



College of computer science & mathematics

Dep. Of Computer Science

DATA STRUCTURE

هياكل البيانات

Lecture 2 :
Complexity analysis

Prepared & Presented by
Mohammed B. Omar

2024 -2025

Algorithms in Computer Science: Definition

Algorithms can be seen as the backbone (العمود الفقري) of computer science as they form the basis for creating **efficient and effective software programs** (لأنشاء برامج ذات كفاءة عالية وفعالة).

They are a set of well-defined rules or instructions used to solve a specific problem.

An algorithm is defined as a sequence of steps that accomplishes a particular task if followed .

Algorithm Meaning

An algorithm in Computer Science is a well-structured, unambiguous (غير غامضة) and step-by-step set of instructions used to solve a problem or achieve a certain objective.

In addition, every algorithm must satisfy the following criteria.

1. Input: algorithm must take zero or more input.
2. Output: algorithm must produce at least one output
3. Definiteness: The definiteness property insists that each step in the algorithm is unambiguous. A step is said to be unambiguous if it is clear, in the sense that, the action specified by the step can be performed without any dilemma/confusion. يقال إن الخطوة لا لبس فيها إذا كانت واضحة، بمعنى أنه يمكن تنفيذ الإجراء المحدد بالخطوة دون أي معضلة/ارتباك.
4. Finiteness: The finiteness property states that the algorithm must terminate its execution after a finite number of steps (or after a finite period). That is to say, it should not get into an endless process
5. Effectiveness: Each step must be easily convertible to code.

Representation of algorithm can written By:-

In natural language (English) / pseudo-code / diagrams (Flow chart) / etc.

Pseudo- code:-

A mixture of natural language and high – level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm. Pseudo-code is more structured than usual language but less formal than a programming language.

مزيج من اللغة الطبيعية ومفاهيم البرمجة عالية المستوى التي تصف الأفكار الرئيسية وراء التنفيذ العام لبنية البيانات أو الخوارزمية. الكود الزائف أكثر هيكلية من اللغة المعتادة ولكنه أقل رسمية من لغة البرمجة.

E.g.:- Algorithm arrayMax (A, n)

input: An array A sorting n integers s3a3zs4cvb

output: The Maximum element in A

currentMax \leftarrow A[0]

for i \leftarrow 1 to n-1 do

if currentMax < A[i] then currentMax \leftarrow A[i]

return currentMax

What Makes a Good Algorithm?

Suppose you have two possible algorithms or data structures that basically do the same thing; which is better?

- ✓ Faster.
- ✓ Less space.
- ✓ Easier to code.
- ✓ Easier to maintain.

Importance of Data Structures in Algorithms

Data structures are crucial (حاسمة في) in the development and implementation of efficient algorithms.

Algorithms utilize data structures to **solve computational problems**, and **choosing the right data structure** can mean the difference between a solution and an optimal solution. A well-chosen data structure can improve an algorithm's efficiency dramatically.

A **Data Structure** in Computer Science is a way of **organizing** and **storing** data so that operations such as **insertions, deletions, and searches** can be done efficiently.

Understanding Algorithm Analysis

Algorithm Analysis is at the heart of computer science, serving as a **toolset that allows you to evaluate and compare the performance of different algorithms in solving specific tasks**. (هو بمثابة مجموعة أدوات تسمح لك بتقييم ومقارنة أداء الخوارزميات المختلفة في حل مهام محددة.)

Measure Algorithm Efficiency

Space utilization: amount of memory required.

Time efficiency: amount of time required to accomplish the task.

As space is not a problem nowadays

- **Time efficiency is more emphasized.**
- **But, in embedded computers or sensor nodes, space efficiency is still important.**

COMPLEXITY ANALYSIS

To gain a more detailed understanding of the complexity analysis, let's examine its types:

- ❖ **Time Complexity:** This examines the total amount of time an algorithm takes to run as a function of the size of its input.
- ❖ **Space Complexity:** This analyses the amount of memory an algorithm uses concerning the size of its input.

