**Algorithm**

* An algorithm can be defined as a well-defined computational procedure that takes some values, or the set of values, as an input and produces some value, or the set of values, as an output.
* It describes specific computational procedures for achieving the input-output relationship.
* **For Example**, We need to sort the sequence of number into ascending order. Here is how we define the sorting problem.

**Input:**

A sequence of n number (a1, a2,......an)

**Output:**

A permutation (reordering) (a'1, a'2 .... a'n) of the input sequence such that (a'1 ≤ a'2 ≤ ....≤ a'n)

# Need of Algorithm

1. To understand the basic idea of the problem.

2. To find an approach to solve the problem.

3. To improve the efficiency of existing techniques.

4. To understand the basic principles of designing the algorithms.

5. To compare the performance of the algorithm with respect to other techniques.

6. It is the best method of description without describing the implementation detail.

7. The Algorithm gives a clear description of requirements and goal of the problem to the designer.

8. A good design can produce a good solution.

9. To understand the flow of the problem.

10. To measure the behaviour (or performance) of the methods in all cases (best cases, worst cases, average cases)

11. With the help of an algorithm, we can also identify the resources (memory, input-output) cycles required by the algorithm.

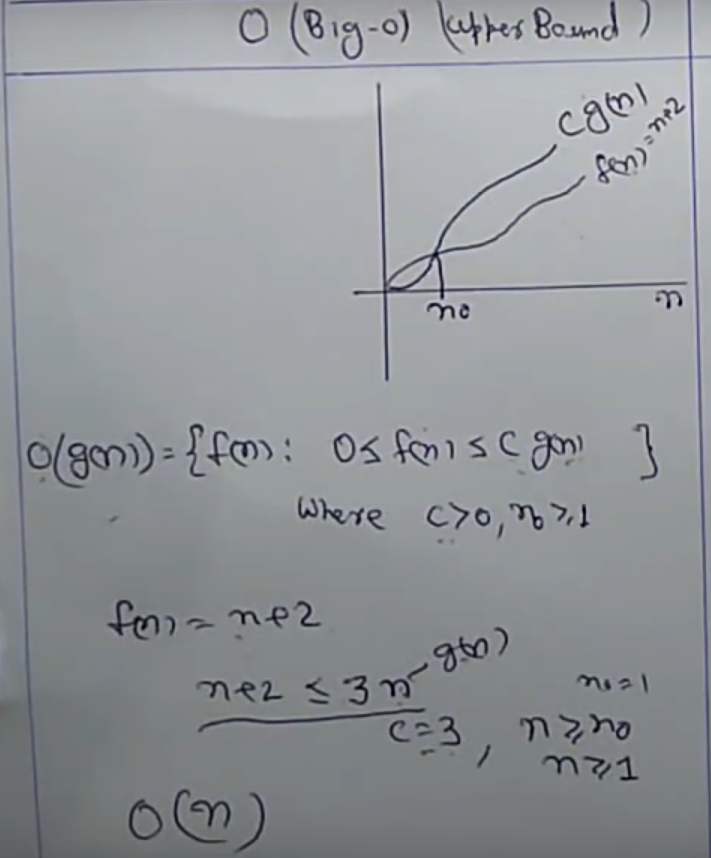
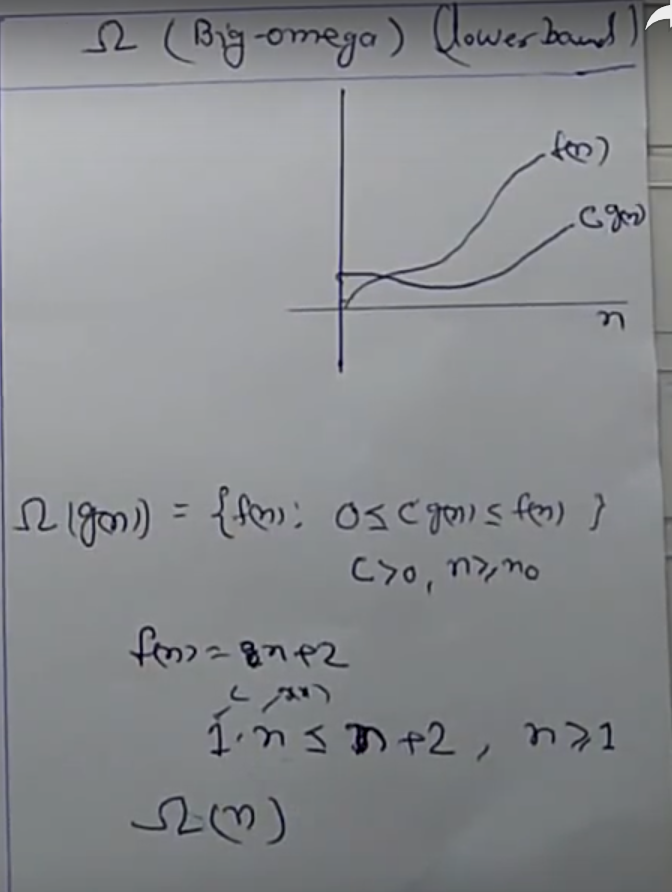
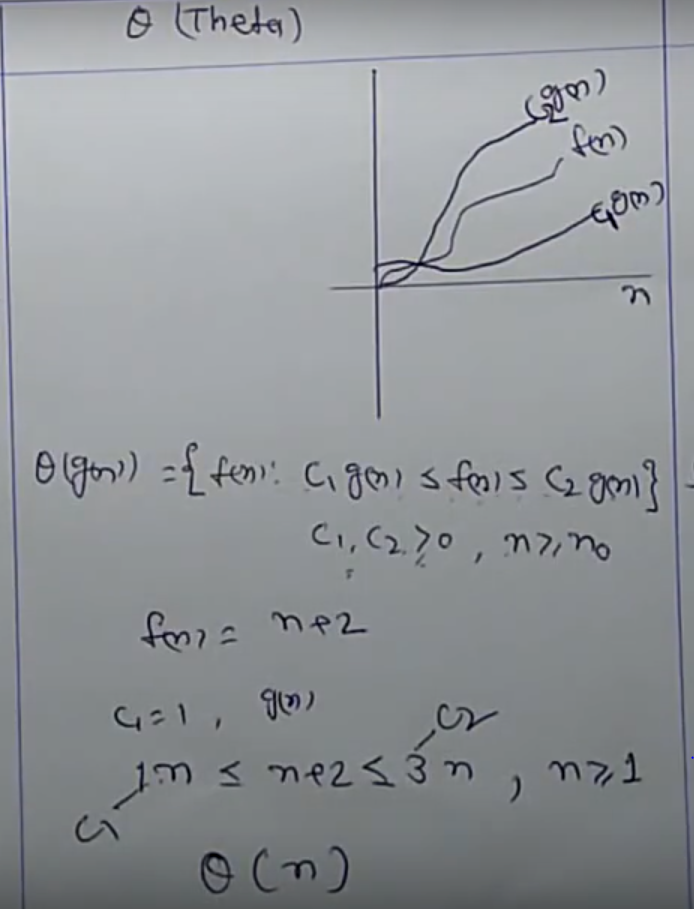
12. With the help of algorithm, we convert art into a science.

