ZOAC 2019

Zero One Algorithm Contest 2019

김재현 @TheKinGoD
박성우 @hellogaon
윤병서 @bsyun0571

대회가 종료되었습니다.

수고하셨습니다.

Thanks to

- 개최
  - 한양대학교 ERICA 알고리즘 연구 학회 0&1

- 후원

Thanks to

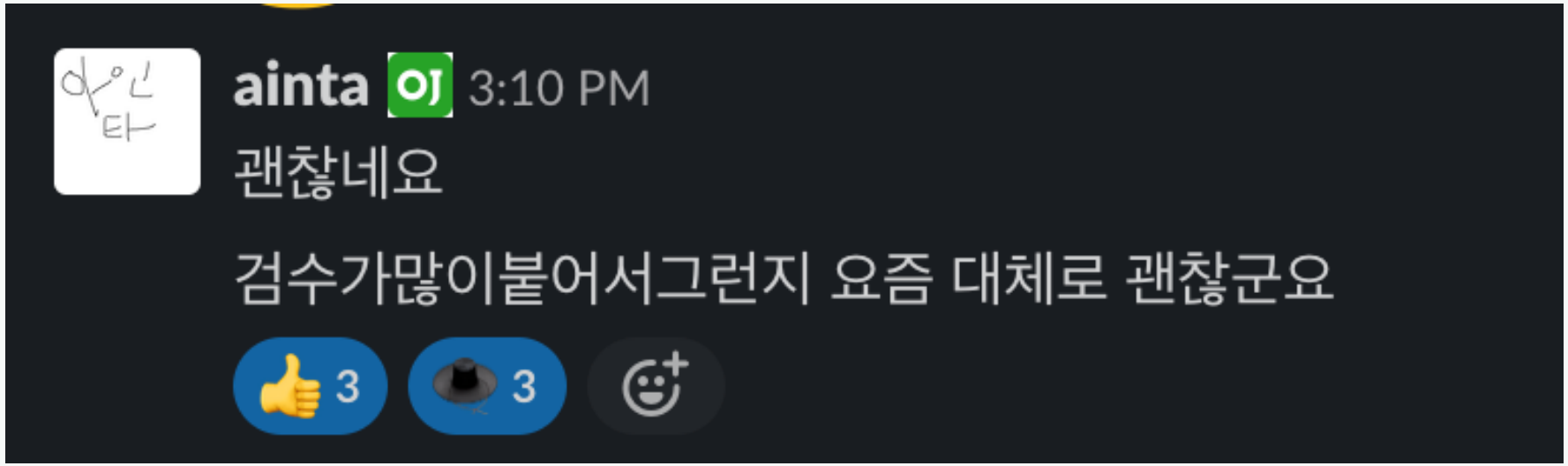
- 출제 위원
  - 김재현 @TheKinGoD
  - 박성우 @hellogaon
  - 윤병서 @bsyun0571

- 검수진
  - cheetose functionx h0ngjun7 jh05013 oree2113 rdd6584 (사전 순)

감사합니다!

출제진이 생각한 난이도

- A < H < E < F < G ≤ B < C < D < I



A. ZOAC 2

- 출제자: 김재현

- 성우가 문자열을 출력할 때 걸리는 시간의 최솟값

A. ZOAC 2

- 이전 문자에서 현재 문자로 오는 2가지 방법 중에 최솟값만 더해 주시면 됩니다.
  - min(abs(해당 문자 – 이전 문자), 26 – abs(해당 문자 – 이전 문자))

- O(N(모든 문자열 탐색))

A. ZOAC 2

- Tag
  - 쉬운 문제
  - 그리디
  - ZOAC의 전통

- First Solve: 이잉~앗살라말라이쿰~ (4 min)

H. 이상한 나라의 암호

- 출제자: 윤병서

- 하영이가 문장에 숨겨진 암호를 해독하는 것을 도와주자!

H. 이상한 나라의 암호

- **“Was it a cat?”** 이 입력으로 들어올 때까지 석판에 쓰여 있는 대로 i번째 줄은 첫 번째 글자에서 i칸씩 건너 뛰며 해석해주면 됩니다.

```
count = 2
While(input == Was it a cat?){
    for (i , 0 ~ input.size,  i += count)
        input[i] 출력
    count ++;
}
```

H. 이상한 나라의 암호

- Tag
  - 시뮬레이션
  - 구현
  - 문자열

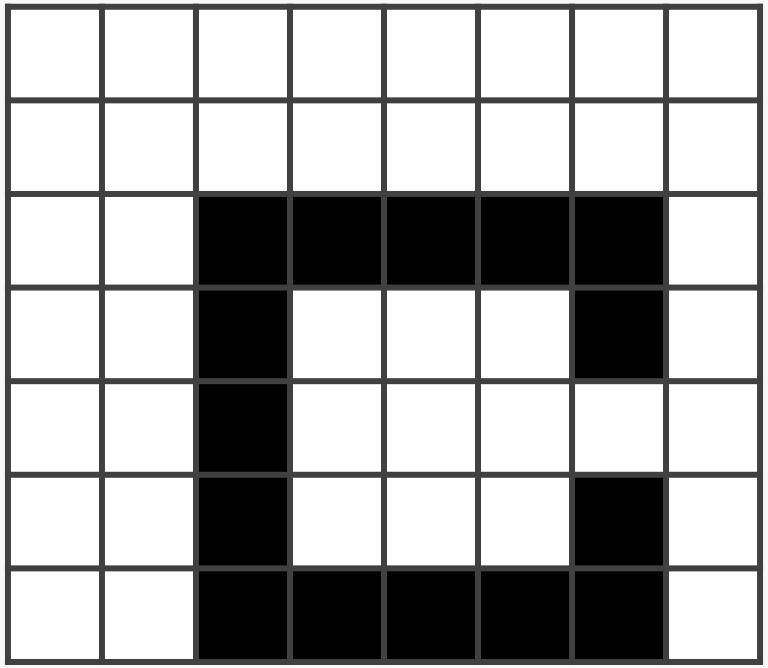
- First Solve: 후추 (25 min)

E. 네모네모 시력검사

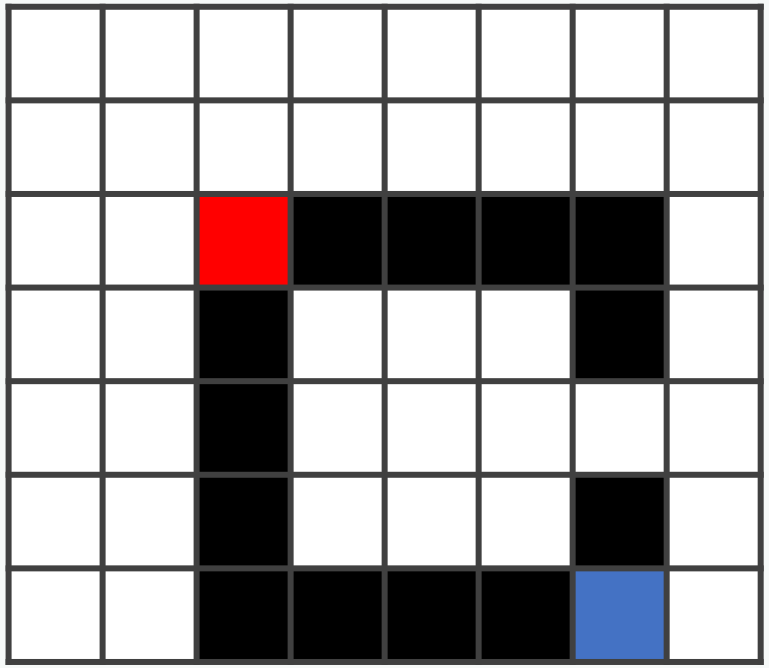
- 출제자: 박성우

- 정사각형의 네 변 중 한 변의 가운데는 색칠하지 않을 때 색칠하지 않은 변의 방향을 맞추어보자!

E. 네모네모 시력검사



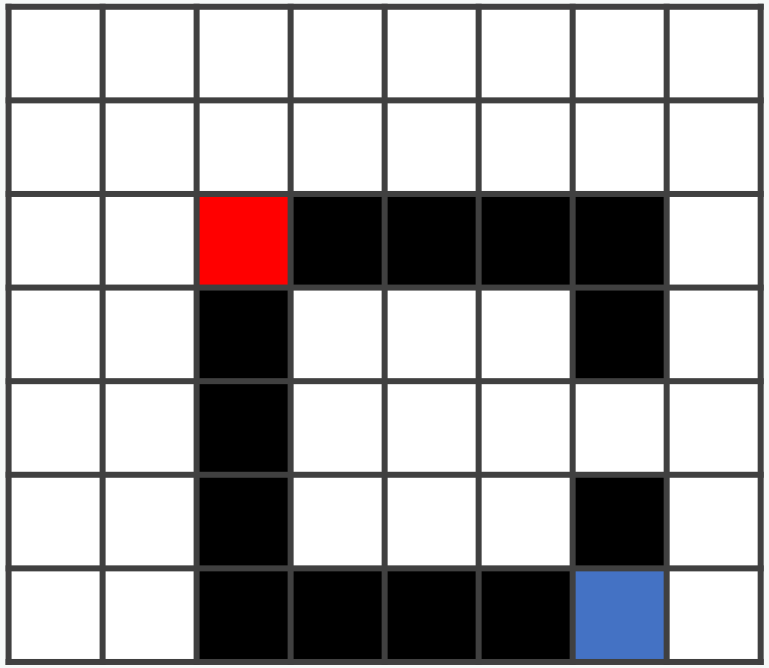
E. 네모네모 시력검사



Red : min(색칠한 사각형의 좌표)

Blue : max(색칠한 사각형의 좌표)

E. 네모네모 시력검사



Yellow : (R_y, B_x)

Green : (B_y, R_x)

E. 네모네모 시력검사

- 네 꼭지점들의 가운데 점이 색칠되어 있지 않은 변을 찾아서 정답을 출력!

E. 네모네모 시력검사

- Tag
  - 구현
  - 이중배열

- First Solve : 부정행위로 수상이 취소되었습니다. (28 min)

F. Small World Network

- 출제자: 김재현

- 모든 사람들이 6단계 이하로 연결되어 있는지 확인하자!
  - ~~특히 나와 이지은님이~~!

F. Small World Network

- 모든 노드들에 대해서 BFS를 돌리는데, 돌릴 때 마다 한 노드에 대하여 6단계 이상인 노드가 있으면 Big World!를 출력해 주시면 됩니다.
  - for i 1 ~ N:

    BFS(i)

    for j 1 ~ N:

      if node[j] > 6 then Big World!

- O(N(모든 노드 탐색) * N(BFS)) == $O(N^2)$

F. Small World Network

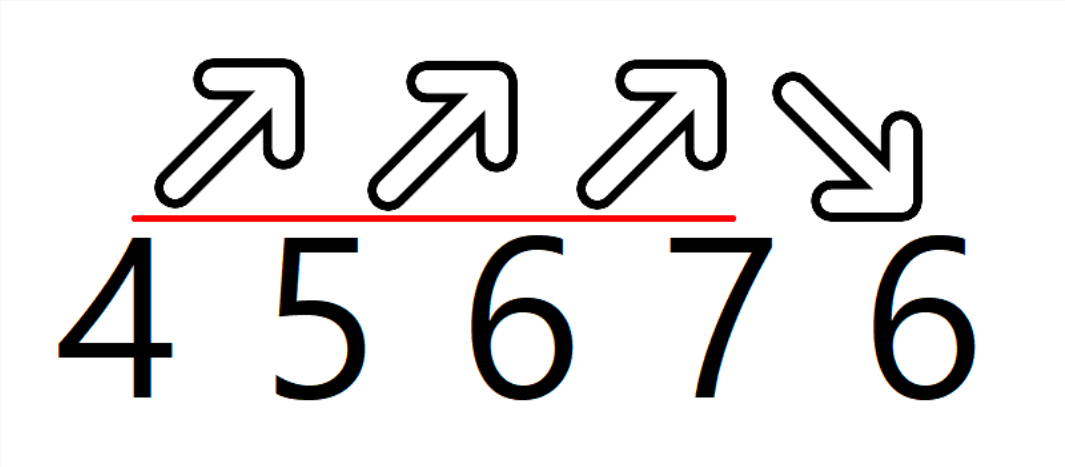
- Tag
  - 그래프
  - BFS
  - 김태형 교수님 소셜네트워크분석 들으세요

- First Solve: (0_<) (35 min)

G. 변형 계단 수

- 출제자: 김재현

- 모든 자리의 차이가 1이고 연속으로 3번 이상 증감하면 안댐!



G. 변형 계단 수

- 이전이 자신의 숫자(0~9)보다 ±1 인 경우에 연속하고 있다면 연속+1 감소하고 있다면 감소+1 해서 3번 연속 증감일 때는 안 더해 주시면 댑니다.
  - 0, 9 일 때는 연속으로 감소/증가 중일 때 밖에 없습니다.
  - 연속으로 감소/증가하다가 증가/감소하면 연속으로 1번 증가/감소입니다.

