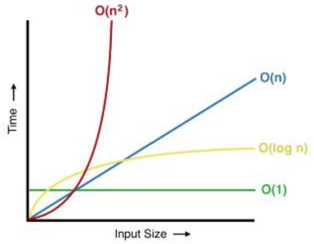
Data Structures and Algorithms

Big O Notation and Time Complexity



Prepared by:

Eng. Malek Al-Louzi

School of Computing and Informatics – Al Hussein Technical University

Spring 2021/2022

Outlines

- Time Complexity
- Big O notation
- Big O notation and Time Complexity
- General Rules

What is a time complexity of a program and why it is important?

Time Complexity (1/8)

- Let us begin with an example.
- John and Sarah are two students, they have both been given an assignment to find a whether a number is a prime or not.
- A prime number is a number that can be divided by exactly two numbers, One and the number itself.
- Example: 2, 3, 5, 11…etc.

Time Complexity (2/8)

- Both John and Sarah figured out different solutions for this Assignment.

John

```
for i ← 2 to n-1
    if i divides n
        n is not prime
```

Sarah

```
for i ← 2 to √n
    if i divides n
        n is not prime
```

- Both solutions will work fine.

Time Complexity (3/8)

- When they both run their program for large input sizes, Sarah was happy, and John was sad.

John

for $i \leftarrow 2$ to $n-1$
    if i divides n
        n is not prime

Sarah

for $i \leftarrow 2$ to $\sqrt{n}$
    if i divides n
        n is not prime

Time Complexity (4/8)

- Assume Computer will take 1mS for division operation:

John ☹

1 ms for a division

$(n-2)$ times

$n = 11$ — 9 ms

$n = 101$ — 99 ms

$n = 1000003 = 10^6 + 3$ — $\approx 10^6$ ms $= 10^3$ sec $= 16.66$ min

$n = 10000000019 = 10^{10} + 19$ — $\approx 10^{10}$ ms $= 10^7$ Sec ≈ 115 days

Sarah ☺

$(\sqrt{n} - 1)$ times

$(3-1) = 2$ ms

9 ms

$(\sqrt{10^6 + 3} - 1) \approx 10^3$ ms $= 1$ sec

$\approx 10^5$ ms $= 100$ Sec $= 1.66$ min

Time Complexity (5/8)

- The correctness of the program is not the only thing that we care about.
- How the program behaves for larger input sizes is also important.
- We should always try to write a program that behaves well for larger input sizes.

Example 1: Simple Array (1/2)

- You are given an array like this:

```
given_array = [1, 4, 3, 2, ..., 10]
```

- We want to write a function takes an array and returns the sum of all the elements in this array.
  - This array could be any length.

```
static int sumArray(int []n) {

    int sum = 0;

    for(int i = 0; i < n.length; i++) {
        sum += n[i];
    }

    return sum;
}
```

Example 1: Simple Array (2/2)

- How much time does it take to run this function?
- Hard question to answer, It depends on number of factors:

Running time depends upon:

1) Single vs multi processor
2) Read/write speed to memory
3) 32 bit vs 64-bit
4) Input

↳ rate of growth of time

Big O notation and Time Complexity

- **How does the runtime of this function grow?**
- It is easier to answer.
- It will depend on the size of the input (how many elements there are in the given array).
- We will use a pair of tools called: **Big O notation** and **Time Complexity**.

