

INTRODUCTION TO ALGORITHM

Chapter1

Prepared by: Enas Abu Samra

KEY POINTS OF CHAPTER1

- Things you will get from this course.
- Inventor of Algorithms.
- Definition of an Algorithm.
- Important problem types.
- Why We Study Algorithms?
- Characteristics of an Algorithm.
- Steps for Developing an Algorithm.

KEY POINTS OF CHAPTER1

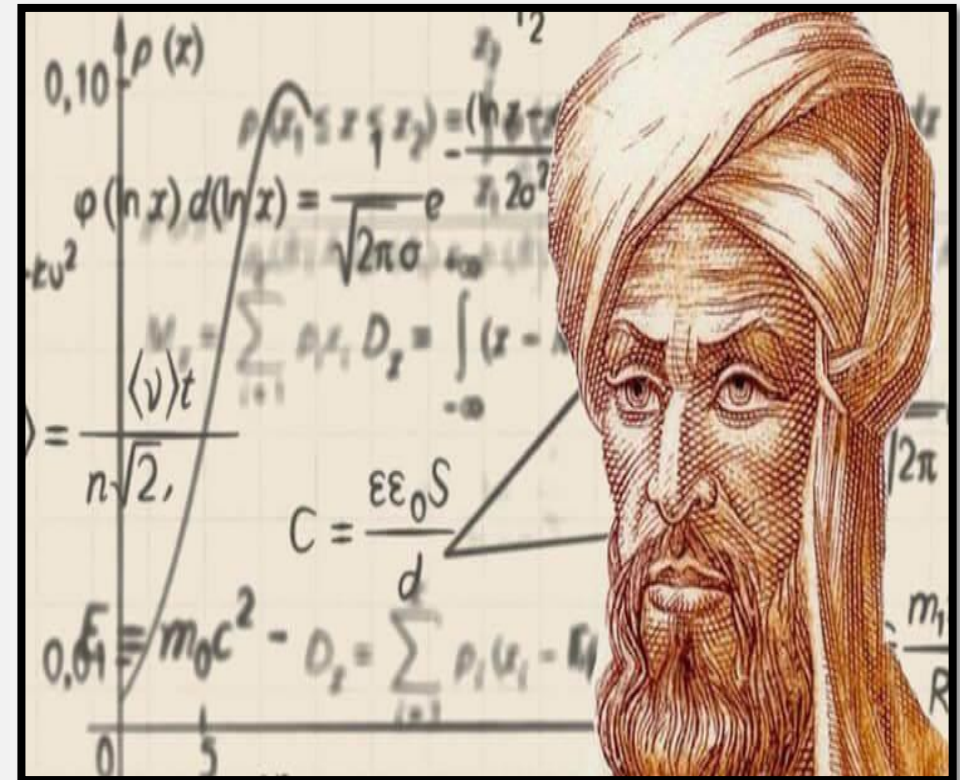
- Algorithm Phases.
- The Quality of a good Algorithm.
- Operations in an Algorithm.
- Algorithm design strategies.
- Representing Algorithms.
 - ✓ Natural Language
 - ✓ Formal Programming Language
 - ✓ Pseudocode
 - ✓ Flowcharts

THINGS YOU WILL GET FROM THIS COURSE

- Learn about a number of known algorithms for solving a wide range of problems.
- Learn about design techniques for designing your own algorithm.
- Become better at analyzing the running time of algorithms.
- Prepare for technical interviews!

INVENTOR OF ALGORITHMS

- The word Algorithm comes from the name of the Persian author, "Abu Ja'afar Mohammed Ibn Mousa Al Khawarizmi (825 A.D)".
- Al Khwarizmi wrote several books in Mathematics and Algebra.
- Linear and quadratic equations: Some of the first algorithms.



DEFINITION OF AN ALGORITHM

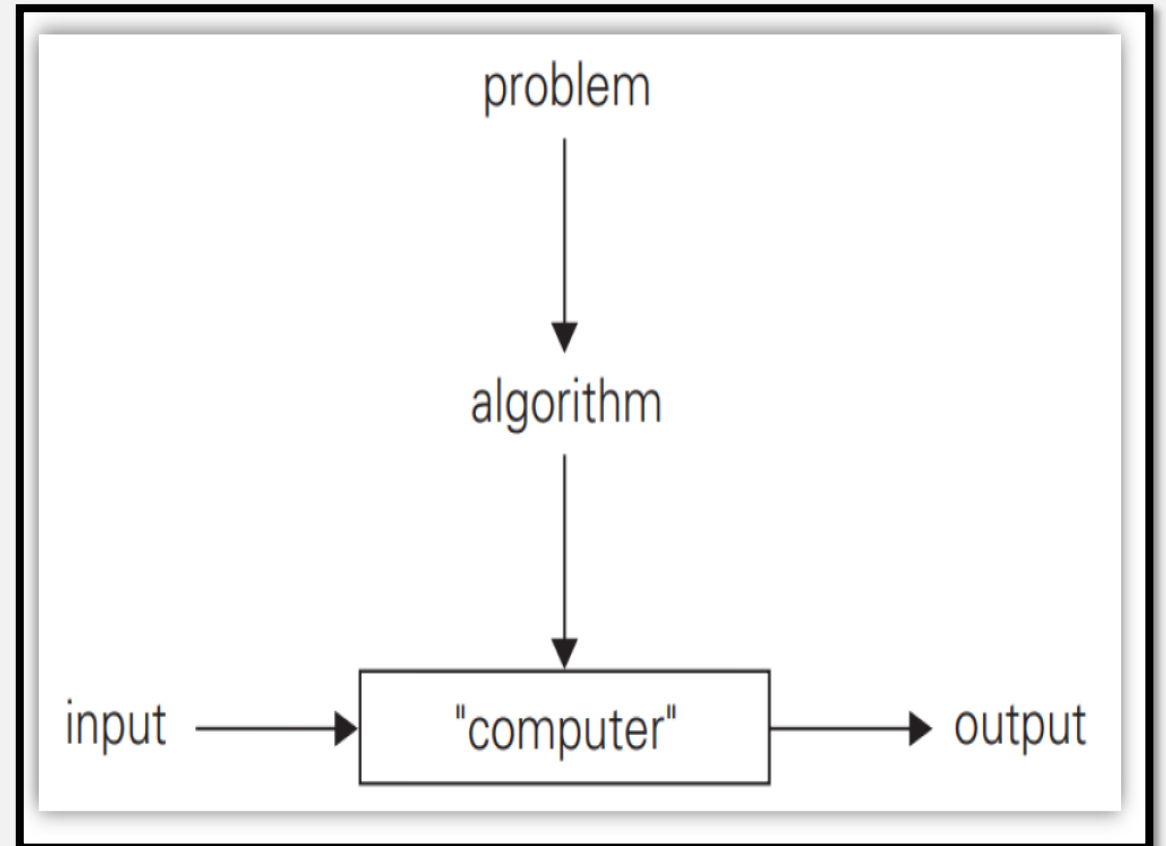
- The word Algorithm means "a process or set of rules to be followed in calculations or other problem-solving operations". Therefore, Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed in order to get the expected results.
- An Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

