Contents

[Algorithms 2](#_Toc66393770)

[complexity 4](#_Toc66393771)

[binary search 8](#_Toc66393773)

[Divide and Conquer 9](#_Toc66393774)

[Recursion 11](#_Toc66393775)

# Algorithms

Algorithms is a sequence of steps that we follow to reach a goal, step by step procedure for calculations, a finite list of well-defined instructions, a sequence of operations to solve a particular problem.

We can solve problems as human but human can confuse if the problem getting bigger, so for example if we ask someone for a calculation 2 + 27 it will not take much time to do that in when we think about it but if we ask for a big calculation it will take more time and many reasons, so machines did solve this problem and now they can do many things faster than us and accurate than humans. But we still have a problem how machines understand us or how it understands algorithms.

Algorithm are very useful, organising the program to do not doing useful things, it is like a magic trick they do solve mathematical problems, let us make the program very fast and it is important to get rid of pieces of inefficiency and little bits, the most important thing is how do we design our program and make it fast.

Example, if we have two persons speak different language and they use another language to understand each other so if they use English that means there’s an intermediate language between them, machines needs intermediate languages too, to understand what people need and let us talk to machine we need compiler or interpreter.

Such as any programming language (Python, C ,C++, Java… ).

Hence, any algorithm must be transferred into intermediate language, algorithms passes through two phases:

1. Solving the required problem
2. Implementation, implement the algorithm as a code.

A programming algorithms is like a method we do, to get result from the machine and be able to control or do something and it can repeat that every time in the exact way but we need to use language that computers can understand we have to know the concept of algorithms.

1. Well defined sequence of steps.
2. Mathematical instruction or rules that computers can solve
3. Algorithm is a mathematical process to solve a problem and we can make that finite number of steps not what to be done but how to be done.
4. Algorithm is a platform independent we are able to apply algorithm on any programing language

for example, “if we want to do something and it can be anything we make a cup of coffee we know how to do that first we need coffee, milk, sugar and water then we start we boil and put coffee when water is boiled we add water and then milk, so in the end we get a coffee after that we can drink it or use it” so just steps we follow to reach goals.

We can represent algorithms by using pseudo code like we type the logic of the program step by step using English words to describe everything in the program to make it clear and make sense or by drawing the flowchart of the programme, to represent the flowchart so we need to use shapes, it should be clear and understandable for human if it was not understandable or not clear that called “ambiguous” in computer lingo.

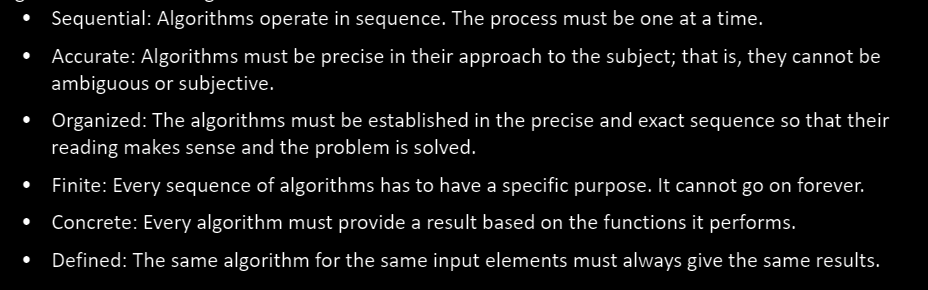
Flowcharts shapes: input, processor, output.

