Introduction to Offline Query

2019-2020 Winter

20141574 임지환 (Sogang University)

- Online Vs. Offline

Online : 각 쿼리에 대한 access를 **이후에 주어지는 쿼리를 받지 않고** 처리(받는 대로 처리)

Offline : 모든 쿼리를 받아 놓은 후 처리

- Online Vs. Offline

Online : 각 쿼리에 대한 access를 **이후에 주어지는 쿼리를 받지 않고** 처리(받는 대로 처리)

Offline : 모든 쿼리를 받아 놓은 후 처리

Why?

Offline query의 사용 조건

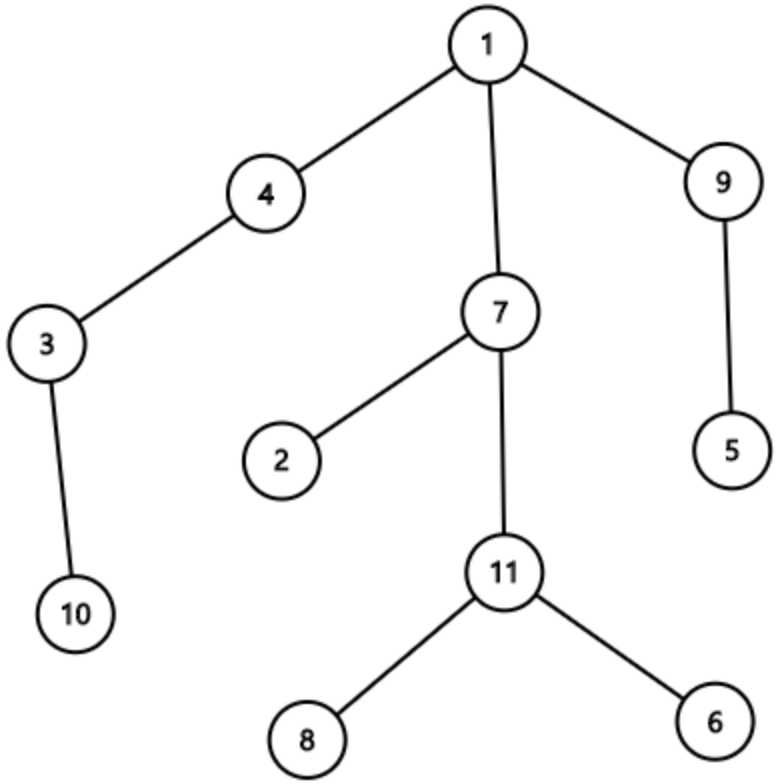
- 특정 쿼리 문제들의 경우, 쿼리를 랜덤으로(입력 순서대로) 처리 하는 것이 어렵다.
- 쿼리 각각을 분리하여 처리하는 것보다 **연관성을 고려하여 적절히 정렬**한 후 처리

  13544 수열과 쿼리 3

#13306 트리

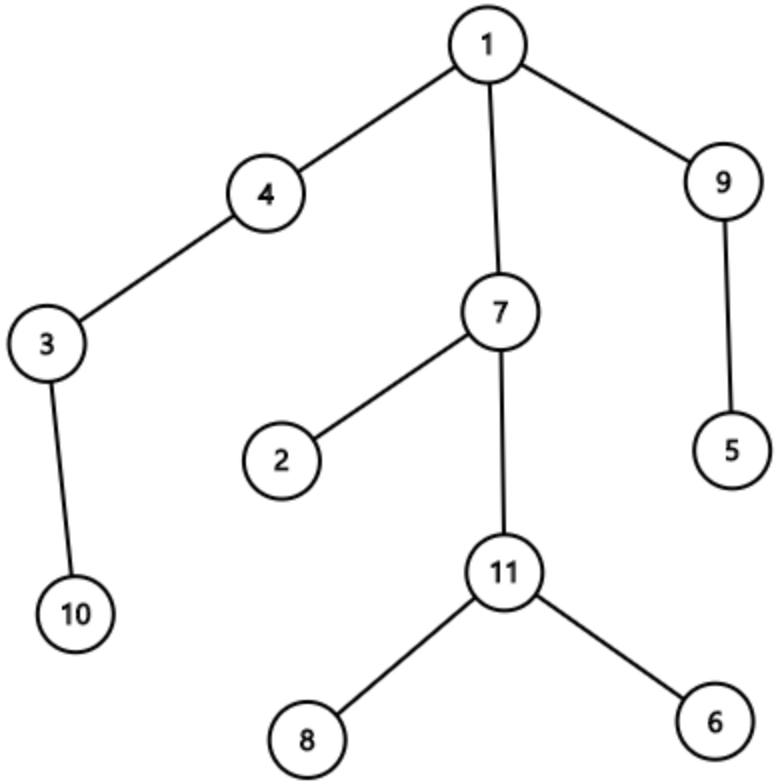
- 노드 $N(1 \leq N \leq 200{,}000)$개로 구성된 루트가 1인 트리
- 각 노드에 대해 부모 정점이 주어짐
- 두 종류의 쿼리 $(1 \leq Q \leq 200{,}000)$
  1. U의 부모 정점과 U를 연결하는 edge 제거
  2. U와 V가 주어졌을 때 U에서 V로 가는 경로 유무
- Update(edge 제거)쿼리의 경우, N-1개가 주어짐.

- Naïve Solution



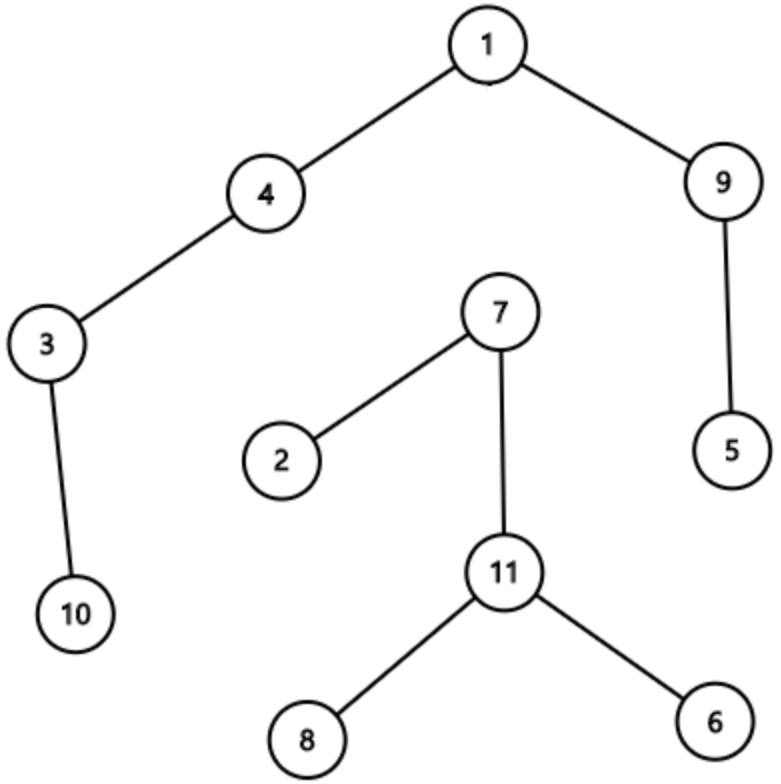
- Naïve Solution

1. 경로가 존재 = 같은 connected component
   - Can solve by Union-Find



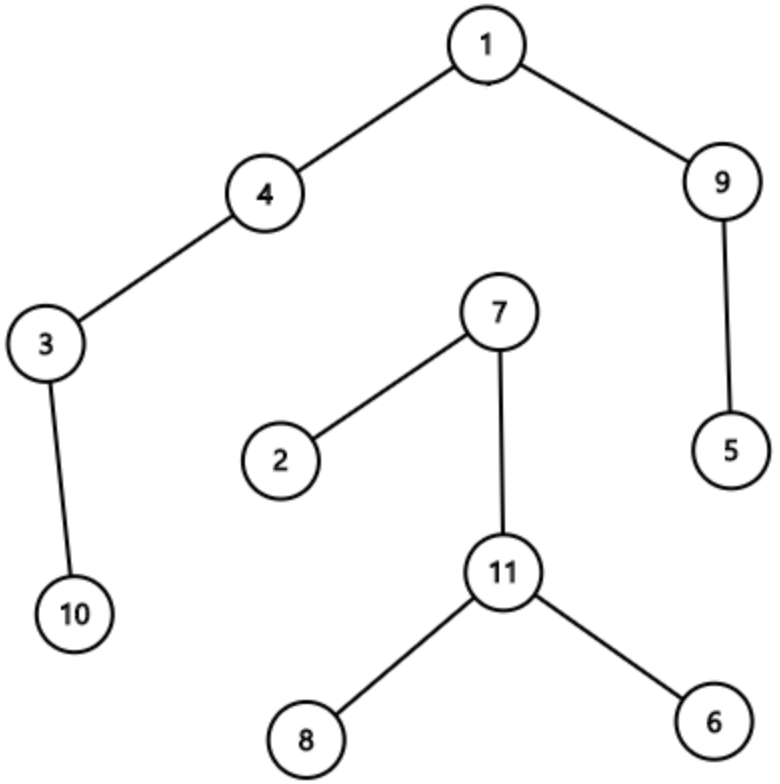
- Naïve Solution

1. 경로가 존재 = 같은 connected component
   - Can solve by Union-Find
2. $\overline{(1,7)}$이 제거된다면?