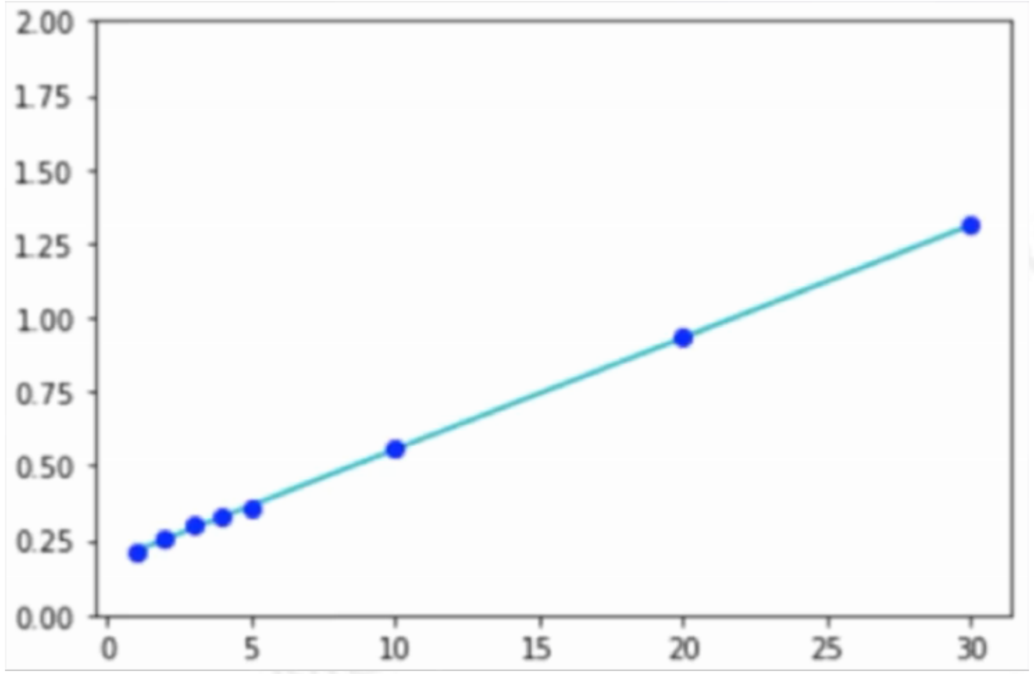
*Runtime: time it takes to execute a piece of code.

Let us complete about

“Time complexity”

Time Complexity (6/8)

- The following is the average time it takes to run the previous function (sumArray) for different number of sizes (number of elements in the array):

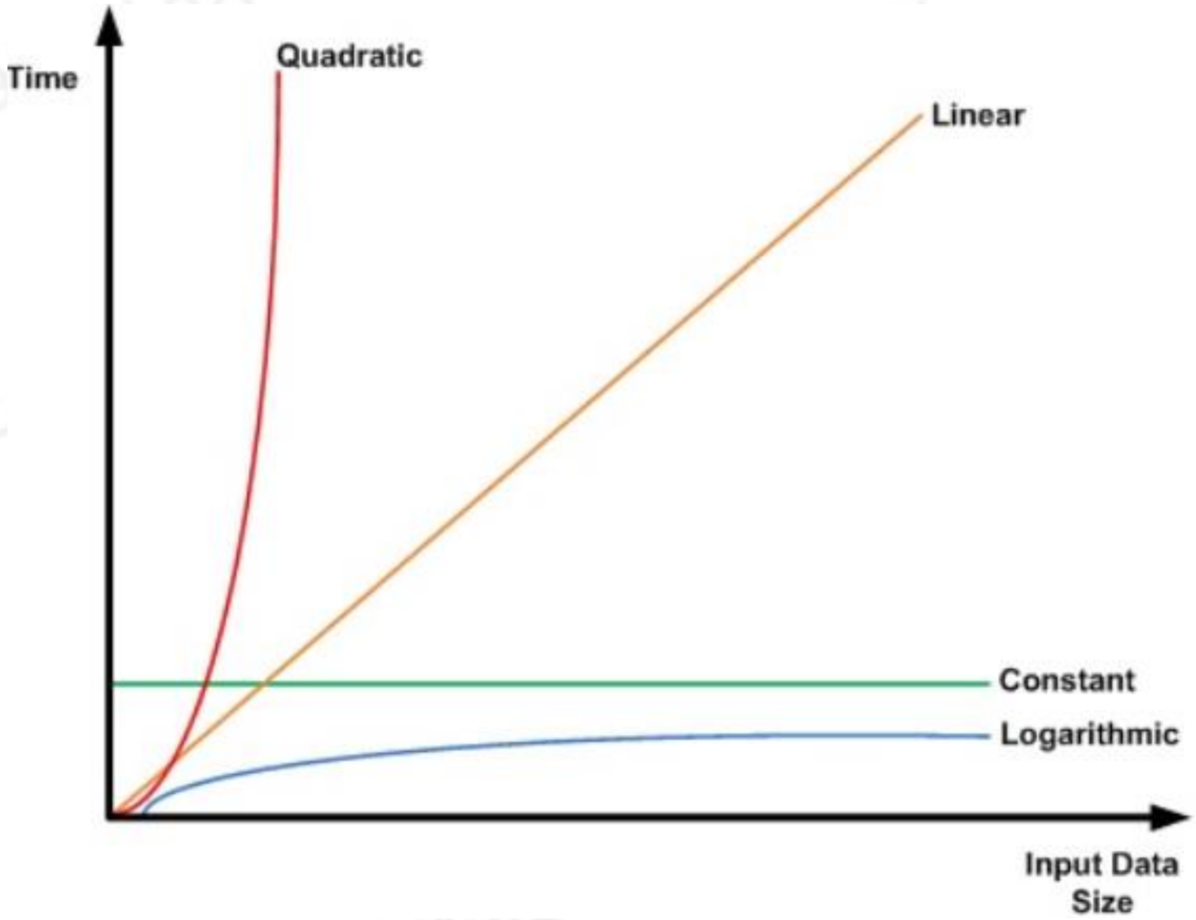


*X axis: the size of the input (n), Y axis: the average time in Microseconds.

