Lecture #4: Divide and Conquer

School of Computer Science and Engineering
Kyungpook National University (KNU)

Woo-Jeoung Nam

Matrix multiplication

- **Problem**
  - If $A=a_{ij}$ and $B=b_{ij}$ are square **n x n** matrices, then compute $C=A \cdot B$
  - In C, we compute entry c_{ij}, for i,j=1,2,…n by $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$

$$
\begin{array}{cc}
 & n \\
n & \begin{bmatrix} c_{11} & c_{12} & & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ & \dots & & \\ c_{n1} & c_{n2} & & c_{nn} \end{bmatrix}
\end{array}
=
\begin{bmatrix} a_{11} & a_{12} & & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & \dots & & \\ a_{n1} & a_{n2} & & a_{nn} \end{bmatrix}
\times
\begin{bmatrix} b_{11} & b_{12} & & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ & \dots & & \\ b_{n1} & b_{n2} & & b_{nn} \end{bmatrix}
$$

  - Input: $A=a_{ij}$ and $B=b_{ij}$ where i,j=1,2,…n
  - Output: $C= c_{ij} = C=A \cdot B$

Matrix multiplication

- **Pseudocode**

MAT_MUL(A, B)

n=A.rows	c → 상수
let C be a new n x n matrix	c
for i = 1 to n	cn
for j = 1 to n	cn^2
$c_{ij}=0$	cn^2
for k = 1 to n	cn^3
$c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$	cn^3
return C	

Running time = $O(n^3)$

$\Omega(\quad)$ $\Theta(n^3)$

Matrix multiplication

- **Divide and conquer for MAT_MUL**
  - **Create smaller subproblems by dividing a n x n matrix into four n/2 x n/2 matri ces**
  - **Split subproblems until base case, where n=1**

$$\underbrace{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}_{B}$$

Matrix multiplication

- **Divide and conquer for MAT_MUL**
  - Create smaller subproblems by dividing a n x n matrix into four n/2 x n/2 matrices
  - Split subproblems until base case, where n=1

1/1 → 더 나눌 수 없는 사례

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Rewrite $C = A \cdot B$ as

$A_{11}B_{11} + A_{12}B_{21}$

m k n m

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

giving the four equations

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned}$$

8 mults of $(n/2) \times (n/2)$

4 adds of $(n/2) \times (n/2)$

분할정복

Matrix multiplication

$\theta(1) + 8T(\frac{n}{2}) + n^2$

by master theorem

$8T(\frac{n}{2}) + n^2$

- **Divide and conquer for MAT_MUL (Pseudo code)**

```
MAT_MUL_DC(A, B)
n=A.rows                                               c
let C be a new n x n matrix                            c
if n==1 (base case=O(1))                               c
    c11=a11 · b11                                      c
else partition A, B and C into four (n/2) matrices     c
    C11= MAT_MUL_DC(A11, B11) + MAT_MUL_DC(A12, B21)   2T(n/2)
    C12= MAT_MUL_DC(A11, B12) + MAT_MUL_DC(A12, B22)   2T(n/2)
    C21= MAT_MUL_DC(A21, B11) + MAT_MUL_DC(A22, B21)   2T(n/2)
    C22= MAT_MUL_DC(A21, B21) + MAT_MUL_DC(A22, B22)   2T(n/2)
```

$8T(n/2)$

$\frac{n^2}{4}$ ×4 $= n^2$

$O(n^{\log 8}) = O(n^3)$

$$T(n) = \Theta(1) + 8T\left(\frac{n}{2}\right) + n^2$$

* data points in each submatrix = $n^2/4$

* so, summation of the matrices = $O(n^2)$

Faster than the standard method?

$$= 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$= \Theta(n^{lg8}) \text{ (by master theorem, next class)}$$

$$= \Theta(n^3)$$

$T(n) = 8T(\frac{n}{2}) + n^2 + \theta(1)$

$= 8T(\frac{n}{2}) + \theta(n^2)$

➢ 결과적으로 일반 곱셈보다 더 빠르다고 할 수 없다

$\theta(n^{\log_2 8})$

$\theta(n^3)$

$8 > 2^2$

Matrix multiplication- Strassen's method

- 재귀트리를 덜 무성하게 만들자
  - By some proof (pg 80), instead of 8 recursive multiplications of n/2 x n/2 matrices, only 7 multiplications are done (however, as a tradeoff, some additional summations are required – which is at least better than an extra multiplication)
  - 쉽게 말해서, 행렬의 곱셈을 1번 줄이는 대신 행렬의 덧셈을 여러 번 수행하면 더 빨리질 수 있다

재귀 트리를 덜 무성히

"7번의 곱셈"

Matrix multiplication- Strassen's method

- 재귀트리를 덜 무성하게 만들자

$$\underset{C}{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}} = \underset{A}{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}} \times \underset{B}{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}$$

- **Create 10 matrices -> create 7 matrices**

$$\begin{aligned}
S_1 &= B_{12} - B_{22}\,, \\
S_2 &= A_{11} + A_{12}\,, \\
S_3 &= A_{21} + A_{22}\,, \\
S_4 &= B_{21} - B_{11}\,, \\
S_5 &= A_{11} + A_{22}\,, \\
S_6 &= B_{11} + B_{22}\,, \\
S_7 &= A_{12} - A_{22}\,, \\
S_8 &= B_{21} + B_{22}\,, \\
S_9 &= A_{11} - A_{21}\,, \\
S_{10} &= B_{11} + B_{12}\,.
\end{aligned}$$

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22}\,, \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22}\,, \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11}\,, \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11}\,, \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}\,, \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}\,, \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}\,.
\end{aligned}$$

Matrix multiplication- Strassen's method

- 재귀트리를 덜 무성하게 만들자

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

C A B

- 이제 C를 다음과 같이 계산 가능

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 , \\ C_{12} &= P_1 + P_2 , \\ C_{21} &= P_3 + P_4 , \\ C_{22} &= P_5 + P_1 - P_3 - P_7 . \end{aligned}$$

$7T(n/2) + \Theta(n^2)$

θ by master theorem

$\Theta(\lg n)$

Matrix multiplication- Strassen's method

- **The cost for Strassen's method**
  - Step 1: Dividing A and B into (n/2) x (n/2) matrices $O(1)$
  - Step 2: Computing P1 to P7 $7T\left(\frac{n}{2}\right) + \Theta(n^2)$
  - Step 3: Computing the C matrix (C11, …, C22) $\Theta(n^2)$