DEFINITION OF AN ALGORITHM

- An algorithm is a well-ordered collection of unambiguous and effectively computable operations that when executed produce a result and halts in a finite amount of time.

Note:

Algorithms are generally created independent of underlying languages, i.e., an algorithm can be implemented in more than one programming language.



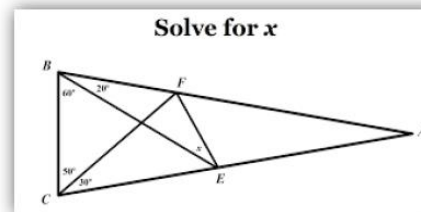
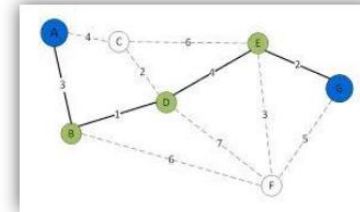
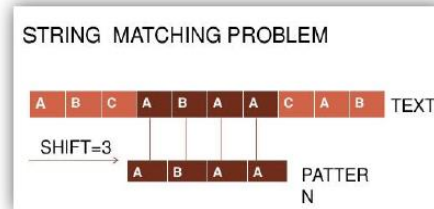
IMPORTANT PROBLEM TYPES

- Sorting
- Searching
- String processing
- Graph problems
- Geometric problems
- Numerical problem



Search 20.

12	5	10	15	31	20	25	2	40
0	1	2	3	4	5	6	7	8



A car is moving on a straight road at 5 m/s. It is accelerated at 3 m/s. What will be its velocity after 4 seconds?

EXAMPLE OF COMPUTATIONAL PROBLEM: SORTING

- **Statement of problem:**
- **Input:** A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$
- **Output:** A reordering of the input sequence $\langle a_1, a_2, \dots, a_n \rangle$ so that $a_i \leq a_j$ whenever $i < j$
- **Instance:** The sequence (input) $\rightarrow \langle 5, 3, 2, 8, 3 \rangle$, output $\rightarrow \langle 2, 3, 3, 5, 8 \rangle$.
- **Algorithms:**
- Selection sort
- Insertion sort
- Merge sort
- (many others)

NO SINGLE ALGORITHM FITS ALL SITUATIONS BEST!

- <Sorting>
- Some of the algorithms are simple but relatively slow, while others are faster but more complex
- Some work better on randomly ordered inputs, while others do better on almost-sorted lists
- Some are suitable only for lists residing in the fast memory, while others can be adapted for sorting large files stored on a disk

NO SINGLE ALGORITHM FITS ALL SITUATIONS BEST!

- <Searching>
- Some algorithms work faster than others but require more memory.
- Some are very fast but applicable only to sorted arrays.
- Organizing very large data sets for efficient searching poses special.
- Challenges with important implications for real-world applications.

WHY WE STUDY ALGORITHMS?

- Easier to solve and code the problem
 - ✓ The programming process is a complicated one. You must first understand the program specifications, of course, and then you need to organize your thoughts and create the program. This is a difficult task when the program is not trivial (i.e., easy).
- The program may be inefficient!
- If the program is run on a large data set, then the running time becomes an issue.

WHY WE STUDY ALGORITHMS?

- Writing an algorithm will save you time later during the construction & testing phase of a program's development.
- To be familiar with different strategies and approaches for solving different problems.
- To Find out which one of the known algorithms solves the problem in an efficient manner.

CHARACTERISTICS OF AN ALGORITHM

- **Correctness**, for every input instance, it halts with the correct output.
- **Clear** and Unambiguous - Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- The algorithm must be **general**, that is, it should solve the problem for all possible input sets.

CHARACTERISTICS OF AN ALGORITHM

- **Input** - An algorithm should have 0 or more well-defined inputs.
- **Output** - An algorithm should have 1 or more well-defined outputs and should match the desired output.
- **Efficient** – run as quickly as possible, and use a little memory as possible.

CHARACTERISTICS OF AN ALGORITHM

- Algorithms are **well-ordered**.
- **Finiteness** - Algorithms must terminate after a finite number of steps.
- It halts in a finite amount of time.
- **Feasibility** – Should be feasible with the available resources. Algorithms have effectively computable operations.

