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.

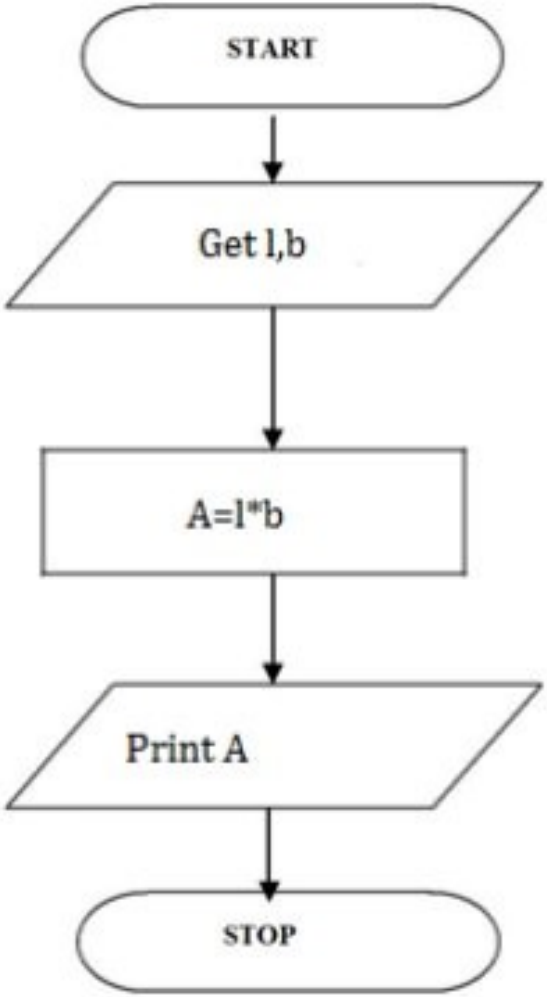
Difference between Algorithm, Pseudocode and Program