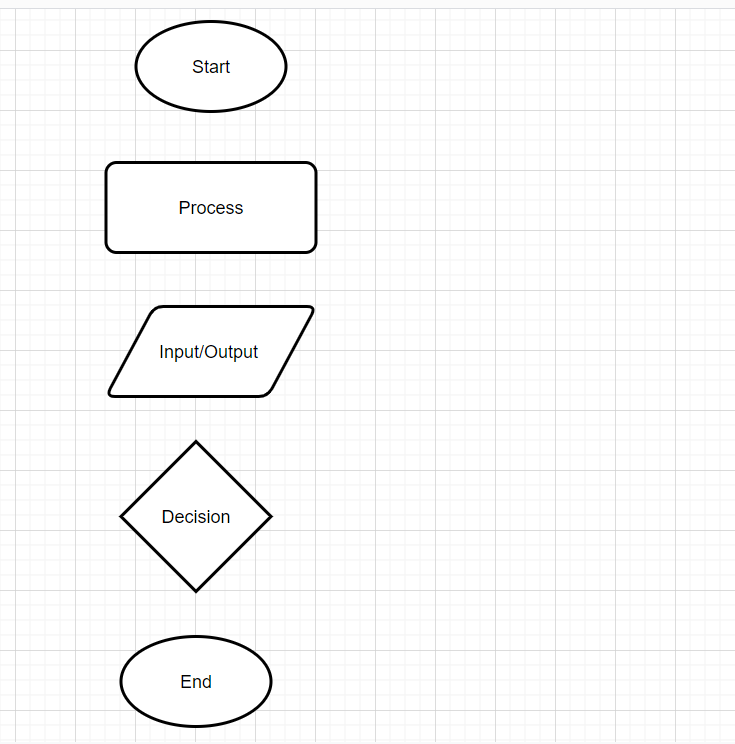
Input: the data need to work on like for example, arguments in the functions

Process: it is like the body for the function, it takes the argument to work on it with the logical operation.

Output: it is the result that function return, what it sends back after the operation.

Algorithm has some features like:





*“back in the days when phone books were without an index and we need to find a phone number from that, we need to search each page and look at names so that will take time, but if that phone book has an index and sorted in alphabetical order from A-Z we can find the phone number in less time and less effort that idea helps us to store and access data in programming.”*

That idea saving time and effort if we want to search for a name in phone book.

Python language has implicit support for data structure which enable us to store and access data like Dictionary, List, Tuple and Set some have different methods like lists have (Append, Extend) we use them to modify lists they are methods for list because it is immutable class and other methods like len() works on sequences of something but not on integer , there are addresses for the elements we call that index and it starts from 0, 1, 2, 3… or we can go from the other side -1, -2, -3… with the negative we go backwards. There are two types of data structure in Python, Built-in Data Stricture and User-Defined Data Structure.

Types of algorithms in Python

Non-computational Algorithms

Computational Algorithms

Qualitative Algorithms

Quantitative Algorithms

Computational Algorithms: the solution of the algorithms depending on the calculation, and it can be developed by computer or calculator

Non-Computational Algorithms: that cannot be solved by a computer it should be solved by a human.

Qualitative Algorithms: algorithm that does not have numerical calculation in it like (numbers), but it has logical (and/or) and formal sequence

Quantitative Algorithms: it is an algorithm that the solution depends on the mathematical calculation.

### complexity

time complexity and the space fall under the performance for the algorithm but now we focus more on the time because the machine like computers, laptops, and phones have good storage.

When we build a program it will characterize by high performance and more efficient so to make sure that the programme has high performance we ask ourselves is the program fast? or slow, and is the algorithm I am using for the program taking extra space? or not.

That means when need to reach the goal with less steps and make it from worse to good.

How much (time and space) takes to approach the goal?

we cannot compare the speed for a program by real time like 20 second or whatever, and between two machines like (computers) the reason that depending on the hardware of the machines, each machine has different hardware and speed.

The running time will be different other machine for the same program because of many factors like:

1. Single Processor or Multi-Processor 🡪 some machine have single core and some have multi cores
2. Core Speed (GHz)
3. Architecture 32/64 Bit
4. Read/Write Ram speed
5. Programming language
6. Compiler
7. Input

## time complexity

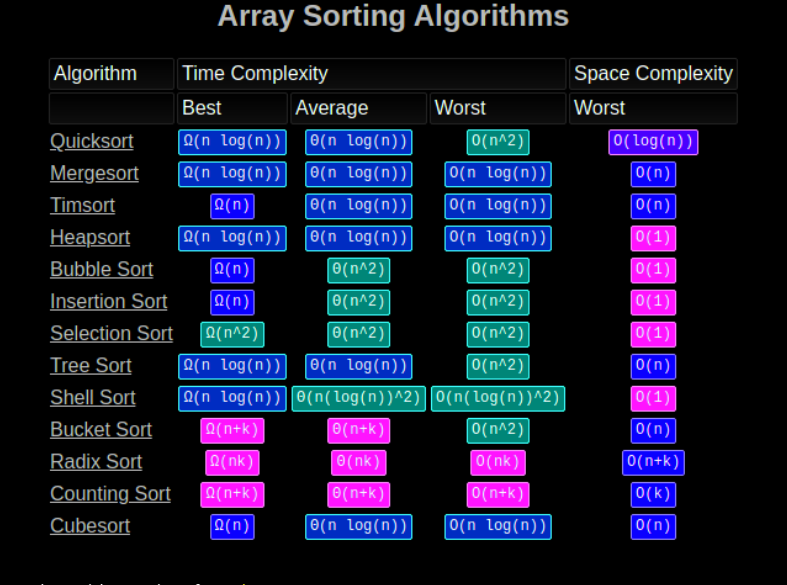
When the input of an algorithm changes the running time changes so the size of the input matters. we can clearly see that each machine has some properties not same as others and how many programs are running in the same time.

