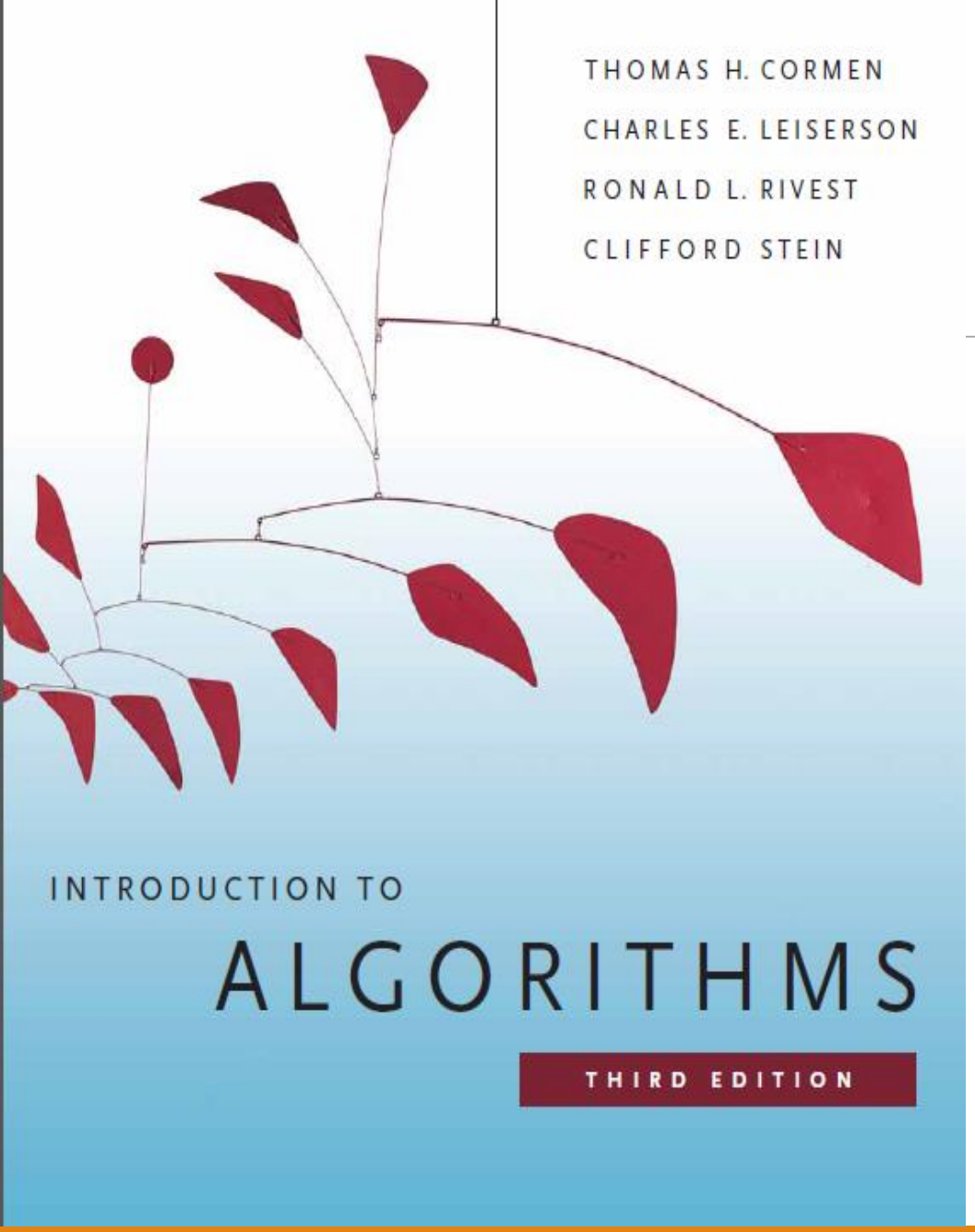
Design and Analysis of Algorithms

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg



Text Book

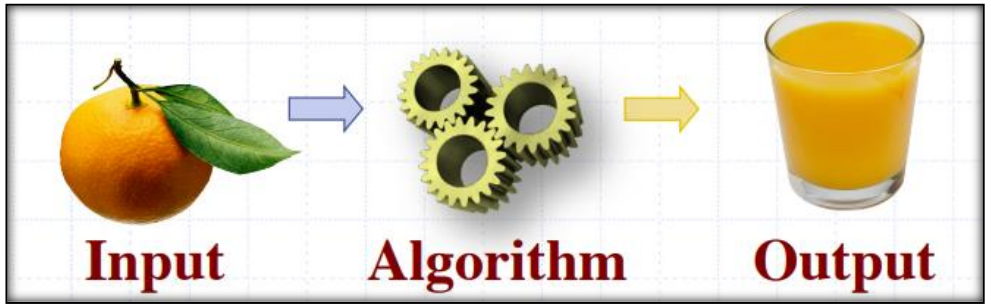
MIT Press

(Massachusetts Institute of Technology)

https://www.amazon.com/Introduction-Algorithms-3rd-MIT-Press/dp/0262033844

What is an algorithm and why are they important?

- An **algorithm** is any well-defined **computational procedure** (sequence of instructions) that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**. An algorithm is thus a sequence of computational steps that transform the input into the output.

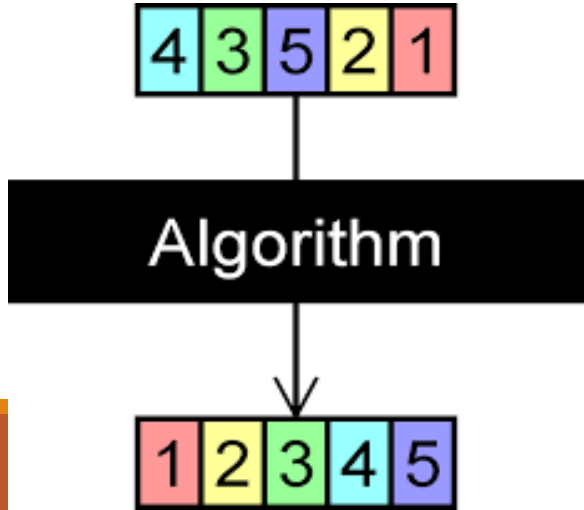


- For example, **sorting problem**, we might need to sort a sequence of numbers into non-decreasing order. This problem provides fertile ground for introducing many standard design techniques and analysis tools.

Define the sorting problem:

Input: A sequence of **n** numbers $(a_1, a_2, \ldots\ldots, a_n)$.

Output: A permutation (reordering) $(a'_1, a'_2, \ldots\ldots, a'_n)$ of the input sequence such that $(a'_1 \leq a'_2 \leq \ldots\ldots \leq a'_n)$.



Writing an Algorithm

- Algorithm is generally developed before the actual coding is done. It is written using **English like language** so that it is easily understandable even by non-programmers. Examples of an algorithm. Here's what **baking a cake** might look like, written out as a list of instructions, just like an algorithm:

