Basic Algorithms (V22.0310); Fall 2005; Yap

HOMEWORK 1

Date Due: Sep 21

Instructions

- Please read our homework policy under Course Information, in class homepage.
- There are two ways to submit homeworks:
  (1) Non-programs must be handed in as hardcopies. These are due during class hours, and it should be properly bound (stapled) together.
  (2) Programs must be handed in electronically, in ONE single email. These are due by midnite of the due date. Please send them directly to the grader, but cc'ed to me. Your grader is Ankit Sunil Malpani (ankit.malpani@nyu.edu). If you have to change some files, please resend the ENTIRE set in one email (we will just delete the earlier email).
- GENERAL RULE ABOUT PROGRAMMING ASSIGNMENTS: you must submit a Makefile to accompany your program. There are links on our homepage to resources for Java and Make programs. We should be able to type "make" to compile your program, and type "make test" to run the program. We normally tell you what to name your main program (e.g., "Matching.java") because this simplifies our grading task.
- GRADING RULES: If your program does not compile, you get 0 point. But please read the "Grader Instruction" in the class homepage for the general rules.

1 Question 1

(5 points) Please do Exercise 1, page 23, chapter 1. True or false: "In every instance of the Stable Matching Problem, there is a stable matching containing a pair (m, w) such that m is ranked first by w, and w is ranked first by m."

2 Question 2

(5 points) Please do Exercise 2, page 23, chapter 1. True or false: "Consider an instance of the Stable Matching Problem in which there exists a man m and a woman w such that m ir ranked first by w, and w is ranked first by m. Then (m, w) must appear in every stable matching."

3 Question 3

(5 points) Please do Exercise 3, page 23, chapter 1. We have two television networks, A and B. Network A wants to schedule a set $\{a_1, \ldots, a_n\}$ of shows into n time slots. Network B also wants to schedule a set $\{b_1, \ldots, b_n\}$ of shows into the same n slots. Let $\rho : \{a_1, \ldots, a_n, b_1, \ldots, b_n\} \to \mathbb{R}$ denote the ratings of each show (no 2 shows have the same rating). If a_i and b_j are scheduled to the same slot, then we say network A wins this slot if $\rho(a_i) > \rho(b_j)$; otherwise network B wins this slot.

Does there exist a pair of schedules for networks A and B that is stable in the following sense: no network can unilaterally change its scheduling to win more slots.

4 Question 4

(5 points) Consider the following statement:

$$(\forall x \in \mathbb{Z})(\exists y \in \mathbb{R})(\exists z \in \mathbb{R}) \left[(x > 0) \Rightarrow ((y^2 < x < y) \wedge (z < x < z^2) \wedge (z < y))\right] \tag{1}$$

Note that the range of variable x is $\mathbb{Z}$, not $\mathbb{R}$. This is called a "universal statement" because the leading quantifier is the universal quantifier ($\forall$). Similarly, we have "existential statements".

(i) Negate the statement (1), and then apply De Morgan's law to rewrite the result as an existential statement.

HINT: all our formal statements have the form $(Q_1)(Q_2)\cdots(Q_n)\,[\ldots predicate \ldots]$ where Q_i is the ith quantifier part. A predicate is a function from some domain to the set of Boolean values (true or false). E.g., the predicate "x is married" might be written as the function $Married(x)$ whose domain is the set of humans; $Married(x)$ is true iff x is married.

(ii) Is the statement (1) true? Justify it or give a counter example.

5 Question 5

(5 points) We provide the following program MergeSort.java, and an accompanying Makefile for you. The MergeSort program is mostly written for you – you just need to fill in the "merge" routine. The purpose of this question is to (a) introduce you to the Makefile program, (b) ensure that you have a basic Java programming environment.

```
/* MergeSort.java
 * Basic Algorithms, V22.0310.001, Spring 2001
 *
 * Merge sort is a popular recursive sorting algorithm.
 * In this program we implement merge sort to operate on integers.
 * Another purpose of this program is to illustrate simple
 * techniques used in Java.
 *
 * You can specify as a command line argument the number of elements
 * that you wish to sort. Then an array of the specified size is
 * populated by random numbers and finally sorted.
 *
 * If you put this file in a file called MergerSort.java,
 * then you can compile and run this program as follows:
 *
 *       % javac MergerSort.java
 *       % java MergerSort 100
 *
 * In this case you will be sorting 100 numbers. The output (the list of
 * sorted numbers) is sent to a data file called "output" that resides in the
 * current directory.
 */

class MergeSort
{
    /**
     * @param  num  an array of numbers in any order
     * @return      a sorted array of the same numbers
     **/
    static int[] mergeSort(int num[])
    { int size = num.length;  // size of array holding the numbers to be sorted
        if (size <= 1)  // base step
{   return num;
}
else  // recursive step
{   int mid = size / 2;
    int firstHalf[] = new int[mid];
    int secondHalf[] = new int[size - mid];
    // Copy first half of the array
    for (int i = 0; i < mid; i++)
    { firstHalf[i] = num[i];
    }
    // Copy second half of the array
    for (int i = mid; i < size; i++)
    { secondHalf[i - mid] = num[i];
    }
    return merge(mergeSort(firstHalf), mergeSort(secondHalf));
}
    }

    /**
     * @param  first   array of sorted numbers
     * @param  second  another second array of sorted numbers
     * @return         a sorted array containing all numbers from both first and second
     **/
    static int[] merge(int first[], int second[])
    {
// First, create a new array to store the answer. You
// can say something like:
// int res[] = new int[first.length + second.length];

// Then traverse through the arrays and compare their
// elements one by one and insert them in the new
// array (res) in sorted order.

// At last return the sorted array
// return res;
    }

    /**
     * @param   num    array which is to be populated with randomly-generated integers
     **/
    static void populate(int num[])
    { for (int i = 0; i < num.length; i++)
{   num[i] = (int) (Math.random() * 1000);  // numbers will be in the range 1-1000
}
    }

    /**
     * @param    num    array whose elems are to be sent to output
     **/
    public static void output(int num[])
    { java.io.PrintWriter out;
try
    {   out = new java.io.PrintWriter(new java.io.FileOutputStream("output"), true);
    for (int i = 0; i < num.length; i++)
    {   out.println(num[i]);
    }
}
        catch (java.io.IOException ioe) {ioe.printStackTrace(); }
    }

    /**
     * @param    args  command-line argumants. In this case number of elems
     *                 to be srted is expected.
     **/
    public static void main(String[] args)
    { int numbers[] = new int[Integer.parseInt(args[0])];
populate(numbers);
long start = System.currentTimeMillis();
int res[] = mergeSort(numbers);
long end = System.currentTimeMillis();
output(res);
System.out.println((end - start) + " ms");
    }
} // end class MergeSort
```

HERE IS THE BASIC ”Makefile”: to use it, just type ”make” to compile MergeSort.java and type ”make run” to run the MergeSort program.

```
# Sample Makefile
#    for Basic Algorithms Class
##########################################

##########################################
# define the variables p and arg:

p=MergeSort
arg=100

##########################################
# First target (the default target):

c compile: $(p).java
javac $(p).java

##########################################
# To run the compiled program:

r run: $(p).class
java $(p) $(arg)

##########################################
```