THE THINGS ARE NEEDED AS A PRE- REQUISITE TO WRITE AN ALGORITHM

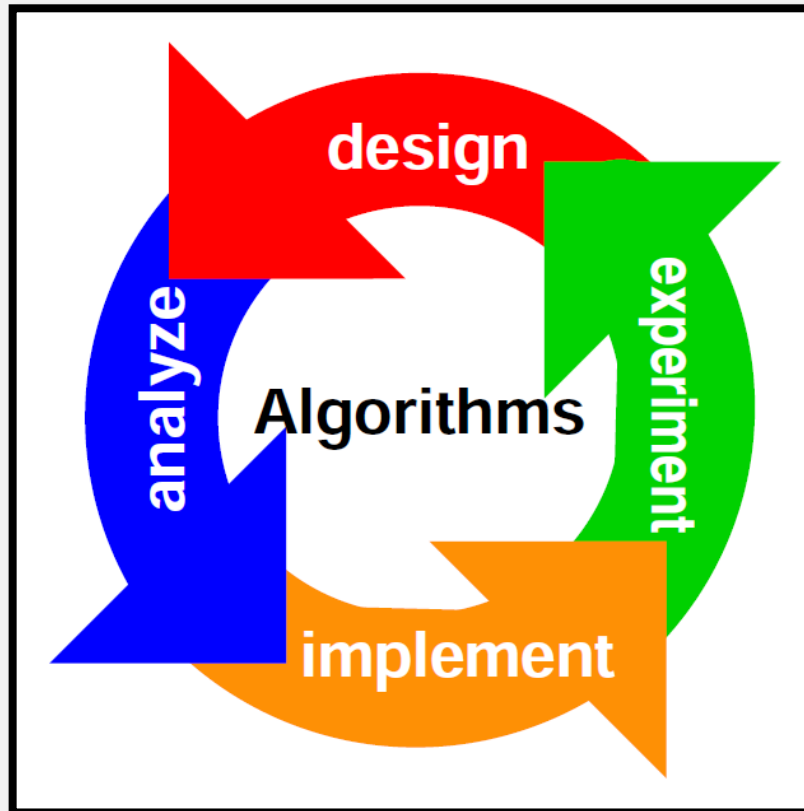
- The **constraints of the problem** that must be considered while solving the problem.
- The **input** to be taken to solve the problem.
- The **output** to be expected when the problem is solved.
- The **solution** to this problem, in the given constraints.
- Then the **algorithm** is written with the help of above parameters such that it solves the problem.

STEPS FOR DEVELOPING AN ALGORITHM

1. **Define the problem:** State the problem you are trying to solve in clear and concise terms.
2. List the **inputs** (information needed to solve the problem) and the **outputs** (what the algorithm will produce as a result)
3. Describe the **steps** needed to convert or manipulate the inputs to produce the outputs. Start at a high level first and keep refining the steps until they are effectively computable operations. (Write an algorithm).
4. **Test the algorithm:** choose data sets and verify that your algorithm works!

ALGORITHM PHASES

- Design
- Analyze
- Implement
- Experiment



ALGORITHM PHASES

- **Design Phase**

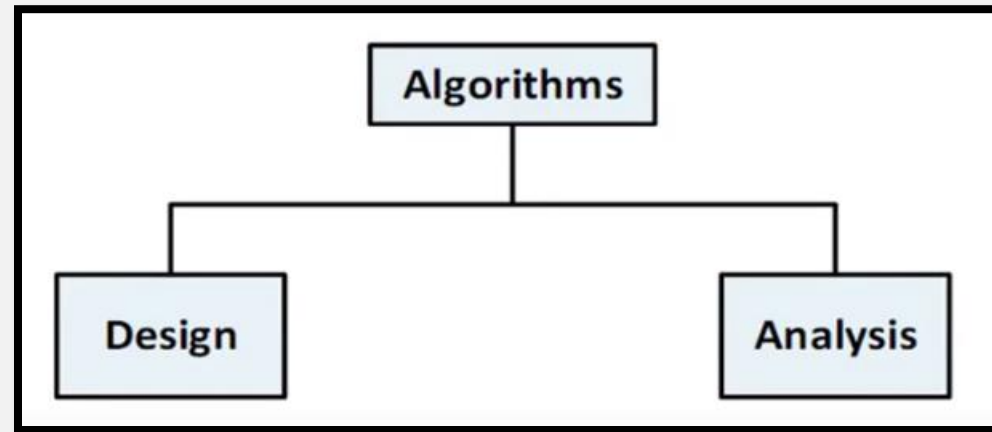
- ✓ The first stage is to identify the problem and fully understand it.
- ✓ Consultation with people interested in similar problems.

- **Analysis Phase**

- ✓ Analysis of an Algorithm is the theoretical study of computer program performance and resource usage.
- ✓ Many solution algorithms can be derived for a given problem.
- ✓ The next step is to analyze those proposed solution algorithms and choose the best suitable solution.

ALGORITHM PHASES

- **Design** → Methods and techniques that yield good and useful algorithms to solve the problem.
- **Analyze** → Mathematical Comparison of algorithms without actually implementing it.



ALGORITHM PHASES

- **Implement**
 - ✓ Writing and coding the algorithm.
- **Experiment (Test)**
 - ✓ Test with different cases and variables.

THE QUALITY OF A GOOD ALGORITHM

- Time
- Memory
- Accuracy
- Sequence
- Generality

OPERATIONS IN AN ALGORITHM

- **Sequential operations**

- ✓ A sequential operation carries out a single well-defined task. When that task is finished, the algorithm moves on to the next operation.

- **Conditional operations**

- ✓ A conditional operation is the "question-asking" instructions of an algorithm. It asks a question and then selects the next operation to be executed according to the question-answer.

- **Iterative operations**

- ✓ An Iterative operation is a “looping” instruction of an algorithm. It tells us not to go on to the next instruction, but, instead, to go back and repeat the execution of a previous block of instructions.