- Preheat the oven
- Gather the ingredients
- Measure out the ingredients
- Mix together the ingredients to make the batter
- Grease a pan
- Pour the batter into the pan
- Put the pan in the oven
- Set a timer
- When the timer goes off, take the pan out of the oven
- Enjoy!

Five major components of any algorithm:

1) **Start:** Specify the algorithm’s starting point.
2) **Input:** Enter the problem’s input parameters.
3) **Task (calculations):** Carry out your calculations (math).
4) **Output:** Display the problem’s output parameters (results).
5) **End:** Specify the algorithm's end point.

Five major components of any algorithm [Example1]:

Create an algorithm to compute and display the gross salary of an hourly employee.

SOLUTION

As mentioned before, any algorithm consists of 5 major components:

- Start
- Input
- Calculation
- Output
- End

Five major components of any algorithm [Example1]:

- **Start** (The 1st step of the algorithm is usually "Start" and is placed at the beginning of the algorithm.)
  It is written in the algorithm as:
  1. Start.

- **Input** (In this example, we need to know the number of hours (N) and the hourly payment (R) to calculate the gross payment. As a result, there are two inputs, N and R. We can enter N and R in two different steps.)
  It is written in the algorithm as:
  2. Input the number of hours (N).
  3. Input the hourly pay rate (R).

