Problem B: Lenny's Lucky Lotto Lists (taken from 2004 North Central Regional)

Lenny likes to play the game of lotto. In the lotto game, he picks a list of N unique integers in the range from 1 to M. If his list matches the list of N integers that are selected randomly, he wins.

Lenny has a scheme that he thinks is likely to be lucky. He likes to choose his list so that each integer in it is at least twice as large as the one before it. So, for example, if N = 4 and M = 10, then the possible lucky lists Lenny could like are:

1 2 4 8
1 2 4 9
1 2 4 10
1 2 5 10

Thus, Lenny has four lists from which to choose.

Your job, given N and M, is to determine from how many lucky lists Lenny can choose.

Input File Format

There will be multiple cases to consider. The input for each is a pair of integers giving values for N and M, in that order. You are guaranteed that $1 \leq N \leq 10$, $1 \leq M \leq 2000$, and $N \leq M$. The input for the last case will be followed by a pair of zeroes.

Output Format

For each case display a line containing the case number (starting with 1 and increasing sequentially), the input values for N and M, and the number of lucky lists meeting Lenny's requirements. The desired format is illustrated in the sample shown below.

Sample Input (lotto.in)

```
4 10
2 20
2 200
0 0
```

Sample Output

```
Case 1: n = 4, m = 10, # lists = 4
Case 2: n = 2, m = 20, # lists = 100
Case 3: n = 2, m = 200, # lists = 10000
```

Deliverables

Please turn in your two solutions to these problems, **typedist.java** and **lotto.java**, respectively on WebCourses. Make sure both of these programs read from typedist.in and lotto.in, respectively.

Summer 2010 COP 3503 Program #4 Grading Criteria
Total Points: 100

Part A: Typing Distance

Programming style: 5 pts (use of white space, variable names, indenting, commenting)

Test cases: 10 test cases, 3 pts each. (30 pts total)

Points for code: Set up array for dynamic programming, etc. (5 pts)
Had three cases in the calculation for each case (5 pts)
Had a reasonable approach to calculating the distance btw keys (5 pts)

Part B: Lotto

Programming style: 5 pts (use of white space, variable names, indenting, commenting)

Test cases: There are 10 test cases, three points a case, all or nothing per case.

Points for code: Set up array for dynamic programming or memoization (5 pts)
Uses long or BigInteger (5 pts)
Attempts to set up some sort of DP formula or uses memoization (5 pts)

Note: This is set up so that if a student used int instead of long or BigInteger, they'll get a maximum score of 80. They will miss 5 test cases (the last five) for 15 points off and they'll lose 5 points for not using long or BigInteger. This double jeopardy is intentional because it's possible a student used long but didn't get the correct answer to all test cases. A student in this latter group has the opportunity to earn some points that a student who used int doesn't.

COP 3503 Summer 2010 Programming Assignment #1
Assigned: Tuesday, 5/18/10
Due: Thursday, 5/27/10

Problem: Maximum Sum (adaptation of acmuva problem #108)

The Problem

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the maximal sub-rectangle. A sub-rectangle is any contiguous sub-array of size 1x1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

is in the lower-left-hand corner:

9	2
-4	1
-1	8

and has a sum of 15.

The Input

The input consists of several NxN array of integers. The first line of the input is an single positive integer k, signifying the number of test cases. Each test case will follow. The first line of each test case will contain a single positive integer N ($0 < N < 500$) indicating the size of the square two-dimensional array. This is followed by N lines that contain N integers each separated by white space. The k^{th} line of input contains the values of the k^{th} row of the array, in order. The numbers in the array will be in the range [-127, 127]. ***The input will be from the file "sum.in"***

The Output

You will output a single line for each test case. The line will follow the following format:

```
Test case #i: The maximal sum is X.
```

where i represents the test case being processed, and X represents the maximal sum of that test case. ***The output should be to the screen.***

Sample Input File

```
3
2
3 -1
-5 2
3
6 -2 3
5 1 -7
8 9 -2
4
0   -2    -7   0
9    2    -6   2
-4   1    -4   1
-1   8    0   -2
```

