Due Date: Friday 13th Sept. 2013 at 17:00 (week 07).

You are required to attempt all the questions listed below, and submit your work in a single .zip file containing document (in either MS word or PDF format) and java source code.

The indicative marking scheme is shown below.

Question 1: [60 marks, all subquestions have equal marks]

You are required to calculate the time complexity using the big-O notation of the myAlgorithm(int n) method defined by the following Java code.

Code: Algorithm_1	##
void algorithm_1(int n) {	01
if $(n < 1)$ return;	02
System.out.println($q(1, n)*n$);	03
System.out.println(r(n));	04
System.out.println($q(1, n+n) + r(n+n)$);	05
}	06
int q(int i, int n) {	01
return $i+(i \ge n ? 0 : q(i+i, n));$	02
}	03
int r(int n) {	01
int sum $= 0;$	01
for (int i=1; i <= $n+n$; i++)	03
sum + = i + q(1,n);	04
return sum;	05
}	06
int t(int n) {	01
for (int i=1; i <= n+n; i++) {	02
for (int $j=1; j < i; j++$)	03
sum + = i + q(1,n);	04
sum + = 1 + q(1,n);	05

return sum;	06
}	

- 1. For n > 0, what is the time complexity of the method q(1, n). Show the details of your calculation of $O(q(1, n) \approx O(?))$.
- 2. For n > 0, what is the time complexity of the r(n) method. Show the details of you calculation of $O(r(n)) \approx O(?)$.
- 3. For n > 0, what is the time complexity of the *algorithm_l(int n)* method. Show the details of you calculation of $O(algorithm_l(n)) \approx O(?)$.

Question 2: [40 marks]

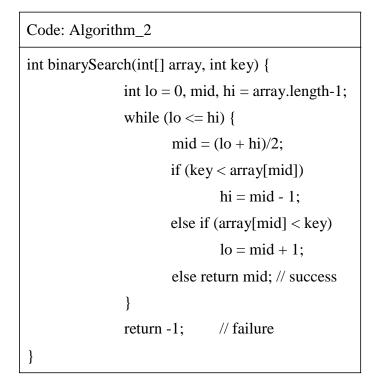
You are required to calculate the time complexity using the big-O notation of the *Algorithm_2* given by the following methods defined by the following Java code.

1. Explain how the *binarySearch(array[n], key)* algorithm works support your answer with an illustration of this search algorithm.

[15 marks]

2. For n > 0, what is the time complexity of the *binarySearch(array[n], key)* algorithm. Show the details of you calculation of $O(binarySearch(array[n], key)) \approx O(?)$.

[15 marks]



3. Write a Java program that counts the number of operations the *binarySearch* algorithm executes to search a given array of size *n*. Hint: you can simply extend the above code say the *while loop* to count the number of iterations it executes for a given array size *n*. The program should output say two variables *n* and *iterated* respectively for the array size and the number of iterations the *binarySearch* has executed.

Assuming, that *eachAlgorithm_2* iteration takes a constant *timet* 0.01 s (seconds), plot the experimental time complexity of *binarySearch(n)*, which can be calculated as: t(n)*iterated*t*.

[10 marks]