the pin
    for j in range(start, end):

        # If the pin is greater than an element being compared, the
        # counter variable i is incremented and the values stored in
        # positions i and j are swapped
        if array[j] <= pinValue:
            i += 1
            array[j], array[i] = array[i], array[j]

    # i is incremented
    i += 1

    # The pin is now swapped back in it's appropriate position in the list
    array[i], array[end] = array[end], array[i]

    # The counter value i is returned
    return i


# A function which takes a sorted list (1st parameter passed) and determines the
# repeated integers. These are then added to a new list which is then displayed
# The second parameter simply defines the size of the array being passed
# Nothing is returned back
def repeat(numbers, x):

    # The array used to store the repeated numbers
    repeated = ["REPEATED NUMBERS: "]

    # The block of code which determines the repeated numbers and stores them
    # in the new list entitled 'repeated' if and only if they haven't been
    # stored there already
    for i in range(x-1):

        if numbers[i] == numbers[i+1] and (numbers[i] != repeated[-1]):
            repeated.append(numbers[i])

    # The list is the displayed to the user
    print("\n", repeated)


print("Welcome to this program which selects repeated numbers from a list of randomly generated integers")

# The input entered by the user for the size, min and max values of the list
# are validated in the user_input() function. These are then
# returned and stored in the variables a, b & c respectively
a, b, c = user_input()

# An empty array is initialised with dimensions as specified by the user
arr = []

# The function generate_list() is then called and the placeholders a, b, c are
# passed as parameters to this function together with the array to be filled
print("Generated List: ", generate_list(a, b, c, arr))

# The quick_sort() function is called which sorts the list
quick_sort(arr, 0, a-1)
print("\nSorted List: ", arr)

# Finally the repeat() function is called to determine the repeated integers
repeat(arr, a)
```

Task 9 Testing:

Upon hitting run, the user is asked to enter 3 values for the list. Consider the user entered an invalid followed by a valid input each time.

```
Welcome to this program which selects repeated numbers from a list of randomly generated integers

Enter the max amount of integers in the list: dkfvjn
Invalid input!

Enter the max amount of integers in the list: 30

Enter the min value an integer present in the list can have: one
Invalid input!

Enter the min value an integer present in the list can have: 1

Enter the max value an integer present in the list can have: 50y
Invalid input!

Enter the max value an integer present in the list can have: 50
All inputs were successfully stored and the list is shown below:
```

The program then creates the list, sorts it and displays the repeated values to the user as shown.

```
All inputs were successfully stored and the list is shown below:

Generated List:  [37, 6, 31, 18, 29, 37, 10, 26, 38, 19, 44, 29, 21, 34, 1, 11, 34, 9, 31, 49, 4, 33, 24, 2, 20, 45, 25, 35, 4, 38]

Sorted List:  [1, 2, 4, 4, 6, 9, 10, 11, 18, 19, 20, 21, 24, 25, 26, 29, 29, 31, 31, 33, 34, 34, 35, 37, 37, 38, 38, 44, 45, 49]

 ['REPEATED NUMBERS: ', 4, 29, 31, 34, 37, 38]

Process finished with exit code 0
```

In the case above, it can be seen that the values: 4, 29, 31, 34, 37 and 38 were repeated twice. Consider now a case in which numbers appear repeated more than just twice. The user input for this case is shown below:

```
Welcome to this program which selects repeated numbers from a list of randomly generated integers

Enter the max amount of integers in the list: 20

Enter the min value an integer present in the list can have: 10

Enter the max value an integer present in the list can have: 15
```

This meant that a list of length 20 with values between 10-15 could be found in the generated list. This forced the creation of a list with a lot of repeated values. In fact, all the numbers from 10-15 were appended to the ‘repeated’ list and displayed to the user as shown:

```
All inputs were successfully stored and the list is shown below:

Generated List:  [12, 12, 15, 15, 10, 10, 12, 15, 15, 13, 14, 13, 11, 11, 10, 14, 15, 12, 14, 10]