Sample Corresponding Output

```
Test case #1: The maximal sum is 3.
Test case #2: The maximal sum is 27.
Test case #3: The maximal sum is 15.
```

Theoretical and Experimental Analysis

You will analyze your code to determine a theoretical run-time and verify this work by doing an experimental analysis of your implementation as shown in class. Do this work in a separate Word document. First include the theoretical analysis, highlighting the relevant parts of your code. Then, provide a chart with appropriate experimental run-times for various values of N, the value of one dimension of the input array. Use this chart to draw a conclusion about the experimental run-time of your algorithm. After you are done with this part of the assignment, edit the code so it runs according to the sample input and output listed above.

Grading Details

Using n as defined above, there are algorithms that solve this problem with the following running times: $\theta(n^6)$, $\theta(n^5)$, $\theta(n^4)$, and $\theta(n^3)$. Your program will be graded based on its correctness AND efficiency. A correctly running program with a run-time of $\theta(n^6)$ will earn a maximum of 70 points. A correctly running program with a run-time of $\theta(n^5)$ will earn a maximum of 85 points. A correctly running program with a run-time of $\theta(n^4)$ will earn a maximum of 100 points. If you find the $\theta(n^3)$ algorithm, you may earn some extra credit. But, remember that correctness is most important. You will lose points for all incorrect test cases.

What to turn in

Turn in your code in a single file called ***maxsum.java*** over WebCourses.
Turn in both your theoretical and experimental analysis in a file called ***Analysis.doc*** over WebCourses. (You may turn in a .txt file instead if you prefer.)

CS2 Summer 2010 Homework #1 Grading Criteria

100 points total

Code Points: 30 points (0 points for the n^6 algorithm, 15 points for the n^5 algorithm, 30 points for n^4 or better)

Correctness: 40 points (Run 10 test cases, 4 points each.)

Documentation and Style: 10 points – 2 for header comment, 2 for use of white space, 2 for good variable names, 2 for good indenting, 2 for comments in code

Gathering data: 10 points: Take off 3 points if they have 0 ms for anything. They should have at least 5 data points. Take off 2 points for each "missing" data point.

Analysis: 10 points: To get full credit here, they need to utilize the method prescribed in the notes. It's fine if their experimental data doesn't support the theoretical outcomes. But, they just state this and give possible explanations to get full credit. Decide partial credit.

Computer Science II
Program 2: Printer Priority Queue
Consult WebCourses for the due date and late cut-off date

The Problem

Your boss doesn't like waiting for printouts. He has asked you to rewrite the printer queue for him so that if necessary, he can "pull rank" and get his printouts before someone else who might have submitted a job earlier. Of course, if you do a good job on this project, he'll allow you to put submissions into the queue with high priority also.

In essence, he wants you to implement a priority queue. When each job is inputted into the printer queue, it will be paired with a priority number. The lower this number, the higher the priority for the job will be. Right after the printer finishes one job, it will move to the job with the lowest priority number of those that are waiting. If there are multiple such jobs, it will simply take the one that was requested first. Since the printer queue runs on a sequential processor, no two requests ever occur at the same time, so this tie breaker is sufficient in determining which job to print next.

The number of seconds the printer takes to process a single print job is 2 plus the number of pages in the job. (The two extra seconds are the "extra" time to get the job ready for its specifications before the actual printing begins.)

Input File Specification (printer.txt)

The input file has a single positive integer, *n*, on its first line, specifying the number of printer queue scenarios in the input file.

The first line of each printer queue case will have a positive integer, *r (r < 1001)*, where *r* represents the number of printer requests. The next *r* lines will have information about each printer request in the order in which it was made, with no two requests occurring at the exact same time.

Each of the input lines will have the following format:

```
TIME DOC_NAME PRIORITY PAGES
```

`TIME` represents the number of seconds after the simulation has begun that the request was put in. This will be non-negative and not exceed 28800. (A job may finish after this time.)

