San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu
www.cs.sjsu.edu/~yazdankhah

Computation Complexity

Lecture 27

Day 31/31

CS 154

Formal Languages and Computability

Spring 2018

Agenda of Day 31

- About Final Exam
- Summary of Lecture 26
- Lecture 27: Teaching ...
  - Computation Complexity

About Final Exam

Reminder 3

- **Value**: 20%
- **Topics**: Everything covered from the beginning of the semester
- **Type**: Closed all materials

	Section 1	Section 2	Section 3
Date	Thursday, May 17^{th}	Thursday, May 17^{th}	Tuesday, May 22^{nd}
Time	2:45 - 5:00 pm	5:15 - 7:30 pm	2:45 - 5:00 pm
Venue	MH 233	MH 233	SCI 311

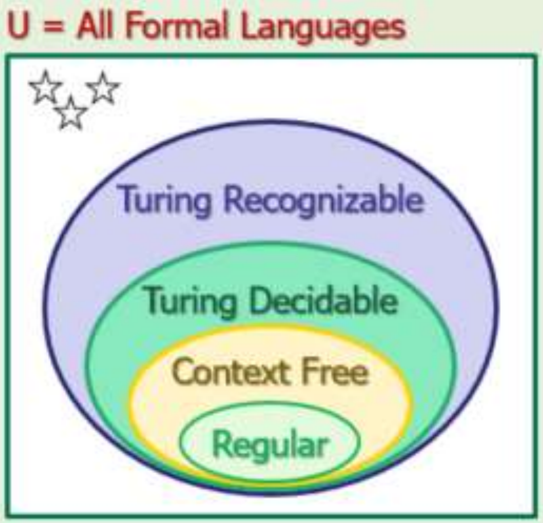
- We won't need whole 2:15 hours.
- As usual, I'll announce officially the type and number of questions via Canvas. (study guide)

Summary of Lecture 26: We learned ...

Computability

- Turing Thesis
  - Any computation carried out by a mechanical procedure can be performed by a TM.
  - We cannot prove or refute it.
- A language is Turing-recognizable if there is a TM that accept it.
- We have problem with the rejecting of the strings of $\overline{L}$.
  - Because the TM might get stuck in a forever loop.
- We prefer TMs that always halt.
- We called these TMs as deciders.

- A language is called Turing-decidable if there is a decider for it.

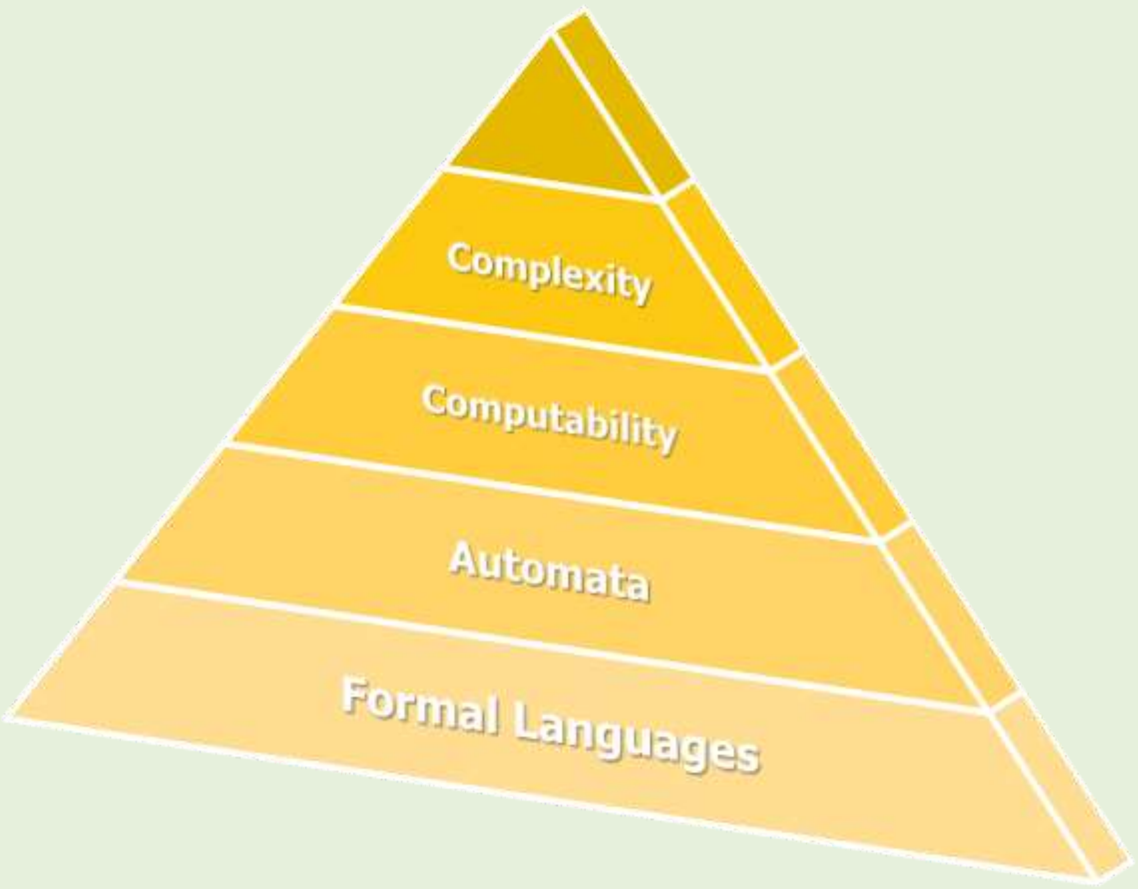


- Universal TM is a TM that can simulate other TMs.
- Halting problem shows the limitation of the theory of the computation.

Any question?

Big Picture: Computer Science Foundation

Recap



Objective of This Lecture

- What is complexity?

- What do we mean when we say:

  Computation A is more complex than computation B.

- How do we classify the problems based on their complexity?

- What classes of complexities do we have?

Computation Complexity

What is Computation Complexity?

- To answer this question, we need to be clear about two things:
  1. What is computation ?
  2. What is complexity?

- We've already defined computation as:

Recap

- The sequence of configurations from when the machine starts until it halts.

- Now, let's see what complexity is.

What is Complexity?

- Specifically, what do we mean when we say:

  Computation A is more complex than computation B?

- It means, computation A needs more resources.

- What are the resources?
- Time and space

- We do have other resources such as energy, number of CPU, etc., but time and space are usually our main concerns.

What is the Computation Complexity?

- Since we have two major types of resources, "time" and "space", so, we can talk about two types of complexities:
  1. Time-complexity
  2. Space-complexity
- In this lecture we focus on "time-complexity" that is usually of more concerns.

Time-Complexity

Assumptions

1. We are interested in "worst-cases" that needs the highest resources.
2. We define the efficiency of an algorithm as its complexity.

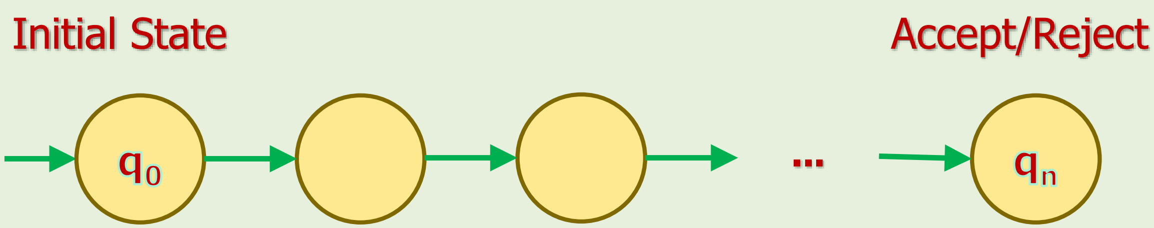
   The complexity of a computation = Efficiency of its algorithm

- Before going further, we need to have a clear understanding about the "computation time".

