Time complexity of algorithm

Time complexity of an algorithm refers to the amount of time it takes to execute as a function of the size of the input data. It helps us understand how the algorithm's performance scales with larger inputs.

Let's consider a simple example:

Example: Linear Search

Suppose you have an array of numbers, and you want to find a specific number in the array. You use a linear search algorithm that goes through the numbers one by one until it finds the desired number.

- If there are 10 numbers in the array, and the number you're looking for is the first number in the array, the algorithm performs only one comparison (best-case scenario).
- If the number you're looking for is the last number in the array, the algorithm will perform 10 comparisons (worst-case scenario).

In this example, the time complexity is O(n), where 'n' is the number of elements in the array. This means that the time the algorithm takes increases linearly with the size of the input data.

Here's a breakdown:

- Best Case: O(1) - Constant time (the desired number is found immediately).
- Average Case: O(n/2) - Approximately half of the elements are compared on average.
- Worst Case: O(n) - All elements are compared before finding the desired number.

In simple terms, the time complexity tells you how the algorithm's performance changes as the input size grows. In the case of linear search, the time taken increases proportionally to the number of elements in the array.

Space complexity

Space complexity of an algorithm refers to the amount of memory or space required by the algorithm to run as a function of the input data size. It helps us understand how much memory the algorithm needs to execute.

Let's consider a simple example:

Example: Sum of an Array

Suppose you want to calculate the sum of all numbers in an array. You use a straightforward algorithm that keeps a variable to store the sum. The space complexity here is minimal because you're only using a single variable to store the result.

```
#include <iostream.h>

void main() {
    int arr[100];
    int n, total = 0;

    clrscr(); // Clear the screen

    // Input the number of elements in the array
    cout << "Enter the number of elements: ";
    cin >> n;

    // Input the elements
    cout << "Enter the elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Calculate the sum of the elements
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }

    // Display the result
    cout << "Sum of the elements: " << total;

    getch(); // Wait for a key press before exiting
}
```

In this example, the space complexity is O(1), which means that the algorithm's memory usage remains constant regardless of the size of the input array. It only requires a fixed amount of memory to store the 'total' variable.

Here's a breakdown:

- Space Complexity: O(1) - Constant space (the algorithm uses the same amount of memory regardless of input size).

In simple terms, the space complexity tells you how efficiently the algorithm uses memory. In this case, the space required by the algorithm doesn't increase with larger input data; it remains constant