C++ Course Notes | Time and space complexity analysis | Week 8

Hello everyone welcome to the weekly lecture notes

Topics to be covered:

- Time Complexity
- Space Complexity

Time Complexity

- Applying the asymptotic analysis to measure the time requirement of an algorithm as a function of input size is known as time complexity. We assume each instruction takes a constant amount of time for time complexity analysis.
- Total time complexity of a program is equal to the summation of all the running time of disconnected fragments.
- Need for Time Complexity : When analyzing any algorithm, we need to evaluate the effectiveness of that algorithm. Accordingly, we need to prefer the most optimized algorithm so as to save the time taken by it to execute. An example for the same could be linear search and binary search. Let's suppose we need to search a given value in a sorted array of size 109. This would take 109 iterations for linear search whereas it would just take log(109) ~ 30 iterations for binary search to search the same element, thereby saving a lot of time. Time Complexity for both binary search and linear search will be discussed in further lectures.

Types of Time Complexity Analysis:

1. **Worst Case Time Complexity**: It is that case where the algorithm takes the longest time to complete its execution. It is represented by the Big-O notation. By default we calculate the worst case time complexity when we refer to the time complexity of a code.
2. **Best Case Time Complexity**: It is that case where the algorithm takes the least time to complete its execution. It is represented by the Omega Notation (Ω-Notation).
3. **Average Case Time Complexity**: As the name suggests, it gives average time for a program to complete its execution. It is represented by the theta notation (Θ-Notation).

**

Let’s go through a few of the examples to grasp how to calculate time complexity :

Example 1:

Consider the case when we need to output “Hello World”.

```
cout << "Hello World";
```

Explanation:

The time complexity for printing just a single Hello World will be O(1).

Example 2:

Consider the following function where we need to traverse the array and calculate the total sum of all the elements present in the array.

```
int sum = 0;
for(int i=0;i<N;i++) // 0 based indexing
sum += a[i];
```

Explanation:

Here we are running the loop N times and checking each time if the value at that particular index is 1 and if that is true then increasing the count variable. The checking in if statement as well as the increment count is O(1) and we are doing it N times in total while traversing through the array so the total time complexity is O(N).

Example 3:

Consider the case when we need to traverse on 2 individual loops of length n and m respectively.

```
int c1 = 0;
for(int i=0;i<n;i++)c1 += 1;
int c2 = 0;
for(int i=0;i<m;i++)c2 += 1;
```

Explanation:

Here we have two independent for loops one executing n times and the other executing m times. The time complexity for the first loop is O(n) and for the second loop it is O(m). Hence the total time complexity is `O(n+m)`.

Example 4:

Time Complexity for Nested Loops

```
int count = 0;
for(int i=0;i<N;i++)
  for(int j=0;j<i;j++)
    count += 1;
```

Explanation:

Here in this case for every particular iteration in the first loop, the inner loop is being iterated i times, thereby forming a case of arithmetic progression of 0 + 1 + 2 + ... + N-1. The total number of iterations will be the sum of all the elements in this progression i.e. N (N+1)/2.

N(N+1)/2 when expanded becomes (N2 + N) / 2. While calculating the time complexity, we refer to the worst case time complexity hence, we only consider the dominant terms and ignore the lower power terms.

Hence the time complexity for Nested Loops is `O(N^2)`.

Example 5:

Another example of Nested for loops

```
int count = 0;
for(int i=0;i<N;i++)
  for(int j=0;j<Math.sqrt(N);j++)
    count+=1;
```

Explanation:

Here in this case, the second loop iterates only logarithmic of N times having a time complexity of sqrt(N) and the first loop iterates through the whole loop thereby giving us a total time complexity of `O(N sqrt(N))`.

Try this:

Calculate the time complexity for the following code snippet.

```
int val = 0;
for(int i = 1; i <= N; i *= k){
  val++;
}
```

Explanation:

Here to calculate the time complexity, we have to calculate the number of iterations in the above loop. Let the total number of iterations be T.

Then values of i will be -

1, k, k2, k3, ... kT

Here k is the constant i is multiplied at each iteration.

kT <= N

So T comes out to be the logarithmic base k of N.

Overall time complexity will be `O(logN)`.

Space Complexity

- Space Complexity is the extra memory space requirement of an algorithm by applying asymptotic analysis.
- We only care about what is known as auxiliary space complexity, which does not include the input size.

- Need for Space Complexity: The amount of space a system has can be limited and therefore we need to optimize the memory/space taken by the algorithm to execute on that particular system with bounded space limits.

Example 1:

Calculate the sum of all the array elements.

```
int sum=0;
for(int i=0;i<N;i++)
  sum+=a[i];
```

Explanation:

Here the space complexity for the variable sum will be `O(1)`.

Example 2: Consider the following function to store first N natural numbers in an array:

```
int a[N];
for(int i=0;i<N;i++){
  a[i] = i+1;
}
```

Explanation:

Here we declare an array of size N and assign the index the required values. Hence the space complexity of that array will be `O(N).`

Note:

Natural numbers are integers greater than 0 i.e., 1, 2, 3, ...

Example 3:

Let's consider a 2-d matrix with all elements 1 in it.

```
int A[N][N];
for(int i=0;i<N;i++)
  for(int j=0;j<N;j++)
    A[i][j] = 1;
```

Explanation:

Here we have created a 2 dimensional array of size NxN and inserted value 1 in every index in it. The space complexity for this 2d array will be `O(N^2)`.

Note:

If we create additional data structures like arrays, matrices, etc., in our function, we need to account for those in our space complexity computation.