What is Computation Time?

- For any computation, the machine makes some transitions starting from the initial state until it halts.
  - Recall that if it doesn't halt, there won't be any computation!

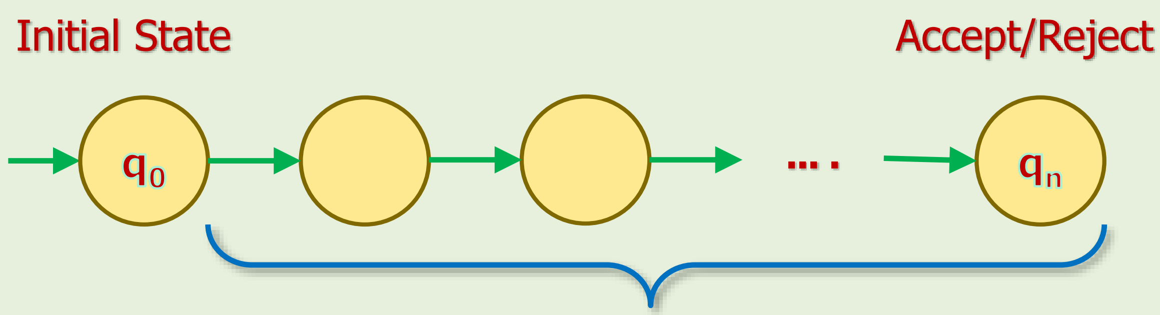
- For example, the following "one-dimensional projection" shows a computation for a process.



Computation Time of a Process

Definition

- The computation time of a process is the number of transitions from when the process starts until it halts.



Computation Time = Number of Transitions

- What would be the computation time when a machine gets stuck in an infinite-loop?

Computation Time of Nondeterministic TMs

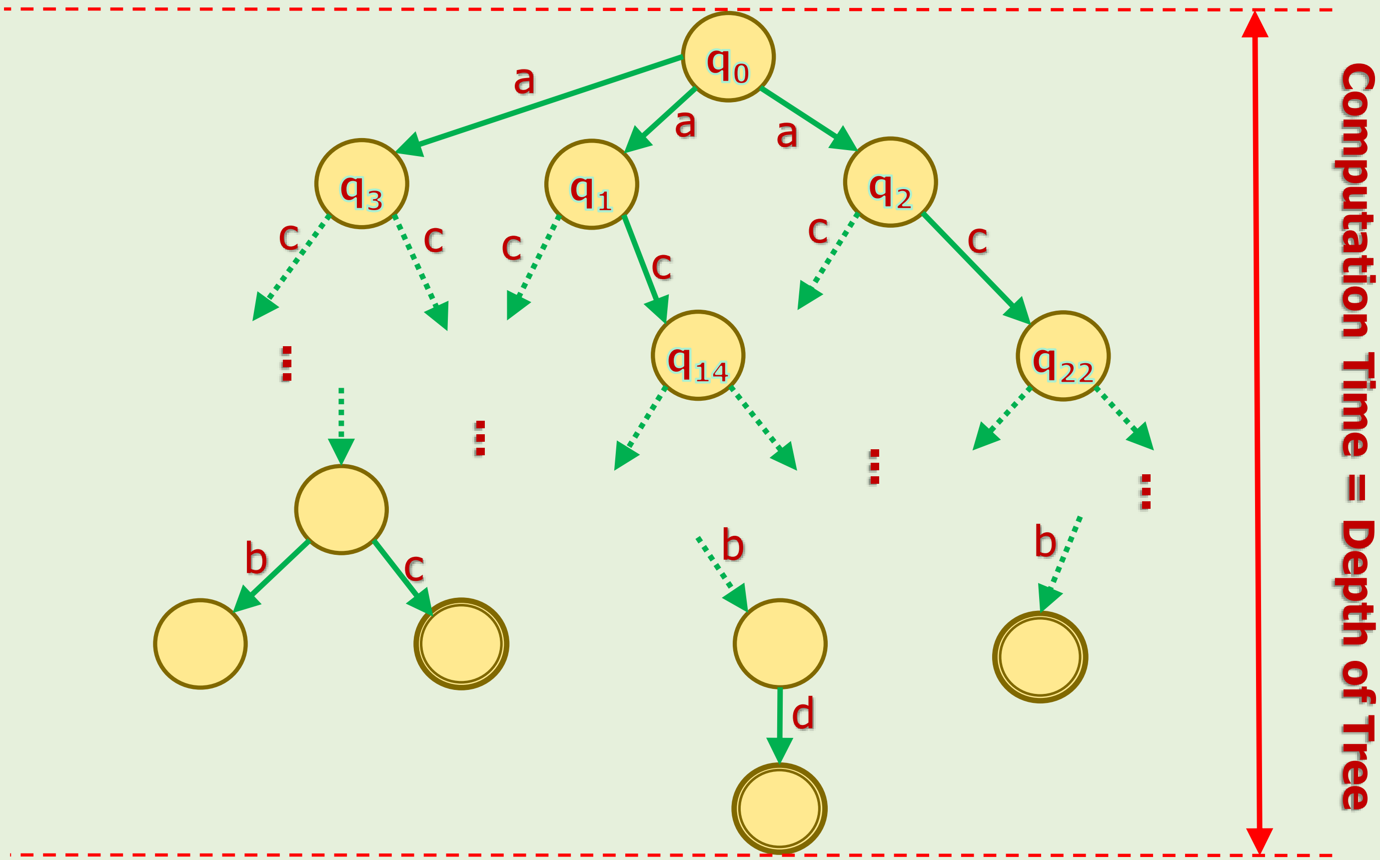
- Note that we defined the computation time for a single process.
  - Standard TMs use a single process, so the definition of computation time covers them.

What is the computation time of nondeterministic TMs?

- Since all processes of a nondeterministic TM run concurrently, then the computation time would be the computation time of the longest process.

- In the next slide, we combined all processes of a nondeterministic TM in a tree.
- The computation time would be the depth of the tree.

Computation Time of Nondeterministic TMs



- Note that just input symbol of the labels are shown for readability purpose.

Our Primary Concern

- How fast the resources requirements grow when the input size becomes larger.
- This is called "growth rate of resources".
- Definitely, slower growth of the resources requirements is desirable.
- To understand this concept, let's be more precise!

Growth Rate of Resources

- Consider the following deterministic automaton:

- We define the worst case computation time of this machine by f(n) that is a function of the input size n.

- What does f(n) look like for different types of automata?
- For example, if M is a DFA, how f(n) looks like?

$$f(n) = n$$

- How about for TMs?