- O(10(자릿수) * N(변형 계단 수 길이)) == O(N)

G. 변형 계단 수

DP[i(101)][j(10)][k(5)] : i : 변형 계단 수의 길이, j : 끝나는 자릿수(0~9)

k : 연속 증감(3: 증감 X, 연속 증가:+1, 연속 감소:-1)

dp[1][all][3] = 1

dp[2][all][2, 4] = dp[1][all±1][3]

for i 3 ~ N:

dp[i][all][(2), (4)] = dp[i-1][all±1][(4, 5), (1, 2)]

dp[i][all][(1), (5)] = dp[i-1][all±1][(2), (4)]

G. 변형 계단 수

- Tag
  - 다차원 DP

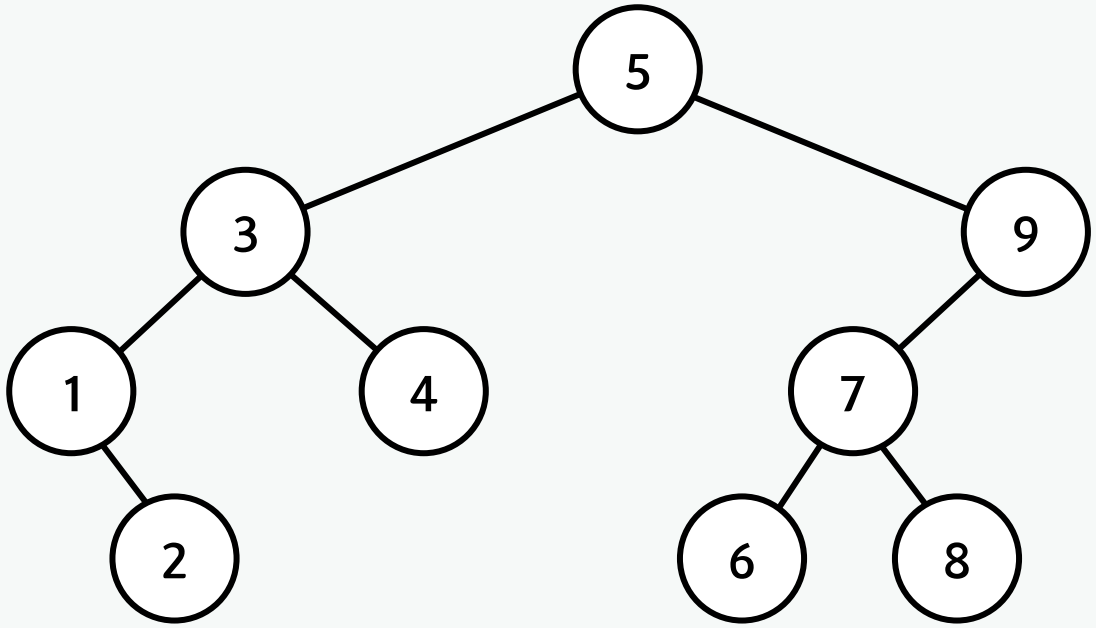
- First Solve: 후추 (61 min)

C. 이진 탐색 트리 복원하기

- 출제자: 박성우

- 이진 탐색 트리에 1부터 N까지의 모든 정수를 한 번씩 포함하고 있는 수열을 차례대로 삽입했을 때의 깊이가 주어진다. 원래의 수열을 출력하자.

C. 이진 탐색 트리 복원하기

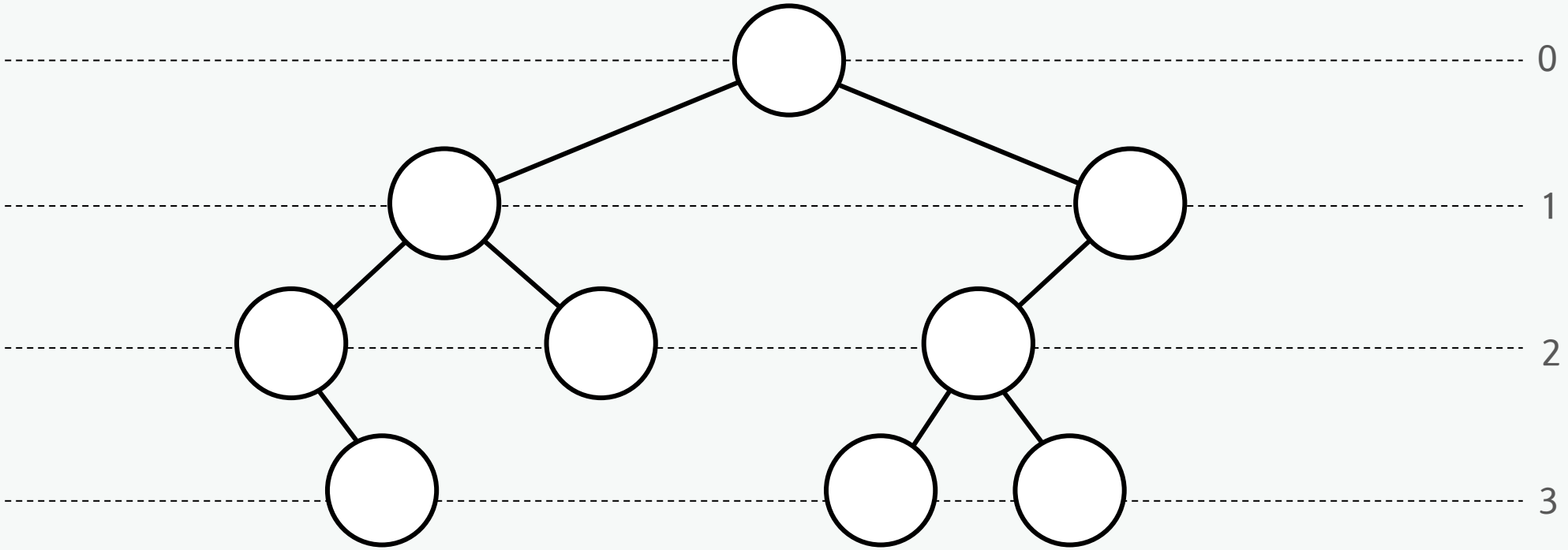


깊이 : 0 1 2 2 1 3 2 3 3

수열 : 5 3 1 4 9 2 7 8 6

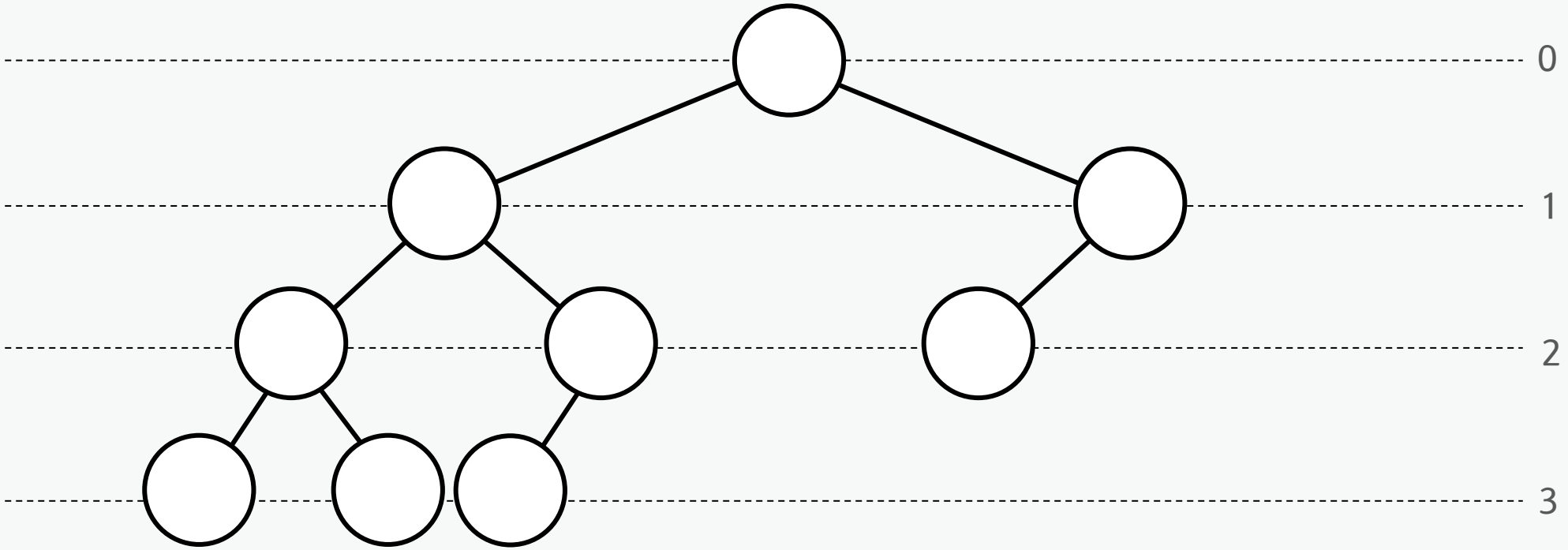
C. 이진 탐색 트리 복원하기

깊이 : 0 1 2 2 1 3 2 3 3



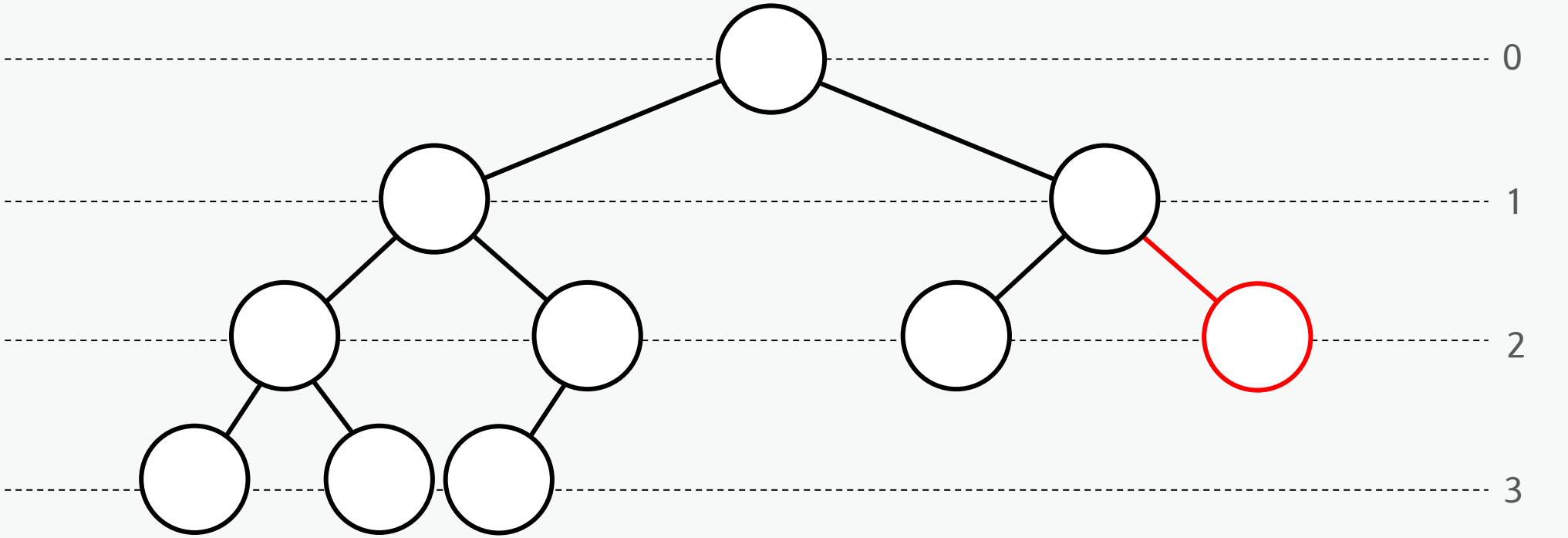
C. 이진 탐색 트리 복원하기

깊이 : 0 1 2 2 1 3 2 3 3



C. 이진 탐색 트리 복원하기

깊이 : 0 1 2 2 1 3 2 3 3 2 2



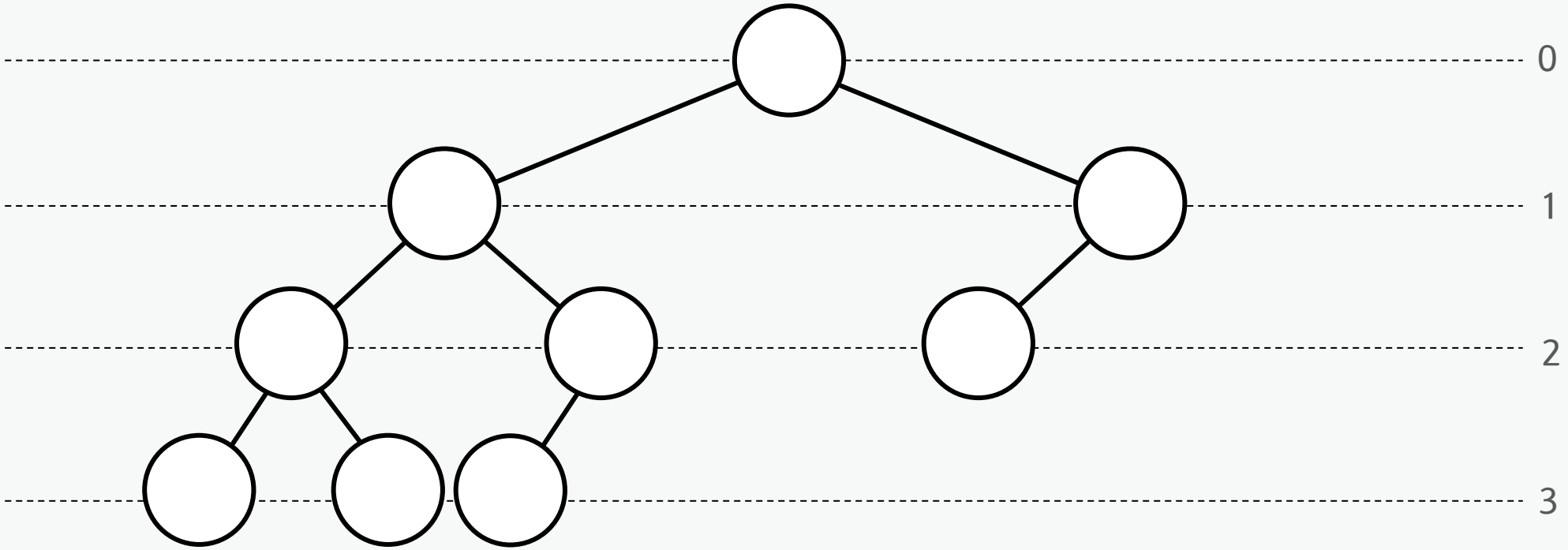
C. 이진 탐색 트리 복원하기

- 이번에 추가하려는 노드의 깊이가 K일 때,
  현재 깊이가 K인 노드의 개수가 X개, 깊이가 K-1인 노드의 개수가 Y개라면
  $2 \times Y < X + 1$ 일 경우 불가능!

- 가능한 경우 이제 1 ~ N까지만 잘 적어주면 된다!

