Lecture 1:
Basic of Algorithm & Computational Complexity

김수경

이화여자대학교 인공지능융합전공 소속

저작권 안내

(주)업스테이지가 제공하는 모든 교육 콘텐츠의 지식재산권은 운영 주체인 (주)업스테이지 또는 해당 저작물의 적법한 관리자에게 귀속되어 있습니다.

콘텐츠 일부 또는 전부를 복사, 복제, 판매, 재판매 공개, 공유 등을 할 수 없습니다.
유출될 경우 지식재산권 침해에 대한 책임을 부담할 수 있습니다.

유출에 해당하여 금지되는 행위의 예시는 다음과 같습니다.

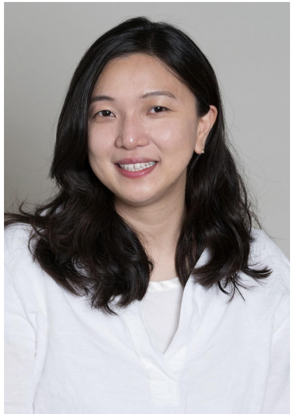
- 콘텐츠를 재가공하여 온/오프라인으로 공개하는 행위
- 콘텐츠의 일부 또는 전부를 이용하여 인쇄물을 만드는 행위
- 콘텐츠의 전부 또는 일부를 녹취 또는 녹화하거나 녹취록을 작성하는 행위
- 콘텐츠의 전부 또는 일부를 스크린 캡쳐하거나 카메라로 촬영하는 행위
- 지인을 포함한 제3자에게 콘텐츠의 일부 또는 전부를 공유하는 행위
- 다른 정보와 결합하여 Upstage Education의 콘텐츠임을 알아볼 수 있는 저작물을 작성, 공개하는 행위
- 제공된 데이터의 일부 혹은 전부를 Upstage Education 프로젝트/실습 수행 이외의 목적으로 사용하는 행위

강사 소개

김수경

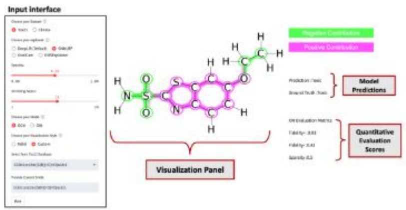
現 이화여자대학교 인공지능융합전공 소속 조교수

parc
Part of SRI International

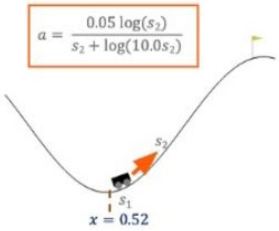


관심 연구 분야

- Explainable AI (XAI)
- NLP and Medical AI
- AI for Science
- RL based Optimization



Explainable AI



RL based Optimization

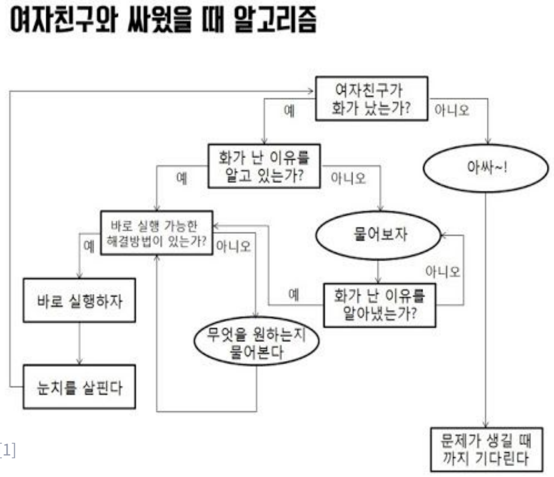
01

Algorithms & Complexity

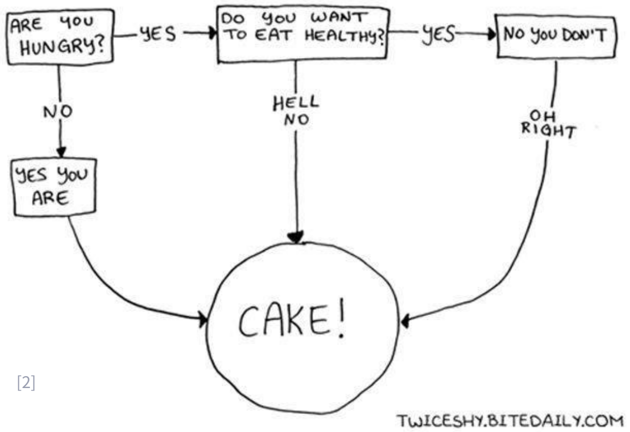
upstage Education

What is Algorithm?

- **Computational procedure** to solve a problem



[1]



[2]

[1] https://blog.naver.com/sumr2002/220359932991
[2] http://jokesandfun.de/infographic/the-cake-is-a-factflowchart/

Efficiency of an Algorithm

Do you like fast/efficient computer program, or slow one?