ALGORITHM DESIGN STRATEGIES

- Brute force
- Greedy approach
- Divide and conquer
- Decrease and conquer
- Transform and conquer
- Dynamic programming
- Backtracking and branch-and-bound

REPRESENTING ALGORITHMS

- How to Express Algorithms in a Clear, Precise, and Unambiguous manner?
 - ✓ In terms of Natural Language
 - ✓ In terms of Formal Programming Language
 - ✓ In terms of Pseudocode
 - ✓ Flowcharts

REPRESENTING ALGORITHMS

- **(1) In Term of Natural Language**
 - ✓ Assume the first item is largest. Look at each of the remaining items in the list and if it is larger than the largest item so far, make a note of it. The last noted item is the largest in the list when the process is complete.
- **Advantages**
 - ✓ Familiar
- **Disadvantages**
 - ✓ Verbose
 - ✓ Imprecise → Ambiguity
 - ✓ Rarely used for complex or technical algorithms

REPRESENTING ALGORITHMS

- **(2) In Term of Formal Programming Language**
- **Advantages**
 - ✓ Precise → Unambiguous
 - ✓ Will ultimately program in these languages
- **Disadvantages**
 - ✓ Can be too low-level for algorithm design.
 - ✓ It has syntactic details which are not important at the algorithm design phase.
 - ✓ Not familiar for the person who is not interested in this programming language.

```
#Sum of natural numbers up to num
num = int(input("enter a number"))
if num < 0:
    print("Enter a positive number")
else:
    sum = 0
    while(num > 0):
        sum += num
        num -= 1
    print("The sum is", sum)
```

REPRESENTING ALGORITHMS

- **(3) In Term of Pseudocode**
- **Advantages**
 - ✓ A middle-ground compromise.
 - ✓ Resembles many popular programming languages.
 - ✓ Relatively free of grammatical rules.
 - ✓ Only well-defined statements are included "A Programming language without details".
- **Disadvantages**
 - ✓ Compared with flowchart it is difficult to understand the program logic.

```
1. Get array list and its size n
2. Assign max = list[1]
3. For i = 2 to n Do
    1. IF (list[i] > max) THEN
        1. max = list[i]
    2. End If
4. End For
5. Display max
```

WHAT IS A PSEUDOCODE?

- Pseudocode (derived from pseudo and code) is a compact and informal high-level description of a computer programming algorithm that uses the structural conventions of some programming language, but typically omits details that are not essential for the understanding of the algorithm, such as subroutines, variable declarations and system-specific code.
- **SOME NOTES**
 - ✓ No standard pseudo code syntax exists.
 - ✓ A program in pseudocode is not an executable program.
 - ✓ Flowcharts can be thought of as a graphical alternative to pseudocode.

PSEUDOCODE

- Consists of natural language-like statements that precisely describe the steps of an algorithm or program.
- Statements describe actions.
- Focuses on the logic of the algorithm or program.
- Avoids language-specific elements.
- Written at a level so that the desired programming code can be generated almost automatically from each statement.
- Steps are numbered. Subordinate numbers and/or indentation are used for dependent statements in selection and repetition structures.

PSEUDOCODE LANGUAGE CONSTRUCTS

- **Computation/Assignment**
 - ✓ **Compute** var1 as the sum of x and y
 - ✓ **Assign** expression to var2
 - ✓ **Increment** counter1
- **Input/Output**
 - ✓ **Input: Get** var1, var2, ...
 - ✓ **Output: Display** var1, var2, ...

PSEUDOCODE LANGUAGE CONSTRUCTS

- **Selection**
- **Single-Selection IF**
 - ✓ **IF** condition **THEN** (IF condition is true, then do subordinate statement 1, etc. If condition is false, then skip statements)
 - ✓ statement 1
 - ✓ etc.
- **Double-Selection IF**
 - ✓ **IF** condition **THEN** (IF condition is true, then do subordinate statement 1, etc. If condition is false, then skip statements and execute statements under **ELSE**)
 - ✓ statement 1
 - ✓ etc.

PSEUDOCODE LANGUAGE CONSTRUCTS

- ✓ **ELSE** (else if condition is not true, then do subordinate statement 2, etc.)
- ✓ statement 2
- ✓ statement 3
- **SWITCH expression TO**
 - ✓ case 1: action1
 - ✓ case 2: action2
 - ✓ etc.
 - ✓ default: action

PSEUDOCODE LANGUAGE CONSTRUCTS

- **Repetition**
- **WHILE condition** (while condition is true, then do subordinate statements)
 - ✓ statement 1
 - ✓ etc.
- **FOR condition** (a specialized version of WHILE for repeating execution of statements a specific number of times)
 - ✓ statement 1
 - ✓ etc.

PSEUDOCODE LANGUAGE CONSTRUCTS

- **DO- WHILE structure** (like WHILE, but tests condition at the end of the loop. Thus, statements in the structure will always be executed at least once.)
 - ✓ DO
 - ✓ statement 1
 - ✓ etc.
 - ✓ WHILE condition
 - ✓ statement 1
 - ✓ etc.

PSEUDOCODE EXAMPLE







- Express an algorithm to get two numbers from the user (dividend and divisor), testing to make sure that the divisor number is not zero, and displaying their quotient using pseudocode.
- Solution:

1. Get dividend, divisor
2. IF divisor = 0 THEN
 - 1) Display error message "divisor must be non-zero"
 - 2) Exit algorithm
3. Compute quotient = dividend/divisor
4. Display dividend, divisor, quotient

REPRESENTING ALGORITHMS

- **(4) Flowcharts**

A flowchart is a graphical representation of an algorithm. Once the flowchart is drawn, it becomes easy to write the program in any high-level language.