$7T(\frac{n}{2}) + \Theta(n^2) + \Theta(n^2) + \Theta(1)$

Matrix multiplication- Strassen's method

- **The cost for computing P1 to P7**
  - P1 = A11 · (B12 - B22)
  - P2 = (A11 + A12) · B22
  - P3 = (A21 + A22) · B11
  - P4 = A22 · (B21 - B11)
  - P5 = (A11 + A22) · (B11 + B22)
  - P6 = (A12 - A22) · (B21+B22)
  - P7 = (A11 - A21) · (B11 + B12)

\# matrix multiplications = 7 $\quad 7T(n/2)$

\# matrix add/subs = 10 $\quad \Theta(n^2)$

"$M_1 \sim M_7$" 을 만들기

10번의 덧셈

$7T(n/2)$

Matrix multiplication- Strassen's method

- So, the recurrence for the running time T(n) will be

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7T\left(\frac{n}{2}\right) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

$$\Theta\left(n^{lg7}\right) = \Theta\left(n^{2.81}\right)$$

- 아직 O(n^2)알고리즘은 없음… 밝히지 못한것인가?
- 불가능 하다는 것도 밝힌 사람은 없다

K-selection 알고리즘

- 입력 배열에서 k번째로 작은 값을 찾는 알고리즘
- SELECT(A,k):
  - n: length of array A, k는 {1,…,n}의 원소

A

7	4	3	8	1	5	9	14

- SELECT(A, 1) = 1
- SELECT(A, 2) = 3
- SELECT(A, 3) = 4
- SELECT(A, 8) = 14

- SELECT(A, 1) = MIN(A)
- SELECT(A, n/2) = MEDIAN(A)
- SELECT(A, n) = MAX(A)

편의를 위해서,
배열의 모든 원소들은 **고유한 값**을 갖는다고 가정합시다!

K-selection 알고리즘

- **How to solve?**
  - Intuitive approach: Sort -> find!
  - 우리는 Merge sort를 배웠다
- **SELECT(A,k):**
  - A=MergeSort(A)
  - Return A[k-1]
- 병합정렬의 시간복잡도는 O(nlogn)
- 끝났네!
- 혹시 더 잘 풀수 있나?

K-selection 알고리즘

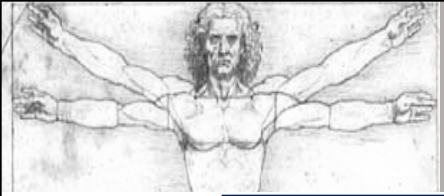
- 발상 #1: SELECT(A, 1)의 시간 복잡도는? (k=1로 고정된 문제)
  - MIN(A)와 동일한 기능을 하는 함수가 된다.
- MIN(A):

```
ret = ∞
For i=0, ..., n-1:
    If A[i] < ret:
        ret = A[i]
Return ret
```

O(1)

바깥의 반복문은 O(n) 반복

- 즉, SELECT(A, 1)의 시간 복잡도는 O(n).



K-selection 알고리즘

- i번째 문제를 해결하면 -> i+1번째 경우의 문제로 확장
  - ➢ '일반화'를 하는 과정을 통해 문제의 실마리를 찾아보자.
- SELECT2(A): 두번째로 작은

```
ret = ∞
minSoFar = ∞
For i=0, .., n-1:
    If A[i] < ret and A[i] < minSoFar:
        ret = minSoFar
        minSoFar = A[i]
    Else if A[i] < ret and A[i] >= minSoFar:
ret = A[i]
Return ret
```

둘 다 작아야 한다.

→ minSoFar 만

보다는 크다, 그럼 ret만 업데이트

초기: ret ∞ 3

minSoFar. 3

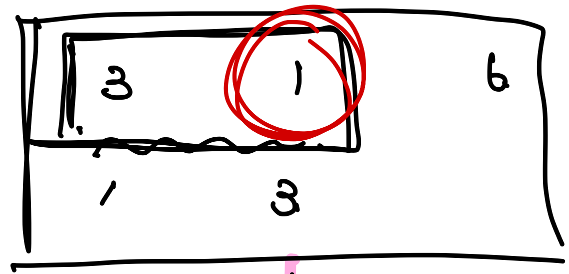
3 1 2 5 7

- 아직 까지는 O(n)이다

K-selection 알고리즘

- **SELECT(A, n/2), 즉, MEDIAN(A) 문제로 확장하면?**

- MEDIAN(A):
  - ret = ∞
  - minSoFar = ∞
  - secondMinSoFar = ∞
  - thirdMinSoFar = ∞
  - fourthMinSoFar = ∞
  - ....

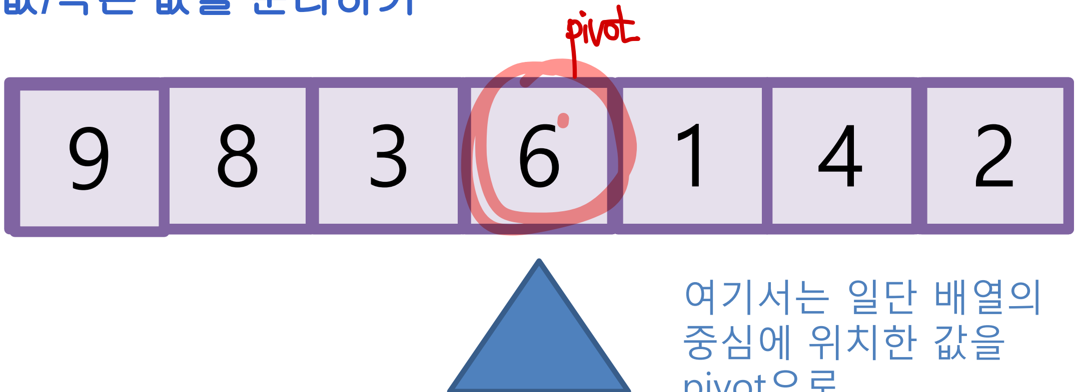


삽입 정렬문제로 치환됨

- **매우 큰 k에 대해서는 그리 좋은 접근법은 아님.. ($k = n/2, n$)**
- **이런 형태는 사실상, "삽입 정렬"의 형태로 문제가 치환되는 셈.**
- **시간복잡도: $O(n^2)$**

K-selection 알고리즘

- 아이디어: 분할 정복 사용하기!
- SELECT(A, k)에 대해서,
  - 1) "피봇(pivot)" 값 선택하기
  - 2) "피봇" 값을 기준으로,
    큰 값/작은 값을 분리하기