`DOC_NAME` is a string with no white space representing the name of the document to be printed. (Each of these will be distinct, within a single test case.)

`PRIORITY` is a positive integer in between 1 and 100, inclusive, representing the priority of this job, with 1 representing the most important.

`PAGES` is the number of pages in the print job. It's guaranteed to be a positive integer less than 1000.

Output Specification
For each printer scenario, print out a header with the following format:

```
Printer #k:
```

where k represents the day of the simulation ($1 \leq k \leq n$).

Follow this with a blank line.

For each print job, print out a single line with the following format:

```
DOC_NAME completed printing at time TIME.
```

Print out these r lines in the order the jobs finished, ie, in increasing order of TIME.

Implementation Restrictions
Information about the print jobs must be stored in a heap. Your code should have a Heap class that just manages the necessary heap operations. A separate class should be used to run the program.

Sample Input File

```
2
10
3 Program2.doc 10 5
6 Resume.doc 5 2
7 FantasyFootball.xls 4 20
15 Sudoku.java 8 10
17 PhoneBook.xls 8 20
30 Program3.doc 10 6
31 Book.doc 1 200
100 Grades.xls 3 5
101 Recitation2.doc 7 2
1000 directions.txt 9 1
5
215 a.txt 1 100
231 b.txt 1 200
244 c.txt 1 100
312 d.txt 1 100
1000 e.txt 1 500
```

Sample Output

```
Printer #1:

Program2.doc completed printing at time 10.
FantasyFootball.xls completed printing at time 32.
Book.doc completed printing at time 234.
Grades.xls completed printing at time 241.
Resume.doc completed printing at time 245.
Recitation2.doc completed printing at time 249.
Sudoku.java completed printing at time 261.
PhoneBook.xls completed printing at time 283.
Program3.doc completed printing at time 291.
directions.txt completed printing at time 1003.

Printer #2:

a.txt completed printing at time 317.
b.txt completed printing at time 519.
c.txt completed printing at time 621.
d.txt completed printing at time 723.
e.txt completed printing at time 1502.
```

Making Data

In addition to your solution, you need to make your own data file. Make sure your data file adheres to the specifications in this document. Also, make a thorough set of test cases. Try to come up with as many significantly different scenarios as possible. Remember to test both minimum and maximum cases and any unusual cases you can think of. **In addition to being 30% of the grade, the person who creates the most comprehensive (in my opinion) data file will win $20.**

Deliverables

Turn in the following over WebCourses:

1) The set of .java files that solves this problem. (Note: My solution has three java files: Heap.java, PrintJob.java and PrintQueue.java.)
2) A file with your test cases named printer.txt.

As always, make sure to include ample comments and use good programming style, on top of the requirements that are given in the program description above.

CS2 Summer 2010 Homework #2 Grading Criteria

100 points total

Code Points(20 total pts): 10 points for declaring a heap class with a reasonable set of methods, 10 points for observing good object-oriented coding practices.

Test Cases(40 total points): 4 points per test case –

You may give partial credit on a test case that isn't 100% correct.

If a case crashes, then 0 points are earned automatically for the rest of the test cases. Also, if anything crashes there is an automatic 15 point deduction. (So, even if cases 1-9 are correct and case 10 crashes, then the score here will be a 25.)

There are two possible outputs that could be considered correct, depending on a detail I didn't put in the problem specification. Compare the student's output with both and give them the score that is better between the two.

Documentation and Style: 10 points – 2 for header comment, 2 for use of white space, 2 for good variable names, 2 for good indenting, 2 for comments in code

Test cases (30 points): 3 points for satisfying the following criteria

1. A test case with exactly 1 print job
2. A test case with 1000 print jobs
3. One test case that has all the priorities the same, to isolate other issues
4. A case where there's a job in the queue, then it's empty for a while, then there's another job.
5. A case where all the jobs go in the queue (at least 5) before any job finishes
6. A case with a time close to 28000.
7. A case with somewhat random data.
8. A case with every possible priority (1-100) of print job.
9. All cases fall within the given specifications.
10. The full range of pages (anywhere from 1 to 999) is tried.