- Naïve Solution

1. 경로가 존재 = 같은 connected component
   - Can solve by Union-Find
2. $\overline{(1,7)}$이 제거된다면?
   - Need to update parent of every 7's children include '7'



- **문제 해결 과정**

1. Update 쿼리가 N-1개

Link

- **문제 해결 과정**

1. Update 쿼리가 N-1개 = 최종적으로 모든 간선이 제거

Link

- **문제 해결 과정**

1. Update 쿼리가 N-1개 = 최종적으로 모든 간선이 제거
2. 주어진 쿼리를 역순으로 본다면 :

Link

- **문제 해결 과정**

1. Update 쿼리가 N-1개 = 최종적으로 모든 간선이 제거
2. 주어진 쿼리를 역순으로 본다면 :
    1) Connection check : same
    2) 간선 제거 쿼리 : 간선 추가

Link

- **문제 해결 과정**

1. Update 쿼리가 N-1개 = 최종적으로 모든 간선이 제거
2. 주어진 쿼리를 역순으로 본다면 :
   1) Connection check : same
   2) 간선 제거 쿼리 : 간선 추가
3. 쿼리 정보를 모두 받은 후 역순으로 처리

Link

Grid Compression

좌표 압축

- 문제에서 주어지는 좌표(or 수열의 값)의 range가 아주 큰 경우
- but **등장하는 좌표의 개수가 적은 경우** 각각의 값을 특정 index에 대응시키는 방법

Ex)

$10^{18}+1$	3	50	2×10^{7}	$10^{6}+9$	10,003	$10^{9}+7$	1	$10^{18}+2$	$10^{18}+3$

Ex)

상대적 위치에 따라 대응 :

$10^{18}+1$	3	50	2×10^{7}	$10^{6}+9$	10,003	$10^{9}+7$	1	$10^{18}+2$	$10^{18}+3$

Ex)

상대적 위치에 따라 대응 :

$10^{18}+1$	3	50	2×10^{7}	$10^{6}+9$	10,003	$10^{9}+7$	1	$10^{18}+2$	$10^{18}+3$
0	1	2	3	4	5	6	7	8	9

Ex)

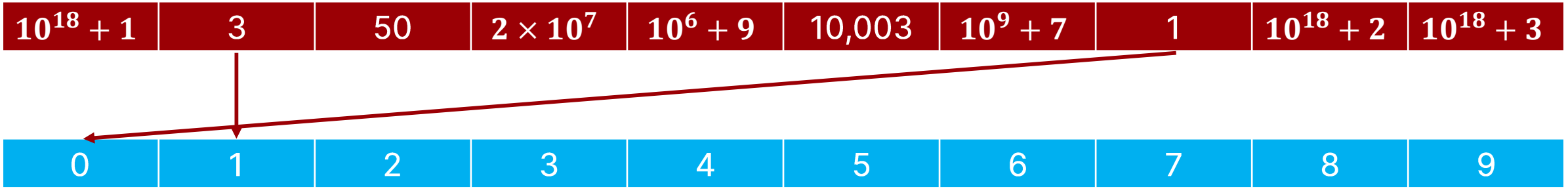
상대적 위치에 따라 대응 :

$10^{18}+1$	3	50	2×10^{7}	$10^{6}+9$	10,003	$10^{9}+7$	1	$10^{18}+2$	$10^{18}+3$

0	1	2	3	4	5	6	7	8	9

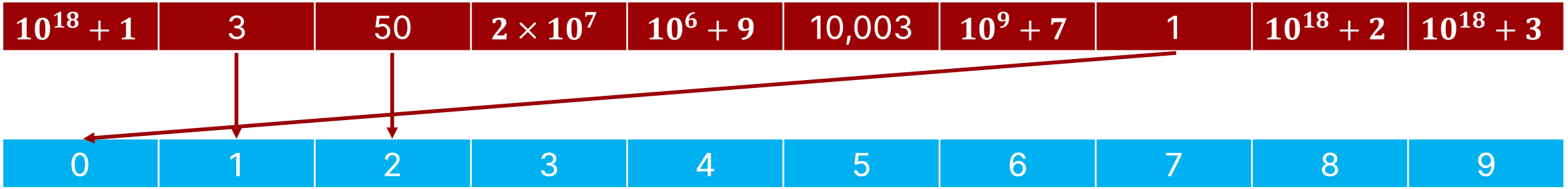
Ex)

상대적 위치에 따라 대응 :



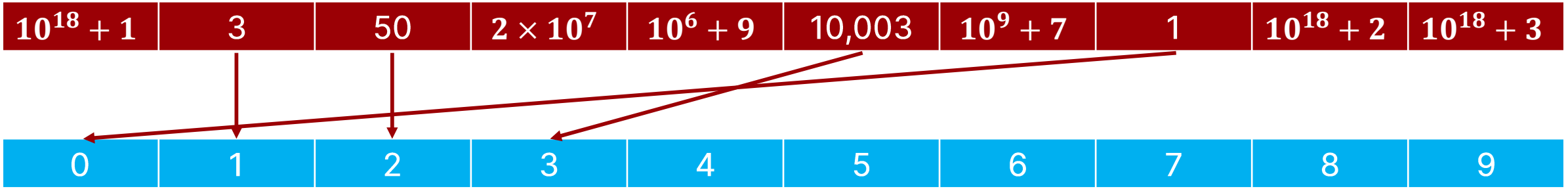
Ex)

상대적 위치에 따라 대응 :



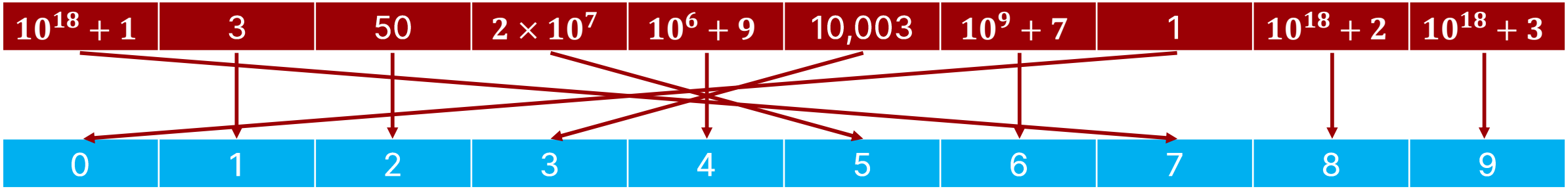
Ex)

상대적 위치에 따라 대응 :



Ex)

상대적 위치에 따라 대응 :



Ex)

상대적 위치에 따라 대응 :

7	1	2	5	4	3	6	0	8	9

1	3	50	10,003	$10^6 + 9$	2×10^7	$10^9 + 7$	$10^{18} + 1$	$10^{18} + 2$	$10^{18} + 3$

- Compression with set & map

```
 4
 5    set<int> xlist;
 6    map<int, int> xs;
 7
 8    cin >> N;
 9    for (int i = 0; i < N; i++) {
10        cin >> a[i].x;
11        xlist.insert(a[i].x);
12    }
13    int idx = 0;
14    for (int &it : xlist) xs[it] = idx++;
15    for (int i = 0; i < N; i++)
16        a[i].x = xs[a[i].x];
17
```

#5 : 등장하는 모든 x좌표의 관리를 위한 set

#6 : 배열 값의 상대적인 위치 대입을 위한 map

#14 : 각 x값에 대해 상대적 위치 부여

#15,16 : 배열 값을 상대적인 index로 대응(optional)

performance check : Link

- Compression with vector

```
4
5     vector<int> xlist;
6     cin >> N;
7     for (int i = 0; i < N; i++) {
8         cin >> a[i].x;
9         xlist.push_back(a[i].x);
10    }
11    sort(xlist.begin(), xlist.end());
12    xlist.erase(unique(xlist.begin(), xlist.end()), xlist.end());
13    for (int i = 0; i < N; i++)
14        a[i].x = (lower_bound(xlist.begin(), xlist.end(), a[i].x) - xlist.begin());
15
```