여기서는 일단 배열의
중심에 위치한 값을
pivot으로

A[pivot]보다 작은 값들이 모인 배열 L

A[pivot]보다 큰 값들이 모인 배열 R

K-selection 알고리즘

- 아이디어: 분할 정복 사용하기!
- SELECT(A, k)에 대해서,
  - 1) “피봇(pivot)” 값 선택하기
  - 2) “피봇” 값을 기준으로,
    큰 값/작은 값을 분리하기

k=5

6

여기서는 일단 배열의
중심에 위치한 값을
pivot으로

A[pivot]보다 작은 값들이 모인 배열 L

3	1	4	2

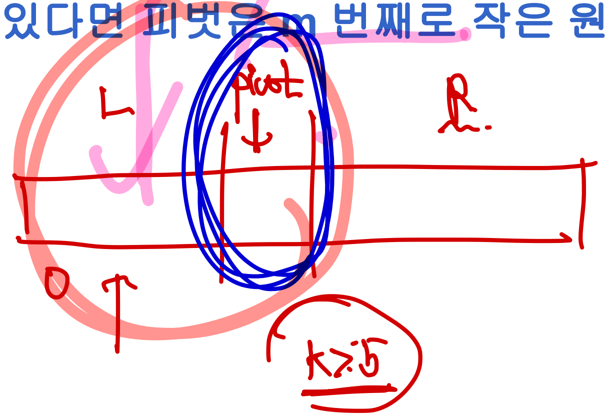
A[pivot]보다 큰 값들이 모인 배열 R

9	8

K-selection 알고리즘

- 원소들을 정렬하였을 때 피벗이 정확히 맨 앞에서 부터 몇 번째에 위치하는지 알 수 있다
  - ➢ 다시 말해 피벗 앞에 원소가 m−1 개 있다면 피벗은 m 번째로 작은 원소가 됨
- M 과 k가 일치하면 끝난다

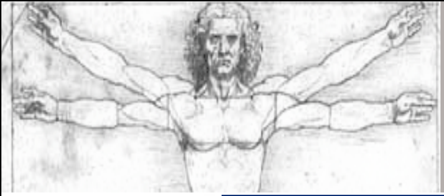
If k = 5 = len(L) + 1:
    A[pivot]을 반환한다.
If k < 5:
    SELECT(L, k)를 반환한다.
If k > 5:
    SELECT(R, k – 5)를 반환한다



k = len(L)+1 → 찾고자 하는 값.
k < 5 → select(L, k)
k > 5 → select(R,

이제 재귀호출로 구현할 수 있는 형태의 알고리즘이 구성되었다!

하지만 여전히 pivot 값을 어떻게 고르는 것이 좋을지는 생각해봐야 한다…



K-selection 알고리즘

- Select(A,k):

```
If len(A) <= 25:
        A = MergeSort(A)
        Return A[k-1]
p = getPivot(A)
        L, pivotVal, R = Partition(A,p)
if len(L) == k-1:
        return pivotVal
Else if len(L) > k-1:
        return Select(L, k)
Else if len(L) < k-1:
        return Select(R, k – len(L) – 1)
```

case1 : k번째 값을 찾음.

k-1

k-(len L+1)

Base Case: If len(A) = O(1), then any sorting algorithm runs in time O(1).

Case 1: We got lucky and found exactly the k'th smallest value!

Case 2: The k'th smallest value is in the first part of the list

Case 3: The k'th smallest value is in the second part of the list

- **getPivot**(A) 은 적당한 피봇 값을 골라서 반환해준다.
- **Partition**(A,p) 은 배열 A를 L, A[p], R로 분할해준다.

K-selection 알고리즘

- "시간은 얼마만큼 소요되는가?": 지금 우리에게 제일 중요한 문제

→ pivot 을 기준으로 partition 하는 작업

- $$T(n) = \begin{cases} T(\mathbf{len(L)}) + O(n) & \mathbf{len(L)} > k-1 \\ T(\mathbf{len(R)}) + O(n) & \mathbf{len(L)} < k-1 \\ O(n) & \mathbf{len(L)} = k-1 \end{cases}$$

$T(n) =$

- **len(L) 과 len(R) 은 어떻게 정할 수 있을까?**
  - ➢ 피봇 값을 어떻게 고르느냐에 따라서 달라질 수 있다. → median of medians
  - ➢ '좋은' 피봇 값과 '나쁜' 피봇 값은 어떤 것일까?

→ len(L), len(R) 이 가변이다

$$T(n) = \begin{cases} T(\mathrm{len}(L)) + O(n) & \mathrm{len}(L) > k-1 \\ T(\mathrm{len}(R)) + O(n) & \mathrm{len}(L) < k-1 \\ O(n) & \mathrm{len}(L) = k-1 \end{cases}$$

$$T(n) = \begin{cases} \\ T\left(\frac{n}{2}\right) + O(n) \end{cases}$$

K-selection 알고리즘

- 이상적인 피벗값
  - 정확히 배열을 반으로 나눌 수 있는 median 값
  - $N + 2/N + 4/N + \cdots + 1 = N(1 + 1/2 + 1/4 + \cdots) = 2N$
- 마스터 정리를 사용해보면
  - Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

$$T(n) = \begin{cases} O\left(n^d \log(n)\right) & \text{if } a = b^d \\ O\left(n^d\right) & \text{if } a < b^d \\ O\left(n^{\log_b(a)}\right) & \text{if } a > b^d \end{cases}$$

- $T(n) \leq T(n/2) + O(n)$
- 따라서, $a = 1, b = 2, d = 1$ 로 두면,
  - 이 경우 시간복잡도는 $O(n)$

정확히 반

0 median N

$T(n/2) + O(N)$

partition 복잡도.

$O(end - start)$

1/4 3/4로 나눠도

최악의 상황 가정

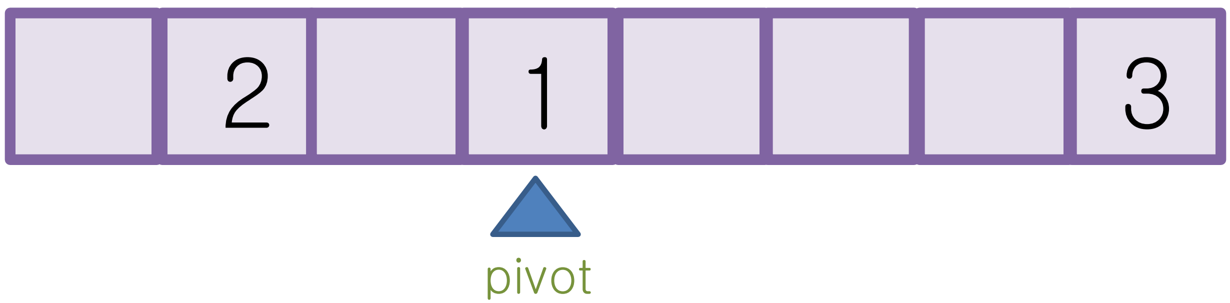
$N + \left(\frac{3}{4}\right) + \left(\frac{3}{4}\right)^2 + \cdots = \frac{1}{1-\frac{3}{4}} N = 4N$

