NAME: **ID#:**

COMP 250 Fall 2013 - Midterm examination 1A

October 4th 2013, 10:35-11:25
This is an open book exam, but no electronic equipment is allowed.

1 Java programming (25 points)

a) (8 points) What will the following Java program print when executed?

```
class question1 {
   static public void questionA(int x) {
      x = x + 5;
   }

   static public int questionB(int x) {
      x = x + 3;
      return x;
    }

    static public void questionC(int array[]) {
       array[0] = array[0] + 1;
    }

    public static void main(String args[]) {
       int x, y;
       int a[] = new int[10];
       x = 1;
       y = 8;
       a[0] = 1;
       questionA(x);
       y = questionB(y);
       questionC(a);

       System.out.println("x = " + x);
       System.out.println("y = " + y);
       System.out.println("a[0] = " + a[0]);
   }
}
```

Answer:
x = 1
y = 11
a[0] = 2

b) (17 points) We say that two students are evil twins if the sum of their student ID numbers is exactly 999999999. Write the Java code for the containsEvilTwins method below, which takes as argument a studentList and return true if the studentList contains at least one pair of evil twins, and false otherwise. Your program must run in time $O(n \log_2 n)$, but you don't need to actually demonstrate that it does.

You can use any of the Java methods seen in class or in the assignments, without having to give the code for them.

```
class studentList {
    int studentID[];
    int numberOfStudents;
    String courseName;

    public static boolean containsEvilTwins( studentList L ) {
    /* WRITE YOUR CODE HERE */

    mergeSort(studentID, 0, numberOfStudents - 1);
    for (int i = 0; i< numberOfStudents; i++) {
            if (binarySearch( studentID, 0, numberOfStudents-1, 999999999-studentID[i]) != -1 ) {
                return true;
            }
    }
    return false

   }
}
```

2 Execution of recursive algorithms (24 points)

Consider the following recursive algorithm that computes the n-th Fibonacci number:

Algorithm RecursiveFibonacci(n)
Input: A non-negative integer n
Output: The value of the n-th Fibonacci number

print n
if $(n \leq 1)$ **then return** n
else
. $a =$ RecursiveFibonacci$(n-1)$
. $b =$ RecursiveFibonacci$(n-2)$
. **return** $a + b$

a) (12 points) What will be *printed* when RecursiveFibonacci(4) is called?

What gets in printed

RF(4)	4
RF(3)	3
RF(2)	2
RF(1)	1
RF(0)	0
RF(1)	1
RF(2)	2
RF(1)	1
RF(0)	0

b) (12 points) Let $P(n)$ be the number of elements that will be printed during the execution of RecursiveFibonacci(n). Give a recurrence for $P(n)$. You *don't* need to prove your answer. You *don't* need to obtain an explicit formula for your recurrence.

$$P(n) = \begin{cases} 1 + P(n-1) + P(n-2) & \text{if } n>1 \\ 1 & \text{if } n= 0 \text{ or } 1 \end{cases}$$

3 Big-O notation (20 points)

a) (12 points) Indicate, for each pair of functions (f, g) in the table below, whether $f(n)$ is $O(g(n))$, and whether $g(n)$ is $O(f(n))$. Write either "yes" or "no" in each box. No justifications are necessary. [Correct answer: 1.5 point, wrong answer = -0.5 point].

$\mathbf{f(n)}$	$\mathbf{g(n)}$	$\mathbf{f(n) \in O(g(n))}$	$\mathbf{g(n) \in O(f(n))}$
$9n + n\log_2(n) + 1$	n^2	Yes	No
n	$n \cdot (\sin(n) + 1) + 5$	Yes	Yes
$\log_2(n^2)$	$\log_3(n^3)$	Yes	Yes
$n^{1.01}$	$n\log_2(n)$	No	Yes

b) (8 points). Give the formal proof for *one* of eight Yes/No answer you gave in (a). You are free to pick whichever one you find easiest. Your proof should only rely on the formal definition of the big-O notation.

Let's do the second one, proving that n (sin (n) + 1) + 5 is O(n)

To do this, we must find constants c and n0 such that n (sin(n)+1) + 5 <= c n for all n>=n0

We note that
n (sin(n) + 1) + 5 <= n (1 + 1) + 5
= 2n + 5
<= 2n + 5n (if n>=1)
= 7 n
So, if n0=1 and c = 7, we get that n (sin(n)+1) +5 <= c n for all n>=n0.
Thus, n (sin(n)+1) + 5 is O(n).

4 Running time analysis (10 points)

The following algorithm takes two arrays of integers as input and counts how many elements they have in common.