Sorted List:  [10, 10, 10, 10, 11, 11, 12, 12, 12, 12, 13, 13, 14, 14, 14, 15, 15, 15, 15, 15]

 ['REPEATED NUMBERS: ', 10, 11, 12, 13, 14, 15]
```

As seen, 10 was present 4 times in the list but was only appended once to the ‘repeated’ list due to the optimized if statement.

Task 10 Explanation:

In this task, a function was written to recursively call itself to find the largest number in a given list of integers.

Starting off, a ‘user_input()’ function was written, identical to that of task 9, were a user is asked to enter the length of a list to be generated, followed by the min and max values possibly present in the list. This functions aim is to validate this input and ensure that the user enters integer values.

Next, based on the entered values, a function ‘generate_list()’ is called and based on the values entered, a created list is populated. This function is once again identical to that created for the previous task. The numbers are then returned at the end of the function.

The last function present is the ‘find_max()’ function itself. This function consists of a for loop which checks if there is only 1 number in the list and if so, returns that number as the max. If not, the max of a number present in the list and a recursive call to the same function is found.

Task 10 was successfully implemented and fit all the specified requirements without having any bugs.

Task 10 Code:

```
# importing the module 'random' to be able to generate random numbers
import random


# A function which validates the input entered by the user such that the user is informed
# and requested to re-enter a value if characters are inputted instead of numbers.
# This function has no parameters passed to it but returns 3 integer values
def user_input():

    # A counter variable is initialised to 1
    i = 1

    # A while loop which runs indefinitely
    while True:

        # Section of code which handles exceptions
        try:

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 2
            if i is 1:
                x = int(input("\nEnter the max amount of integers in the list: "))
                i += 1
                continue

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 3
            elif i is 2:
                y = int(input("\nEnter the min value an integer present in the list can have: "))
                i += 1
                continue

            # This section of code runs again each time until correct input is entered by the
            # user and accepted by the program thus increasing the value of i to 4
            elif i is 3:
                z = int(input("\nEnter the max value an integer present in the list can have: "))
                i += 1
                continue

            # The user is informed in this part that all inputs were correct and these
            # values are then returned back
            elif i is 4:
                print("All inputs were successfully stored and the list is shown below:\n")
                return x, y, z

        # Informs the user if anything other than an integer is inputted
        except ValueError:
            print("Invalid input!")
```

```
51  # A function which based on the 3 parameters passed (size of list, min value in list and
52  # max value in list) which were previously entered by the user, generates a list of randoms
53  # These numbers are then returned
54  def generate_list(x, y, z):
55
56      # An empty list to store the random numbers
57      numbers = []
58
59      # A for loop which loops through each individual empty element in the array
60      # and populates it
61      for i in range(x):
62
63          # Generates a random number from y-z each time and stores it in the array
64          numbers.append(random.randint(y, z))
65
66      # The numbers are then returned
67      return numbers
68
69
```

```
70  # A recursive function which finds the largest number in the list passed as a
71  # first parameter and the size of it as the second
72  # The max number found is returned back
73  def find_max(numbers, x):
74
75      # if there is only 1 number in the list then just that number is returned
76      if x == 1:
77          return numbers[0]
78
79      # else the max number of all the numbers in the list is determined by
80      # recursively calling the function each time
81      else:
82          return max(numbers[x-1], find_max(numbers, x-1))
83
84
```

```
85  print("Welcome to this program which finds the largest number from a list of randomly generated integers")
86
87  # The inputs entered by the user for the size, min and max values of the list
88  # are validated in the user_input() function. These are then
89  # returned and stored in the variables a, b & c respectively
90  a, b, c = user_input()
91
92  # The function generate_list() is then called and the placeholders a, b, c are
93  # passed as parameters to this function
94  # The list generated by this function is returned and stored in the sequence
95  # placeholder
96  sequence = generate_list(a, b, c)
97
98  # This sequence is then printed out
99  print(sequence)
100
101 # The find_max() function is called and the max value found is returned
102 print("\nThe max value found is:", find_max(sequence, a))
```

Task 10 Testing:

When running this program, the user is prompted to enter 3 values at the start. Consider the user enters an invalid value followed by a valid one each time.