COP 3503 Program #3
Assigned: 6/22/10 (Tuesday)
Due: 7/1/10 (Thursday)

Problem: Railroad Building

You have been given the daunting task of designing a railway system between a set of cities. Your goal is to minimize the cost of the tracks laid. Given information about the cost of building railway tracks between each pair of cities that needs to be connected, you must determine one set of tracks to build that connects each city to one another at minimal cost.

Implementation Details

You must implement Kruskal's algorithm to solve the problem. Furthermore, you must implement a Disjoint Set to aid in cycle detection during the algorithm. Also, rather than sorting all of the edges, please either write your own Heap class **or use your Heap class from assignment #2** and create a heap out of all of the edges and extract the minimum edges from this heap as needed. Thus, you must turn in at least three files: *Railroad.java, DisjointSet.java, and Heap.java*. Please implement both the Disjoint Set and Heap using an array. Also, if you need to sort the edges to satisfy the output requirements, you may use Java's prewritten sorting method.

Your program will read in input from the file ***"railroad.in"*** and then output each corresponding solution to the screen.

Input File Format

The first line of the input file will contain a single positive integer *n*, representing the number of test cases in the file. Each test case will follow, one by one. For each test case, the first line will contain a single positive integer *m,* representing the number of possible tracks to build. The following *m* lines will contain information about one possible track each. Each of these lines will contain three pieces of information separated by spaces: two cities, each strings, and a positive integer representing the cost of building, in dollars, railroad tracks in between those two cities. All city names will only contain uppercase alphabetic characters and underscores. You will be guaranteed that the data will describe a set of cities that are all connectable with the set of possible tracks given.

Output Format

For each test case, the first line of output will be of the following format:

```
The minimum cost of the railway system is X.
```

where X represents the minimum cost (in dollars) of connecting all of the cities in the input case.

The following lines will each contain information about one railroad track that should be built. The number of lines will always be one less than the total number of cities in the input case. The information on each line should follow the following format:

```
CITY1 CITY2 COST
```

where CITY1 comes before CITY2 alphabetically, and COST is the cost (in dollars) to build railroad tracks to connect CITY1 and CITY2. All three components are separated by a single space.

Furthermore, the output should be listed in alphabetical order by the first city listed. If the first city listed is the same, then the tie should be broken by the second city listed alphabetically. Thus, if there should be tracks built between ORLANDO and TAMPA, and ORLANDO and PALM_BAY, then ORLANDO and PALM_BAY should be listed before ORLANDO and TAMPA.

Also, separate the output for each case with a blank line.

Sample Input File

```
2
4
A B 10
D B 5
B C 6
A C 8
6
ORLANDO TAMPA 100
MIAMI JACKSONVILLE 300
TAMPA MIAMI 275
MIAMI ORLANDO 230
JACKSONVILLE TAMPA 190
ORLANDO JACKSONVILLE 120
```

Sample Output

```
The minimum cost of the railway system is 19.
A C 8
B C 6
B D 5

The minimum cost of the railway system is 450.
JACKSONVILLE ORLANDO 120
MIAMI ORLANDO 230
ORLANDO TAMPA 100
```

COP 3503 Summer 2010 Program #3
Grading Criteria

Documentation (10 pts)
Appropriate header comment - 2 pts
Comments in code - 4 pts
Use of White Space - 1 pts
Appropriate variable names - 2 pt
Appropriate Indentation - 1 pts

Implementation Details (30 pts)
A reasonable DisjointSet class is included:
- Constructor (2 pts)
- Find operation (5 pts)
- Union operation (5 pts)
- Union of the shorter tree with the larger one (3 pts)

A reasonable Heap class is included:
- Constructor (2 pts)
- percolateDown function (5 pts)
- MakeHeap function (3 pts)
- DeleteMin function (4 pts)