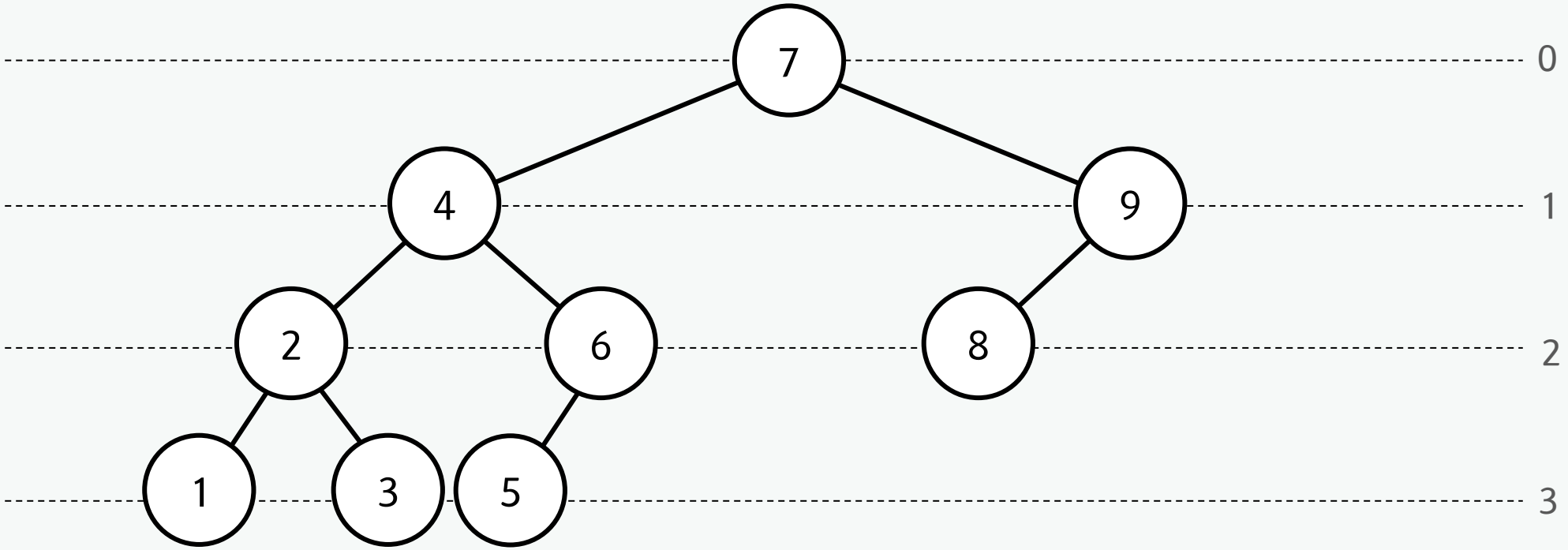
C. 이진 탐색 트리 복원하기

깊이 : 0 1 2 2 1 3 2 3 3



C. 이진 탐색 트리 복원하기

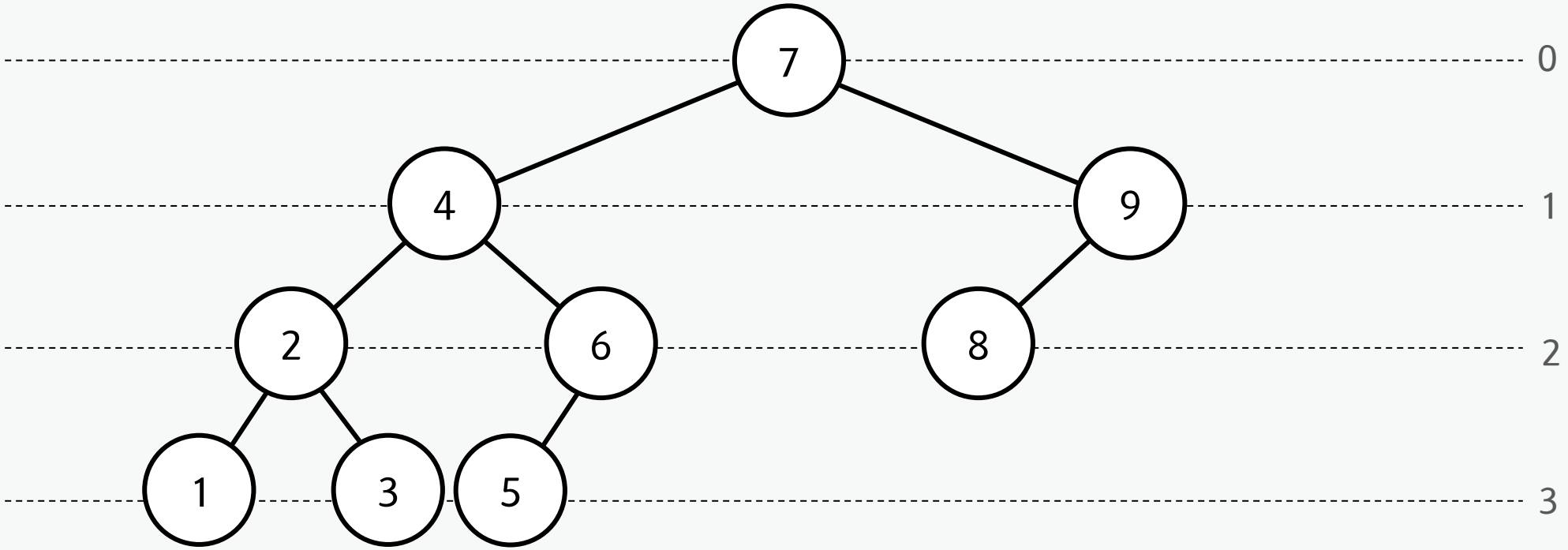
깊이 : 0 1 2 2 1 3 2 3 3



Inorder (left – root – right)

C. 이진 탐색 트리 복원하기

깊이 : 0 1 2 2 1 3 2 3 3



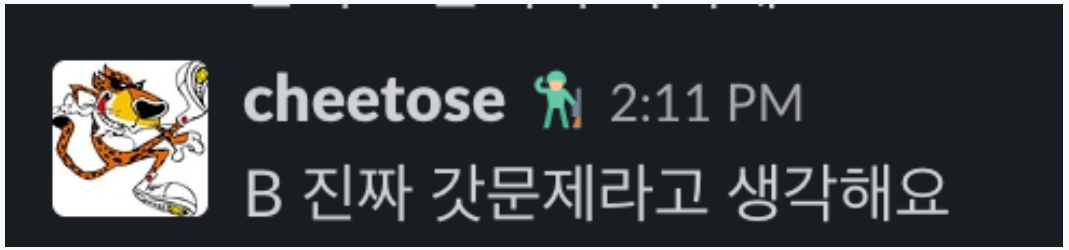
수열 : 7 4 2 6 9 1 8 3 5

C. 이진 탐색 트리 복원하기

- Tag
  - 트리
  - 트리 순회
  - 이진 탐색 트리
  - 스페셜 저지

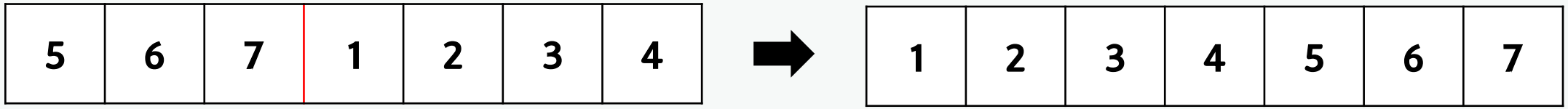
- First Solve : ??? (? min)

B. 편안한 수열 만들기



- 출제자: 박성우

- 편안한 수열 : 1부터 N까지의 정수가 한번씩 오름차순으로 있는 수열
  shift 되어 있는 편안한 수열을 swap, reverse 연산을
  정확히 5번 이용하여 원래의 편안한 수열로 되돌리자!

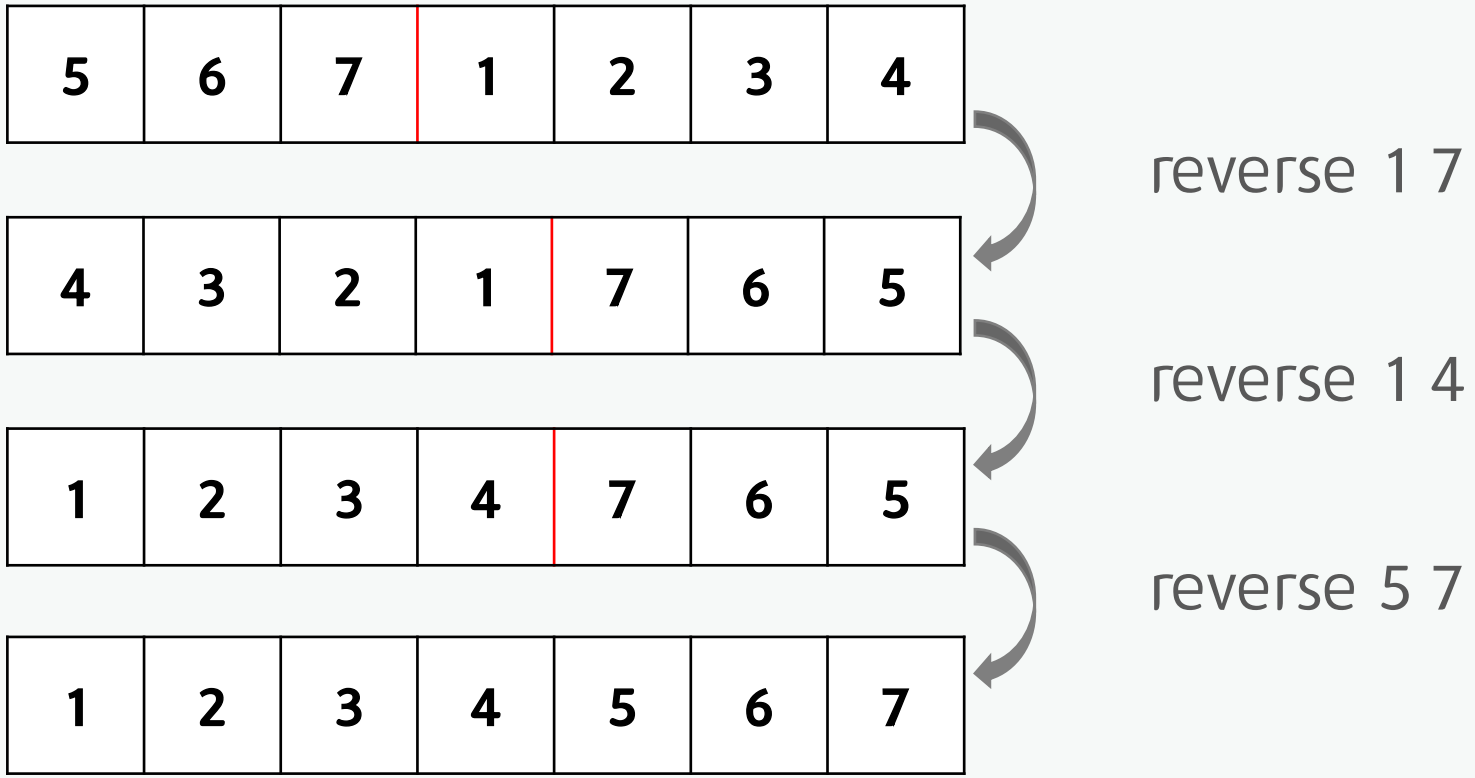


B. 편안한 수열 만들기

- 3번의 연산을 통하여 편안한 수열을 만들 수 있다면,
  남은 2번의 연산은 같은 연산 2번을 해주면 해결 가능

- 일반적인 경우에 대하여 생각해보자!

B. 편안한 수열 만들기

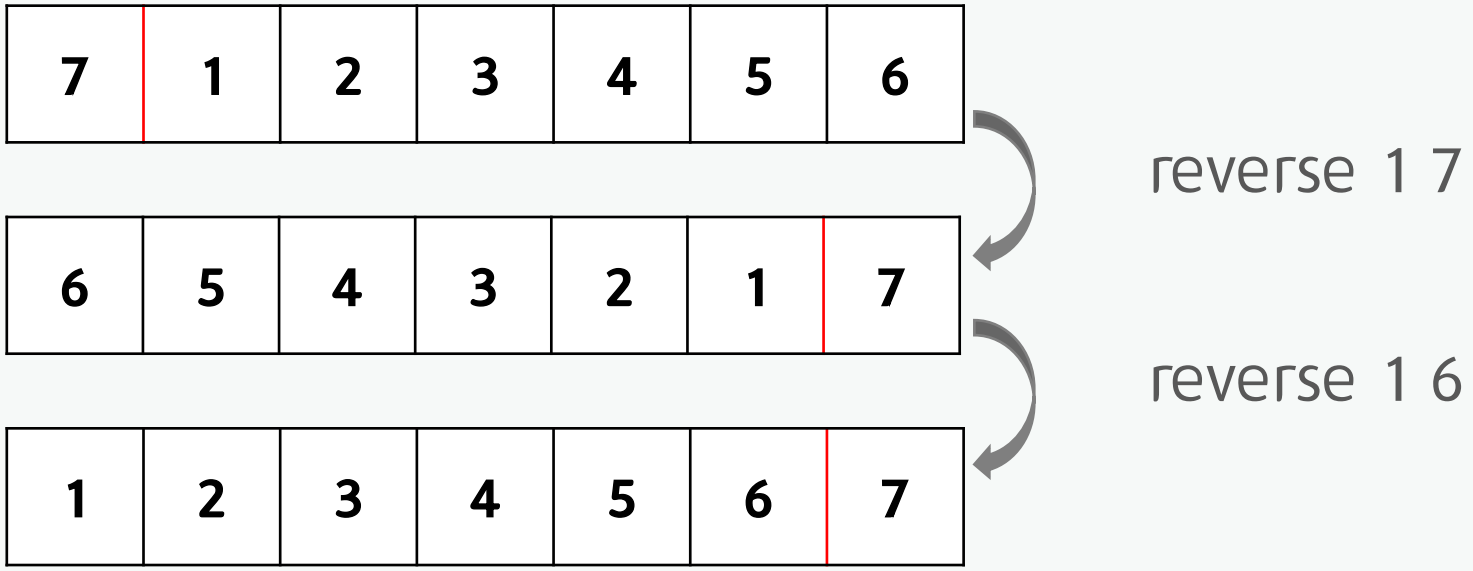

5 6 7 1 2 3 4
reverse 1 7
4 3 2 1 7 6 5
reverse 1 4
1 2 3 4 7 6 5
reverse 5 7
1 2 3 4 5 6 7

B. 편안한 수열 만들기

- 이러한 방법을 적용할 수 없는 경우는 언제일까?
  - Case 1) K = 1일 경우
  - Case 2) K = N-1일 경우
  - Case 3) ??
  - Case 4) ??

