San José State University
College of Science / Department of Computer Science
CS 146 Data Structures and Algorithms
Section 4/7, Spring 2015
Instructor: Dr. Angus Yeung

Assignment 1
Due Date: Monday, February 9, 11:59 pm

Submission:

1. Create a folder called `hw1` on your computer.
2. Create a WORD document (or any similar word processing document) and put all of your answers in the same document. Your word processing software must allow you to enter math equations.
3. Save the file as `hw1`, e.g., `hw1.doc` or `hw1.docx`. Export the Word document (or similar word processing document) to PDF file. For example, File -> Save As -> choose Format as "PDF".
4. Alternatively, you can write your answers on paper, scan and save your work as a PDF file.
5. Copy or move your PDF file, `hw1.pdf`, to folder `hw1`
6. For programming assignment, follow the instruction in the question to submit your `.java` files.
7. When you are ready for submitting your work, zip up the `hw1` folder and upload it to Assignment 1 on Canvas.

Only softcopy is acceptable. Do not hand in your solutions in hardcopy. Do not encrypt your zip file with a password.

Problems (Total: 100 Points):

Part A contains written assignment questions. Follow the instruction in each question carefully and show all of your work. (40 Points)

Part B contains both written and programming assignment questions. Follow the guidelines in each assignment for programming and submission requirements. (60 Points)

PART A (40 Points)

1.1 (5 Points) Prove the following formula by induction: $\sum_{i=1}^{N} i^3 = (\sum_{i=1}^{N} i)^2$. You must show base case, inductive hypothesis and proof in your solution.

1.2 (5 Points) Write a recursive method that returns the number of 1's in the binary representation of N. Use the fact that this is equal to the number of 1's in the representation of N/2, plus 1, if N is odd.

1.3 (5 Points) Prove that $\sum_{i=1}^{N} i \times i! = (N+1)! - 1$ by induction. You must show base case, inductive hypothesis and proof in your solution.

1.4 (15 Points) List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

N^2	N	$\sqrt{N}$	$N \log N$	48
N^3	$N^2 \log N$	$N \log N$	$N^{1.5}$	$N \log \log N$
$N (\log N)^2$	$N \log(N^2)$	$1/N$	$2^{N/2}$	2^N

1.5 (10 Points) For each of the following pairs of functions $f(N)$ and $g(N)$, determine whether $f(N) = O(g(N))$, $g(N) = O(f(N))$, or both. You must provide both **answers** and **explanations** for this question.

$f(N) = (N^2 - N + 3)/3, g(N) = 6N$
$f(N) = N + 2\sqrt{N}, g(N) = N^2$
$f(N) = N \log N, g(N) = N\sqrt{N}/2$
$f(N) = 2(\log N)^2, g(N) = \log N + 1$
$f(N) = 4N \log N + N, g(N) = (N^2 - N)/2$

PART B (60 Points)

1.6 (15 Points) Define a class that provides `getLength` and `getWidth` methods. Using the `findMax` routines in Figure 1.18 (listed below), write a `main` that creates an array of `Rectangle` and finds the largest `Rectangle` first on the basis of area, and then on the basis of perimeter. (15 Points)

```
    // Generic findMax, with a function object.
    // Precondition: a.size( ) > 0.
    public static <AnyType>
    AnyType findMax( AnyType [ ] arr, Comparator<? super AnyType> cmp )
    {
        int maxIndex = 0;

        for( int i = 1; i < arr.size( ); i++ )
            if( cmp.compare( arr[ i ], arr[ maxIndex ] ) > 0 )
                maxIndex = i;

        return arr[ maxIndex ];
    }

class CaseInsensitiveCompare implements Comparator<String>
{
    public int compare( String lhs, String rhs )
      { return lhs.compareToIgnoreCase( rhs ); }
}

class TestProgram
{
    public static void main( String [ ] args )
    {
        String [ ] arr = { "ZEBRA", "alligator", "crocodile" };
        System.out.println( findMax( arr, new CaseInsensitiveCompare( ) ) )
    }
}
```

Figure 1.18 Using a function object as a second parameter to `findMax`; output is ZEBRA

How to Submit:

Additional requirements for submission:

(1) Create the folder "`findRectangle`" that contains all required `.java` file(s).
(2) The folder should be part of the `hw1.zip` file that you upload to Canvas.
(3) Do not include any other file types inside the `findRectangle` folder except `.java` file(s).
(4) Do not declare and use any "`package`" in your `.java` file(s).
(5) Use "`findRectangle`" as the class name – so you can run as `%java findRectangle`.

There is one point penalty for each requirement that a student fails to follow (total penalty: 5 points).