Time Complexity (7/8)

- Time complexity for the previous example: **Linear Time**.
- Time complexity: is a way of showing how the runtime of a function increases as the size of the input increases.
- There are many different types:
  - Linear Time.
  - Constant Time.
  - Quadratic Time.
  - Logarithmic Time.

Time Complexity (8/8)



Big O notation

- Another mathematical way to express different types of time complexity.

linear time	$O(n)$
constant time	$O(1)$
quadratic time	$O(n^2)$

Big O notation and Time Complexity (1/5)

- By changing what is inside Big O expression, we can express different types of time complexity.
- How do we know what is inside the Big O expression exactly?
- From the graph of the previous example (sumArray):

$$T = an + b$$

- Where n is the size of the input, a and b are two constants.

Big O notation and Time Complexity (2/5)

- We need to follow two steps:

1. find the fastest growing term
2. take out the coefficient

- After applying the two steps:

$$T = an + b = O(n)$$

Example 1

- Another example:

$$T = cn^2 + dn + e = O(n^2)$$

- In Computer Science we care more about larger inputs than smaller inputs, because when the input is small the execution time is fast.

Example 2 (1/2)

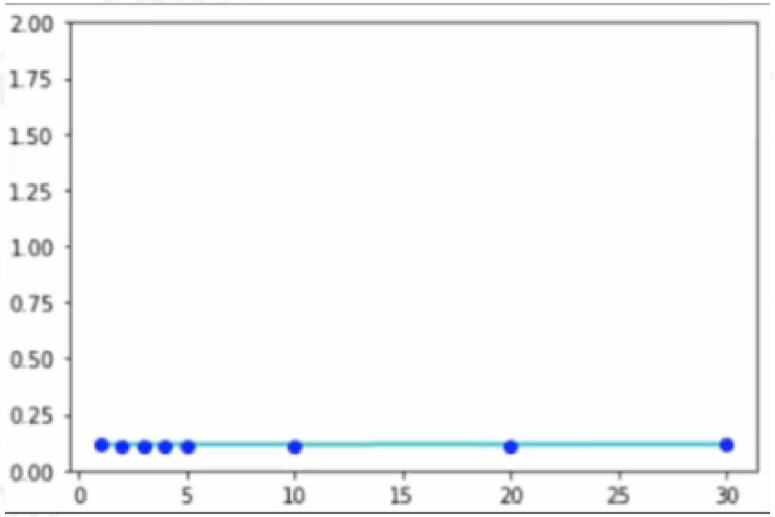
- Another example:

```java
static int testFunction(int []n) {

    int total = 0;

    return total;
}
```

- After measuring the average execution time for different n:



Example 2 (2/2)

- To express the time complexity and Big O notation:

