Algorithm Specification

Dr. Bibhudatta Sahoo

Communication & Computing Group

Department of CSE, NIT Rourkela

Email: bdsahu@nitrkl.ac.in, 9337938766, 2462358

Algorithm

- Algorithm: is a procedure that consists of a *finite set of instructions* which, given an *input* from some set of possible inputs, enables us to obtain an *output* if such an output exists or else obtain nothing at all if there is no output for that particular input through a *systematic execution* of the *instructions*.
- **Algorithm:** It's an organized logical sequence of the actions or the approach towards a particular problem.
- A programmer implements an algorithm to solve a problem.
- Algorithms are expressed using natural verbal but somewhat technical annotations.

Algorithm definition

An algorithm is a finite set of instructions that, is followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

1. **Input**: Zero or more quantities
2. **Output**: At least one
3. **Definiteness** : Each instruction is clear and unambiguous
4. **Finiteness**: Algorithm terminates in finite number of steps
5. **Effectiveness:** Every instruction must be very basic and feasible.

Algorithm Description

How to describe algorithms independent of a programming language

- **Text (Pseudo-Code)** = a description of an algorithm that is
  - ❑ more structured than usual prose but
  - ❑ less formal than a programming language

- **Diagrams** e.g., flowchart, sequence diagram (good for complex interactions among objects) Communicate visually, however, time-consuming to create, hard to maintain, separate from source file.

Example: find the maximum element of an array.

```
Algorithm arrayMax(A, n):
    Input: An array A storing n integers.
    Output: The maximum element in A.
    currentMax ← A[0]
    for i ← 1 to n -1 do
        if currentMax < A[i] then currentMax ← A[i]
    return currentMax
```

What is Pseudocode?

- **Pseudocode** (pronounced SOO-doh-kohd) is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language.
- **Pseudocode** is sometimes **used** as a detailed step in the process of developing a program
- **Pseudocode** is a plain-text description of a piece of code or an algorithm. It's not actually coding; there is no script, no files, and no programming. As the name suggests, it's "fake code".
- **Pseudocode** is not written in **any particular programming language**. It's written in plain English that is clear and easy to understand.
- Writing a full program in pseudocode requires a lot of different statements and keywords much like regular programming. In fact, once you get far enough along in your pseudocode it will start to look very close to a real program.

Pseudo code

- **Pseudo code:** It's simply an implementation of an algorithm in the form of annotations and informative text written in plain English.
- It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer.

Advantages of Pseudocode

- Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.
- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.
- The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

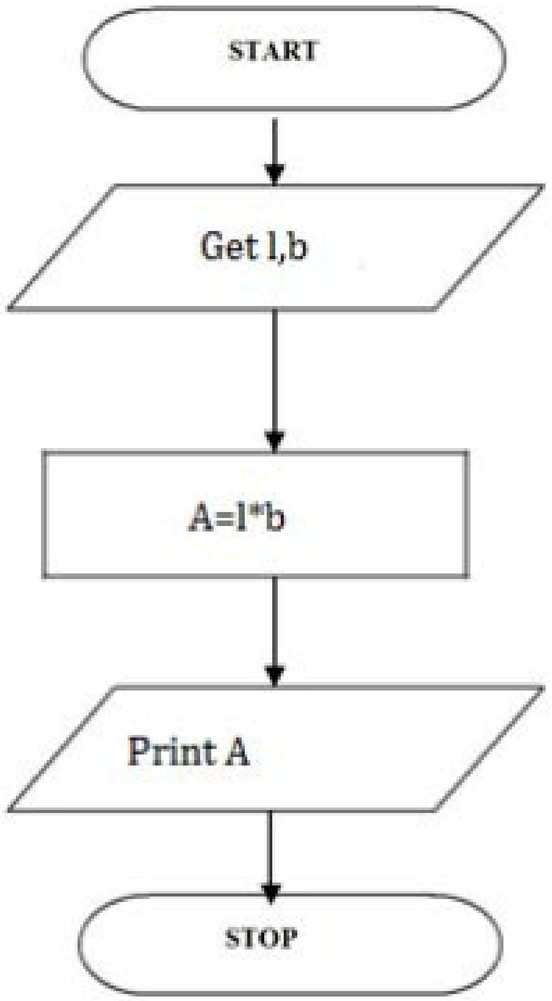
Difference between Algorithm, Pseudocode and Program

- **Algorithm : Systematic logical approach** which is a well-defined, step-by-step procedure that allows a computer to solve a problem.

- **Pseudocode :** It is a **simpler version of a programming code** in plain English which uses short phrases to write code for a program before it is implemented in a specific programming language.

- **Program :** It is exact code written for problem **following all the rules of the programming language.**

Write an algorithm to find area of a rectangle?