```
Welcome to this program which finds the largest number from a list of randomly generated integers

Enter the max amount of integers in the list: asdj
Invalid input!

Enter the max amount of integers in the list: 20

Enter the min value an integer present in the list can have: two
Invalid input!

Enter the min value an integer present in the list can have: 2

Enter the max value an integer present in the list can have: one hundred
Invalid input!

Enter the max value an integer present in the list can have: 100
```

Next, the list is generated and displayed to the user. Following this, the max value in the list is found and displayed.

```
All inputs were successfully stored and the list is shown below:

[78, 97, 24, 35, 27, 40, 49, 84, 98, 32, 10, 6, 37, 91, 37, 13, 32, 30, 9, 11]

The max value found is: 98

Process finished with exit code 0
```

As can be seen above, the function worked as expected and 98 was printed out as the max value found. Consider now the case in which a list having repeated elements is traversed to find its max value. The parameters entered for this list are the following:

```
Welcome to this program which finds the largest number from a list of randomly generated integers

Enter the max amount of integers in the list: 30

Enter the min value an integer present in the list can have: 0

Enter the max value an integer present in the list can have: 5
```

Given a length of 30 in which numbers from 0 to 5 can be inputted into the list, the numbers generated are forced to be repeated. But as expected, no errors were given and the program still worked as expected returning 5 as the max number present in the list.

```
All inputs were successfully stored and the list is shown below:

[3, 2, 4, 5, 5, 2, 1, 3, 5, 0, 0, 2, 5, 0, 2, 5, 4, 3, 0, 5, 4, 1, 5, 5, 5, 4, 0, 1, 2, 4]

The max value found is: 5
```

Task 11 Explanation:

In this program, functions for the sine and cosine series expansions were written and tested out.

To start out, 2 variables were created to store input from the user. The first one stored the value given to the angle and the second, the position of the last term of the expansion the user wants to compute.

Next a cosine function was created which has these 2 variables passed as parameters. In this function, an answer variable is declared to 0 at the start. Following this, a for loop iterates based on the user input and the mathematical representation of the cosine function is evaluated, each time storing the result in the answer variable. When looping finishes, the answer is returned back to the user and displayed.

Two other variables dedicated to a sine function are also created to store a value for the angle and the position of the last term in the sequence to be computed. Once again, another function for sine is created which has the same layout and functionality as the cosine function, with the only difference being the formula involved to compute the result. In this case, the result is also returned and displayed to the user.

Task 11 was successfully implemented and fit all the specified requirements without having any bugs.

Task 11 Code:

```
# Importing the module 'math' to be able to use mathematical functions
import math


# A function which computes the Cosine maclaurin's series expansion
# This takes the angle x as the first parameter and n as the nth term to be computed
# The answer of this expansion is then returned back
def cosine(x, n):

    # This variable is initialised to 0
    answer = 0

    # This section of code iterates through each individual term in the cosine series
    # expansion, each time adding the value obtained to the answer variable
    for i in range(n+1):

        answer += (((-1)**i)*(x**(2*i)))/math.factorial(2*i)

    # The final answer for a given expansion is returned back
    return answer


# A function which computes the Sine maclaurin's series expansion
# This takes the angle x as the first parameter and n as the nth term to be computed
# The answer of this expansion is then returned back
def sine(x, n):

    # This variable is initialised to 0
    answer = 0

    # This section of code iterates through each individual term in the sine series
    # expansion, each time adding the value obtained to the answer variable
    for i in range(n+1):

        answer += (((-1)**i)*(x**((2*i)+1)))/(math.factorial((2*i)+1))