B. 편안한 수열 만들기

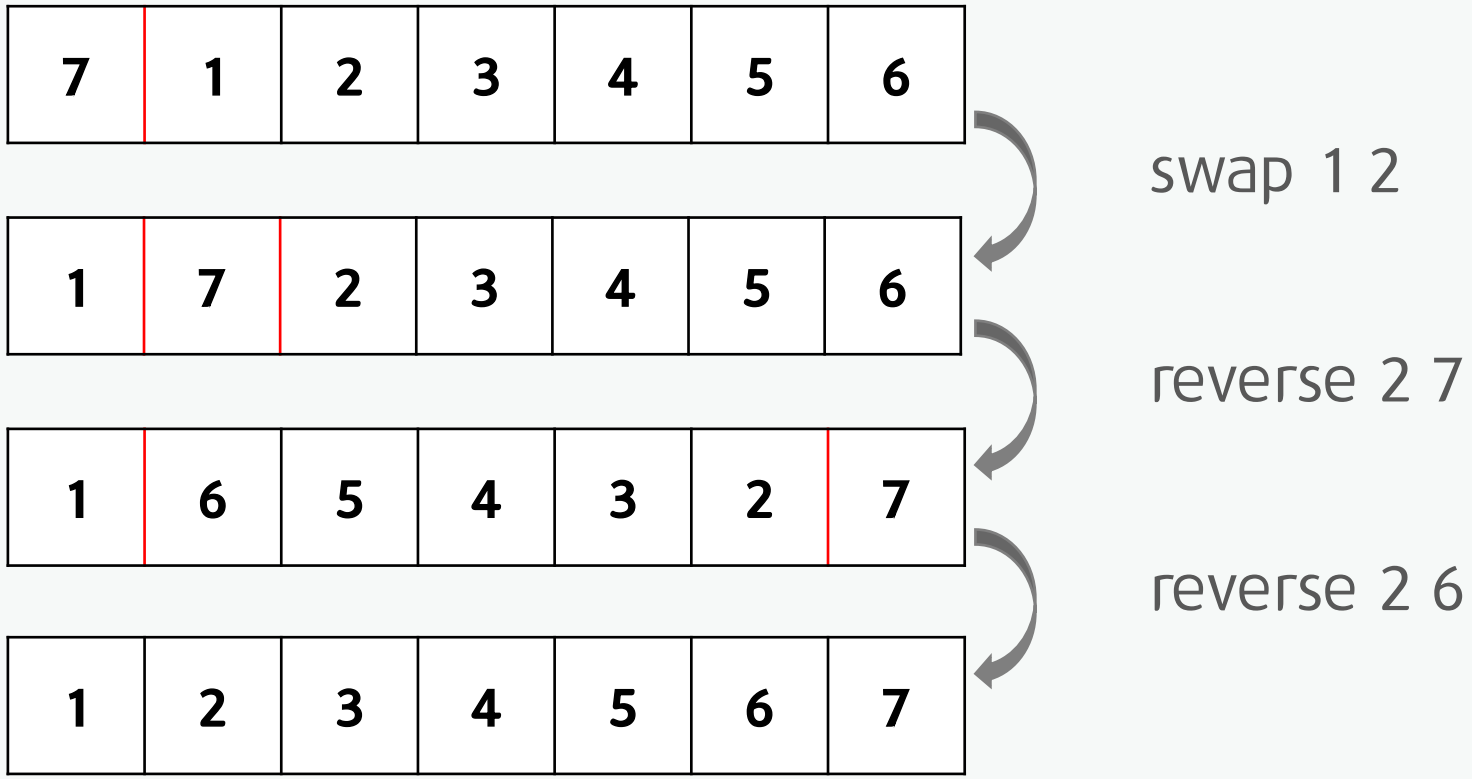
Case 1), Case 2)



2번만에 편안한 수열이 완성이 되어 다른 방법을 구상해 봐야한다.

B. 편안한 수열 만들기

Case 1), Case 2)

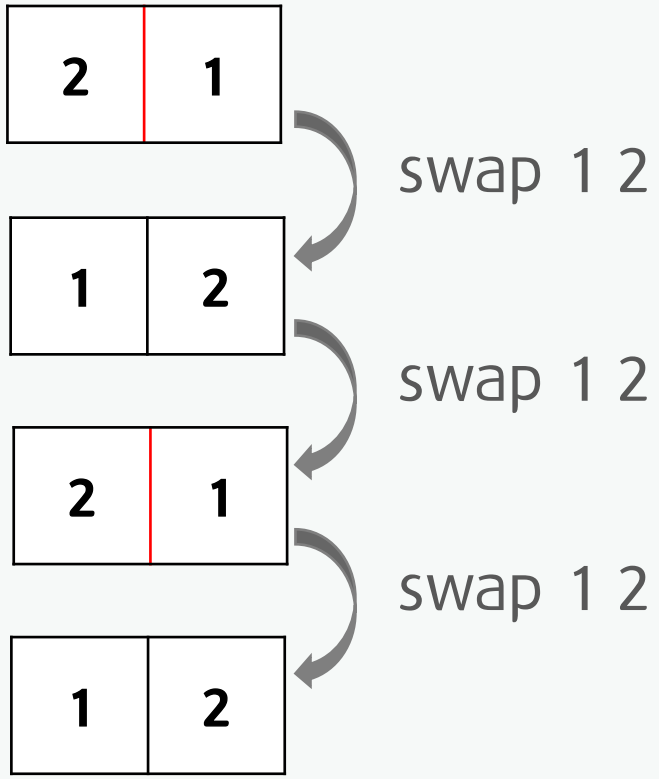


B. 편안한 수열 만들기

- 이러한 방법을 적용할 수 없는 경우는 언제일까?
  - Case 1) K = 1일 경우
  - Case 2) K = N-1일 경우
  - Case 3) N = 2일 경우
  - Case 4) ??

B. 편안한 수열 만들기

Case 3)



B. 편안한 수열 만들기

- 이러한 방법을 적용할 수 없는 경우는 언제일까?
  - Case 1) K = 1일 경우
  - Case 2) K = N-1일 경우
  - Case 3) N = 2일 경우
  - Case 4) N = 3일 경우

B. 편안한 수열 만들기

Case 4)

모든 경우의 수를 조사해보면, N = 3일 때는 불가능하다는 것을 알 수 있다.

B. 편안한 수열 만들기

- swap의 경우 인접한 경우에만 사용을 하다보니 reverse로 대체 가능!
- swap이 없어도 되는 문제!
- 다양한 풀이가 있을 수도??

B. 편안한 수열 만들기

- Tag
  - 케이스 분류
  - 스페셜 저지

- First Solve: ??? (? min)

D. 문자열 게임

- 출제자: 윤병서

- 주어진 문자열 S에서 N번의 명령에 따라 문자열 W를 제거하는 문제

D. 문자열 게임

- 문제에서 주어지는 명령을 수행하기 위해 문자열 S를 왼쪽, 오른쪽부터 담을 자료 구조인 덱(Deque)을 각각 선언합니다.

- 이후 풀이에서는 왼쪽 덱을 DL, 오른쪽 덱을 DR이라고 표현하겠습니다.

D. 문자열 게임

- 명령 L이 주어지면, DL의 크기가 문자열 W의 길이와 같아질 때까지 문자열 S의 앞부분부터 한 글자 씩 삽입합니다.

- DL의 크기가 제거해야 하는 문자열 W의 길이와 같아지는 순간부터, 문자열 W와 문자열 W의 길이 만큼의 DL의 뒷부분을 비교합니다.

D. 문자열 게임

- 비교 했을 때, W와 DL의 뒷부분이 같으면 DL에서 문자열 W를 제거한 후 작업을 종료합니다.

- 다르면 같을 때까지 DL에 한 글자 씩 삽입한 후, 비교하는 작업을 수행합니다. 만약, 끝까지 탐색 후 발견하지 못하면, 명령이 실패한 것이므로 아무 일도 하지 않습니다.

D. 문자열 게임

- 명령 R이 주어지면, DR의 크기가 문자열 W의 길이와 같아질 때까지 문자열 S의 뒷부분부터 한 글자 씩 삽입합니다.

- DR의 크기가 제거해야 하는 문자열 W의 길이와 같아지는 순간부터, 문자열 W와 문자열 W의 길이 만큼의 DR의 앞부분을 비교합니다.

D. 문자열 게임

- 비교 했을 때, W와 DR의 앞부분이 같으면 DR에서 문자열 W를 제거한 후 작업을 종료합니다.

- 다르면 같을 때까지 DR에 한 글자 씩 삽입한 후, 비교하는 작업을 수행합니다. 만약, 끝까지 탐색 후 발견하지 못하면, 명령이 실패한 것이므로 아무 일도 하지 않습니다.

D. 문자열 게임

- 명령을 실행하다 보면, DL과 DR에 주어진 문자열 S가 잘려진 상태, 또는 한 쪽에 몰린 상태로 가득 차게 됩니다.

- 이때, 제거해야 하는 문자열 W를 ()로 나타내면, DL과 DR에 들어가 있는 글자들을 합친 문자열 안에서 W들은 ((())) 의 형식으로 남아있게 됩니다.

D. 문자열 게임

- 따라서, 이때부터는 명령이 L, R인지와는 관계없이 실행한 명령이 N개가 될 때까지 DL과 DR을 합친 문자열의 가운데에 존재하는 W를 제거합니다.

- 명령이 모두 종료된 후, 성공한 명령의 개수, 명령을 수행한 후의 문자열, 성공 여부를 각각의 줄에 출력하시면 됩니다!

D. 문자열 게임

- Tag
  - Stack / Deque
  - 구현
  - 시뮬레이션

- First Solve: ??? (? min)