13. To understand the principle of designing.

14. We can measure and analyze the complexity (time and space) of the problems concerning input size without implementing and running it; it will reduce the cost of design.

# Mathematical foundation – Growth functions

Sources:

1. PPT
2. 
3. 
4. 
5. https://www.youtube.com/watch?v=EL9T1ngiCqA

# Complexity analysis of algorithm

Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance different algorithms and choose the best one to solve a particular problem. While analysing an algorithm, we mostly consider time complexity and space complexity.

1. Time complexity: Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. (NOTES + <https://www.youtube.com/watch?v=N2BQMAWKEHk> , https://www.youtube.com/watch?v=gCsfk2ei2R8)
2. Space complexity: Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Time and space complexity depends on lots of things like hardware, operating system, processors, etc. However, we don't consider any of these factors while analyzing the algorithm. We will only consider the execution time of an algorithm.

POINTS TO REMEMBER

|  |  |
| --- | --- |
| ALGORITHM | PROGRAM |
| 1. Algorithm is written in “Design phase” of a problem solution. | 1. Program is written in the “Implementation phase” of the problem solution. |
| 1. Anyone who has Domain Knowledge can write the algorithm. | 2. Programmer creates it. |
| 1. Any language can be used to create it | 3. Only Programming language can be used. |
| 1. Independent of H/W and OS. | 4. Dependent of H/W and OS. |
| 1. After creation, we analyse the algo in terms of time and space complexity to know its effectiveness. | 5. After creation, we test the program by executing it using compiler and its required virtual environment. |
| 1. In this, “Priori Analysis” is done which incudes time and space functions that calculate the time and space that implementation of the algo will require. | 6. “Posteriori Analysis” is done case of programs where we watch the actual time and bytes used in the execution of the program. |

