Test 1: Compsci 100

Owen Astrachan

October 1, 2008

Name: _____ (3 points)

Login: _____

Honor code acknowledgment (signature) _____

	value	grade
Problem 1	30 pts.	
Problem 2	16 pts.	
Problem 3	16 pts.	
Problem 4	10 pts.	
TOTAL:	75 pts.	

This test has 12 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 75 minutes.

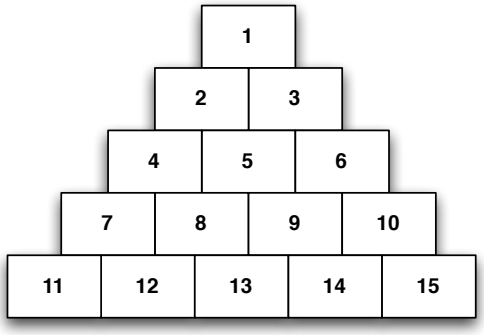
In writing code you do not need to worry about specifying the proper `import statements`. Assume that all libraries and packages we've discussed are imported in any code you write.

Unless indicated otherwise, here's the `Node` class for this test.

```
public class Node{
   public String info;
   public Node next;
   public Node(String s, Node link){
       info = s;
       next = link;
   }
}
```

PROBLEM 1 : (***Aztec, Mayan (30 points)***)

The picture below shows a five-rowed, two-dimensional pyramid constructed of rectangles – this will be called a *5-pyramid* in this problem because it has five rows. The top rectangle is number one, then the rectangles are numbered left-to-right in a row and top-to-bottom as shown. In the fifth row there are five rectangles; in general in the N^{th} row there are N rectangles. In answering questions below assume we're discussing an N-pyramid with N rows for a large value of N. Assume each rectangle is centered on the two rectangles below it so that the vertical side of a rectangle is in the middle of the rectangle below it.



Part A (2 points)
What is the number of the right-most rectangle in the seventh row?

Part B (4 points)
What is the *exact* value of both the right-most and the left-most rectangle in the 100^{th} row?

Part C (2 points)

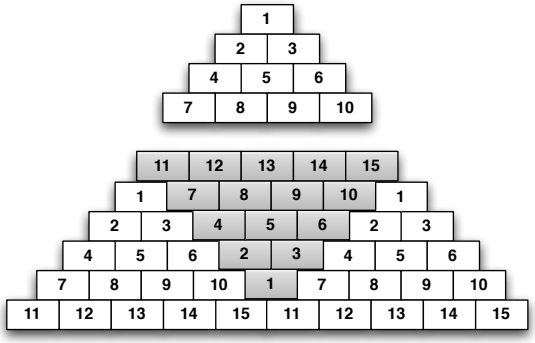
Using big-Oh what is the number of rectangles in the middle row of a pyramid with N rows (assume N is odd). Justify your answer.

Part D (4 points)

Using big-Oh what is the total number of rectangles in a pyramid with N^2 rectangles on the bottom row? Justify your answer.

Part D (4 points)

The diagram below shows three 5-pyramids and a 4-pyramids being combined to make a 10-pyramids. If three N-pyramids and one $(N-1)$-pyramid are similarly combined to make a large pyramid how many total rectangles will be in the resulting pyramid; use big-Oh and justify your answer.



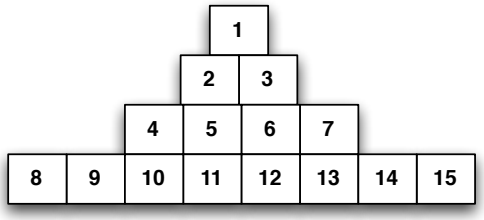
Part F (4 points)

Assume all rectangles in a N-pyramid are unit squares that is with sides equal to one foot. The perimeter of an N-pyramid is the distance around the top, bottom, and stair-shaped sides. Complete the Java method below to return the perimeter of an N-pyramid constructed of unit squares. Your method should work in $O(1)$ time for an N-pyramid.

```
public int perimeter(int n){




}
```

Part G (10 points)

Consider a new pyramid as shown below on the left. The number of rectangles doubles in each row so that as shown the first row contains one rectangle, the fourth row contains eight rectangles and in general the N^{th} row contains 2^{N-1} rectangles.



