

Data Structures and Algorithms

Prepared by Prof. Shereen Taie

IT251
Information Technology Department

Lecture 2



Data Structures and Algorithms

Lecture 2

IT251

Information Technology Department

Prepared by
Prof. Shereen Taie

Algorithm

Chapter

01

Chapter

01

Algorithm Design



- **Algorithm can be described in three ways:**

- **Natural language like English**
- **Graphic representation called flowchart**
- **Pseudo-code Method**

Algorithm Design:

1. Natural language like English:

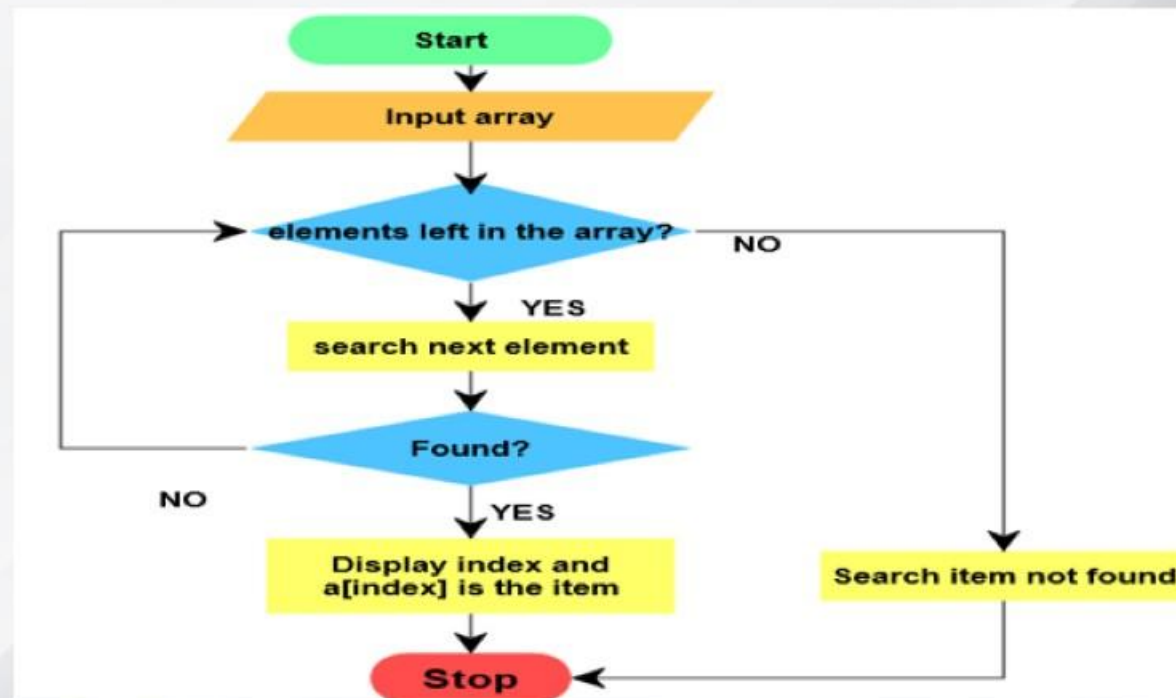
When this way is chosen care should be taken, **we should ensure that** each & every statement is definite.

- Starting with the first item in S ,
- compare x with each item in S in sequence until x is found or until S is exhausted.
- If x is found, answer yes; if x is not found, answer no.

Algorithm Design:

2. Graphic representation called flowchart:

This method will work well when the algorithm is small & simple.



Algorithm Design:

3. Pseudo-code Method:

In this method, we should typically describe algorithms as program.