Step 1: Start

Step 2: get l, b values

Step 3: Calculate A=l*b

Step 4: Display A

Step 5: Stop

To check greatest of three numbers

Step1: Start

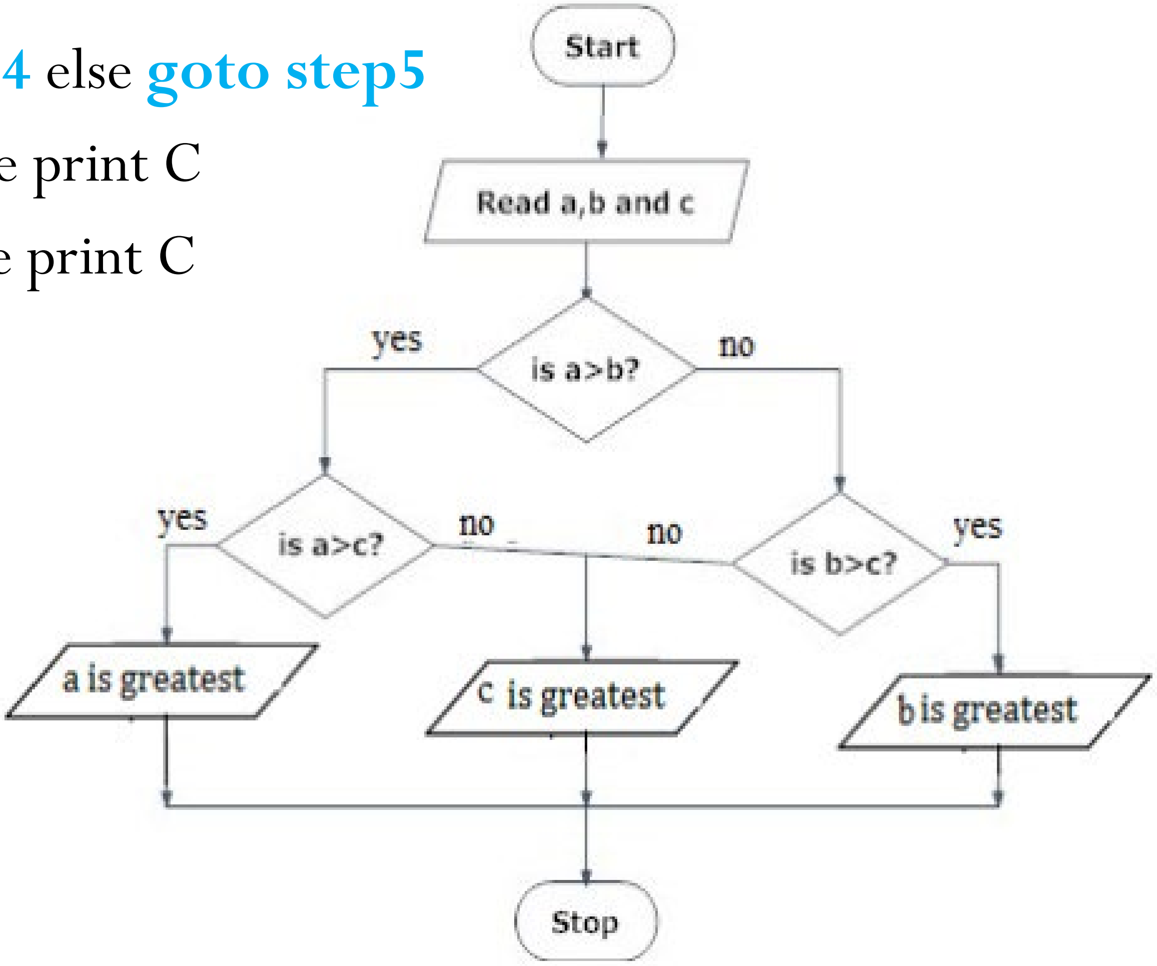
Step2: Get A, B, C

Step3: if(A>B) **goto Step4** else **goto step5**

Step4: If(A>C) print A else print C

Step5: If(B>C) print B else print C

Step6: Stop



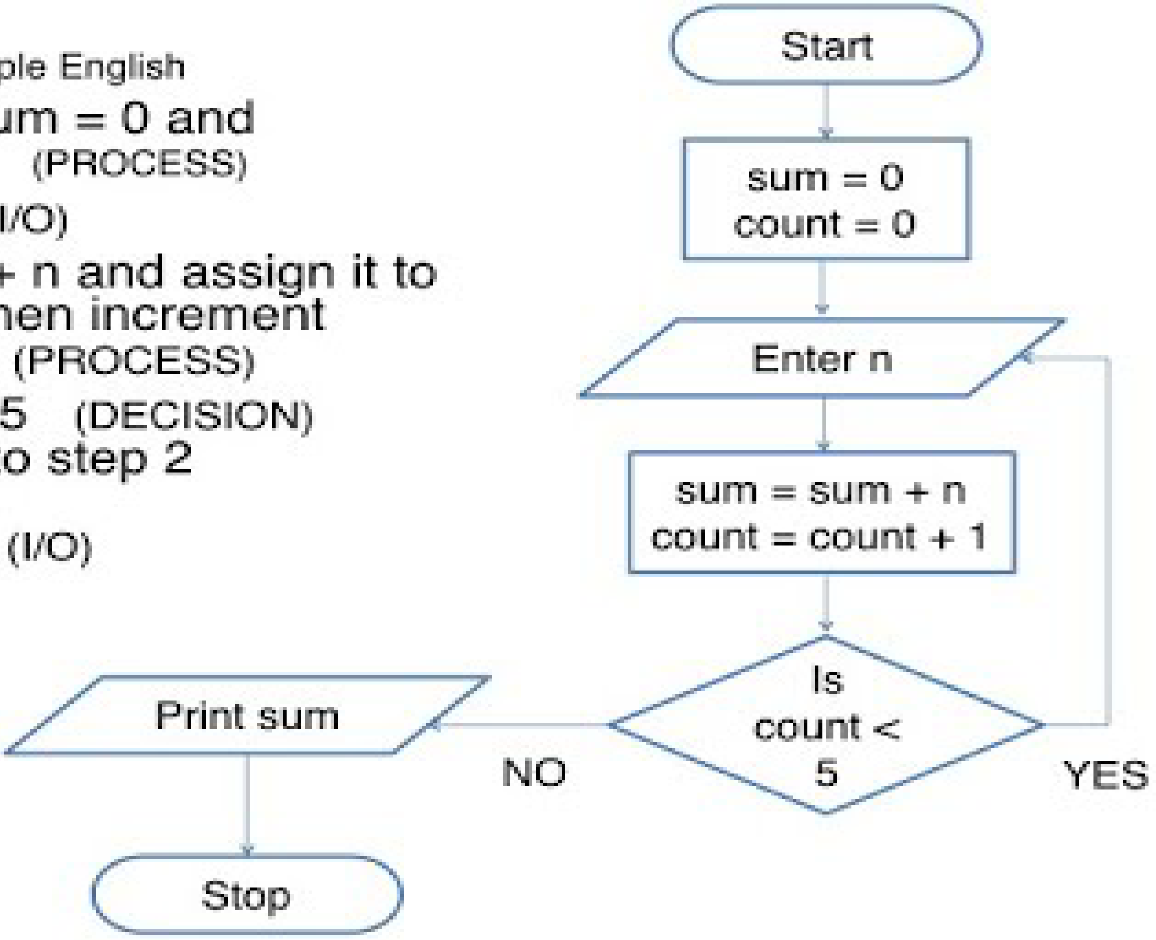
Example: Algorithm & Flowchart

Find the sum of 5 numbers

Algorithm in simple English

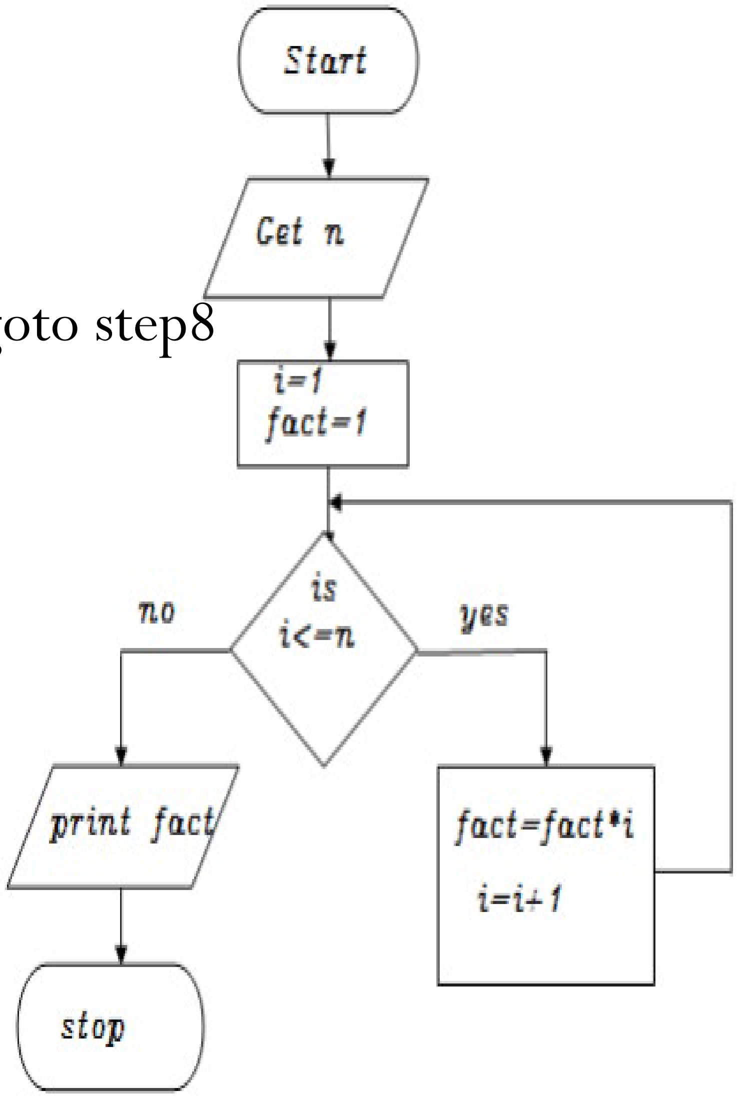
1. Initialize sum = 0 and count = 0 (PROCESS)
2. Enter n (I/O)
3. Find sum + n and assign it to sum and then increment count by 1 (PROCESS)
4. Is count < 5 (DECISION)
   if YES go to step 2
   else
   Print sum (I/O)

Flowchart



Write an algorithm to find factorial of a given number

```
Step 1: start
step 2: get n value
step 3: set initial value i=1, fact=1
Step 4: check i value if(i<=n) goto step 5 else goto step8
step 5: calculate fact=fact*i
step 6: increment i value by 1
step 7: goto step 4
step 8: print fact value
step 9: stop
```

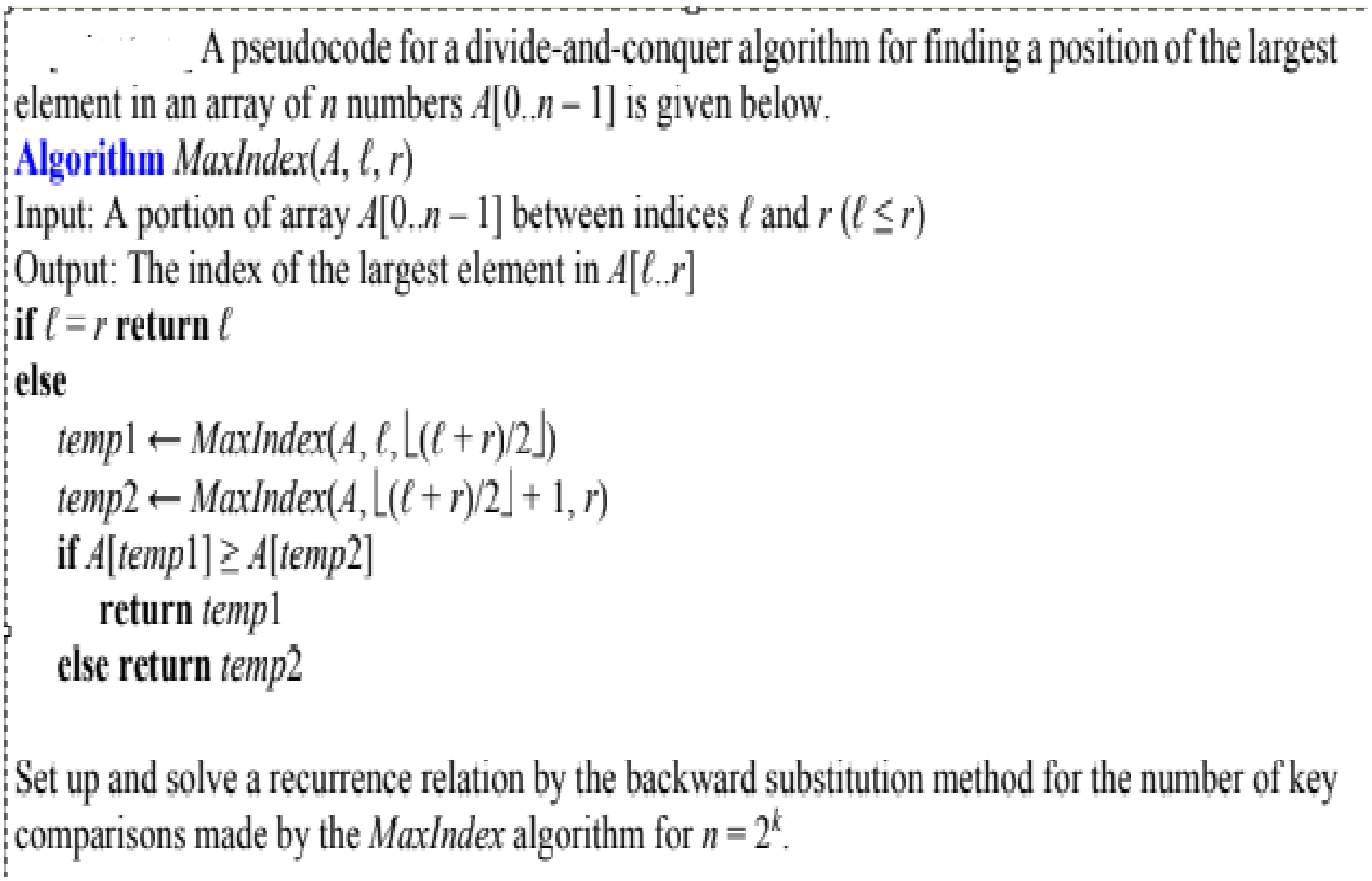


Algorithmic Notation

Pseudocode standard

- Pseudocode standard refers to a set of conventions or guidelines used to write pseudocode—a high-level description of a computer program or algorithm.
- Pseudocode isn't bound by the syntax of any specific programming language but uses plain English to describe steps in a way that's clear for humans.
- There is no single universal standard, pseudocode typically follows some common conventions to enhance readability and maintain consistency.
- The pseudocode needs to be complete. It describe the entire logic of the algorithm so that implementation becomes a rote mechanical task of translating line by line into source code.

Example: 1


A pseudocode for a divide-and-conquer algorithm for finding a position of the largest element in an array of n numbers $A[0..n-1]$ is given below.

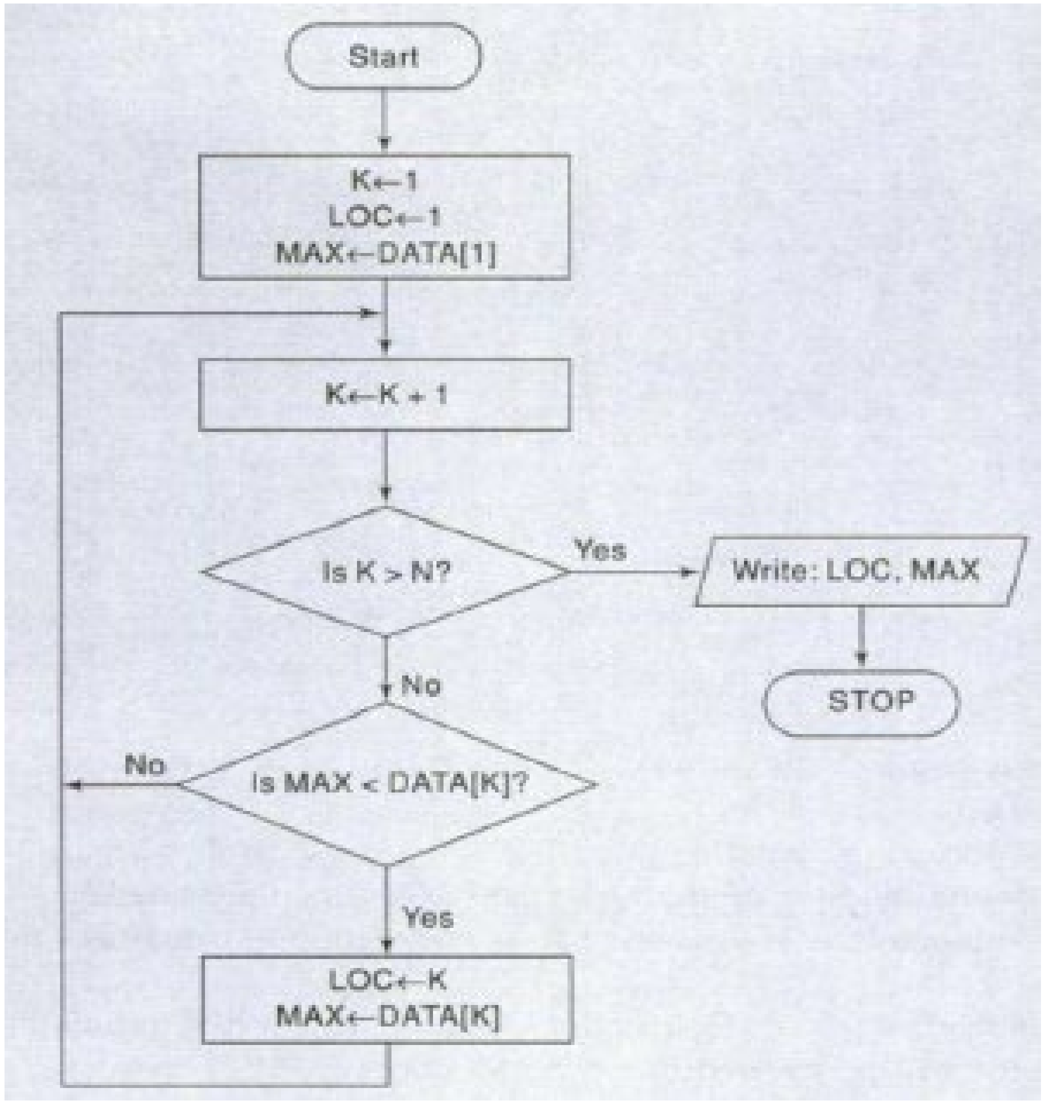
```
Algorithm MaxIndex(A, ℓ, r)
Input: A portion of array A[0..n − 1] between indices ℓ and r (ℓ ≤ r)
Output: The index of the largest element in A[ℓ..r]
if ℓ = r return ℓ
else
    temp1 ← MaxIndex(A, ℓ, ⌊(ℓ + r)/2⌋)
    temp2 ← MaxIndex(A, ⌊(ℓ + r)/2⌋ + 1, r)
    if A[temp1] ≥ A[temp2]
        return temp1
    else return temp2
```

