Intro:

"Today, we explore the big O notation O(log n), a fundamental concept often associated with the efficient 'divide and conquer' search technique in sorted arrays."

A1:

"Imagine a sorted array where you need to find a specific number, like the number one. The most efficient method isn't to look at each item sequentially but to divide the array into halves repeatedly to narrow down where the number could be."

A2:

"Let's visualize this with an example: we have an array of eight items. If we divide the array in half and determine the number one isn't in the second half, we've immediately eliminated four items from our search. We continue this process, each time halving the number of items we consider."

A3:

"After a few divisions, specifically three in our case, we find the number one. Here’s the key: each 'cut' reduces the possible locations exponentially. For an array of eight items, it takes three steps because 2³ (two raised to the power of three) equals eight."

A4:

"This process translates mathematically to what we call a logarithm. In simpler terms, for our array of eight, the log base two of eight equals three—indicating that three divisions are necessary. The beauty of O(log n) is its efficiency, especially noticeable in large datasets."

A5:

"For example, if you have a billion items, log base two of a billion is about 31. This means any item can be found in just 31 steps, rather than a billion steps required for a linear search."

Outro:

"On a graph, O(log n) appears almost flat, especially when compared to O(n) or O(n²), showcasing its efficiency. We'll also touch upon O(n log n), crucial in some sorting algorithms, but that’s a topic for another discussion. For now, understanding O(log n) provides a solid foundation in evaluating algorithm efficiency."