Complexity

Complexity Theory

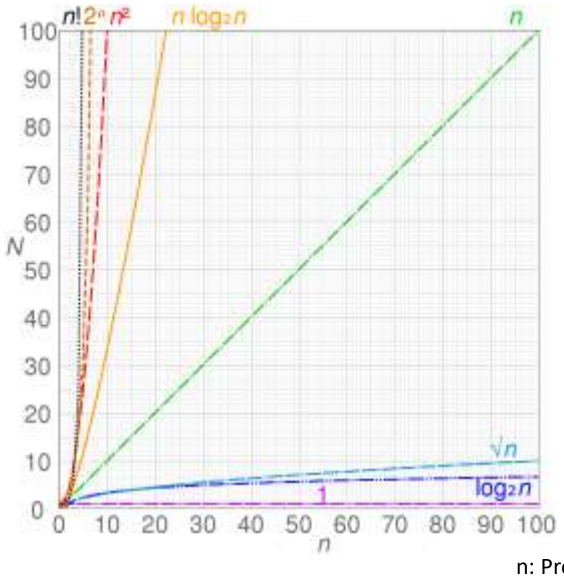
AI

Computability Theory
Can we solve this problem?

Complexity Theory
How much time/resources are required to solve the problem?

Time Complexity
The amount of computer time to solve a problem
The number of elementary operations performed to solve a problem

Time Complexity



n: Problem size
Ex: Number of elements to sort

3

Time Complexity

log_2n	n	$nlog_2n$	n^2	2^n
2	4	8	16	16
3	8	24	64	256
4	16	64	256	65,536
5	32	160	1,024	4,294,967,296
6	64	384	4,096	1.84×10^{19}
7	128	896	16,384	3.40×10^{38}
8	254	2,048	65,363	1.16×10^{77}
9	512	4,608	262,144	1.34×10^{154}
10	1024	10,240	1,048,576	1.80×10^{308}
30	1,073,741,824	32,212,254,720	1.15×10^{18}	???

A few sec.

~100 years

~10 years

4

Language & Complexity Class

KENTECH Korea Institute of Energy Technology AI

Class **R**: Problems that can be solved by a computer

Class **P**: Problems that can be solved ***efficiently*** by a computer

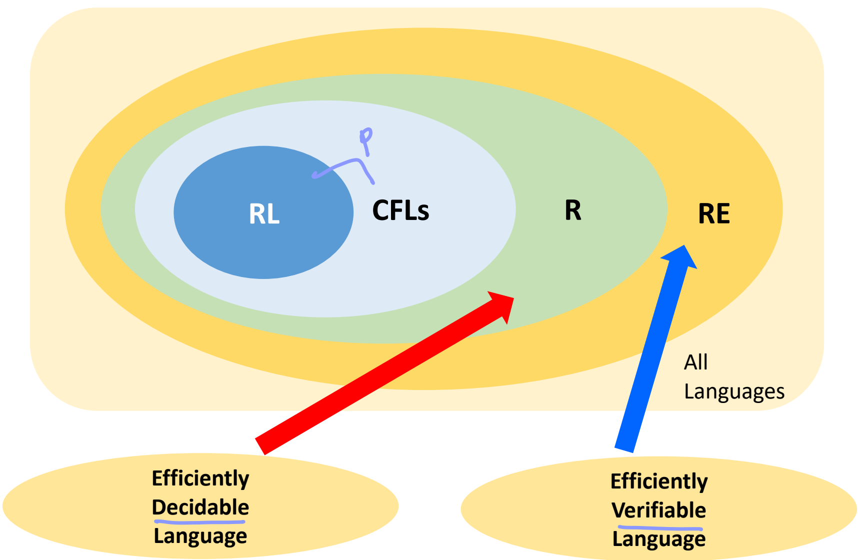
Class **RE**: Problems that where "yes" answers can be verified by a computer

