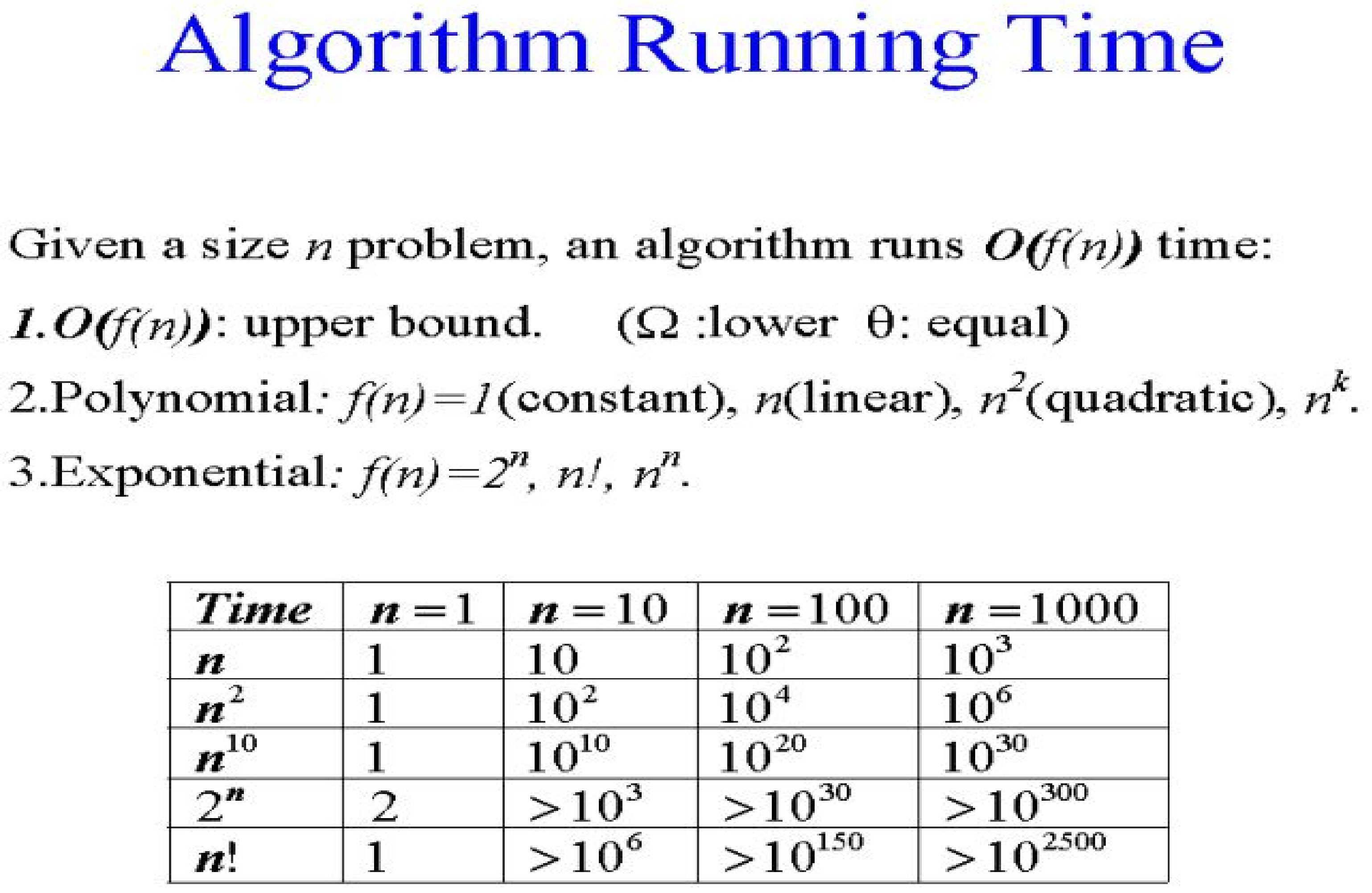
NP –HARD AND NP – COMPLETE PROBLEMS
Basic Concepts



Algorithm Running Time

Given a size n problem, an algorithm runs $O(f(n))$ time:

1.O(f(n)): upper bound. (Ω :lower θ: equal)

2.Polynomial: $f(n)=1$(constant), n(linear), n^2(quadratic), n^k.