    # The final answer for a given expansion is returned back
    return answer
```

```
41  # Values for the angle and the final term to be computed for the cosine expansion are entered and stored
42  print("This program will calculate the Maclaurin Expansion of cos(x).")
43  value1 = int(input("Enter a value for x in degrees: "))
44  upper1 = int(input("Enter the position of the last term of this expansion you want to compute: "))
45
46  # The cosine() function previously defined is called and the answer is printed on the screen
47  print("The answer of cos(" + str(value1) + ") from 0 to " + str(upper1) + " is:", cosine(value1, upper1))
48  print("\n-----------------------------------------------------\n")
49
50  # Values for the angle and the final term to be computed for the sine expansion are entered and stored
51  print("This program will calculate the Maclaurin Expansion of sin(x).")
52  value2 = int(input("Enter a value for x in degrees: "))
53  upper2 = int(input("Enter the position of the last term of this expansion you want to compute: "))
54
55  # The sine() function previously defined is called and the answer is printed on the screen
56  print("The answer of sin(" + str(value2) + ") from 0 to " + str(upper2) + " is:", sine(value2, upper2))
```

Task 11 Testing:

When running the program, the user is asked for 2 input values. Consider the case in which the user enters 30 followed by 5. The result of computation for the cosine series expansion is displayed:

```
This program will calculate the Maclaurin Expansion of cos(x).
Enter a value for x in degrees: 30
Enter the position of the last term of this expansion you want to compute: 5
The answer of cos(30) from 0 to 5 is: -147430091.85714287
```

Next 2 values for the sine series expansion are required. Consider the user enters 20 followed by 6. The result of computation for the sine series expansion is displayed:

```
This program will calculate the Maclaurin Expansion of sin(x).
Enter a value for x in degrees: 20
Enter the position of the last term of this expansion you want to compute: 6
The answer of sin(20) from 0 to 6 is: 9207216.73741896

Process finished with exit code 0
```

Both results were compared to results on a calculator and were found to be correct.

Task 12 Explanation:

In task 12, the Fibonacci sequence of numbers was implemented iteratively.

The user was asked to input the position of the last element in the sequence he/she wanted printed out. This value was then inserted in a for loop which iterated from 1 up till the number entered. This for loop was constantly printing out the result of the ‘fibonacci()’ function created.

The ‘fibonacci()’ function consisted of 2 variables initialized to 0 and 1 respectively at the start. These variables were then inserted in a for loop which set the value of n1 to the present value of n2 and the new value of n2 to the added value of both variables. Nonetheless, this setting of variables was performed in a single line of code. The final value of n1 was finally returned back and printed to the user.

Task 12 was successfully implemented and fit all the specified requirements without having any bugs.

Task 12 Code:

```
# A function which implements the Fibonacci Sequence using a non-recursive method
# The nth integer of the sequence to be printed out is passed as a parameter and
# the values coming out of this list are returned one after the other
def fibonacci(num):

    # These 2 variables are initialised
    n1 = 0
    n2 = 1

    # A for loop to determine the next integer in the sequence
    for i in range(num):
        n1, n2 = n2, n1 + n2

    # The next integer in the sequence found is returned
    return n1


print("This program will display the Fibonacci Sequence.")

# User input is entered, converted to an int variable and stored in n
n = int(input("Please enter the position of the last integer of the sequence to be printed out: "))


print("\n\n\tThe Fibonacci Sequence up till the integer in position", n)
print("--------------------------------------------------------------------------------")

# A for loop which iterates through the sequence whilst calling the fibonacci() function each time
for count in range(1, n + 1):
    print(fibonacci(count))
```

Task 12 Testing:

The program was run and the user was prompted to enter the position of the last element in the sequence he wanted to compute. Consider the case in which the user enters 5:

```
This program will display the Fibonacci Sequence.
Please enter the position of the last integer of the sequence to be printed out: 5


    The Fibonacci Sequence up till the integer in position 5
--------------------------------------------------------------------------------
1
1
2
3
5

Process finished with exit code 0
```

Consider now the case in which the user enters 20:

```
This program will display the Fibonacci Sequence.
Please enter the position of the last integer of the sequence to be printed out: 20


    The Fibonacci Sequence up till the integer in position 20
--------------------------------------------------------------------------------
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765

Process finished with exit code 0
```

---------- END OF ASSIGNMENT ----------