What is a Hard Problem?

1. It is hard to find a solution.
2. It is hard to find an algorithm to solve the problem.

A problem may be hard for the human but "**easy**" for the computer.
Find n! say n=5000
A problem may be easy for the human but "**hard**" for the computer.
Find intersection of
1. x+2y=5, and 2x+y=3 in 3D? an infinite line parallel to z-axis through (1,2)
2. $x^2 + y^2 + z^2 = 25$ and $z=0$

Note: Example: $n = 10^6$ $A(n)=2^{n/10\text{^}6}$ $B(n) = n^{10}$
Up to a problem of size $n = 10^6$ A is better $A(10^6)=2$, $B(10^6)= 10^{60}$
but **asymptotically** B is better.

In Computer science we are always looking for faster solutions, once we have on solution.
Why do we want to know if the problem is hard or easy?
Suppose you want to find an efficient solution (polynomial time) to a problem. You may be spending time and come up with exponential time solution but not with polynomial time solution. This is not what you want, i.e. you want efficient solution. Instead of investing additional efforts, if you can prove that it is an NP-Complete problem, you can convince yourself that there is no efficient solution, and save yourself from wasting lot of time in searching for an efficient solution (which does not exist)..

We will make the meaning of easy and hard clear now.

What is a Hard problem in Computer Science?

An algorithm is "**hard**", it is of exponential order e.g. $\Omega(a^n)$ where $a > 1$.

In **hard** problems, changing the input n to 2n, the time complexity changes by a much larger scale (a function of n), e.g. 2^n to $2^{2n} = 2^n \bullet 2^n$

In Computer Science, a problem is **efficient/tractable** if it has polynomial time solution, otherwise **inefficient/hard/intractable**.

There three types of problems

Problems it can be proved to be polynomial time
P is the class of decision problems that can be solved in **polynomial** time.

Normally, we will say sorting algorithm of order $O(n^2)$ is inefficient but of order O(n log n) is efficient.
easy - Sorting n elements $O(n^2)$.
easy – SSSP and MST $O(n^2)$, $n=|E|$.

easy – NetWorkFlow $O(n^3)$, n=|E|.

We have seen the **SSSP** and **MinCostSpanningTree** (O(|E| log |V|)) or O(|E||V|)are polynomial time algorithms. We can generate Brute force non-polynomial time algorithms also. That is, find all simple circuits and select one. For n vertices, there are (n-1)! possibilities. Exponential order.

In **easy** problems, changing the input n to 2n, the time complexity changes by a constant factor, e.g. from n^2 to $4 \bullet n^2$

In other words, with a four times as fast computer, in this easy problem (sorting problem), we can increase the size of problem to double the size in order to solve it in the same amount of time. Thus, we can do $4n^2$ steps instead of n^2 steps in the same amount of time.

Problems it can be proved to be not polynomial time

hard - Tower of Hanoi Problem $\Omega(2^n)$

Where as, in this (Tower of Hanoi) **hard** problem, with four times as fast computer, we cannot increase the size of problem to double the size in order to solve it in the same amount of time. That is, we can increase the size of the problem by just a fraction, e.g. instead of 2^n steps, we can now do $4 \bullet 2^n = 2^{n+2}$ which means n can be increased to n+2 in stead of 2n.

An algorithm that produces an answer by making decisions based on the available information is called **deterministic** algorithm.

NP

An algorithm that checks all possible solutions in parallel to determine a correct solution is called a **non-deterministic** algorithm.

This means, an algorithm that **guesses** a correct solution and checks that the **guess** is in fact a solution is called a **non-deterministic** algorithm.

NP-is the class of problems whose non-deterministic solution can be verified in polynomial time.

NP stands for **N**on deterministic **P**olynomial time algorithm.

Guesss and verify in polynomial time.

What is non-deterministic (Hit and Trial) algorithm?

Hit and Trial method In layman's language, a non-deterministic algorithm solves a problem by guessing a solution, then verifying its correctness. If the answer is false, it tries another one and tests it. It keeps trying until all possibilities are exhausted.

Note. A Parallel computer can test all possible solutions in parallel, in the same amount of time whereas the standard (normal, sequential) computer will take time depending on the multiplicity of possible solutions.