3.Exponential: $f(n)=2^n$, $n!$, n^n.

Time	$n=1$	$n=10$	$n=100$	$n=1000$
n	1	10	10^2	10^3
n^2	1	10^2	10^4	10^6
n^{10}	1	10^{10}	10^{20}	10^{30}
2^n	2	$>10^3$	$>10^{30}$	$>10^{300}$
$n!$	1	$>10^6$	$>10^{150}$	$>10^{2500}$

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Conepts

Decision Problems

To keep things simple, we will mainly concern ourselves with decision problems. These problems only require a single bit output: ``yes'' and ``no''.

How would you solve the following decision problems?

- -Is this directed graph acyclic?
- -Is there a spanning tree of this undirected graph with total weight less than w?
- -Does this bipartite graph have a perfect (all nodes matched) matching?
- -Does the pattern p appear as a substring in text t?

NP –HARD AND NP – COMPLETE PROBLEMS

BASIC CONCEPTS

- The computing times of algorithms fall into two groups.
- Group1– consists of problems whose solutions are bounded by the polynomial of small degree.

Example – Binary search o(log n) , sorting o(n log n), matrix multiplication 0($n^{2.81}$).

NP –HARD AND NP – COMPLETE PROBLEMS

- **Group2** – contains problems whose best known algorithms are non polynomial.

- Example –Traveling salesperson problem $0(n^2 2^n)$, knapsack problem $0(2^{n/2})$ etc.

- There are two classes of non polynomial time problems
  1) NP- hard
  2) NP-complete

- A problem which is NP complete will have the property that it can be solved in polynomial time iff all other NP – complete problems can also be solved in polynomial time.

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Concepts

The class NP (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle.''

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back, and it is easy to see that the answer is
correct once you see it.

Example... Does matrix A have an LU decomposition?

No guarantee if answer is ``no''.

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Concepts

Another way of thinking of NP is it is the set of problems that can solved efficiently by a really good guesser.

The guesser essentially picks the accepting certificate out of the air (Non-deterministic Polynomial time). It can then convince itself that
it is correct using a polynomial
time algorithm. (Like a right-brain, left-brain sort of thing.)

Clearly this isn't a practically useful characterization: how could we build such a machine?

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Conepts

Exponential Upperbound

Another useful property of the class NP is that all NP problems can be solved in exponential time (EXP).

This is because we can always list out all short certificates in exponential time and check all O(2nk) of them.

Thus, P is in NP, and NP is in EXP. Although we know that P is not equal to EXP, it is possible that NP = P, or EXP, or neither. Frustrating!

NP –HARD AND NP – COMPLETE PROBLEMS
Basic Conepts

NP-hardness

As we will see, some problems are at least as hard to solve as any problem in NP. We call such problems NP-hard.

How might we argue that problem X is at least as hard (to within a polynomial factor) as problem Y?

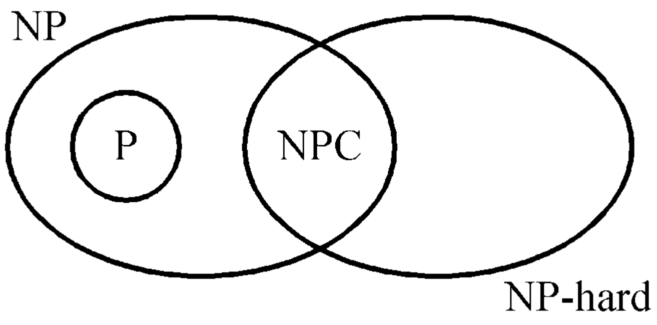
If X is at least as hard as Y, how would we expect an algorithm that is able to solve X to behave?

NP –HARD AND NP – COMPLETE PROBLEMS

- If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.

- All NP-complete problems are NP-hard, but all NP-hard problems are not NP-complete.

- The class of NP-hard problems is very rich in the sense that it contain many problems from a wide variety of disciplines.



- **P**: the class of problems which can be solved by a deterministic polynomial algorithm.

