What is an Algorithm?

A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.
An algorithm is a procedure or formula for solving a problem, based on conducting a sequence of specified actions

- **Finiteness -** A finite sequence of steps
- **Definiteness -** Each step should be clearly and precisely stated
- **Input -** There should be some input presented to it
- **Termination/output -** It should terminate and produce and output
- **Correctness -** The output produced should be correct for the given input
- **Effectiveness** - All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

Expressing Algorithm
An algorithm may be expressed in a number of ways, including:

- **natural language**: usually verbose and ambiguous
- **flow charts**: avoid most (if not all) issues of ambiguity; difficult to modify w/o specialized tools; largely standardized
- **pseudo-code**: also avoids most issues of ambiguity; vaguely resembles common elements of programming languages; no particular agreement on syntax
- **programming language**: tend to require expressing low-level details that are not necessary for a high-level understanding

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.
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.
-

Disadvantages of Pseudocode

- Pseudocode does not provide a visual representation of the logic of programming.
- There are no proper format for writing the for pseudocode.
- In Pseudocode their is extra need of maintain documentation.
- In Pseudocode their is no proper standard very company follow their own standard for writing the pseudocode.

Features of a good algorithm (Wiki)

- **Precision**: a good algorithm must have a certain outlined steps. The steps should be exact enough, and not varying.
- **Uniqueness**: each step taken in the algorithm should give a definite result as stated by the writer of the algorithm. The results should not fluctuate by any means.
- **Feasibility**: the algorithm should be possible and practicable in real life. It should not be abstract or imaginary.
- **Input**: a good algorithm must be able to accept a set of defined input.
- **Output**: a good algorithm should be able to produce results as output, preferably solutions.
- **Finiteness**: the algorithm should have a stop after a certain number of instructions.
- **Generality**: the algorithm must apply to a set of defined inputs.

Another good resource is available in below link
https://courses.cs.vt.edu/cs2104/Fall14/notes/T16_Algorithms.pdf

Parameters that define how efficient an algorithm is ?

The simplest approach could be to write a code for that algorithm and log the time taken by the program to execute. But, there are certain drawbacks with this.

1. Dependency on hardware and software - It is very difficult to compare the running time of two algorithms, unless they have both been implemented using the same software and hardware.
2. Need to implement and run the algorithm on a specific hardware and software, which might not be always possible.
3. Experiments can only be done only on a limited set of inputs, Therefore, care should be taken to ensure that these are representative. Input should cover different possibilities as well. Like different set of inputs, different order of input, different size of input, etc.

Two notions that can be used to identify complexity of an algorithm.

1. **Time complexity** - Time complexity is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm
2. **Space complexity** - Space complexity is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm.

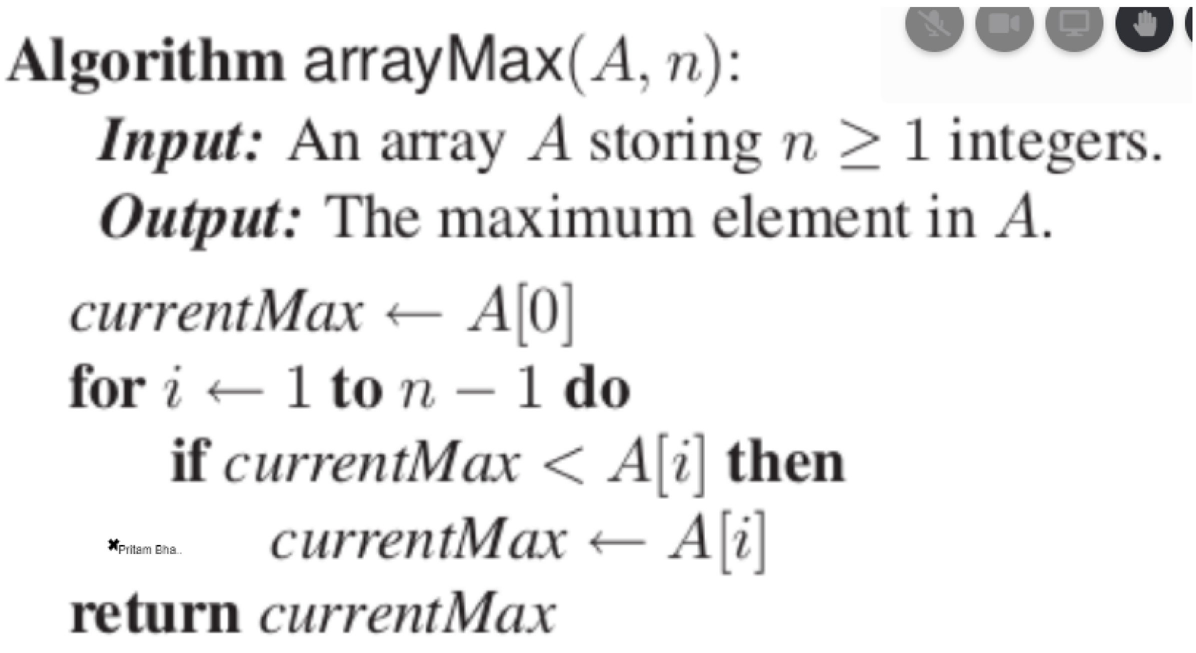
Source : https://www.cs.utexas.edu/users/djimenez/utsa/cs1723/lecture2.html

