



분할정복법

Divide and Conquer

분할 정복법: Divide and Conquer

1. **[Divide]** 해결하고자 하는 문제를 하나 혹은 그 이상의 더 작은 크기의 **동일한** 문제로 분할한다.
2. **[Conquer]** 각각의 분할된 문제들을 **순환적으로** 해결한다.
3. **[Combine]** 분할된 문제들에 대한 해를 결합하여 원래 문제의 해를 구한다.

- 이진검색(binary search)
- 합병정렬(merge sort)
- 빠른정렬(quicksort)
- 거듭제곱(power)

$$power(x, n) = \begin{cases} 1 & n = 0 \\ power(x^2, n/2) & n \text{ is even} \\ x * power(x^2, (n - 1)/2) & n \text{ is odd} \end{cases}$$

종이 자르기

예제

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

N=8인 경우
(N은 항상 2의 거듭제곱수)

모든 조각들이 단일 색이 될 때 까지 4등분하면
몇개의 조각으로 나뉘어 지는가?

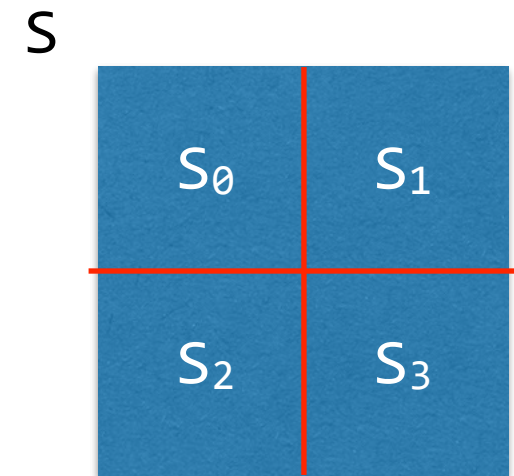
예제

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

4등분한 후
각각의 subsquare에 대해서
카운트하여 더 한다.

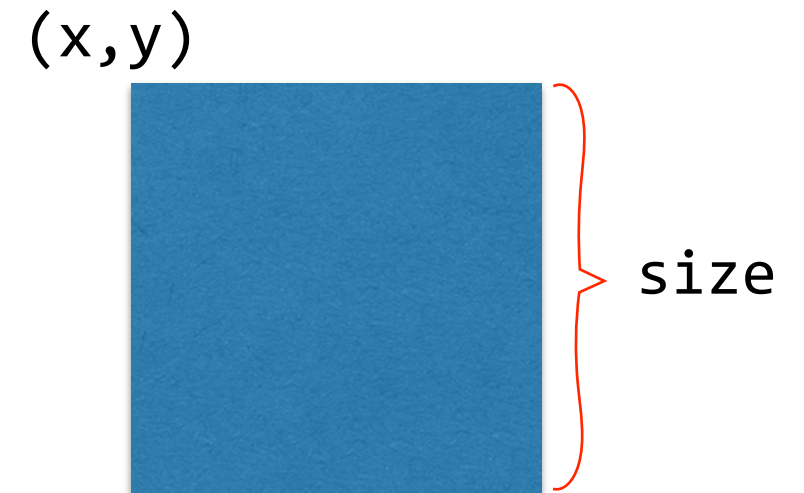
Designing Recursion

```
int count( a square S )
{
    if S is uni-coloured
        return 1;
    else
        let  $S_0$ ,  $S_1$ ,  $S_2$  and  $S_3$  be 4 sub-squares;
        return  $\text{count}(S_0) + \text{count}(S_1) + \text{count}(S_2) + \text{count}(S_3)$ ;
}
```



예제

```
int count( int x, int y, int size )
{
    if (unicolor(x, y, size))
        return 1;
    else
        return count(x,y,size/2)
               + count(x,y+size/2, size/2)
               + count(x+size/2, y, size/2)
               + count(x+size/2, y+size/2, size/2);
}
```