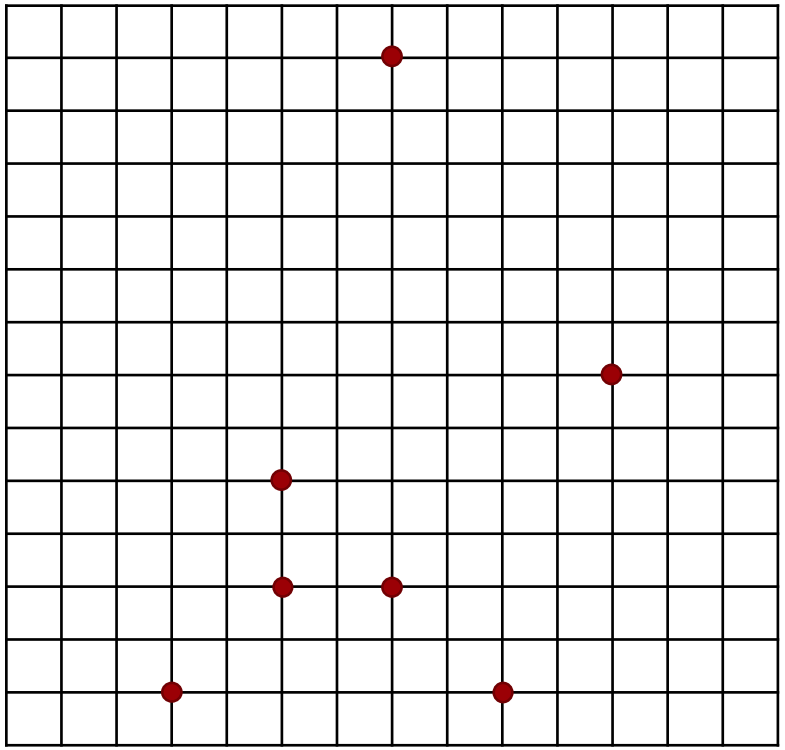
#5 : 등장하는 모든 x좌표의 관리를 위한 vector

#13,14 : 배열 값을 상대적인 index로 대응(optional)

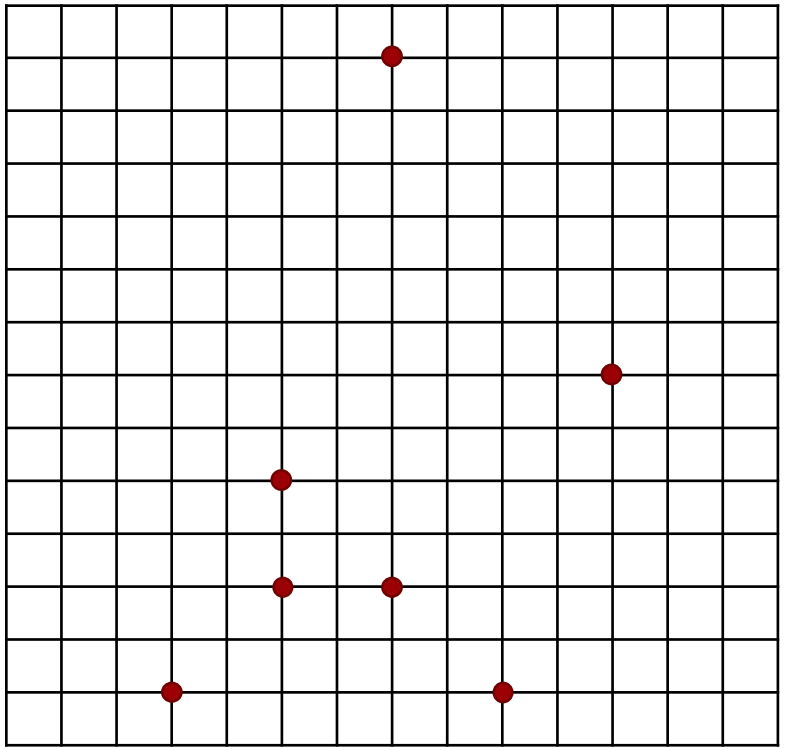
performance check : Link

#11997 Load Balancing

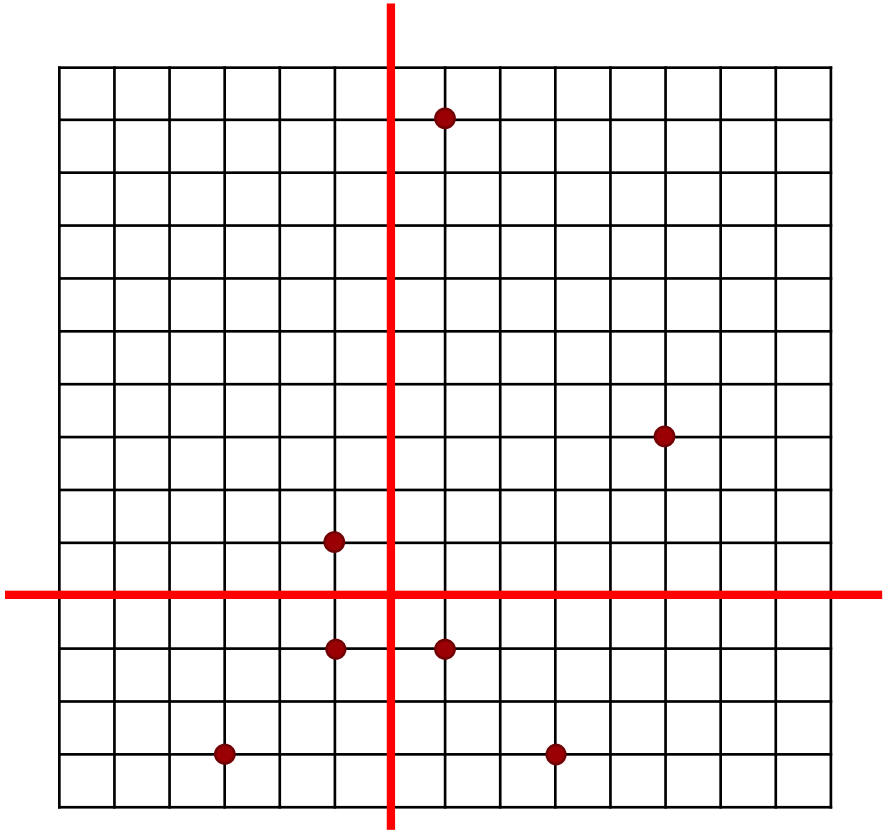
- 2차원 평면
- $N(1 \leq N \leq 1000)$마리의 소가 (x_i, y_i)상에 위치(x_i, y_i는 홀수, $0 \leq x_i, y_j \leq 10^6$)
- $x = a, y = b$ (a, b는 짝수)인 두 직선을 세웠을 때 생기는 네 구역 중 소의 마리 수가 가장 많은 곳의 소의 수를 M이라 할 때, 최소 M의 값?