K-selection 알고리즘

- **최악의 피벗값**
  - ‘t번째 pivot을 고를 때, 반드시 그 배열에서 가장 작은 값’을 골라주는 경우
  - $N + (N-1) + (N-2) + \cdots = O(N^2)$

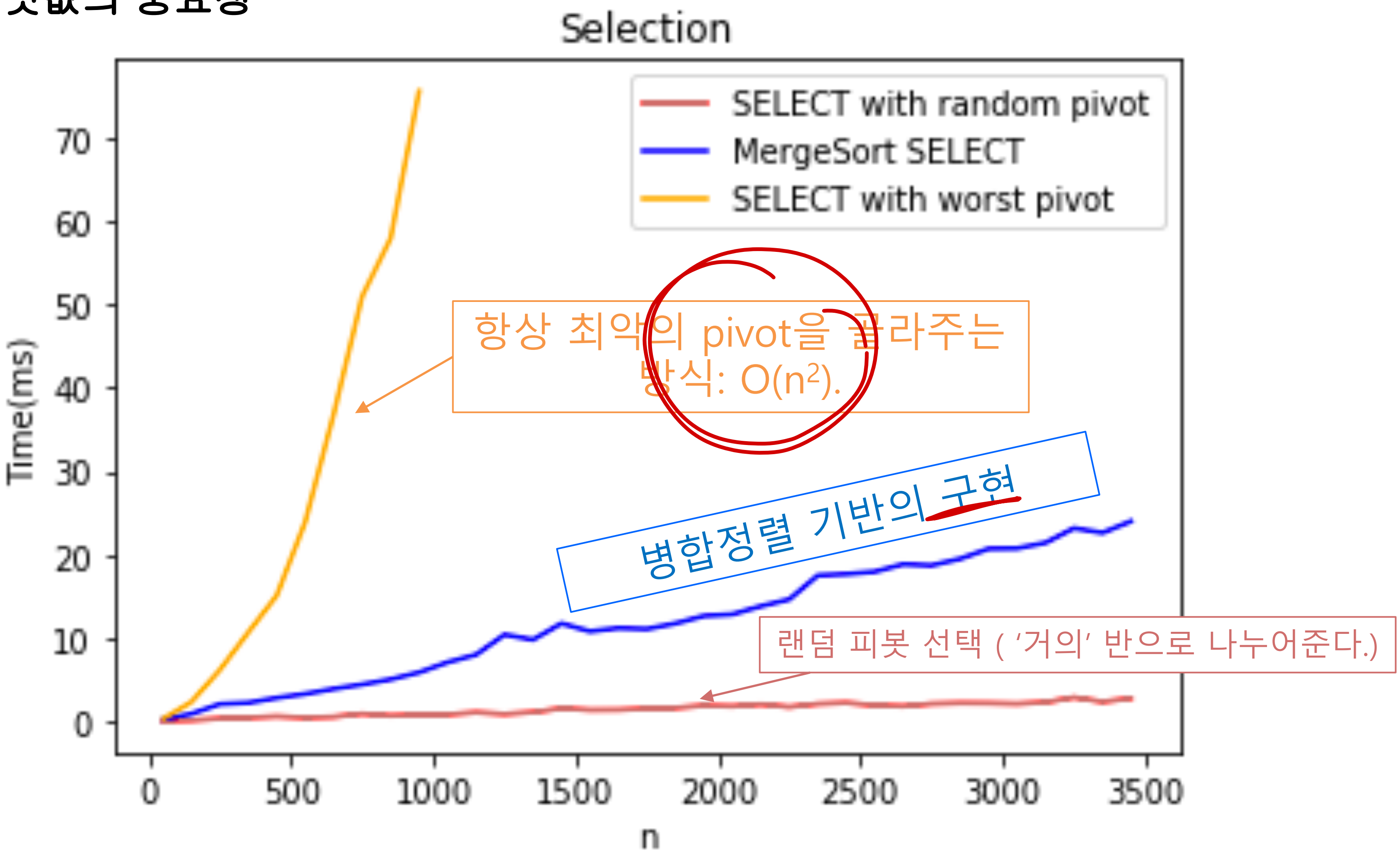
최악의 경우

$O(N^2)$
↓
최악의 경우



K-selection 알고리즘

- 피벗값의 중요성



K-selection 알고리즘

- 좋은 피벗값 고르기
- 임의의 값 (랜덤)으로?

→ 거의 100% $O(N)$으로 실행됨을 보장

→ 대략 $2n(1+\ln 2+o(1))$로 실행됨이 알려져 있음

  - 난수를 항상 '최악의 pivot'을 고르도록 누군가 조작해서 생성하지 않는다면.. 그럭저럭 괜찮은 방법이다.
  - 하지만, '항상' 적절한 결과를 도출한다고 보기는 힘들다.
  - 그래도, '대부분의 경우', '충분히 큰 n에 대해서'는 그럭저럭 잘 동작한다.
    - 경험론적인 결론

- 운에 맡기기 보다는, 확실한 방법은 없을까?

K-selection 알고리즘

- 좋은 피벗값 고르기 – 짱구를 굴려보자
- 접근법
  - 1) '이상적인 pivot은 무엇일까?'에 대한 문제를 파악해보기
    - 하지만, 항상 절대적인 '최선'의 방식은 존재하지 않을 것 같다..
      - 왜?
  - 2) 그렇다면, 차선책으로 '현실적으로 제법 괜찮은' pivot 고르기 방식을 고찰
    - 하지만, 마찬가지로 당장에는 아이디어가 떠오르지 않는다…

K-selection 알고리즘

- 좋은 피벗값 고르기 – 짱구를 굴려보자
- 1) '이상적인 pivot은 무엇일까?'
  - 이상적인 pivot: 항상 입력 값을 (배열) 반으로 나누어 줄 수 있는 pivot
    - 즉, SELECT(A, n/2)를 찾아서 써먹을 수 있게 해주는 pivot!
- 2) 차선책: '현실적으로 제법 괜찮은' pivot 고르기
  - "Approximation": 근사적 해를 구할 수 있는 방법 찾기
    - 꼭 절반까지는 아니더라도, 대충 절반 '가량' 분할 할 수 있는 방법
    - 1) 보다는 훨씬 쉽고 현실적인 문제

K-selection 알고리즘

- **Medians of medians**
  - 1979 년에 5 명의 걸출한 컴퓨터 과학자들 (Blum, Floyd, Pratt, Rivest, Tarjan) 에 의해 Medians of medians 라는 알고리즘이 발표됨
  - 줄여서 BFPRT라고 불리기도 함
  - 중간값을 정확히 찾지는 않지만 중간값과 근접한 값을 찾아주는 알고리즘
    - 정확히 말하자면, 데이터 셋에서 상위 30% 에서 70% 사이의 값을 언제나 찾아줌.
    - 피벗 찾기에 적용할 경우 셋의 크기가 매번 최소 7/10으로 줄어들기 때문에 이 정도만 되도 괜찮습니다

BFPRT!