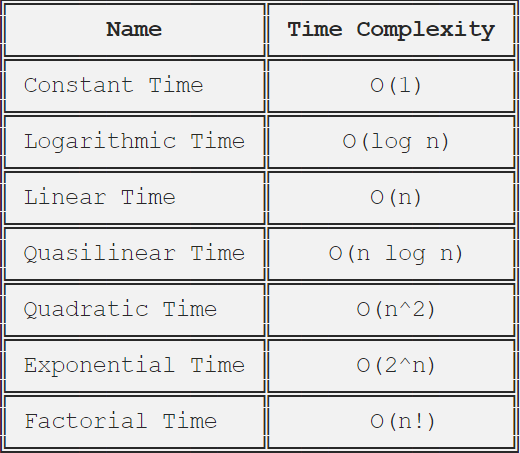
We focus more on the time complexity because now in these days we have more space in the machines we still take care of that but we look at the time complexity first.

Big O for Python dictionary is O(1) because it use the hash of the input to access the value using almost immediately

we need a method to measure the time complexity for algorithms.

If we need to search for an element in this Array 10,5,15,2,25,43 and the element was in the beginning of the array we call that best case but if it was in the middle we call that mid case or average case and if it was in the end of the array we call that worst case so to know we label them so we represent best with a symbol and we call that Omega Notation, we represent average case with different symbol and we call it Theta Notation, and we represent worst case as Big Order Notation or “O()” so we need to focus on big O because it is the worst case we work on it to make it best or average.



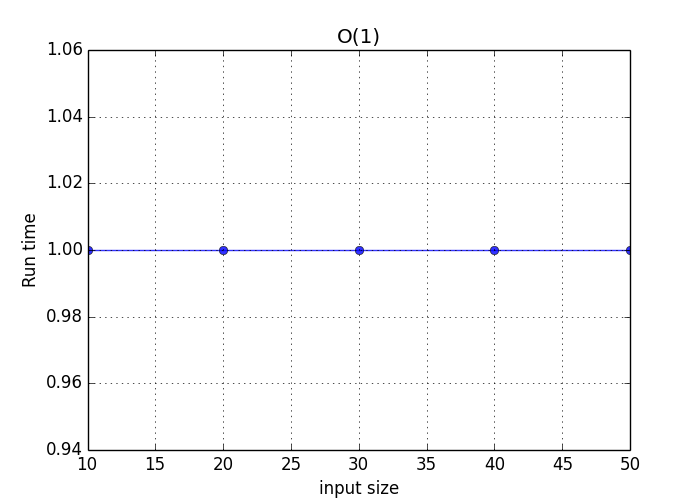


This idea will let us measure the program and let us count the steps that we did to reach the goal.

In this example below we count them as one unit of time. That means if we have a function that calculate two numbers like this a + b and return the sum of a and b.

In this case two things happened Arithmetic Operators and Assignment Operator the cost of (result =) is 1 Unit, (a + b) another 1 unit and the return result 1 Unit also so we have 3 units of time in this algorithm f(n) = 3 and that will be constant because it will not change something in the function like loop, it will do the same steps every time we run it so if the “a” or “b” changed like if a is 1000 and b is 2000 also it will take the same unit of time but different running time. When we calculate the time complexity for this function Big O(1)

We can also graph that on the Y and X axis then we can see the time complexity difference in this chart.



We can clearly see that the running time is constant it does not change, like in the picture the line is straight, even when the size of the input increases so that why we call this constant time or Big O (1) if the time complexity of this 10.

T = 10, anything multiply 1 the result will be the number itself 10

So if we use 1 instead of the number it will be O(1) instead of O(10) that is why we use Big O of 1.

