Asymptotic Notation is a mathematical framework used to describe the efficiency of algorithms in terms of their growth rate as input size increases. It provides a high-level way to compare algorithms regardless of hardware or implementation details. The most common forms are Big O (O) for upper bounds, Big Omega (Ω) for lower bounds, and Big Theta (Θ) for tight bounds. For example, if an algorithm's runtime increases proportionally to the number of vertices plus edges in a graph, we denote it as O(V + E). This notation is crucial for analyzing algorithms like Hopcroft–Tarjan, which achieves linear-time performance.

Depth-First Search (DFS) is a fundamental graph traversal technique that explores as far as possible along one branch before backtracking. It is typically implemented using recursion or a stack. DFS plays a critical role in many graph algorithms, such as detecting cycles, finding connected components, and performing topological sorting. The Hopcroft–Tarjan algorithm—which efficiently finds articulation points and biconnected components—heavily relies on DFS as its core mechanism. During traversal, DFS helps maintain discovery times, low values, and parent relationships, which are essential for identifying the vertices and edges that, if removed, would disconnect the graph.

Recurrence Analysis is a mathematical method used to determine the time complexity of recursive algorithms by expressing the overall running time in terms of smaller subproblems. For example, an algorithm with recurrence $T(n)=2T(n/2)+O(n)$ can be solved using the Master Theorem, yielding $T(n)=O(n\log n)$. While DFS-based algorithms like Hopcroft–Tarjan are not strongly recursive, recurrence analysis helps explain how repeated traversal over vertices or edges contributes to linear time complexity. Each vertex and edge is processed once, leading to the recurrence of $T(V, E) = T(V-1, E-1) + O(1)$, which simplifies $O(V + E)$.

The Hopcroft–Tarjan algorithm exemplifies the intersection of these three concepts. It applies DFS to systematically visit each vertex and track connectivity properties using discovery and low values. Its efficiency is explained through asymptotic notation, as it runs in linear time relative to the size of the graph. Meanwhile, recurrence analysis provides theoretical justification for why each operation contributes a constant amount of work per vertex and edge, ensuring scalability even for large graphs. Together, these concepts form the theoretical backbone that proves the Hopcroft–Tarjan algorithm's optimal performance in graph analysis.