Algorithm intersectionSize(A, B, n)
Input: A is an array of n distinct integers; B is an array of n distinct integers
Output: The number of elements that are present in both arrays

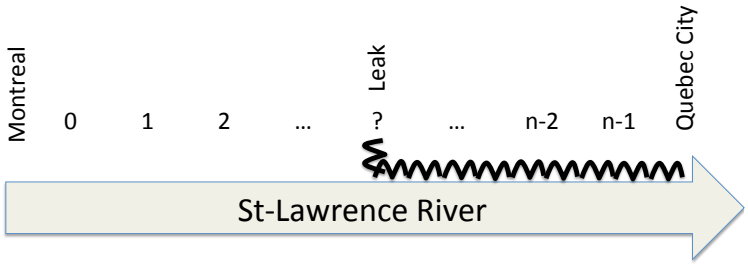
```
count ← 0                                                          1
mergeSort(B, 0, n − 1)                                             2 + (10n log(n) + 5n + 3 = 10 n log(n) + 5n +5
for i ← 0 to n − 1                                                 init: 1, condition: 3, Loop n times
.    if (binarySearch(B, 0, n − 1, A[i]) ≠ −1) then                5 + (3 log(n) + 4) = 3 log(n) + 9
.        count ← count + 1                                         2
return count                                                       increment: 2, last condition check: 3, 1
```

Assume that the worst case running time of mergeSort on an array of size n is $10n\log_2(n)+5n+3$ and the worst case running time of binarySearch on an array of size n is $3\log_2(n)+4$. Let $T(n)$ be the worst case total number of primitive operations performed by the intersectionSize algorithm when executed on arrays of size n. Give an exact formula for $T(n)$.

T(n) = 7 + 10 n log(n) + n (3 log(n) + 16) + 4
= 13 n log(n) + 16 n + 11

5 Recursive algorithms (21 points)

Between Montreal and Quebec City, there are n plants located along the St-Lawrence River, as shown below. The current of the river flows from Montreal to Quebec City. One of the plants has a chemical leak. The chemicals are flowing from that plant downstream and have contaminated all the sites downstream of plant (including in front the plant itself). It is your job to find out which plant has the leak.



You can send a robot to test a water sample in front of plant k by calling the following method:

Algorithm isContaminated(k)
Input: The index k of the plant in front of which the water is to be tested
Output: True if the water in front of plant k is contaminated, and false otherwise

However, testing water samples is expensive, so you can only afford to perform $\log_2(n)$ tests. Complete the algorithm below, which returns the index of the plant that has the leak.

Algorithm findGuiltyPlant(firstPlant, lastPlant)
Input: The index of the first and last plants to be tested.
Output: The index of the plant that has the leak

```
if (firstPlant = lastPlant) then
        if (isContaminated(firstPlant)) return firstPlant
        else return -1
else
        mid = (firstPlant + lastPlant) / 2
        if (isContaminated(mid)) then return findGuiltyPlant(firstPlant, mid)
        else return findGuiltyPlant(mid+1,lastPlant)
```

6 Bonus question (5 points)

Let $T(n)$ be defined as follows:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/2) + n + 1 & \text{if } n > 1 \end{cases}$$

Give an explicit formula for $T(n)$.

We will only consider cases where n is a power of 2.
Let's first get the first few values of T(n), for verification purposes.
T(1) =1, T(2) = 6, T(4) = 23, T(8) = 78

Using the substitution method, we get
T(n) = 3 T(n/2) + n + 1 (1)
= 3 (3 T(n/4) + n/2 + 1) + n + 1 = 9 T(n/4) + n(3/2 + 1) + 3 +1 (2)
= 9 (3 T(n/8) + n/4 + 1) + n(3/2 + 1) + 3 +1 = 27 T(n/8) + n (9/4+3/2+1) + 9 +3 +1 (3)
= …
= 3^k (T(n/2^k) + n (sum_{i=0}{i=k-1} (3/2)^i) + sum_{i=0}{i=k-1} 3^i (k)
= 3^k (T(n/2^k) + n ((3/2)^k - 1) / (3/2 -1) + (3^k - 1)/(3-1)
= 3^k (T(n/2^k) + 2 n ((3/2)^k - 1) + (3^k - 1)/2
The recursion ends when n/2^k = 1, i.e. when k = log_2 (n). We then get
T(n) = 3^log(n) T(1) + 2 n (3^log(n) / 2^log(n) -1) + 3^log(n)/2 - 1/2
= 3^log(n) T(1) + 2 n (3^log(n) / n -1) + 3^log(n)/2 - 1/2
= 3^log(n) + 2 (3^log(n)) -2n + 3^log(n)/2 - 1/2
= 7/2 (3^log(n)) - 2n - 1/2

Conclusion: T(n) = 7/2 3^log(n) - 2n - 1/2
Verification: T(1) = 1, T(2) = 6, T(4) = 23, T(8) = 78. So everything matches…

Note: 3^log(n) = n^log(3) = n^1.57… so T(n) = 3.5 n^1.57 - 2n - 0.5