We have also linear time and quadratic time we will see that down below and how to count the time complexity for them. we can use O (1), O (log n), O(n), O (n power 2)

And linear time is O(n)

The quadratic time is O(n power 2)

O(log n)

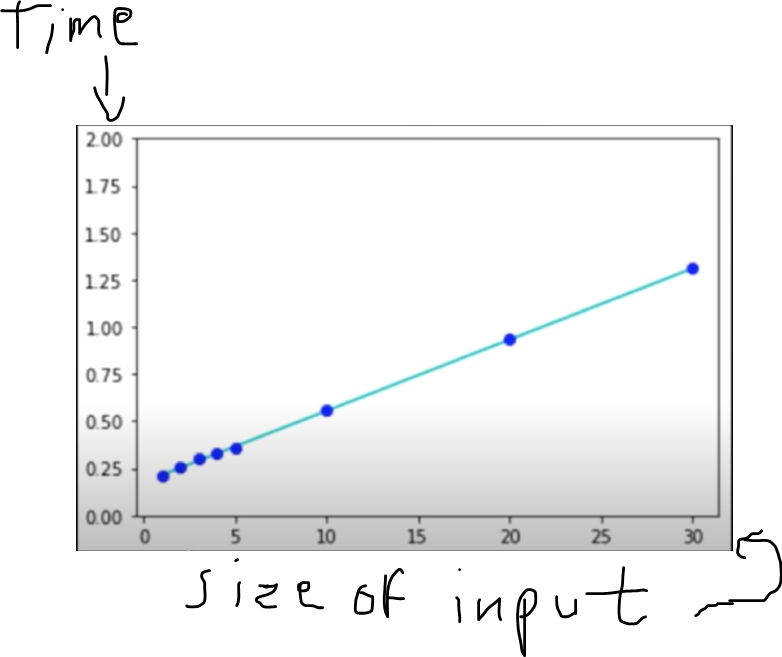
So (n) 🡪 is the number of the element in the giving array so if the length of the array is 9 so n=9

To get the time complexity and the big O of an algorithm we have two main steps to follow.

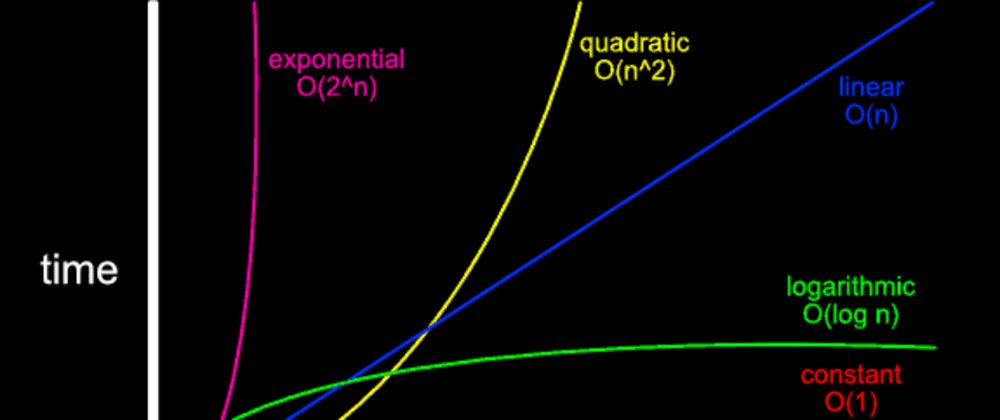
1. Find out the coefficient of the term.

2. Take out the coefficient of the term.

We call that linear time because time complexity is going up when the size of the input is bigger we can see that in the picture below so if we want to calculate the time complexity for this we would get big O(n) as we see it is increasing that means when the input is bigger the time complexity and running time go up O(n) takes more time than O(1).



Quadratic is nested loops so if we have two nested loop we call that Big O (n power 2) also we can have more than 2 nested loops and then we count them depends on how many nested loops we have, so if they were 3 we would count them as Big O(n power3) it depends how many we are using and they take more time than O(n) and O(1).



For example if we run an algorithm on new fast computer, and we got the time complexity for an that as big O (log n ) then we take the same algorithm and we run it on different computer like old version we get the same time complexity it does not change so in general no matter how fast the computer is, or what language we are using, so if we are using the same algorithm or the same steps to solve a problem we still get the same time complexity and Big O.