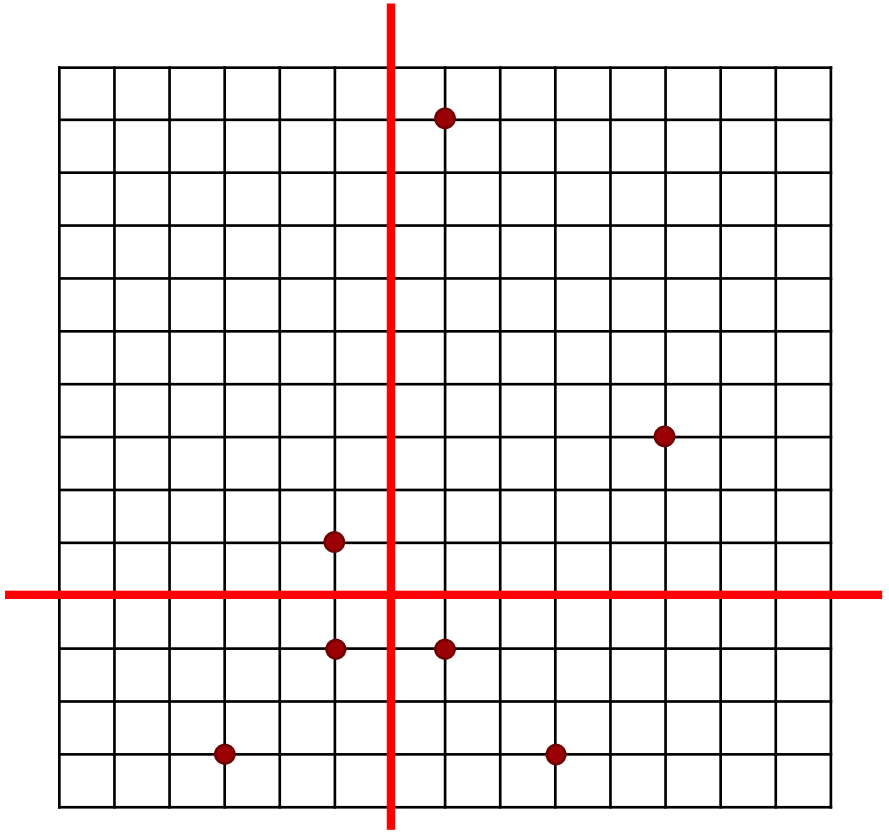
- Naïve solution



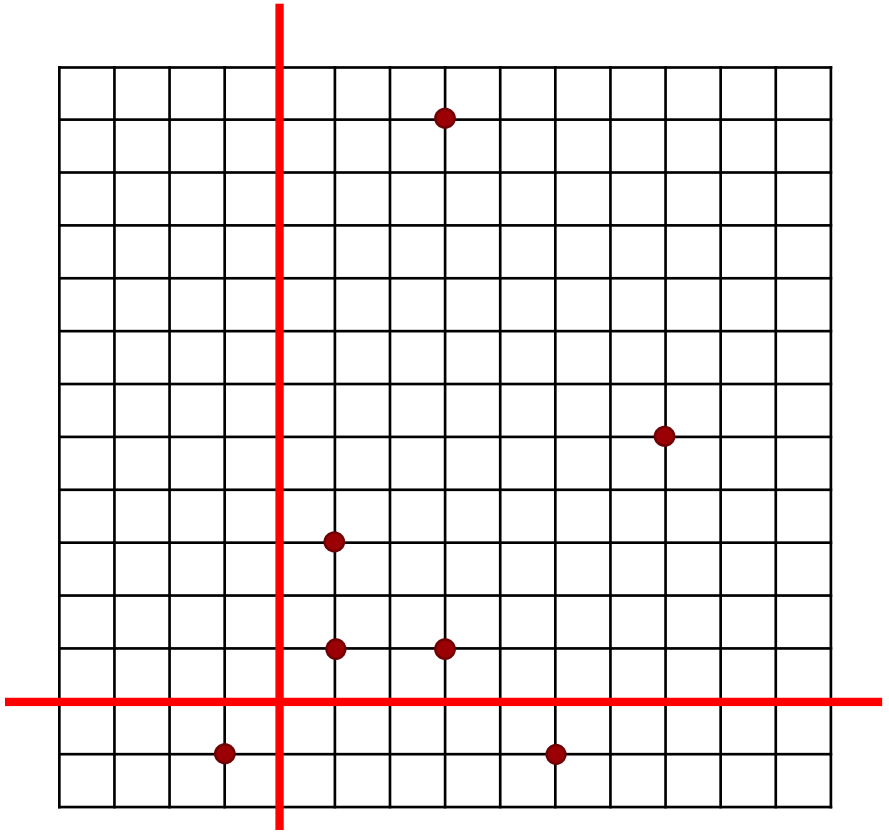
- Naïve solution



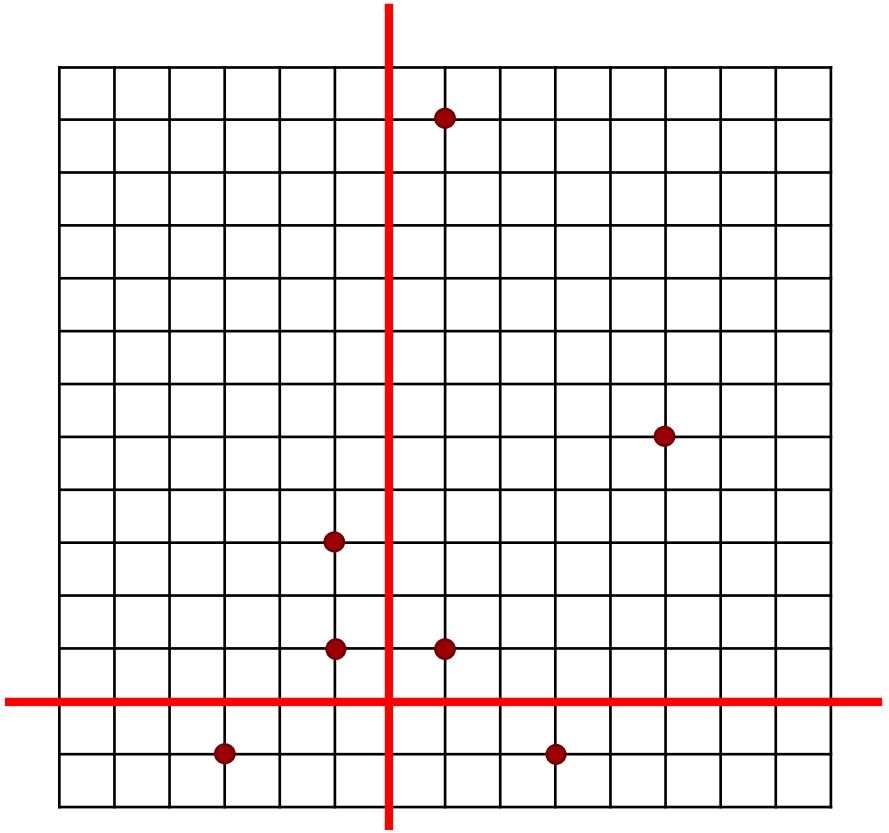
- Naïve solution
  - $psum[i][j] := (0,0) \sim (i,j)$을 끝점으로 하는 사각형에 포함되는 점의 개수
  - 전처리 : $O(K^2)$ $(K : size\ of\ grid)$