- **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

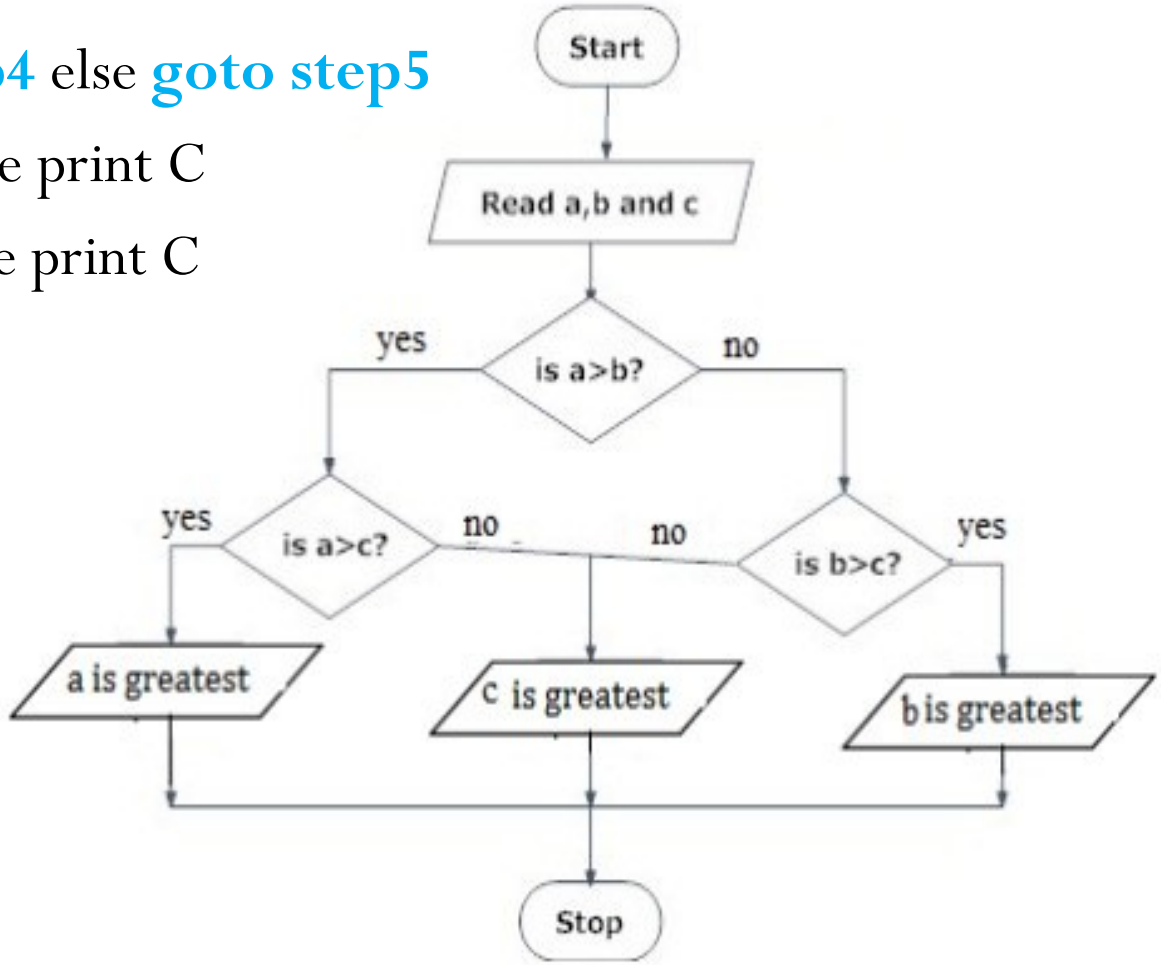
Step2: Get A, B, C

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



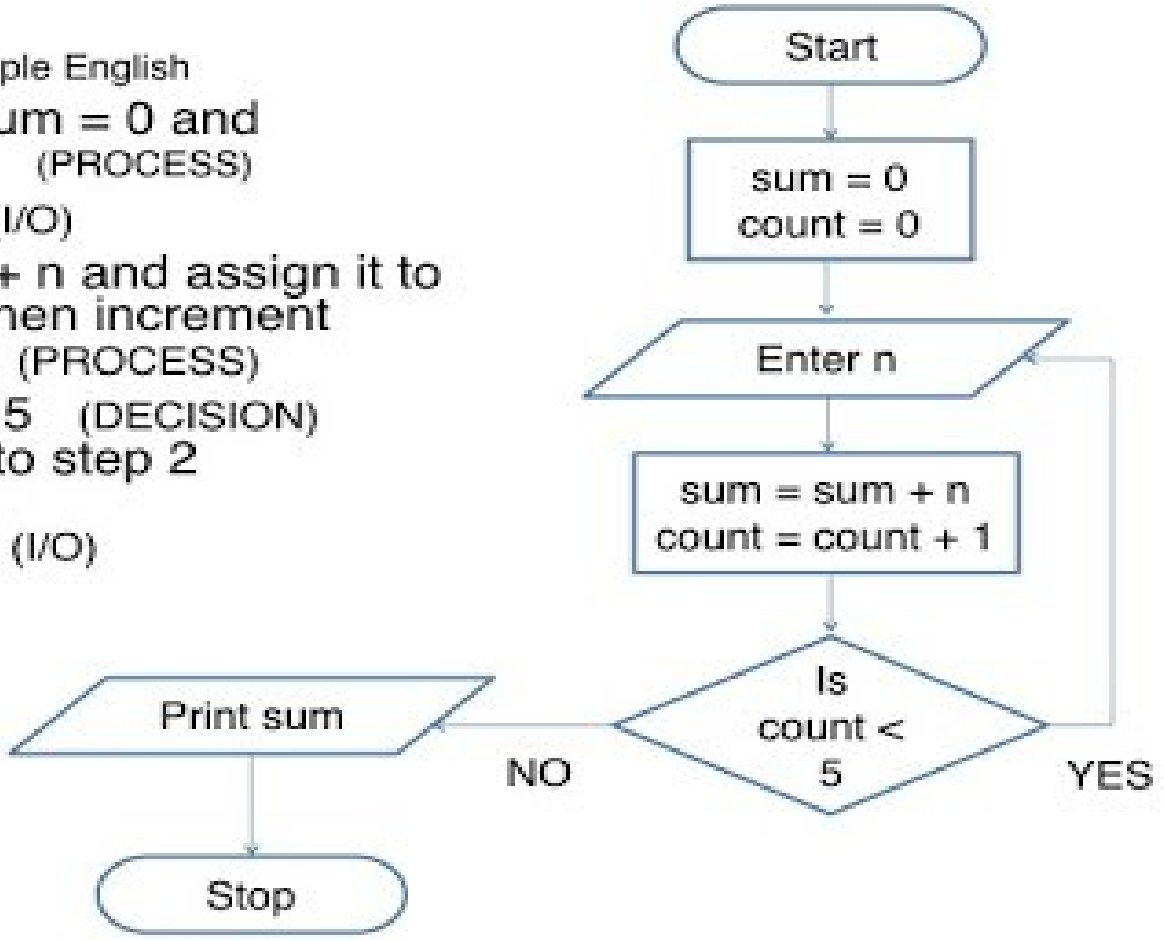
Example: Algorithm & Flowchart

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

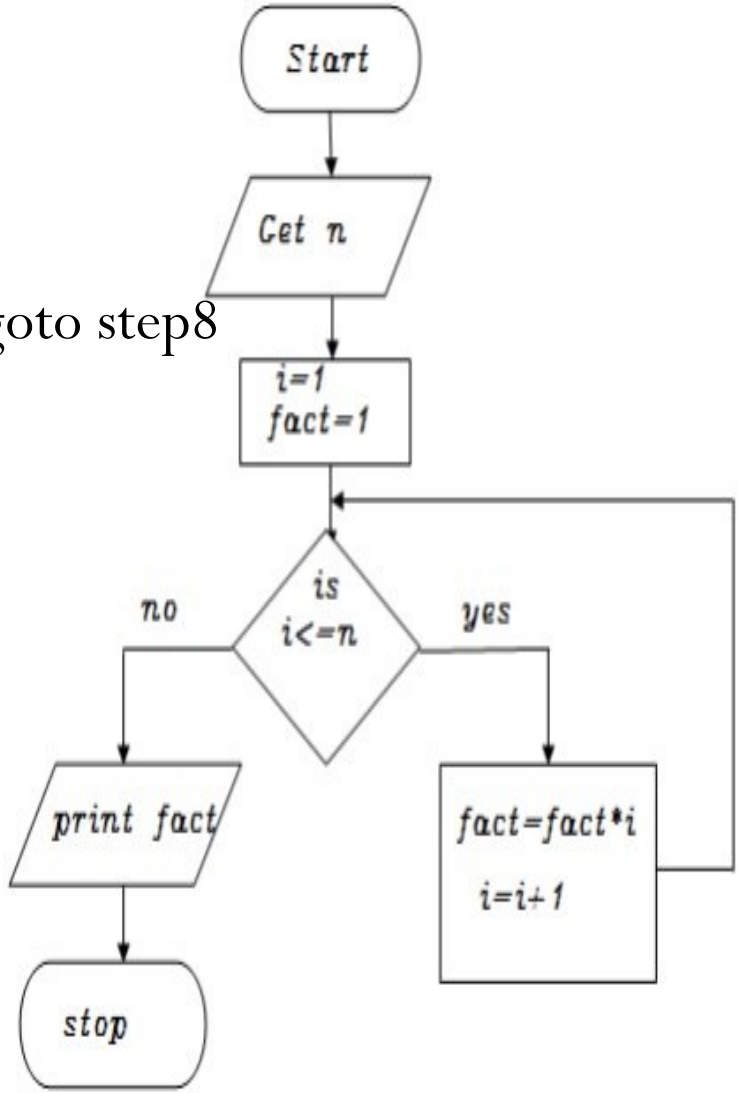
step 5: calculate fact=fact*i

step 6: increment i value by 1

step 7: goto step 4

step 8: print fact value

step 9: stop



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*** <,≤, =,≠,≥, 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))*

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

Not an algorithm

- [Selection sort]

```
1    for i := 1 to n do
2    {
3        Examine a[i] to a[n] and suppose
4        the smallest element is at a[j];
5        Interchange a[i] and a[j];
6    }
```

Algorithm 1.1 Selection sort algorithm

Algorithm finds and 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 i := 2 to n do
6          if A[i] > Result then Result := A[i];
7      return Result;
8  }
```

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

Algorithm: Selection Sort

