Summary DSA

Sessions No 02(29-11-2022)

- **DSA** - We use DSA to optimize our code and increase its performance

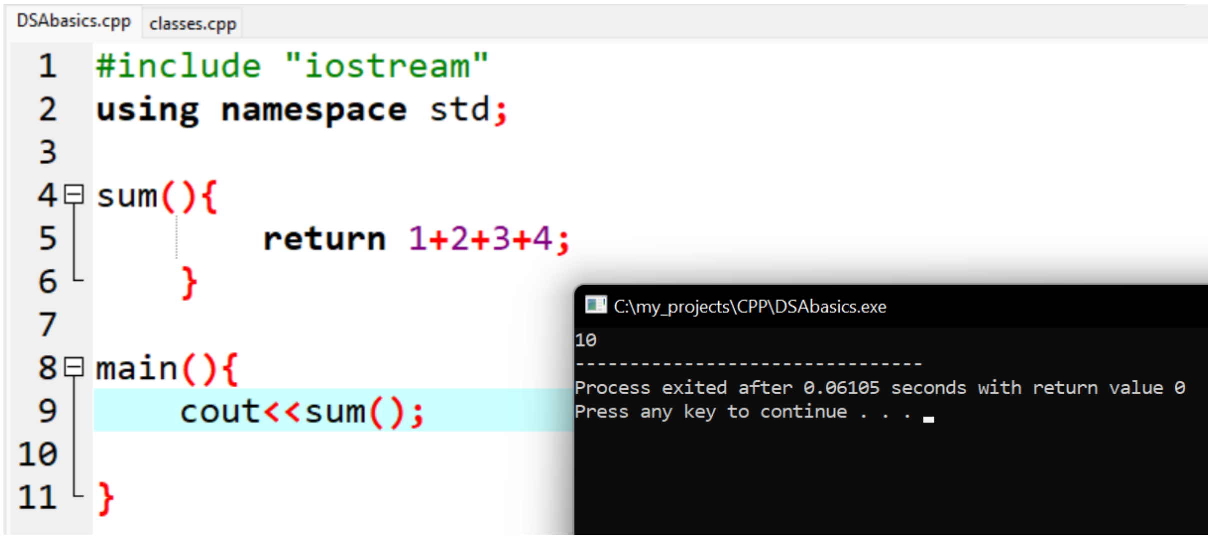
- A *function* is equal to *algorithms*, The code we write inside the function is called an algorithm. The good practice is that for different goals create different algorithms(and those algorithms we will write inside a function.)

- **Time complexity** is the computational complexity that describes the amount of CPU time it takes to run an algorithm. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm.

- We as a developer divide our applications into different *functions* and we write every function in an optimized way that has less time complexity.

- Now in the below example, we have created a function named "***sum***" which does the addition of *1,2,3, and 4* and then return back the output. Now the code we have written inside the function is known as the **algorithm**, And before running the algorithm we can do our **asymptotic analysis** to understand how much time complexity this algorithm has.

  Now while doing *asymptotic analysi*s we see that it is always doing the *addition* of *1,2,3,4.* So every time we run this algorithm it will take the same unit of CPU time, So here we can denote this algorithm with time complexity of **O(constant)** Now a *constant* we can also denote it as **1**, because we already know how much time this algorithm takes(fixed time), So we can already arrange that much of resources in our company, and we know it is taking constant time no matter when we are running it so we will denote that this algorithm as **time complexity** of **O(1)**.

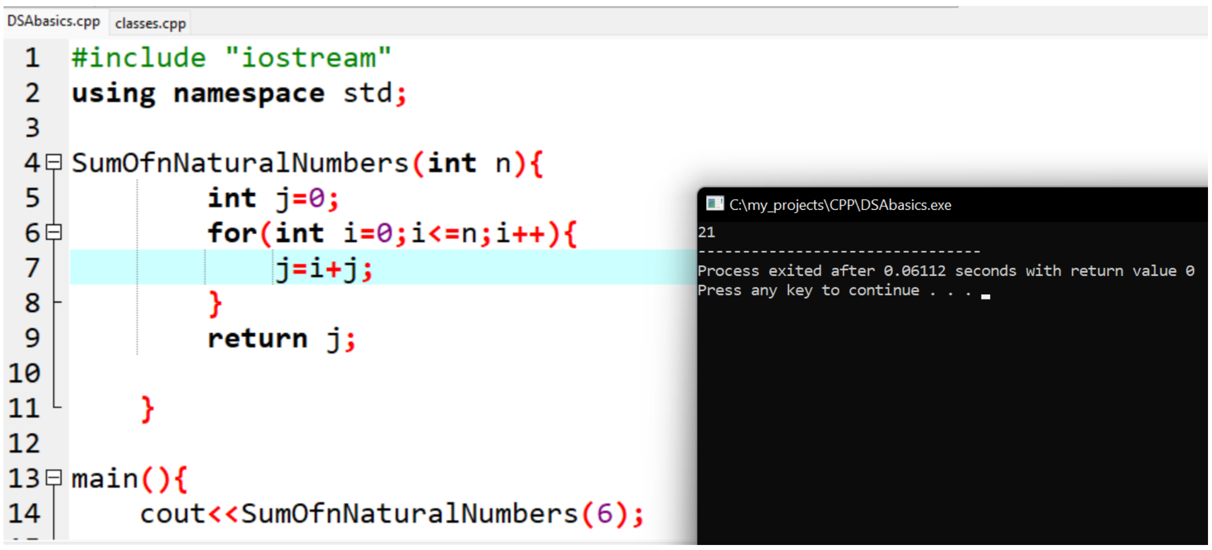


```cpp
#include "iostream"
using namespace std;

sum(){
        return 1+2+3+4;
    }

main(){
    cout<<sum();

}
```

```
10
--------------------------------
Process exited after 0.06105 seconds with return value 0
Press any key to continue . . .
```

- Now let's do some changes and create a new algorithm that takes input from the user and then does the *sum of n natural numbers*, so checks the below example.

  Here our function is taking the *integer* input and then with the help of *for loop* it is doing the addition of *n natural numbers*. Now, in this case, we can't tell how much CPU unit of time this algorithm will take, As because it is keep on changing as per the user input(*input is deciding how much time for loop will run*). In the case of **n=6**, we can see statement **j=i+j** will run **6 times**, and in the case of **n=7**, that statement will run **7 times,** and so on, So here we can see the linear behavior between the *user input* and the ***CPU time***, so in this case, we got to know after doing **asymptotic analysis** that our CPU time totally depends on the input provided by the user. So in this case **time complexity** will not be constant, It would be **O(n)**.



```cpp
#include "iostream"
using namespace std;

SumOfnNaturalNumbers(int n){
        int j=0;
        for(int i=0;i<=n;i++){
            j=i+j;
        }
        return j;

    }

main(){
    cout<<SumOfnNaturalNumbers(6);
```

```
21
--------------------------------
Process exited after 0.06112 seconds with return value 0
Press any key to continue . . .
```