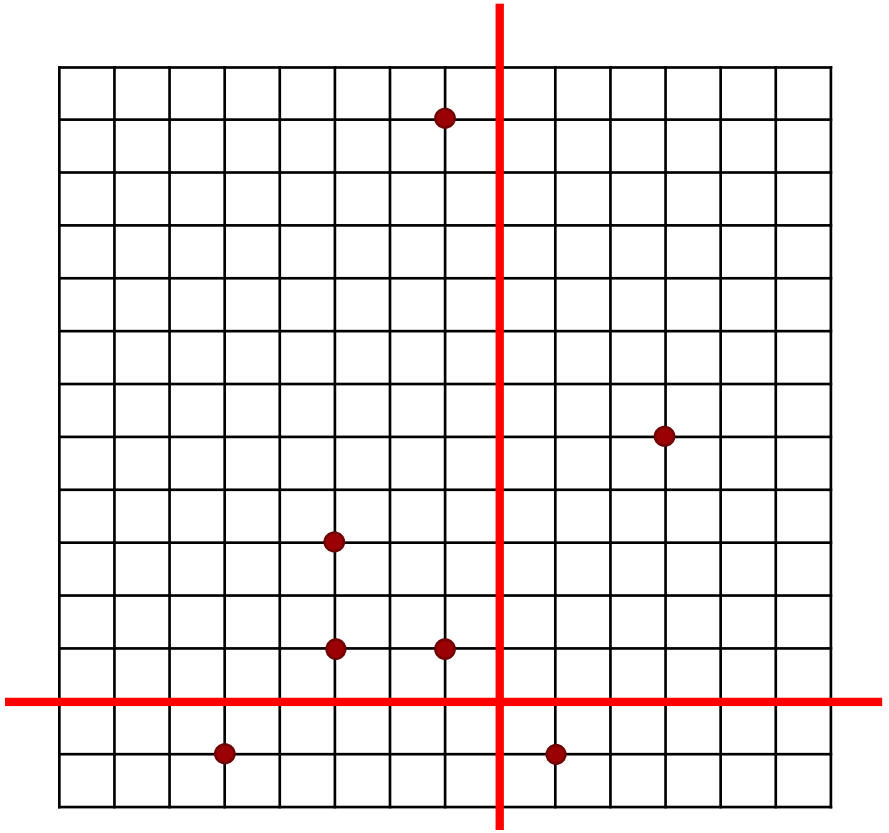
#11997 Load Balancing



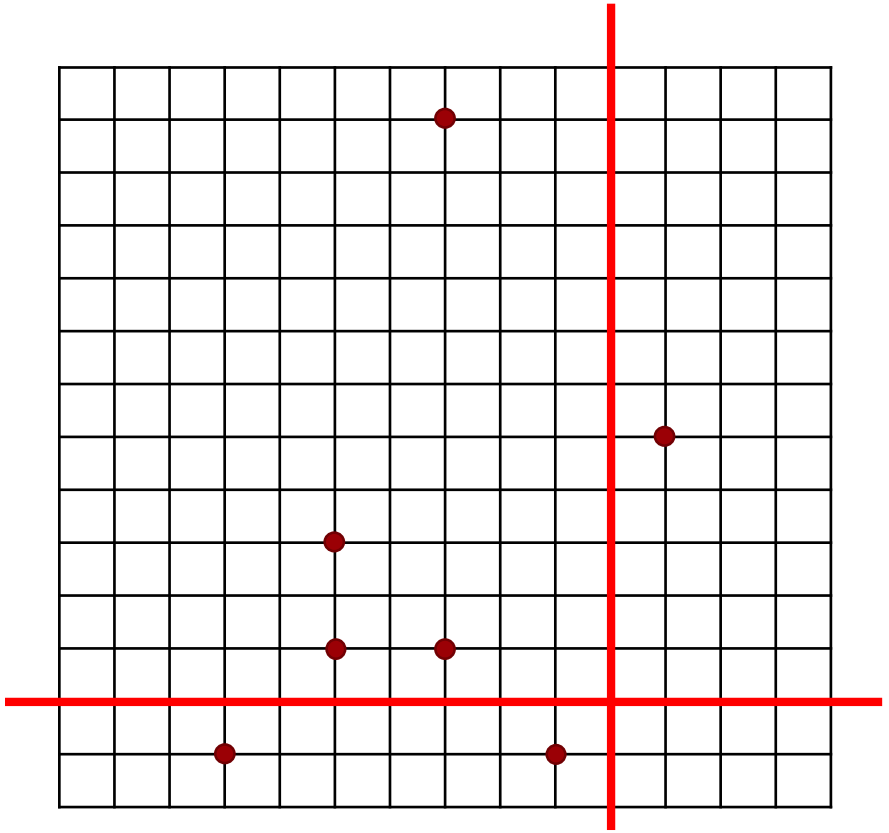
#11997 Load Balancing



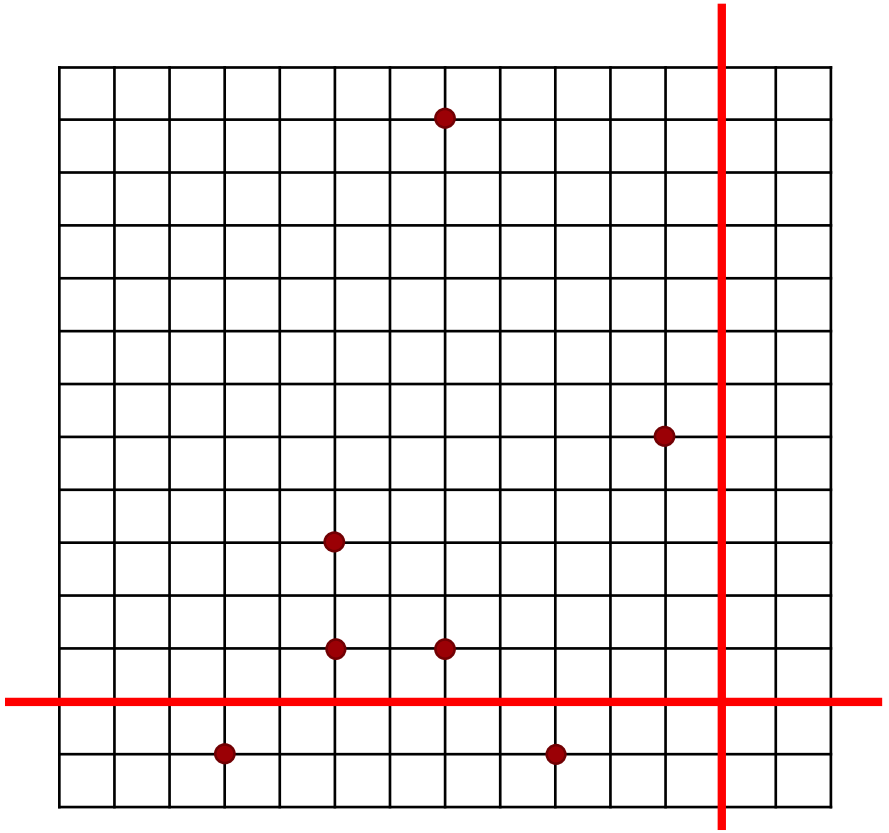
#11997 Load Balancing



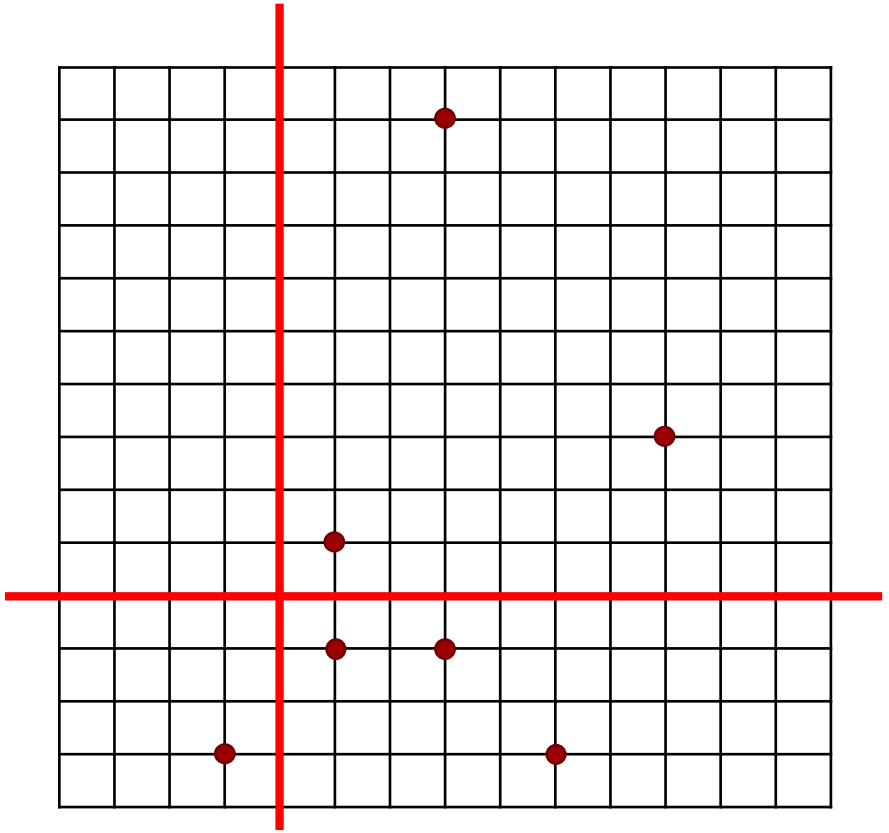
#11997 Load Balancing



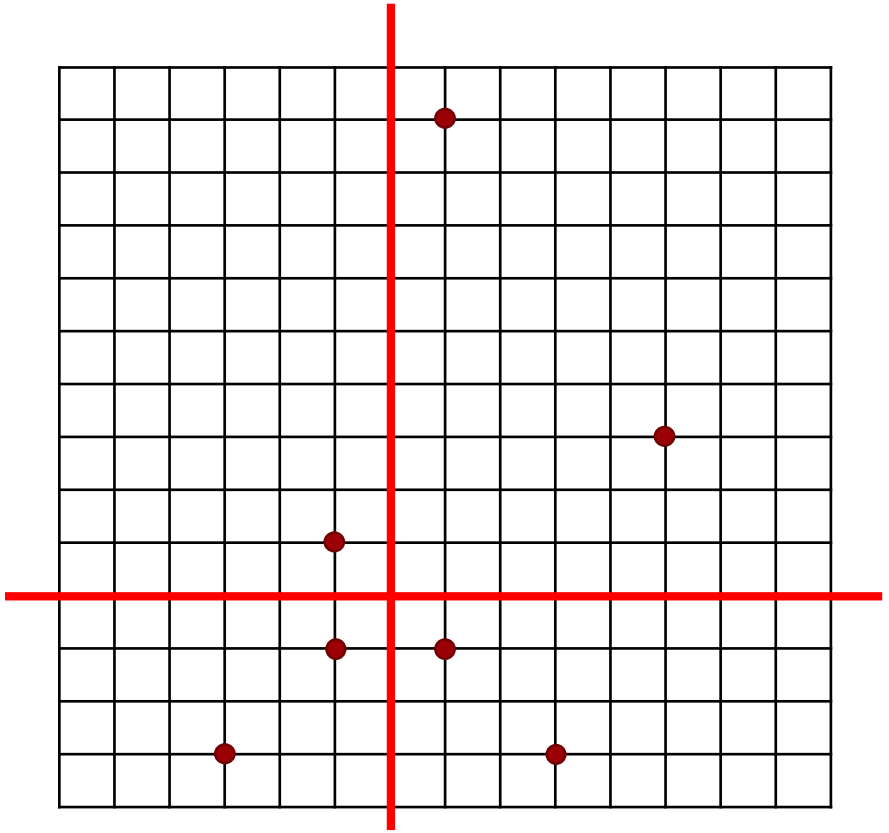
#11997 Load Balancing



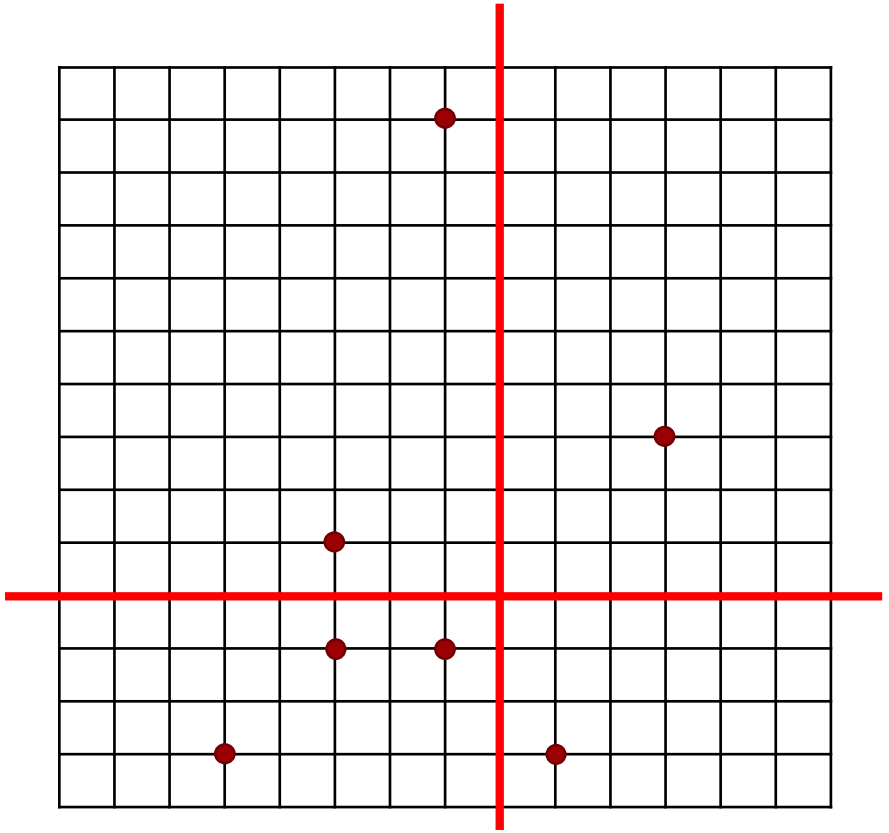
#11997 Load Balancing



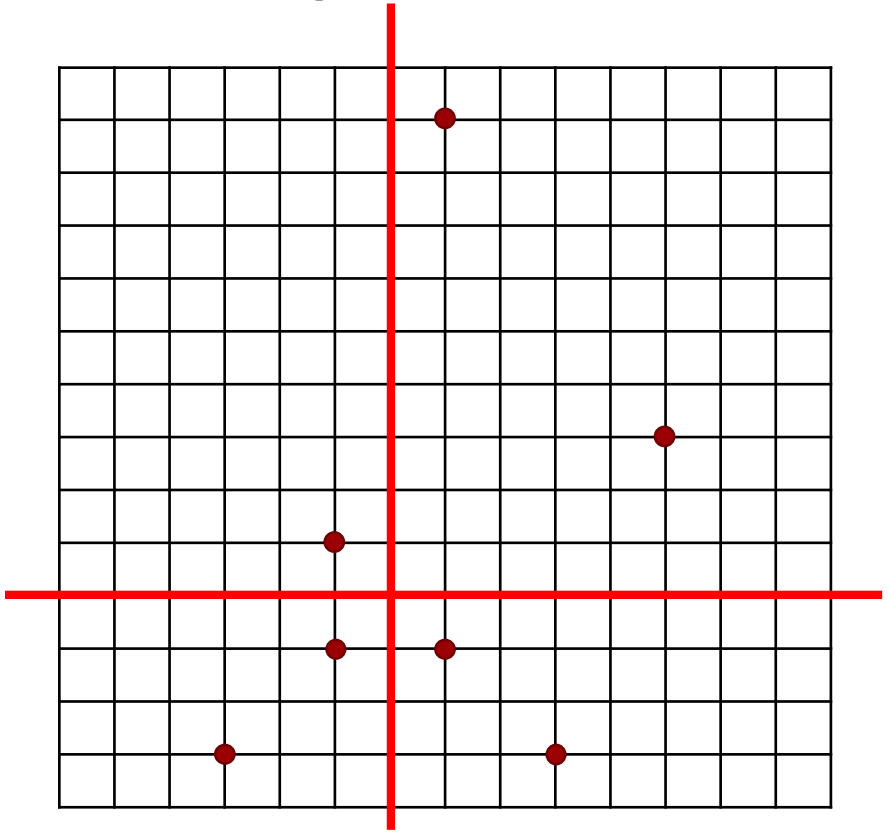
#11997 Load Balancing



#11997 Load Balancing



- Naïve solution
  - $psum[i][j] := (0,0) \sim (i,j)$을 끝점으로 하는 사각형에 포함되는 점의 개수
