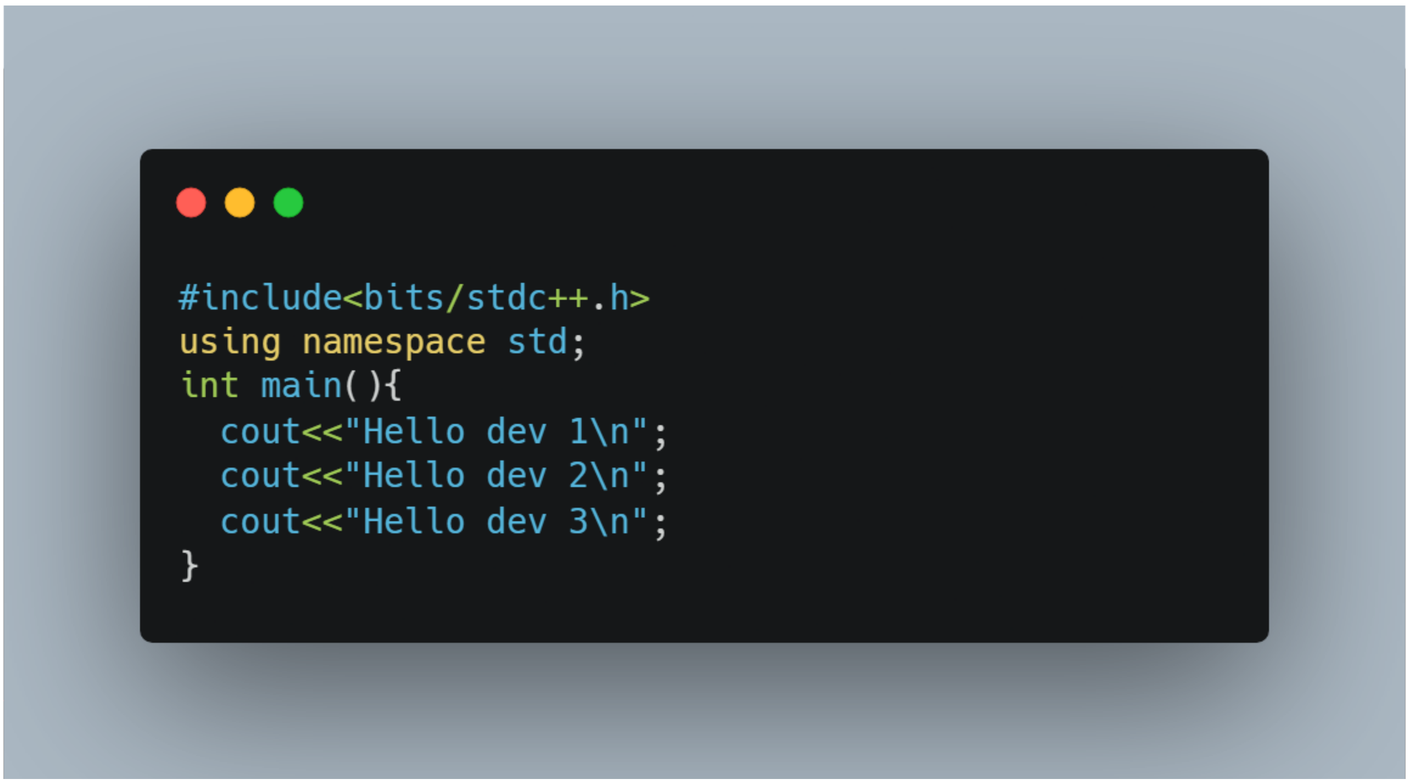
What is time complexity?

Time complexity is a programming term that quantifies the amount of time it takes a sequence of code or an algorithm to process or execute in proportion to the size and cost of input.

It will not look at an algorithm's overall execution time. Rather, it will provide data on the variation (increase or reduction) in execution time when the number of operations in an algorithm increases or decreases.

Yes, as the definition suggests, the amount of time it takes is solely determined by the number of iterations of one-line statements inside the code.

