1. **Distinguish between Deterministic and non-deterministic algorithms.**

| **Deterministic Algorithm** | **Non-deterministic Algorithm** |
| --- | --- |
| A deterministic algorithm is one whose behavior is completely determined by its inputs and the sequence of its instructions. | A non-deterministic algorithm is one in which the outcome cannot be predicted with certainty, even if the inputs are known. |
| For a particular input, the computer will give always the same output. | For a particular input the computer will give different outputs on different execution. |
| Can solve the problem in polynomial time. | Can’t solve the problem in polynomial time. |
| Can determine the next step of execution. | Cannot determine the next step of execution due to more than one path the algorithm can take. |
| Operation are uniquely defined. | Operation are not uniquely defined. |
| Like linear search and binary search | like 0/1 knapsack problem. |
| Deterministic algorithms usually have a well-defined worst-case time complexity. | Time complexity of non-deterministic algorithms is often described in terms of expected running time. |
| Deterministic algorithms are entirely predictable and always produce the same output for the same input. | Non-deterministic algorithms may produce different outputs for the same input due to random events or other factors. |
| Deterministic algorithms usually provide precise solutions to problems. | non-deterministic algorithms often provide approximate solutions to the problems. |
| Deterministic algorithms are commonly used in applications where precision is critical, such as in cryptography, numerical analysis, and computer graphics. | Non-deterministic algorithms are often used in applications where finding an exact solution is difficult or impractical, such as in artificial intelligence, machine learning, and optimization problems. |
| Examples of deterministic algorithms include sorting algorithms like bubble sort, insertion sort, and selection sort, as well as many numerical algorithms. | Examples of non-deterministic algorithms include probabilistic algorithms like Monte Carlo methods, genetic algorithms, and simulated annealing. |

1. **What are differences between NP-Hard and NP-Complete classes? Explain with examples.**

**NP Problem:**   
The NP problems set of problems whose solutions are hard to find but easy to verify and are solved by [Non-Deterministic Machine](https://www.geeksforgeeks.org/difference-between-deterministic-and-non-deterministic-algorithms/) in polynomial time.

[**NP-Hard Problem**](https://www.geeksforgeeks.org/tag/nphard/)**:**   
A Problem X is NP-Hard if there is an NP-Complete problem Y, such that Y is reducible to X in polynomial time. NP-Hard problems are as hard as NP-Complete problems. NP-Hard Problem need not be in NP class.

If every problem of NP can be polynomial time reduced to it called as NP Hard.

A lot of times takes the particular problem solve and reducing different problems.

**example :**

1. Hamiltonian cycle .
2. optimization problem .
3. Shortest path

[**NP-Complete Problem**](https://www.geeksforgeeks.org/algorithms-gq/np-complete-gq/)**:**

A problem X is NP-Complete if there is an NP problem Y, such that Y is reducible to X in polynomial time. NP-Complete problems are as hard as NP problems. A problem is NP-Complete if it is a part of both NP and NP-Hard Problem. A non-deterministic  Turing machine can solve NP-Complete problem in polynomial time.

A problem is np-complete when it is both np and np hard combines together.

this means np complete problems can be verified in polynomial time.

**Example**:

1. Decision problems.
2. Regular graphs.

| **NP-hard** | **NP-Complete** |
| --- | --- |
| NP-Hard problems(say X) can be solved if and only if there is a NP-Complete problem(say Y) that can be reducible into X in polynomial time. | NP-Complete problems can be solved by a non-deterministic Algorithm/Turing Machine in polynomial time. |
| To solve this problem, it do not have to be in NP . | To solve this problem, it must be both NP and NP-hard problems. |
| Time is unknown in NP-Hard. | Time is known as it is fixed in NP-Hard. |
| NP-hard is not a decision problem. | NP-Complete is exclusively a decision problem. |
| Not all NP-hard problems are NP-complete. | All NP-complete problems are NP-hard |
| Do not have to be a Decision problem. | It is exclusively a Decision problem. |
| It is optimization problem used. | It is Decision problem used. |
| Example: Halting problem, Vertex cover problem, etc. | Example: Determine whether a graph has a Hamiltonian cycle, Determine whether a Boolean formula is satisfiable or not, Circuit-satisfiability problem, etc. |