Workshop Test 1: Algorithm Analysis (Your marks: (out of 10))

Student ID: Name:

Tasks: Attempt all five questions below. Note: all workings must be shown!

1. (2 marks)
   (i) Using a manual method, apply *floor*( ) and *ceiling*( ) functions to $log_2(996)$.
   (ii) Under what condition/s, *floor*($\log_2(x)$) = *ceiling*($\log_2(x)$)? Why?

2. (1 mark)
   Find the Greatest Common Divisor (GCD) of 462 and 105 by manually executing the Euclid GCD algorithm shown on slides 7~9 of Lecture1.ppt.

3. (3 marks)
   (i) Consider the *binary search* algorithm given in slides 24, Lecture03.ppt (also see the algorithm in page No. 409 of the textbook, or the Algorithm in page No.43 of the Java Collections). Hand test this algorithm with the following array of words:

   apple, banana, grape, lime, mango, orange, pear, pineapple, plum, strawberry

   with each of the following target values: **plum; Lychee**.
   (Hint: refer to the examples in slides 26 & 27, Lecture03.ppt, and track changes of *l* , *r* and *m*).
   (ii) How many comparisons are required in each case?

4. (2 marks)
   Assume that the following expression is the function of a given algorithm (*n* is input size),

$$(7n+8)^4 - 72 * (n+1)^9 / (n-3)^4 + n * (log_2(n))^{12} + 1029$$

   Determine the time complexity of the algorithm in *O*-natation.

5. (2 marks)
   (i) An array A[0…*n*-1] is in *descending order* if A[*i-1*] ≥ A[*i*] holds for all *i (0< i <n).*

   Assume an array, **A**[0… *n*-1], contain distinct integers. Write an algorithm, ***descending(*A, *n)***, that determines whether or not the components of **A** are in descending order ($n \geq 0$). If they are in descending order, the algorithm returns *n*, otherwise it returns an integer *m*, which is the minimum index value of **A**, such that $m \leq n$ and **A**[*m-1*] <***A***[*m*].

   For instance, let **A**[ ] = {62, 50, 40, 22, 21, 20, 11}and **B**[ ] = {62, 50, 40, 20, 21, 22, 11}. Then ***descending(*A, *7)*** would return 7 and ***descending(*B, *7)*** would return 4**.**

   (ii) Determine the time complexity of your algorithm, using *O*-notation.
   (Hint: refer to the example in slides 33, Lecture03.ppt).