Part G.1 (2 points)

If a pyramid contains N rectangles in all its rows together, how many rectangles are in the bottom row using big-Oh? Justify your answer.

Part G.2 (2 points)

If there are N rectangles in the bottom row, how many rows are there using big-Oh? Justify your answer.

(continued)

Part G.3 (4 points)

Write method `rowNum` that returns the row of a rectangle given the rectangle's number. For example, `rowNum(13)` and `rowNum(15)` both return 4; `rowNum(18)` returns 5, and `rowNum(1000)` returns 10.

You can write a loop, use recursion, or use `Math.floor`, `Math.log10`, and `Math.log`. Note that `Math.floor` returns a double, but it's the greatest integer less than it's argument, i.e., `Math.floor(3.7)` is 3.0 as is `Math.floor(3.001)`. The log methods return the base-10 and natural log, respectively for `Math.log10` and `Math.log`. You may find this property of logs useful:

$$\log_b(x) = \frac{\log_d(x)}{log_d(b)}$$

```
public int rowNum(int rec) {




}
```

Part G.4 (2 points)

What is the big-Oh complexity for the running time of the call `rowNum(N)` for the method you wrote above? Justify your answr.

PROBLEM 2 : (***Mercator (16 points)***)

Part A (8 points) Write code to return the value that occurs most often in an array of strings. Your code should run in $O(n)$ time for an n-element array. Use a `HashMap` to keep track of how many times every string occurs since calls to `put`, `get`, and `containsKey` are $O(1)$ for a `HashMap`. Assume that the maximally occurring value is unique, i.e., there are no ties in determining which string occurs most often.

```
public String occursMost(String[] list) {

    HashMap<String,Integer> map = new HashMap<String,Integer>();









}
```

Part B (8 points)

A string in the form `"parent child"` describes a parent/child relationship. One space separates the parent name from the child name. In this problem assume every person has a different name, and only one parent is listed for any child, i.e., a name occurs only once as the second name in a parent/child string of the parameter `families` below. Consider this example:

```
"owen adam" "gail owen" "jane gail" "gail josh" "john jane"
```

Here, adam's parent is owen, owen's parent is gail, gail's parent is jane, and jane's parent is john. This makes john the person who is adam's oldest ancestor. Complete the method below so that it returns the oldest ancestor of `child` given the relationships in the array `families`. If `child` has no ancestor, e.g., like "john" or "fred" in the example above, return an empty string `""`.

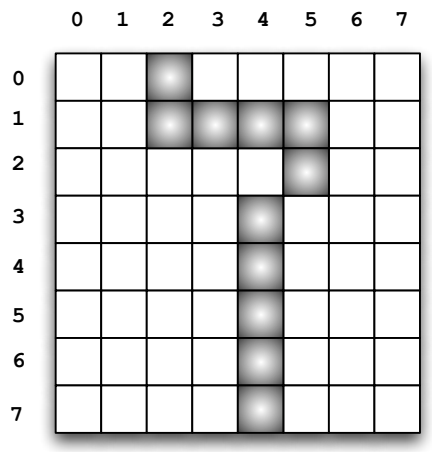
(hint: you can use a map in which the key represents a child and the corresponding value represents the key's parent)

```
public String oldestAncestor(String[] families, String child){




    for(String s : families){
        String[] pc = s.split(" ");



    }









}
```

PROBLEM 3 : (*Grid-iron (16 points)*)

The 8×8 grid below shows a nearly complete path from the top (row zero) to the bottom (row 7) of the grid. The path isn't complete because squares in a path must be connected horizontally or vertically.

Coloring the square at either (3,5) or (2,4) would result in a path from the top to the bottom.



In this problem the grid will be represented by a two-dimensional boolean array `myGrid`. If `myGrid[r][c]` is true the corresponding grid square (r,c) is shaded as in the diagram and can be part of a path. If `myGrid[r][c]` is false than the square is not shaded and so cannot be part of a path.

In the 8×8 grid pictured on the left there are 11 grid elements that would be true in the boolean array and 53 that would be false since 11 of 64 are shaded.