Code Points (30 pts)
Basic overall input/output is set up correctly (5 pts)
Heap of edges is formed and DeleteMin is called appropriately. (10 pts)
For each edge extracted, the appropriate Union function on the DisjointSet is called, if no cycle is formed. (10 pts)
For each edge extracted, if a cycle is formed, no Union is done and the edge is not added. (10 pts)

Execution Points (30 pts)
Use the 6 sample cases worth 5 points each in the attached file. If all cases return a correct MST but not in the proper order, then deduct 5 points from them out of 30. If only one or two cases have the correct MST but not in the proper order, then deduct 2 or 4 points, respectively. If a case does not have the correct MST weight, then it's automatically worth 0 points.

COP 3503 Homework #5
Assigned: 7/22/10 (Thursday)
Due: 8/3/10 (Tuesday 11:55pm WebCourses time, NO LATE ASSIGNMENTS)

Sudoku Puzzles (Call your program sudoku.java)

Sudoku Puzzles have recently caught on as a hot new item amongst games in newspapers. The game is as follows:

1) You are given a 9x9 grid, with some squares filled in with positive integers in between 1 and 9, inclusive.

2) Your goal is to complete the grid with positive integers in between 1 and 9, inclusive, so that each row, column and mini 3x3 square that is designated contain each integer in the range 1 through 9 exactly once.

Below is an example of a Sudoku puzzle.

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

(taken from www.sudoku.com on 11:45am on 2/1/06)

Here is the puzzle solved:

9	6	**3**	1	**7**	4	**2**	5	**8**
1	**7**	8	3	**2**	5	6	**4**	**9**
2	**5**	**4**	**6**	**8**	**9**	**7**	**3**	1
8	**2**	**1**	4	**3**	7	**5**	**9**	6
4	**9**	6	**8**	**5**	**2**	3	**1**	**7**
7	**3**	**5**	9	**6**	1	**8**	**2**	4
5	**8**	**9**	**7**	**1**	**3**	**4**	**6**	2
3	**1**	7	2	**4**	6	9	**8**	**5**
6	4	**2**	5	**9**	8	**1**	7	**3**

Your program will read in a file of sudoku puzzles, all of which either have no solution or a unique solution. You will output to the screen for each test case, either the filled in game board, or the statement, "No solution possible."

Input File Format

The first line of the input file will be a positive integer *n*, representing the number of puzzles to solve in the input file. The following 9*n* lines will contain the n cases, with each case taking exactly 9 lines. Within each test case, the i^{th} line will contain the 9 values on the i^{th} row of the unsolved puzzle, in order, from left to right. In particular, all blank entries of the puzzle will be indicated with a 0, and the 9 integer values (all in between 0 and 9, inclusive) on the line each will be separated by a space.

Output Format

The first line for each test case will be of the format:

```
Test case k:
```

where k ranges in between 1 and n, inclusive and represents the test case number.

If the puzzle has no solution, then the second line of output will be:

```
No solution possible.
```

If the puzzle has a solution, output it on the following 9 lines, with each value separated by one space. Put a blank line between cases. For the puzzle above the output would be:

```
9 6 3 1 7 4 2 5 8
1 7 8 3 2 5 6 4 9
2 5 4 6 8 9 7 3 1
8 2 1 4 3 7 5 9 6
4 9 6 8 5 2 3 1 7
7 3 5 9 6 1 8 2 4
5 8 9 7 1 3 4 6 2
3 1 7 2 4 6 9 8 5
6 4 2 5 9 8 1 7 3
```

Sample Input File

```
1
0 6 0 1 0 4 0 5 0
0 0 8 3 0 5 6 0 0
2 0 0 0 0 0 0 0 1
8 0 0 4 0 7 0 0 6
0 0 6 0 0 0 3 0 0
7 0 0 9 0 1 0 0 4
5 0 0 0 0 0 0 0 2
0 0 7 2 0 6 9 0 0
0 4 0 5 0 8 0 7 0
```