Symbol	Name/Meaning	Symbol	Meaning
	<u>Process</u> – Any type of internal operation: data transformation, data movement, logic operation, etc.		<u>Connector</u> – connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow
	<u>Input/Output</u> – input or output of data		<u>Terminal</u> – indicates start or end of the program or algorithm
	<u>Decision</u> – evaluates a condition or statement and branches depending on whether the evaluation is true or false		<u>Flow lines</u> – <i>arrows</i> that indicate the direction of the progression of the program

OTHER FLOWCHART SYMBOLS



Terminator

Indicates the beginning or end of a program flow in your diagram.



Process

Indicates any processing function.



Decision

Indicates a decision point between two or more paths in a flowchart.



Delay

Indicates a delay in the process.



Data

Can represents any type of data in a flowchart.



Document

Indicates data that can be read by people, such as printed output.



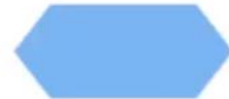
Multiple documents

Indicates multiple documents.



Subroutine

Indicates a predefined (named) process, such as a subroutine or a module.



Preparation

Indicates a modification to a process, such as setting a switch or initializing a routine.



Display

Indicates data that is displayed for people to read, such as data on a monitor or projector screen.



Manual input

Indicates any operation that is performed manually (by a person).



Manual loop

Indicates a sequence of commands that will continue to repeat until stopped manually.



Loop limit

Indicates the start of a loop. Flip the shape vertically to indicate the end of a loop.



Stored data

Indicates any type of stored data.



Connector

Indicates an inspection point.



Off-page connector

Use this shape to create a cross-reference and hyperlink from a process on one page to a process on another page.



Off-page connector



Off-page connector



Off-page connector



Or

Logical OR



Summing junction

Logical AND



Collate

Indicates a step that organizes data into a standard format.



Sort

Indicates a step that organizes items list sequentially.



Merge

Indicates a step that combines multiple sets into one.



Database

Indicates a list of information with a standard structure that allows for searching and sorting.



Internal storage

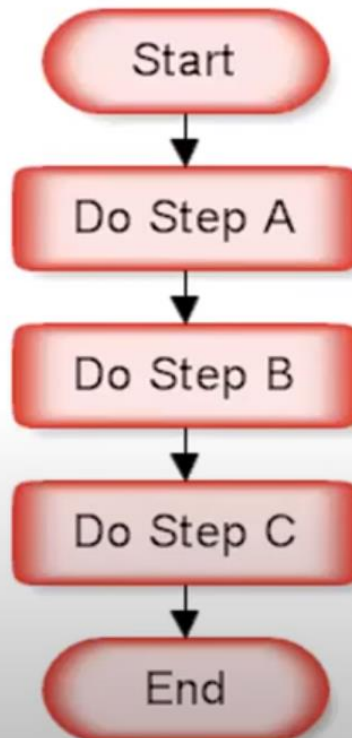
Indicates an internal storage device.

FLOWCHARTS

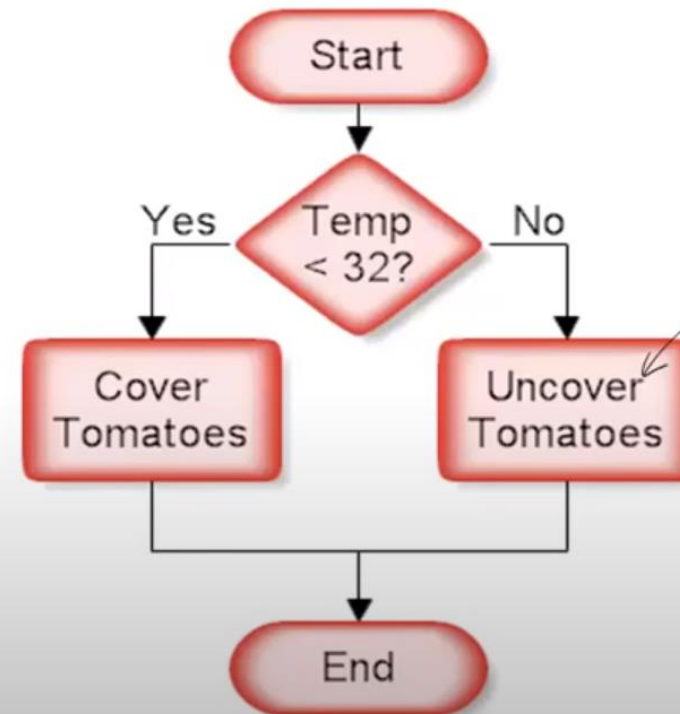
- **Benefits of Flowcharts**
 - ✓ Makes Logic Clear Effective Analysis
 - ✓ Easy to Code and Test
- **Limits of Flowcharts**
 - ✓ Difficult to use for large Programs.
 - ✓ Difficult to Modify.

