Learn DAA : From B K Sharma

# TCS-503: Design and Analysis of Algorithms

## Unit V: Selected Topics: NP Completeness

# Unit V

- Selected Topics:
  - NP Completeness
  - Approximation Algorithms
  - Randomized Algorithms
  - String Matching

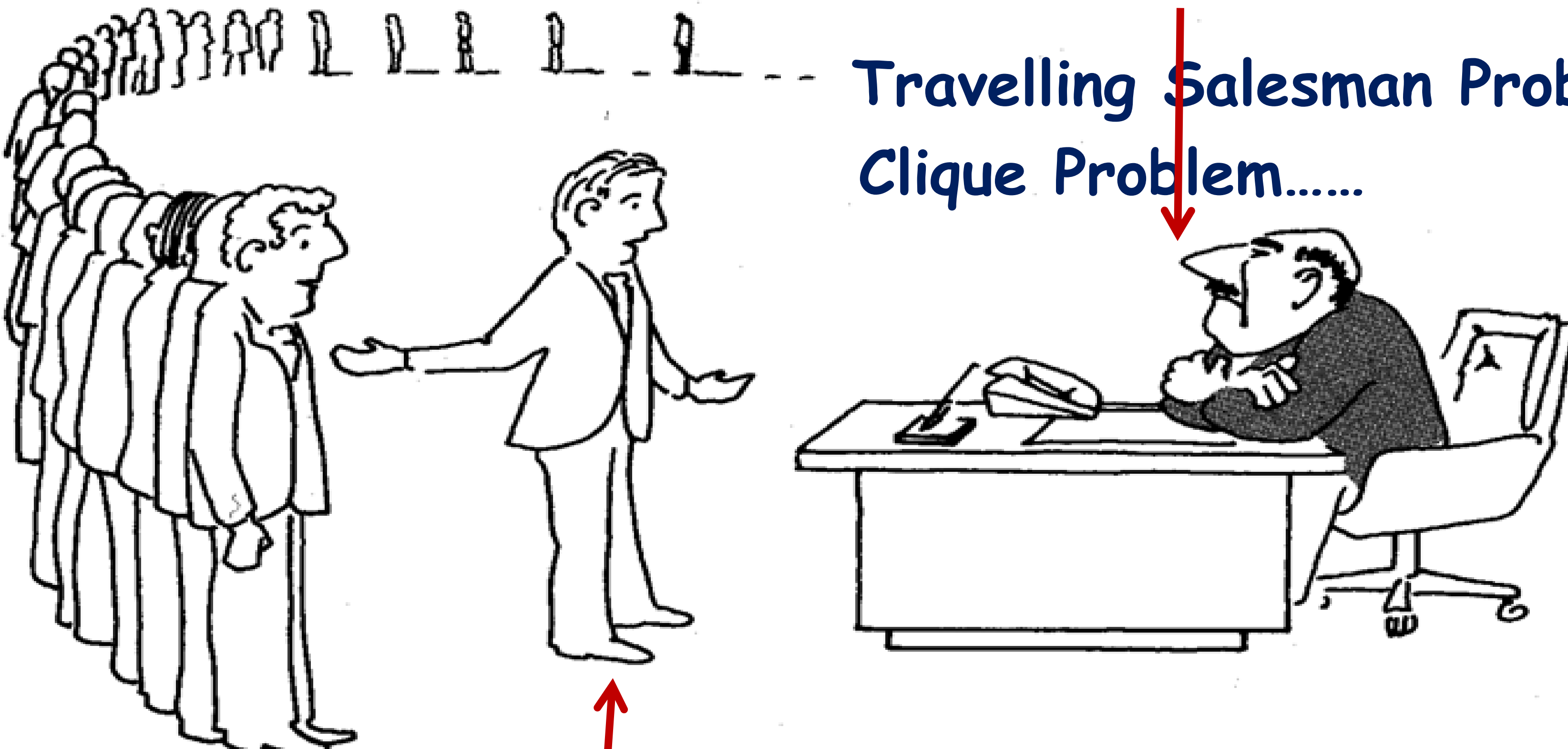
Learn DAA : From B K Sharma

# *NP-completeness*

Do Your Best, Gentlemen.