  - 전처리 : $O(K^2)$ $(K : size\ of\ grid)$
- 좌표의 크기가 크므로 $10^6 \times 10^6$의 grid 생성 불가



- 문제 해결 과정

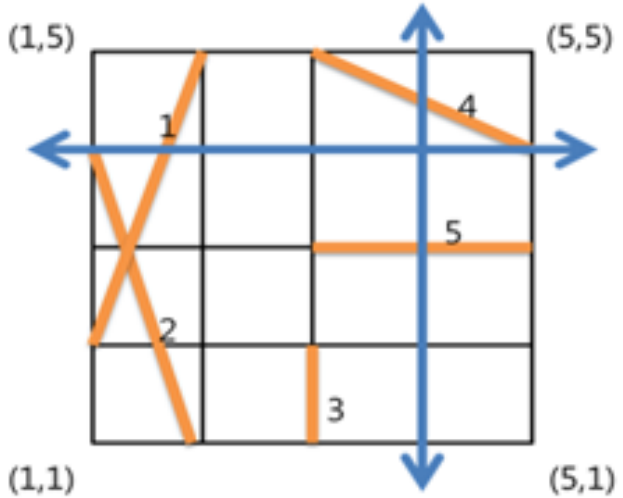
1. 좌표의 실질적인 값이 중요하지도, 직선의 값도 중요한 값은 아니다.
   ⇒좌표 압축
2. prefix sum 전처리
3. 등장하는 모든 x, y에 대해 4개의 partition 중 max값을 구하고, 그 중 minimum 구하기

Link

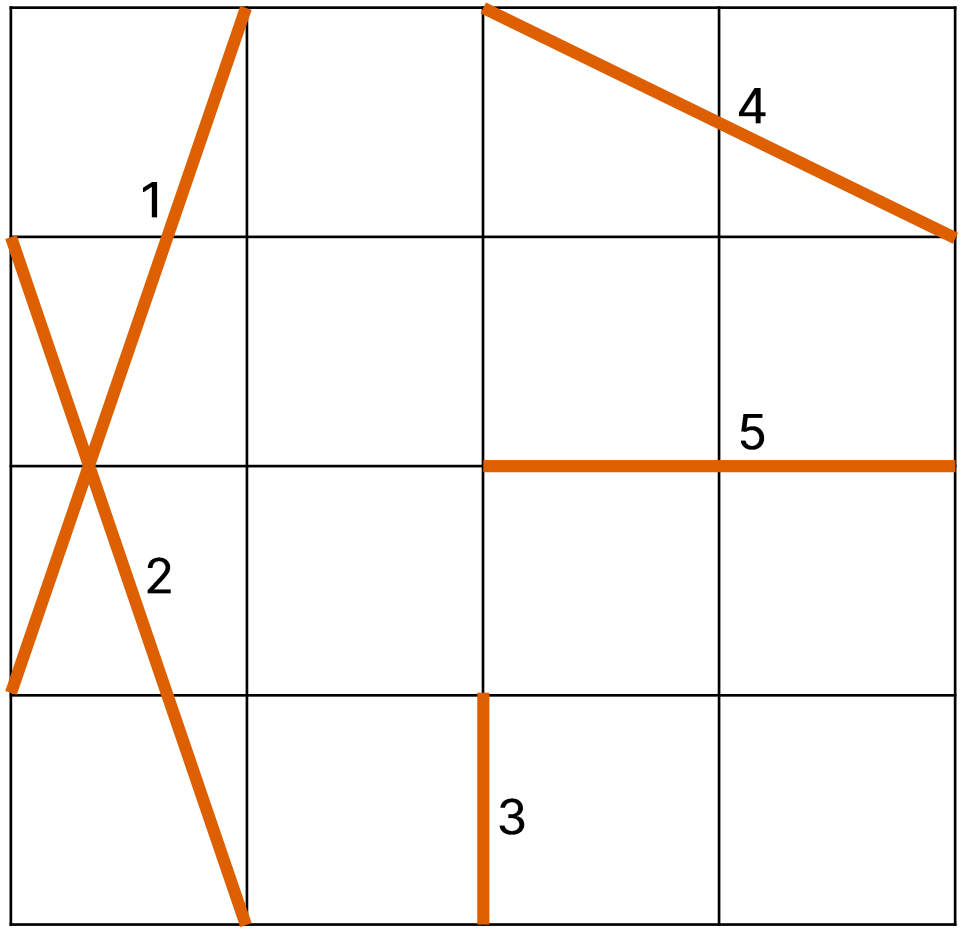
#5480 전함

- $N \times N$크기의 좌표 $(1 \leq N \leq 10^9)$
- 전함의 수 k, 대포 발사 횟수 l $(1 \leq k, l \leq 100{,}000)$
- 전함의 정보 : 양 끝점 $(x, y), (x', y'), w$ $\quad (1 \leq x, y, x', y' \leq N), (1 \leq w \leq 10^6)$
- 대포의 정보 : 수직 또는 수평, 관통성