Class **NP**: Problems that "yes" answers can be verified ***efficiently*** by a computer

verify
: 맞는지 아닌지 구별

$P \subset NP$

KENTECH Korea Institute of Energy Technology AI



RL
CFLs] P

Measurement of Complexity

- How do we measure the complexity of a decider, D?
  - Number of states
  - Amount of tape required
  - Number of lines in TM
  - **Amount of time required**
  - …

7

Big Oh Notation

8

Complexity Analysis

KENTECH Korea Institute of Energy Technology AI

- Consider a program that finds the largest element from a given vector

```
int vectorMax(Vector &v) {
    int currentMax = v[0];
    int n = v.size();
    for (int i=1; i < n; i++) {
        if (currentMax < v[i]) {
            currentMax = v[i];
        }
    }
    return currentMax;
}
```

- How long will it take to complete the program with a vector of n elements?

Detailed Analysis

KENTECH Korea Institute of Energy Technology AI

```
int vectorMax(Vector &v) {
    int currentMax = v[0];
    int n = v.size();
    for (int i=1;  i < n;  i++) {
        if (currentMax < v[i]) {
            currentMax = v[i];
        }
    }
    return currentMax;
}
```

2 ops

2 ops

n ops — i<n

1 op

2(n-1) ops — each 2 per ++; (n-1) times

2(n-1) ops

0 ~2(n-1) ops — if ordered descending → 0 operation (never); " ascending → 2(n-1) operations (always)

1 op

Minimum: 2+2+1+n+2(n-1)+2(n-1)+1 = 5n+2 operations
Maximum: 2+2+1+n+2(n-1)+2(n-1)+2(n-1)+1 = 7n operations

Simplified Analysis

KENTECH Korea Institute of Energy Technology AI

- In most cases, we do not need detailed analysis
- Enough to know that the time increases

Linear proportionally to n ($5n$, $7n+2$)

➔ If n increases 10 times, then time increases 10 times

complexity

important (n이 커질수록)

- Assume that exact analysis of an algorithm is $3n^2 + 6n + 500$

➔ As n doubles, the time quadruples approximately when n is large → unrealistically large n (million)

Coefficient not important

Ignore smaller polynomials

Algorithm Analysis: Bog Oh, O(·)

KENTECH Korea Institute of Energy Technology AI

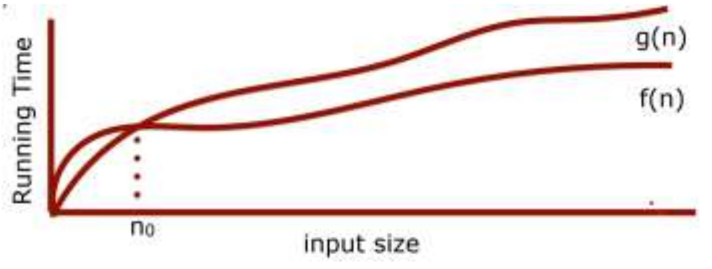
- Simplification uses a construct known as "Big-O" notation — think "O" as in "on the Order of
- "Big-O notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions."

Algorithm Analysis: Big Oh

KENTECH Korea Institute of Energy Technology AI

Mathematically (6 notations)

- Let f(n) and g(n) be functions mapping nonnegative integers to real numbers
- We say f(n) is O(g(n)) if there is a real constant c > 0 and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$



Examples

KENTECH Korea Institute of Energy Technology AI

- Detailed analysis

  5n+2 → O(n)

  10nlogn + 4n → O(nlogn)

  $0.00001n^2 + 500000n - 4000000000$ → O(n^2)

  $0.00001nlogn + 500000n - 4000000000$ → O(nlogn)

Example: Matrix Multiplication

AI

```
int matrixProduct (real A, B) {                         ⇒ O(n³)
Loop →  for (int i=0; i<n; i++) {       → O(n)
            for (int j=0; j<n; j++) {   → O(n)
                C[i][j] = 0;
                for (int k=0; k<n; k++) {   → O(n)
                    C[i][j] += A[i][k]*B[k][j];
                }
            }
        }
}
```