Travelling Salesman Problem,

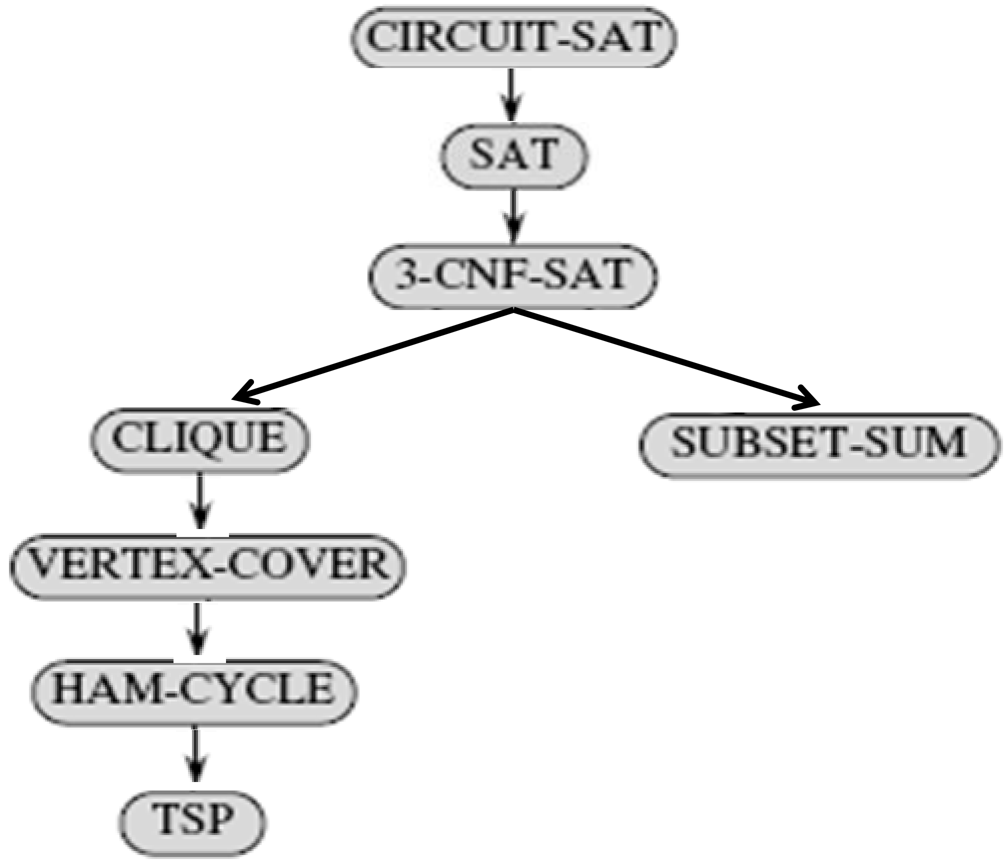
Clique Problem……



I can't find an efficient algorithm for,

But neither can these famous people.

# NP-completeness



Learn DAA : From B K Sharma

# Polynomial Time Algorithms

**Time Complexity:** $O(n^k)$ ,$O(n^k.m^k)$ ,$O(n^k.\log(n^k))$

**Where,**

k is a constant, k=1,2,3.......

| | | | | |
|---|---|---|---|---|
| n = | 2, | 10 | 20 | 30 |
| $n^k$= | $2^1$ | $10^2$ | $20^3$ | $30^4$ |
| = | 2 | 100 | 8000 | 810000 |

# Non-Polynomial Time Algorithms /Exponential time Algorithms

**Time Complexity:** $O(2^n)$ ,$O(n.2^n)$,$O(n!)$ ,$O(n^n)$

| | | | | |
|---|---|---|---|---|
| n= | 2 | 10 | 20 | 30 |
| $2^n$= | 4 | 1024 | 1 million | 1000 million |

# Introduction

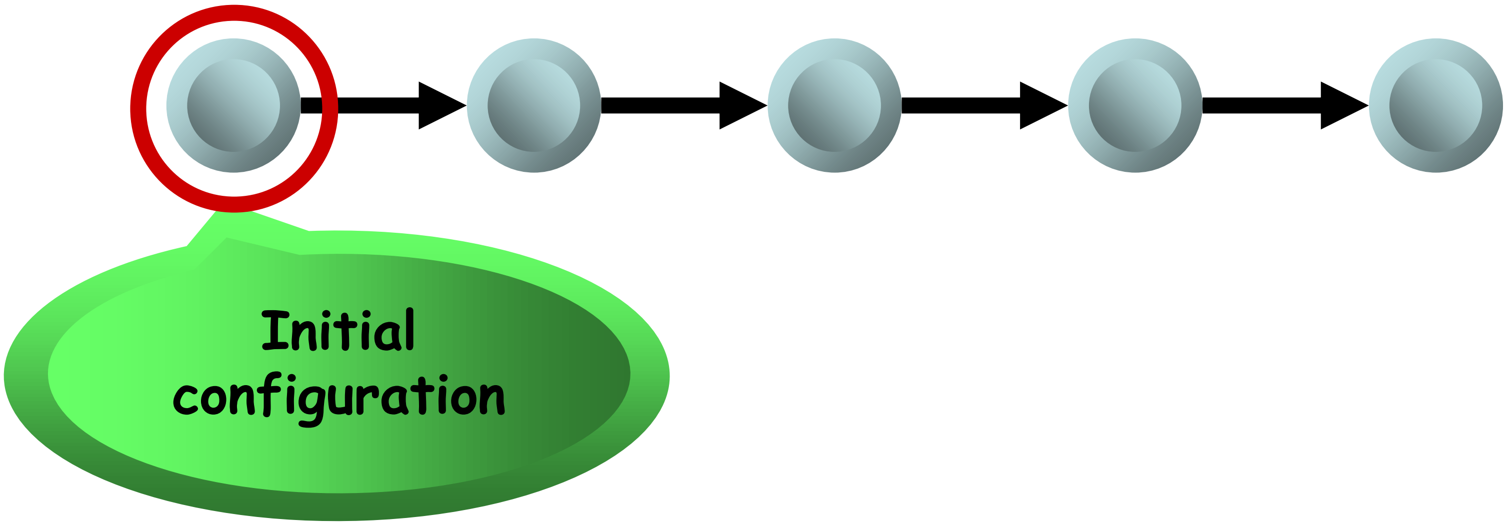
**Almost all the algorithms we have studied thus so far have been polynomial-time algorithms:**