- The pseudocode is similar, but not identical, to C++.
- Not discuss the implementation of algorithms in any particular programming language.
- It aims to present algorithms clearly so they can be readily understood and analyzed.

```
if (low ≤ x ≤ high) {  
    ⋮  
}
```

rather than

```
if (low <= x && x <= high) {  
    ⋮  
}
```

```
exchange x and y;
```

rather than

```
temp = x;  
x = y;  
y = temp;
```

PSEUDO-CODE RULES:

1. Comments begin with `//` and continue until the end of line.
2. Blocks are indicated with matching braces `{and}` or Indentation المسافة
3. An identifier begins with a letter. الفارغة
 - The data types of variables are not explicitly declared.
4. Compound data types can be formed with records.

PSEUDO-CODE RULES:

Here is an example:

Here link is a pointer to the record type node. Individual data items of a record can be accessed with -> and period.

```
Node. Record  
{  
  data type - 1    data-1;  
  data type - n    data - n;  
  node * link;  
}
```

PSEUDO-CODE RULES:

5. Assignment of values to variables is done using the assignment statement.

$\langle \text{Variable} \rangle := \langle \text{expression} \rangle$; Or $\langle \text{Variable} \rangle \leftarrow \langle \text{expression} \rangle$;

6. There are two Boolean values **TRUE** and **FALSE**.

- Logical Operators AND, OR, NOT
- Relational Operators $<$, $\leq$, $>$, $\geq$, $=$, $\neq$

PSEUDO-CODE RULES:

7. The following looping statements are employed.

For, while and repeat-until

While Loop:

While < condition > do

{

 <statement-1>

 <statement-n>

}

PSEUDO-CODE RULES:

For Loop:

```
For variable: = value-1 to value-2 do  
{  
    <statement-1>  
    <statement-n>  
}
```

PSEUDO-CODE RULES:

repeat-until:

repeat

<statement-1>

<statement-n>

until<condition>

PSEUDO-CODE RULES:

8. A conditional statement has the following forms.

If <condition> then <statement>

OR

If <condition> then <statement-1>

Else <statement-2>

PSEUDO-CODE RULES:

Case statement:

```
select case(expression)
{
    case 1 : <statement-1>
    case n : <statement-n>
    default : <statement-n+1>
}
```

PSEUDO-CODE RULES:

9- Input and output are done using the instructions read & write.

10- There is only one type of procedure:

- Algorithm, the heading takes the form,
- Algorithm Name (Parameter lists)

PSEUDO-CODE RULES:

As an example, the following algorithm finds & returns the maximum of 'n' given numbers:

```
1. algorithm Max(A,n)
2. // A is an array of size n
3. {
4.     Result := A[1];
5.     for K ← 2 to n do
6.         if A[K] > Result then
7.             Result ← A[K];
8.     return Result;
9. }
```

Chapter

01

How to implement?



Examples for how to implement:

- Given 1000 integers, find how many 3 numbers with sum 400?
 - We can do 3 loops and try every 3 numbers!
 - Hmm..1000000000 operations! Too much **Computations**!
- Given 1000 integers, sort them?
 - Hmm...nothing direct in mind...I need to think more!
- **Input:** Given airport locations and cost of every direct flight,
- **Output:** find flight from Cairo to New York with the **minimum cost**?
 - Hmm...how to represent airports relations!

Examples for how to implement:

- **Input:** Facebook site has 1.4 Billion users,

Output: please suggest to Mostafa all **new friends** from them,

Restrictions: where there is exactly **2 friends** between Mostafa and each of a new friend?

- For every user in the system
- Count intersection of Mostafa users and this user
- If count == 2, we suggest this friend
- Hmm...too many billion operations
- Hmm...**system memory** can't hold even all of that!
- Our solution **performance** is too bad!

Performance:

- In real life applications, we care with many factors
 - E.g. Usability, Security, Maintainability, Reliability...etc
- One of the major factors is the **performance**
- A **memory** and **time efficient** applications are desirable features.
 - Imagine if facebook is a slow application!
- This is where **algorithms** field arises!
 - **Computational solution** **that is efficient** **time** and **memory**

High performance real life apps:



Multiplication!

What's the best way to multiply two numbers?