1.7 (30 Points) For each of the following six program fragments:

a. Give an analysis of the running time (Big-Oh will do).
b. Implement the code in Java, and give the running time for several values of N.
c. Compare your analysis with the actual running times.

```
1. sum = 0;
   for( i = 0; i < n; i++ )
       sum++;

2. sum = 0;
   for( i = 0; i < n; i++ )
       for( j = 0; j < n; j++ )
           sum++;

3. sum = 0;
   for( i = 0; i < n; i++ )
       for( j = 0; j < n * n; j++ )
           sum++;

4. sum = 0;
   for( i = 0; i < n; i++ )
       for( j = 0; j < i; j++ )
           sum++;

5. sum=0;
   for( i = 0; i < n; i++ )
       for( j = 0; j < i * i; j++ )
           for( k = 0; k < j; k++ )
               sum++;

6. sum=0;
   for( i = 1; i < n; i++ )
       for( j = 1; j < i * i; j++ )
           if( j % i == 0 )
               for( k = 0; k < j; k++ )
                   sum++;
```

How to Submit:

The Big-Oh estimation and the analysis for actual running time must be submitted with other written questions in the same `hw1.pdf` file. Below is the additional requirements for Java program submission:

(1) Create the folder "`fragments`" that contains all required `.java` file(s).
(2) The folder should be part of the `hw1.zip` file that you upload to Canvas.
(3) Do not include any other file types inside the `fragments` folder except `.java` file(s).
(4) Do not declare and use any "`package`" in your `.java` file(s).
(5) Use "`fragment1`" as the class name – so you can run as `%java fragment1` for a working program that contains the first code fragment.
(6) Repeat Step (5) for other code fragment, e.g., `fragment2, fragment3,`...

A student will not receive full credits on this problem if the student fails to follow all steps listed in above.

1.8 (15 Points) Suppose you need to generate a random permutation of the first N integers. For example, `{4, 3, 1, 5, 2}` and `{3, 1, 4, 2, 5}` are legal permutations, but `{5, 4, 1, 2, 1}` is not, because one number (1) is duplicated and another (3) is missing. This routine is often used in simulation of algorithms. We assume the existence of a random number generator, `r`, with method `randInt(i, j)`, that generates integers between i and j with equal probability. Here are three algorithms:

1. Fill the array `a` from `a[0]` to `a[n−1]` as follows: To fill a[i], generate random numbers until you get one that is not already in `a[0], a[1], . . . , a[i-1]`.
2. Same as algorithm (1), but keep an extra array called the `used` array. When a random number, `ran`, is first put in the array `a`, set `used[ran] = true`. This means that when filling `a[i]` with a random number, you can test in one step to see whether the random number has been used, instead of the (possibly) i steps in the first algorithm.
3. Fill the array such that `a[i] = i + 1`. Then

```
for( i = 1; i < n; i++ )
     swapReferences( a[ i ], a[ randInt( 0, i ) ] );
```

a. Prove that all three algorithms generate only legal permutations.
b. Give as accurate (Big-Oh) an analysis as you can of the expected running time of each algorithm.
c. Write (separate) programs to execute each algorithm 10 times, to get a good average. Run program (1) for N = 250, 500, 1,000, 2,000; program (2) for N = 25,000, 50,000, 100,000, 200,000, 400,000, 800,000; and program (3) for N = 100,000, 200,000, 400,000, 800,000, 1,600,000, 3,200,000, 6,400,000.
d. Compare your analysis with the actual running times.

How to Submit:

All analysis of algorithms including Big-Oh, comparison analysis, and worst-case analysis list in above must be submitted with other written questions in the same `hw1.pdf` file. Follow these additional requirements for Java program submission:

(1) Create the folder "`permutation`" that contains all required `.java` file(s).
(2) The folder should be part of the `hw1.zip` file that you upload to Canvas.
(3) Do not include any other file types inside the `fragments` folder except `.java` file(s).
(4) Do not declare and use any "`package`" in your `.java` file(s).
(5) Use "`permutation1`" as the class name – so you can run as `%java permutation1` for a working program that contains the first algorithm.
(6) Repeat Step (5) for other two algorithms, e.g., `permutation2` and `permutation3`.

A student will not receive full credits on this problem if the student fails to follow all steps listed in above.

Note: For Algorithm 3 in Problem 1.8, you must write your own `swapReferences()` method similar to the one shown below. You will need to modify the code in order to use with Algorithm 3.

```
/**
 * Method to swap to elements in an array.
 * @param a an array of objects.
 * @param index1 the index of the first object.
 * @param index2 the index of the second object.
 */
public static <AnyType> void swapReferences( AnyType [ ] a, int index1, int index2 )
{
    AnyType tmp = a[ index1 ];
    a[ index1 ] = a[ index2 ];
    a[ index2 ] = tmp;
}
```