- Bubble Sort, Selection Sort: $O(n^2)$
- Insertion Sort: $O(n^2)$
- Quick Sort: $O(n^2)$
- Counting Sort, Radix Sort, Bucket Sort: $O(n)$
- Other Algorithms: $O(nm^2)$, $O(n\lg n)$

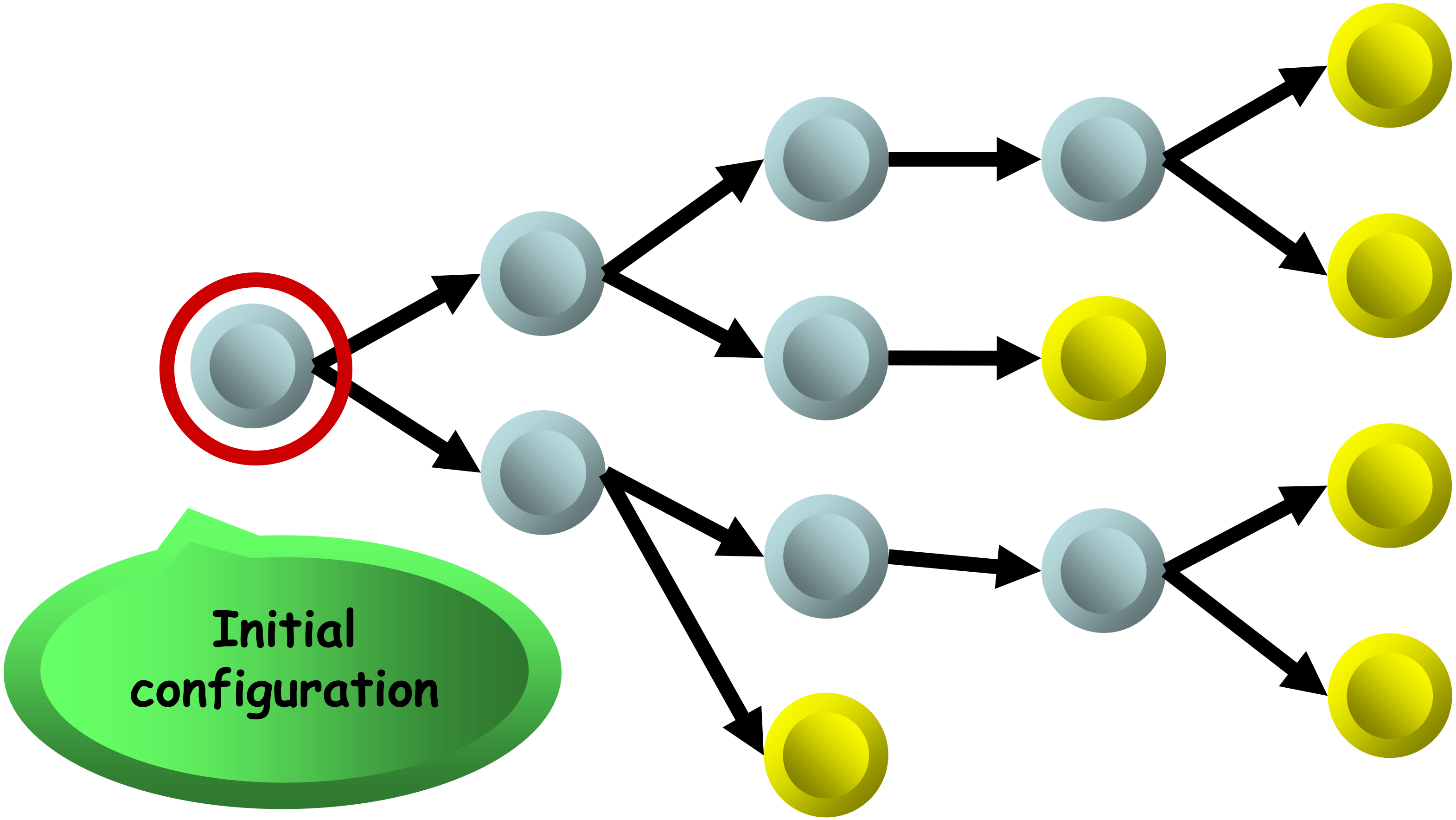
**Non-polynomial Time Algorithms:**

- Travelling Salesman Problem: $O(n\ 2^n)$
- Clique Problem: $O(n\ 2^n)$

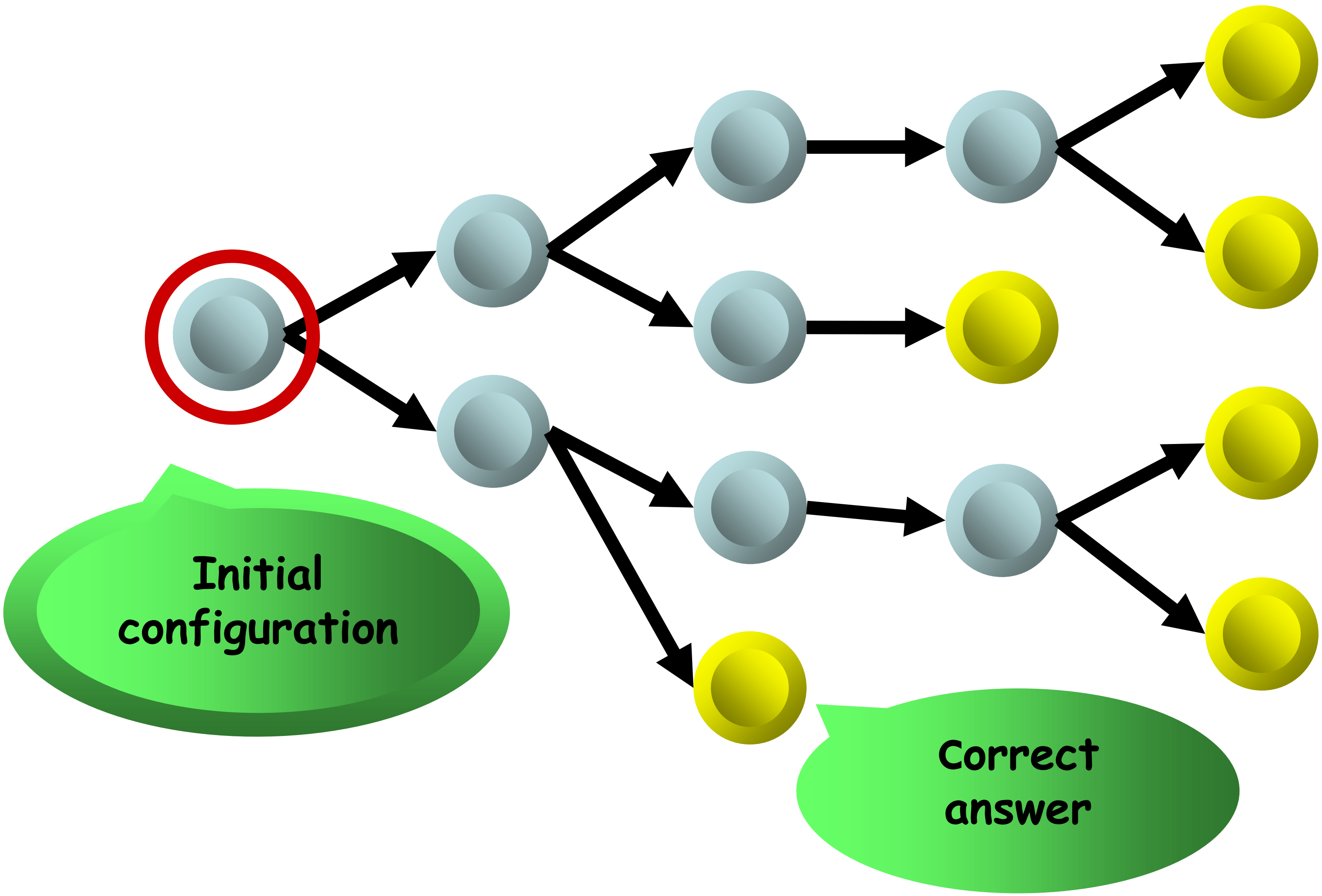
## Deterministic algorithm



# Non-Deterministic algorithm

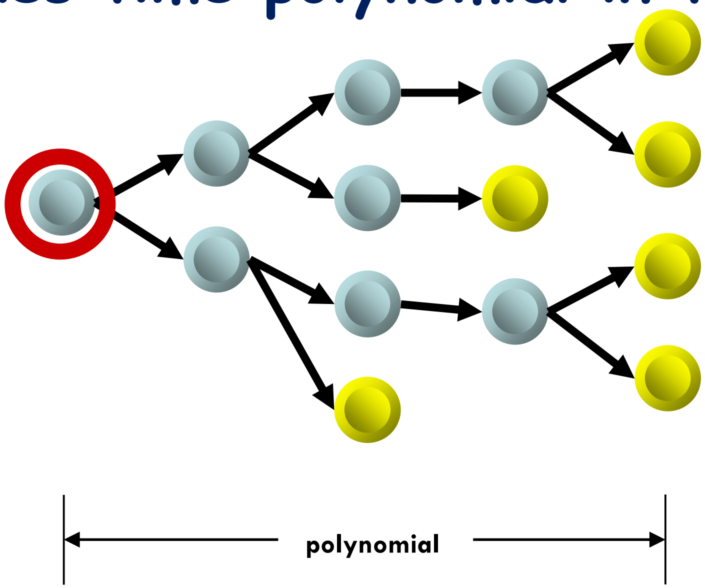


# Non-Deterministically Solving a problem



# Non-Deterministic polynomial time algorithm

We say that a non-deterministic algorithm N runs in polynomial time if for any input x of N, any computation of N on x, takes time polynomial in the size of x.