Grade-school Multiplication:

Input: 2 non-negative numbers, x and y (n digits each)

Output: the product $x \cdot y$

$$\begin{array}{r} 5678 \\ \times 1234 \\ \hline 7006652 \end{array}$$

Grade-school Multiplication:

$$\begin{array}{r} 45 \\ \times 63 \\ \hline 135 \\ 2700 \\ \hline 2835 \end{array}$$

Grade-school Multiplication:

Algorithm description (informal):

compute partial products (using multiplication

& “carries” for digit overflows),

and add all (properly shifted) partial products together

$$\begin{array}{r}
 45 \\
 \times 63 \\
 \hline
 135 \\
 2700 \\
 \hline
 2835
 \end{array}$$

Grade-school Multiplication:

$$\begin{array}{r} 45123456678093420581217332421 \\ \times 63782384198347750652091236423 \\ \hline \end{array}$$


Grade-school Multiplication

$$\begin{array}{r}
 \overbrace{45123456678093420581217332421}^{n \text{ digits}} \\
 \times 63782384198347750652091236423 \\
 \hline
 \end{array}$$



How efficient is this algorithm?
(How many single-digit operations are required?)

Grade-school Multiplication

$$\begin{array}{r}
 \overbrace{45123456678093420581217332421}^{n \text{ digits}} \\
 \times 63782384198347750652091236423 \\
 \hline
 \end{array}$$

☹

How efficient is this algorithm?

(How many single-digit operations
in the worst case?)

n partial products: $\sim 2n^2$ ops (at most n multiplications & n additions per partial product)

adding n partial products: $\sim 2n^2$ ops
(a bunch of additions & “carries”)

Grade-school Multiplication

$$\begin{array}{r}
 \overbrace{45123456678093420581217332421}^{n \text{ digits}} \\
 \times 63782384198347750652091236423 \\
 \hline
 \end{array}$$

☹

How efficient is this algorithm?

(How many single-digit operations
in the worst case?)

n partial products: $\sim 2n^2$ ops (at most n multiplications & n additions per partial product)

adding n partial products: $\sim 2n^2$ ops
(a bunch of additions & “carries”)

$\sim 4n^2$ operations in the worst case

Grade-school Multiplication

n digits

0581217332421

2091236423



THE QUESTION IS...
**CAN WE DO
BETTER?**

How efficient is this?

(How many single-digit

in the worst case

$\sim 2n^2$ ops (at most n multiplications & n

operations per partial product)

g n partial products: $\sim 2n^2$ ops

(a bunch of additions & “carries”)

$\sim 4n^2$ operations in the worst case

Chapter
01

Algorithms Analysis



Algorithms Analysis:

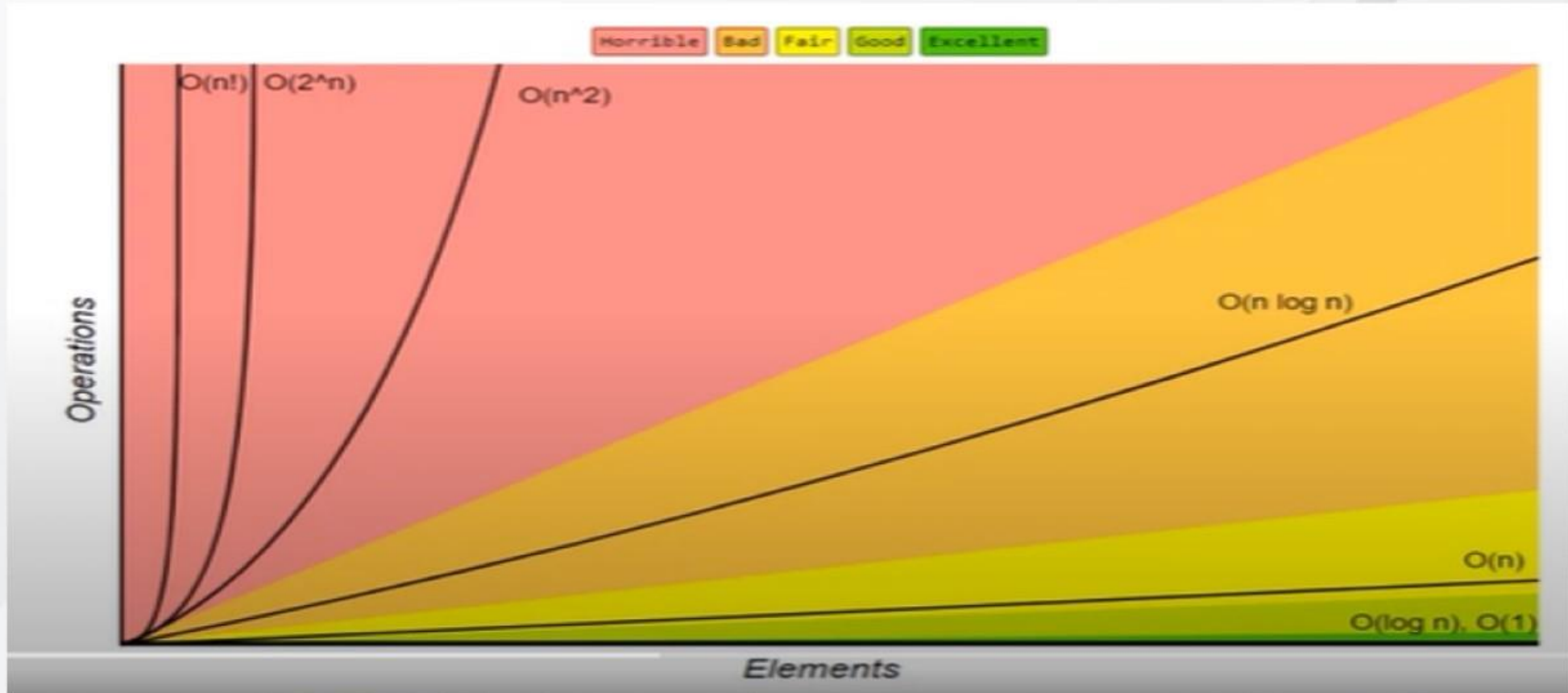
- Given an algorithm, we need to
 - 1) Prove its Correctness
 - 2) Measure time efficiency
 - 3) Measure memory efficiency
- Steps 2 and 3 allow us to **compare** algorithms
 - Fall in complementary area: **Computational Complexity**

Algorithms Analysis:

- An example of sorting algorithms
 - **Selection sort**, orders N numbers using $N*N$ operations
 - **Quicksort**, orders N numbers using $N*\log_2 N$ operations

Then **Quicksort** is faster...and better algorithm

Growth Rates of Common Functions:



Analysis of Algorithm:

The main goal is to determine the cost of running an algorithm and how to reduce that cost.

- **Cost is expressed as Complexity:**
 - **Time Complexity:** how much time an algorithm needs.
 - **Space Complexity:** how much memory an algorithm needs.

Time Complexity

Time Complexity Depends on:

- Machine Speed
- Size of Data and
- Number of Operations needed (n)

Space Complexity

Space Complexity Depends on:

- Size of Data
- Size of Program

Time Complexity

- How to measure absolute time ?
 - A new computer will be faster than an old computer.
 - Super computers vs. Smartphones/ Personal computers.

- **Solution:**

Considering: Number of steps

- It is generic and Machine independent.

Time Complexity

- Expressed as $T(n)$ = number of basic operations required.
- (n) is the Problem Size: n could be the number of specific operations, or the size of data (e.g. an array) or both.
- Not only depends on the input size n , but also depends on the arrangement of the input items.

Time Complexity

- Each "simple" operation (+, -, =, <, >=) is one operation.
- Loops and function calls are not simple operations, but **depend upon the size of the data and the contents of a function.**
- **Each memory access is one operation.**



Thank You