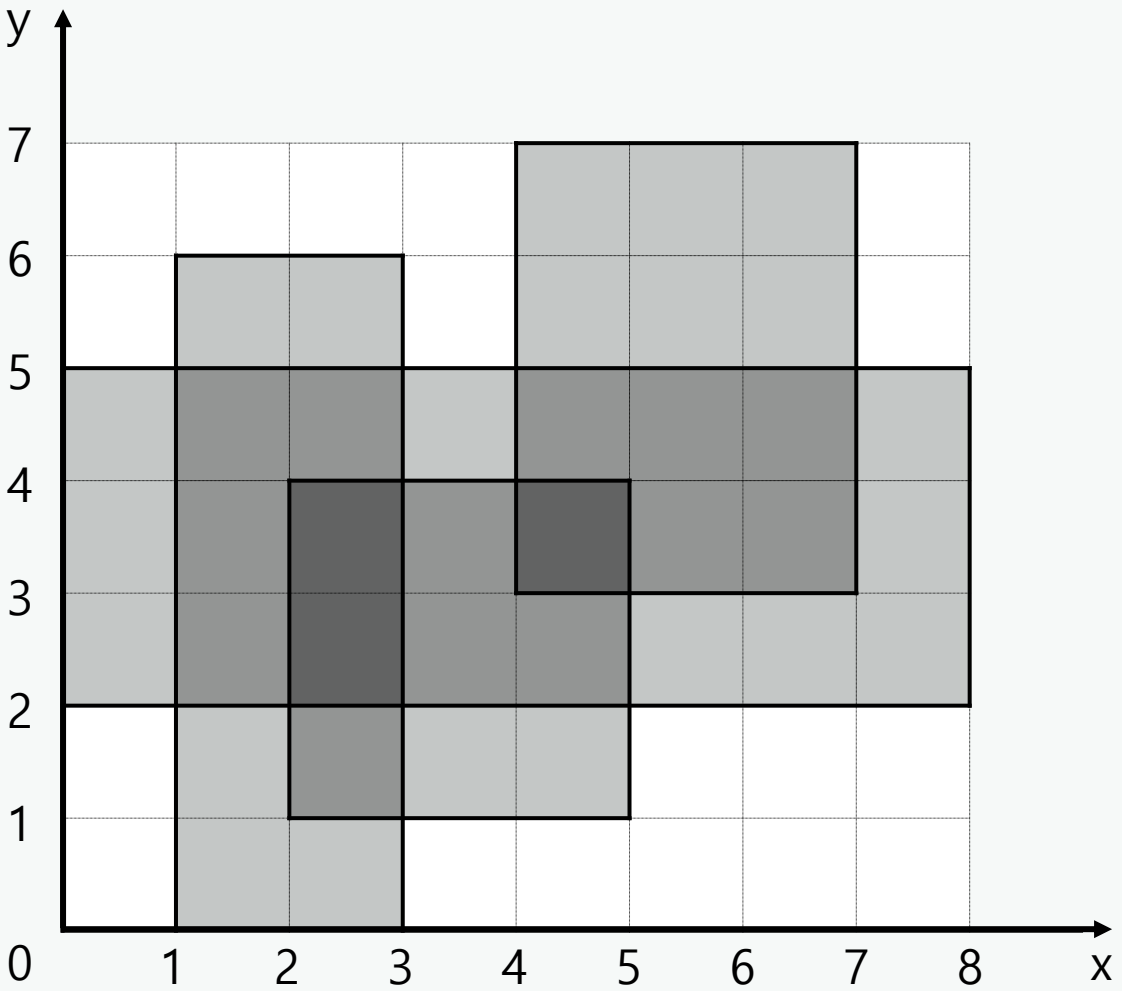
I. 색종이와 쿼리

- 출제자: 박성우

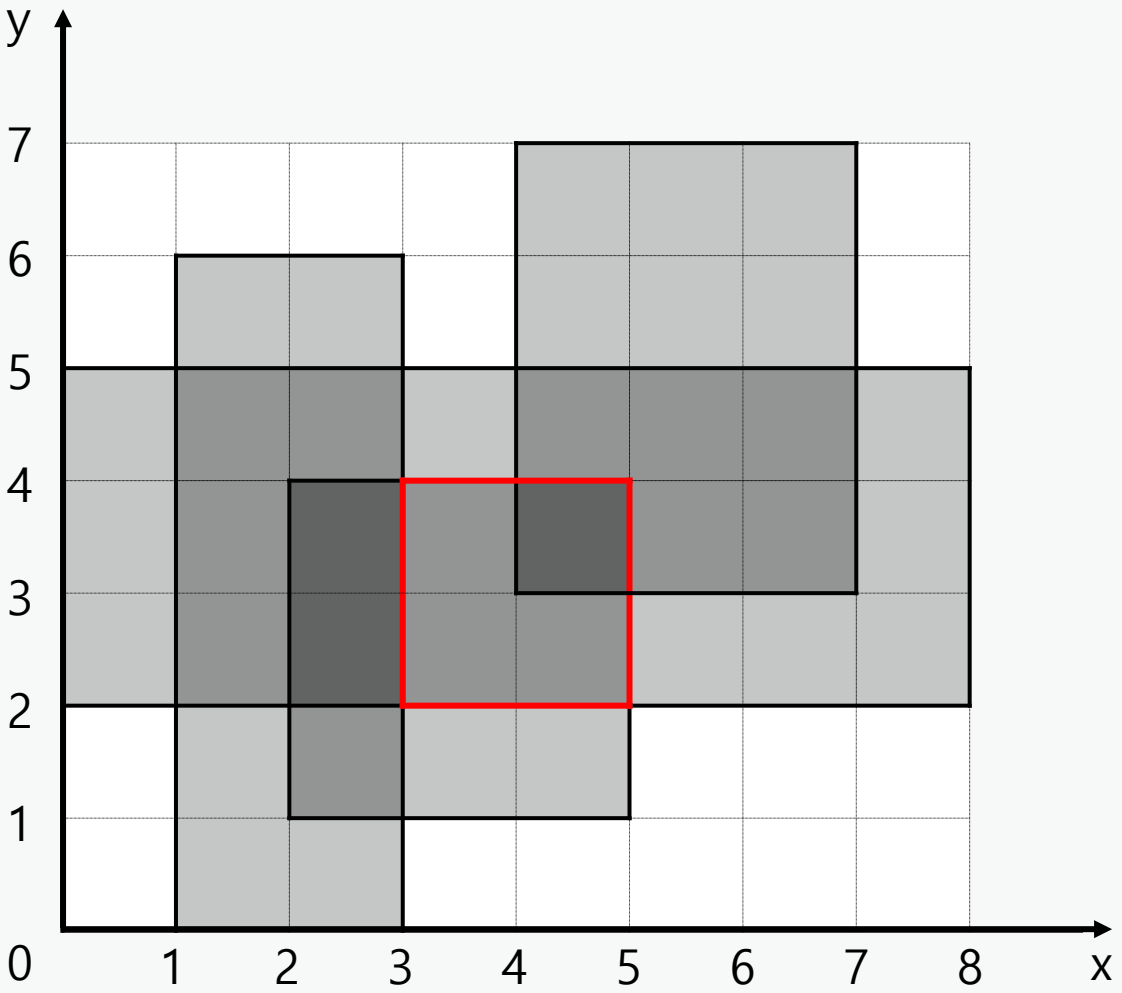
- 색종이 N장이 좌표평면 상에 놓여있을 때
  직사각형 영역에서 가장 많이 겹쳐 있는 영역에 놓여 있는 색종이의 장 수를 구해라!

- $1 \leq$ 색종이의 개수, 쿼리의 개수 $\leq 100{,}000$

I. 색종이와 쿼리



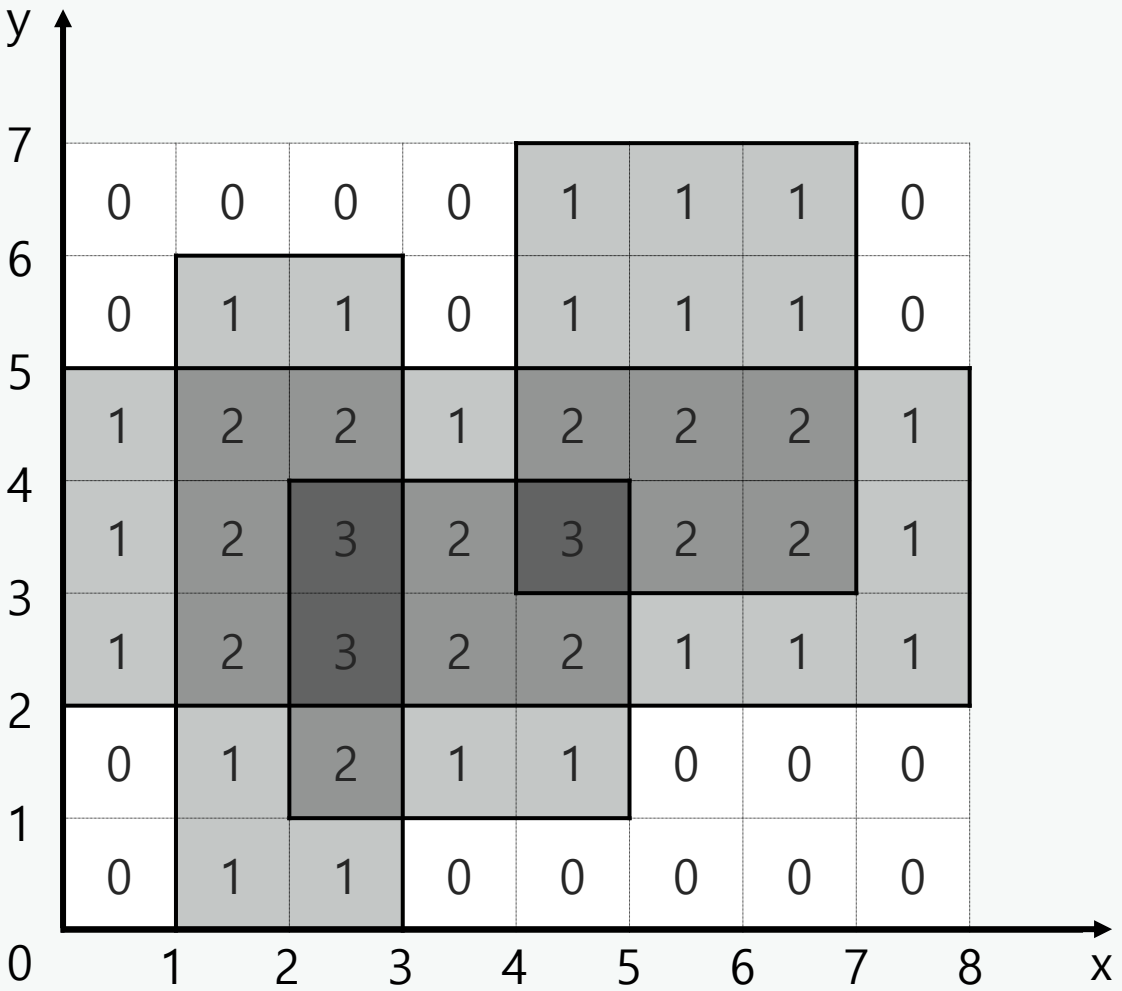
I. 색종이와 쿼리



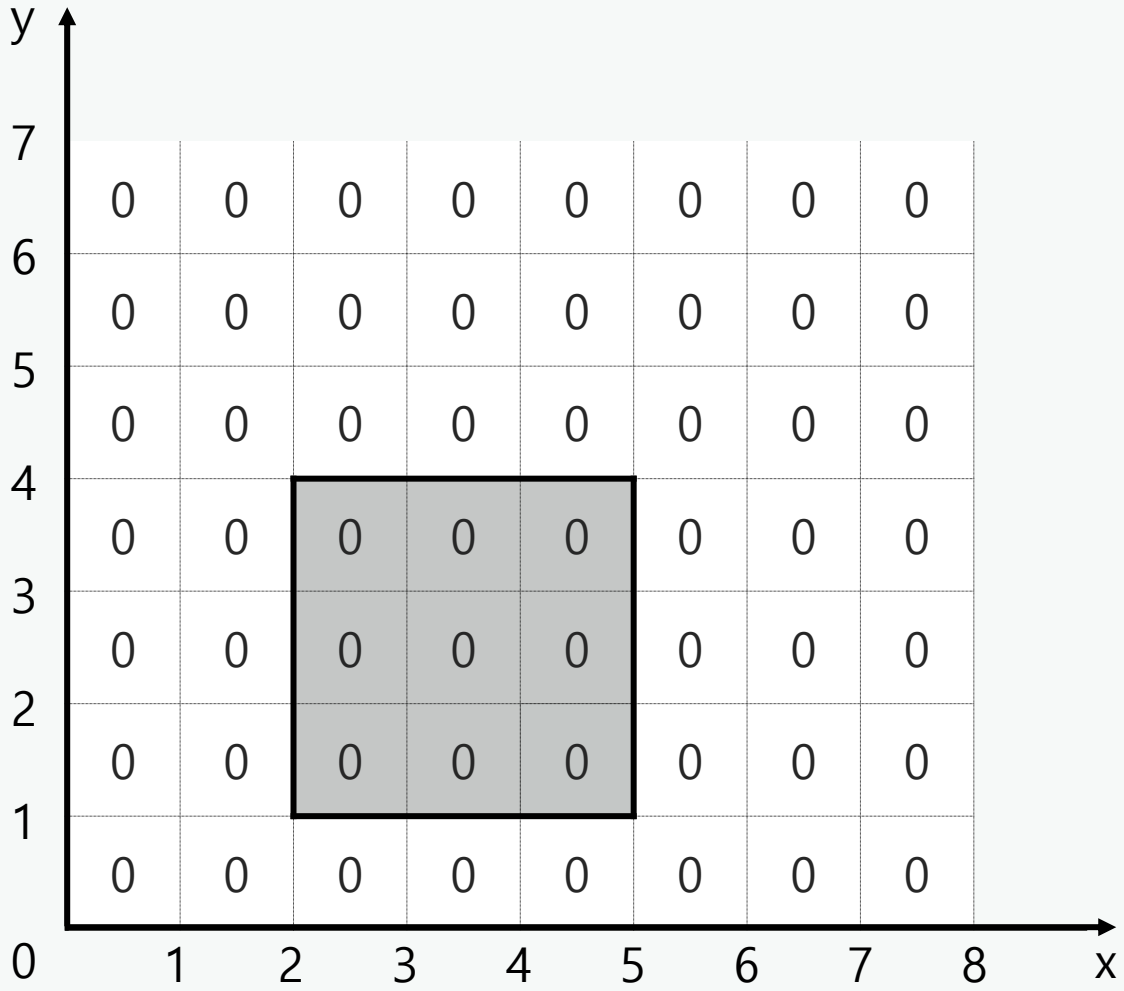
I. 색종이와 쿼리

- 1단계 : 각 칸마다 몇 장의 색종이가 겹쳐있는 지 확인해보자
  - 색종이의 개수가 최대 100,000개이기에 모든 사각형 내부를 for문으로 돌면서 센다면 시간 초과
  - 어떻게 하면 효율적으로 구할 수 있을까?
    - 좌표가 0 ~ 1500인 것을 이용하여 **부분합!!**

I. 색종이와 쿼리

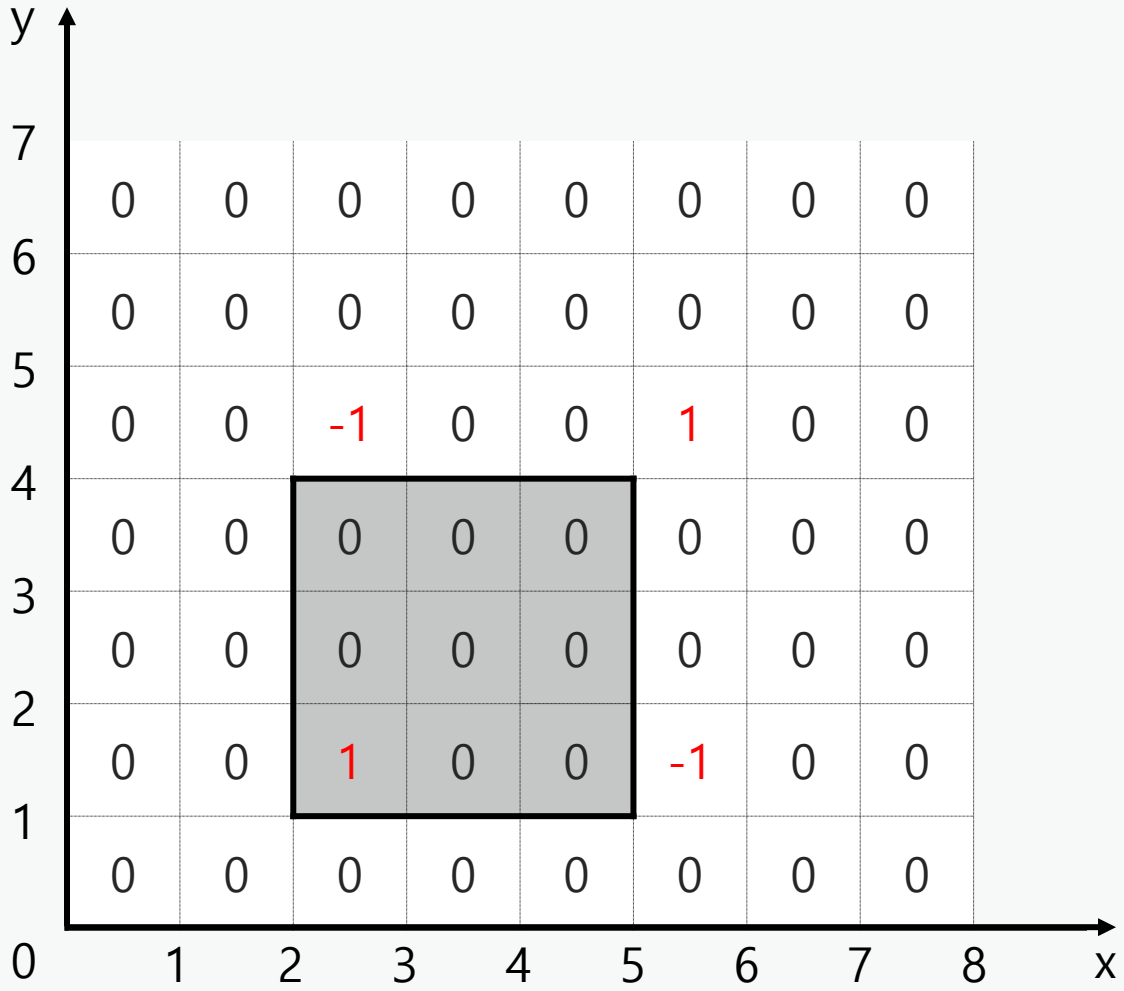

y
7
6
5
4
3
2
1
0
0 0 0 0 1 1 1 0
0 1 1 0 1 1 1 0
1 2 2 1 2 2 2 1
1 2 3 2 3 2 2 1
1 2 3 2 2 1 1 1
0 1 2 1 1 0 0 0
0 1 1 0 0 0 0 0
1 2 3 4 5 6 7 8
x