## Optimization Problem

Find a solution with the "best" value.

Find a path between u and v that uses the fewest edges.

## Decision Problem

Given an input and a question regarding a problem, determine if the answer is yes or no.

*Does a path exist from u to v consisting of at most k edges?*

# Optimization Vs. Decision Problem

Optimization problems can be cast as decision problems that are easier to study or at least no harder than the optimization problem..

In other words, if an optimization problem is easy, its related decision problem is also easy.

In other words, if we can provide evidence that a decision problem is hard, we also provide evidence that its related optimization problem is hard.

# Four Classes of Problems

- **P** — Polynomial
- **NP** — Non-Deterministic Polynomial
- **NPH** — NP- Hard
- **NPC** — NP-Complete

# Class P

The class P consists of all decision problems that can be solved in polynomial time $O(n^k)$, by **deterministic** , **computers** (the ones that we have used all our life!).

For examples:

Adding two numbers: $O(1)$ "constant"

Looking for an element in an array: $O(N)$ "lineal"

Extracting the element with the highest priority from a heap:

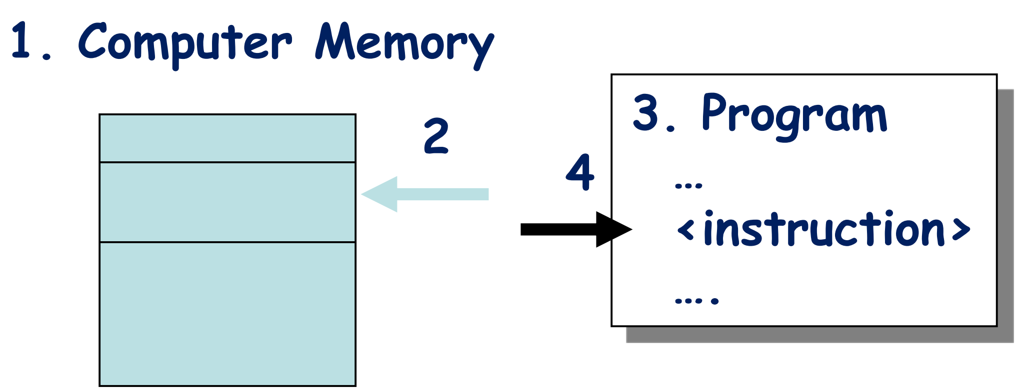
$O(\log N)$ "logarithmic" (thus, $O(N)$ because $N \geq \log N$ )

Looking for the MST:

$O(N \log N)$ (thus, $O(N^2)$ )