For Example

لنفترض أنك تواجه التحدي المتمثل في اختيار خوارزمية لمعالجة كمية هائلة من البيانات. نظرًا لقلّة معرفتك بتحليل التعقيد، قد تختار عن طريق الخطأ حلاً يتباطئ بشكل كبير مع زيادة حجم البيانات. إن الاعتراف بتحليل التعقيد وفهمه من شأنه أن يرشدك في اختيار خوارزميات عالية الكفاءة، وتحسين سرعة معالجة البيانات والأداء.

Big O notation: is the language we use for talking about how long an algorithm takes to run. It's how we compare the efficiency of different approaches to a problem. It is used in Computer Science to describe the performance or complexity of an algorithm. Big O specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

Note

Imagine you have a list of 10 objects, and you want to sort them in order. There's a whole bunch of algorithms you can use to make that happen, but not all algorithms are built equal. It is simply a way of comparing algorithms and how long they will take to run.

Note

❖ O : order

❖ كلما زادت N سوف يزداد الوقت التنفيذي للكود وكلما قلت يقل الوقت التنفيذي

Key Concepts in Algorithm Analysis

To begin, it's vital to understand basic terminologies involved in algorithm analysis.

- ❖ **Big O Notation:** This notation describes an **upper bound of the complexity of an algorithm**. It provides an approximation of the maximum time taken by an algorithm for all input sizes.
- ❖ **Big Ω Notation:** This notation describes a **lower bound of the complexity**, determining the minimum time required by an algorithm.
- ❖ **Big Θ Notation:** This notation characterizes **both the upper and lower bounds of the complexity**. It denotes the exact asymptotic behavior.
- ❖ **Asymptotic Analysis:** (التحليل المتقارب) This is the method of describing **limiting behavior** and often ties in closely with the aforementioned notations.

ترتبط ارتباطاً وثيقاً


Big O , big Ω , and big Θ Notation are analytical tools (أدوات تحليلية) used to describe an algorithm's efficiency as the size of its input approaches infinity. هي أدوات تحليلية تستخدم لوصف كفاءة الخوارزمية عندما يقترب حجم مدخلاتها من اللانهاية. Big O و big Ω و big Θ .

General Rules

- Ignore constants (Considers long term growth only)
5n becomes $O(n)$
- Terms are dominated by others
- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!)$

Big O – Notation

$$\begin{array}{l} f(n): 6n^2 + 100n + 300 \longrightarrow O(n^2) \\ f(n): 2n^2 + 150n + 330 \longrightarrow O(n^2) \\ f(n): 2n^3 + 3n^2 + 100n \longrightarrow O(n^3) \\ f(n): 4\log_2(n) + 40 \longrightarrow O(\log_2(n)) \end{array}$$



- Used to measure the **performance** of any algorithm.
- By providing the **order of growth** of the function.

$O(1)$

- **$O(1)$ describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.**

$O(N)$

- **$O(N)$ describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set.**
- **Big O notation will always assume the upper limit where the algorithm will perform the maximum number of iterations.**

$O(N^2)$

- **$O(N^2)$ represents an algorithm whose performance is directly proportional to the square of the size of the input data set.**
- **This is common with algorithms that involve nested iterations over the data set.**
- **Deeper nested iterations will result in $O(N^3)$, $O(N^4)$ etc.**

$O(2^N)$

- $O(2^N)$ denotes an algorithm whose growth doubles with each addition to the input data set.
- An example of an $O(2^N)$ function is the recursive calculation of Fibonacci numbers.

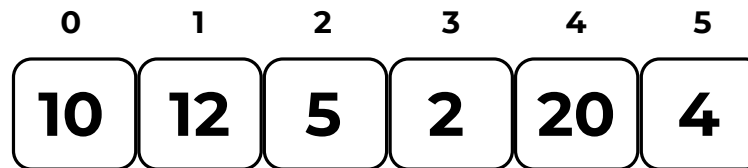
$O(\log N)$.

