> What's the relationship between the Best Conceivable Runtime and Best Case Runtime? Nothing at all! The Best Conceivable Runtime is for a *problem* and is largely a function of the inputs and outputs. It has no particular connection to a specific algorithm. In fact, if you compute the Best Conceivable Runtime by thinking about what *your* algorithm does, you're probably doing something wrong. The Best Case Runtime is for a specific algorithm (and is a mostly useless value).

Note that the best conceivable runtime is not necessarily achievable. It says only that you can't do *better* than it.

An Example of How to Use BCR

Question: Given two sorted arrays, find the number of elements in common. The arrays are the same length and each has all distinct elements.

Let's start with a good example. We'll underline the elements in common.

```
A: 13  27  35  40  49  55  59
B: 17  35  39  40  55  58  60
```

A brute force algorithm for this problem is to start with each element in A and search for it in B. This takes $O(N^2)$ time since for each of N elements in A, we need to do an $O(N)$ search in B.

The BCR is $O(N)$, because we know we will have to look at each element at least once and there are 2N total elements. (If we skipped an element, then the value of that element could change the result. For example, if we never looked at the last value in B, then that 60 could be a 59.)

Let's think about where we are right now. We have an $O(N^2)$ algorithm and we want to do better than that—potentially, but not necessarily, as fast as $O(N)$.

```
Brute Force:        O(N^2)
Optimal Algorithm:  ?
BCR:                O(N)
```

What is between $O(N^2)$ and $O(N)$? Lots of things. Infinite things actually. We could theoretically have an algorithm that's $O(N \log(\log(\log(\log(N)))))$. However, both in interviews and in real life, that runtime doesn't come up a whole lot.

> Try to remember this for your interview because it throws a lot of people off. Runtime is not a multiple choice question. Yes, it's very common to have a runtime that's $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^2)$ or $O(2^N)$. But you shouldn't assume that something has a particular runtime by sheer process of elimination. In fact, those times when you're confused about the runtime and so you want to take a guess—those are the times when you're most likely to have a non-obvious and less common runtime. Maybe the runtime is $O(N^2K)$, where N is the size of the array and K is the number of pairs. Derive, don't guess.

Most likely, we're driving towards an $O(N)$ algorithm or an $O(N \log N)$ algorithm. What does that tell us?

If we imagine our current algorithm's runtime as $O(N \times N)$, then getting to $O(N)$ or $O(N \times \log N)$ might mean reducing that second $O(N)$ in the equation to $O(1)$ or $O(\log N)$.

> This is one way that BCR can be useful. We can use the runtimes to get a "hint" for what we need to reduce.