# What does Deterministic Computer Means? (Idea)

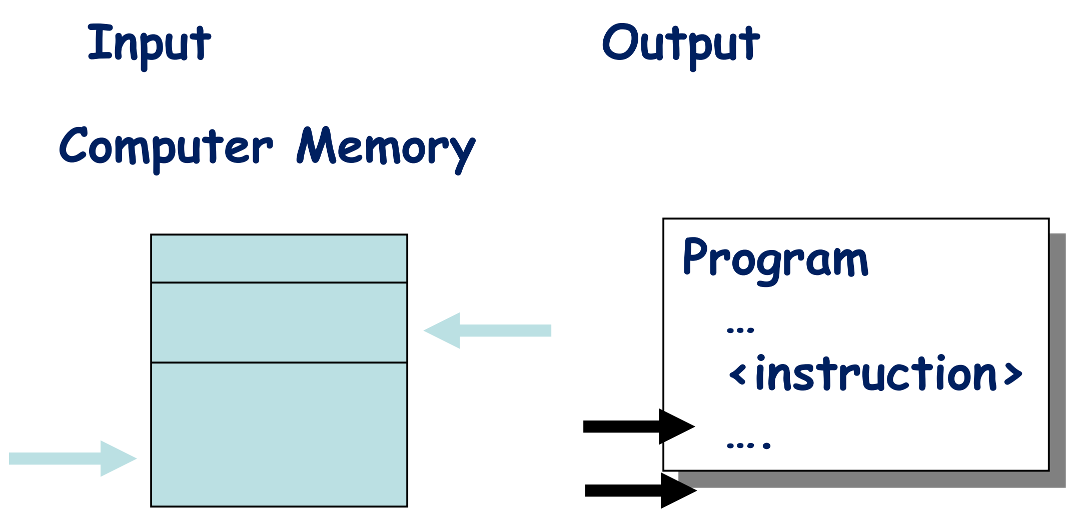
**At every computational cycle we have the so-called state of the computation:**




State = (1. memory, 2. location of memory being pointed at, 3. program, 4. current instruction)

# What does Deterministic Computer Means? (Idea II)

In a deterministic computer we can determine in advance for every computational cycle, the output state by looking at the input state.



# What does Deterministic Computer Mean? (Example)

//input: an array A[1..N] and an element el that is in the array

//output: the position of el in A

```
search(el, A, i)
  {
    if (A[i] = el) then return i
    else
       return search(el, A, i+1)
  }
```

el = 9

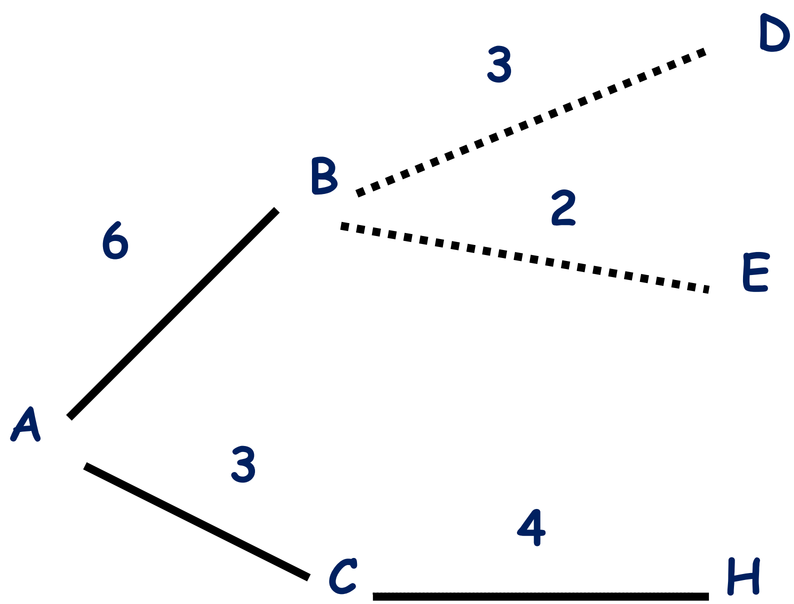
7 5 3 8 9 3

Complexity: O(N)

Learn DAA : From B K Sharma

# Djikstra's Shortest Path Algorithm

If the source is A, which edge is selected in the next iteration?




Complexity: O(N log N) ( N = number of edges + vertices)