Set up and solve a recurrence relation by the backward substitution method for the number of key comparisons made by the *MaxIndex* algorithm for $n = 2^k$.

Example:

```
Algorithm SUM_VALUES (A, N)
This algorithm computes sum of N elements of a given array A
1. [ Initialize SUM]
        SUM ← 0
2. Repeat for I = 1 to N
3. SUM ← SUM + A[I]
        [end of step 2 ]
4.[Output result computed]
        Write(SUM)
5. [Finished]
        Exit
```

Question: Write algorithm from given flow chart?



Identifying Algorithm

- Algorithm Maximum(A, N, MAX)

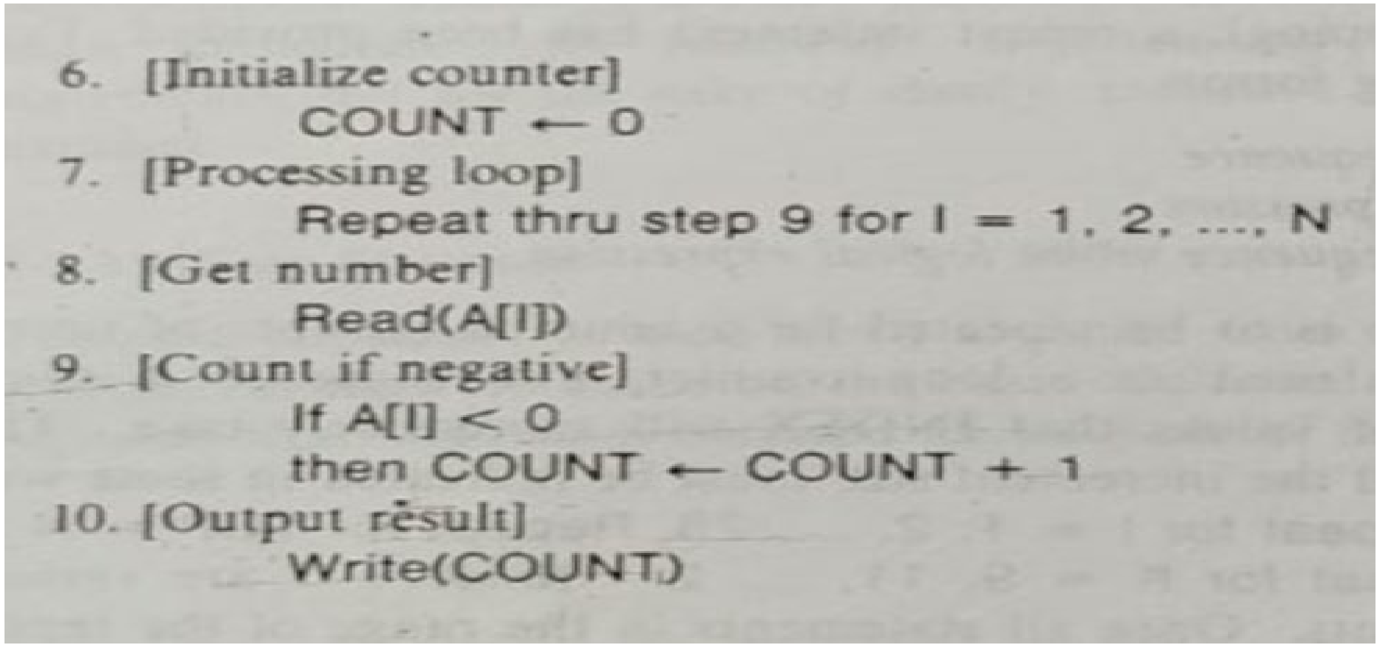
- Algorithm Maximum(A, N)

Comments

- In pseudocode, comments are used to explain what the code or algorithm is doing.
- Comments clarify complex steps, describe the purpose of certain parts of the pseudocode, and make the logic easier to understand for others who may read the pseudocode.
- Comments in pseudocode are usually written in plain English and do not affect the flow or execution of the code. They are purely informational and can be ignored by the actual implementation when coding in a real programming language.

Syntax for Comments in Pseudocode:

- There is no formal standard for writing comments in pseudocode, but common conventions include: Preceding the comment with // (similar to many programming languages).
- Enclosing the comment in parentheses ( ) or [ ].
- Writing comments on their own line or at the end of a line of pseudocode.



```
6.  [Initialize counter]
            COUNT ← 0
7.  [Processing loop]
            Repeat thru step 9 for I = 1, 2, ...., N
8.  [Get number]
            Read(A[I])
9.  [Count if negative]
            If A[I] < 0
            then COUNT ← COUNT + 1
10. [Output result]
            Write(COUNT)
```

Benefits of Using Comments in Pseudocode:

- **Clarification**: Comments help explain what specific lines or blocks of pseudocode are doing, especially if the logic is complex.
- **Documentation**: Comments provide insight into the programmer's thought process and help future readers understand the code's purpose without needing to decipher it line by line.
- **Readability**: Pseudocode with comments is easier to follow, especially for those who might not be familiar with the underlying logic or for those working in a team setting.

Best Practices for Comments in Pseudocode:

- **Be Concise**: Comments should be brief and to the point.
- **Explain the Why**: Focus on explaining why something is done rather than what is being done (since the pseudocode itself should already explain what is happening).
- **Avoid Redundancy**: Don't state the obvious. If the pseudocode is self-explanatory, you may not need comments.

Steps, Control & Exit

- **Steps**: Steps in pseudocode refer to the sequence of operations or instructions that need to be followed to complete a specific task within the algorithm. Each step is usually a clear, concise statement that describes a single operation, such as assigning a value to a variable, performing a calculation, or calling a function.
- **Control**: Control structures in pseudocode dictate the flow of execution of the algorithm. These include decision-making structures like `if`, `else if`, `else`, and loops like `for`, `while`, and `repeat-until`. Control structures help in branching (choosing between alternative paths) and iteration (repeating steps a certain number of times or until a condition is met).
- **Exit:** Exit in pseudocode refers to the point at which the algorithm stops executing. This could be because the algorithm has successfully completed its task and returns a result, or because a particular condition triggers an early termination (`**break**` in loops).
- An `Exit` might also occur if there are errors or exceptions that prevent further execution.

Example

```
6. [Initialize counter]
        COUNT ← 0
7. [Processing loop]
        Repeat through step 9 for I = 1,2,…N
8. [Get number]
        Read (A[I])
9. [Count if negative]
        If A[I] < 0
        then COUNT ← COUNT +1
10. [Output result]
        Write(COUNT)
5. [Finished]
        Exit
```

Example

Algorithm SUM_VALUES (A, N)

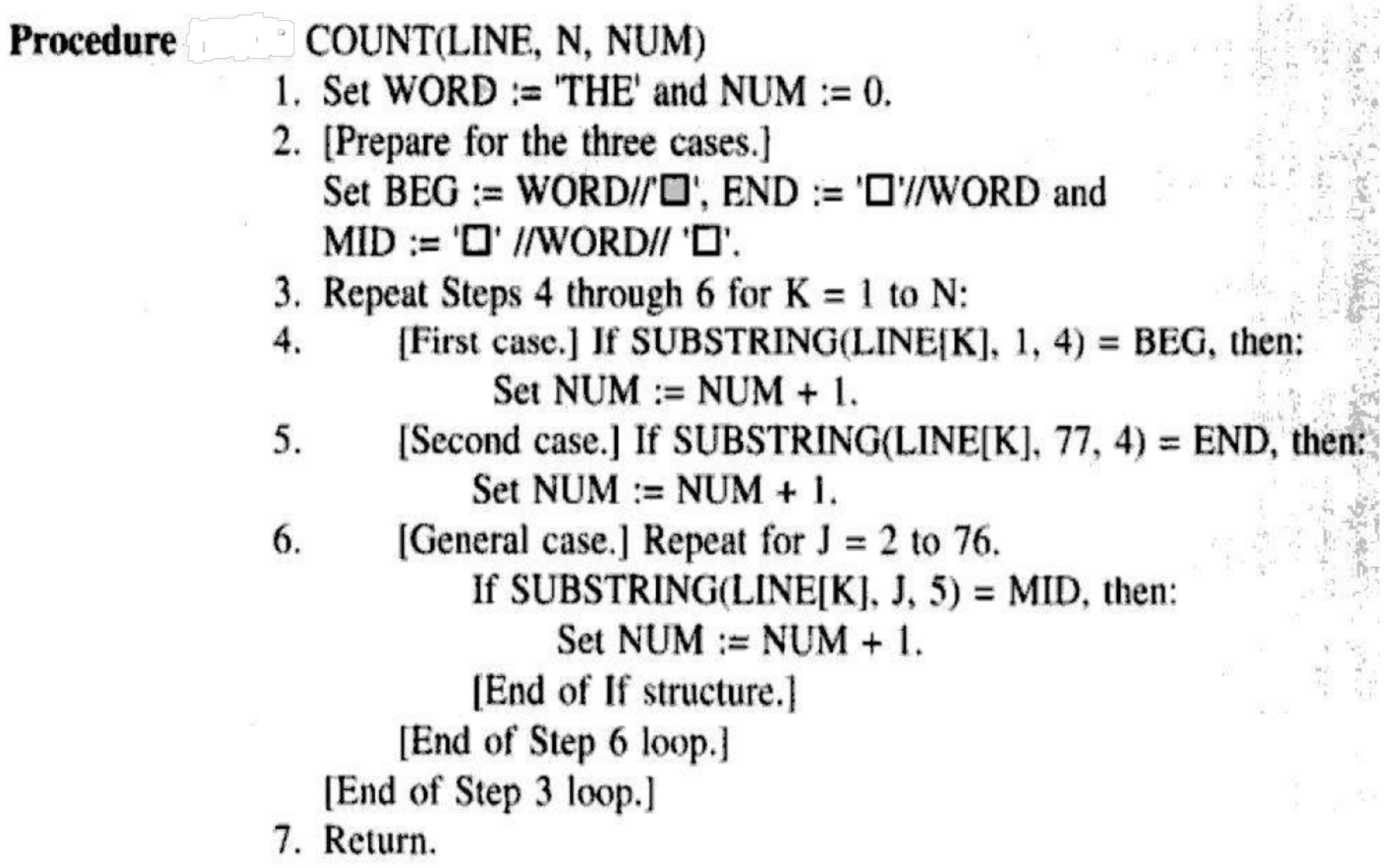
This algorithm computes sum of N elements of a given array A

```
1. [ Initialize SUM]
        SUM ← 0
2. Repeat for I = 1 to N
3. SUM ← SUM + A[I]
        [end of step 2 ]
4.[Output result computed]
        Write(SUM)
5. [Finished]
        Exit
```