- The iterative halving of data sets described in the binary search example produces a growth curve that peaks at the beginning and slowly flattens out as the size of the data sets increase.
- e.g. an input data set containing 10 items takes one second to complete, a data set containing 100 items takes two seconds, and a data set containing 1000 items will take three seconds.

	<i>constant</i>	<i>logarithmic</i>	<i>linear</i>	<i>N-log-N</i>	<i>quadratic</i>	<i>cubic</i>	<i>exponential</i>
<i>n</i>	$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

COMPLEXITY ANALYSIS

Example : Search for a specific element within the following array?



1. There is an algorithm that will work a certain way to search for the item.
2. I can find the element in the first place and this is called (Best Case) or (Lower bound).
3. I can find the element in the last place and this is called (Worst Case) or (Upper bound).
4. I can find the element in the middle place and this is called (Average Case). or (Tight bound)

Best Case

Omega Notation

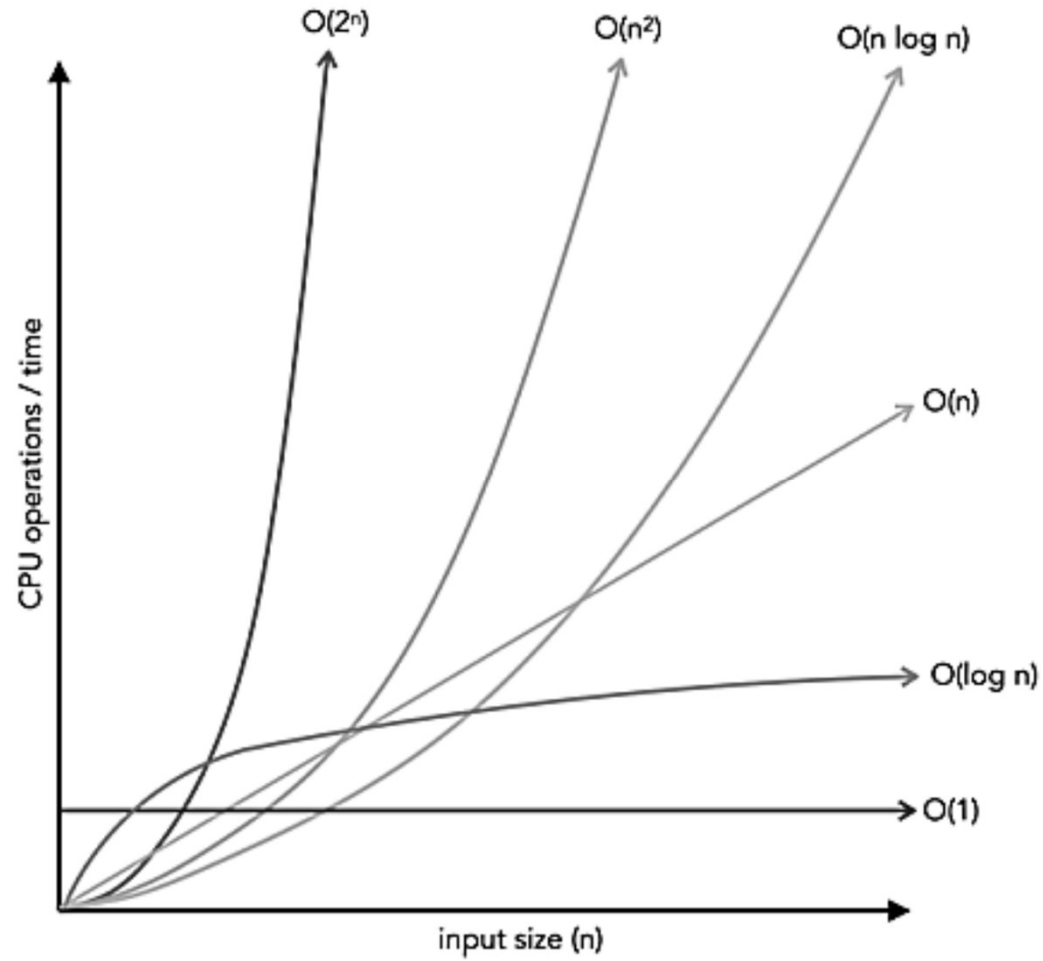
Average Case

Theta Notation

Worst Case

Big O Notation

Our focus on the worst cases to address them for the better.



