Computational Thinking and Problem Solving (COMP1002) and Problem Solving Methodology in Information Technology (COMP1001)

Assignment FOUR
(Due at noon on **13 November 2020**)

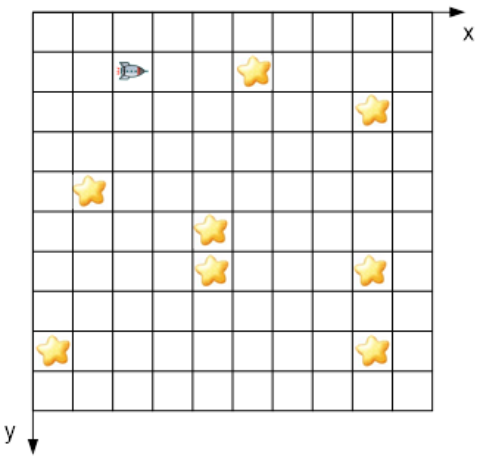
1. [15 marks] The following program implements a sorting algorithm called *merge sort*. The only missing part is the implementation of the **`merge()`** function. Given two lists of data sorted in ascending order. The function returns a list which contains all the data items from the two lists and they are sorted in ascending order. For example, assume that the two lists are [1, 4, 6, 10] and [5, 7, 8, 11, 12]. The list to be returned is [1, 4, 5, 6, 7, 8, 10, 11, 12]. ***Without*** using any functions from built-in or imported modules, other than the function **`len()`**, implement the function **`merge()`**.

```
def merge(lefthalf, righthalf):
    #
    # Implement your code here
    #

def mergesort(x):
    if len(x) == 0 or len(x) == 1:
        return x
    else:
        middle = int(len(x)/2)
        a = mergesort(x[:middle])
        b = mergesort(x[middle:])
        return merge(a,b)

def main():

    aList = [10, 5, 2, 9, 6, 3, 4, 8, 1, 7]
    print(mergesort(aList))

    list1 = [1, 4, 6, 10]
    list2 = [5, 7, 8, 11, 12]
    print(merge(list1, list2))

main()
```

Expected output of the program:

```
>>> ================================ RESTART ================================
>>>
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 5, 6, 7, 8, 10, 11, 12]
>>>
```

2. [35 marks] Suppose there is a spaceship, named Alpha, navigating in the space which can be abstracted by a 10 x 10 2-dimensional coordinate system and Alpha can freely move around this system. In this system, the upper left-hand corner is (0, 0) and so Alpha in below example is locating at (2, 1). There are eight stars residing in a number of locations in the system and Alpha should avoid colliding with the stars. Below illustrates the abstraction:



Suppose you have to write a Python program to model the above system.

(a) (10 marks) Describe, in details, how you can store the locations of the spaceship and the stars in your program. Your design should consider that the location of Alpha will be dynamic based on user input, while those of the stars are static.
(b) (25 marks) Implement your design in (a) using Python code. The location of Alpha (x) and the stars (*) are randomly generated (Hint: https://docs.python.org/3/library/random.html). The execution result of your program should look like below:

```
  0 1 2 3 4 5 6 7 8 9
0       *           x
1           *       *
2               *
3
4
5
6   *   *
7           *   *
8
9
>>> main()
  0 1 2 3 4 5 6 7 8 9
0       *           *
1                   *
2           *
3
4   x       *
5         *   *
6
7                   *
8
9
>>>
```

Type your answer in (a) as comments at the beginning of your program in (b).

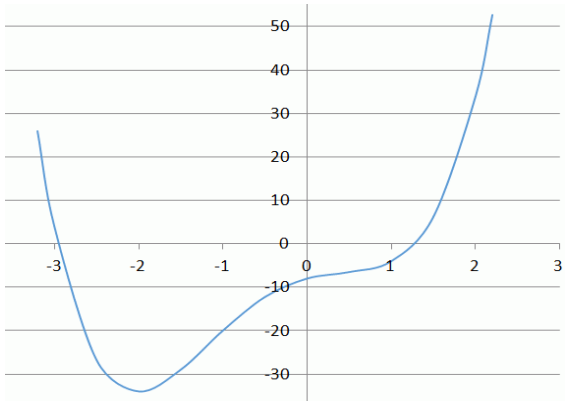
3. [25 marks] In Assignment 3, we have implemented several functions so that we are able to add together multiple arbitrarily large numbers, via simple table lookup under the concept of *symbolic computation*. Now, we are to extend what we have implemented in Assignment 3 so that we are able to ***multiply*** multiple arbitrarily large numbers. In order to multiply several large numbers, we need to first multiply two large numbers. In order to multiply two large numbers, we need to first multiply one large number by a small number and this can be further refined. As such, as we are adopting a *top-down approach* in solving this problem step by step, until we are able to make use of our program in Assignment 3, or solve it otherwise.

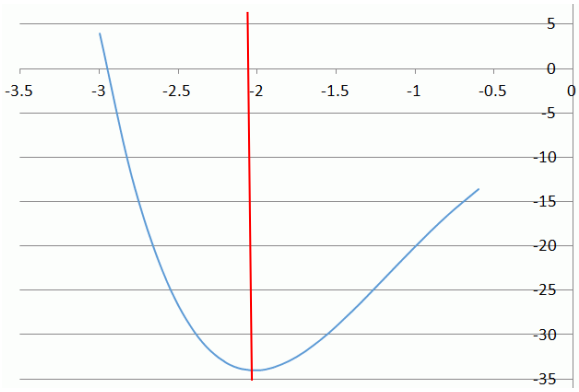
   (a) (10 marks) Write the pseudo-code to multiply a list of large numbers, based on the concept of symbolic computation. As with Assignment 3, no mathematical operators could be used. Briefly explain your pseudo-code whenever appropriate. You can assume that all 4 functions in Assignment 3 are available, i.e., you can make a "call" to those functions. ***Type your pseudo-code*** as comments at the beginning of your Python program in (b).