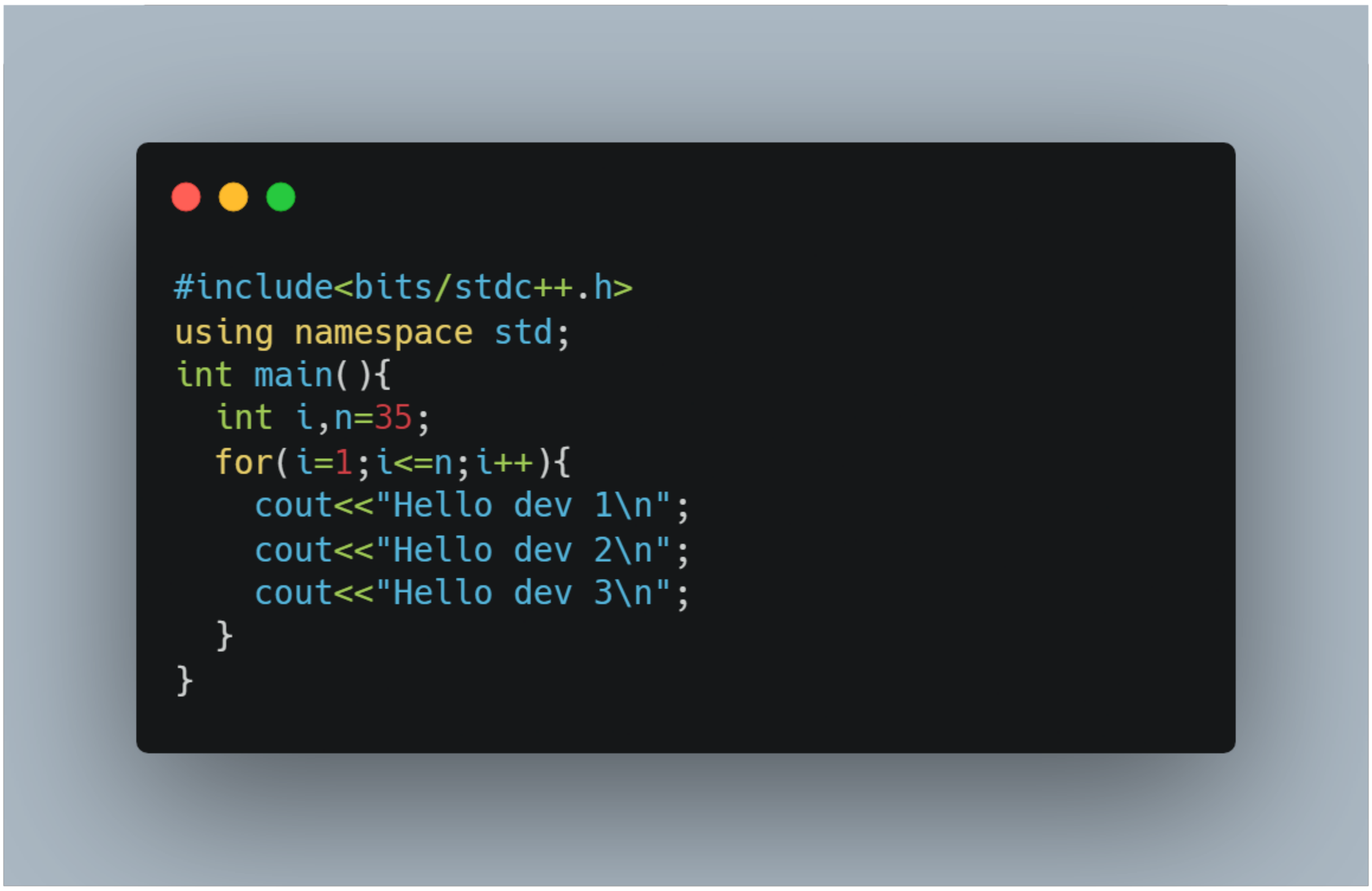
Take a look at this piece of code:



```
#include<bits/stdc++.h>
using namespace std;
int main(){
  cout<<"Hello dev 1\n";
  cout<<"Hello dev 2\n";
  cout<<"Hello dev 3\n";
}
```

Although there are three separate statements inside the **main()** function, the overall time complexity is still **O(1)** because each statement only takes unit time to complete execution.

Whereas, observe this example:



```cpp
#include<bits/stdc++.h>
using namespace std;
int main(){
  int i,n=35;
  for(i=1;i<=n;i++){
    cout<<"Hello dev 1\n";
    cout<<"Hello dev 2\n";
    cout<<"Hello dev 3\n";
  }
}
```

Here, the **O(1)** chunk of code (the 3 cout statements) is enclosed inside a looping statement which **repeats iteration for 'n' number of times**. Thus our overall time complexity becomes n*O(1), i.e., **O(n)**.

You could say that when an algorithm employs statements that are only executed once, the time required is constant. However, when the statement is in loop condition, the time required grows as the number of times the loop is set to run grows.

When an algorithm has a mix of single-executed statements and LOOP statements, or nested LOOP statements, the time required increases according to the number of times each statement is run.

For example, if a **recursive function** is called multiple times, identifying and recognizing the source of its time complexity may help reduce the overall processing time from **600 ms** to **100 ms**, for instance. That's what time complexity analysis aims to achieve.