Scaler Companion beta

Revision Notes: Understanding Time Complexity and Basic Algorithmic Concepts

Introduction

In this session, we focused on various foundational aspects of programming and problem-solving. These concepts are crucial, especially when dealing with algorithms and data structures. Here's a detailed revision of everything covered in the class.

1. Understanding Factors

What is a Factor?

A factor of a number N is an integer i that divides N completely without leaving any remainder. Mathematically, i is a factor of N if:

- $N \% i = 0$

Example Problem

Question: Count the factors of a number N.

- For instance, for N = 24, the factors are 1, 2, 3, 4, 6, 8, 12, and 24, which totals to 8 factors.

2. Mathematical Concepts

Sum of Natural Numbers

For calculating the sum of the first N natural numbers:

- Formula: $S = \frac{n \times (n+1)}{2}$

Geometric Progression (GP)

- GP is a sequence where each term after the first is found by multiplying the previous term by a fixed, non-zero number called the ratio.
- Sum of a finite GP: $S_n = a\frac{1-r^n}{1-r}$ where a is the first term, r is the common ratio, and n is the number of terms.

3. Comparing Algorithms

Time Complexity

Time complexity is a way of comparing two algorithms by focusing on how their execution time grows with the input size, denoted by Big O notation (O).

- **Big O**: It tells you the upper limit of the running time for the worst-case scenario.
- **Example**: If an algorithm has a time complexity of O(N), it indicates that the number of operations needed is proportional to the number of elements N.

Practical Example:

- Code 1: Iterating through N elements.
- Code 2: Iterating with a step, potentially reducing operations.

```
for (i = 1 ; i <= N ; i++) {
   if (i%2 != 0) {
     count++;
   }
}
```

Here, Code 2 with an iteration step of 2 would still be O(N) since constant factors are ignored in Big O.

4. Time Complexity Analogy

Consider comparing two sorting algorithms:

demonstrating that actual time isn't always a fair method of comparison. Instead, theoretical complexity such as Big O is a fairer metric.

5. Real-World Observations

Successful problem-solving often relies on recognizing patterns and leveraging observations:

- For example, figuring out the number of factors or realizing that factors occur in pairs can greatly optimize solutions from years to seconds without advanced mathematics【4:14†transcript】.

Conclusion

This class reinforced the importance of mastering time complexity and observing patterns in algorithms for efficient problem-solving. Continue to practice these concepts as they form the backbone of algorithmic efficiency.

This concludes the revision notes for this class. Practice and revisions are key to mastering these basics!