Intro:

"Today, we're exploring another fundamental big O notation: O(1), also known as constant time complexity. This concept illustrates the efficiency of operations that do not scale with the size of the input."

A1:

"We'll start with a simple function named addItems. This function performs a straightforward task: it returns the sum of n and n. In this case, there's only one operation involved—addition."

A2:

"Regardless of whether n is one, a hundred, or a million, the number of operations remains constant at one. This is the essence of O(1) complexity."

A3:

"What if we adjust addItems to include another addition, making it two operations? While initially, this might seem like O(2), in big O notation, we simplify this to O(1). The rationale? The operation count doesn't scale with n; it remains constant."

A4:

"This is why O(1) is often described as the most efficient form of big O notation. No matter how large the input, the computation time stays the same."

Outro:

"On a graph, O(1) is represented as a flat line, indicating no change in operation count regardless of input size. Remember, when you hear the term 'constant time', it's referring to O(1). It’s a pinnacle of efficiency in algorithm design, ensuring the fastest possible execution time."