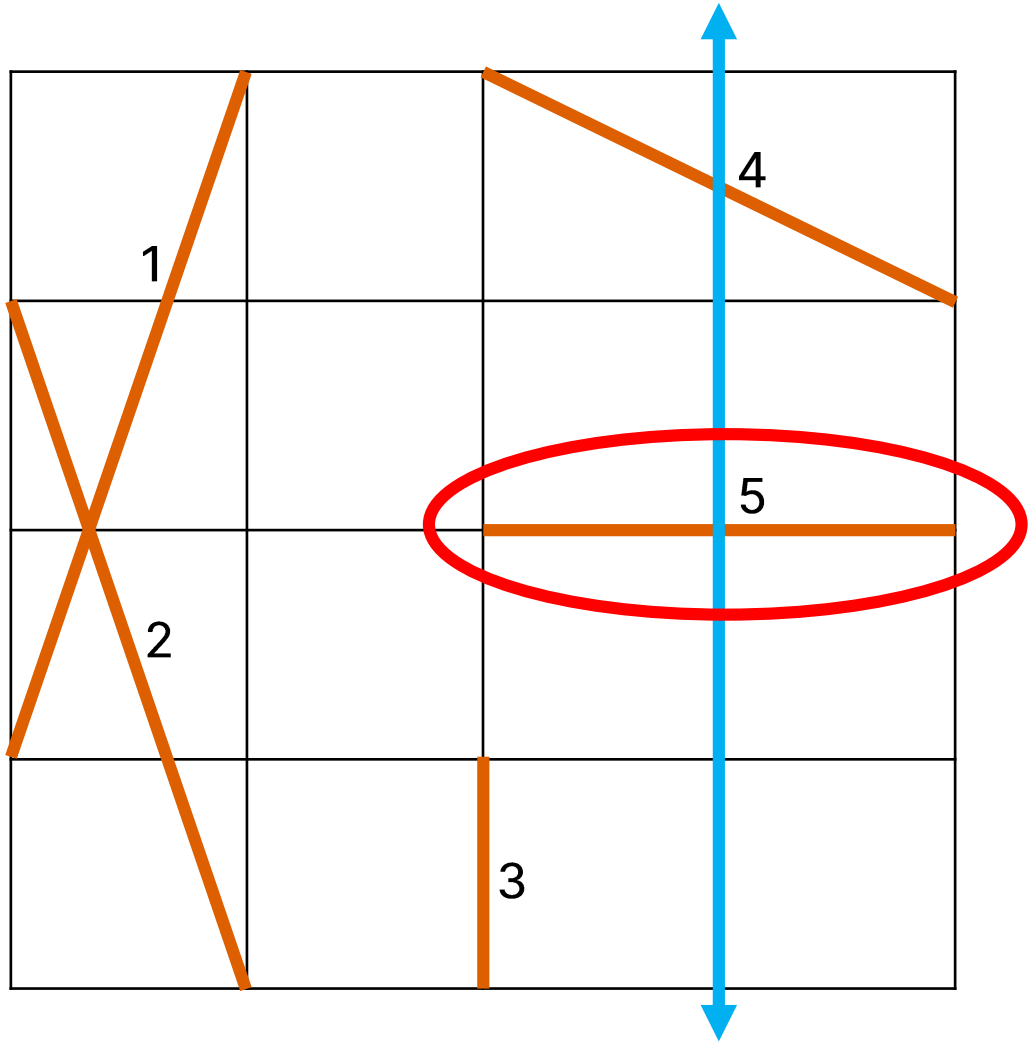
- 대포를 발사할 때마다, 파괴한 전함 중 가장 무거운 전함의 무게



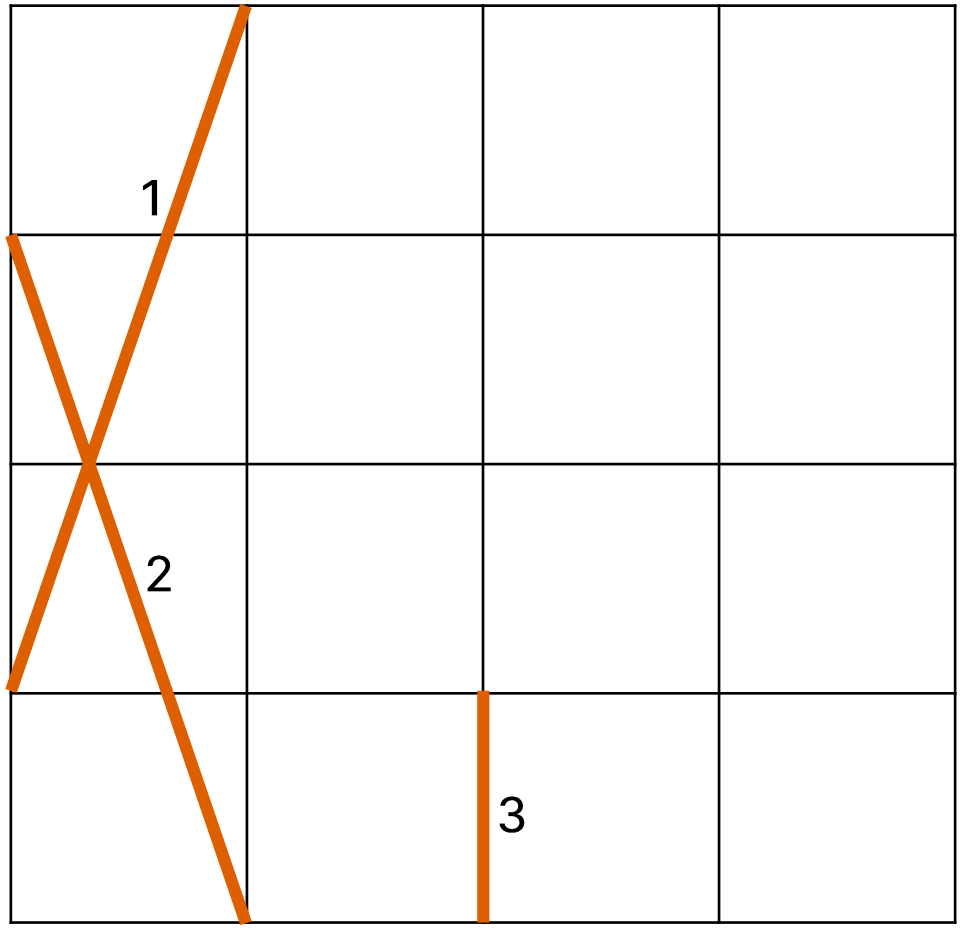
Ex) First Laser : $x = 4$



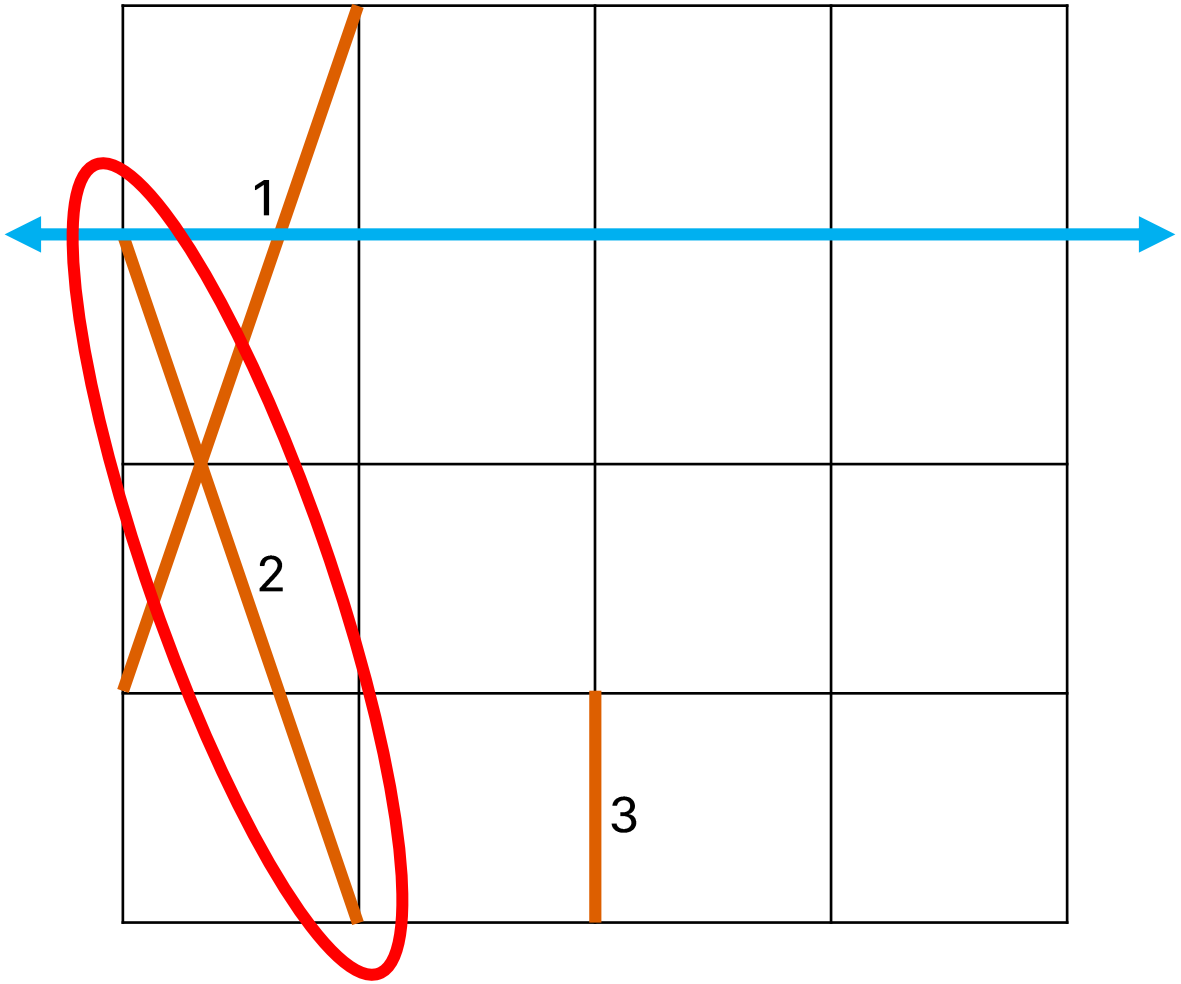
Ex) First Laser : $x = 4$



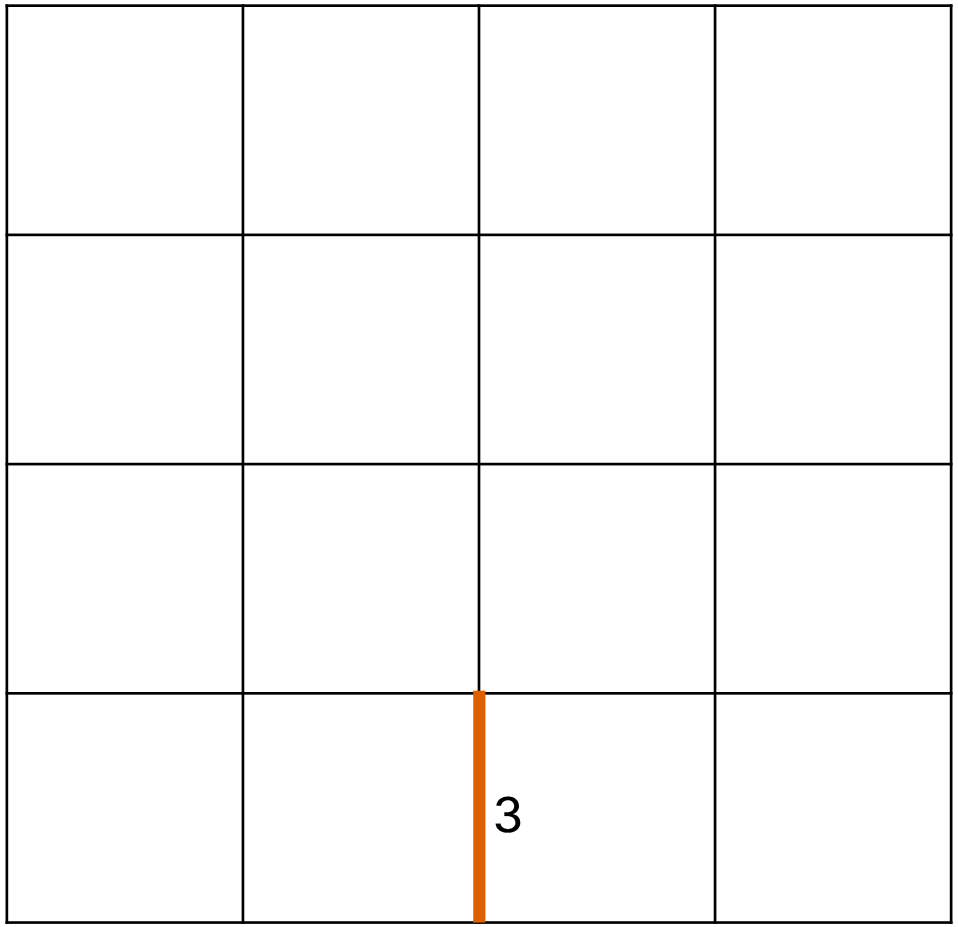
Ex) Second Laser : $y = 4$



Ex) Second Laser : $y = 4$



Ex)



- **문제 해결 과정**

1. 좌표의 크기가 크므로 좌표압축 필요

- **문제 해결 과정**

1. 좌표의 크기가 크므로 좌표압축 필요
2. Target : 각 대포마다 부술 수 있는 가장 무거운 전함

- 문제 해결 과정

1. 좌표의 크기가 크므로 좌표압축 필요
2. Target : 각 대포마다 부술 수 있는 가장 무거운 전함
   = 각 전함이 몇 번째 대포에 의해 부서지는가?

- **문제 해결 과정**

1. 좌표의 크기가 크므로 좌표압축 필요
2. Target : 각 대포마다 부술 수 있는 가장 무거운 전함
   = 각 전함이 몇 번째 대포에 의해 부서지는가?
3. 무거운 전함부터 보며 몇 번째 대포에 의해 부서지는지 계산

- **문제 해결 과정**

1. 좌표의 크기가 크므로 좌표압축 필요
2. Target : 각 대포마다 부술 수 있는 가장 무거운 전함
   = 각 전함이 몇 번째 대포에 의해 부서지는가?
3. 무거운 전함부터 보며 몇 번째 대포에 의해 부서지는지 계산
4. 전함은 x좌표, y좌표에 대한 range로 표현되므로 배를 표현하는
   x range 내의 minimum laser index