최 7/10로 줄여줌.

K-selection 알고리즘

- **Medians of medians**
  - 1979 년에 5 명의 걸출한 컴퓨터 과학자들 (Blum, Floyd, Pratt, Rivest, Tarjan) 에 의해 Medians of medians 라는 알고리즘이 발표됨
  - 줄여서 BFPRT라고 불리기도 함
  - 중간값을 정확히 찾지는 않지만 중간값과 근접한 값을 찾아주는 알고리즘
    - 정확히 말하자면, 데이터 셋에서 상위 30% 에서 70% 사이의 값을 언제나 찾아줌.
    - 피벗 찾기에 적용할 경우 셋의 크기가 매번 최소 7/10으로 줄어들기 때문에 이 정도만 되도 괜찮습니다
  - 동작과정
    - 입력 배열을 크기가 일정한 부분 배열들로 나눕니다. 이때, 각 부분 배열의 크기는 일정한 값을 가집니다. 이 예시에서는 5로 설정합니다.
    - 각 부분 배열에서 중앙값(median)을 계산합니다. 이때, 중앙값을 계산하기 위해 부분 배열을 정렬하지 않습니다. 대신, 계산 과정에서 선택 알고리즘을 사용합니다.
    - 각 부분 배열에서 계산된 중앙값들의 중앙값을 계산합니다. 이것이 pivot 값이 됩니다.

K-selection 알고리즘

- **Medians of medians**
  - 1~100 까지의 무작위로 정렬된 수
  - 5개의 그룹으로 묶어서 생각

62	21	51	87	8	4	61	23	78	2	74	66	88	65	17	13	92	99	89	6
70	50	12	43	38	96	55	34	26	86	80	73	83	9	93	52	46	97	20	30
11	14	45	69	67	100	82	90	91	71	35	39	29	84	37	27	33	81	98	18
1	48	24	54	59	68	31	72	5	60	41	95	76	19	79	28	25	47	49	15
16	32	57	64	75	40	44	77	53	85	63	7	56	42	22	94	36	3	58	10

  - 각 그룹의 크기는 5로 정해져 있어 중간값을 찾는것은 상수시간 내 할수 있음
    - 일단 심플하게 정렬됐다고 칩시다 (작은 단위수기 때문에 정렬하는데 걸리는시간을 상수처리)

1	14	12	43	8	4	31	23	5	2	35	7	29	9	17	13	25	3	20	6
11	21	24	54	38	40	44	34	26	60	41	39	56	19	22	27	33	47	49	10
16	32	45	64	59	68	55	72	53	71	63	66	76	42	37	28	36	81	58	15
62	48	51	69	67	96	61	77	78	85	74	73	83	65	79	52	46	97	89	18
70	50	57	87	75	100	82	90	91	86	80	95	88	84	93	94	92	99	98	30

  - 그룹의 총 개수는 N/5이고 중간값을 찾는데 걸리는 시간은 O(1)
  - 따라서 위 과정은 O(N)으로 수행

K-selection 알고리즘

- **Medians of medians**
  - 여기서 알고리즘의 아이디어
  - 중간값들의 중간값은 실제 데이터와 얼마나 차이가 날까?

1	14	12	43	8	4	31	23	5	2	35	7	29	9	17	13	25	3	20	6
11	21	24	54	38	40	41	34	26	60	41	39	56	19	22	27	33	47	49	10
16	32	45	64	59	68	55	72	53	71	63	66	76	42	37	28	36	81	58	15
62	48	51	69	67	96	61	77	78	85	74	73	83	65	79	52	46	97	89	18
70	50	57	87	75	100	82	90	91	86	80	95	88	84	93	94	92	99	98	30

$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_5$

- 우리가 찾은 중간값은
  - $a1 < a2 < a3 < a4 \ldots < a_{\frac{n}{5}}$
  - 그럼 이들의 중간값은 $a_{\frac{n}{10}}$
  - $a1$ 그룹에 $a_{\frac{n}{10}}$보다 작은 애들의 수는?
    - A1포함 최소 3 개는 있음
    - 이는 $a_{\frac{n}{10}}$그룹 전까지 같으므로 $\frac{n}{10} * 3 = \frac{3n}{10}$
  - 반대로 최대값은?
    - 비슷한 논리로 $\frac{n}{10} * 7 = \frac{7n}{10}$
  - 항상 30%에서 70% 사이에 떨어진다

$a_1 < a_2 < a_3 < a_4 \ldots < a_{n/5}$

중간 값은 $a_{n/10}$

$3 \times \frac{n}{10} = \frac{3n}{10}$

$\frac{n}{10} \times 3 = \frac{3}{10}n$

$\frac{1}{10}n$

K-selection 알고리즘

뭔..? ㅋㅋ k-select

- **Medians of medians**
  - 근데 문제는 각 중간값들의 후보군에서 중간값을 찾아야한다
  - 이는 다시 K-selection알고리즘을 통해 찾아낸다
- 무슨 말도 안되는 소리?
  - K-selection의 좋은 피벗을 위해 Medians of medians 사용
  - 근데 Medians of medians를 사용하기 위해 또 K-selection 사용?
  - 뭔 소리?
  - 근데 맞다

피벗

Quick Select 알고리즘의 O(N) 성능을 보장하기 위해서

하지만 피벗을 찾기 위해선 Quick Select

또 medians of medians

K-selection 알고리즘

- 추출된 중간값들을 K-selection 으로 중간값 찾음
  - 입력 배열에서 임의의 값을 pivot으로 선택
  - pivot을 기준으로 입력 배열을 분할합니다. pivot보다 작은 값들은 왼쪽 부분 배열로, 큰 값들은 오른쪽 부분 배열로 이동
  - pivot보다 작은 값들의 개수를 count합니다. 이때, count 값이 k보다 작은 경우, 오른쪽 부분 배열에서 (k-count)번째 작은 값을 찾음
  - count 값이 k보다 큰 경우, 왼쪽 부분 배열에서 k번째 작은 값을 찾음
  - k 값이 pivot보다 작은 경우, 왼쪽 부분 배열에서 k번째 작은 값을 재귀적으로 찾음
  - K 값이 pivot보다 큰 경우, 오른쪽 부분 배열에서 (k-count)번째 작은 값을 재귀적으로 찾음
- 결과적으로 대략적인 중앙값에 근사한 값을 가짐

count k-count

K-selection 알고리즘

K-selection 알고리즘

- 시간복잡도를 계산해보자
  - 배보다 배꼽이 더 커지지 않나?
  - 데이터를 쪼개서 정렬하는것은 O(1)으로 취급
  - n/5개의 중간값들중에 중간값을 찾는것은?
    - K-selec알고리즘 사용, $T(\frac{n}{5})$
  - 찾은 대략적인 중간값을 사용하면 최소 3대 7로 쪼개짐
  - 따라서 최악의 경우여도 $T(\frac{7n}{10})$
- 최종적인 관계식은?
  - $T(n) = O(n) + T(\frac{5}{n}) + T(\frac{7n}{10})$