Five major components of any algorithm [Example1]:

- **Calculation** (The value of the gross payment can be calculated by multiplying N by R.)
  It is written in the algorithm as:
  4. Multiply the number of hours worked by the hourly pay rate.

- **Output** (In this step, we will display the result of the gross payment according to the requirement of the example.)
  It is written in the algorithm as:
  5. Display the result of the calculation that was performed in step 4.

- **End** (This step shows the end point of the pseudocode, and is usually placed after the results output.)
  It is written in the algorithm as:
  6. End.

Five major components of any algorithm [Example1]:

Finally, the algorithm can be written as:

1. Start.
2. Input the number of hours (N).
3. Input the hourly pay rate (R).
4. Multiply the number of hours worked by the hourly pay rate.
5. Display the result of the calculation that was performed in step 4.
6. End.

Five major components of any algorithm [Example2]:

Create an algorithm to find the sum of any two numbers.

the algorithm can be written as:

1. Start.
2. Enter the two numbers in the variables N and M.
3. Sum the two numbers and save the result in the variable S.
4. Display the result S.
5. End.

Five major components of any algorithm [Example3]:

Create an algorithm to change the temperature from Fahrenheit to Centigrade. $^oC = \frac{5}{9}(\,^oF - 32)$

the algorithm can be written as:

1. Start.
2. Enter the value of temperature in Fahrenheit.
3. Subtract 32.
4. Multiply the result in step (2) by 5.
5. Divide the result in step (3) by 9.
6. Write down the answer.
7. End.

Algorithms and Pseudo codes

- Programmers frequently utilize **two tools** to assist them in converting algorithm into code :

1. **Pseudocode**
2. **Flow charts**

- Because small mistakes like misspelled words and forgotten punctuation characters can cause syntax errors, programmers find it helpful to write program in pseudo code before they write it in the actual code of programming languages such as C++ or Matlab.

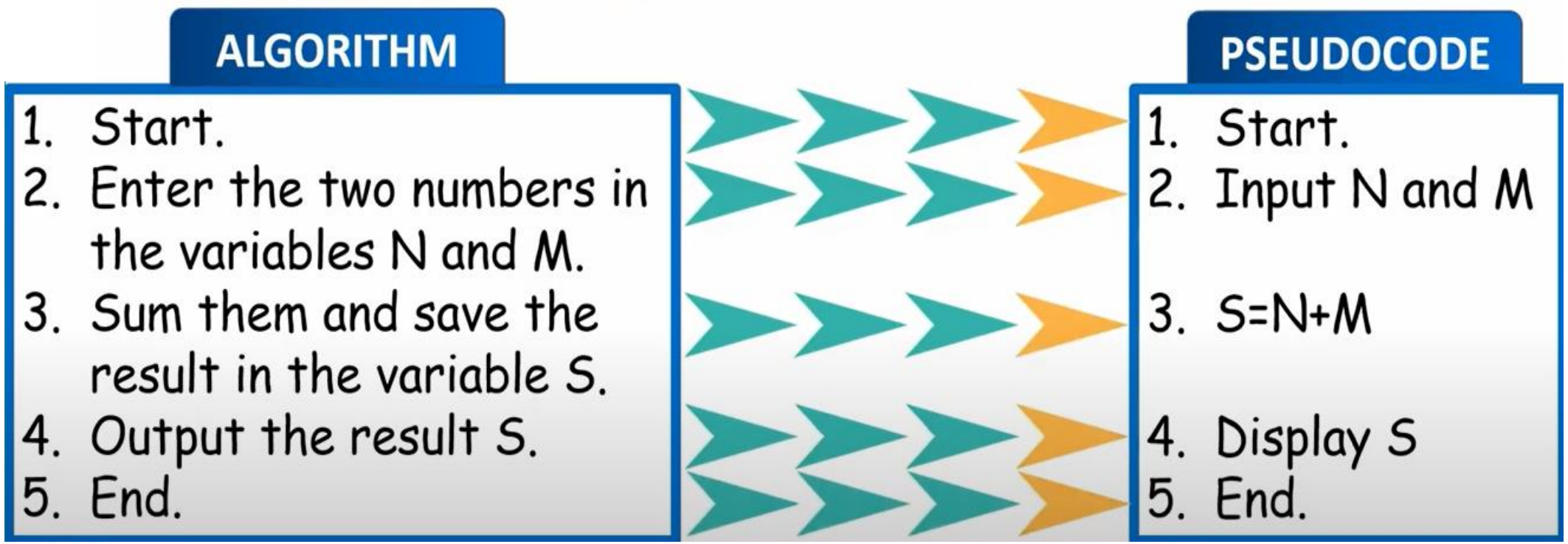
- Pseudo code is an informal language that has no syntax rules and is not meant to be compiled or executed.

