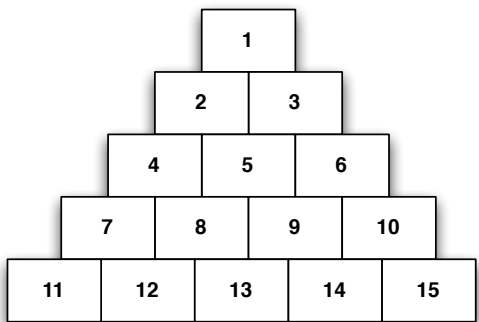


**PROBLEM 1 :** (*Aztec, Mayan (30 points)*)

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  $N^{\text{th}}$  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  $100^{\text{th}}$  row?

**Part C (2 points)**

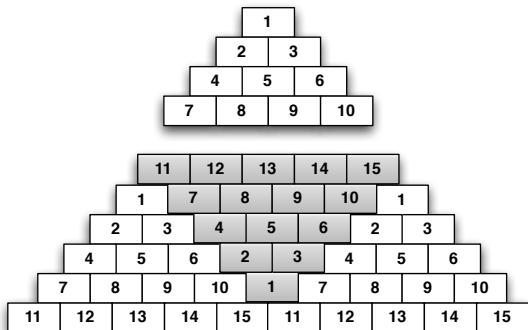
Using big-Oh what is the number of rectangles in the middle row of a pyramid with  $N$  rows (assume  $N$  is odd). Justify your answer.

**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.



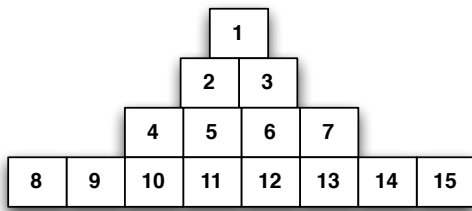
**Part F (4 points)**

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 answer.

**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(" ");
```

```
    }
```

```
}
```