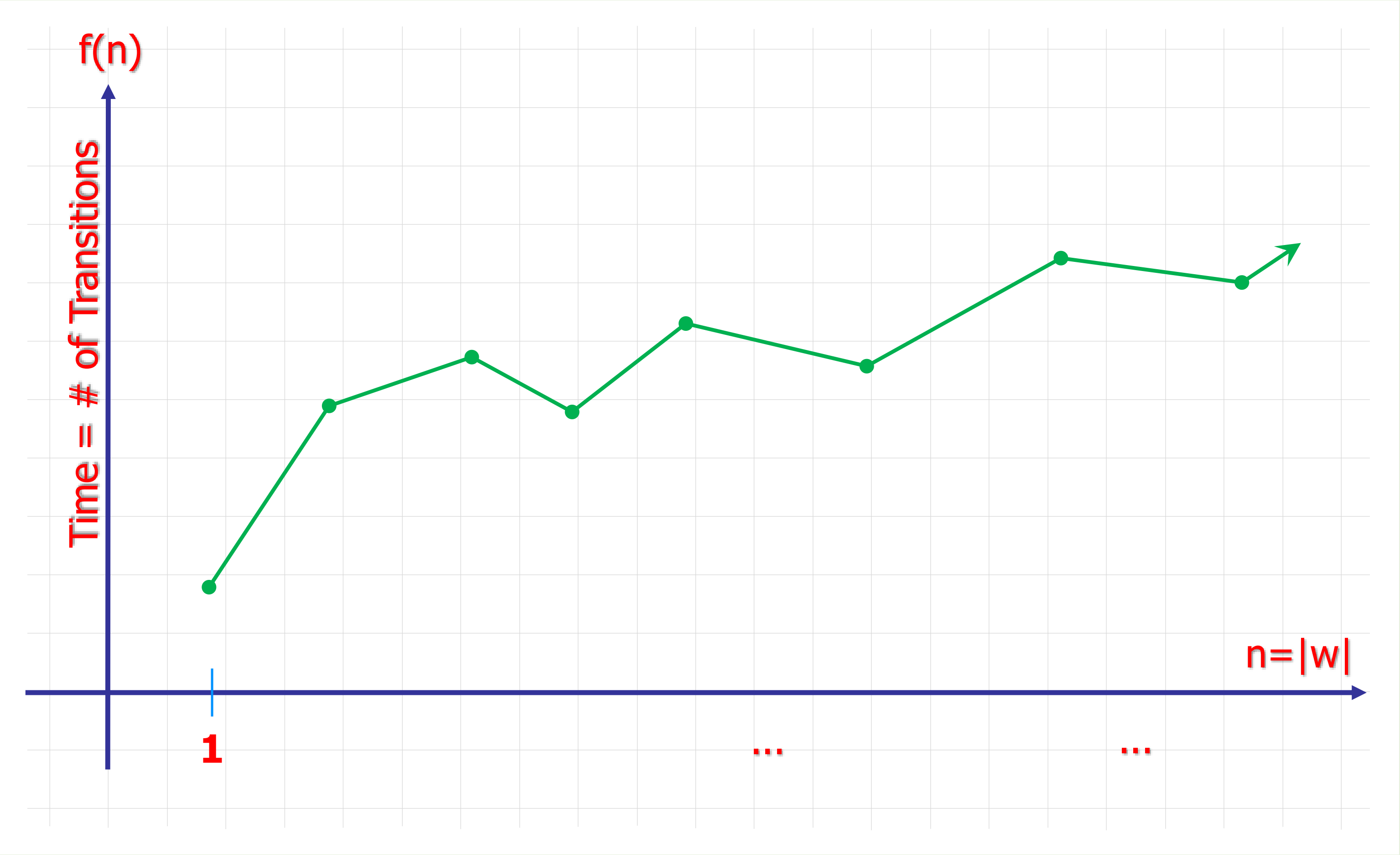
Growth Rate of Resources

- For TMs, the computation time for w's with the same size might vary.
- For example, a TM might have the following values for input size 3:

$$\mathrm{f}(3) = \begin{cases} 3 & if\ w = aaa \\ 5 & if\ w = aba \\ 5 & if\ w = baa \\ \dots & \quad\dots \\ 4 & if\ w = bbb \end{cases}$$

- In this case, we pick the worst case that is the longest one, 5, for f(3).
- If we do this for all sizes of w, we get f(n).
- Next slide shows an example of f(n) for a TM.

Example of a TM's Computation Graph

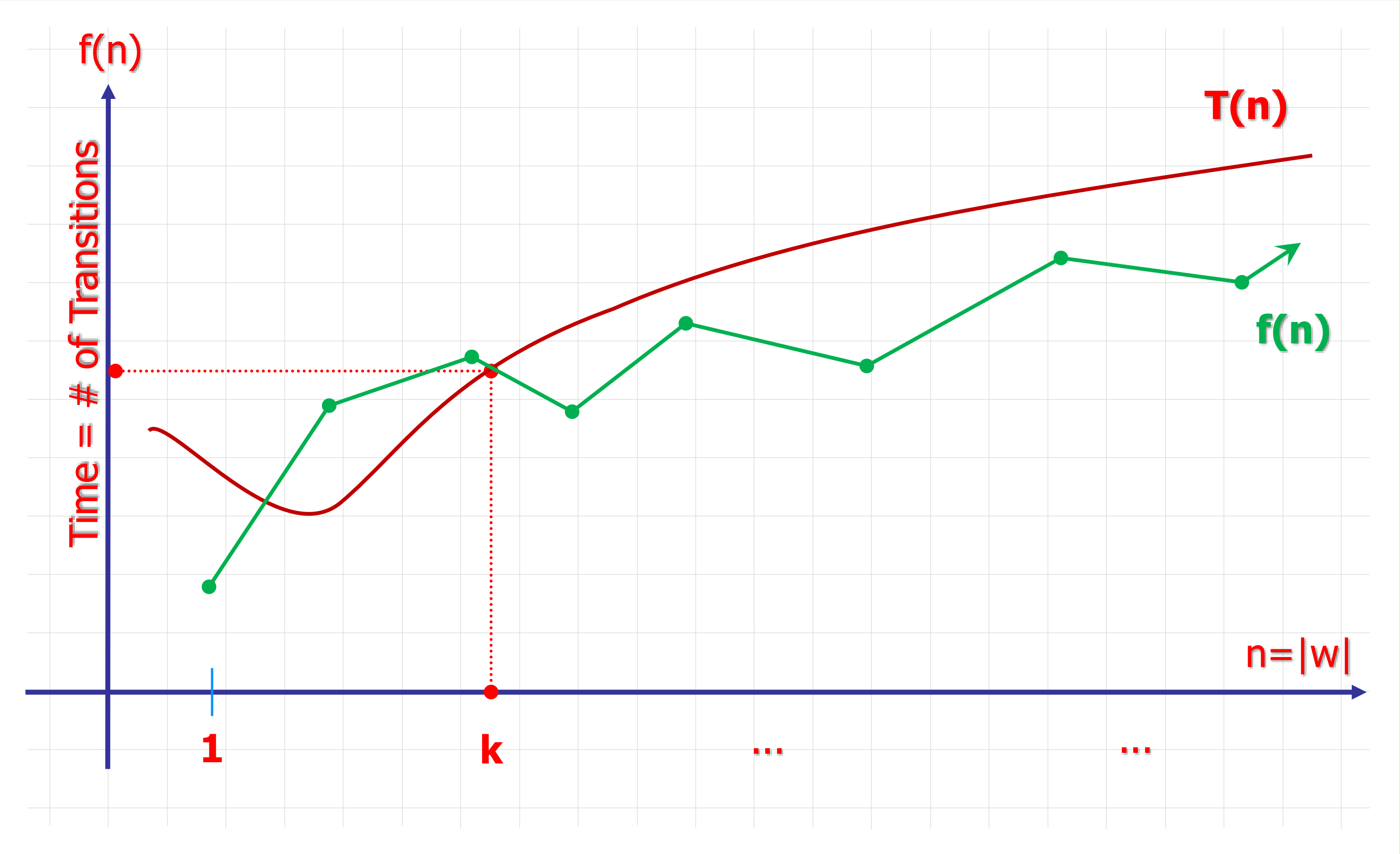


Growth Rate of Resources

- Almost always, the function f(n) is an unknown function.
  - Know functions such as f(n) = n, or Sin n, ...