상수시간이라 가정 했음

n이 커지면 엄청한 상수시간이라 가정할 수 없게 됨

O(N)

$T(n) = T(\frac{7}{10}n) + T(\frac{n}{5}) + O(n)$

최악의 경우 여3.

$T(n) = O(n) + T(\frac{5}{n}) + T(\frac{7}{10}n)$

$T(n) = O(n) + T(\frac{5}{n}) + T(\frac{7}{10}n)$

$O(1) \times n/5 = O(n)$

len(L), len(R) x.

최소한 3/10 n 씩은 줄어짐

K-selection 알고리즘

$c > 10a$

- 수학적 귀납법을 통해 증명해보자
  - $T(n) = O(n) + T\left(\frac{5}{n}\right) + T\left(\frac{7n}{10}\right)$
  - $T(n) \le a * n + T\left(\frac{5}{n}\right) + T\left(\frac{7n}{10}\right)$

$T(n) \le cn$ 을 만족 <귀납법>

$T(n) \le an + T\left(\frac{5}{n}\right) + T\left(\frac{7}{10}n\right)$

- 우리는 귀납법으로 어떠한 $c > 10a$ 가 있어서 $T(n) \le c \cdot n$ 을 만족함을 보이겠음
- n보다 작은 모든 $k < n$에 대해 $T(k) < c * k$가 성립함을 가정
- 이제 n일때 성립함을 보여주면 된다
  - $T(n) \le a \cdot n + \frac{c5}{n} + \frac{c7n}{10} = n(\frac{9c}{10} + a)$
  - $T(n) \le n\left(\frac{9c}{10} + a\right) \le n\left(\frac{9c}{10} + \frac{c}{10}\right) = c \cdot n$, (앞서 $c > 10a$로 정의하였기 때문)
- 따라서 놀랍게도 $T(n) = O(n)$ 임을 만족한다

어떠한 $c > 10a$

$T(n) \le cn$

$k<n$

$T(k) < c \cdot k$

가 성립한다 가정

$T(n)$

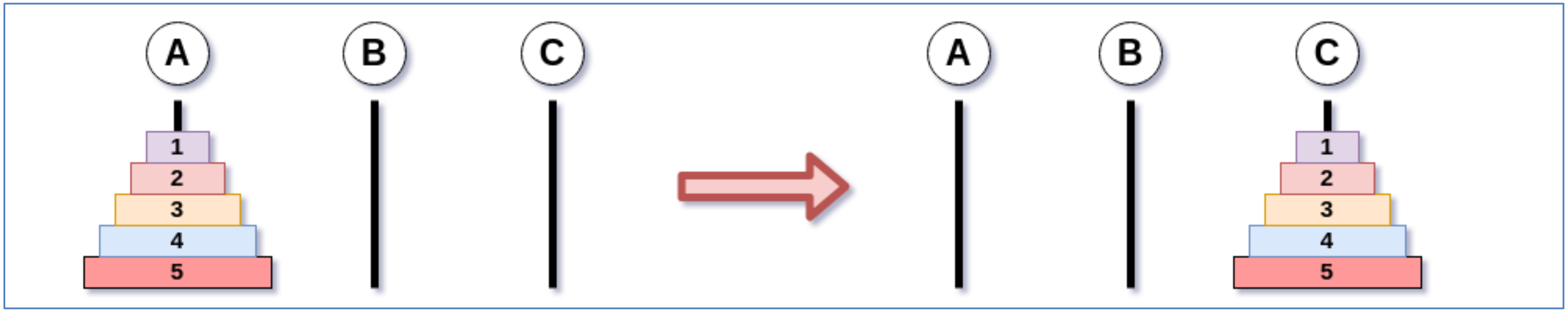
K-selection 알고리즘

- 그래서 이걸 많이 쓰나?
  - Medians of medians 알고리즘을 쓰는 K-selection은 최악의 경우에도 O(n)이 보장
  - 하지만 실제로는 쓰는일 자주 없다
  - 상수항이 너무 크기때문
    - Ex) 작은 배열 n/5개 배열을 정렬 또는 중앙값을 찾는 O(1)이 실제로는 많은 시간을 잡아먹음
- 하지만 아이디어는 기똥차다
  - 많은 알고리즘이 여기서 영감을 받는다
  - 이론적으로는 중요하다!

숨어 있는 상수항이 너무 큼.