I. 색종이와 쿼리



일단 하나의 직사각형을 보자

I. 색종이와 쿼리



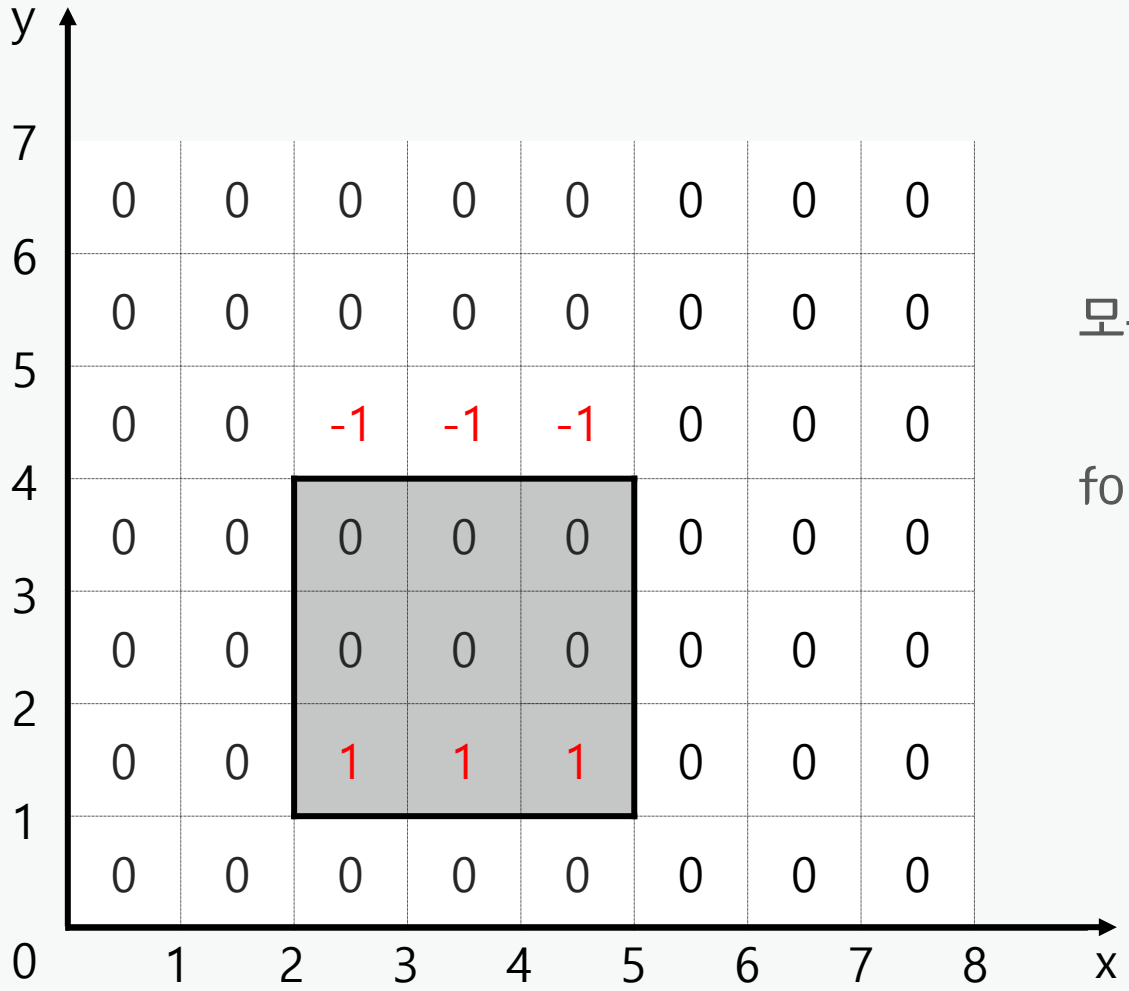
mat[y1][x1] = 1

mat[y1][x2] = -1

mat[y2][x1] = -1

mat[y2][x2] = 1

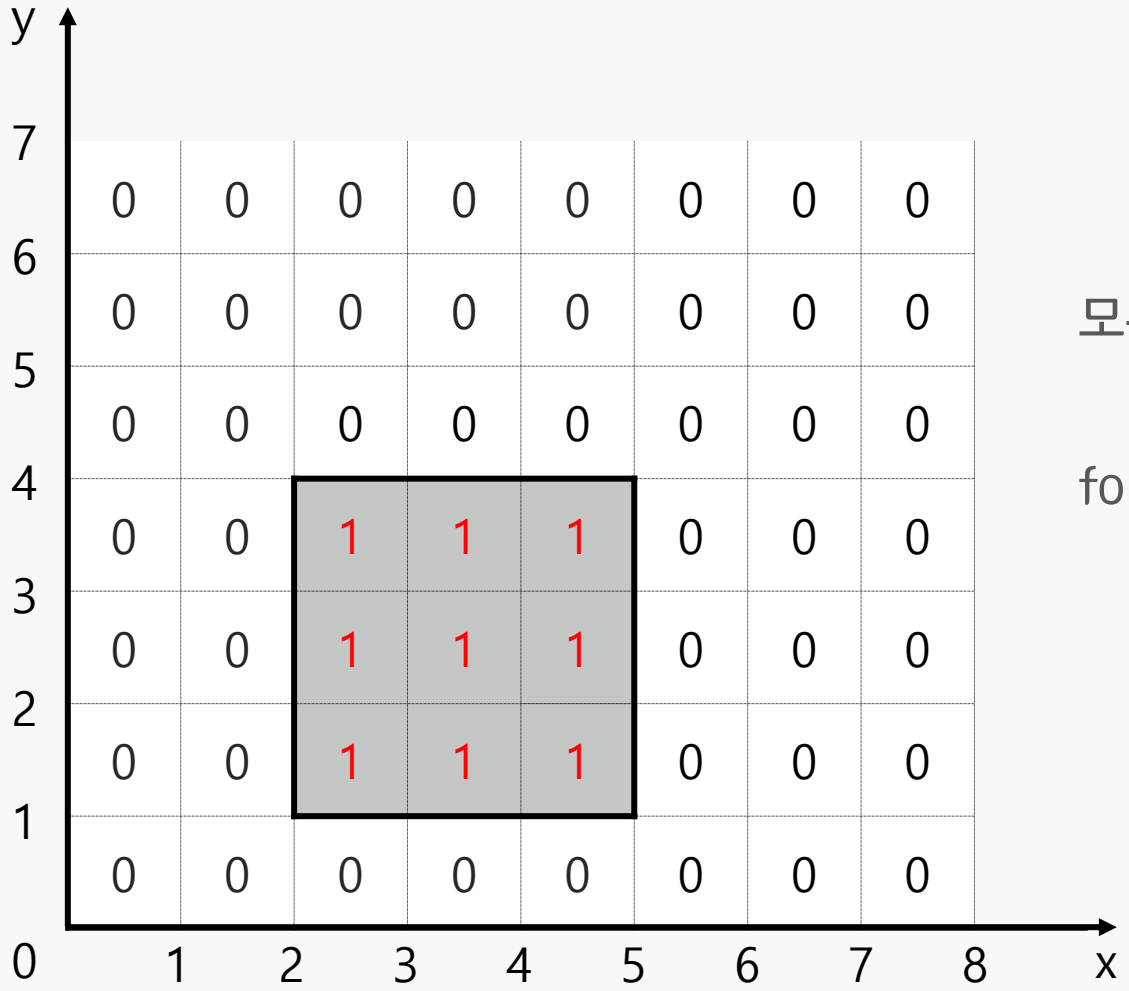
I. 색종이와 쿼리



모든 y축에 대하여 x축 부분합

```
for(int i=0;i<1500;i++)
        for(int j=1;j<1500;j++)
                mat[i][j] += mat[i][j-1];
```

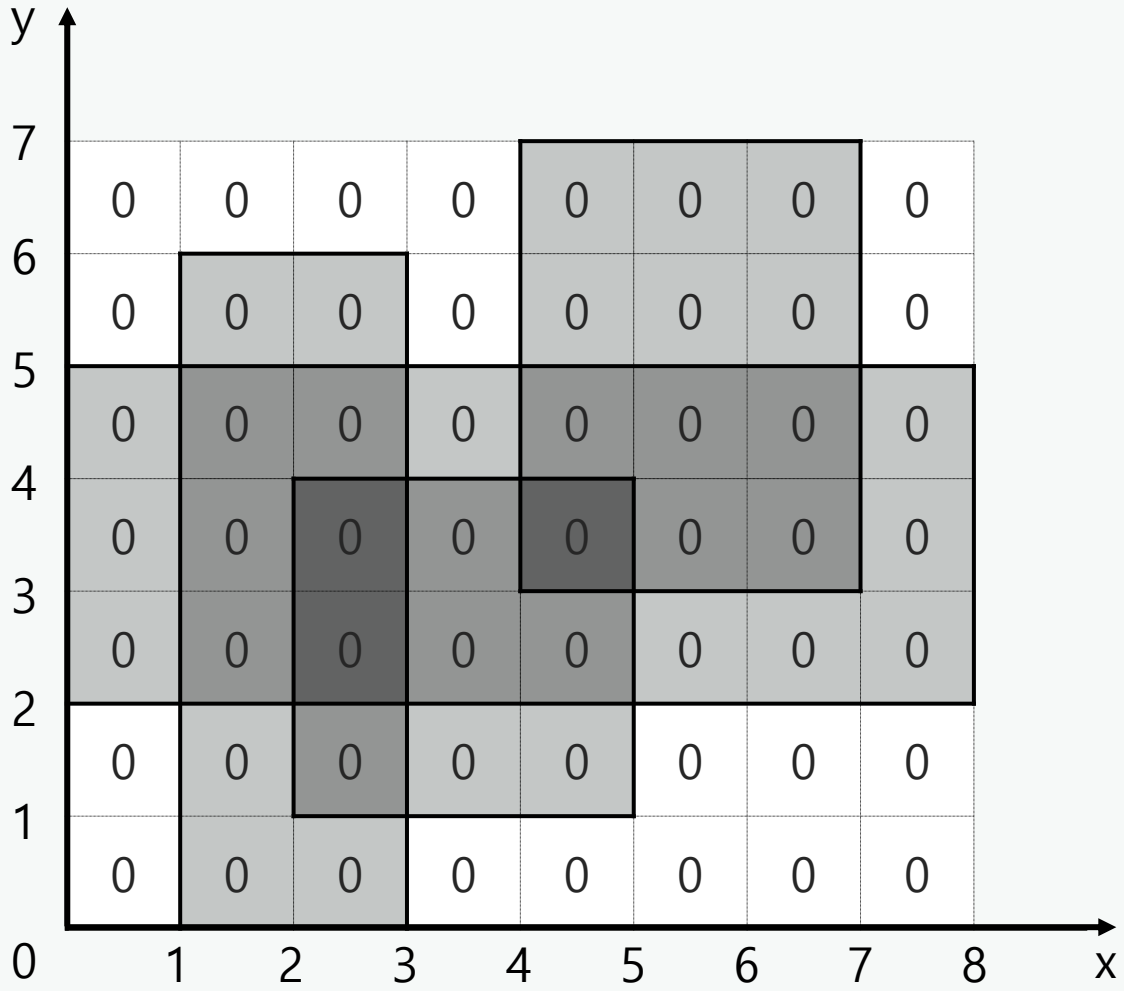
I. 색종이와 쿼리



모든 x축에 대하여 y축 부분합

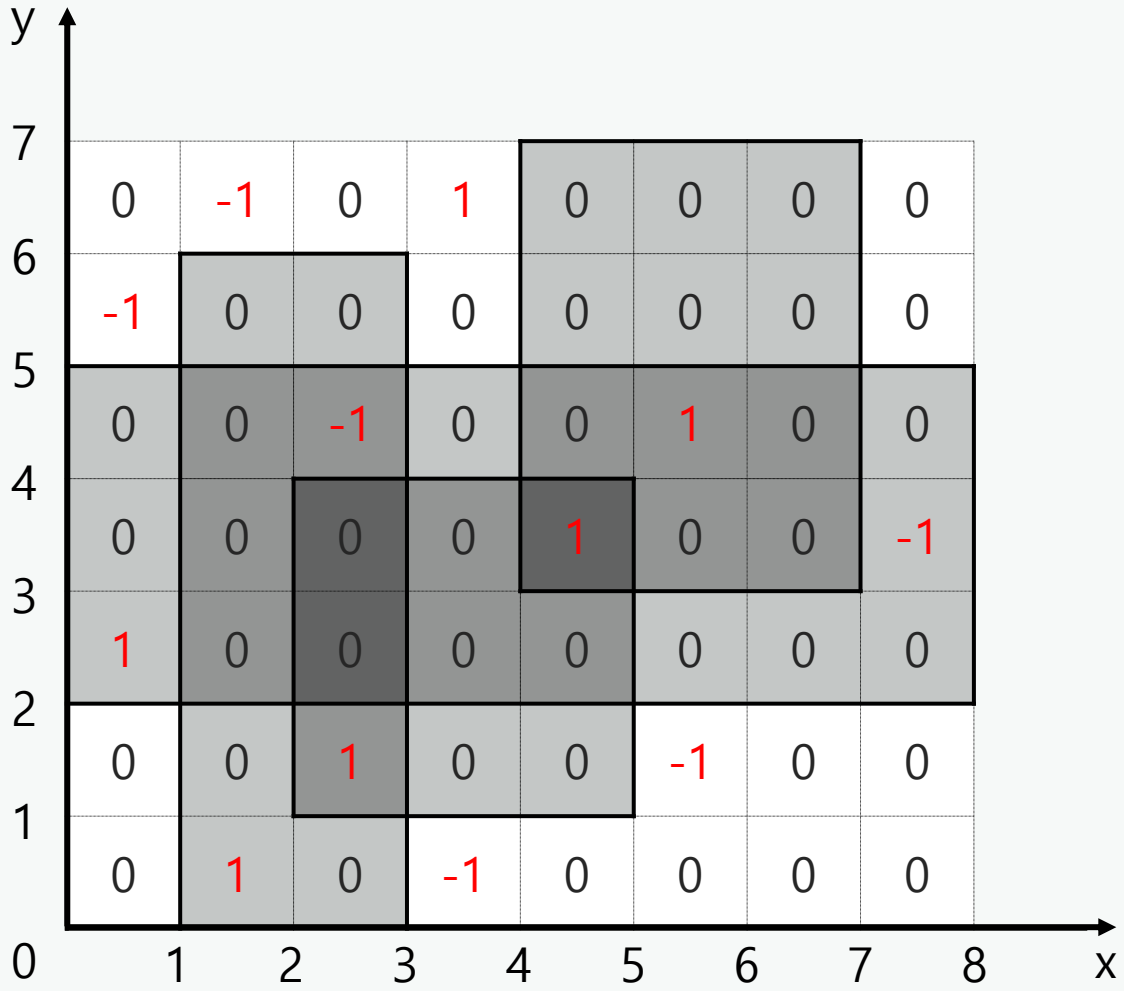
```
for(int j=0;j<1500;j++)
        for(int i=1;i<1500;i++)
                mat[i][j] += mat[i-1][j];
```