```
1    Algorithm SelectionSort(a, n)
2    // Sort the array a[1 : n] into nondecreasing order.
3    {
4        for i := 1 to n do
5        {
6            j := i;
7            for k := i + 1 to n do
8                if (a[k] < a[j]) then j := k;
9            t := a[i]; a[i] := a[j]; a[j] := t;
10       }
11   }
```

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.

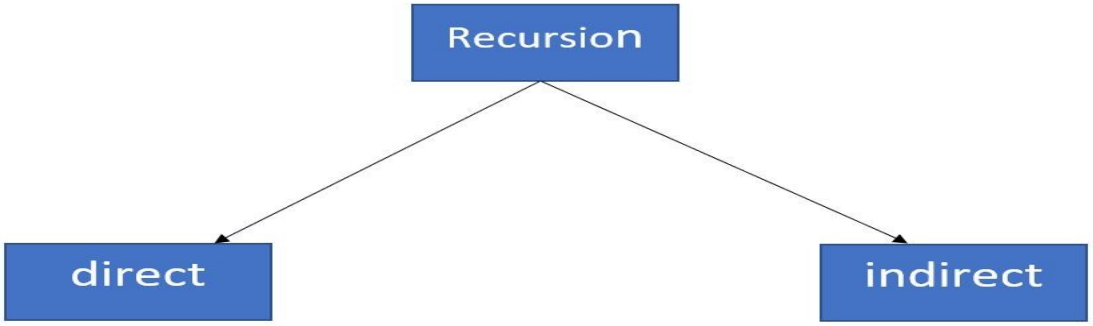
<u>These split into 2 sub-properties in the definition above.</u>

- **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

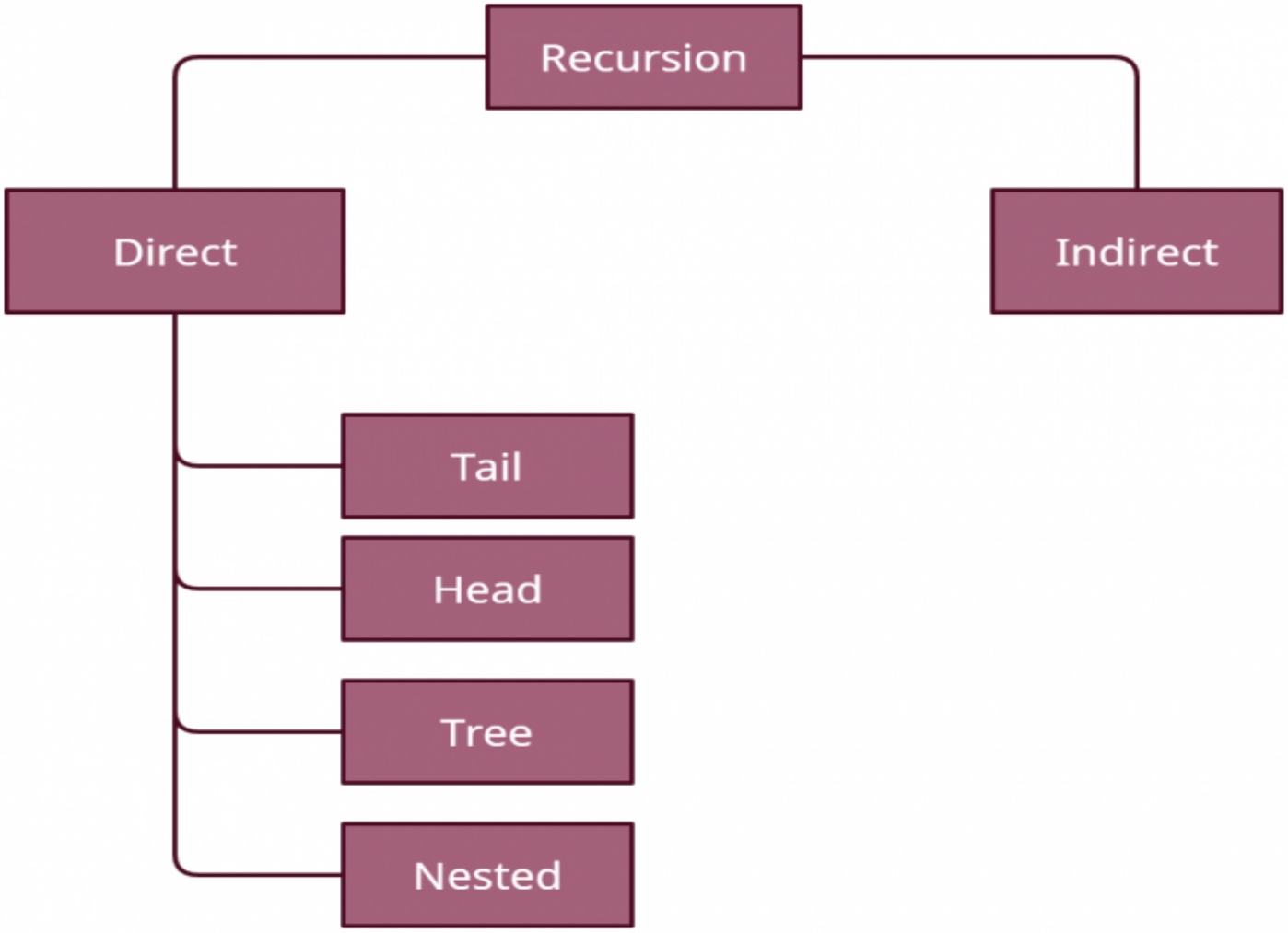
Recursive Algorithms

- 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:

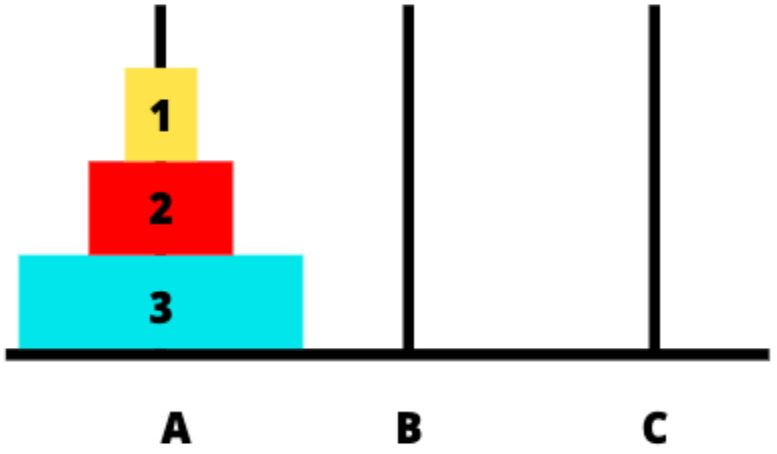
- In this recursion, a recursive function will pass the parameter as a recursive call. That means **"recursion inside recursion".**

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