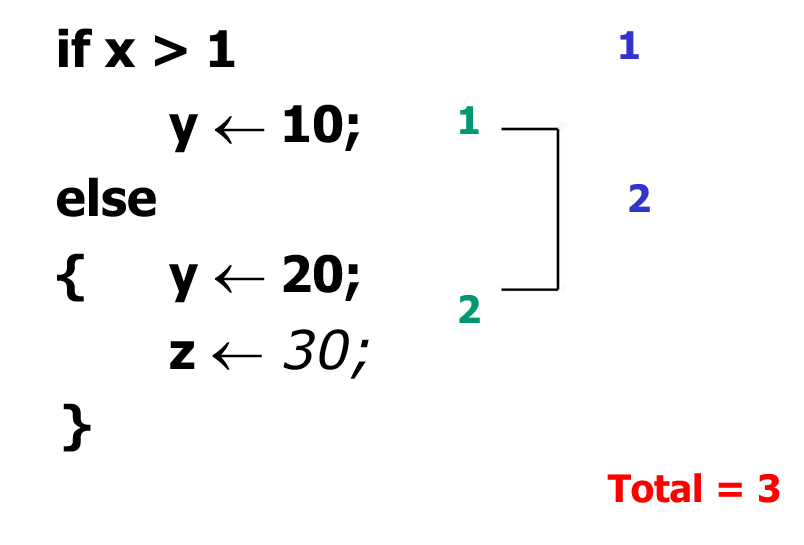
1. Characteristics of algorithm:
2. Input
3. Output
4. Definiteness (clear statements)
5. Finiteness (must terminate at some point--- limited duration.).
6. Effectiveness (In terms of time and space complexity )
7. How to analyse an algorithm?
8. Time
9. Space
10. Network (network consumption or cloud-based applications)
11. Power consumption
12. CPU Registers (for device driver programs)
13. Frequency count method: Number of steps or statements needed by the algo to finish.
14. a = 10 (count = 1)
15. b = 2\*a (count = 1)
16. if (<condition>) --(count = 1)

<s1>;

Else

<s2>;

Sum = 1 + Max{count(s1), count(s2)}

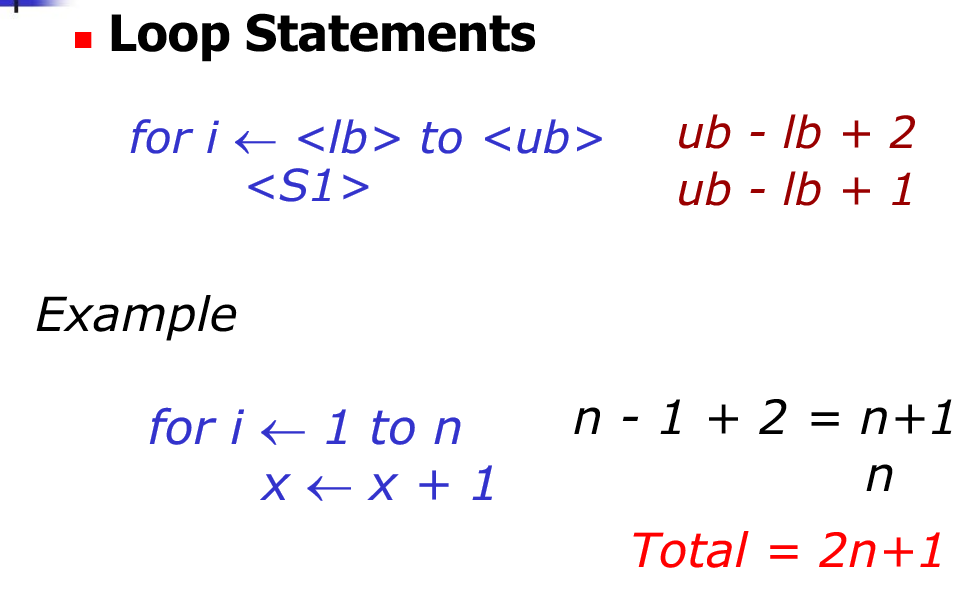


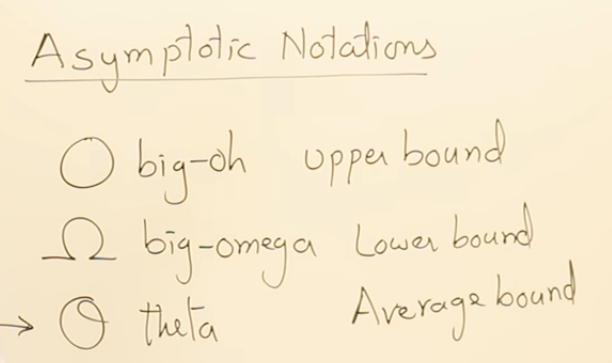
1. loop statement :

upper\_bound – lower\_bound + 2

inner statement :

upper\_bound – lower\_bound + 1



1. 
2. 