FLOWCHART CONSTRUCTS

Sequence



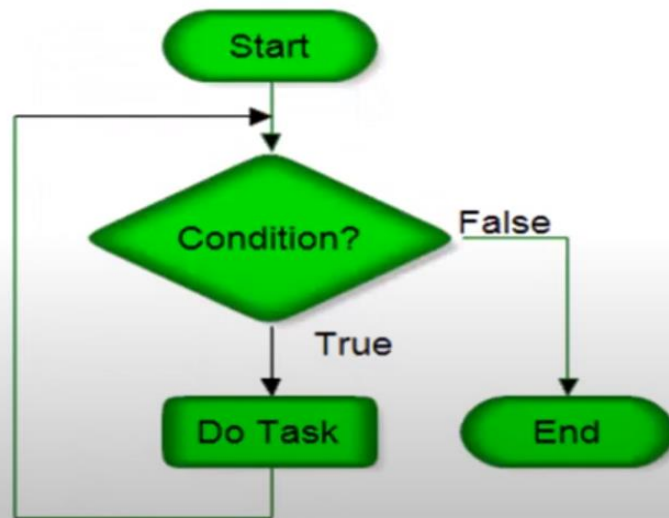
Decision



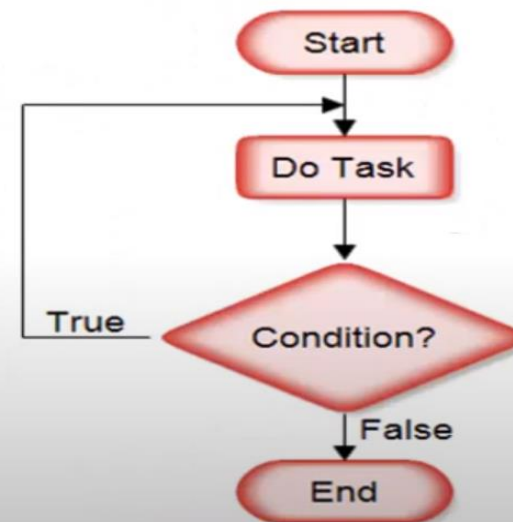
FLOWCHART CONSTRUCTS

Loop

While Loop

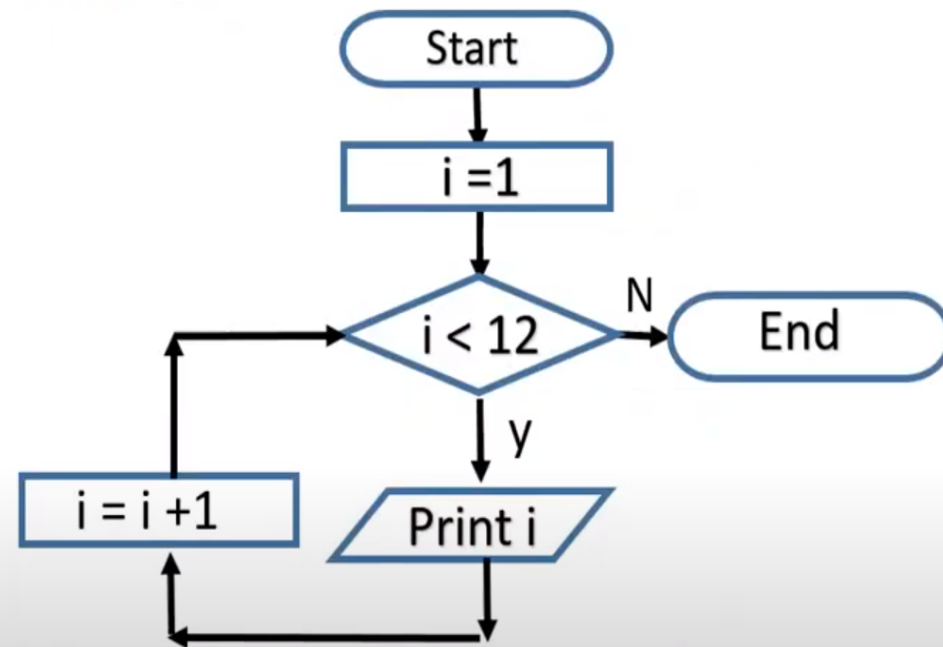


Do While Loop



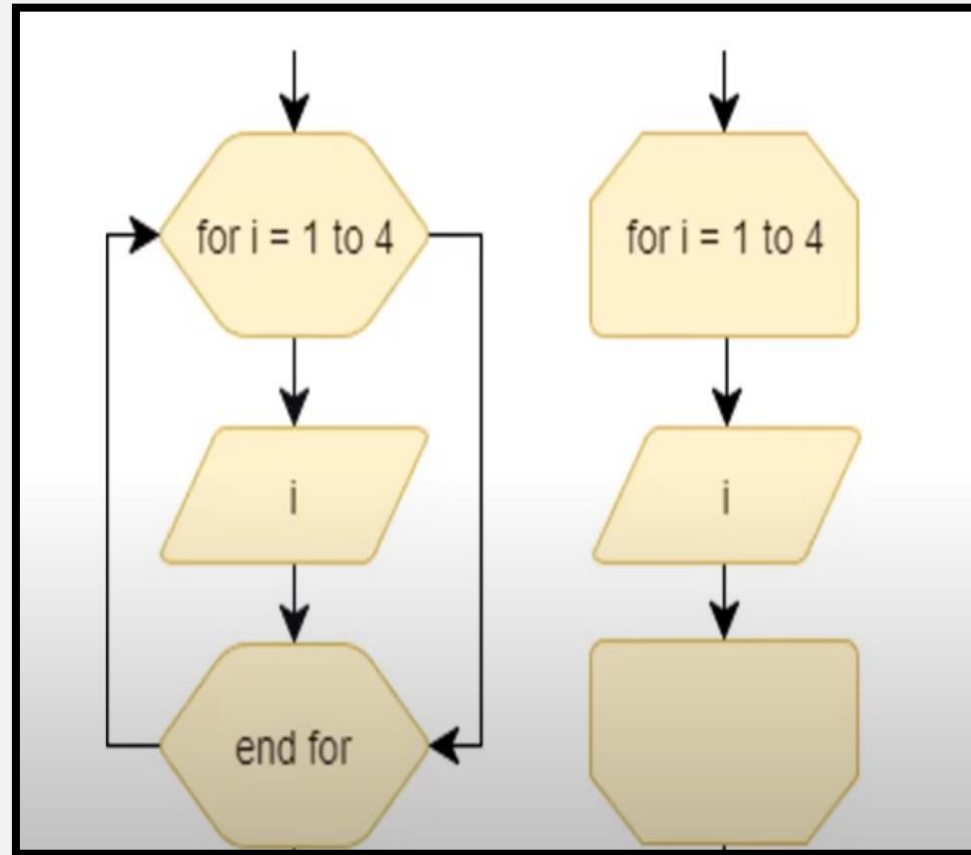
FLOWCHART CONSTRUCTS

for i in range (1,12):
print(i)



