Revision Notes: Time and Space Complexity

Introduction to Time Complexity

Time complexity is an essential concept in computer science that measures the efficiency of an algorithm based on the relation of input size to the number of operations it performs. The analysis often employs Big O notation, which describes the upper bound on time complexity, emphasizing the worst-case scenario 【4:10†transcript.txt】【4:1†typed.md】.

Big O Notation

- **Purpose**: Represents the growth rate of the algorithm's running time as the input size increases.
- **Focus**: The trend, ignoring constant factors and coefficients.
- **Primary Consideration**: Only the highest-order term in polynomial expressions is considered important. For instance, in an algorithm with execution times represented by `100*N^2 + 32*N^3 + 51*N + 1000`, Big O notation simplifies it to `O(N^3)` 【4:6†typed.md】【4:9†transcript.txt】【4:17†transcript.txt】.

Key Concepts:

- **Order of N (O(N))**: Common complexity when a loop iterates `N` times 【4:0†transcript.txt】.
- **Order of N + M (O(N + M))**: When a loop executes `N` and another loop/method executes `M` times independently【4:0†transcript.txt】.
- **Order of N Log N (O(N log N))**: Common in algorithms that perform a halving operation multiple times, like quicksort【4:5†transcript.txt】.
- **Order of 1 (O(1))**: Algorithms that require a constant amount of time, independent of the input size【4:7†transcript.txt】.

Examples & Analogies:

- **Loop Executions**: If you have a loop running `N` times and an embedded loop running `log N`, the overall complexity becomes `O(N log N)

it doesn't rely on input size, it results in `O(1)`, not `O(100)` 【4:7†transcript.txt】【4:8†transcript.txt】.

Space Complexity

Space complexity measures the total storage space required by the algorithm as a function of the input size. It assesses both the input data and any extra space used by the algorithm. Like time complexity, it is also expressed using Big O notation 【4:6†typed.md】【4:0†transcript.txt】.

Key Considerations:

- **Inbuilt vs. Extra Space**: Consider space complexity only when extra space is created; the input space itself isn't counted 【4:2†transcript.txt】.
- **Order of 1 (O(1))**: Common when no additional space other than the input is used 【4:2†transcript.txt】.

Arithmetic and Geometric Progressions (AP & GP)

These mathematical concepts were revisited briefly to provide foundational support for understanding algorithmic analysis:

Arithmetic Progression (AP)

Defined by a constant difference `d` between consecutive terms. For an AP, the nth term is defined as `a + (n-1)*d`. The sum of the first N terms: `n/2 * (first term + last term)` 【4:12†transcript.txt】【4:19†transcript.txt】.

Geometric Progression (GP)

Each term is a constant multiple `r` of the previous term. The sum of the first N terms of a GP is given as `a * (r^n - 1)/(r-1)` 【4:13†transcript.txt】.

Practical Applications and Problem Solving

optimizations. Recognition of recurring patterns, like nested loops and halving mechanisms in algorithms, enables predictions of complexity impacts【4:15†transcript.txt】【4:16†transcript.txt】.

Conclusion

Understanding time and space complexities grants deeper insight into algorithm efficiency and is crucial for writing optimal code. The session elucidated these principles using examples and drew analogies to mathematics to foster comprehension 【4:10†transcript.txt】【4:14†transcript.txt】.

Being proficient in these concepts not only aids in problem-solving but is an essential competency in software engineering, where efficient coding can dramatically affect execution time and resource consumption.