Example

```
Algorithm  SUMPROD (A, N)
This algorithm computes sum and product of N elements of a given array A
1. [Initialize SUM and PROD]
            SUM ← 0
            PROD ← 1
2. Repeat through 4 for I = 1 to N
3. SUM ← SUM + A[I]
4.  PROD ← PROD * A[I]
            [end of step 2 ]
5.[Output result computed]
            Write(SUM)
            Write (PROD)
6. [Finished]
            Exit
```

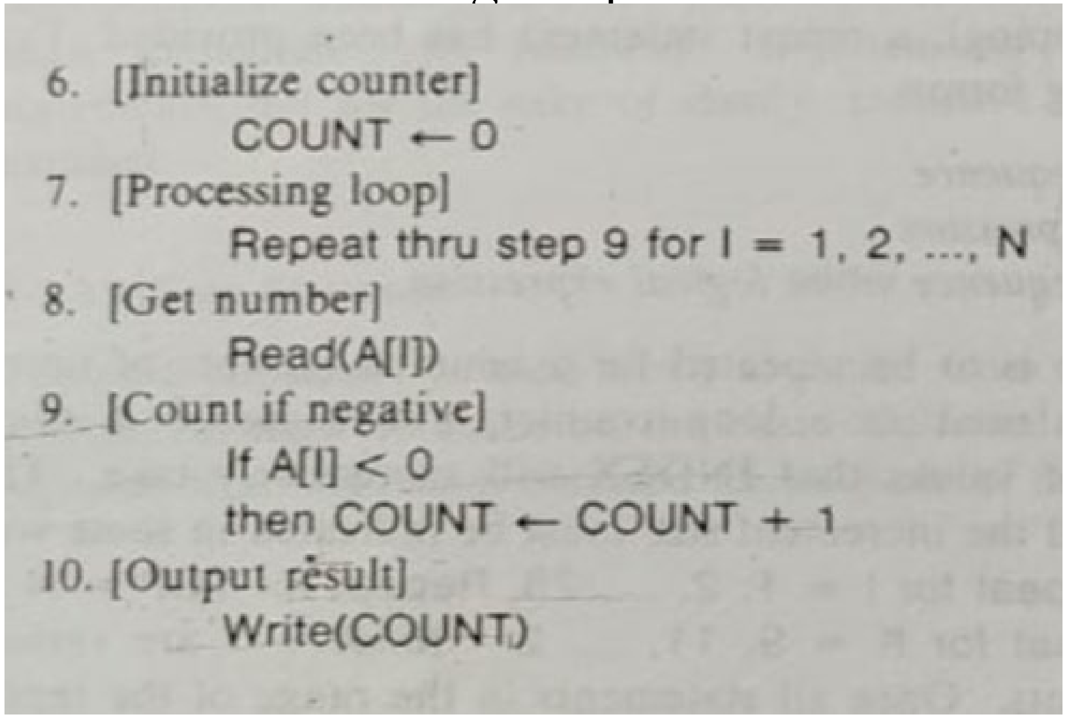
Example:



```
Procedure COUNT(LINE, N, NUM)
1.  Set WORD := 'THE' and NUM := 0.
2.  [Prepare for the three cases.]
    Set BEG := WORD//'□', END := '□'//WORD and
    MID := '□' //WORD// '□'.
3.  Repeat Steps 4 through 6 for K = 1 to N:
4.          [First case.] If SUBSTRING(LINE[K], 1, 4) = BEG, then:
                    Set NUM := NUM + 1.
5.          [Second case.] If SUBSTRING(LINE[K], 77, 4) = END, then:
                  Set NUM := NUM + 1.
6.          [General case.] Repeat for J = 2 to 76.
                  If SUBSTRING(LINE[K], J, 5) = MID, then:
                          Set NUM := NUM + 1.
                  [End of If structure.]
            [End of Step 6 loop.]
    [End of Step 3 loop.]
7.  Return.
```

Variable Name

- In pseudocode, a variable name is used to represent a storage location in memory that can hold values.
- These values may change during the execution of the algorithm.
- Variable names in pseudocode should be descriptive, helping to indicate the purpose of the variable and making the pseudocode easier to understand



```
6.  [Initialize counter]
            COUNT ← 0
7.  [Processing loop]
            Repeat thru step 9 for I = 1, 2, ..., N
8.  [Get number]
            Read(A[I])
9.  [Count if negative]
            If A[I] < 0
            then COUNT ← COUNT + 1
10. [Output result]
            Write(COUNT)
```

Assignment Statement

- In a pseudocode algorithm, an assignment statement is used to assign a value to a variable.
- Assignment statement essentially tells the computer to calculate the value on the right-hand side of the assignment operator (usually represented as ←) and store it in the variable on the left-hand side.
- The assignment operator can be thought of as a means of giving the variable a new value based on some computation or input.

Input and Output

- **Input**: These are the data items that the pseudocode receives from external sources, which could be users, files, other systems, or any data stream.
- Inputs are necessary for the algorithm to perform its function
- They are often specified at the beginning of the pseudocode or within the steps that require external data.

```
8. [Get number]
        Read (A[I])
9. [Count if negative]
        If $A[I] < 0$
        then COUNT $\leftarrow$ COUNT +1
10. [Output result]
        Write(COUNT)
```

Input and Output

- **Output**: These are the results produced by the pseudocode after processing the input data.
- Outputs can be directed to the screen, saved in a file, sent over a network, etc.
- Outputs are usually defined in pseudocode at points where results need to be displayed or utilized.

```
8. [Get number]
        Read (A[I])
9. [Count if negative]
        If A[I] < 0
        then COUNT ← COUNT +1
10. [Output result]
        Write(COUNT)
```

Function or Procedure

- **Function**: A function is a named section of a program that performs a specific task and can return a value.
- It usually takes some inputs, called parameters, does some processing, and often returns a result.
- Functions are used to modularize and reuse code across different parts of a program.
- **Procedure**: A procedure is similar to a function but typically does not return a value.
- It is used to execute a sequence of steps. Like functions, procedures can take parameters and are used to break down tasks into smaller, manageable parts, often focusing on performing a specific operation or action within the program.

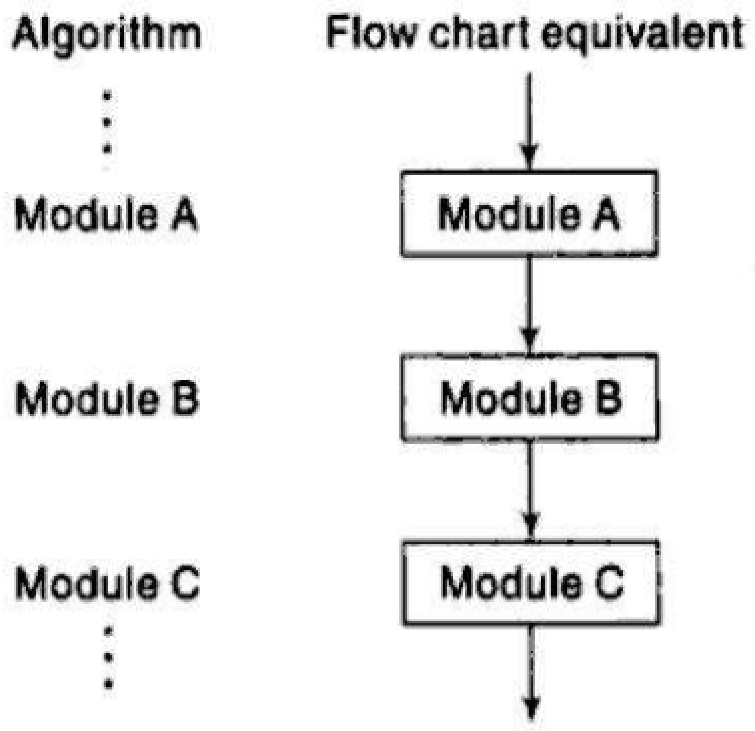
Control Structures

- Algorithms and their equivalent computer programs are more easily understood if they mainly use self-contained modules and three types of logic ,or flow of control, called

1. Sequence logic. or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

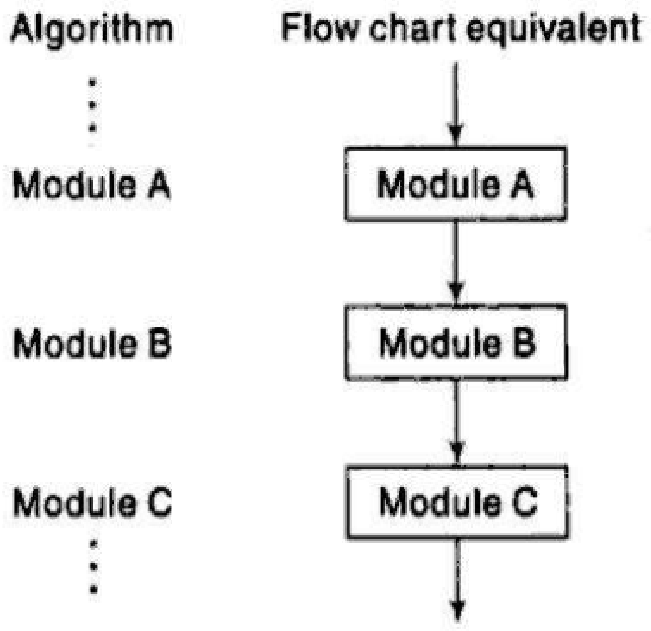
Sequence logic (Sequential Flow)

- Sequential logic in programming refers to the process of executing instructions in a specific order, one after the other, as they appear in the code.
- Each instruction or statement is executed exactly once, from the top of the code to the bottom, unless some control structure alters the flow (like conditionals, loops, or functions).