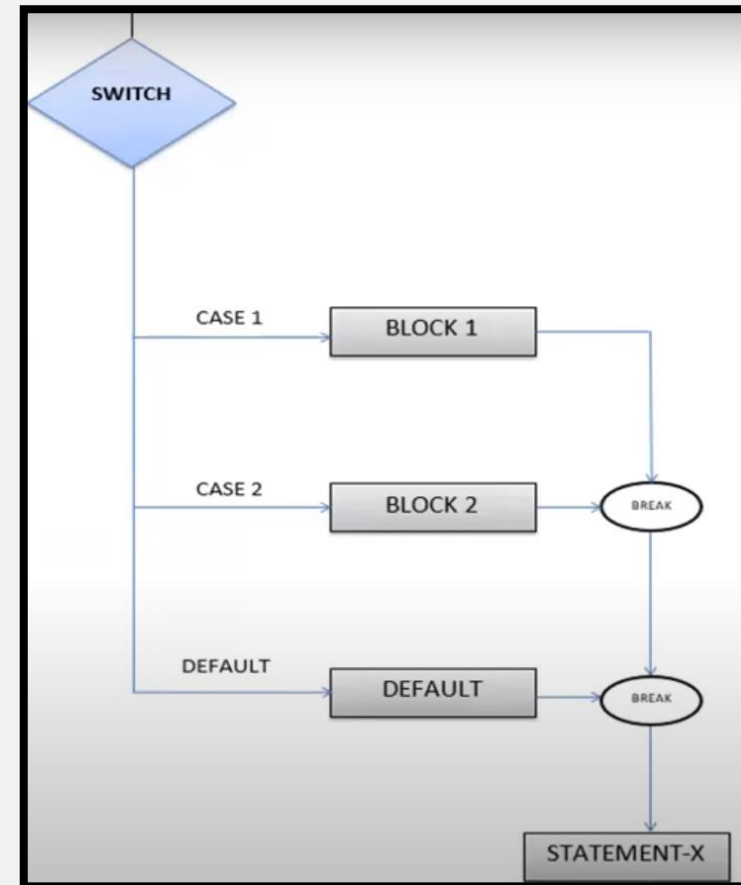
FLOWCHART CONSTRUCTS

- Loops are often drawn by the decision symbol. In some diagrams, you can encounter a notation using the preparation symbol or the loop limit symbol.



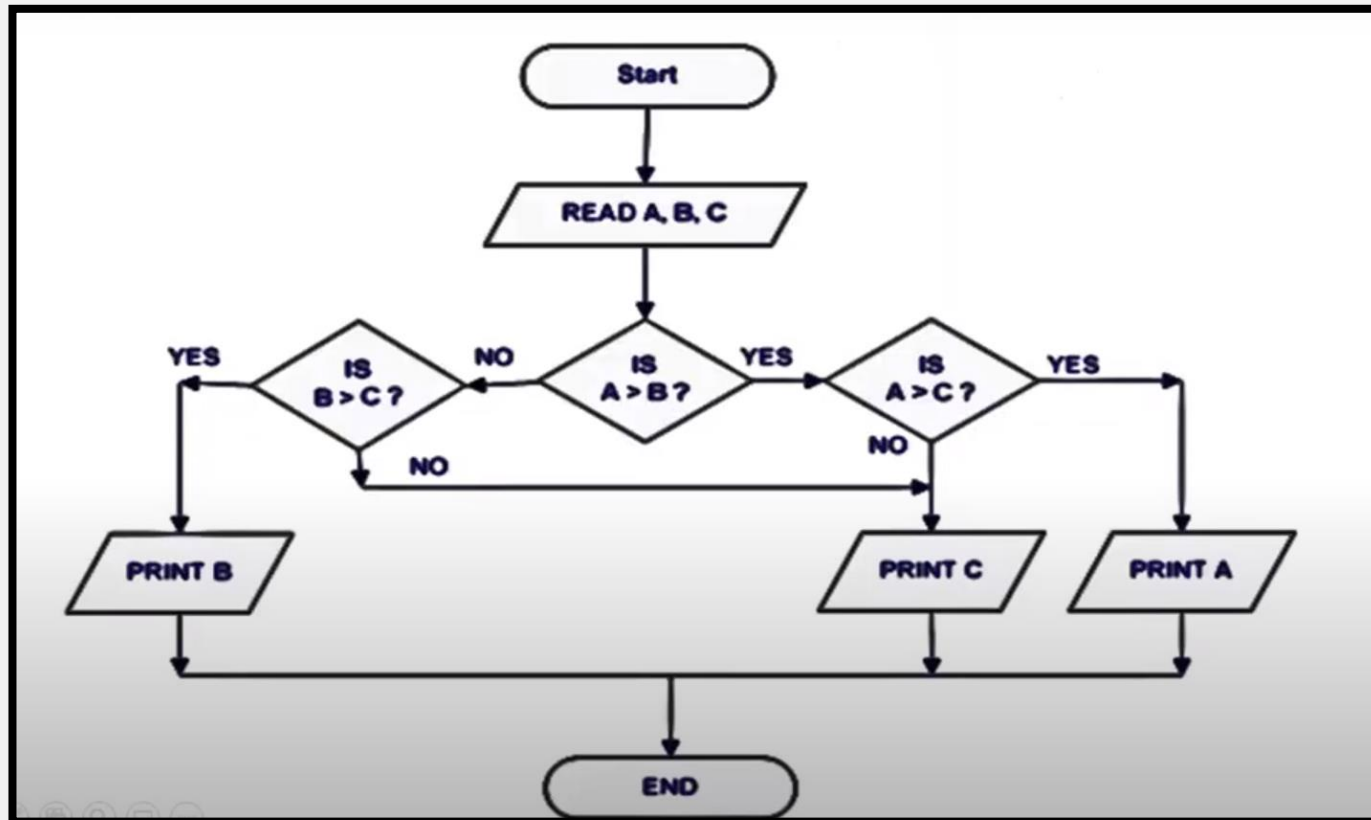
FLOWCHART CONSTRUCTS

- Switch Case tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that case is executed. Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found.