If we are comparing the time complexity between two functions one is O(n) and another is O(1), definitely O(1) is better to use than the rest constant is better than O(log n), or O(log n) better than O(n).

Time complexity not the only factor when we decide between multiple algorithms they solve a problem, we should consider how easy to read the code, Also the space of the program how much memory takes, code should be clear without useful things in the code.

# binary search

We have binary search so when we have a sorted list of numbers and we need to search for a specific number if the length of the list is 8 elements like this [4,9,15,21,34,67,68,91] sorted so if we want to iterate all the list we need more running time to reach the element we are looking for and time complexity will be O(n), but if I am looking for a specific number in a list has 1000 element we don’t do that iteration on all the list very time consuming. in this case I do not need to waste time on iterating each element and check that element so we already know it is sorted list we need to find the middle element and divide the list into to two groups so if we are looking for number (68) in a sorted list like this [4,9,15,21,34,67,68,91] divide it to two groups we will have something like this [4,9,15,21, 34,67,68,91] definitely it won’t be under the 21 that in the first group, it will be in the second half this is first step we repeat the same method to the rest of the list so now we work on the second half we divide that to two parts so the half of that 34,67, 68,91 so we clearly number (68) not in the first group because it is greater than 67 so it will be in the last group in the list then we divide that again if we need to until we get that in a very small group this is called binary search.

Here in binary search the first step was iteration [4,9,15,21,34,67,68,91] 1 = n/2 [4,9,15,21, 34,67,68,91] when we did divide all the list into two groups then in the second step we were looking for number in the second group so we divide the iteration [4,9,15,21, 34,67, 68,91] 2 = (n/2)/2 and then we divided that into two groups the last iteration 3 [4,9,15,21,34,67,68 ,91] = ((n/2)/2) /2 we compare that with the number we are looking for and we got that in the first group so we did reach the goal in 3 iterations so if we have to do that on big input we will do use this logic (k = n/2 base k) also we can convert that into Big O. iteration k = n/2 base k 🡪 1 = n/2 base k 🡪 n = 2 base k (log2 n = log2 2k 🡪 log2 n = k \* log2 2) 🡪 k = log n 🡪 O(log n )

In that example above we had 8 elements we can apply order log n like this k= O (log n) 🡪 log2 8 🡪 log2 2 power 3 🡪 3 \* log2 base 2 🡪 3 iterations

# Divide and Conquer

divide and conquer means divide all the problem into several small problems and work on each group to get it solved then we combine them again and try to get them back as one thing solved for example if we have a list of numbers that is not sorted to sort that we can do this idea to help us like we divide the list into smaller pieces as much as we can and we work on each group by itself then when we solve the groups we start combine them again until we get the list sorted and all the numbers in it.

For example: if we have a list has 9 numbers in it like this 7,8,9,5,3,6,2,1

And the question is to sort this list from smaller to greater.

First step we take the list and divide it into smaller groups like this 7,8, 9,5, 3,6, 4,2, 1 so the last element left by itself as we see because the length is odd numbers 9.

Then we start working on each group to tidy them up so the smaller should be on the left side. Now we have a list like this after we done the first step 🡪 7,8, 5,9 ,3,6, 2,4, 1

Then we move on to the next step which divides the list into bigger groups in this case we could make the list 2 groups and it will look like this but the last element which is 1 left by itself again

🡪 7,8,5,9 ,3,6,2,4, 1

Now we have two groups as we see so we need to take the first group and compare them first 2 elements in the group with the other elements in the same group like this is 7,8,5,9 ,3,6,2,4, 1 = 🡪 is 7 < 5? If yes, we keep them if not we swap them and is 8<9? If yes we keep them if not we swap, again is 7 < 9? And is 8 < 5?

Then we do the same thing to the other group in the list to sort it.

Now we have a list like this 5,7,8,9,2,3,4,6,1 now we have sorted only 2 groups and last element (1) so we have 3 groups now like this 5,7,8,9 , 2,3,4,6, 1

Then next step is to take first and the second group to compare the numbers inside the groups with each other like this is 2 < 5? If yes, we swap, is 3 < 7? Is 4 < 8? And so on until we compare all the element in the first group with the elements in the second group and we swap them so after we finish that we will have the list in this format 2,3,4,5,6,7,8,9,1 we left with two groups one is almost sorted and the last group in the list which is number 1.

