Intro:

"In today's discussion on simplifying big O notation, we delve into a technique known as 'Drop Non-dominants.' This method helps us focus on the most significant terms that affect algorithm complexity."

A1:

"Returning to our previous example, we have a nested for loop structure. I'll enhance this by adding an additional for loop after the nested ones. This new loop will execute independently of the nested loops."

A2:

"In DevTools, we execute this code with an input of ten. Initially, the nested loops generate pairs from zero zero up to nine nine, representing O(n²) operations. Following these, the additional for loop runs from zero to nine, clearly demonstrating an O(n) operation."

A3:

"When these two loops are combined in our analysis, the total operation count appears as O(n² + n). To put this in perspective, if n were 100, n² would result in 10,000 operations, whereas the added n contributes just 100 more operations, a relatively minor increase."

A4:

"This discrepancy highlights that the n term, while it adds to the total, does not significantly impact the overall complexity compared to n². In big O notation, since n² is the dominant term and n is non-dominant, we simplify the expression by dropping the non-dominant n, referring to the total complexity as merely O(n²)."

Outro:

"This practice of dropping non-dominant terms is a key strategy in big O notation to keep our complexity descriptions both accurate and manageable. Understanding which terms to consider and which to omit allows us to better evaluate and communicate the efficiency of our algorithms."