Efficient Algorithms: Class P

Searching Finite Space

- Many decidable problems can be solved by searching over a large but finite space of possible options
- Searching this space might take a staggeringly long time, but only finite time
- From a decidability perspective, this is totally fine!
- From a complexity perspective, this may be totally unacceptable

Efficiency

- Every person may have their own definitions of "Efficiency"
- When dealing with problems that search for the "best" solution of some sort, there are often at least exponentially many possible options
- Brute-force solutions tend to take at least exponential time to complete
  - $O(2^n)$, O(n!), etc
- Clever algorithms often run in time O(1), O(logn), O(n), O($nlogn$), or O(n^2), etc.

good

practicality

Polynomials and Exponentials

KENTECH Korea Institute of Energy Technology AI

- An algorithm runs in ***polynomial time*** if its runtime is some polynomial in n
  - O(n^k) for some constant k
- Polynomial functions "scale well."
  - Small changes to the size of the input do not typically induce enormous changes to the overall runtime
- Exponential functions scale terribly
  - Small changes to the size of the input induce huge changes in the overall runtime

Cobham-Edmonds Thesis

KENTECH Korea Institute of Energy Technology AI

- A language L can be decided efficiently if there is a TM that decides it in polynomial time
- Equivalently, L can be ***decided efficiently*** if it can be decided in time O(n^k) for some k ∈ ℕ
- What are the efficient runtimes?

$4n^2 - 3n + 137$ → $O(n^2) \in P$

10^{500} → $O(1)$ (Constant) $\in P$

2^n → $O(2^n) \notin P$

1.000000001^n → $O(2^n) \notin P$

$n^{1,000,000,000,000}$ → $\in P$

$n^{log n}$ → $O(n^{\log n}) \notin P$

Why Polynomials?

- Polynomial time somewhat captures efficient computation, but has a few edge cases
- However, polynomials have very nice mathematical properties:
  - The sum of two polynomials is a polynomial. (Running one efficient algorithm, then another, gives an efficient algorithm.)
  - The product of two polynomials is a polynomial. (Running one efficient algorithm a "reasonable" number of times gives an efficient algorithm.)
  - The composition of two polynomials is a polynomial. (Using the output of one efficient algorithm as the input to another efficient algorithm gives an efficient algorithm.)

The Complexity Class P

- The complexity class P (for polynomial time) contains all problems that can be solved in polynomial time
- Formally:

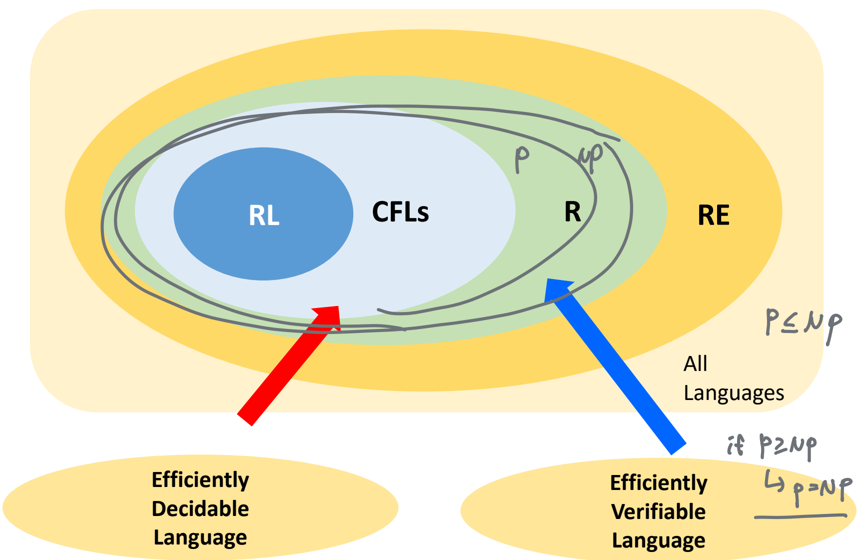
  algorithm, TM, program ...

