CMPSCI 187 Discussion #8: A Guessing Game
Individual Handout

David Mix Barrington
31 October 2012

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. say, is a good first guess and 3 is a bad one. The most sensible strategy to find the number in as few guesses as possible *appears* to be to maintain the range of possible answers that are consistent with the answers so far, and always make a guess that cuts that range into as equal pieces as possible.

My father, a computer programmer since the 1960's, wrote the variant of this game that I've coded up in `GuessANumber.java`. Try playing it a few times with $N = 100$. (We'll always call the initial range N.) I 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. (My father wanted to show me that computers are not to be trusted.)

If you played a good strategy, you should have been within one more 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. (But you may 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 be succeeding in *all* cases.

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