[1]

[2]

[3]

Of course, we always expect our computers to do their jobs **most efficiently**!

[1] https://www.vecteezy.com/photo/20621853-stressed-and-overworked-businessman
[2] https://stock.adobe.com/search?k=smashed+computer&asset_id=52541504
[3] https://www.istockphoto.com/kr/%EC%82%AC%EC%A7%84/%EC%97%AC%EC%9E%90-%EC%BB%B4%ED%93%A8%ED%84%B0-gm118986833-12293508

Computation Complexity

- Cost of algorithm = Sum of operation costs

- Model of computation specifies
  - What **operations** an algorithm is allowed to use
  - Cost (time, space) of each operation
- Execution costs
  - **Time** complexity of a program: how much **time**?
  - **Space** complexity of a program: how much **memory**?

upstage Education

Measuring Time Complexity

- **Measure execution time** in seconds using a client program (*e.g.*, time module)
    - (+) Easy to measure
    - (+) Gives actual time
    - (-) Large amounts of time might be required.
    - (-) Results depend on lots of factors (machine, compiler, data…)

- Count the number of operations **in terms of input size** N
    - (+) Machine independent.
    - (+) Gives algorithm's **scalability.**
    - (-) Tedious to compute…
    - (-) Does not give actual time.

- Fortunately, we care only about asymptotic behavior (with a very large N – Big Data!)

Elementary School Algorithm

Example: Integer multiplication

- Input: two *N*-digit numbers *x*, *y*
- Output: product of *x* and *y*
- Primitive operations allowed:
  - Add 2 single-digit numbers
  - Multiply 2 single-digit numbers

$$
\begin{array}{r}
2\,3\,3 \\
5678 \\
\times 1234 \\
\hline
22712 \\
17034 \\
11356 \\
5678 \\
\hline
7006652
\end{array}
$$

Elementary School Algorithm

Example: Integer multiplication

- Input: two *N*-digit numbers *x*, *y*
- Output: product of *x* and *y*
- Primitive operations allowed:
  - Add 2 single-digit numbers
  - Multiply 2 single-digit numbers

How many primitive operations used?

```
      2 3 3
       5678
     × 1234
   --------
      22712
     17034
    11356
   5678
   --------
    7006652
```

For each row:

***N* multiplications**

(up to) ***N*-1 additions**

***N* rows**

In total,

***N*(2*N*-1) operations**

(up to) ***N*^{2} additions**

Total operations $\leq 3N^2$

upstage Education

Software Engineer's Example

```
def linear_search(list, value):
  for i in range(len(list)):
    if list[i] == value:
      return i
  return -1
```

```
def selection_sort(list):
  for i in range(len(list)):
    smallest = i
    for j in range(i+1, len(list)):
      if list[j] < list[smallest]:
        smallest = j
    list[i], list[smallest] = list[smallest], 
list[i]
```

Let's denote `len(list)` as *N*:

Operation	Count
==	*1 to N*

Operation	Count
smallest = i	N
<	$(N^2 - N)/2$
smallest = j	0 to $(N^2 - N)/2$
swap	N

02

Big O Notation

How to Characterize Time Complexity more formally and simply?

Simplification of Time Complexity

1. We care only about the worst-case performance!

← because we do not know what input data we will get in advance.

Operation	Count
smallest = i	N
<	$(N^2 - N)/2$
smallest = j	0 to $(N^2 - N)/2$
swap	N

upstage Education

Simplification of Time Complexity

2. Focus only on a single operation with the highest order of growth (=most expensive).

- There may be multiple good choices. Then, just choose any of them.

Operation	Count
smallest = i	$\cancel{N}$
<	$\cancel{(N^2 - N)/2}$
smallest = j	$(N^2 - N)/2$
swap	$\cancel{N}$

upstage Education

Simplification of Time Complexity

3. Remove lower order terms.

← They do not really matter, since the higher order one dominates.

Operation	Count
smallest = i	*ignored*
<	*ignored*
smallest = j	$(N^2 \cancel{- N})/2$
swap	*ignored*

Simplification of Time Complexity

4. Remove constants.

← We have already thrown away information at step 2. At this stage, constants are not meaningful.

Operation	Count
smallest = i	*ignored*
<	*ignored*
smallest = j	$(N^2)/\cancel{2}$
swap	*ignored*

Worst-case **order of growth**: N^2

upstage Education

Formal Definition

- If a function $T(N)$ has its **order of growth** less than or equal to $f(N)$, we write this as $T(N) \in O(f(N))$, where O is called Big-O notation.