하노이탑 쌓기

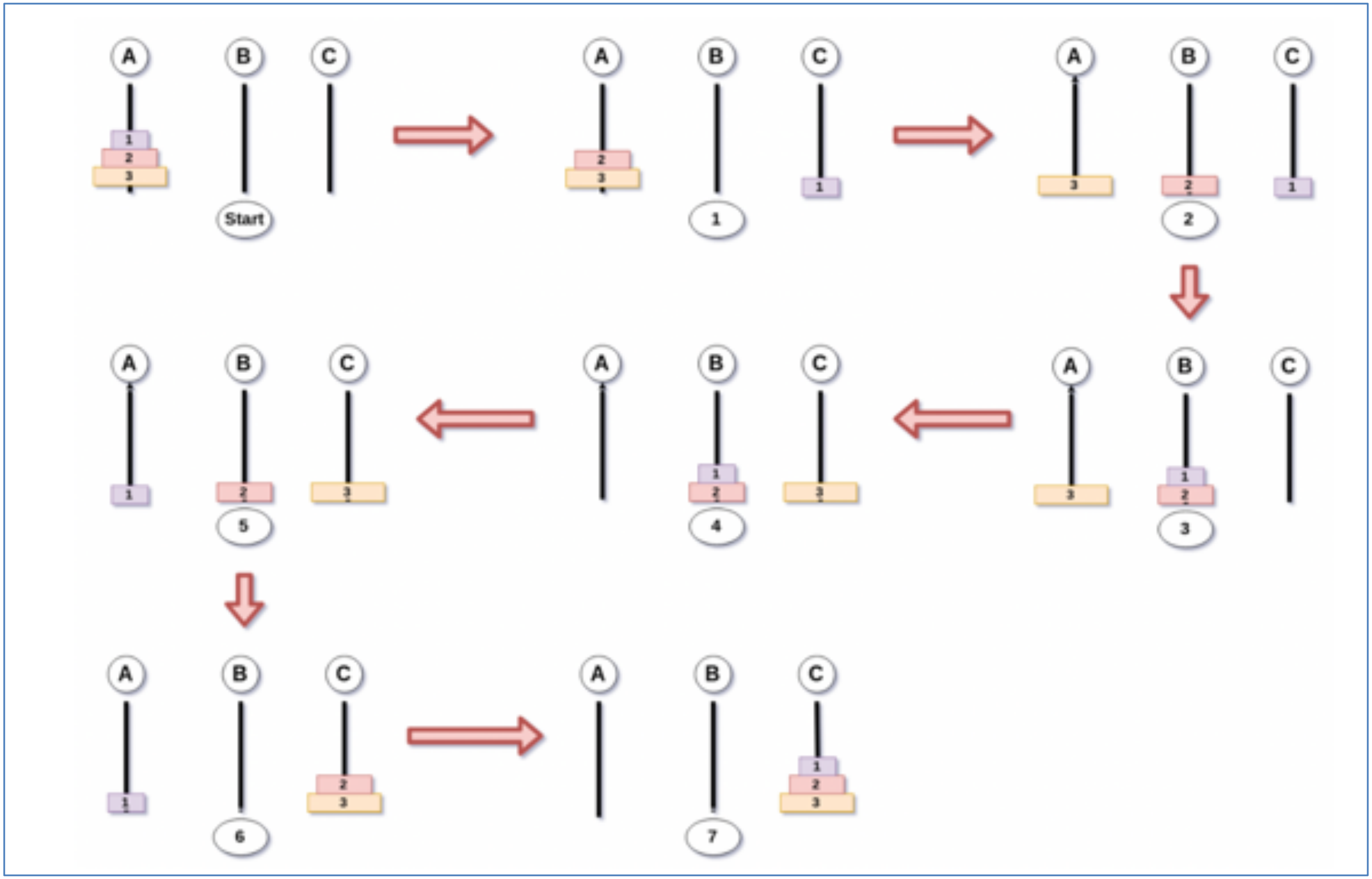
- Tower of Hanoi, 프랑스 수학자 에두아르 뤼카가 소개한 문제



- 원반을 옮기는데는 몇가지 조건이 따른다
  - 한번에 움직일 수 있는 원반은 기둥위에 놓인 원반 하나뿐
  - 어떤 원반 위에 그보다 더 큰 원반을 쌓을수 없다
- 최소의 이동횟수로 옮기는 가짓수는?

