National University of Singapore
School of Computing
IT5003: Data Structure and Algorithm
Semester I, 2019/2020

Tutorial x Lab 2
Sorting II & Abstract Data Type

General idea: Tutorial type questions will be listed first in this document. Some tutorial question will then be implemented in the lab portion (later part of this document). Lab exercise cases to be submitted on Coursemology will be marked [SUBMISSION] at the front.

1. [Sorting – Leftover from TL01-Q3] You are given the following array: `1285, 5a, 150, 4746, 602, 5b` and `8356`. Subscripts 'a' and 'b' are used to just distinguish the two 5's. Trace each of the following sorting algorithms as it sorts the above array into **ascending** order.
   e. Quick-sort [Discuss in tutorial]
   f. Radix-sort [Discuss in tutorial]

2. Using time complexity of **merge sort** and **radix sort**, suggest a usage scenario where merge sort is preferred. Then suggest a second scenario where radix sort is preferred.

3. As discussed in lecture, quick sort can degenerate into worst case if the input is (mostly) sorted (either ascending or descending order). Let's try the following:
   a. Suggest a few ways to make quicksort less susceptible to skewed input.
   b. Suggest how do we make the partition() implementation **user-adaptive**, i.e. can allow the coder to apply idea in (a) easily.

4. From the example tracing (Q1, part e), we know that quicksort is unstable.
   a. Focusing on the partition algorithm, discuss the cause(s) of this problem. i.e. identify step in the algorithm that can cause unstable sorting.
   b. Suggest a way to make quicksort stable. Discuss briefly the impact on time / space complexity.

For question 5 and 6. We explore the idea of **abstract data type**. For each question:

a. Draft a **specification** in Python (i.e. class with empty methods). Focus on the functionalities, the parameters (arguments) for the methods. As the problem is quite open ended, try to draft an **essential set of functionalities.**

b. **Discuss** possible implementation(s). Focus on a) how / what to store by objects of the class to support the specification. Discuss the involved algorithm / data structure only when necessary.

5. We want to support **quadratic equation** in Python. A quadratic equation has form:

$$ax^2 + bx + c = 0$$

where a, b and c are **real numbers** and a != 0.

If you are feeling a bit rusty on this topic 😓, take a look at the Wikipedia page https://en.wikipedia.org/wiki/Quadratic_equation as a quick review.

Discuss the specification and implementation for this **QEquation** ADT.

6. We want to support **a deck of playing cards (poker cards)** in Python. A playing card has 4 possible **suits** {Spade, Heart, Club, Diamond} and 13 possible **ranks** {Ace, Two,…., Ten, Jack, Queen, King}. A deck of playing cards has 52 cards (4 suits x 13 ranks).

The **Deck** ADT should be designed to support simple card game.

~~~ Lab Questions ~~~

1. Refer to the given **`Deck.py`**, which gives:
    a. A simple **Card ADT** using Python Enumeration (Enum).
    b. A sample **Deck ADT** with specification but no implementation.
    c. A sample main function that draws two poker hands (5 cards each) from the deck and show on the screen. There are also commented off code segment to highlight some basic usage of the Card ADT. You can uncomment as needed to try out.

    Use ideas from the tutorial discussion to complete (b), i.e. the Deck ADT.

Sample Run (no user input):

```
Deck has 52 cards at the start

First poker hand:  //note your cards may be different due to the randomized shuffling
FOUR HEART
THREE HEART
KING HEART
TWO CLUB
JACK DIAMOND

Second poker hand:
NINE CLUB
SEVEN HEART
FOUR CLUB
TEN CLUB
QUEEN DIAMOND
Deck has 42 cards at the end
```

Feel free to build actual card games from the Deck ADT 😊.

2. **[Submission]** Similarly, use the given **`QEquation.py`** to complete **`QEquation` ADT**. Read the specification carefully on the functionality and output requirement. You should restrict the coding only in **`class QEquation`**. The **`main()`** function is written to allow user input for you to quickly test the various functionalities, feel free to change it to suit your needs.

   Use the sample public test cases on Coursemology to check your functionalities.

   Submit ONLY `QEquation` class on coursemology, i.e. not even the **`main()`**.

   [Hint: For the intersection method, you can use discriminant in some way to help.]

From this lab onward, we will evaluate both correctness and programming style of your submitted code:

a. [Correctness 70%]: The code works according to the functional requirement, e.g. output value, output format, side effect etc.

b. [Programming Style 30%]: Reuse own code whenever appropriate, modularity, good variable / method naming convention, comments (only if appropriate)