- More mathematically, $T(N) \in O(f(N))$ means that there exists a positive constant c such that $T(N) \leq c\,f(N)$ for all values of N greater than some positive N_0 (*i.e.*, large N).

upstage Education

Example

- $T(N) = N^2 + 5N^5$
- $T(N) \in O(N^5)$?
  - That is, is there a positive constant c such that $N^2 + 5N^5 \le cN^5$ for large N?
  - Yes!
    - $N^2 + 5N^5 < N^5 + 5N^5 = 6N^5$
    - With $c = 6$, it holds.
- $T(N) \in O(N^7)$?
  - That is, is there a positive constant c such that $N^2 + 5N^5 \le cN^7$ for large N?
  - Yes!
    - $N^2 + 5N^5 < N^5 + 5N^5 = 6N^5 < 6N^7$
    - With $c = 6$, it holds.

upstage Education

Example(cont'd)

- $T(N) = N^2 + 5N^5$
- $T(N) \in O(N^4)$?
  - That is, is there a positive constant c such that $N^2 + 5N^5 \leq cN^4$ for large N?
  - No 😕
    - Even if we set very large c, still for $N > c$, $5N^5 > N^5 > cN^4$.
- We are usually interested in the **tightest** order; in this example, $O(N^5)$.

upstage Education

Exercise(cont'd)

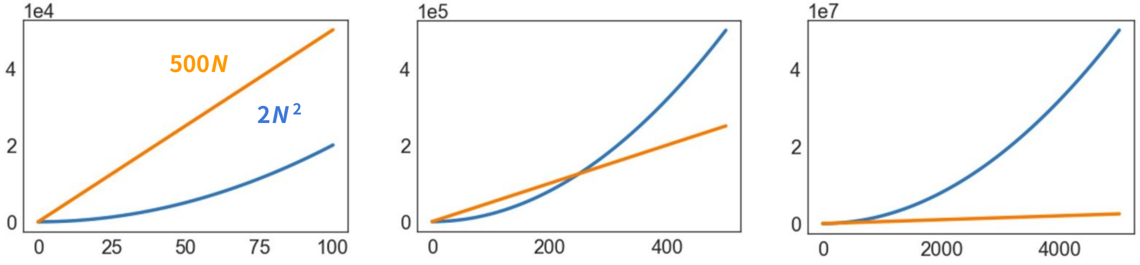
- $T(N) = 2N^2 + 3N$ **$O(N^2)$**
- $T(N) = 1/N + 100$ **$O(1)$**
- $T(N) = 100\cos(N) + 50N^2$ **$O(N^2)$**
- $T(N) = \log N + 2N$ **$O(N)$**
- $T(N) = 2^N + N^2$ **$O(2^N)$**

upstage Education

What is Important for Asymptotic Analysis?

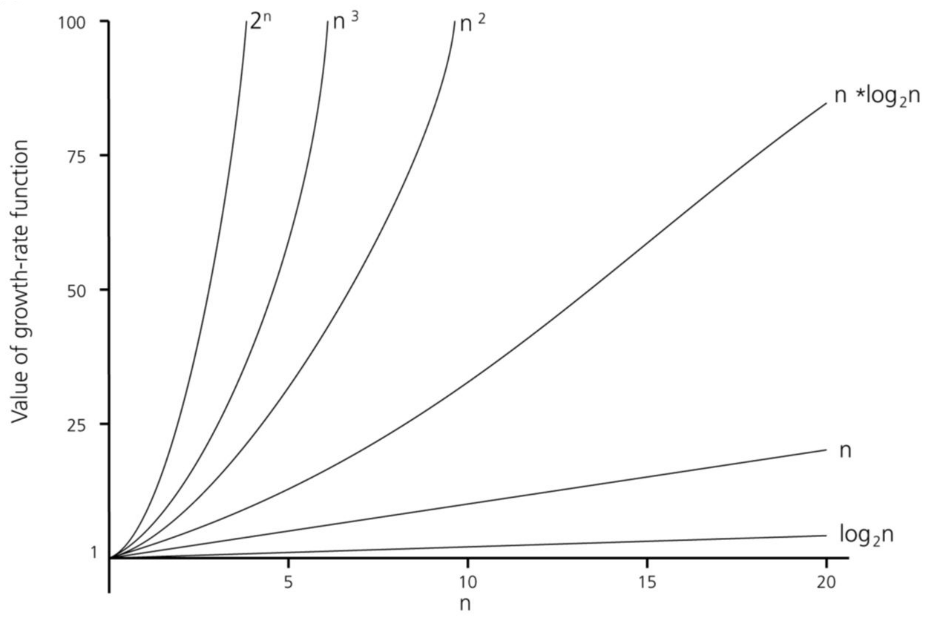
- Compare the two algorithms below:
  - Algo1 requires $\mathbf{2N^2}$ operations, while
  - Algo2 requires $\mathbf{500N}$ operations.
  - Algo1 is faster than Algo2 for a small N, but becomes much slower for a very large N.
  - What is important? **How fast function is growing**!

- Order of growth:



upstage Education

Asymptotic Analysis



Building intelligence for the future of work