Key Characteristics of Sequential Logic:

- **Linear Flow**: Instructions are executed in a linear sequence. The output of one instruction might become the input for the next.
- **No Branching or Jumping**: Unlike branching logic (which uses IF statements or loops), sequential logic doesn't alter the flow of control.
- **Predictable Execution**: Since the instructions are executed in the order they are written, the behavior of the program is predictable and easy to follow.



Use of Sequential Logic:

- Simple Operations: Sequential logic is often used for simple tasks where steps need to happen in a fixed order.
- Initialization: Setting up variables or preparing the environment usually follows sequential logic before entering more complex flow control structures.
- Linear Tasks: Processes that don't require decisions or repeated actions often follow a sequential flow, such as reading data from a file and processing it line by line.

Selection logic (Conditional Flow)

- Selection logic, also known as conditional flow or branching, allows a program to make decisions and execute specific blocks of code based on conditions or criteria.
- Sequential logic, where **every statement is executed one after the other**, selection logic directs the flow of the program based on whether certain conditions are met.
- Selection logic is fundamental to making **decisions in programming**. It allows the program to handle different scenarios dynamically based on conditions and is a key component in building responsive, interactive applications.

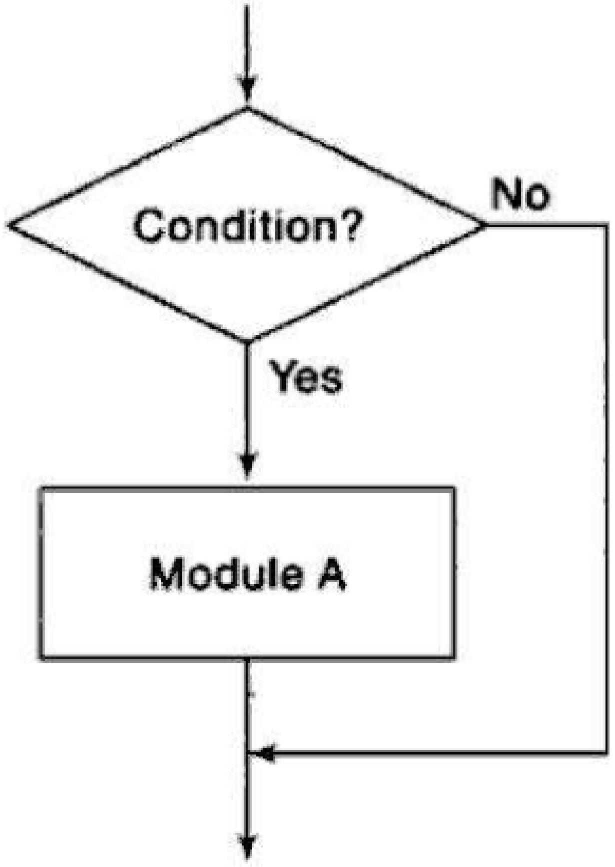
Selection logic (Conditional Flow)

- Selection logic, also known as conditional flow or branching, allows a program to make decisions and execute specific blocks of code based on conditions or criteria. Unlike sequential logic, where every statement is executed one after the other, selection logic directs the flow of the program based on whether certain conditions are met.
- **Key Elements of Selection Logic:**
- **Conditional Statements**: These are used to evaluate a condition and control the flow of execution based on that condition.
- **Boolean Expressions**: Conditions are usually represented as Boolean expressions (expressions that evaluate to either True or False).
- **Branching**: Different branches of code can be executed based on the outcome of the condition. This creates different paths in the program depending on the input or state

Single alternative

This structure has the form

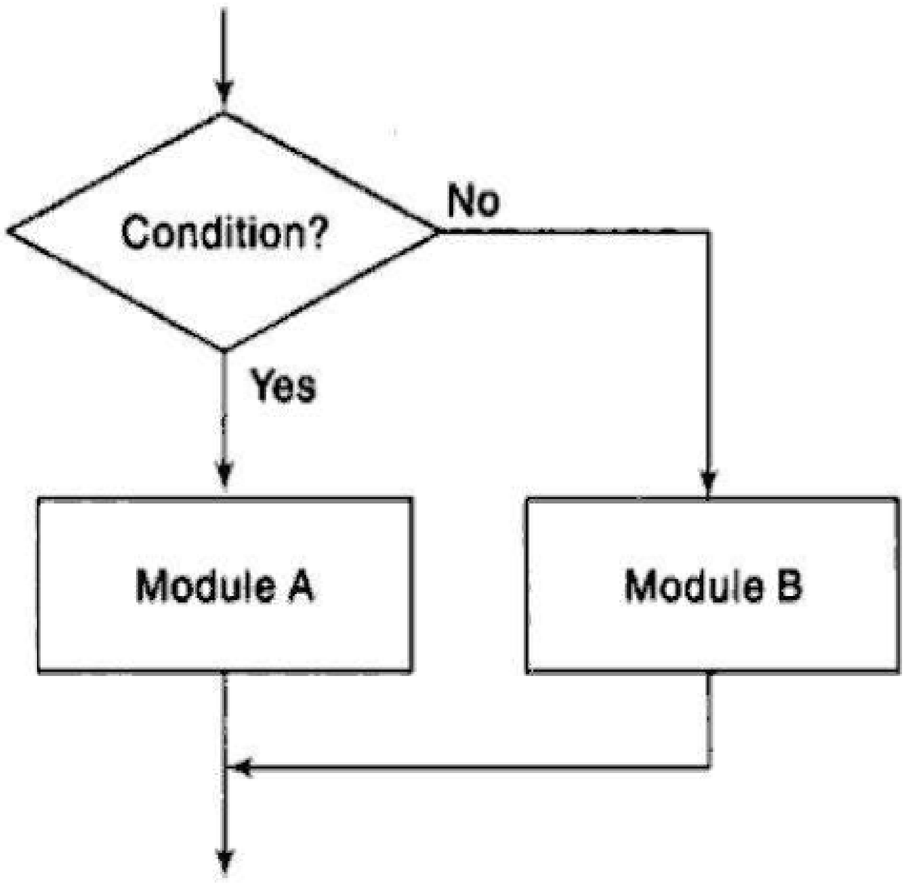
```
If condition, then:
    [Module A]
[End of If structure.]
```



(a) Single alternative.

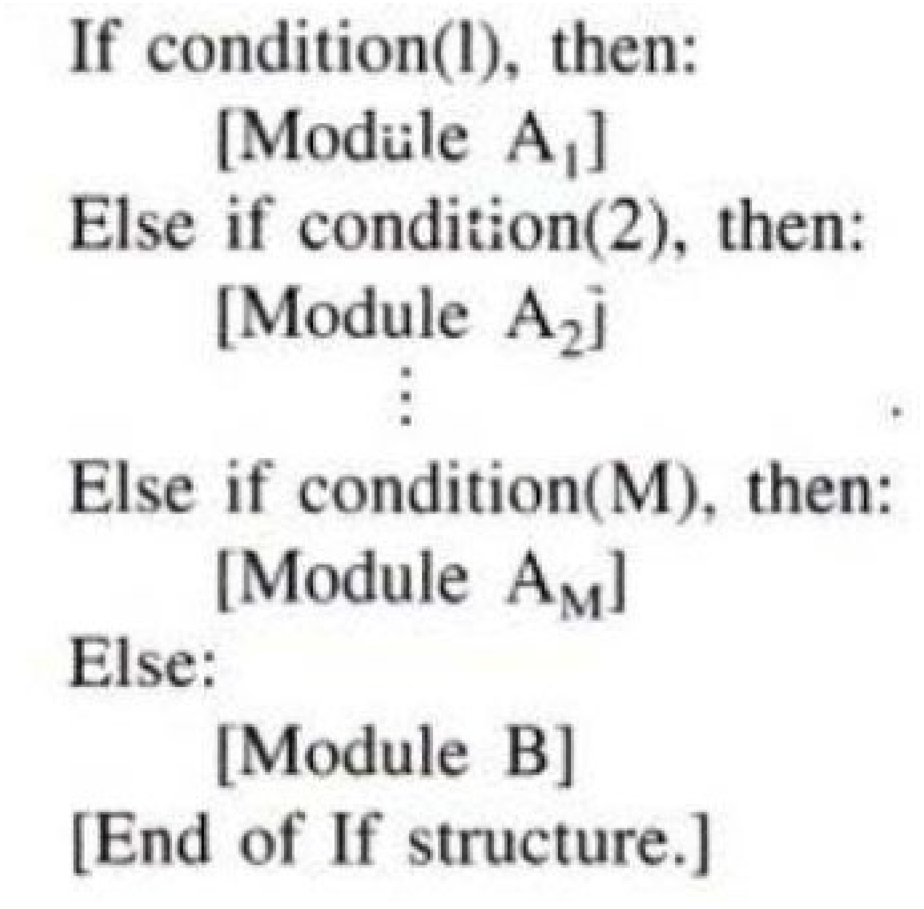
Double alternative

```
If condition, then:
        [Module A]
Else:
        [Module B]
[End of If structure.]
```



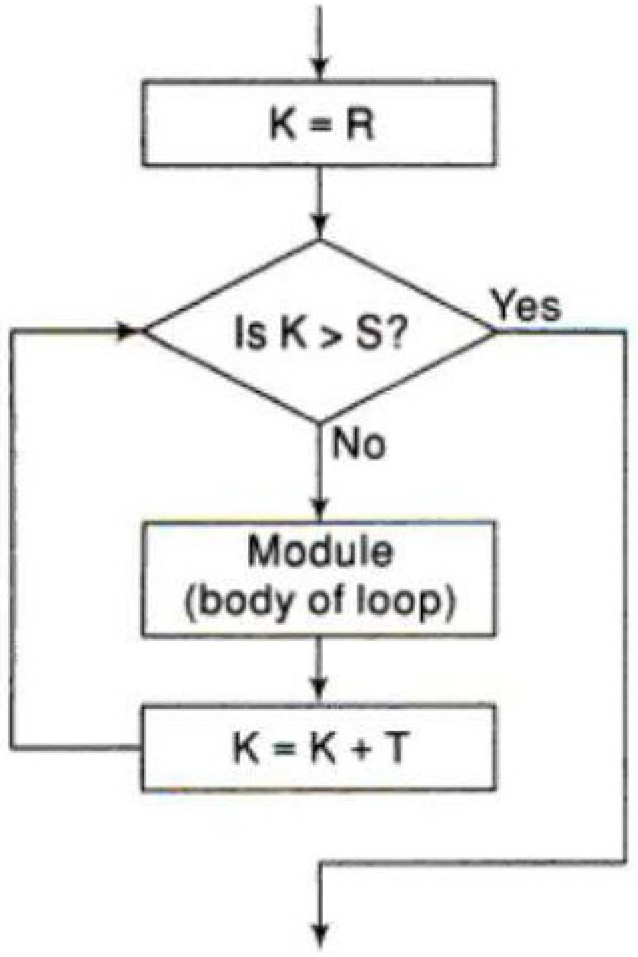
(b) Double alternative.