Algorithms and Pseudo codes

- We can easily generate pseudo code from the algorithm.
- The pseudo code can be written in simple English by using short words.
- The pseudo code can be written in a format similar to that of a high-level programming languages such as Matlab, C++, and python, but in a simplified version without adhering to strict syntax that programming languages use.

Algorithms and Pseudo codes

Create an algorithm to find the sum of any two numbers.



General approaches to algorithm design:

- Divide and conquer
- Greedy method
- Dynamic programming
- Basic search and traversal technique
- Graph theory
- Linear programming
- Approximation algorithm
- Np problem

The study of Algorithm

- How to devise algorithms.
- How to express algorithms (Pseudo codes).
- How to validate algorithms.
- How to analyze algorithms (Time and Space).
- How to test a program (Testing the Code).

Importance of Analyze Algorithm

- Need to recognize limitations of various algorithms for solving a problem.
- Need to understand the relationship between problem size and running time
- Need to learn how to analyze an algorithm's running time without coding it.
- Need to learn techniques for writing more efficient codes.

Expectation from an algorithm

1) Correctness:

- **Correct**: Algorithms must produce correct result.
- **Produce an incorrect answer:** Even if it fails to give correct results all the time still there is a control on how often it gives wrong result.
- **Approximation algorithm:** Exact solution is not found, but near optimal solution can be found out.

Expectation from an algorithm

2) Less resource usage:

- Algorithms should use less resources. Mainly the **resource usage** can be divided into:

1. Memory (space).
2. Time.

- The **time** is considered to be the primary measure of **efficiency**. We are also concerned with how much the respective algorithm involves the computer memory. But mostly time is the resource that is dealt with. And the actual **running time** depends on a variety of backgrounds: like the **speed of the computer**, the **language** in which the algorithm is implemented, the **compiler**/interpreter, **skill of the programmers** etc.

Basic time analysis

Basic time analysis

- The main goal is to learn how to analyze algorithms for efficiency.

- The analysis of algorithms attempts to optimize some critical resource. The critical resource usually chosen is running time.

- Time is effected by what?

    1. Hardware.
    2. Compiler.
    3. Input of the algorithm (input size).

Basic time analysis (continued)

