**Types of Problems:**

**Decision problems** are those which give the solution of the problem in the form of yes or no. E.g. existence of Hamiltonian cycle in a given graph is a decision problem or graph has a spanning tree or not.

**Optimization problems** are those in which we must maximize or minimize something, and which give a numeric value as answer. E.g., travelling salesperson problem is an optimization problem, minimum cost spanning tree. Corresponding to every optimization problem there must exist a decision problem.

**Polynomial time** algorithms are those whose time complexity is polynomial time i.e., O(nk)

**Non polynomial time** algorithms are those whose time complexity is exponential time i.e., O(an).

**Deterministic algorithms** are those in which the result of every statement is uniquely defined.

* A deterministic algorithm is one whose behavior is entirely predictable and consistent.
* When provided with the same input under the same conditions, a deterministic algorithm will produce the same output every time.
* Deterministic algorithms are used in a wide range of applications and are generally easier to design, analyze, and debug.
* Common examples of deterministic algorithms include sorting algorithms like quicksort and merge sort, as well as basic arithmetic operations.

**Non-Deterministic algorithms** are those in which result of every statement is defined in terms of set containing some elements.

* A non-deterministic algorithm is one whose behavior is not entirely predictable and can exhibit different outcomes for the same input.
* Non-deterministic algorithms are often used in situations where finding an exact solution is computationally infeasible or would take an impractical amount of time.
* Non-deterministic algorithms may involve randomness, making use of random number generators or probabilistic choices.
* Examples of non-deterministic algorithms include certain optimization algorithms like simulated annealing and genetic algorithms, as well as some machine learning techniques.

It's important to note that non-deterministic algorithms are not necessarily random in the sense of producing completely arbitrary results. Instead, they introduce controlled randomness or use heuristics to explore solution spaces efficiently. These algorithms are commonly used in complex optimization problems, search problems, and situations where finding an optimal solution may be challenging using deterministic methods.

The key distinction is that randomized algorithms use randomness to improve efficiency or provide probabilistic solutions, whereas non-deterministic algorithms allow multiple choices or paths as part of their decision-making process, aiming to explore a space of possibilities efficiently.

**Classes of Problems:**

**Class P** is collection of those **Decision problems** which are solvable in **Polynomial time** (meaning that the algorithm's running time is bounded by a polynomial function of the input size. ) through **a Deterministic algorithm**.

Examples of problems in P **include sorting, searching, and basic arithmetic operations**.

**Class NP** is a collection of those **Decision problems** which are solvable in **Polynomial time** through a **Non-Deterministic or Deterministic algorithm**. E.g., Problem: **Decide if a graph has a minimum spanning tree of weight at most K**.

Algorithm:

1. non-deterministically choose a set T of n-1 edges

2. Test that T forms a spanning tree

3. Test that T has weight at most K

Analysis: Testing takes O(n+m) time, so this algorithm runs in polynomial time. Here n is number of vertices and m is number of edges.

Are the classes *P* and *NP* identical? This is an open problem. It may well be the biggest open problem of mathematics at the beginning of the 21st century. It is not hard to show that every problem in *P* is also in *NP,* but it is unclear whether every problem in *NP* is also in *P.* The best we can say is that thousands of computer scientists have been unsuccessful for decades to design polynomial-time algorithms for some problems in the class *NP.* This constitutes overwhelming *empirical* evidence that the classes *P* and *NP* are indeed distinct, but no formal mathematical proof of this fact is known.

**Polynomial-time reducibility**

Let *E* and *D* be two decision problems. We say that *D* is *polynomial-time reducible to E* if there exists an algorithm *A* such that.

* *A* takes instances of *D* as inputs andalways outputs the correct answer “Yes” or “No” for each instance of *D.*
* *A* uses as a subroutine a hypothetical algorithm *B* for solving *E.*
* There exists a polynomial *p* such that for every instance of *D* of size *n* the algorithm *A* terminates in at most *p(n)* steps *if each call of the subroutine B is counted as only m steps, where m is the size of the actual input of B.*

E.g. The Hamiltonian cycle problem is polynomial time reducible to the decision version of TSP.

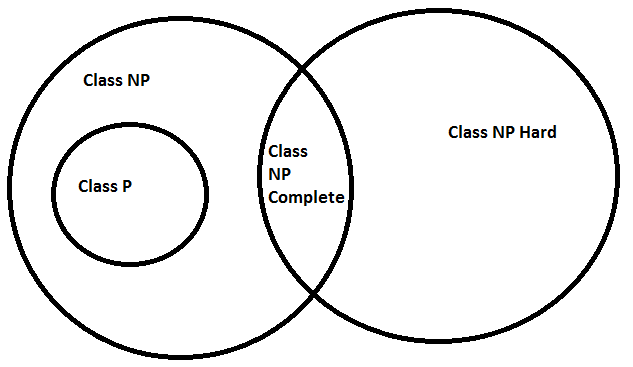
Given an instance *G* with vertices *v1 , … , vn* of the Hamiltonian cycle problem, let *H* be the weighted

complete graph on *v1 , … , vn* such that the weight of an edge {*vi , vj* } in *H* is 1 if {*vi , vj* } is an edge in *G*, and is 2 otherwise. Now the correct answer for the instance *G* of the Hamiltonian cycle problem can be obtained by running an algorithm on the instance *(H,n+1)* of the TSP.

**Class NP complete** is collection of those **Decision problems** which are solvable in **Polynomial time** through a **non-deterministic algorithm**. **The Hamiltonian cycle problem, the decision versions of the TSP and the graph coloring problem**, as well as literally hundreds of other problems are known to be NP-complete.

The first problem proven to be NP-complete was the Boolean Satisfiability Problem (SAT).

**Class NP Hard** is a collection of those **Decision or Optimization** problems which are solvable in **polynomial or non-polynomial time** through a **non-deterministic algorithm**. It can also be defined as optimization problems whose decision version is NP Complete are called NP Hard. **TSP is an example of NP hard problem**.



Relationship between the various classes of problems