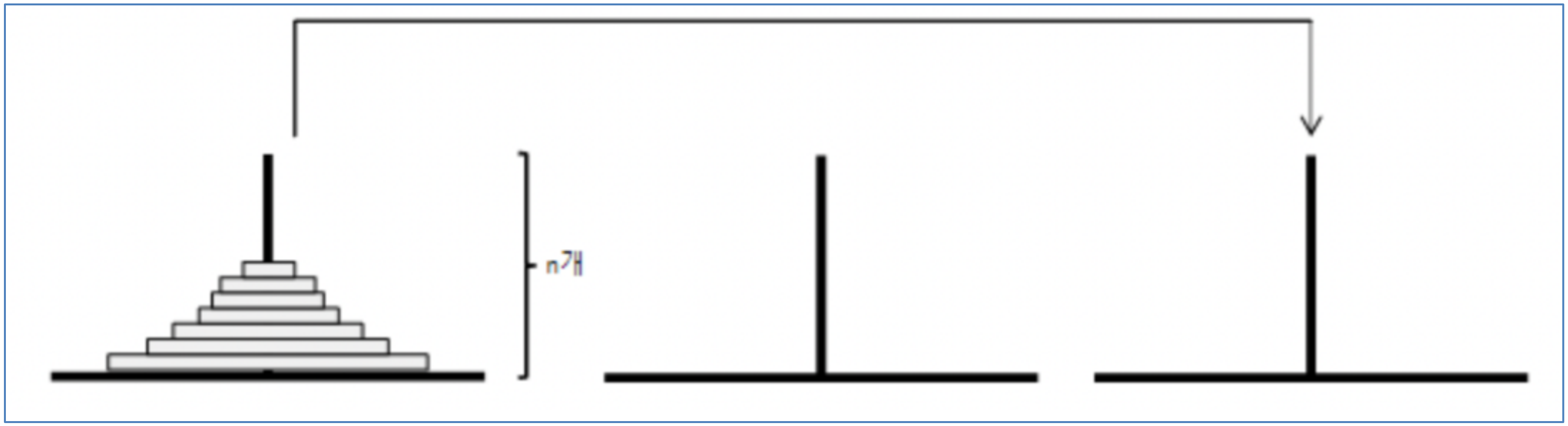
하노이탑 쌓기

- 우선 쉽게 다 해보자



하노이탑 쌓기

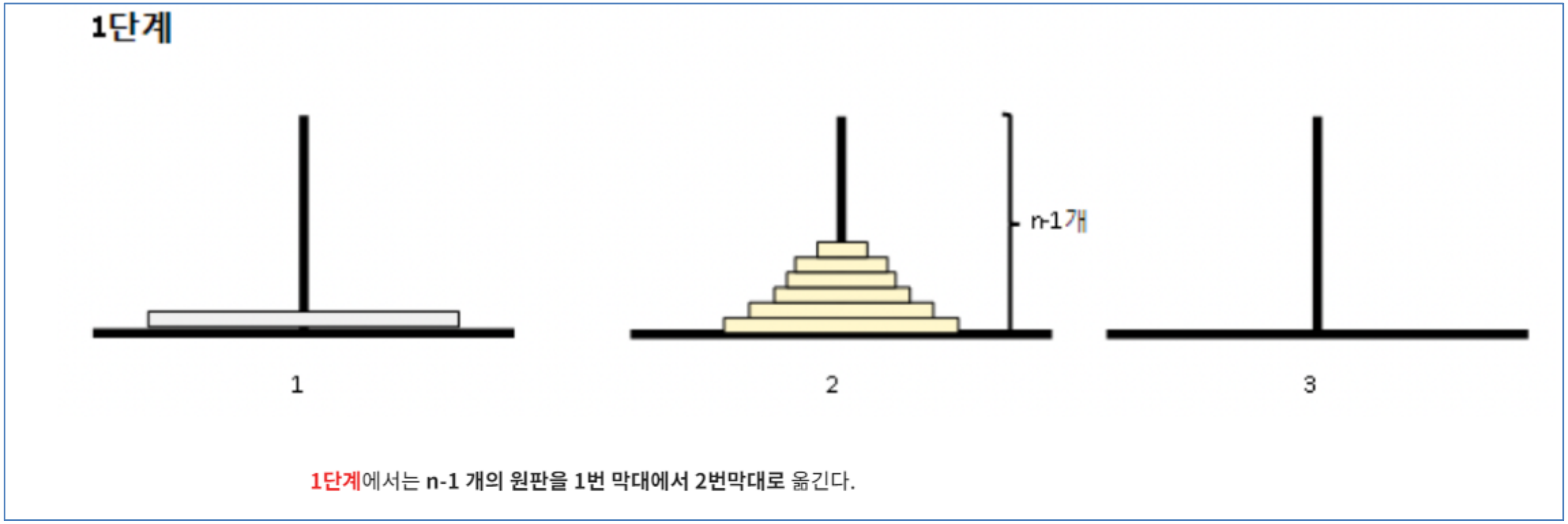
- N개 일때는…?아이디어를 내자



- 우리는 n개의 원판을 세번째로 옮겨야된다

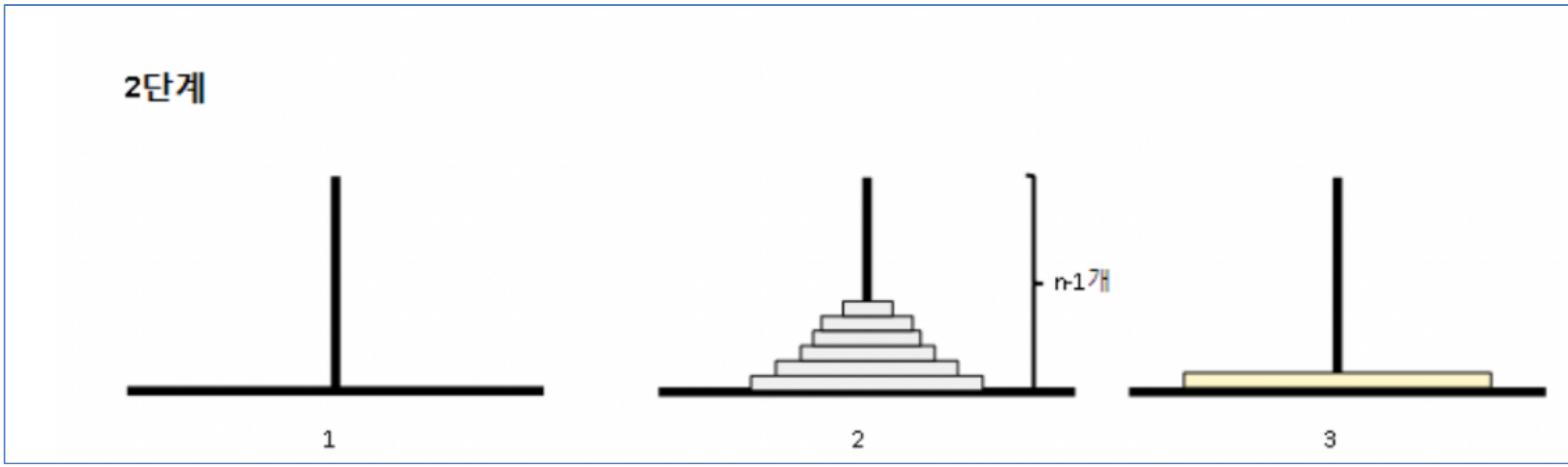
하노이탑 쌓기

- **1단계**
  - **N-1개의 원판을 2번 막대로 옮긴**



하노이탑 쌓기

- **2단계**
  - ➢ 남은 1개의 원판을 막대 3번으로 옮긴다



3 6

1 5

2 n

7층. 3→2
6층. 3→3
5층. 2→3
4층. 1→3
3층. 1→2
2층

하노이탑 쌓기

- **3단계**
  - ➢ **N-1개의 원판을 2번에서 3번으로 옮긴다**

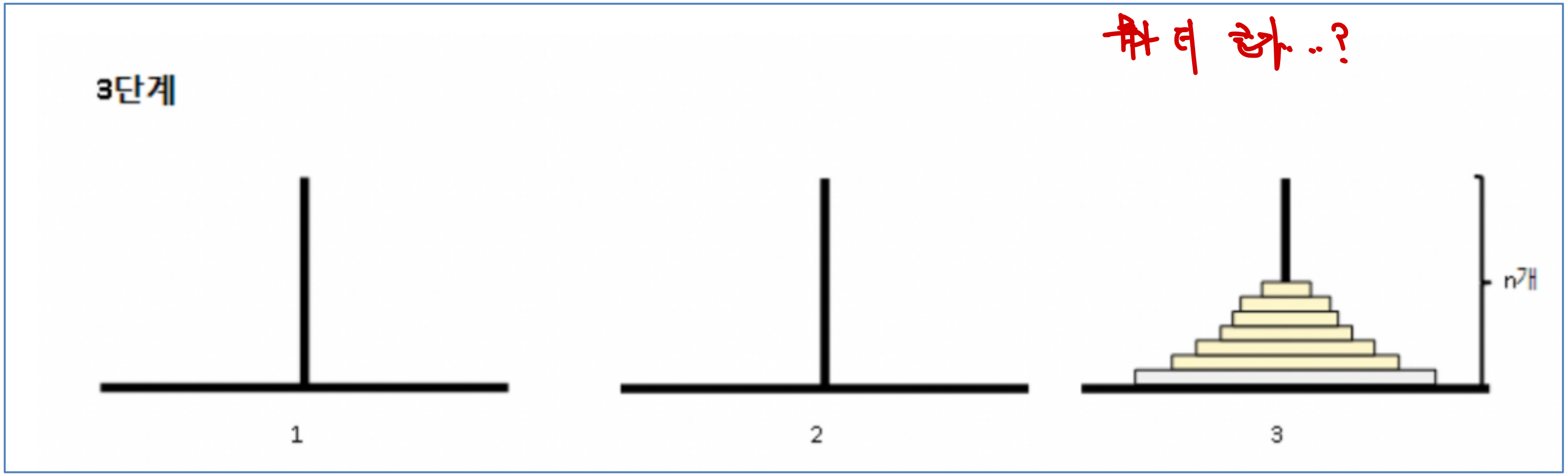
1.

이동순서는 지켜.

2.

4

뭐 더 할까..?



- **나머지는 재귀로 구현하면 된다...(과제)**