Part A (3 points)

In an $N \times N$ grid what is the minimum number of squares that can be shaded/true so that there is a path from the top to the bottom. Justify your answer.

Part B (3 points)

Is it possible for $O(N^2)$ squares to be shaded/true with no path from the top to the bottom in an $N \times N$ grid? Justify your answer.

Part C (2 points)

The method below sets the diagonal of a square grid so that all its elements are true. What is the big-Oh complexity of this method for an $N \times N$ grid? Justify your answer.

```
public void fill(boolean[][] grid){
    for(int r=0; r < grid.length; r++){
        grid[r][r] = true;
    }
}
```

Part D (8 points)

The code on the next page shows method `pathToBottom` that correctly returns true if and only if there is a path from grid position (r,c) to the bottom of the grid. You'll complete two methods from this class.

Part D.1 (4 points)

Complete method `pathFound` so that it returns true if and only if there is a path from the top row (row zero) to the bottom row.

```
public boolean pathFound(){
    myVisited = new boolean[myRows][myCols];
    // add code here









    return false;
}
```

Part D.2 (4 points)

Complete method `addOne` so that it returns true if and only if it is possible to set one grid value/cell to true and complete a path from the top to the bottom. Assume `pathFound` works correctly.

```
public boolean addOne(){









    return false;
}
```

```
import java.util.*;
public class Grid {

    private boolean[][] myGrid;
    private boolean[][] myVisited;
    private int myRows, myCols;

    public Grid(){
        myRows = 8;
        myCols = 8;
        myGrid = new boolean[myRows][myCols];
    }

    public boolean pathFound(){
        myVisited = new boolean[myRows][myCols];
        // add code here




        return false;
    }

    public boolean addOne(){
        // add code here
        return false;
    }

    private boolean pathToBottom(int r, int c){
        if (r < 0 || r >= myRows) return false;
        if (c < 0 || c >= myCols) return false;
        if (myVisited[r][c]) return false;

        myVisited[r][c] = true;
        if (! myGrid[r][c]) return false;

        if (r == myRows-1) {
            return true;
        }
        return pathToBottom(r-1,c) || pathToBottom(r+1,c) ||
               pathToBottom(r,c-1) || pathToBottom(r,c+1);

    }

}
```

PROBLEM 4 : (*Linkedin (10 points)*)

Using the `Node` class at the beginning of the test the code below prints the string values in a linked list on one line.

```
public void print(Node list){
    while (list != null){
        System.out.print(list.info+" ");
        list = list.next;
     }
     System.out.println();
 }
```

You'll write the method `twin` described below that duplicates each node of a linked list. The code below should generate the output beneath it when `twin` is written.

```
  public void linkTest(){
     Node list = new Node("child", new Node("dog", new Node("cat",  new Node("spit",null))));
     print(list);
     list = twin(list);
     print(list);
  }
```

Output:

child dog cat spit

child child dog dog cat cat spit spit

Part A (5 points)

Given a list of n nodes the code you write in `twin` should create n new nodes and return a pointer to the first node of a list containing $2n$ nodes as described above. Be sure you assign values to the `.next` fields of nodes in linking new nodes into the list.

```
   public Node twin(Node list) {









  }
```

(continued)

Part B (5 points)

Write the method `removeOther` that removes every other node from a linked-list and returns a pointer to the first node of the new list. For example the code below prints the original list, i.e.,

child dog cat spit

```
public void linkTest(){
   Node list = new Node("child", new Node("dog", new Node("cat",  new Node("spit",null))));
   list = twin(list);
   list = removeOther(list);
   print(list);
}
```

To remove a node from a linked list you'll need to use a line of code similar to the one below which links a node "around" the node after it. What you write should include code like this to change links so that nodes are removed from the list.

```
list.next = list.next.next;
```

Conceptually, if the nodes of a list are numbered $1, 2, 3, \ldots n$ your code removes all the nodes whose numbers are even, leaving only the odd numbered nodes from the original list. To remove a node you link around it so that nothing in the list points to it.

```
public Node removeOther(Node list) {









}
```

(nothing on this page)