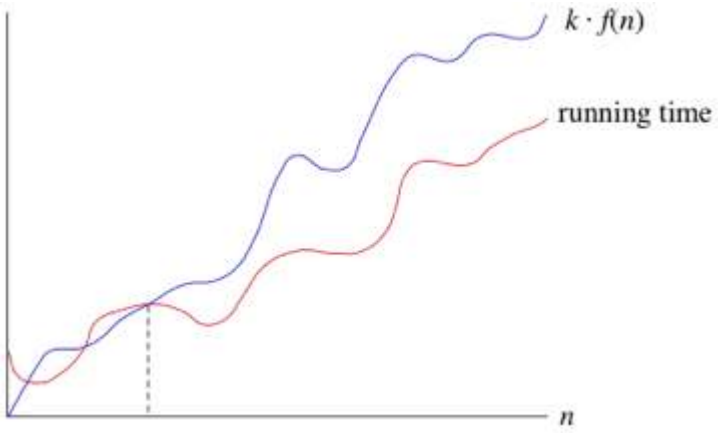
Big-O notation

We use big-Θ notation to asymptotically bound the growth of a running time to within constant factors above and below. Sometimes we want to bound from only above. For example, although the worst-case running time of binary search is **Θ(lg n)**, it would be incorrect to say that binary search runs in **Θ(lg n)** time in all cases. What if we find the target value upon the first guess? Then it runs in **Θ(1)** time. The running time of binary search is never worse than **Θ(lg n)**, but it's sometimes better. It would be convenient to have a form of asymptotic notation that means "the running time grows at most this much, but it could grow more slowly." We use "big-O" notation for just such occasions.

If a running time is **O(f(n))**, then for large enough **n**, the running time is at most **k·f(n)** for some constant **k**. Here's how to think of a running time that is **O(f(n))**:



We say that the running time is "big-O of **f(n)**" or just "O of **f(n)**." We use big-O notation for **asymptotic upper bound**s, since it bounds the growth of the running time from above for large enough input sizes.

Now we have a way to characterize the running time of binary search in all cases. We can say that the running time of binary search is always **O(lg n)**. We can make a stronger statement about the worst-case running time: it's **Θ(lg n)**.

But for a blanket statement that covers all cases, the strongest statement we can make is that binary search runs in **O(lg n)** time.

If you go back to the definition of big-Θ notation, you'll notice that it looks a lot like big-O notation, except that big-Θ notation bounds the running from both above and below, rather than just from above. If we say that a running time is **Θ(f(n))** in a particular situation, then it's also **O(f(n))**. For example, we can say that because the worst-case running time of binary search is **Θ(lg n)**, it's also **O(lg n)**. The converse is not necessarily true: as we've seen, we can say that binary search always runs in **O(lg n)** time but not that it always runs in **Θ(lg n)** time.

Because big-O notation gives only an asymptotic upper bound, and not an asymptotically tight bound, we can make statements that at first blush seem incorrect, but are technically correct. For example, it is absolutely correct to say that binary search runs in **O(n)** time. That's because the running time grows no faster than a constant times n. In fact, it grows slower. Think of it this way. Suppose you have 10 dollars in your pocket. You go up to your friend and say, "I have an amount of money in my pocket, and I guarantee that it's no more than one million dollars." Your statement is absolutely true, though not terribly precise. One million dollars is an upper bound on 10 dollars, just as O(n) is an upper bound on the running time of binary search. Other, imprecise, upper bounds on binary search would be $\mathbf{O(n^2)}$, $\mathbf{O(n^3)}$ and $\mathbf{O(2^n)}$. But none of **Θ(n)**, $\mathbf{\Theta(n^2)}$, $\mathbf{\Theta(n^3)}$, and $\mathbf{\Theta(2^n)}$ would be correct to describe the running time of binary search in any case.