Assignment 05

Alex Clemmer
Student number: u0458675

The first thing we need is way to tell for certain whether a number is larger than the array size. One way to do this is to find the max array size allowed on a given platform for the given language and simply set that maximum to the `high` variable, with 0 as the `low` variable, and binary search using the given out-of-bounds exception as the comparator. I assume that's cheating.

The "obvious" legal way is to simply start with some marker `i = 1` and double it until we throw an out-of-bounds exception. We can then set `i` to be the `high` operator, and then just use binary search to find the end.

There are, of course, both a "good" and a "bad" way to do this binary search. The "good" way would be to save the last position of `i` every time we double it, and then when we actually do throw the exception, all we need to do is binary search in the bounds of the current position of `i` and the last position of `i`. This will always work out to be half the array.

From the perspective of asymptotic analysis, it doesn't matter much whether we binary search the whole array or not—they're both log n, and keeping track or not really only adds one comparison to the binary search. That is, we split the search space in half every time we run the loop, so starting with only half the array really only saves us one comparison.

Now, to the matter at hand. What is the running time? In order to find the upper bound we require log n. Binary searching requires log n time. Since they both require log n time and they're independent, we add them together for log n + log n. Thus the runtime asymptotically is log n.