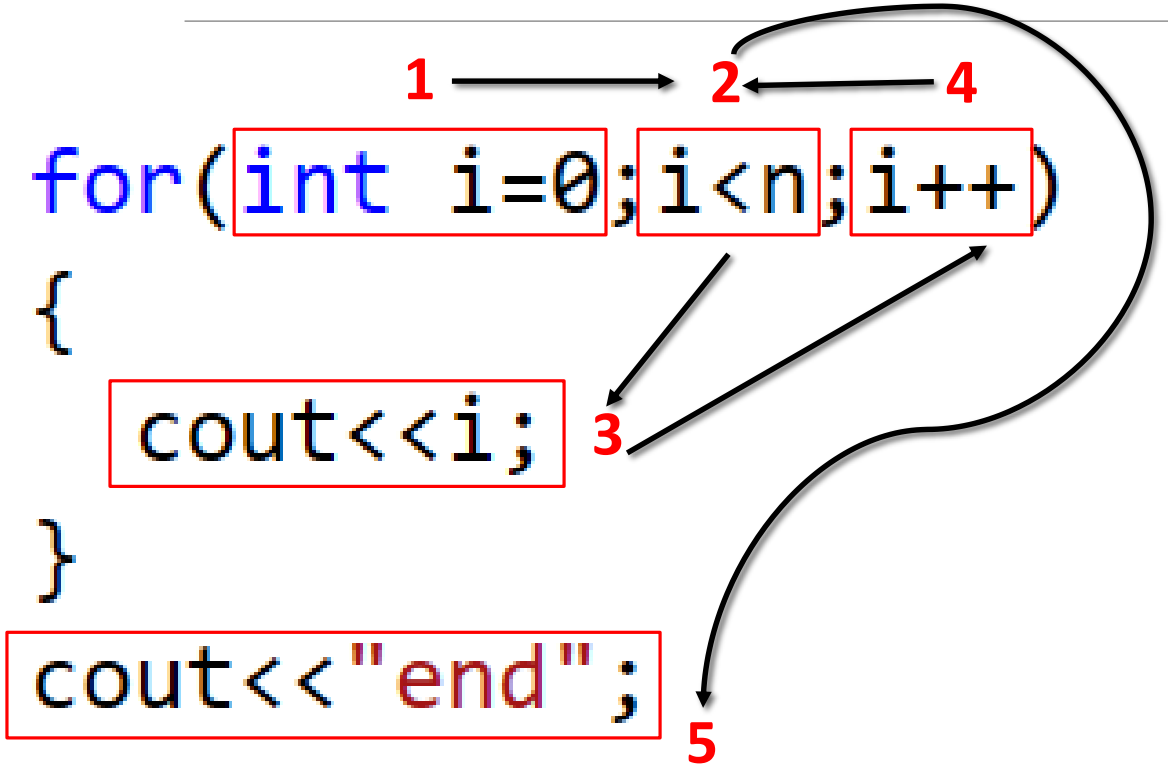
- The best notion for input size depends on the problem being studied. For many problems, such as sorting or computing discrete Fourier transforms, the most natural measure is the number of items in the input: for example, the array size n for sorting.

- The running time of an algorithm refers to the length of time it takes for it to run as a function.

- The most significant factor effecting on running time is n. Thus, running time represents as a function in *n*, T(n).

- An algorithm with more operations will have a lengthier running time. The number of operations of an algorithm in proportion to the size of an input also affects the algorithm's running time.

Determine the relationship between the input size and running time (Example 1)



#	Times
1	1
2	n+1
3	n
4	n
5	1

Assume that each statement (**CPU**) takes time **c,** where **c** is a constant.
F(n)=T(n)= c*(1+(n+1)+n+n+1) = c*(3n+3)

EX: If c=1 milliseconds
T(5)= 18 milliseconds
T(1000)=3003 milliseconds

Example 2

The most Tricky statement

If n is even, How many times statement 1, 2, 3, 4 are executed?

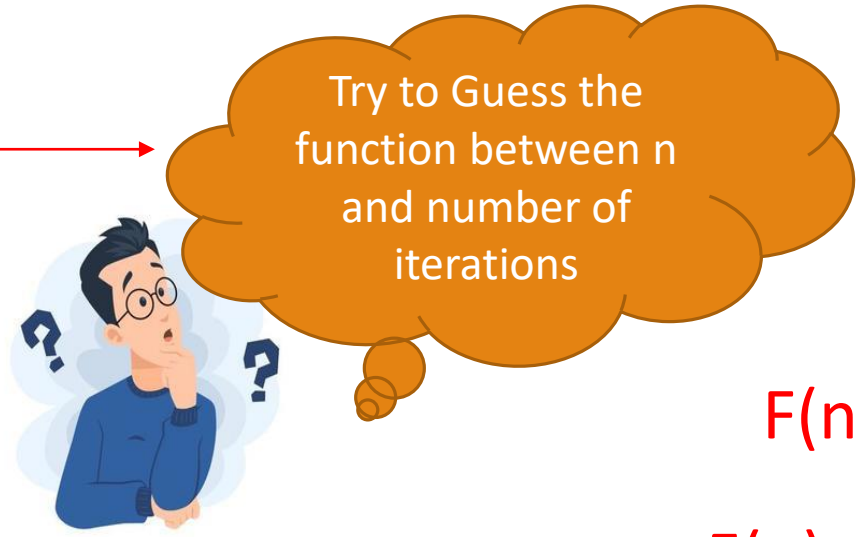
```
1        2        4
for(int i=0;i<n;i+=2)
{
   cout<<i;  3
}
```

#	Times
1	1
2	
3	
4	

Test: n=6

i=0	i<6?	T
i=2	i<6?	T
i=4	i<6?	T
i=6	i<6?	F

4 iterations



F(n)=n-2

OR

F(n)= (n/2)+1

Example 2

The most Tricky statement

```
1          2          4
for(int i=0;i<n;i+=2)
{
   cout<<i;  3
}
```

How many times statement 1, 2, 3, 4 are executed?