# Class P

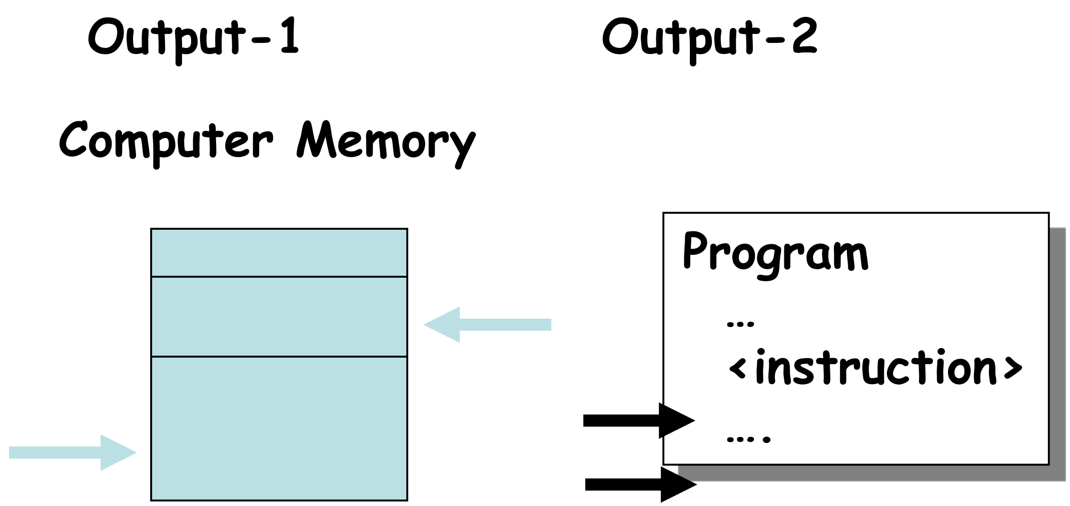
## Formal Definition

*Class P* is a class of decision problems that can be solved in polynomial time by (deterministic) algorithms.

This class of problems is called polynomial.

# What does Non-Deterministic Computer Mean?

In a non-deterministic computer, we may have more than one output state.




The computer "chooses" the correct one ("nondeterministic choice")

# *Nondeterministic algorithm*

## Formal Definition

*A nondeterministic algorithm* is a two-phase procedure that takes as its input an instance *I* of a decision problem and does the following.

**Phase I:** Non-deterministic ("guess") phase:

Generate a candidate solution *S* to the given instance *I*

**Phase II:** Deterministic ("verification") Phase:

*A* deterministic algorithm takes both *I* and *S* as its input, and it outputs yes if *S* is a solution to instance *I*.

# Non-Deterministic Algorithm

## Formal Definition

A nondeterministic algorithm for a problem X is a two-stage procedure:

- In the first phase, a procedure makes a guess about the possible solution for X.
- In the second phase, a procedure checks if the guessed solution is indeed a solution for X.

```
Phase1(el, A)
{
    i ← random(1..N)
     return i
}
```

```
Phase2(i,el, A)
{
    return
}           A[i] == el
```

Note: the actual solution must be included among the possible guesses of phase 1

# Class NP

## Formal Definition

*Class NP* is the class of decision problems that can be solved by nondeterministic polynomial algorithms.

This class of problems is called nondeterministic polynomial.

Contains Problems that are verifiable in polynomial time.

Given a "certificate" of a solution, we can verify that the solution is correct in polynomial time.

# Class NP

The class NP consists of all problems that can be solved in polynomial time by nondeterministic algorithms. (that is, both phase 1 and phase 2 run in polynomial time).

If X is a problem in P then X is a problem in NP because

Phase 1: use the polynomial algorithm that solves X

Phase 2: write a constant time procedure that always returns true.

# NP Class

How to proof that a problem *X* is in NP:

1. Show that *X* is in P, or
2. Write a nondeterministic algorithm solving X that runs in polynomial time.

# NP Class

## Showing that searching for an element in an array is in P:

1. Write the procedure search(el, A, i) which runs in lineal time,

//input: an array A[1..N] and an element el that is in the array

//output: the position of el in A

```
search(el, A, i)
  {
    if (A[i] = el) then return i
    else
      return search(el, A, i+1)
   }
```

el = 9

7 5 3 8 9 3

Complexity: O(N)

# Class NP

## Showing that searching for an element in an array is in P:

OR

## 2. Write a non-deterministic algorithm solving search

```
Phase1(el, A)
{
    i ← random(1..N)
     return i
}
```

```
Phase2(i,el, A)
{
    return
}          A[i] == el
```

(both Phase 1 and Phase 2 run in constant time)

# Class NP

There is a large number of important problems, for which no polynomial-time algorithm has been found, nor the impossibility of such an algorithm has been proved.

Some samples are:

1. Knapsack Problem
2. Traveling Salesman Problem
3. Graph Coloring Problem
4. 3-CNF-SAT Problem

# Is P=NP?

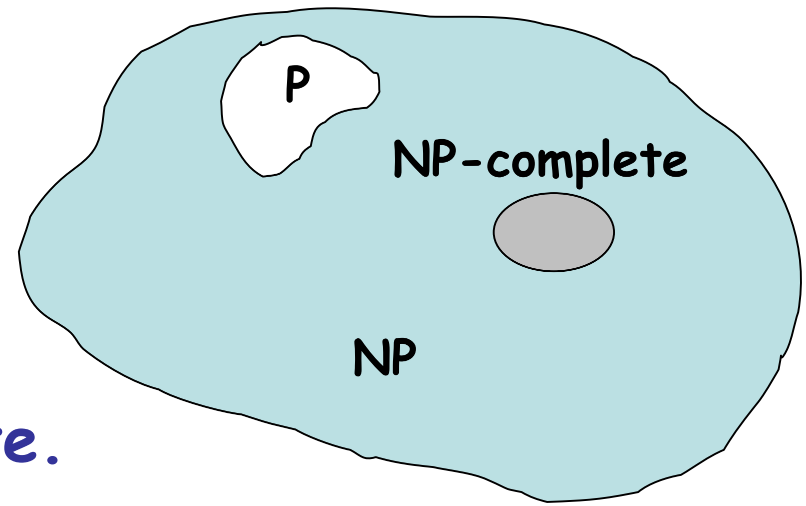
The key question is:

are there problems in NP that are not in P or is P = NP?

