The Josephus Problem works as follows.

(From Wikipedia: <https://en.wikipedia.org/wiki/Josephus_problem> )

“The **Josephus problem** (or **Josephus permutation**) is a theoretical problem related to a certain [counting-out game](https://en.wikipedia.org/wiki/Counting-out_game). …

The problem is named after [Flavius Josephus](https://en.wikipedia.org/wiki/Flavius_Josephus), a Jewish historian living in the 1st century. According to Josephus' account of the [siege of Yodfat](https://en.wikipedia.org/wiki/Siege_of_Yodfat)…” ( And see the rest in Wikipedia).

A simple (and one of the more gentle) descriptions of the problem is as follows:



Knights around a table. Choosing a king.

A bunch of men sit around in a circle. A court jester is asked to choose a king from them and must kill the rest. He decides to kill every other man. So, he spares the first man, kills the second, spares the next, kills the next, etc., going around the circle until there is only one man left alive who becomes the king.

In this assignment, you need to solve find the king using three different methods:

Use an ArrayList<Integer>and kill off every other man until there is one left Use binary arithmetic and circular shifting to identify the one man who will be left; Use floor(log2n)and some clever mathematics to identify the one man who will be left.

Here is a starter kit of code. First, a Main class that sets up a ArrayList<Integer>without the josephusmethod written:

public class Main { public static void main(String[] args) {

System.out.println(josephus(13)); //printsout11 }

} 1. Write a program that solves the Josephus problem using an ArrayList<Integer>. Here are some

recommended steps for a josephusmethod:

1. Create an ArrayList<Integer>. 2. WritepopulateList(ArrayList<Integer>,intn)that populates the ArrayList with the numbers 1..n. 3. Write a loop that deletes every other knight from the ArrayList until only one remains. 4. Return the value of the remaining element in the ArrayList.

2. Now, suppose we number the people at the table, starting at 1, and going around the circle to N. It turns out that if you know N, you can determine which person will survive using binary arithmetic. Here is a table to illustrate:

                 

**Number of contenders**

**Binary**

**Survivor position**

**Binary**

               

1

1

1

1

               

2

10

1

1

               

3

4

11

3

11

           

100

1

1

             

5

101

3

11

               

6

110

5

101

               

7

111

7

111

               

8

1000

1

1

               

9

1001

3

11

               

10

1010

5

101

               

11

1011

7

111

               

12

1100

9

1001

               

13

1101

11

1011

               

14

1110

13

1101

           

       

15

1111

15

1111

  

If you take the binary number for the number of people sitting at the table (contenders), convert it to binary, then remove the first 1 in the binary number and tack it onto the end of that binary number, you get the binary value for the surviving contender. For example, suppose there are 12 contenders. 12 is written as 1100 in binary. We remove the first 1 in 1100 to leave us 100. Then we tack it on the end and have 1001. Then we convert this to decimal and get 9. This moving of the 1 is called a circular left shift. We move all the digits to the left, pushing the leftmost one off the map and wrapping it around to the last position.

It is probably easiest to do this by treating binary numbers as Strings since you can use charAt()and substring()to take a string of 0s and 1s and produce the desired result.

2. Write a Josephus solver that uses binary arithmetic and circular left shifting to determine which contender will survive.

HINT: You may find Integer.toBinary()and Integer.parseInt(num, 2)to be extremely helpful. 3. Write a function that uses log2N and calculates the survivor. (There is a formula--it can be done!)