   y range 내의 minimum laser index 중 minimum value
5. 전함 data & laser data 모두 좌표 압축 필요

#13537 수열과 쿼리 1

- 크기 $N(1 \leq N \leq 100{,}000)$인 수열
- $(l, r, k) \coloneqq [l, r]$구간에서 k보다 큰 원소의 개수?

- 문제 해결 과정

1. 주어진 range에 대하여 k보다 큰 원소의 개수

- 문제 해결 과정

1. 주어진 range에 대하여 k보다 큰 원소의 개수
   = k보다 큰 원소만 update한 상태로 구간 내 원소 개수 구하기

#13537 수열과 쿼리 1

- 문제 해결 과정

1. 주어진 range에 대하여 k보다 큰 원소의 개수
   = k보다 큰 원소만 update한 상태로 구간 내 원소 개수 구하기
2. 쿼리를 k가 큰 순으로 내림차순 정렬
3. empty array로부터 시작, $Q([l,r],k)$에 대하여 k보다 큰 값 만을 update 후 쿼리 처리
   Update : $O(NlogN)$
   Query : $O(QlogQ)$

- Square-root decomposition

N개의 값들을 $\sqrt{N}$개의 연속된 구간들로 나누어 관리

- Mo's algorithm

Update query가 없는 경우, 구간 범위+길이에 따라 적절히 순서를 바꿔서 처리

#17398 통신망 분할
#15586 MooTube (Gold)
#13306 트리
#2843 마블
#17469 트리의 색깔과 쿼리
#16978 수열과 쿼리 22
#15899 트리와 색깔

- Grid Compression required

#11997 Load Balancing (Silver)
#2672 여러 직사각형의 전체 면적 구하기
#15589 Lifeguards (Silver)
#13537 수열과 쿼리 1
#14577 일기예보
#5480 전함