Any problem in P is also in NP:

$$P \subseteq NP$$

We can solve problems in P, even without having a certificate.




The big (and open question) is whether $NP \subseteq P$ or $P = NP$

i.e., if it is always easy to check a solution, should it also be easy to find a solution?

# Is P=NP?

Most computer scientists believe that this is false but we do not have a proof …

We know that P $\subseteq$ NP since a deterministic TM is also a nondeterministic TM.

But it is unknown if P = NP.

The Clay Mathematics Institute has offered a million dollar prize to anyone that can prove that P=NP or that P$\neq$NP.

P is clearly a subset of NP.

**Theorem:** If any NP-Complete problem can be solved in polynomial time $\Rightarrow$ then P = NP.

# A Decision Problem 'A' Polynomially Reducible To A Decision Problem 'B'

$$A \leq_p B$$

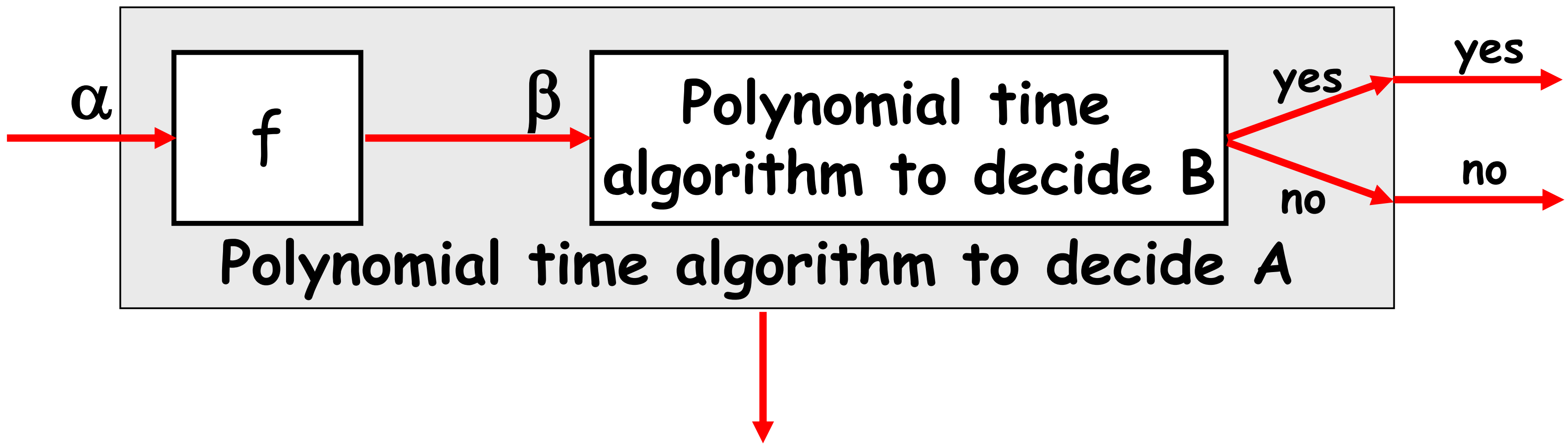
## Formal Definition

A decision problem '*A*' is said to be ***polynomially reducible*** to a decision problem '*B*' if there exists a function '*f*' that transforms instances of *A* ($\alpha$) to instances of *B* ($\beta$) such that

1. '*f*' maps all '*yes*' instances of *A* to '*yes*' instances of B and '*no*' instances of *A* to '*no*' instances of B
2. '*f*' is computable by a polynomial-time algorithm.

# Polynomially Reducible To B Decision Problem

## $A \leq_p B$




If a problem A polynomially reducible to some problem B that can be solved in polynomial time, then problem A can also be solved in polynomial time.

# Class NP-H(Hard)

A problem B is NP-hard if every problem ("complete set") in NP can be reduced to B in polynomial the time.

Every problem in *NP* is polynomially reducible to B.

All NP problems $P_1$, $P_2$, $P_3$, $P_4 \ldots \leq_p$ B


(B is the hardest problem in the set NP)

# Class *NP-C*

A decision problem *X* is said to be *NP-complete* if

1. It belongs to class *NP* and
2. It is NP-hard