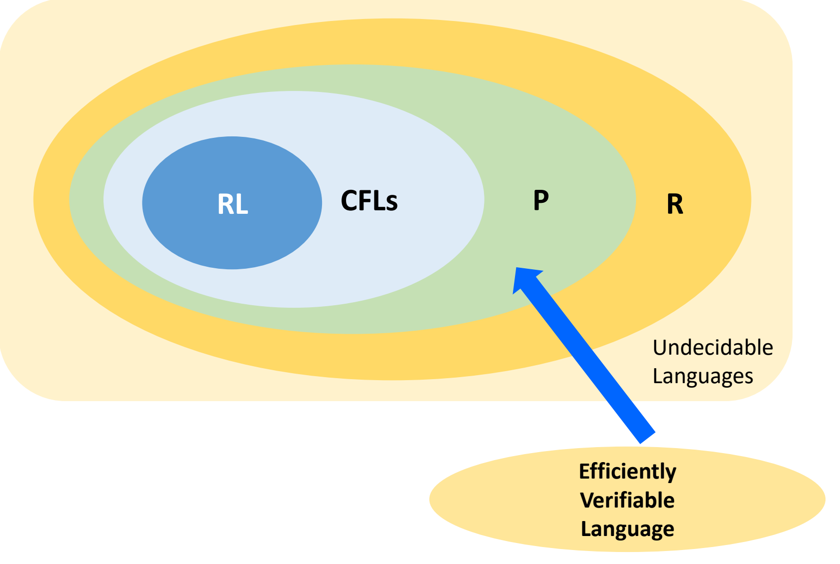
  P = { L | There is a polynomial-time decider for L }
- Assuming the Cobham-Edmonds thesis, a language is in P if it can be decided efficiently

Examples

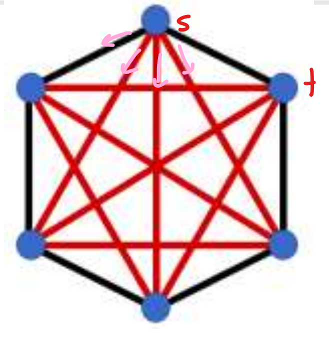
- All regular languages are in P
  - All have linear-time TMs
- All CFLs are in P
  - Requires a more nuanced argument (the CYK algorithm or Earley's algorithm)
- And a ton of other problems are in P as well

P ⊃ RL
CFLs
~~Prt~~ of R





Problems not in P



한번 방문한거 다시 방문 X

How many simple paths are there from one node to another?

$4 \times 3 \times 2 \times 1 = 4!$ → n nodes $(n-2)! = O(n!)$

List all subsets of a set that has n elements

↳ 2^n

Class NP

Verifier

KENTECH Korea Institute of Energy Technology AI

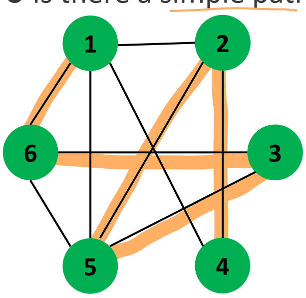
- What if you need to search a large space for a single object?
- Is there an ascending subsequence of length at least 7?

→ find subsequence in certain length → NP

9	3	11	4	2	13	5	6	1	12	7	8	0	10

- Is there a simple path that goes through every node exactly once?

→ NP



Hamiltonian Path

Verifier

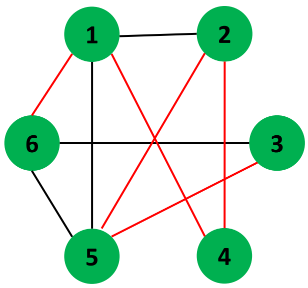
KENTECH Korea Institute of Energy Technology AI

- What if you need to search a large space for a single object?
- Is there an ascending subsequence of length at least 7?