Multiple alternative

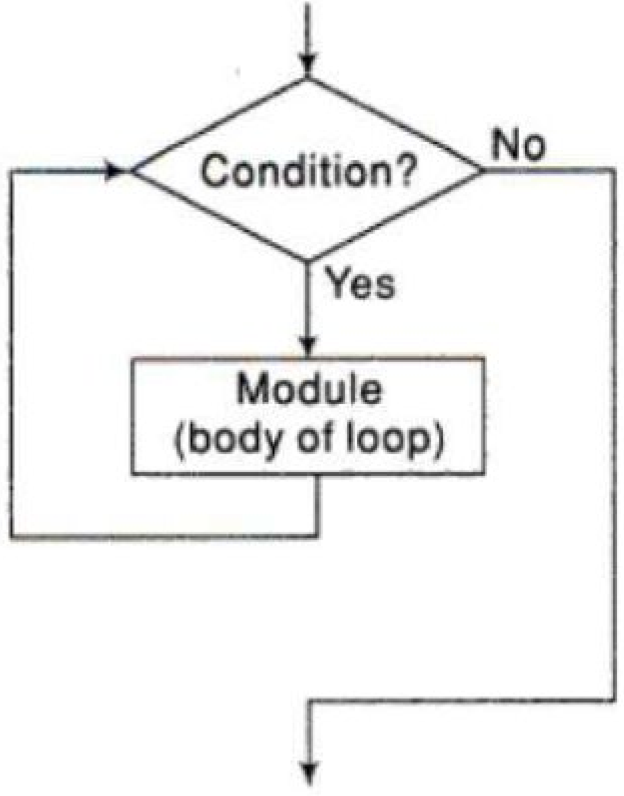


```
If condition(1), then:
        [Module $A_1$]
Else if condition(2), then:
        [Module $A_2$]
            ⋮
Else if condition(M), then:
        [Module $A_M$]
Else:
        [Module B]
[End of If structure.]
```

Iterative logic Repeat Flow



(a) Repeat-For structure.

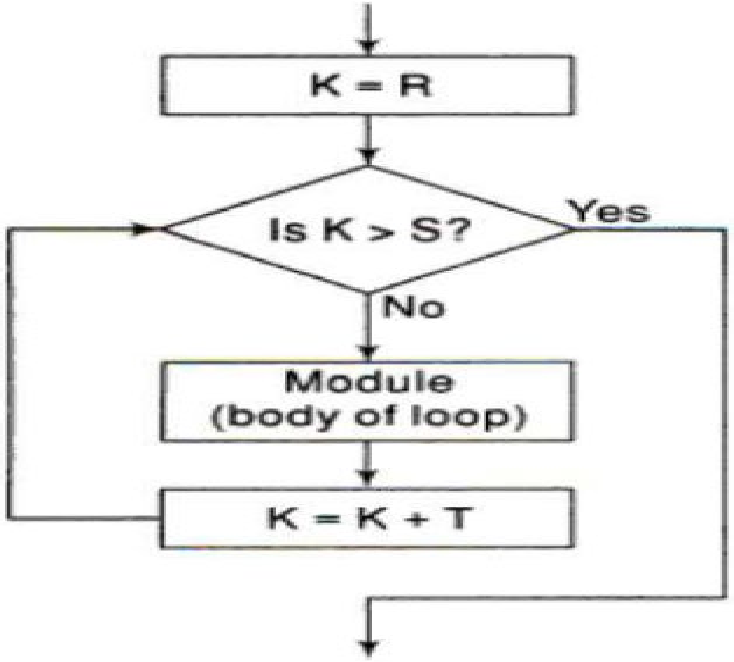


(a) Repeat-While structure.

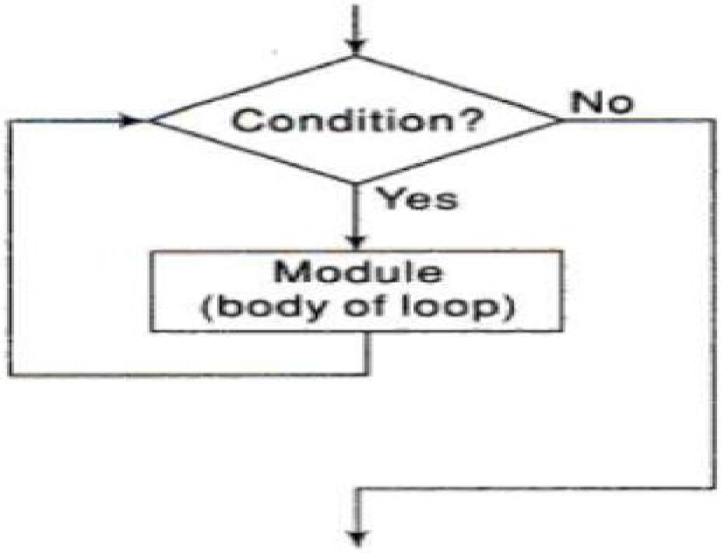
Iterative logic Repeat Flow

```
Repeat for K = R to S by T:
        [Module]
[End of loop.]
```

```
Repeat while condition:
        [Module]
[End of loop.]
```

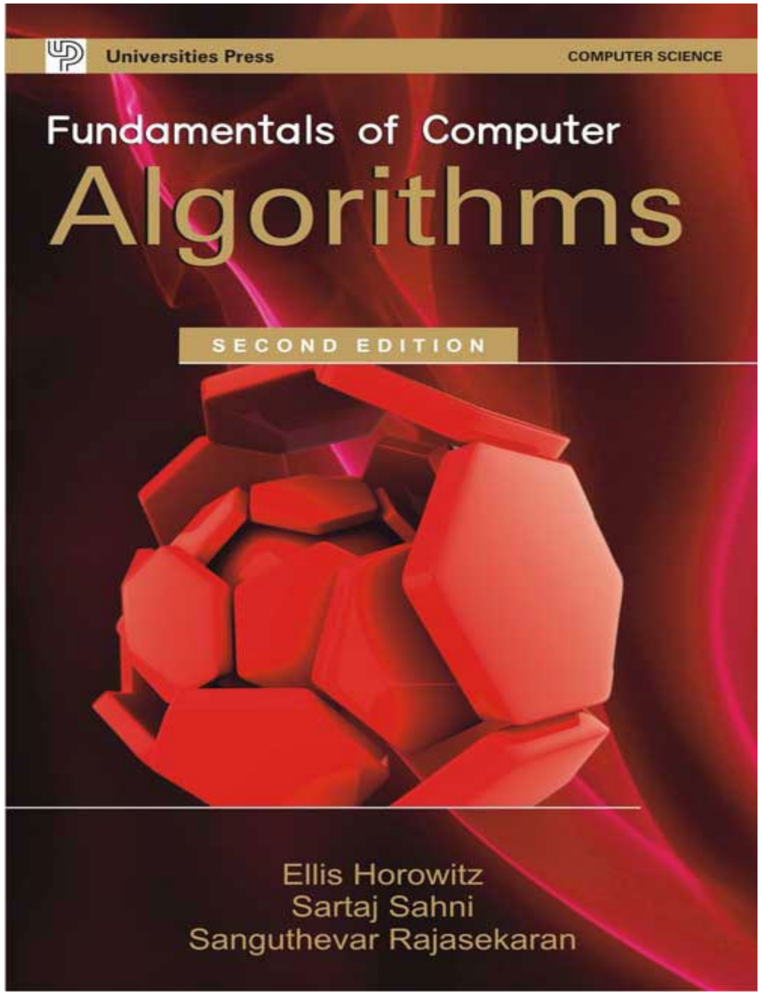


(a) Repeat-For structure.



(a) Repeat-While structure.

Text Book


Universities Press
COMPUTER SCIENCE
Fundamentals of Computer
Algorithms
SECOND EDITION
Ellis Horowitz
Sartaj Sahni
Sanguthevar Rajasekaran

Pseudocode Conventions

[1] Comments begin with / / and continue until the end of line.

[2] Blocks are indicated with ·matching braces: { and }. A compound statement (i.e., a collection of simple statements] can be represented as a block. The body of a procedure also forms a block. Statements are delimited by ; .

[3] An identifier begins with a letter. The data types of variables are not explicitly declared. The types will be clear from the context.

Pseudocode Conventions

[4] Assignment of values to variables is done using the assignment statement

(variable) := (expression);

[5] There are two boolean values **true** and **false**. In order to produce these values, the ***logical operators*** and, **or,** and **not** and the ***relational operators*** $<, \leq, =, \neq, \geq$, and $>$ are provided.

[6] Elements of multidimensional arrays are accessed using [and]. For example, if A is a two dimensional array, the (i, j)th element of the array is denoted as A[i, j]. Array indices start at zero.

Pseudocode Conventions

[7] The looping statements supported by the pseudo code are: for, while, and repeat - until.

The while loop takes the following form

```
While < condition> do
{
        <statement1>
}
```

Pseudocode Conventions

8. A conditional statement has the following forms:

```
if ⟨condition⟩ then ⟨statement⟩
if ⟨condition⟩ then ⟨statement 1⟩ else ⟨statement 2⟩
```

Here ⟨*condition*⟩ is a boolean expression and ⟨*statement*⟩, ⟨*statement* 1⟩, and ⟨*statement* 2⟩ are arbitrary statements (simple or compound).

We also employ the following **case** statement:

```
case
{
        :⟨condition 1⟩: ⟨statement 1⟩
                  ⋮
        :⟨condition n⟩: ⟨statement n⟩
        :else: ⟨statement n + 1⟩
}
```

Pseudocode Conventions

[9] Input and output are done using the instructions *read* and *write*. No format is used to specify the size of input or output quantities.

[10] There is only one type of procedure: *Algorithm*. An algorithm consists of a heading and a body.