   (b) (15 marks) Develop a Python program that implements your pseudo-code. Use functions whenever appropriate. Here are some sample outputs from your program:

```
>>> main(["61431","1002"])
Input list ['61431', '1002']
The product is 61553862

>>> main(["61431","1002","61434","1001","123456789"])
Input list ['61431', '1002', '61434', '1001', '123456789']
The product is 4673186942740798434072l2
```

4. [25 marks] In Lecture 2, we considered the problem of finding the root of an equation by searching within a given interval [*a*,*b*] and provided the pseudo-code. A root is between [1,2] and another is between [-3,-2] for $f(x) = 2x^4+3x^3-6x^2+5x-8 = 0$ (on the left). A related problem to solving an equation, i.e. looking at the point(s) where the function value is technically zero, is to find the minimum (or maximum) value of the function. Once again, the common approach is to search for the point(s) where the function value is smallest (or largest). From the graph of $f(x) = 2x^4+3x^3-6x^2+5x-8$, the minimum would occur somewhere between -3 and -2 (zoomed in on the right). A more careful checking would reveal that it occurs around $x = -2.02$ for $f(x) = -34.01$.





   We are to extend the pseudo-code in Lecture 2 to solve this problem. However, unlike the simple middle point approach as in Lecture 2, you have no clue after getting the middle point on whether you should go searching on the left or go on the right. Therefore, you will start with ***three points***

instead of two, i.e., *a*, *b*, and *c* such that $a < c < b$ and $f(a) > f(c)$ and $f(b) > f(c)$. For example, we may choose $a = -3$, $b = -1$ and $c = -2.2$ for the equation above. Then the minimum should be in the neighborhood of *c*. Suppose that the left part is larger ($c - a > b - c$), find the middle point *m* on the left part. Now, we have 4 points *a*, *m*, *c*, *b*. Select the minimum point among the four points to be the new *c*, and the two points around *c* to be the new *a* and new *b*. One of the old values of either *a* or *b* will be then discarded. The mechanism is similar if the right part is larger ($c - a < b - c$). This procedure will be repeated until we are satisfied that we are close to the minimum (i.e., difference between $f(m)$ and $f(c)$ and difference between *m* and *c* are small enough).

Develop a Python program to realize this concept, printing out results from the intermediate attempts. The program will take in three parameters: *a*, *b*, *c*. To simplify the program, just define a function **`f()`** in Python for the function $f(x)$ to minimize. Use functions whenever appropriate. Here are some sample outputs from your program. You may use **`format`** to print nicely.

```
>>> main(-3,-1,-2.2)
Step 0
a=-3.000000 b=-1.000000 c=-2.200000
f(a)=4.000000 f(b)=-20.000000 f(c)=-33.132800
m=-1.600000 f(m)=-30.540800
Step 1
a=-3.000000 b=-1.600000 c=-2.200000
f(a)=4.000000 f(b)=-30.540800 f(c)=-33.132800
m=-2.600000 f(m)=-22.892800
Step 2
a=-2.600000 b=-1.600000 c=-2.200000
f(a)=-22.892800 f(b)=-30.540800 f(c)=-33.132800
m=-1.900000 f(m)=-33.672800
Step 3
a=-2.200000 b=-1.600000 c=-1.900000
f(a)=-33.132800 f(b)=-30.540800 f(c)=-33.672800
m=-1.750000 f(m)=-32.445312
...
Step 32
a=-2.020493 b=-2.020489 c=-2.020490
f(a)=-34.010302 f(b)=-34.010302 f(c)=-34.010302
m=-2.020491 f(m)=-34.010302
Step 33
a=-2.020491 b=-2.020489 c=-2.020490
f(a)=-34.010302 f(b)=-34.010302 f(c)=-34.010302
m=-2.020490 f(m)=-34.010302
Minimum value=-34.010302 occurring at -2.020490

>>> main(-3,-1,-2)
Step 0
a=-3.000000 b=-1.000000 c=-2.000000
f(a)=4.000000 f(b)=-20.000000 f(c)=-34.000000
m=-1.500000 f(m)=-29.000000
Step 1
a=-3.000000 b=-1.500000 c=-2.000000
f(a)=4.000000 f(b)=-29.000000 f(c)=-34.000000
m=-2.500000 f(m)=-26.750000
...
Step 33
a=-2.020496 b=-2.020489 c=-2.020493
f(a)=-34.010302 f(b)=-34.010302 f(c)=-34.010302
m=-2.020491 f(m)=-34.010302
Step 34
a=-2.020493 b=-2.020489 c=-2.020491
f(a)=-34.010302 f(b)=-34.010302 f(c)=-34.010302
m=-2.020490 f(m)=-34.010302
Minimum value=-34.010302 occurring at -2.020491
```

Submission Instructions

Follow the steps below:

1. Create a folder and name it as <student no>_<your name>,
   e.g., **`12345678d_CHANTaiMan`**
2. For Q1, to Q4, you need to submit the source file (**`.py`**). Name the **`.py`** files as A4_Q<question no>_<student no>_<your name>**`.py`**,
   e.g., **`A4_Q1_12345678d_CHANTaiMan.py`**
3. Put all the **`.py`** files into the folder.
4. Compress the folder (**`.zip`**, **`.7z`**, or **`.rar`**).
5. Submit the file to Blackboard.

A maximum of **3 attempts** for submission are allowed. **Only the last attempt will be graded**. A late penalty of 5% per hour will be imposed.