- But most of times, we can approximate it with a known function as the next slide shows.

Growth Rate of Resources



- The approximate function T(n) should have the same "growth rate", from a point afterward (e.g. k).

Big-O Notation

Recap

- If we find such function, then we use a special notation called "Big-O" (aka "Order of magnitude") to represent it.

$$f(n) = O(T(n))$$

- The meaning of the above notation is:

  c .T(n) is an upper-bound for the growth rate of f(n).

  Where c is a positive real number.

- The above equal sign is an "asymptotic notation", not the regular equal sign.
  - That's why it is a very confusing notation.

Growth Rate Checking

Recap

How to prove that T(n) and f(n) have the same growth rate?

- We should calculate the following calculus equation:

$$\lim_{n \to \infty} \frac{T(n)}{f(n)} = ?$$

- If the result is a constant > 0, it means both have the same growth rate.
- If it is zero, it means f(n) grows faster.
- If it is $+\infty$, it means T(n) grows faster.

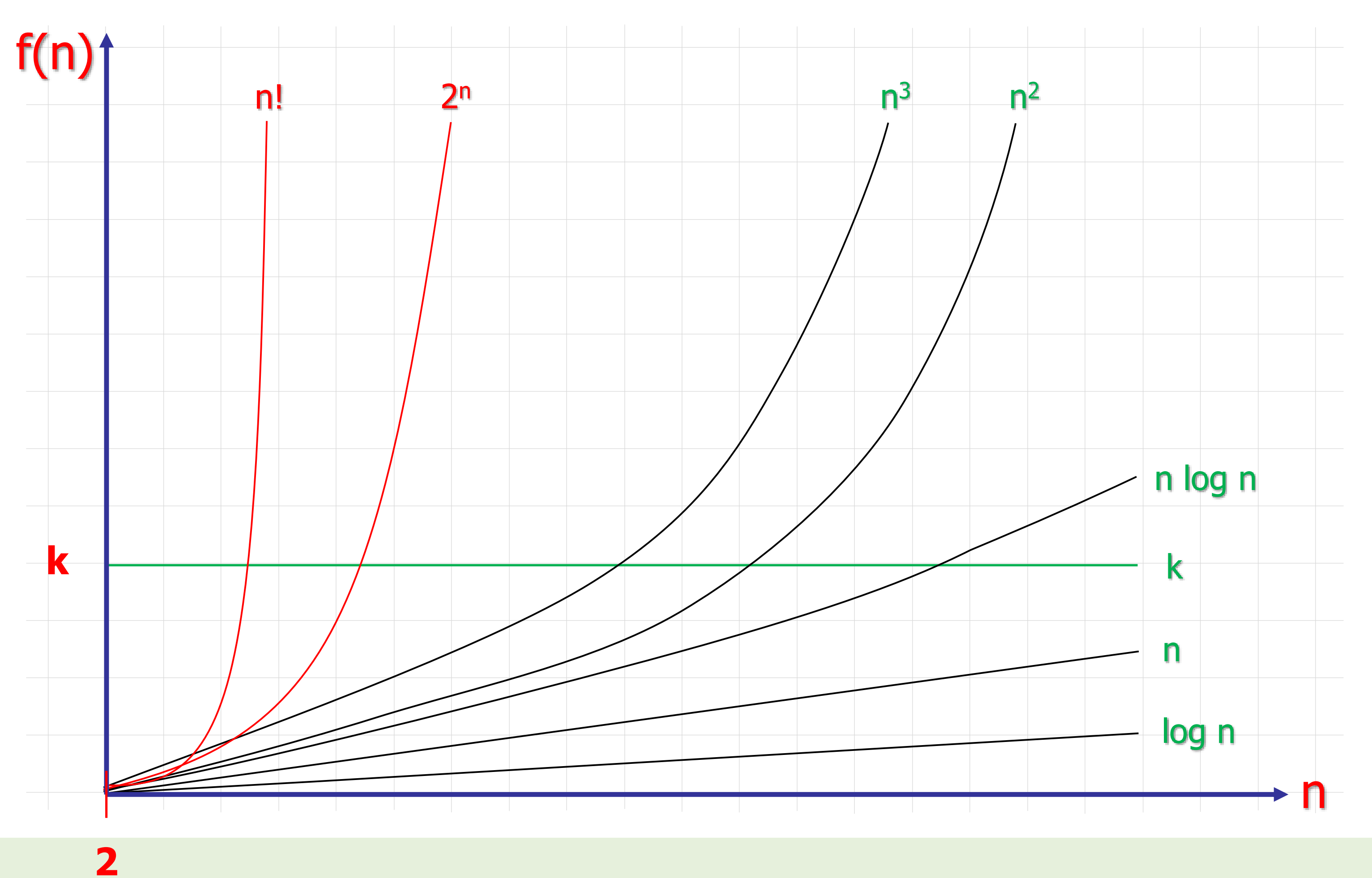
Growth Rate of Some Functions

- The following table shows how different functions grow when the input size grows.

n	k	n	n^2	n^3	2^n
1	k	1	1	1	2
2	k	2	4	8	4
3	k	3	9	27	8
…	…	…	…	…	…
10	k	10	100	1000	1024
…	…	…	…	…	…
100	k	100	10,000	1,000,000	2^{100} = ???

- Next slide shows the graphs of some known functions.

Growth Rate of Some Functions



Computation Complexity Comparison

- To be able to compare complexities, we need to quantify them.
- In computer science, it's been proven that Big-O is the best notation to quantify complexities.

Example 1

- Problem A needs $O(n^2)$ resources.
- Problem B needs $O(n)$ resources.
- Which problem is more complex?

- Problem A ...
- ... because the resource requirement of problem A grows faster than problem B.

Time-Complexity Classes

Time-Complexity Classes

- In this section, we'll classify problems (languages) based on their complexities.

- The goal of this classification is:

To have an engineering feeling about the types of problems that we encounter.

- To solve problems, we can use standard (deterministic) TM or nondeterministic TM.
  - As we'll see later, there is a huge difference between them.

- Let's start with "deterministic TM".

Complexity Class DTIME(n)

- Our first complexity class is called DTIME(n).
- It contains all problems that can be decided in O(n) time.
  - The "D" at the beginning of DTIME shows that we are using "deterministic" TMs.
- Let's see what problems we can put in this class.

Complexity Class DTIME(n)

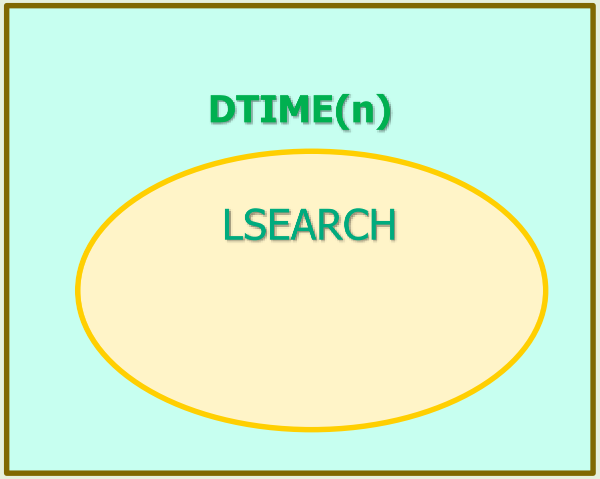
Example 2

- Given an unsorted list of numbers $x_1, x_2, ..., x_n$ and a key number k.
- Search in the list and determine if it contains k (LSEARCH).