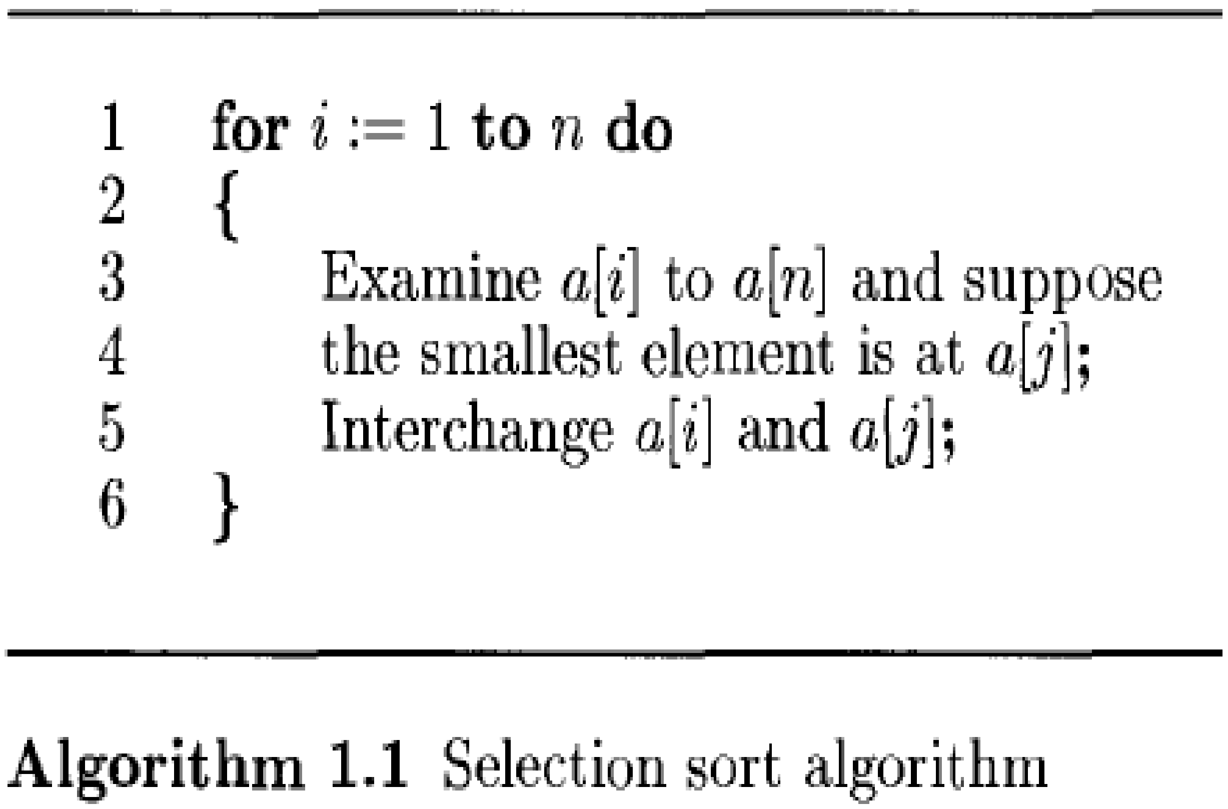
The heading takes the form

Algorithm *Name ((parameter list))*

```
Algorithm Max($A$, $n$)
// $A$ is an array of size $n$.
{
    $Result := A[1]$;
    for $i := 2$ to $n$ do
        if $A[i] > Result$ then $Result := A[i]$;
    return $Result$;
}
```

Not an algorithm

- [Selection sort]



```
for $i := 1$ to $n$ do
{
    Examine $a[i]$ to $a[n]$ and suppose
    the smallest element is at $a[j]$;
    Interchange $a[i]$ and $a[j]$;
}
```

Algorithm 1.1 Selection sort algorithm

Algorithm finds and returns the maximum of *n* given numbers:

```
Algorithm Max($A$, $n$)
// $A$ is an array of size $n$.
{
    $Result$ := $A[1]$;
    for $i$ := 2 to $n$ do
        if $A[i] > Result$ then $Result$ := $A[i]$;
    return $Result$;
}
```

In this algorithm(named Max), A and n are procedure parameters. ***Result*** and ***i*** are local variables

Algorithm: Selection Sort

```
Algorithm SelectionSort($a, n$)
// Sort the array $a[1 : n]$ into nondecreasing order.
{
    for $i := 1$ to $n$ do
    {
        $j := i$;
        for $k := i + 1$ to $n$ do
            if $(a[k] < a[j])$ then $j := k$;
        $t := a[i]$; $a[i] := a[j]$; $a[j] := t$;
    }
}
```

Theorem 1.1 Algorithm SelectionSort(a, n) correctly sorts a set of $n \geq 1$ elements; the result remains in $a[1 : n]$ such that $a[1] \leq a[2] \leq \cdots \leq a[n]$.

Correctness of an algorithm

- In theoretical computer science, **correctness** of an **algorithm** is asserted when it is said that the **algorithm** is correct with respect to a specification.
- Functional **correctness** refers to the input-output behaviour of the **algorithm** (i.e., for each input it produces the expected output).
- An algorithm is correct if:
  - – for any correct input data: it stops and it produces correct output.
  - –Correct input data: **satisfies pre-condition**
  - –Correct output data: **satisfies post-condition**

Total Correctness of Algorithm

- **Definition:** An algorithm which for any correct input data: (i) **stops** and (ii) **returns correct output** is called **totally correct** for the given specification.

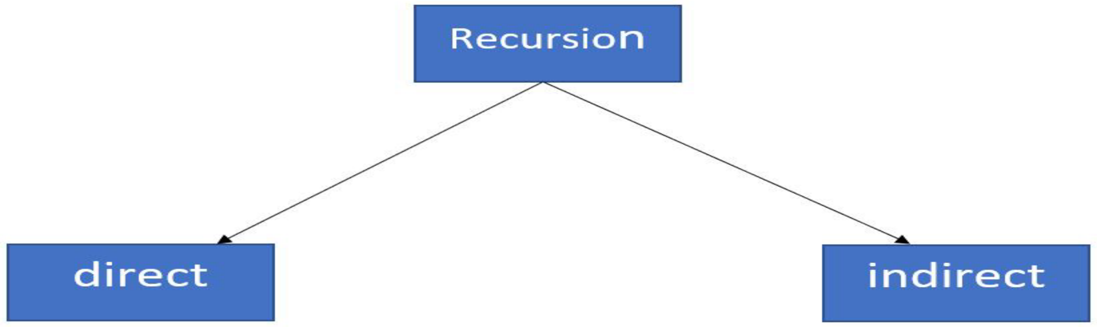
These split into 2 sub-properties in the definition above.

- **correct input data** is the data which satisfies the initial condition of the specification
- **correct output data** is the data which satisfies the final condition of the specification

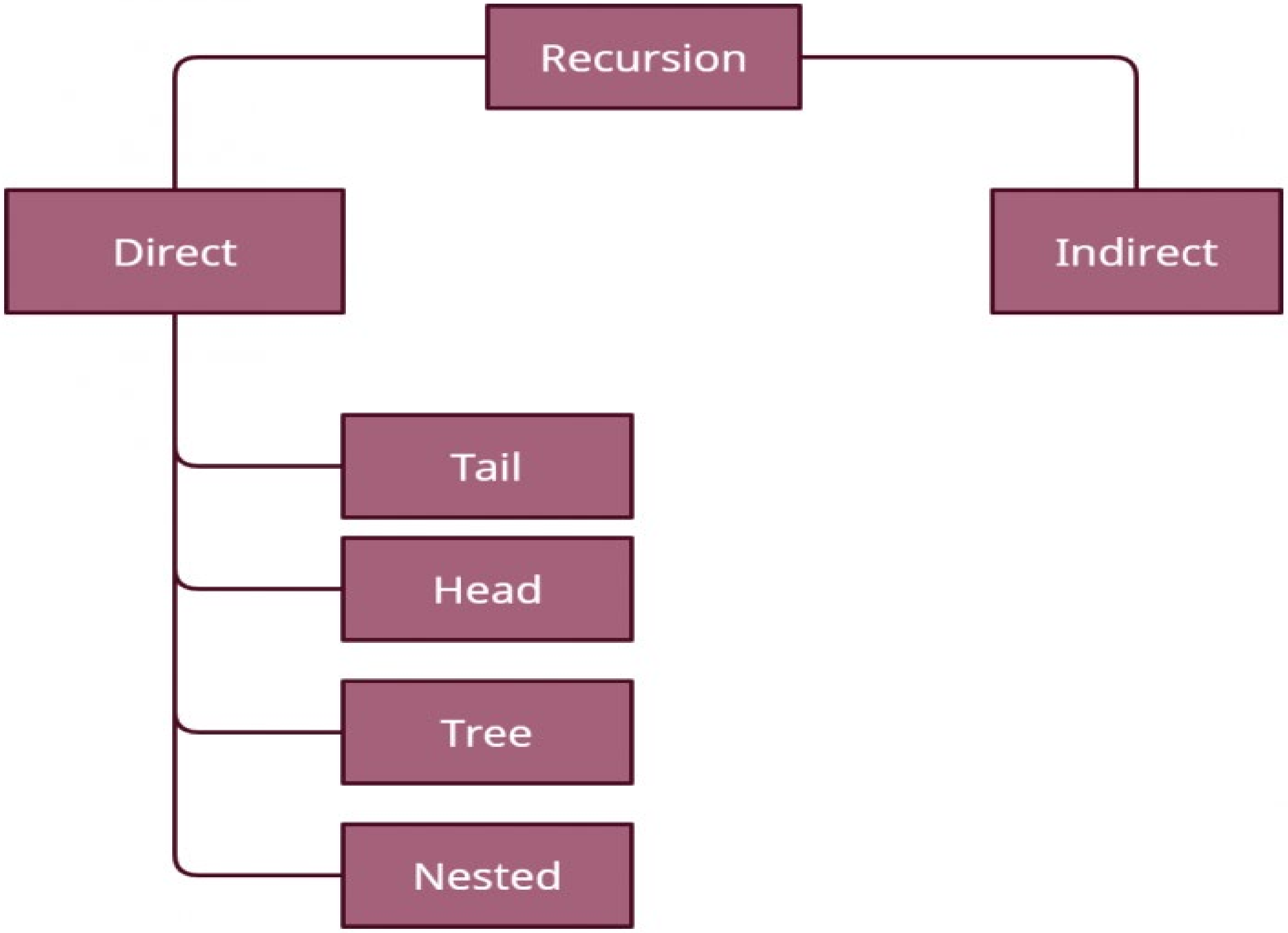
Recursive Algorithms

- A recursive function is a function that is defined in terms of itself. Similarly, an algorithm is said to be recursive if the same algorithm is invoked in the body.
- An algorithm that calls itself is **direct recursive**. Algorithm A is said to be **indirect recursive** if it calls another algorithm B which in turn calls A.

1. **Directly recursive**: a function that calls itself
2. **Indirectly recursive**: a function that calls another function and eventually results in the original function call



Type of recursion



Different type of direct recursion

[1] Tail Recursion:

- If a recursive function calling itself and that recursive call is the last statement in the function then it's known as **Tail Recursion.**
- After that call the recursive function performs nothing. The function has to process or perform any operation at the time of calling and it does nothing at returning time.

[2] Head Recursion:

- If a recursive function calling itself and that recursive call is the first statement in the function then it's known as **Head Recursion.**
- There's no statement, no operation before the call. The function doesn't have to process or perform any operation at the time of calling and all operations are done at returning time.

Different type of direct recursion

[3] Tree Recursion:

- To understand **Tree Recursion** let's first understand **Linear Recursion**. If a recursive function calling itself for one time then it's known as **Linear Recursion**. Otherwise if a recursive function calling itself for more than one time then it's known as **Tree Recursion**.