#	Times
1	1
2	
3	
4	

Test: n=8

i=0	i<8?	T
i=2	i<8?	T
i=4	i<8?	T
i=6	i<8?	T
i=8	i<8?	F

5 iterations

We can conclude that

F(n)=n-2 X

F(n)= (n/2)+1 ✓

Example 2

The most Tricky statement

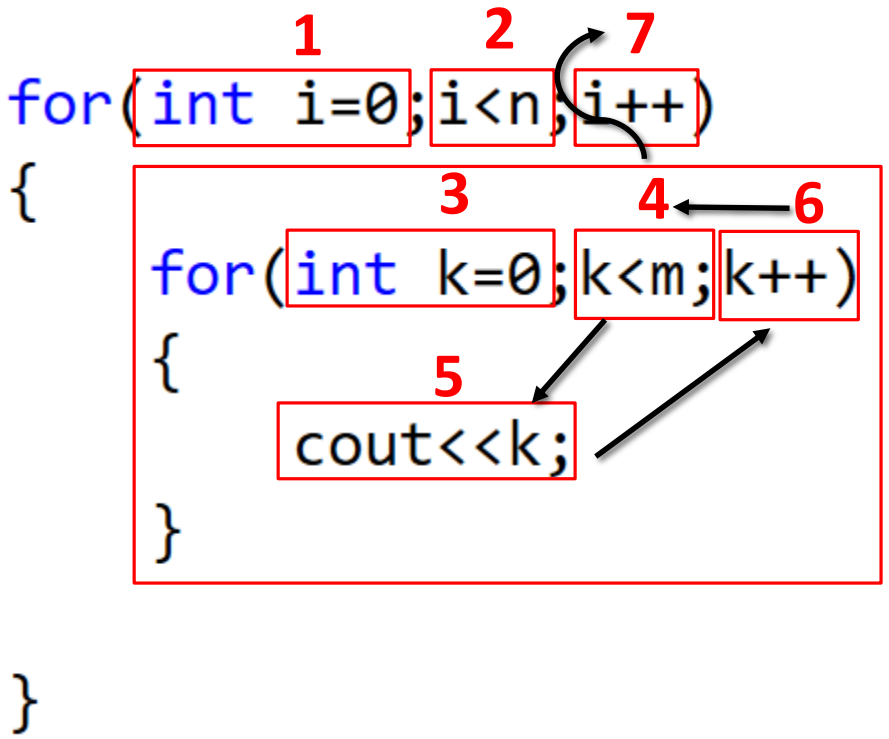
```
1          2        4
for(int i=0;i<n;i+=2)
{
   cout<<i;  3
}
```

How many times statement 1, 2, 3, 4 are executed?

#	Times
1	1
2	(n/2)+1
3	n/2
4	n/2

Assume that each statement **(CPU)** takes time **c,** where **c** is a constant.
F(n)=T(n)= c*(1+((n/2)+1)+(n/2)+(n/2)) = c*(3(n/2)+2).

Example 3 (nested loop)



#	Times
1	1
2	n+1
3	1
4	m+1
5	m
6	m
7	n

n

#	Times
1	1
2	n+1
3	n
4	(m+1)*n
5	m*n
6	m*n
7	n

Assume that each statement (CPU) takes time c, where c is a constant.

$F(n)=T(n)= c*(1+(n+1)+n+ ((m+1)*n))+(m*n)+(m*n)+n)$

$F(n)=T(n)=2+4n+3mn$

Example 4

```
int n=2,m=3;
          1          2     10
for(int i=0;i<n;i++)
{
                   3     4     9
     for(int j=0;j<m;j++)
     {
                        5     6     8
          for(int k=0;k<m;k++)
          {
                    7
               cout<<k;
          }
     }
}
```

2 How many times statement 10 is executed?

6 How many times statement 9 is executed?

18 How many times statement 7 is executed?

If c=1 then T(n)=?

$T(n)= 2+4n+4mn+3nm^2$