- In the worst-case, we need n comparisons.
- So, the time-complexity of this problem is O(n).

- Note that we assume each comparison needs constant amount of time k.
- So, total time needed is n . k.
- In big-O notation, we eliminate constants.

U = All Formal Languages

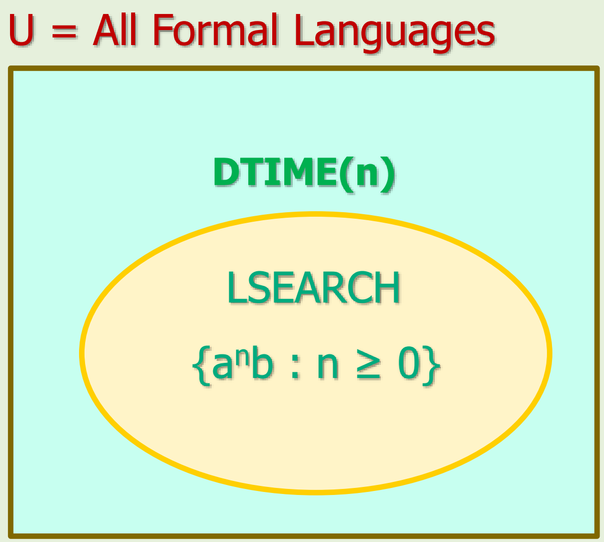


Complexity Class DTIME(n)

Example 3

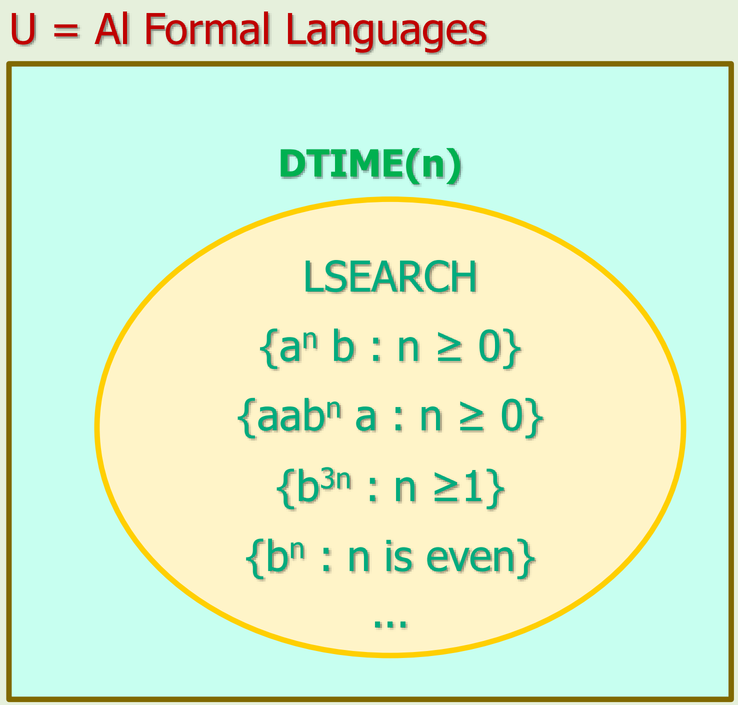
- Given $L = \{a^n b : n \geq 0\}$
- What is the time-complexity of accepting this language?

- L can be decided in O(n) by using a deterministic TM.



Complexity Class DTIME(n)

- Also, the following languages can be decided in O(n) by using a deterministic TM.



Time-Complexity Classes

- What are the complexities of the following languages?

- $L = \{a^n b^n : n \geq 0\}$
- $L = \{ww^R : w \in \{a, b\}^*\}$
- $L = \{ww : w \in \{a, b\}^*\}$

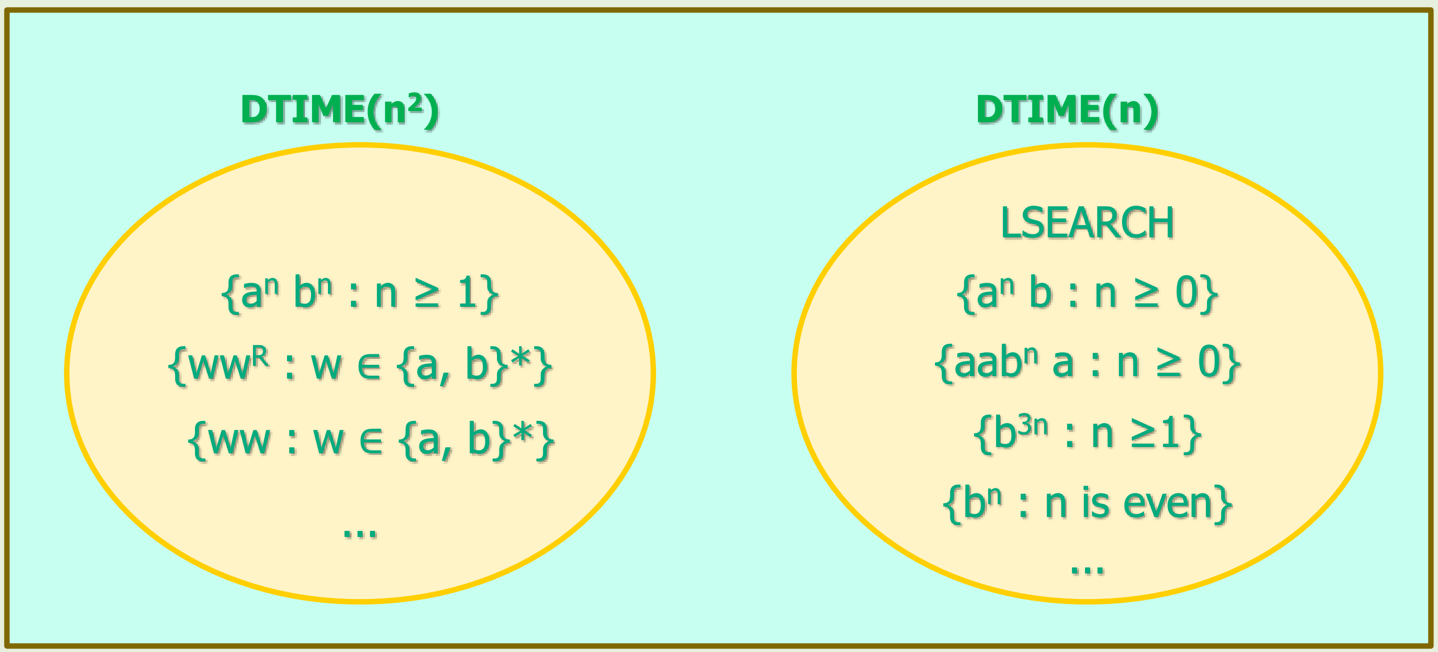
- $O(n^2)$

- So, we need a new class of complexity.