Now we have to compare we have to compare the first number in the first group with the element in the second group and swap them if needs so clearly we can see that the first element is 2 and that element in the second group is number 1 so we ask is number 2 less than 1? It in not so we swap number one and so on…

In divide and conquer there are 3 main steps:

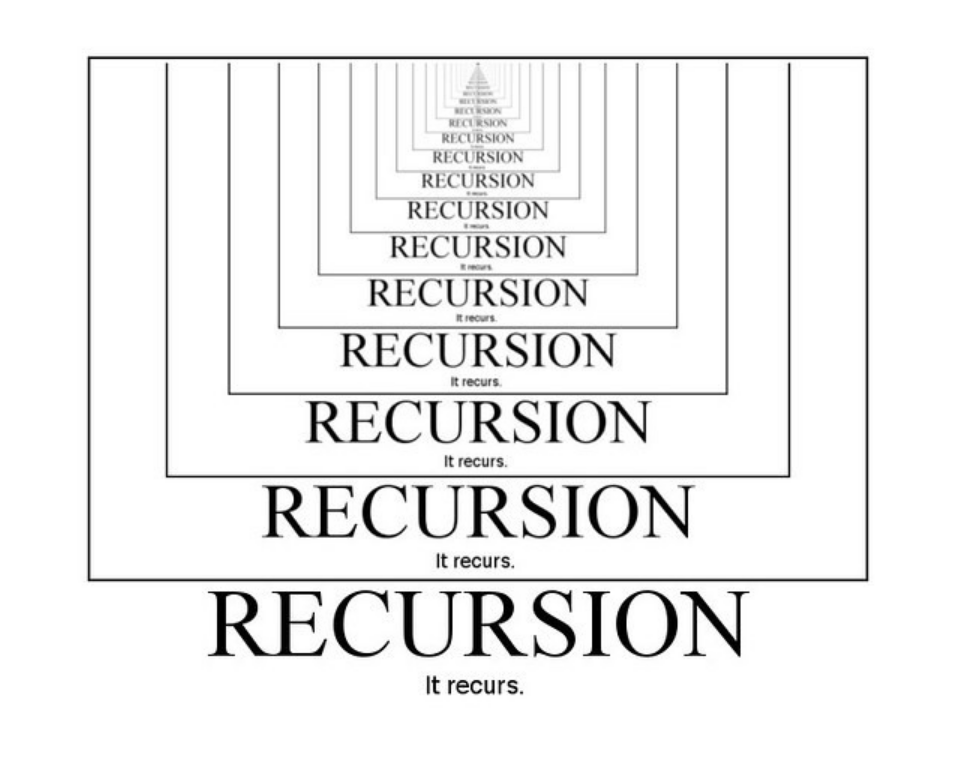
1. Divide 🡪 to divide the problem into smaller groups
2. Conquer 🡪 the steps where we compare the elements with each other
3. Combine 🡪 where we combine groups again together

As we did in the example above we need take the all problem and divide that to smaller groups to work on each group by itself to sort them or to swap the position if we need after that we combine the whole list again then we divide it back to small group but bigger than the groups in the first step and work on them to see if we need to swap the positions of the elements so we sort them then we keep doing that to get the list as one sorted group.

# Recursion

Recursion is a way to solve a problem, by having a function calling itself, repeating the process in a self-similar way, recursion is the process of defining a problem but when function calls itself, this algorithm is powerful tool, it is dynamic programming it is basically way of making the algorithm more efficient by storing intermediate result and it works well when the algorithm has many repetitive computations so that we do not have to repeat those computations over and over again,it comes directly from mathematics there are a lot of problems can be solved using recursion like Fibonacci, Factorial and Towers of Hanoi (TOH), we have to have a decision to stop that from running or when we reach the goal or in certain time, and there is limit for how many times it runs for example if we want to make a recursion function in Python to get factorial for a number it will reach zero and goes to the negative position and keep going down but we do not want that, we need something to stop that when it reach zero like if statement.

Before recursion method, they used to write the code in non-recursive or when it was not allowed function called itself, they used to look for a way to write the code using loops, and they would use more than a function to solve the problem way and that we can solve a problem using different way like non recursive or iterative.



It is like when we look into two mirrors opposite to each other one goes inside the other one so if we want to make a recursion function with Python language we need an if statement to return 1 when n is equal to one or zero then it goes in the opposite direction, it would be like creating copies of itself