[4] Nested Recursion:

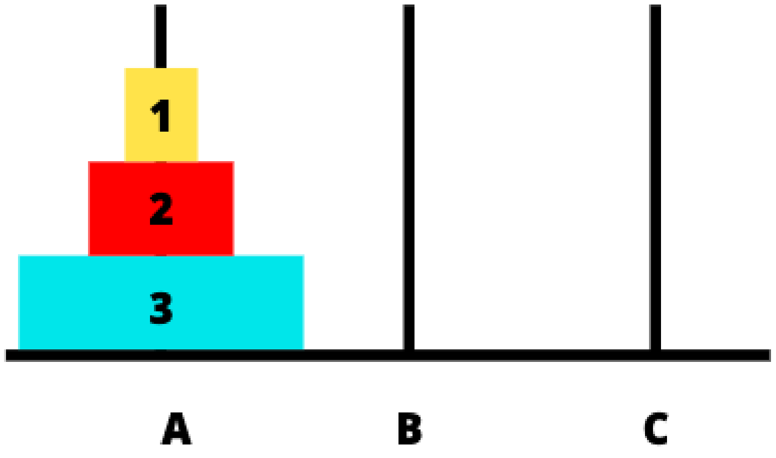
- In this recursion, a recursive function will pass the parameter as a recursive call. That means **"recursion inside recursion".**

Difference between recursion and iteration

Property	Recursion	Iteration
Definition	A program is recursive when a function calls itself repeatedly until a base condition is met.	A program is iterative if a set of instructions is executed repeatedly using a loop.
Infinite recursion	Infinite recursion can lead to a system crash. Infinite recursion can occur due to a mistake in specifying the base condition.	In an iterative process, the program stops when the memory is exhausted.
Code size	Smaller code size	Larger code size
Time and space	Calling a function each time adds a layer to the stack. Performing the push and pop operation in stack memory makes recursion expensive in both processor time and as well as memory space.	No stack is required in the case of an iterative solution. Hence, an iterative process is a time and space-efficient as compared to the recursive process.
Termination	A recursive code terminates when a base condition is met.	A control variable decides the termination of an iterative process.

Recursive algorithm : Towers of Hanoi

```
Algorithm TowersOfHanoi($n, x, y, z$)
// Move the top $n$ disks from tower $x$ to tower $y$.
{
    if ($n \geq 1$) then
    {
        TowersOfHanoi($n - 1, x, z, y$);
        write ("move top disk from tower", $x$,
        "to top of tower", $y$);
        TowersOfHanoi($n - 1, z, y, x$);
    }
}
```



Recursive algorithm: Permutation Generator

Example 1.3 [Permutation generator] Given a set of $n \geq 1$ elements, the problem is to print all possible permutations of this set.

```
Algorithm Perm($a, k, n$)
{
    if ($k = n$) then write ($a[1 : n]$); // Output permutation.
    else // $a[k : n]$ has more than one permutation.
        // Generate these recursively.
        for $i := k$ to $n$ do
        {
            $t := a[k]$; $a[k] := a[i]$; $a[i] := t$;
            Perm($a, k + 1, n$);
            // All permutations of $a[k + 1 : n]$
            $t := a[k]$; $a[k] := a[i]$; $a[i] := t$;
        }
}
```

Recursive binary search

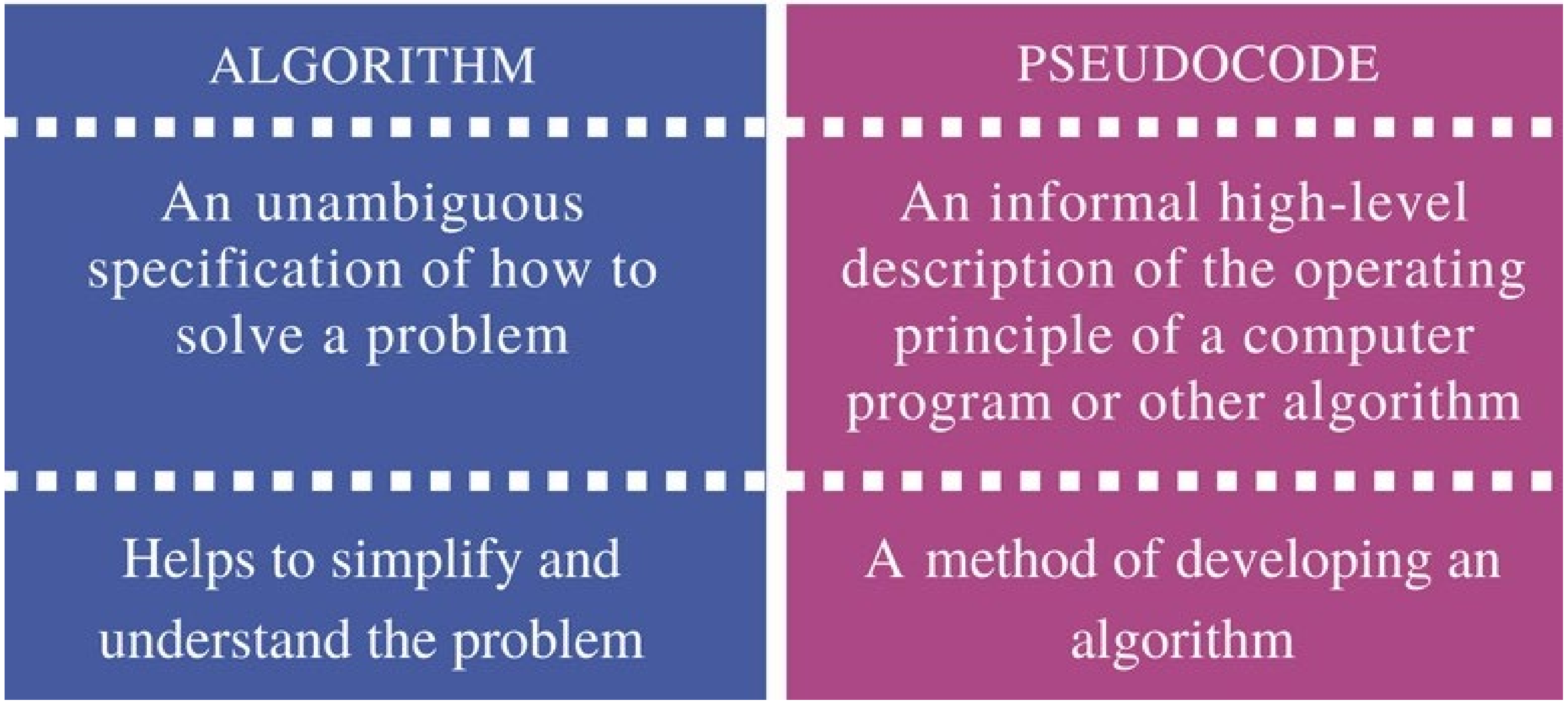
```
Algorithm BinSrch($a, i, l, x$)
// Given an array $a[i : l]$ of elements in nondecreasing
// order, $1 \leq i \leq l$, determine whether $x$ is present, and
// if so, return $j$ such that $x = a[j]$; else return 0.
{
    if ($l = i$) then  // If Small($P$)
    {
        if ($x = a[i]$) then return $i$;
        else return 0;
    }
    else
    { // Reduce $P$ into a smaller subproblem.
        $mid := \lfloor (i + l)/2 \rfloor$;
        if ($x = a[mid]$) then return $mid$;
        else if ($x < a[mid]$) then
                return BinSrch($a, i, mid - 1, x$);
            else return BinSrch($a, mid + 1, l, x$);
    }
}
```

Iterative binary search

```
Algorithm BinSearch($a, n, x$)
// Given an array $a[1:n]$ of elements in nondecreasing
// order, $n \geq 0$, determine whether $x$ is present, and
// if so, return $j$ such that $x = a[j]$; else return 0.
{
    $low := 1$; $high := n$;
    while ($low \leq high$) do
    {
        $mid := \lfloor (low + high)/2 \rfloor$;
        if ($x < a[mid]$) then $high := mid - 1$;
        else  if ($x > a[mid]$) then $low := mid + 1$;
              else return $mid$;
    }
    return 0;
}
```

difference between algorithm and pseudocode

an algorithm is a step by step procedure to solve a given problem while a pseudocode is a method of writing an algorithm.



ALGORITHM	PSEUDOCODE
An unambiguous specification of how to solve a problem	An informal high-level description of the operating principle of a computer program or other algorithm
Helps to simplify and understand the problem	A method of developing an algorithm

Summary

- Pseudocode is an informal way of writing a program.
- It is not exactly a computer program.
- It represents the algorithm of the program in natural language and mathematical notations.
- There is no particular code syntax to write a pseudocode. Therefore, there is no strict syntax as a usual programming language. It uses simple English language.
- A pseudocode is a method of developing an algorithm.
- Pseudocode is a text-based detailed design tool.

Nine open-minded people you need in your life
The listener
The encourager
The inspirer
The optimist
The challenger
The practical motivator
The empathiser
YOU
The brutally honest
The accountability partner
Good morning

Thanks for Your Attention!

Exercises

Exercise # 1: Write algorithms in Pseudo code

1. Sequential Logic: Write a pseudocode algorithm to calculate the area of a rectangle given its length and width.
2. Selection Logic (IF-ELSE): Write a pseudocode to check if a number is positive, negative, or zero.
3. Selection Logic (Nested IF): Write a pseudocode to determine if a student passed or failed based on their score, and if they achieved distinction.
4. Iteration (FOR Loop): Write a pseudocode to print the first 10 natural numbers.
5. Iteration (WHILE Loop): Write a pseudocode to calculate the sum of digits of a number.
6. Iteration (DO-WHILE Loop): Write a pseudocode to prompt the user to enter a number until they enter a positive number.
7. Selection Logic (Switch/Case): Write a pseudocode to print the name of the day of the week baed on a number input (1 for Monday, 2 for Tuesday, etc.).
8. Nested Loops: Write a pseudocode to print a multiplication table for numbers 1 through 5.
9. Procedure/Function: Write a pseudocode to create a function that calculates the factorial of a number.
10. Recursion (Function): Write a pseudocode to calculate the nth Fibonacci number using recursion.

Exercise # 2: Write algorithms in Pseudo code

1. Present an algorithm that searches an unsorted array a[l: n] for the element x. If x occurs, then return a position in the array; else return zero.
2. The factorial function n! has value 1, when n $\leq$ 1and value n * (n-1)!, when n > 1. Write both a recursive and an iterative algorithm to Compute n!.
3. The Fibonacci numbers are defined $f_0 = 0$, $f_1 = 1$,and $f_j = f_{j-1} + f_{j-2}$ for j > 1.Write both **a recursive** and **an iterative algorithm** to compute n^th Fibonacci number.
4. Give an algorithm to solve the following problem: Given n, a positive integer, determine whether n is the sum of all of its divisors, that is, whether n is the sum of all t such that 1$\leq$ t < n, and t divides n.

Exercise #3 : Write algorithms in Pseudo code

5. Given n boolean variables x_1, x_2, x_3, …, and x_n, we wish to print all possible combinations of truth values they can assume. For instance, if n = 2,there are four possibilities: true, true; true, false; false, true; and false, false. Write an algorithm to accomplish this.
6. Devise an algorithm that inputs three integers and outputs them in Non-decreasing order.
7. Give an algorithm to solve the following problem: Given n, a positive integer, determine whether n is the sum of all of its divisors, that is, whether n is the sum of all t such that $1 \leq t < n$, and t divides n.
8. If S is a set of n elements, the power set of S is the set of all possible Subsets of S. For example, if S = (a,b,c), then power set(S)= {( ), (a), (b), (c), (a,b), (a,c),(b,c), (a,b,c)}.Write a recursivealgorithm to compute power set(S).