Recall the definition of $O$.

**Definition 1** We say function $f$ is $O(g)$ if there exist two constants $C > 0$ and $k \geq 0$ such that

$$|f(x)| \leq C \cdot |g(x)|$$

for every $x \geq k$.

Let us illustrate the definition by showing that for $f(x) = 2x$ and $g(x) = x$, we have $f = O(g)$. We need to show a positive real number $C$ and a non-negative real number $k$ such that $|2x| \leq C|x|$ for every $x \geq k$. Since $|2x| = 2|x|$, we seek $C$ and $k$ such that $2|x| \leq C|x|$ for every $x \geq k$. In this example, the constants are clear. For example, take $C = 2$ and $k = 0$, then the inequality holds. That is, it is true that

$$|2x| \leq 2|x|$$

for every $x \geq 0$.

**In practice, the value of the constant $C$ could have a big impact.** Two students devise two different, but correct, algorithms—Algorithm $A$ and Algorithm $B$—for a problem. In the worst case, Algorithm $A$ executes $f_A(n) = 4n^2$ operations on an input of size $n$, and Algorithm $B$ executes $f_B(n) = n^2 + 10000$ steps on the same input. It is a simple exercise to show that both $f_A(n)$ and $f_B(n)$ are $O(n^2)$. Let us take $k = 1$, for example. Then, if we take $C_A = 4$, for all $n \geq k$, we have $4n^2 \leq C_A n^2$, showing that $f_A(n) = O(n^2)$. Similarly, if we take $C_B = 10001$, we have $n^2 + 10000 \leq 10001n^2$ for every $n \geq 1$, showing that $f_B(n) = O(n^2)$.

**What role, if any, does $k$ play, then?** Based on the above analysis, the constant of Algorithm $B$ is much larger than that of Algorithm $A$, giving the impression that, in practice, Algorithm $B$ might very well have a worse performance than Algorithm $A$. But is that truly the case for sufficiently large input sizes? Let us consider, for example, an input of size $n = 100$. For this input, Algorithm $A$ would execute $4 \cdot 100^2 = 40000$ steps, whereas Algorithm $B$ would execute $100^2 + 10000 = 20000$ steps. We can generalize this and ask the question: Is there an input size beyond which Algorithm $B$ would execute at most as many steps as Algorithm $A$ would? Answering this question amounts to finding the value of $n$ for which

$$n^2 + 10000 \leq 4n^2.$$

Equivalently, we seek $n$ such that $3n^2 \geq 10000$. Solving this inequality gives us $n \geq \sqrt{10000/3} = 57.735$. In other words, if we take $k = 58$, then we have $f_A(n) \leq f_B(n)$ for every $n \geq k$. It follows from this that if we choose $k = 58$, then whatever constant $C$ we choose so that $f_A(n) \leq Cn^2$, then for sure $f_B(n) \leq Cn^2$ for $n \geq k$, since $f_B(n) \leq f_A(n) \leq Cn^2$. This analysis shows us that if we ignore inputs of sizes 57 or smaller, the constant for Algorithm $B$ is not worse (i.e., larger) than the constant for Algorithm $A$.

Let us now tighten this analysis by asking the question: Is there a value $k$ such that $f_B \leq n^2$ for all $n \geq k$? That is, is there a value of $k$ that would allow us to choose constant $C_B = 1$ so that $f_B \leq C_B n^2$? The answer is negative, because no matter what $k$ is, we have

$$n^2 + 10000 > n^2$$

1 “Sufficiently large $n$” is the meaning of $n \geq k$ for some constant $k$. 
for every \( n \geq k \). So, \( C_B = 1 \) does not work. But is there a constant \( C_B \) that is very close, but not equal, to 1, such that \( f_B \leq C_B n^2 \)? Think of \( C_B \) as being a real number \( 1 + \epsilon \), where \( \epsilon \) is a number that is very close, but not equal, to 0. So, we solve

\[
 n^2 + 10000 \leq (1 + \epsilon)n^2.
\]

Equivalently, \( n^2 \geq (10000/\epsilon) \), or \( n \geq \sqrt{10000/\epsilon} = 100/\sqrt{\epsilon} \). Keep in mind that \( 100/\sqrt{\epsilon} \) is (a lower bound on) \( k \).

What we have shown is that if we take \( k = 100/\sqrt{\epsilon} \), then \( f_B(n) \leq C_B n^2 \) for \( C_B = (1 + \epsilon) \).

This analysis shows us that if we want a smaller value for \( C_B \), the value of \( \epsilon \) has to be smaller, which means that value of \( 100/\sqrt{\epsilon} \) (that is, the value of \( k \)) has to be larger. In other words, the constant \( k \) plays the role of a “tuning knob” that allows us to “tighten” the value of \( C \). Choose a larger value of \( k \), allows us to choose a smaller value of \( C \).

If we turn our attention now to algorithm \( A \), we notice that no matter what value of \( k \) we choose, the constant \( C_A \) cannot be smaller than 4 for \( f_A(n) \leq C_A \cdot n^2 \) to be true for \( n \geq k \).

To summarize, for Algorithm \( B \), we can choose a value of \( k \) that is large enough so that \( f_B = O(n^2) \) with a constant \( C_B \) that is arbitrarily close to 1. For Algorithm \( A \), no matter what value of \( k \) we choose, the constant \( C_A \) has to have a value of at least 4. In practice, for sufficiently large input sizes, Algorithm \( B \) is better than Algorithm \( A \) in terms of running time.