CMPSCI 187, Spring 2015 Discussion #8
A Guessing Game: Individual Handout

Marc Liberatore and John Ridgway
adapted from a version written by David Mix Barrington

23 March 2015

You may well have first encountered the idea of *binary search* with the following number guessing game: someone picks a number, say between 1 and 100 inclusive, and you make a series of guesses. After each guess you are told "Too high" or "Too low", until you reach the chosen number. You quickly learn that 50 is a good first guess and 3 is a bad one. The most sensible strategy to find the number appears to be to maintain a range of possible numbers that are consistent with the answers so far, and always make a guess that cuts that range into two equal halves (or as close as possible).

David Mix Barrington's father, a computer programmer since the 1960's, wrote the variant of this game that has been re-implemented in `GuessANumber.java`. The range will always be between 1 and N. Try playing it a few times with $N = 100$. We predict that you will lose every time.

> **Question 1:** Play ten games against the program with $N = 2$ and report your results. Explain why you can now be confident that the program is not choosing a random number at the beginning of the game, as it claims. (Prof. Barrington's father wanted to show him that computers are not to be trusted.)

In general, if you play using a good strategy, you should be within one guess of getting the answer. In fact, the program computes how many guesses you would need to be sure of getting the answer, then gives you one fewer guess than that.

> **Question 2:** If the number N is equal to $2^k - 1$ for some positive integer k, explain how you can guarantee to get the correct number in k guesses, even if the program is cheating like this one. (Assume that its "Too high" and "Too low" answers must be consistent with the number it finally "reveals".) If $k = 1, 2^k - 1 = 1$. If $k = 2, 2^k - 1 = 3$, if $k = 3$ it is 7, and so forth. A formal proof that your strategy is correct would take the technique of mathematical induction from CMPSCI 250, but you should be able to explain informally why it works.

Besides being (slightly) amusing, this program demonstrates a significant concept — a lower bound on the required number of guesses. If $N > 2^k - 1$, then no algorithm can find the number in the range from 1 to N, using at most k guesses, in the worst case. Worst-case lower bounds are often proved via an adversary argument — we design an adversary (who wants the algorithm to fail) who can observe the algorithm's behavior and find an input case on which it fails, thus showing that it can't succeed in all cases. Note that this adversary is limited to cheating in ways that cannot be observed by the player. In this case the number it finally chooses must be consistent with all of the answers it has given.

> **Question 3:** When $N > 2^k - 1$, the program can always find a number after k guesses that is consistent with its answers and not equal to any of the numbers you guessed. Explain how.