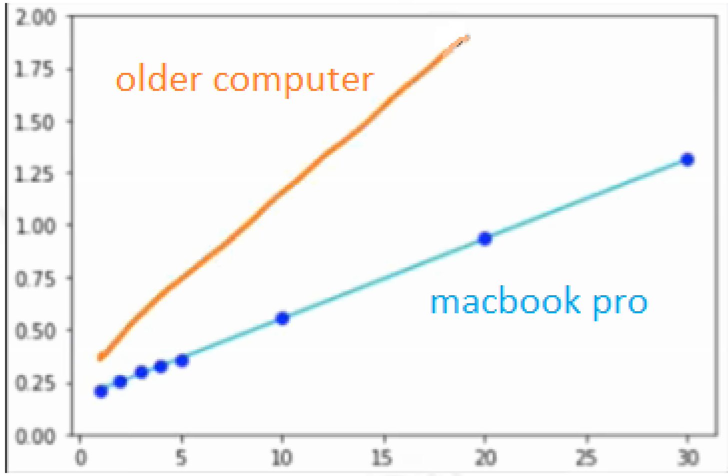
$$T = c = 0.115$$

$$= 0.115 \times 1 = O(1)$$

constant time

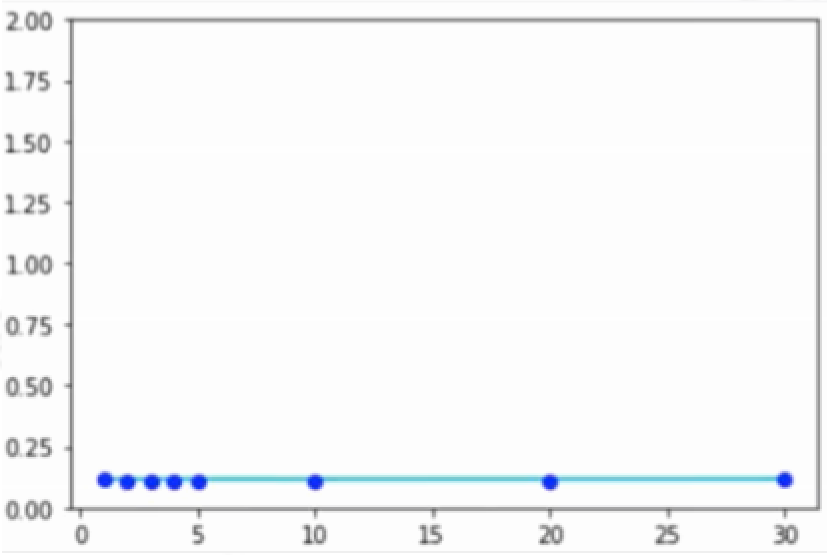
Big O notation and Time Complexity Features

- Show how the function behaves for large inputs.
- **It doesn’t depend on a particular environment.**

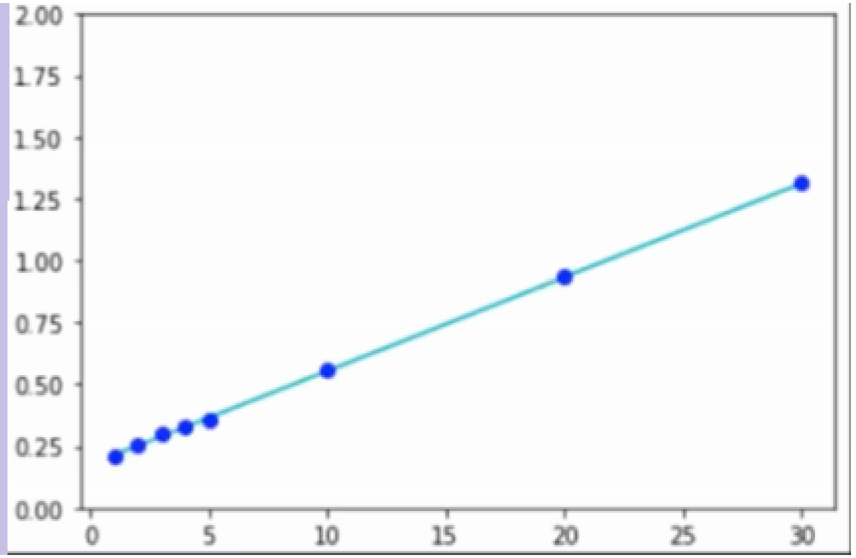


Big O notation and Time Complexity Features

- If we have two functions like this:



Function 1



Function 2

Big O notation and Time Complexity Features

- When we compare the two functions the second function takes more time for large n, regardless where the constant is in the first function.
- This is another feature of time complexity and Big O notation.
  - **Allowing us to quickly compare the performance of multiple functions.**

Big O notation and Time Complexity (3/5)

- So far, we running the function and measure the average execution time for multiple n and plot the result.
- A lot of work to find Time complexity and Big O.
- Is there any faster way?
- If we want to find them for this function, without running any experiments:

```
static int testFunction(int []n) {

    int total = 0;

    return total;
}
```

Big O notation and Time Complexity (4/5)

- We want to consider each of the function lines.
- How much time does it take to execute each of these lines?

```
static int testFunction(int []n) {

    int total = 0;      -> O(1)

    return total;       -> O(1)
}
```

- Once we have Big O expression for each line, we can simply add them up to find the total amount of time.

$$T = O(1) + O(1) = c_1 + c_2$$
$$= c_3 = c_3 \times 1 = O(1)$$

Big O notation and Time Complexity (5/5)

- Remember O(1) is just a constant.
- So, in general:

$$O(1) + O(1) = O(1)$$

Example 3

- Another example:

```
static int sumArray(int []n) {

    int sum = 0;          -> O(1)

    for(int i = 0; i < n.length; i++){
        sum += n[i];  -> O(1)
    }

    return sum;          -> O(1)
}
```