I. 색종이와 쿼리



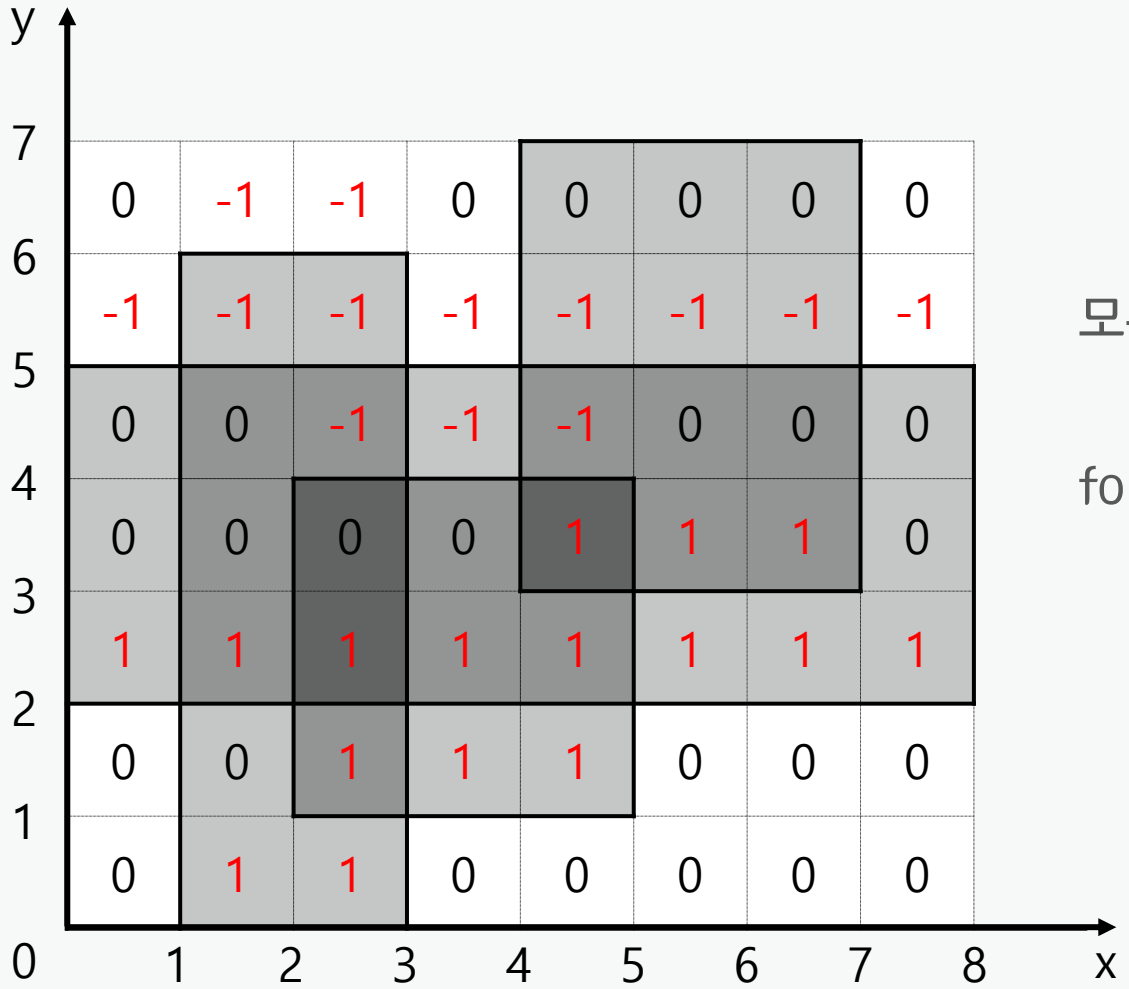
사각형이 여러 개 라면??

I. 색종이와 쿼리




```
mat[y1][x1]++;
mat[y1][x2]--;
mat[y2][x1]--;
mat[y2][x2]++;
```

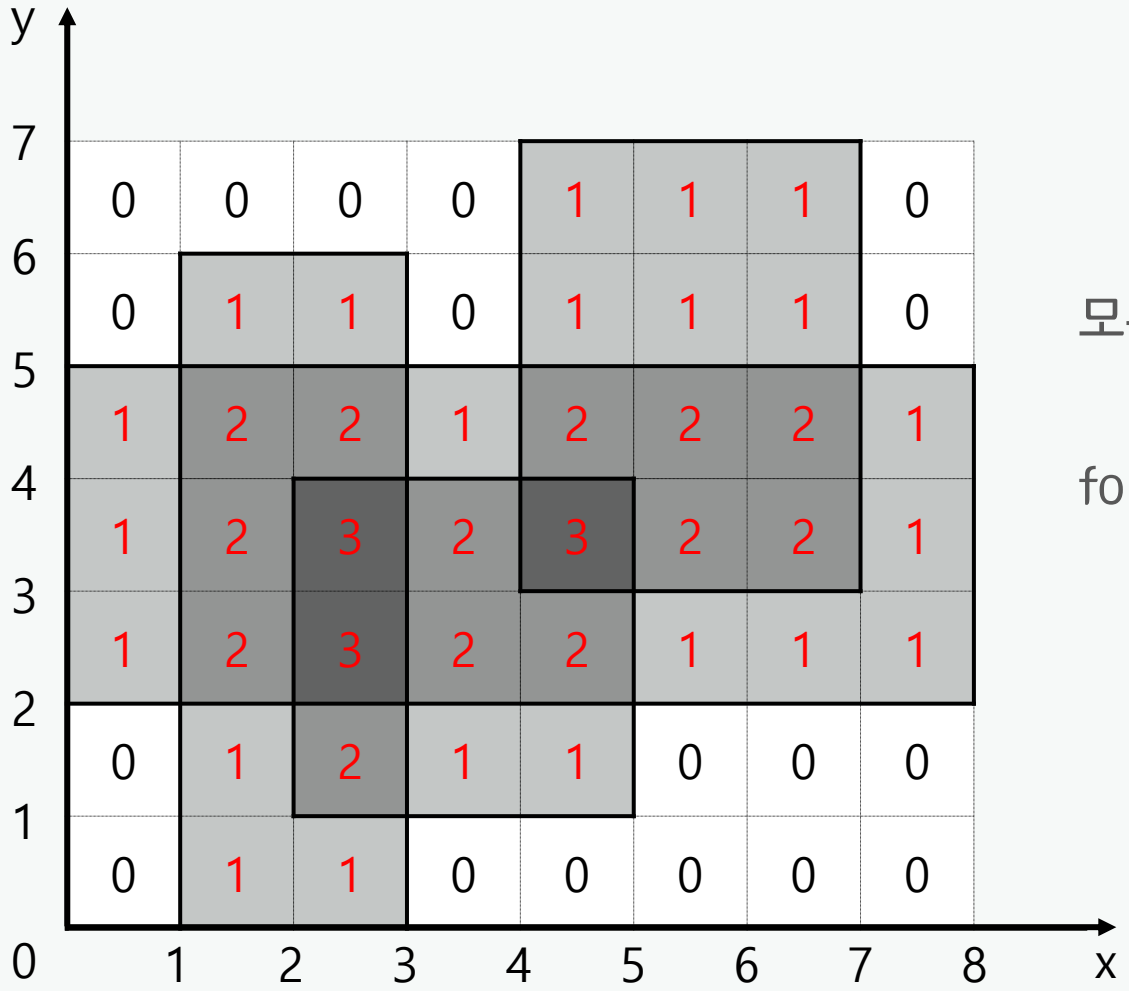
I. 색종이와 쿼리



모든 y축에 대하여 x축 부분합

```
for(int i=0;i<1500;i++)
        for(int j=1;j<1500;j++)
                mat[i][j] += mat[i][j-1];
```

I. 색종이와 쿼리



모든 x축에 대하여 y축 부분합

```
for(int j=0;j<1500;j++)
        for(int i=1;i<1500;i++)
                mat[i][j] += mat[i-1][j];
```

I. 색종이와 쿼리

- 1단계의 시간복잡도 : 좌표의 범위를 P라고 할 때 $O(P^2)$

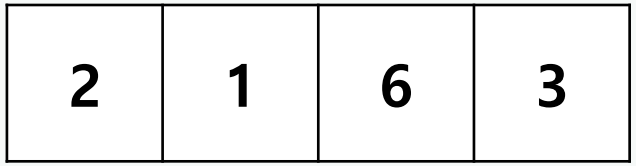
I. 색종이와 쿼리

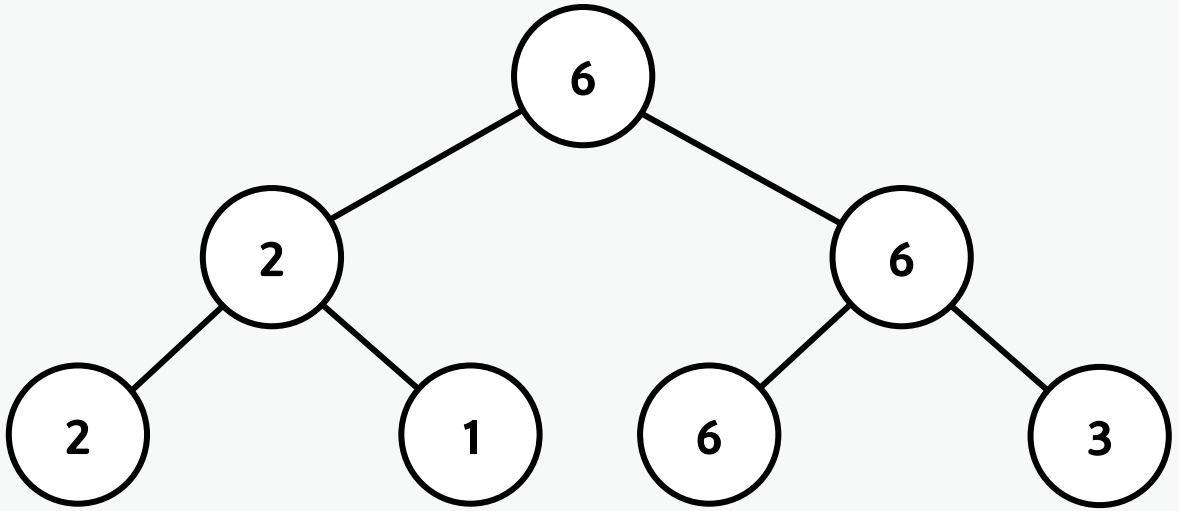
- 2단계 : 직사각형 영역에서의 최댓값을 구해보자
  - 1차원 배열에서의 최댓값은 세그먼트 트리로 구할 수 있을텐데…
  - 2차원 배열에서의 최댓값은 어떻게 하면 좋을까?
    - 2D Segment tree