What is Data Structure ?
A data structure is a particular way to organize data in a computer, so it can be used effectively.
A data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data

values, the relationships among them, and the functions or operations that can be applied to the data.

Properties for analytic framework for evaluation algorithms

1. Takes into account all possible inputs for all algorithm
2. Allow us to evaluate the relative efficiency of any two algorithm in a manner that is independent of the hardware or software environment.
3. Can be performed on a high-level description of an algorithm without having to implement it or run experiments on it.



```
Algorithm arrayMax(A, n):
    Input: An array A storing $n \geq 1$ integers.
    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 $currentMax$
```

Analytical statement for above algorithm is
"Algorithm arrayMax runs in time proportional to N"
If we were to perform experiments, then the actual running time of arrayMax on any input of size n never exceeds c.n, where c is a constant which depends on the software and hardware environment.

Given two algorithms A and B, and A runs in time proportional to n and B runs in time proportional to N^2, we will prefer A to B, since the function n grows at a smaller rate than the function m^2.

Suppose there are two machines, A and B with c1 and c2 as 1000 and 5. Let's support on machine A we are using algorithm with complexity n and for B as complexity n^2

Then clearly
1000n > 5 * n^2 when n < 200 (say)

So even through complexity looks to be higher for machine B, but running time is better than machine A.

As the size of n increase, time taken by machine B will grow significantly, but for machine A it will grow linearly. So, running time is depending on size of input, and it is always better to choose algorithm which has less growth rate depending on N instead of actual running time. This helps in better scalability of system.

Primitive Operation

Any operation which is basic and cannot be subdivided into multiple operation is called primitive operation.
For ex:

1. **a < b + c**

There are two primitive operation

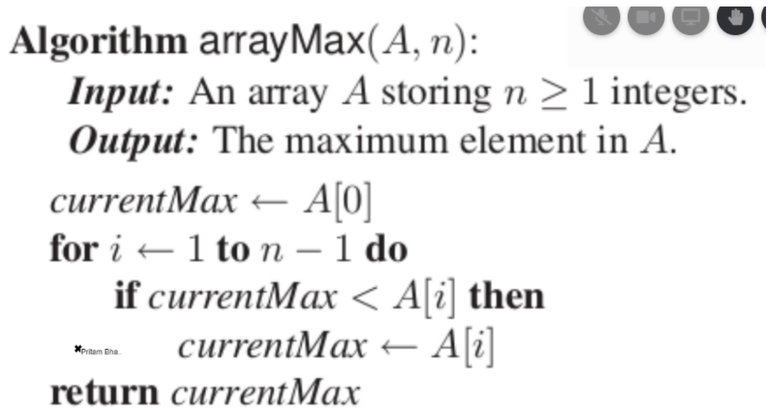
- b + c (p)
- a < p

2. **a < b + c * d**

There are three primitive operation

- c * d (p)
- b + p (sum)
- a < sum

3. **Lets take below algorithm and find primitive operations**



```
Algorithm arrayMax($A, n$):
  Input: An array $A$ storing $n \geq 1$ integers.
  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 $currentMax$
```

This algorithm can have a pseudocode as below

Operations	**Primitive operations**
currentMax = A[0]	**1**
i = 1	**1**
While (i < n)	**1, repeated n times**
If currentmax < A[i]	**1, repeated n times**
currentMax = A[i]	**1, repeated n times**
I = i + 1	**1, repeated n times**
Return currentMax	**1**

So total primitive operation will be < **4n + 3.** As above, operations are assumed for worst case.