Complexity Class DTIME(n^2)

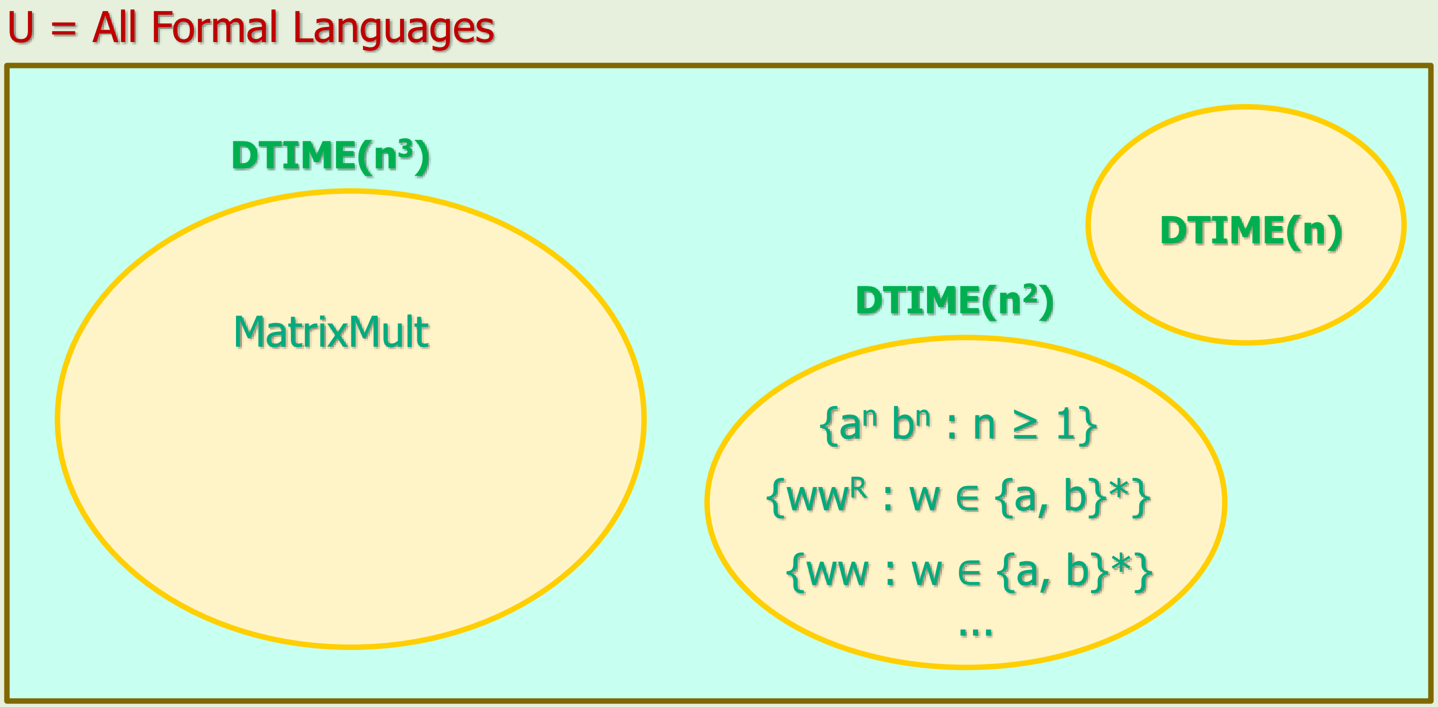
- We create a new class called DTIME(n^2) and put them in this new class.

U = All Formal Languages



Complexity Class DTIME(n^3)

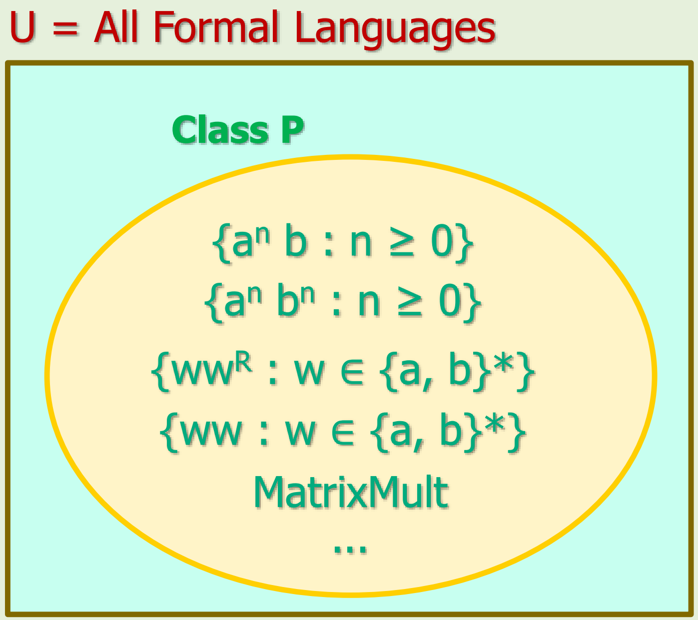
- Matrix multiplication problem can be decided in $O(n^3)$ by using a deterministic TM.
- So, we need another class for $O(n^3)$.



- We can continue this process for $O(n^4)$, $O(n^5)$, ..., $O(n^k)$.

Class P

- Classifying languages based on the degree of n has less practical benefit.
- We define a general class called "polynomial time-complexity" or just "class P".
- Class P is the set of problems that can be decided in $O(n^k)$ by using a deterministic TM.
  - Where $k \geq 0$
- Also, we call these problems "easy" (aka "tractable").
- We'll see within a few minutes why they are "easy"!



Exponential Time Algorithms

Introduction

- We continue our study about the classification of complexities by focusing on "exponential algorithms".
- But first, we need to get familiar with some of those problems.
- In the next slides we'll take some examples of problems that need exponential time to be decided.

Satisfiability Problem (SAT)

Problem

- As an example, consider the following logical expression:

$$X = (p \lor r) \land (\sim q \lor \sim r)$$

- For what values of p, q, and r, the expression X is satisfied (= true)?

Solution

- Using "truth table" is the most reliable way to find all solutions.
- The expression has three variable p, q, and r.
- Therefore, there are $2^3 = 8$ rows in the truth table.
- The algorithm should evaluate X for all rows to find all possible solutions.

Satisfiability Problem (SAT)

$X = (p \vee r) \wedge (\sim q \vee \sim r)$

1. $X = (T \vee T) \wedge (\sim T \vee \sim T) = F$
2. $X = (T \vee F) \wedge (\sim T \vee \sim F) = T$
3. $X = (T \vee T) \wedge (\sim F \vee \sim T) = T$
4. $X = (T \vee F) \wedge (\sim F \vee \sim F) = T$
5. $X = (F \vee T) \wedge (\sim T \vee \sim T) = F$
6. $X = (F \vee F) \wedge (\sim T \vee \sim F) = F$
7. $X = (F \vee T) \wedge (\sim F \vee \sim T) = T$
8. $X = (F \vee F) \wedge (\sim F \vee \sim F) = F$

	p	q	r
1	T	T	T
2	T	T	F
3	T	F	T
4	T	F	F
5	F	T	T
6	F	T	F
7	F	F	T
8	F	F	F

Satisfiability Problem (SAT)

- In the previous example, we used an exhaustive algorithm.

Algorithm

- Construct a truth table for 3 variables p, q, r.
- Evaluate X for every row.
- Pick those rows that X = true.

- So, theoretically this problem is computable.

Efficiency of Satisfiability Problem (SAT)

- If the number of variables is n, the truth table would have 2^n rows .
- We assume the evaluation of one row needs constant time.
  - We ignore the constant coefficients in big-O notation.
- Therefore, the algorithm needs 2^n evaluations.
- So, the efficiency of SAT problem is $O(2^n)$.

Efficiency of Satisfiability Problem (SAT)

- Is this algorithm practically feasible?

- What would happen if we had 100 variables?
- In this case, we'd need a table with 2^{100} rows.

- Do you have any idea how big is this number?

- To answer this question, let's "do some math".

Let's Do Some Math!

Example 4: A Practical Calculation for 2^{100}

- Consider a truth table with 100 variables and 2^{100} rows.

- If a computer processes each row in 1 Nano sec (10^{-9} sec), how long does it take for this computer to process entire table?