COP 3503 Summer 2010 Homework #5 Grading Criteria

1. Properly Reads In File (10 points)
2. Uses all values in a row are distinct, all values in a column are distinct and all values in a 3x3 box are distinct to reduce work. (10 points)
3. Utilizes some form of backtracking. (10 points)
4. 11 test cases, 5 points a piece. (55 points) – if you think partial credit is possible to assign on a single case, do so.
5. Comments, Use of White space, Good consistent coding style (15 points)

COP 3503 Program #4: Dynamic Programming
Assigned: Tuesday, 13, 2010
Due: Thursday, July 22, 2010

Problem A: Typing Distance

Your employer has recently been dissatisfied with the quality of the typing of her employees. There are many misspellings that she ends up having to correct, which ultimately waste her time. She wants you to help her develop a test for future employees that accurately judges their typing accuracy. In particular, she has defined a term "typing distance" which is defined between two strings(or words): a source word and a target word. (We can think of the target word as the correct word to type and the source word as the word that actually got typed.) The definition of "typing distance" is similar to the definition of "edit distance" you learned in CS2 class years ago, so you think that you can quickly write a program to determine the "typing distance" between two strings. In particular, the typing distance between two strings is the minimal score that can be achieved in changing the source string to the target string. The changes that are allowed are as follows:

1) Changing a letter in the source to a letter in the target string.
2) Deleting a letter from the source string.
3) Inserting a letter into the target string.

The score added to the total score for options 2 and 3 is 5. The score added to the total score for option 1 is the distance the two keys in question are away from each other on the keyboard. The distance between two keys on the keyboard is defined the number of keys one must traverse to get from one key to another, when a valid move is to one of the six adjacent keys. (Note that there is potentially one key to the left, one to the right, one key to the upper left, one key to the upper right, one key to the lower left and one key to the lower right.) Of course, some keys have fewer adjacent keys. For example, q only has 2 adjacent keys, w and a. Here is a copy of the keyboard:

```
q w e r t y u i o p
 a s d f g h j k l
  z x c v b n m
```

Here are a couple more examples: the distance from d to y is 3 (d→r→t→y) and the distance between r and m is 5 (r→t→y→h→n→m). Note that the paths in the parentheses are not unique.

Write a program that calculates the typing distance between two strings.

Input File Format

The first line of the input file will contain a single positive integer, *n*, representing the number of pairs of strings in the file. The following *n* lines will contain a pair of strings, consisting of lowercase letters only, representing the source string and target string, respectively, separated by a single space. Both strings will be in between 1 and 100 letters, inclusive.

Output Format

For each test case, output a line with the following format:

```
Test k: The typing distance is Z.
```

where k is the 1-based input case number, and Z is the typing distance between the two given input strings for that test case.

Sample Input (typedist.in)

```
2
dkunr drunk
cat vat
```

Sample Output

```
Test 1: The typing distance is 10.
Test 2: The typing distance is 1.
```

Summer 2010 COP 3503 Program #4 Grading Criteria
Total Points: 100

Part A: Typing Distance

Programming style: 5 pts (use of white space, variable names, indenting, commenting)

Test cases: 10 test cases, 3 pts each. (30 pts total)

Points for code: Set up array for dynamic programming, etc. (5 pts)
Had three cases in the calculation for each case (5 pts)
Had a reasonable approach to calculating the distance btw keys (5 pts)

Part B: Lotto

Programming style: 5 pts (use of white space, variable names, indenting, commenting)

Test cases: There are 10 test cases, three points a case, all or nothing per case.

Points for code: Set up array for dynamic programming or memoization (5 pts)
Uses long or BigInteger (5 pts)
Attempts to set up some sort of DP formula or uses memoization (5 pts)

Note: This is set up so that if a student used int instead of long or BigInteger, they'll get a maximum score of 80. They will miss 5 test cases (the last five) for 15 points off and they'll lose 5 points for not using long or BigInteger. This double jeopardy is intentional because it's possible a student used long but didn't get the correct answer to all test cases. A student in this latter group has the opportunity to earn some points that a student who used int doesn't.