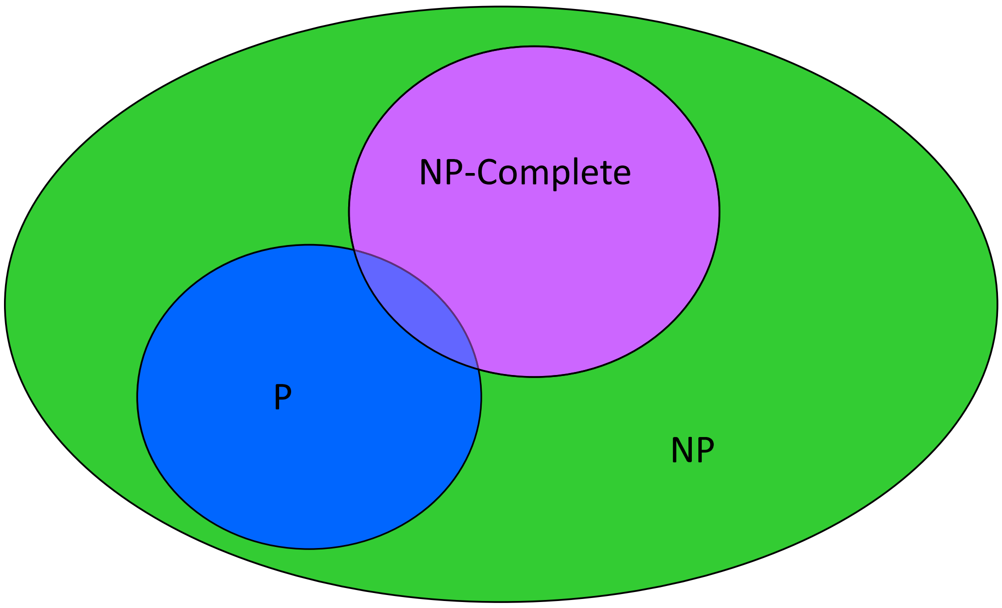
- **NP** : the class of decision problem which can be solved by a non-deterministic polynomial algorithm.

- **NP-hard**: the class of problems to which every NP problem reduces.

- **NP-complete (NPC)**: the class of problems which are NP-hard and belong to NP.

NP-Completeness

- How we would you define NP-Complete?
- They are the “hardest” problems in NP



DETERMINISTIC and NONDETERMINISTIC ALGORITHMS

- Algorithms with the property that the result of every operation is uniquely defined are termed deterministic.
- Such algorithms agree with the way programs are executed on a computer.
- In a theoretical framework, we can allow algorithms to contain operations whose outcome is not uniquely defined but is limited to a specified set of possibilities.

Deterministic and Nondeterministic algorithms

- The machine executing such operations are allowed to choose any one of these outcomes subject to a termination condition.

- This leads to the concept of non deterministic algorithms.

- To specify such algorithms in SPARKS, we introduce three statements

  i) choice (s) ……… arbitrarily chooses one of the elements of the set S.

  ii) failure …. Signals an unsuccessful completion.

  iii) Success : Signals a successful completion.

Deterministic and Nondeterministic algorithms

- Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.

- A nondeterministic algorithm terminates unsuccessfully if and only if there exists no set of choices leading to a successful signal.

- A machine capable of executing a non deterministic algorithm is called a non deterministic machine.

- While non-deterministic machines do not exist in practice they will provide strong intuitive reason to conclude that certain problems cannot be solved by fast deterministic algorithms.

Nondeterministic algorithms

- A nondeterminstic algorithm consists of

  phase 1: guessing

  phase 2: checking

- If the checking stage of a nondeterministic algorithm is of polynomial time-complexity, then this algorithm is called an NP (nondeterministic polynomial) algorithm.
- NP problems : (must be decision problems)
  - e.g. searching, MST

    sorting

    satisfiability problem (SAT)

    traveling salesperson problem (TSP)

Example of a non deterministic algorithm

```
// The problem is to search for an element  x //

// Output j such that A(j) =x; or j=0      if x  is not  in A //

 j $\rightarrow$choice (1 :n )

if A(j) =x then print(j) ; success endif

print (‘0’) ; failure

complexity  0(1);
```

- Non-deterministic decision algorithms generate a zero or one as their output.
- Deterministic search algorithm complexity. Ω(n)