Solution

Let's Do Some Math!

Exhaustive Parsing Algorithm

$S \rightarrow SS \mid aSb \mid bSa \mid \lambda$
$w = abba...b$; $|w| = 50$

- Efficiency of exhaustive search parsing algorithm: $O(|P|^{2|w|})$

- We have a deterministic computer that can process each substitution in 1 Nano sec (10^{-9} sec).
- How long does it take to parse a string of length 50?

Let's Do Some Math Again!

- Let's take another look at the table of growth rate of functions.
- Compare, for example, one million rows of n^3 and the number that we just calculated for 2^{100}.
- One million rows can be processed within less than a second while 2^{100} needs ....
- That's why we call exponential algorithms as "hard" (aka "intractable").

n	k	n	n^2	n^3	2^n
1	k	1	1	1	2
2	k	2	4	8	4
3	k	3	9	27	8
...	...	...	...	...	...
10	k	10	100	1000	1024
...	...	...	...	...	...
100	k	100	10,000	1,000,000	2^{100} = ???

Hamilton Path Problem (HAMPATH)

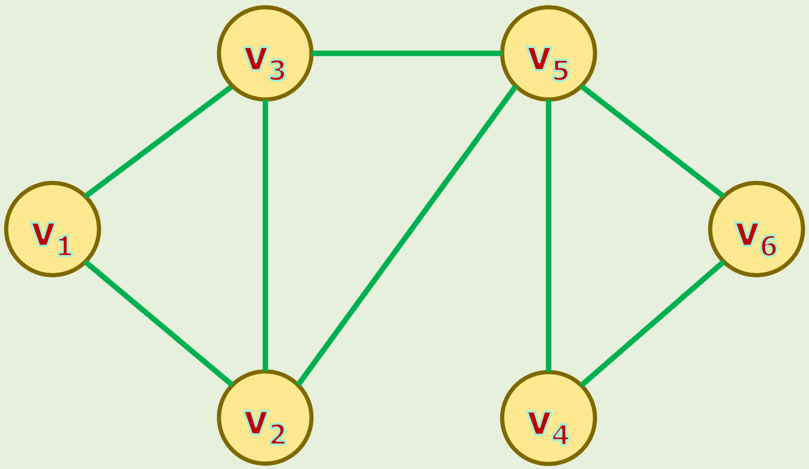
FYI

Problem

- Given an undirected graph with n vertices $v_1, v_2, \ldots, v_n$.
- Find a simple path that passes through all vertices.
  - This path, if exists, is called "Hamilton path".

Example 6

- Is there any Hamilton path in the following graph?

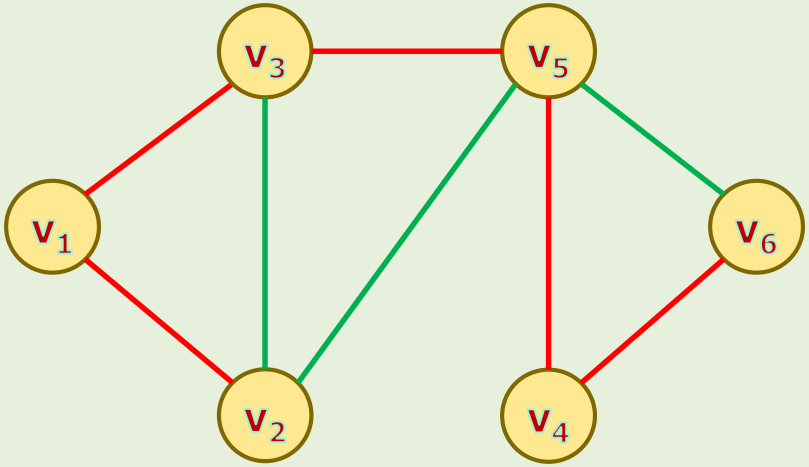


Hamilton Path Problem (HAMPATH)

FYI

Example 6 (cont'd)

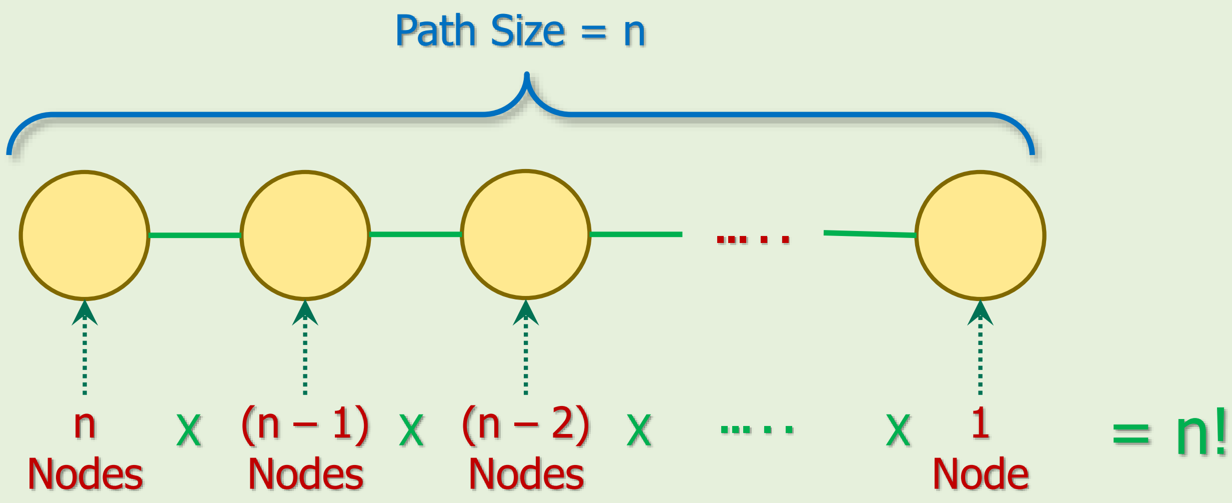
- Yes, $\{(v_2, v_1), (v_1, v_3), (v_3, v_5), (v_5, v_4), (v_4, v_6)\}$



Hamilton Path Problem Complexity

FYI

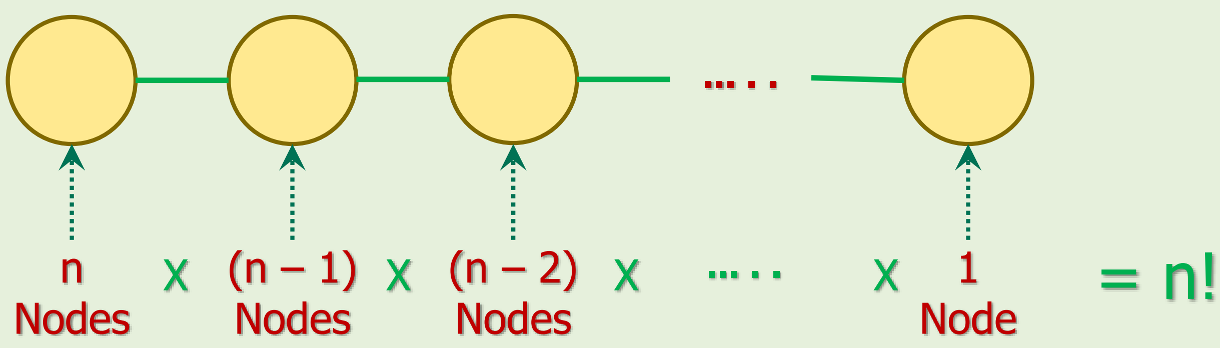
- Let's use an exhaustive algorithm to check all possible paths.
- How many different paths can we recognize?
- Since we need to visit all n vertices, so, all possible paths have size n.