I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4

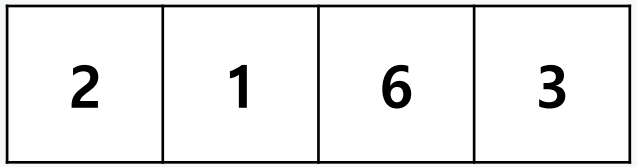
I. 색종이와 쿼리

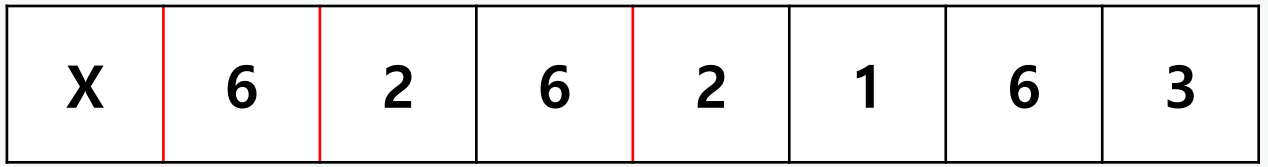




한 행이라면? 구간 최댓값을 다루는 Segment tree

I. 색종이와 쿼리





한 행이라면? 구간 최댓값을 다루는 Segment tree

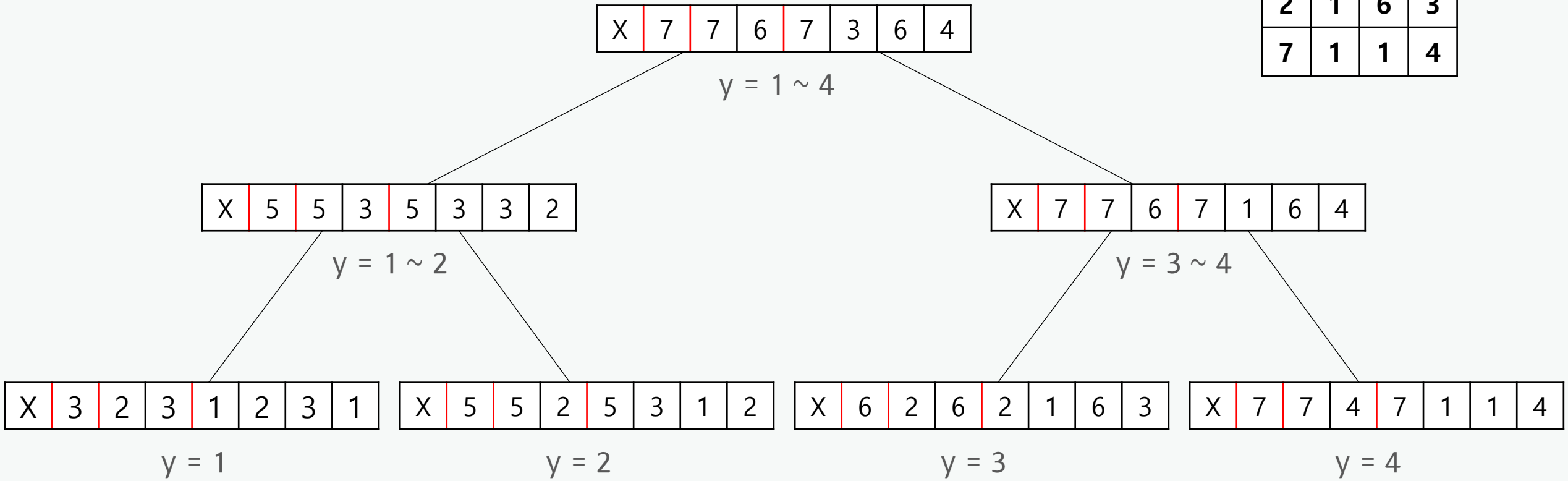
I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4

X	3	2	3	1	2	3	1
X	5	5	2	5	3	1	2
X	6	2	6	2	1	6	3
X	7	7	4	7	1	1	4

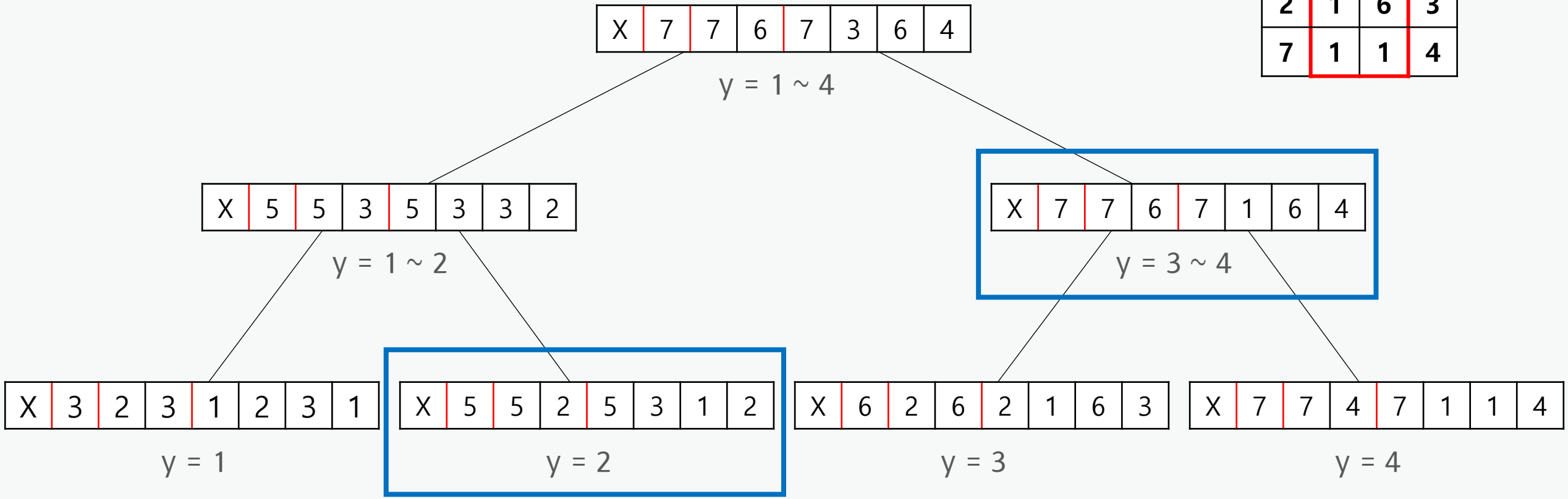
I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4



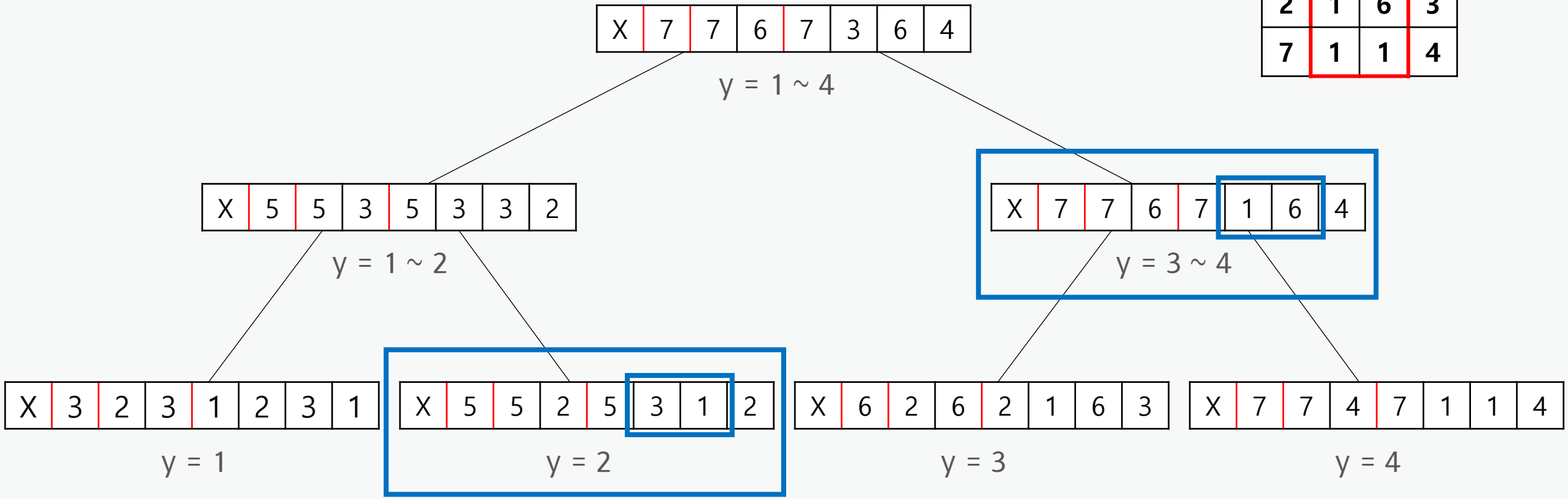
I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4



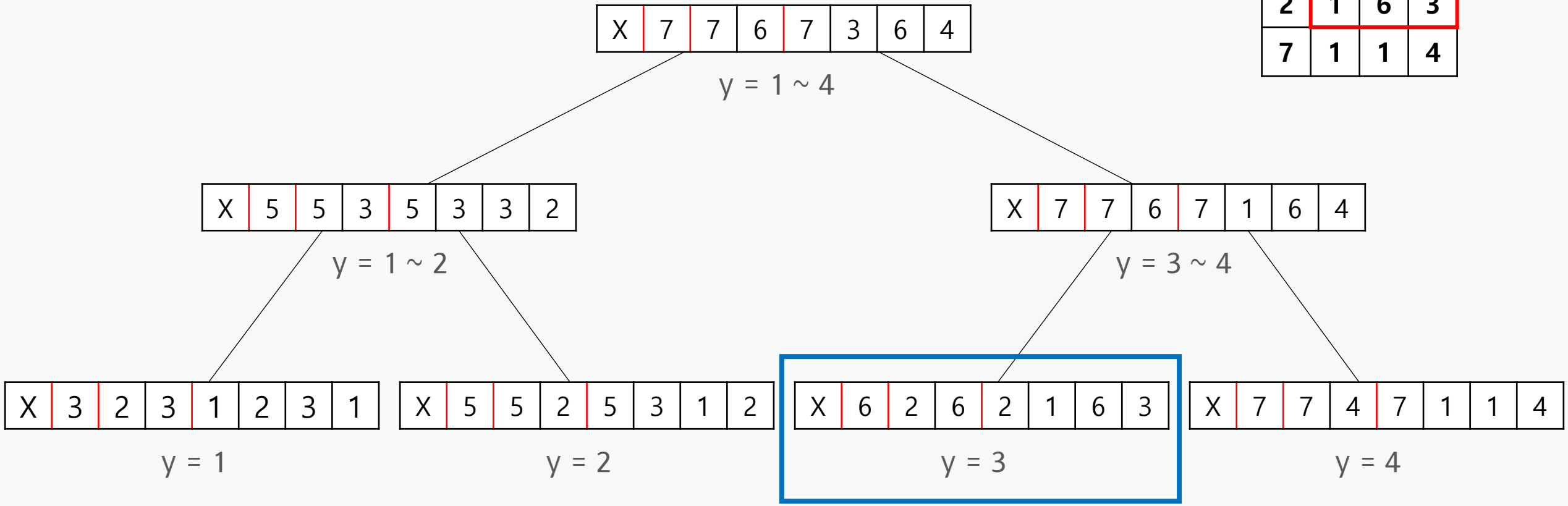
I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4



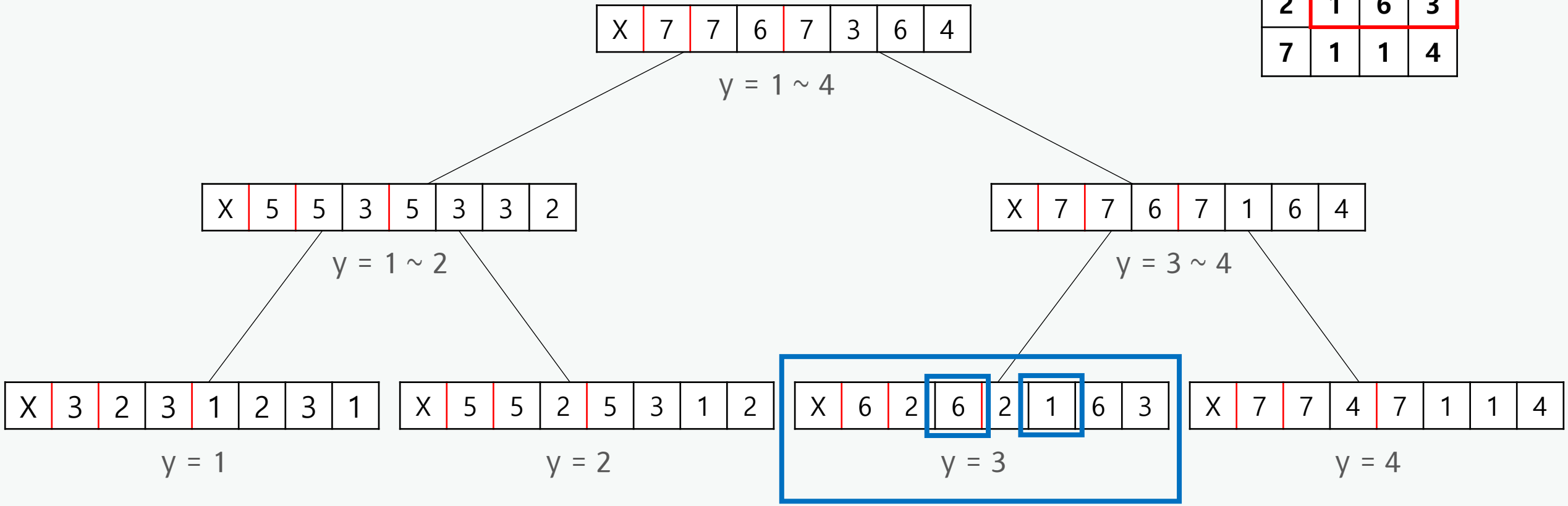
I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4



I. 색종이와 쿼리

1	2	3	1
5	3	1	2
2	1	6	3
7	1	1	4



I. 색종이와 쿼리

- 2단계의 시간복잡도 : 좌표의 범위를 P라고 할 때

  2D segment tree init : $O(P^2)$

  2D segment tree query : O(logP × logP)

I. 색종이와 쿼리

- 최종 시간 복잡도 : $O(P^2 + P^2 + M \times \lg P \times \lg P)$
- 최종 공간 복잡도 : $O(P^2)$

I. 색종이와 쿼리

- Tag
  - 부분합
  - 세그먼트 트리
  - 2D 세그먼트 트리

- First Solve : ??? (? min)

시상

3등

- 상품
  - 50,000원

- 수상자
  - 거시기팀
  - 거시기팀

2등

- 상품
  - 100,000원
- 수상자
  - 거시기팀

1등

- 상품
  - 150,000원
  - ZOAC 2020 문제 출제권
- 수상자
  - 임승현팀

특별상 시상

출제진의 의도를 정확히 파악하셨습니다.

- 조건
  - 출제진이 생각한 난이도 순서대로 풀기
  - + A -> H -> E -> B 보다 A -> H
- 상품
  - 문화상품권 5,000원
- 수상자
  - 이잉~앗살라말라이쿰~

말잘드뤄

- 조건
  - 알파벳 순서대로 풀기
  - + A -> B -> C -> E 보다 A -> B
- 상품
  - 문화상품권 5,000원
- 수상자
  - 배도라지즙

말안드뤄

- 조건
  - 첫 솔브가 A가 아님
- 상품
  - 문화상품권 5,000원
- 수상자
  - ???

모든 문제는 BOJ에 올라갑니다.