```
1    Algorithm TowersOfHanoi(n, x, y, z)
2    // Move the top n disks from tower x to tower y.
3    {
4        if (n ≥ 1) then
5        {
6            TowersOfHanoi(n − 1, x, z, y);
7            write ("move top disk from tower", x,
8            "to top of tower", y);
9            TowersOfHanoi(n − 1, z, y, x);
10       }
11   }
```



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.

```
1   Algorithm Perm(a, k, n)
2   {
3       if (k = n) then write (a[1 : n]); // Output permutation.
4       else // a[k : n] has more than one permutation.
5            // Generate these recursively.
6            for i := k to n do
7            {
8                 t := a[k]; a[k] := a[i]; a[i] := t;
9                 Perm(a, k + 1, n);
10                // All permutations of a[k + 1 : n]
11                t := a[k]; a[k] := a[i]; a[i] := t;
12           }
13  }
```

Recursive binary search

```
1    Algorithm BinSrch(a, i, l, x)
2    // Given an array a[i : l] of elements in nondecreasing
3    // order, 1 ≤ i ≤ l, determine whether x is present, and
4    // if so, return j such that x = a[j]; else return 0.
5    {
6        if (l = i) then  // If Small(P)
7        {
8            if (x = a[i]) then return i;
9            else return 0;
10       }
11       else
12       { // Reduce P into a smaller subproblem.
13           mid := ⌊(i + l)/2⌋;
14           if (x = a[mid]) then return mid;
15           else if (x < a[mid]) then
16                    return BinSrch(a, i, mid − 1, x);
17                else return BinSrch(a, mid + 1, l, x);
18       }
19   }
```

Iterative binary search

```
1    Algorithm BinSearch(a, n, x)
2    // Given an array a[1 : n] of elements in nondecreasing
3    // order, n ≥ 0, determine whether x is present, and
4    // if so, return j such that x = a[j]; else return 0.
5    {
6         low := 1; high := n;
7         while (low ≤ high) do
8         {
9              mid := ⌊(low + high)/2⌋;
10             if (x < a[mid]) then high := mid − 1;
11             else if (x > a[mid]) then low := mid + 1;
12                  else return mid;
13        }
14        return 0;
15   }
```

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.

Thanks for Your Attention!

Exercises

Exercise: 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: 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 Sis 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).