**Tutorial 10 – Sorting**

(Week starting 6 April 2015)

# 1. Sorting Choice

In this question, we will only consider the 4 sorting algorithms: *Insertion Sort*, *Quick Sort*, *Merge Sort*, and *Radix Sort*. Given the 4 algorithms above, consider the following scenarios and choose the **fastest** sorting method to be used for each scenario.

1. You are making a database of the *weight* of the students in NUS for your CCA. However, because of the budget cut, you are facing a problem in the amount of *memory* available for your computer. The memory available is just barely enough to put the entire values in the memory. You should be able to sort the database based on weight.
2. Since your success in creating the database for your CCA, you are hired as an intern in SoC. The database that you are creating now is a student database. It should contains the *name* and *age*. The sorting requirement is that you should sort the student firstly by *name*, then by *age*. That means all the students with the same age will be displayed in a sorted order.
3. After finishing intern in SoC, you are invited to be a guest lecturer on sorting class. You just had final exam for the module, and now you just finished marking the papers *randomly*. You want to display the *result* to your students in a sorted order.
4. Little did you know that you were already marking the *grades* in an *almost sorted order*. However, being a very peculiar skeptics, you still want to sort the result.

**2. Longest Sub-Array**

The *Longest Sub-Array* problem is described as follows: given an array *A* of integers and an integer *k* (the sum), for each index *i* (), print out the leftmost index *j* of the longest consecutive sub-array *B* ending at *i* such that the sum of the all numbers in *B* is *k* if such index exists. In short, the sum of array *A* from index *j* to index *i* is equal to *k*.

Example: k=-8, A=[2, 6, -3, -5, -4, 8, 12, -20, 1]

Result (as pair of <i,j>):

<3,2>: (-3 + -5 = -8)

<7,6>: (12 + -20 = -8)

<8,3>: (-5 + -4 + 8 + 12 + -20 + 1 -8)

1. Write down a naïve algorithm that runs in time. Note that a totally naïve algorithm would actually be a algorithm.
2. Write down an optimized algorithm that runs in time.

*Extra: a similar problem (positive interval): given an array, count the number of intervals in which the elements sum up to a positive number.*

**3. Counting Triangles**

You are given N sticks, numbered with 1…N. The length of the i-th stick is A[i]. You would like to form a triangle with 3 of these sticks. How many possible triangles can you form?

Two triangles are different as long as they use at least one different stick. For example, if the sticks are of lengths [3, 3, 4, 5, 6], there are 9 possible triangles, namely:

<3, 3, 4>, <3, 3, 5>, <3, 4, 5>, <3, 4, 6>, <3, 5, 6>

<3, 4, 5>, <3, 4, 6>, <3, 5, 6>

<4, 5, 6>

1. O(N3) solution
2. O(N2logN) with binary search
3. O(N2) solution.

**4. K-Sum**

Similar to Quesiton2, but this time, you are given 4 arrays of size N. You need are to choose an element from each of the four arrays, such that they sum up to K. Count the number of such 4-tupples.

For example, if K=10 and the arrays are [1, 2], [3, 1], [5, 1], [7, 6], there are 2 such choices, namely <1, 1, 1, 7> and <2, 1, 1, 6>.

1. O(N4) solution
2. O(N2logN2) using sorting and binary search
3. O(N2) solution using hashing.