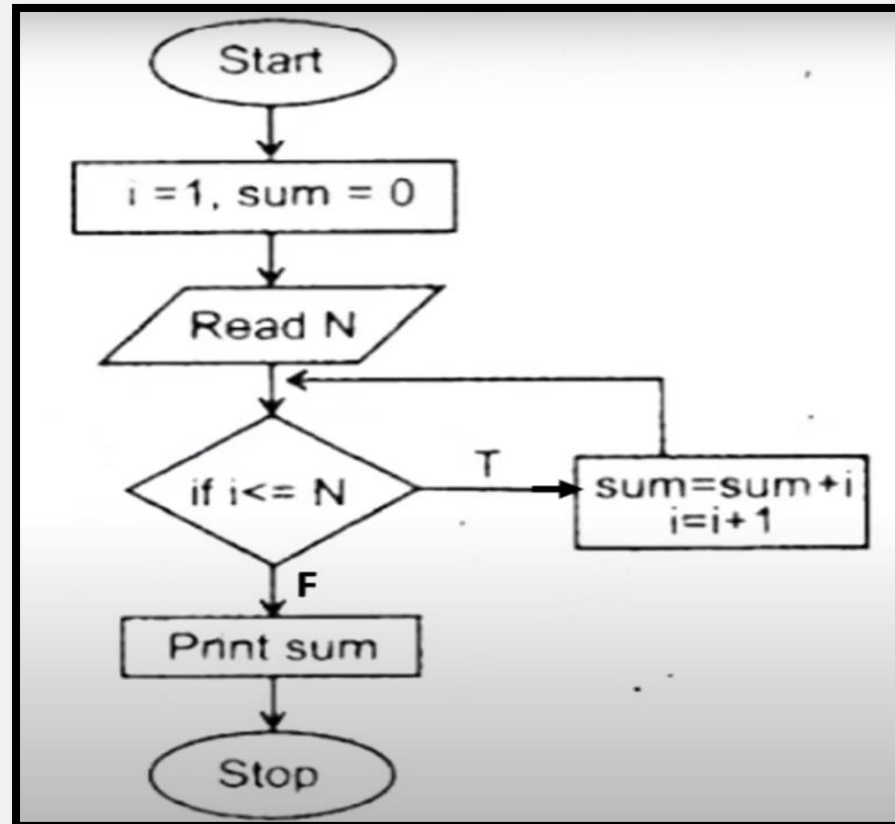
FLOWCHART EXAMPLES

- **Example1:** Draw a flowchart to find the largest of three numbers A,B, and C



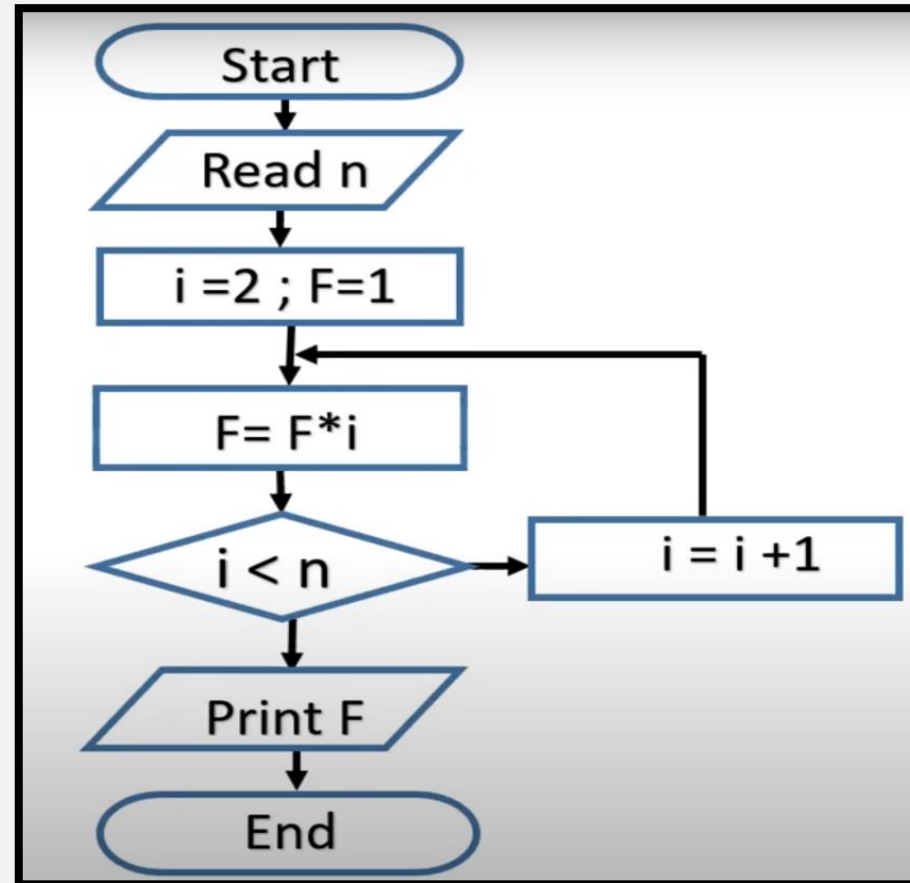
FLOWCHART EXAMPLES

- **Example2:** Sum of N numbers



FLOWCHART EXAMPLES

- **Example3:** Flowchart to find $n!$



END OF CHAPTER1