Bubble Sort and Quick Sort

Hui Chen (huichen@ieee.org)
February, 2017

1 Objective

Via experimenting with an implementation of the Bubble Sort and Quick Sort algorithms in Python, at the conclusion of the discussion in class and this exercise, you are expected to be able to explain and implement the Bubble Sort and Quick Sort algorithms, to understand important sorting algorithm evaluation criteria, in particular, to explain time complexity of the Bubble Sort and Quick Sort algorithms, and to explain the concept of the divide-and-conquer strategy using the design of the Quick Sort algorithm.

2 Resources

This document is available on the Web at the following URL,

- `https://goo.gl/jPnkBf`

The associated presentation slides of these exercises are available at the following URL,

- `https://goo.gl/dQxUqQ`

You may find the source code of the Bubble Sort and Quick Sort implementation written in Python at a `Github` repository via the following URL,

- `https://goo.gl/phTjrS`

If you have `Eclipse` and have also installed the `PyDev` plugin for `Eclipse`, you may directly import the project into `Eclipse` after you have cloned the project using `git` or the `Github Desktop` application.

Alternatively, you can also find most of the source code at `Repl.it` at the following URL,

- `https://goo.gl/QA3pNO`

`Repl.it` is a simple Web-based IDE. To use the simple Web-based IDE, you only need a Web browser, such as, the `FireFox` or `Chrome` Web browser.

Except Exercise 1 in the Exercises section below, you are advised to create a free account at `Repl.it`, create a session of yourself, copy the code to your own session at `Repl.it`, and work on the excises in your own session, if you choose to use `Repl.it` instead of `Eclipse`.

3 Exercises

Complete the following exercises. When completing the exercises, think about and try to explain *why* it happened besides observing *what* happened.

1. Run the main program, observe the relationship between the running time and the size of the lists, and compare the observations between Bubble Sort and Quick Sort.

2. In Exercise 1, the integers in the lists that you are sorting are randomly generated. In this task, you are to generate ordered lists of numbers by revising the provided code. In particular, you will observe the relationship between running time and the size of the lists for the following two cases.

   (a) Sort lists of integers that are ordered in an ascending order. You may generate a list of integers in an ascending order in Python as follows,

   ```
   my_list = list(range(0, 1000, 1))
   ```

   (b) Sort lists of integers that are ordered in a descending order. You may generate a list of integers in a descending order in Python as follows,

   ```
   my_list = list(range(1000, 0, -1))
   ```

   For the two cases, compare the observations between Bubble Sort and Quick Sort, and compare the observations in this exercise with those in Exercise 1.

3. Revise your code to select the middle member of the list as the pivot. Repeat the task specified in Exercise 2. If the length of the list is *odd*, you can select the middle member of the list using its index as follows,

   ```
   middle_member = my_list[(len(my_list)-1)/2]
   ```

   Note that in Python lists are indexed from 0. Compare your observations in this exercise with those in Exercise 2.

4 Acknowledgement

The discussion in class and these exercises are inspired by the following resources.

- The CS Unplugged project.
  - `http://csunplugged.org/`
- The OpenDSA project.
  - `https://opendsa-server.cs.vt.edu/`
- The Runestone Interactive project, in particular, the interactive online book of "Problem Solving with Algorithms and Data Structures using Python",
  - `http://interactivepython.org/runestone/static/pythonds/index.html`

You are encouraged to explore those resources.