9	3	11	4	2	13	5	6	1	12	7	8	0	10

$O(n)$

- Is there a simple path that goes through every node exactly once?



length : $n-1$

Polynomial Verifiers

KENTECH Korea Institute of Energy Technology AI

- Recall that a verifier for L is a TM V such that
  - V halts on all inputs
  - w ∈ L iff ∃c ∈ Σ*. V accepts ⟨w, c⟩
- A polynomial-time verifier for L is a TM V such that
  - V halts on all inputs
  - w ∈ L iff ∃c ∈ Σ*. V accepts ⟨w, c⟩
  - V's runtime is a polynomial in |w| (that is, V's runtime is $O(|w|^k)$ for some integer k)

The Complexity Class NP

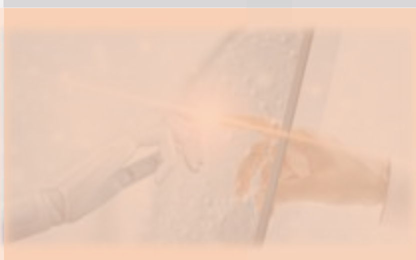
AI

- The complexity class NP (***n***ondeterministic ***p***olynomial time) contains all problems that can be verified in polynomial time
- Formally:

  NP = { L | There is a polynomial-time verifier for L }

  ~~decider~~

- The name NP comes from another way of characterizing NP
- If you introduce nondeterministic Turing machines and appropriately define "polynomial time," then NP is the set of problems that an NTM can solve in polynomial time



P=NP ???

P = NP

KENTECH Korea Institute of Energy Technology AI

- The **most important** problem in theoretical Computer Science

- P=NP; The Million-Dollar Questions

The Clay Mathematics Institute has offered a $1,000,000 prize to anyone who proves or disproves P = NP

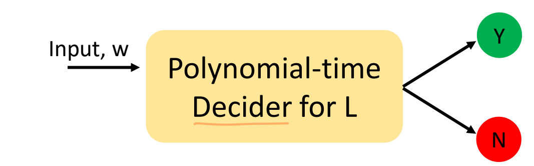
One of seven millennium problems

2 cases [P is NP
P is not NP

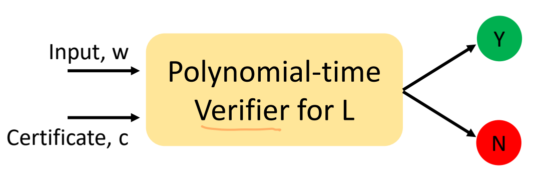
P = NP

KENTECH Korea Institute of Energy Technology AI

- P = { L | There is a polynomial-time ***decider*** for L }



- NP = { L | There is a polynomial-time ***verifier*** for L }



Obviously, $\boxed{P \subseteq NP}$

P is subset of NP

P = NP ?

- With the verifier definition of NP, one way of phrasing this question is

If a solution to a problem can be checked (verified) efficiently, can that problem be solved efficiently?

- An answer either way will give fundamental insights into the nature of computation

35

Why This Matters

if N=P is solved, problems solved efficiently

- The following problems are known to be efficiently verifiable, but have no known efficient solutions:
  - Determining whether an electrical grid can be built to link up some number of houses for some price (Steiner tree problem)
  - Determining whether a simple DNA strand exists that multiple gene sequences could be a part of (shortest common supersequence)
  - Determining the best way to assign hardware resources in a compiler (optimal register allocation)
  - Determining the best way to distribute tasks to multiple workers to minimize completion time (job scheduling)
  - And ***many more***

36

SOTA

- Resolving P $\stackrel{?}{=}$ NP has proven extremely difficult

- In the past 50 years:
  - Not a single correct proof either way has been found
  - Many types of proofs have been shown to be insufficiently powerful to determine whether P $\stackrel{?}{=}$ NP

- Most(?) computer scientists believe P ≠ NP