Hamilton Path Problem Complexity

FYI

- The paths can start from all n nodes.
- The second node in each path can be (n-1) remaining nodes.
- The third node in each path can be (n-2) remaining nodes, and so forth ...
- Based on "multiplication rule of counting", total number of possible paths would be: $n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 = n!$



Hamilton Path Problem Complexity

FYI

- Therefore, the total number of paths in the worst-case is n!.
- So, investigating which path is Hamilton path needs O(n!) time.
- We usually don't have a clear feeling about how big is n!.
- So, we use Stirling's approximation for n!.

$$n! \approx \sqrt{2\pi\,n}\left(\frac{n}{e}\right)^n$$

Conclusion

- Complexity of HAMPATH ≈ $O(n^n)$
- So, if we have 100 vertices, then we need to check 100^{100} possibilities!
- Now you do the math!

Using Nondeterministic TMs

Using Nondeterministic TM

Theorem

- If a deterministic TM solves a problem in an exponential time $O(k^{an})$, a nondeterministic TM solves it in a polynomial time $O(n^p)$.

- Where:
  - p, k and a are constants

Using Nondeterministic TM

Example 8

- The SAT problem complexity = $O(2^n)$ (by using deterministic TM)

- If we solve this problem by using a nondeterministic TM,
  the complexity would be $O(n)$.

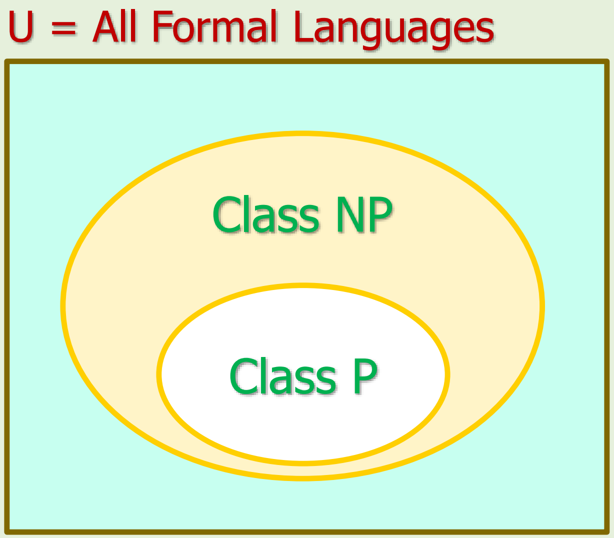
- Do the math again!

Class NP

- Class NP is the set of problems that can be decided in polynomial time by using nondeterministic TMs.
- NP = Nondeterministic Polynomial Time-Complexity

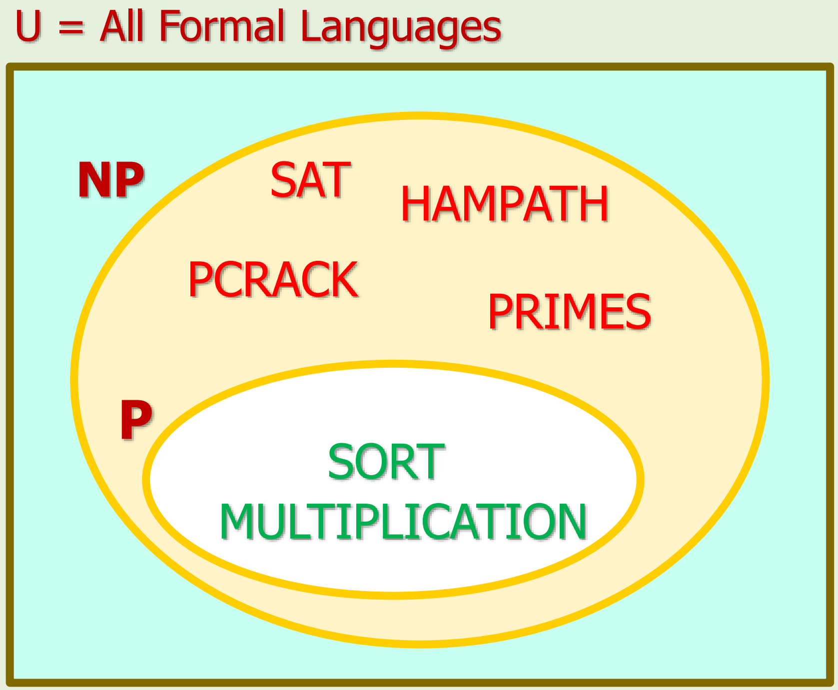
What is the relationship between class P and NP?

- All languages in class P can also be decided in polynomial time by using nondeterministic TM.
- So, $P \subseteq NP$



P vs. NP

- Computer scientists found polynomial time algorithms for some problems such as sorting, multiplication.
- They found exponential algorithms for some others such as SAT, HAMPATH, PRIMES (finding prime numbers), PCRACK (password cracking), ...
- We were lucky to find a polynomial time algorithm for some of them like PRIMES. (Agrawal, Kayal, Saxena / 2004)
- So, we moved PRIMES to class P. (next slide)



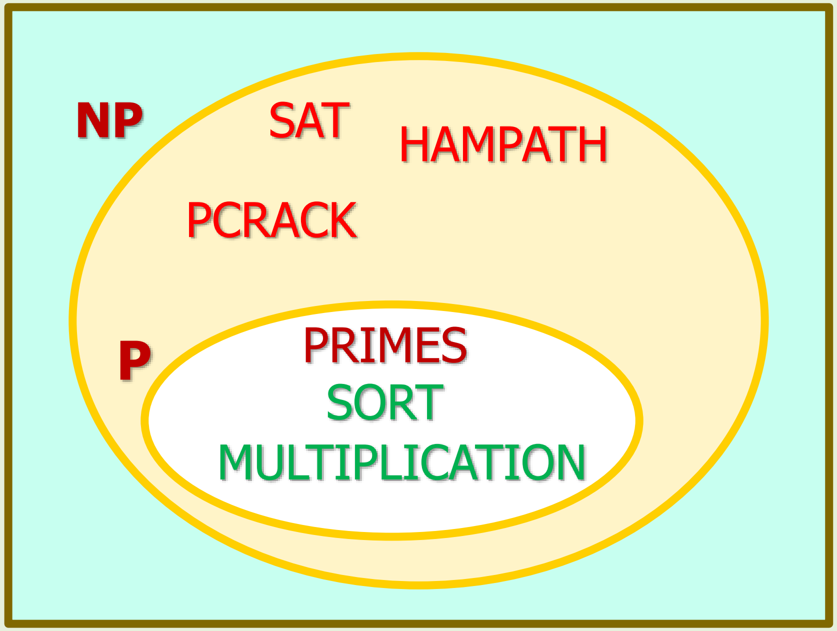
P vs. NP: An Open Question

- Now the question is:

  Can we find polynomial time algorithms for the rest of them?

- In other words, can we expect one day P = NP?
- We don't know yet.
- This is another "open question" of computer science.
- $1,000,000 for the solution!
- http://www.claymath.org

U = All Formal Languages



Last Note

- Note that it is not the case that we just have 2 or 3 classes.
- As of this moment, there are 535 known complexity classes!
- For more information, take a look at the "Complexity Zoo" website at:
  https://complexityzoo.uwaterloo.ca/Complexity_Zoo

The End

I wish you all, the Bests!

References

1. Linz, Peter, "An Introduction to Formal Languages and Automata, 5th ed.," Jones & Bartlett Learning, LLC, Canada, 2012
2. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790