Rules:

Int X;

no computation

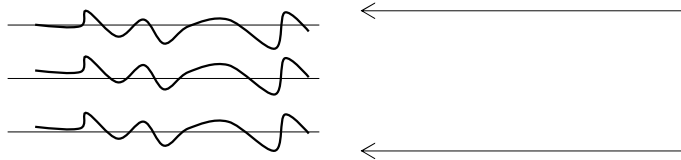
Int y = 4;

Computation

Example

```
Public Void max (int a , int b)
```

```
{
```



```
}
```

no computation

no computation

Computation

no computation

Rules: FOR

Example:

Write method to calculate

$$\sum_{i=1}^n$$

by using $\frac{n(n+1)}{2}$

قانون المتوالية الحسابية

```
Public int sum (int n)
```

```
{
```

```
    int total ;
```

```
    total = n*(n+1)/2
```

```
    return total;
```

```
}
```

no computation

no computation

no computation

1

1

no computation

$$1+1 = 2 \Rightarrow O(1)$$

Note

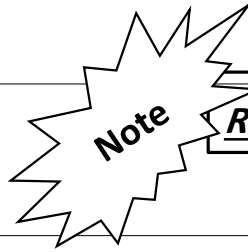
أي رقم مهما كانت قيمته يتحول إلى $O(1)$

Rules: FOR

Example:

Write method to calculate

$$\sum_{i=1}^n \cdot$$



Rule (For loop) = End of condition – Start condition + 1

Public int sum (int n)	→	no computation
{	→	no computation
int i, total;	→	no computation
total = 0;	→	1
for (i = 1; i<=n; i++)	→	n-1+1+1= n+1
{	→	no computation
total = total + i;	→	n
}	→	no computation
return total;	→	1
}	→	no computation

$$1 + (n+1) + (n) + 1 = 2n + 3 \Rightarrow O(n)$$

Rules: FOR

Example:

Write method to calculate

$$\sum_{i=1}^{20} i.$$

by using (for)?

```
Public int sum ( )
```

```
{
```

```
    int i, total ;
```

```
    total = 0
```

```
    for(i=1; i<=20; i++)
```

```
    {
```

```
        total = total + i;
```

```
    }
```

```
    return total;
```

```
}
```

no computation

no computation

no computation

1

20 -1 + 1 + 1 =21

no computation

20

no computation

1

no computation

$$1+21+20+1=43 \Rightarrow O(1)$$

Note

أي رقم مهما كانت قيمته يتحول إلى $O(1)$

Rules: FOR

Example:

Compute step execution and big O for the following Code?

```
X = 0;
For ( l = -2 ; i<n ; i++)
{
    x = x + 1;
    y = x + 2;
}
System.out.print(x);
```

```
1
n - (-2) + 1 = n+3
n + 2
n + 2
1
```

$$1+(n+3)+(n+2)+(n+2)+1=3n+9 = O(n)$$

Rules: FOR

Example:

Compute step execution and big (O) for the following Code?

```
Int func (a[ ], n)
```

```
{
```

```
    int x = 5;
```

```
    for (l = 1; i<=n ; i++)
```

```
    {
```

```
        for(j=1; j<n; j++)
```

```
        {
```

```
            x = x + l + j;
```

```
            s.o.p(x);
```

```
        }
```

```
    }
```

```
}
```

N-1=N

1

$n-1+1+1 = n+1$

$n-1+1 = n$

n-1

n-1

$$1 + (n+1) + n * (n) + n (n-1) + n (n-1)$$

$$= 1 + n+1 + n^2 + n^2 - n + n^2 - n$$

$$= 3n^2 - n + 2$$

$$= O(n^2)$$

**Thank You
&
Good luck**