- Adding them up:

$$T = O(1) + n \times O(1) + O(1)$$

- Following the same rules:

$$= c_4 + n \times c_5 = O(n)$$

Example 4 (1/4)

- 2d array is basically an array of arrays.

```
array_2d = [[1, 4, 3],
            [3, 1, 9],
            [0, 5, 2]]
```

```
array_2d = [[1, 4, 3, 1],
            [3, 1, 9, 4],
            [0, 5, 2, 6],
            [4, 5, 7, 8]]
```

Example 4 (2/4)

- Let us call the number of rows n.
- Suppose we always given a square 2d array.
- Also, suppose Number of columns = number of rows.
- So, we have n^2 elements.
- We want to write a function that takes a 2d array and returns the sum of all the elements inside it.

```
static int sumArray(int [][]n) {

    int sum = 0;

    for(int i = 0; i < n.length; i++) {
        for(int j = 0; j < n[i].length; j++) {
            sum += n[i][j];
        }
    }

    return sum;
}
```

Example 4 (3/4)

- We want to find Time complexity and Big O of this function.

```
static int sumArray(int [][]n) {

    int sum = 0;                    -> O(1)

    for(int i = 0; i < n.length; i++) {
        for(int j = 0; j < n[i].length; j++) {
            sum += n[i][j];         -> O(1)
        }
    }

    return sum;                     -> O(1)
}
```

- To find the total amount of time, we need to add them up.

$$T = O(1) + n^2 \times O(1) + O(1)$$

Example 4 (4/4)

- Following the previous steps:

$$T = O(1) + n^2 \times O(1) + O(1)$$
$$= c_6 + n^2 \times c_7 = O(n^2)$$

- Which is a **Quadratic** time complexity.

Example 5 (1/2)

- In the previous example, if we want to increase the sum to each element.

```java
static int sumArray(int [][]n) {

    int sum = 0;

    for(int i = 0; i < n.length; i++) {
        for(int j = 0; j < n[i].length; j++) {
            sum += n[i][j];
        }
    }

    for(int i = 0; i < n.length; i++) {
        for(int j = 0; j < n[i].length; j++) {
            n[i][j] += sum;
        }
    }

    return sum;
}
```

Example 5 (2/2)

- How would that affect the Big O notation and time complexity?

$$T = O(2n^2) = O(n^2)$$

- Why?
- In this case, T in a full form will be:

$$T = 2n^2 \times c + \ldots = 2n^2 \times c + c_2 n + c_3$$
$$= (2c) \times n^2 + c_2 n + c_3 = O(n^2)$$

General Rules

- Fragment 1: Simple Statements.

```
int a;
a = 5
a++;
```

Simple statements

Fragment 1

O(1)

General Rules

- Fragment 2: Single loop.

```
for(i=0;i<n;i++)
{
  // simple statements
}
```

Single loop

Fragment 2

$O(n)$

General Rules

- Fragment 3: Nested loop.

```
for(i=0;i<n;i++)
{
  for(j=0;j<n;j++)
  {
     // simple statements
  }
}
```

Nested Loop

Fragment 3

$O(n^2)$

Example 6

```
Function ( )
{  int a;
   a = 5
   a++;
   for(i=0;i<n;i++)
   {
      // simple statements
   }
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         // simple statements
      }
   }
}
```

int a; a = 5; a++; → $O(1)$

first for loop → $O(n)$

nested for loops → $O(n^2)$

$$T = \underbrace{O(1) + O(n)} + O(n^2) = O(n^2)$$

General Rules

- The fragment which has the maximum running time decides the overall running time of the program.
- What about conditional statements?
  - When we analyze time complexity, we always try to analyze it in the worst case.

```
Function ( )
{
  if (some condition)
  {
    for(i=0;i<n;i++)
    { //simple statements
    }
  }
  else
  { for(i=0;i<n; i++)
    { for(j=0;j<n;j++)
      { // simple statements
      }
    }
  }
}
```

Example 7

$T(n) = O(n^2)$

```
Function ( )
{
  if (Some Condition)
  {
     for(i=0; i<n; i++)
     {  //simple statements                O(n)
     }
  }
  else
  {  for(i=0; i<n; i++)
     {  for(j=0; j<n; j++)                 O(n²)
        {  //simple statements
        }
     }
  }
}
```

General Rules

Rule: Conditional Statements

Pick complexity of condition which is worst case

Any Questions???...