Finding the permutations of n numbers is of complexity $O(n!) \approx O(n^n)$

However, a given a rearrangement of n numbers, we can determine if it is a permutation of n numbers in O(n lg n) or ($O(n^2)$ gross polynomial estimate) time. That is, it is an algorithm of polynomial time.

Remember Finding the solution to Tower of Hanoi problem is $O(2^n)$. Given a sequence of steps, to verify that it is a solution is not of polynomial time. It is not in **NP**.

Open question **P ?= NP**
A problem X is polynomially reducible to Y if every instance X is poynomially reducible to some instance of Y.
NP-hard : A problem X is NP-hard if every problem in NP is polynomially reducible to X.
Hamilton Circuits and TSP are NP-hard.
Circuit is a simple closed path containing every vertex.
Hamilton Circuits: Given a graph, find a circuit in it.
If every vertex is of even degree, it has Hamilton circuit, not conversely.
TSP find a circuit of min cost.

Problems it can't be proved to be polynomial or not polynomial time NP-Complete
NP-Complete- Traveling Salesman Problem $\Omega(1.26^n)$
KnapSack $\Omega(2^n)$ Dynamic Programming O(nk) if k is small.
Hamilton Circuit $\Omega(n!)$

NP contains P and NPC

A problem can be formulated as an **optimization** problem or **decision** problem.

Is there a shortest path of length k or find the shortest path

What is a Decision Problem?
Any problem which can be formulated as **yes/no** problem is a decision problem.
Examples
Does there exist a simple path in G through all the vertices of G, is a **decision** (**yes/no**) problem.

Does there exist a spanning tree in G with cost less than K is a **decision** (**yes/no**) problem.

What is an Optimization Problem?

A problem which is associated with a value, we want to optimize the that value.
Find SSSP and MST is an optimization problem.
Find a simple path in G through all the vertices of G, is **not** a **decision** (**yes/no**) problem.

Note. If one cannot get the yes/no answer to checking a solution to a given problem in polynomial time, it can not be solved in polynomial time.

What is NP-Complete?
NP-Complete problems are exponential time problems that are verifiable in polynomial time.

NP-Completeness is really a verifiability problem.
How do we check if it is NP-Complete?

NP-Complete if NP and NP-hard

A problem that can be **solved** with **non-deterministic** algorithm in polynomial time is called an **NP-complete** problem.

Example. **Tower of Hanoi** problem is not NP-Complete.
Traveling salesman problem is NP-Complete.
KnapSack, Hamiltonian Circuit are NP-Complete.

What is traveling salesman problem?

Given a complete, weighted, directed graph, find the shortest, simple, cycle which contains every vertex. (**Hamiltonian cycle**)
or
Given a complete, weighted, directed graph, and an integer K, determine whether there is a simple cycle with length $\leq$ K containing every vertex. (**yes or no** problem, **decision** problem)
In a graph with |E| edges, the |E|! paths to check, cannot be done in polynomial time in |E|.

Formulate the problem as a true/false problem. If the **yes** part can be answered in polynomial time **correctly**, then it an **NP-Complete** problem.

Remember **P ⊆ NP**

There are many NP-Complete problems. The subject is wide open to research. Researchers have been working for the 40 years to prove whether **P = NP** or **P ≠ NP** with no proof in sight. None one has ever proved that there is a problem in **NP** but not in **P** . **Every year, proofs are published with errors and withdrawn later on.**

Thomas H Corman, Charles E Leiserson, Ronald L. Rivest, Clifford Stein.etc.

Show that TSP is NP-Complete

1. Show it is in NP. That is it is verifiable in polynomial time. Is it a cycle? Find the sum of costs. Is the cost less than or equal to K.
2. Show it is NP-hard. Show it is equivalent to Hamilton circuit problem. Start with any instance of Hamilton cycle problem, G=(V,E), Construct an instance of TSP:

Create a complete graph with no self loops E' = {(i,j): i,j in V, i≠j}, define cost function c(i,j)= 0, (i,j) in E; 1 otherwise.
G' is an instance of TSP find a simple closed path with min cost.
Now G has a Hamilton cycle if and only if G' had Travelling Salesman Path of min cost.
Suppose h a Hamilton cycle in G, clearly then h is cycle in G' and min cost zero.
Suppose h' is TSP in G', min cost is zero, this means edges are in G, h is a Hamilton cycle.

Halting problem, Conjunctive Normal Form Problem.