unicolor 테스트



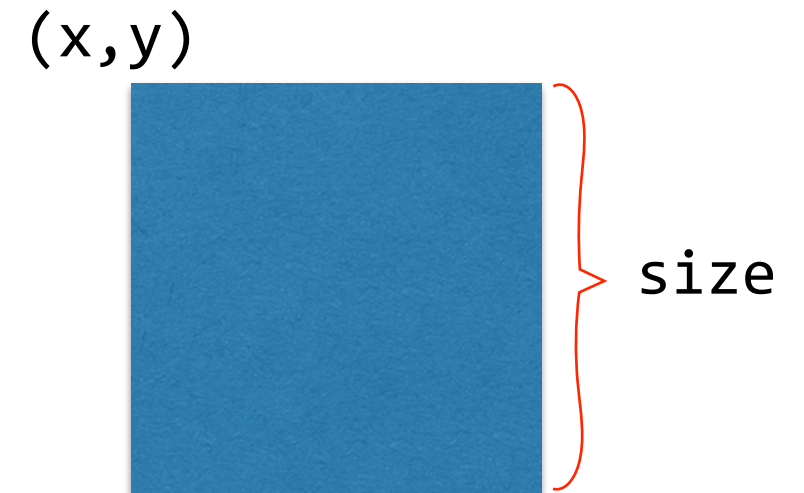
$$\begin{aligned}T(N) &= 4T(N/2) + N^2 \\&= 4\{4T(N/4) + (N/2)^2\} + N^2 = 4^2T(N/2^2) + 2N^2 \\&= 4^2\{4T(N/2^3) + (N/4)^2\} + 2N^2 = 4^3T(N/2^3) + 3N^2 \\&\dots \\&= 4^kT(N/2^k) + kN^2.\end{aligned}$$

Let $N = 2^k$, then

$$T(N) = N^2 + N^2 \log N = O(N^2 \log N)$$

Better Solution

```
int count( int x, int y, int size )
{
    if (size==1)
        return 1;
    else {
        int sum = count(x,y,size/2);
                + count(x,y+size/2, size/2)
                + count(x+size/2, y, size/2)
                + count(x+size/2, y+size/2, size/2);
        int tmp = grid[x][y] + grid[x][y+size/2]
                + grid[x+size/2][y] + grid[x+size/2][y+size/2];
        return (sum>4 || tmp!=0 && tmp!=4 ? sum : 1);
    }
}
```



시간 복잡도

상수 시간



$$\begin{aligned}T(N) &= 4T(N/2) + C \\&= 4\{4T(N/4) + C\} + C = 4^2T(N/2^2) + (1 + 4)C \\&= 4^2\{4T(N/2^3) + C\} + 5C = 4^3T(N/2^3) + (1 + 4 + 4^2)C \\&\dots\end{aligned}$$

$$= 4^kT(N/2^k) + C \sum_{i=0}^{k-1} 4^i = 4^kT(N/2^k) + \frac{4^k - 1}{3}C$$

Let $N = 2^k$, then

$$T(N) = N^2 + \frac{N^2 - 1}{3}C = \Theta(N^2)$$

Counting Inversion

Counting Inversion

- 수열: a_1, a_2, \dots, a_n
- $i < j$ 인데 $a_i > a_j$ 인 경우 (a_i, a_j) 는 역전(inverted)되었다고 말한다.

1	2	3	4	5
1	3	4	2	5

Inversions
3-2, 4-2

- 역전된 쌍의 개수를 카운트하라.
- 응용: 어떤 두 수열이 얼마나 비슷한지에 대한 척도의 역할을 한다.

• Brute Force

- 모든 쌍을 검사
- $O(n^2)$

1. Divide: 둘로 나눈다.

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

Divide: $O(1)$.

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

1. Divide: 둘로 나눈다.
2. 각각에서 역전된 쌍을 센다.

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

Divide: $O(1)$.

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

Conquer: $2T(n / 2)$

5 blue-blue inversions

8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

1. Divide: 둘로 나눈다.
2. 각각에서 역전을 센다.
3. 양쪽에 걸쳐있는 역전을 센다

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

Divide: $O(1)$.

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

5 blue-blue inversions

8 green-green inversions

Conquer: $2T(n / 2)$

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = $5 + 8 + 9 = 22$.

- 각각을 정렬한다.
- 정렬된 두 리스트를 합병하면서 센다.



13 blue-green inversions: $6 + 3 + 2 + 2 + 0 + 0$

Count: $O(n)$



Merge: $O(n)$

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

```
void count_and_sort( int data[], int s, int t )
{
    if ( t-s < 2 )
        return 0;
    else {
        int mid = (s+t)/2;
        int x = count_and_sort( data, s, mid );
        int y = count_and_sort( data, mid+1, t);
        int z = merge_and_count( data, s, mid, t );
        return x+y+z;
    }
}
```

합병정렬과 동일한 시간복잡도 $O(N\log N)$

merge_and_count

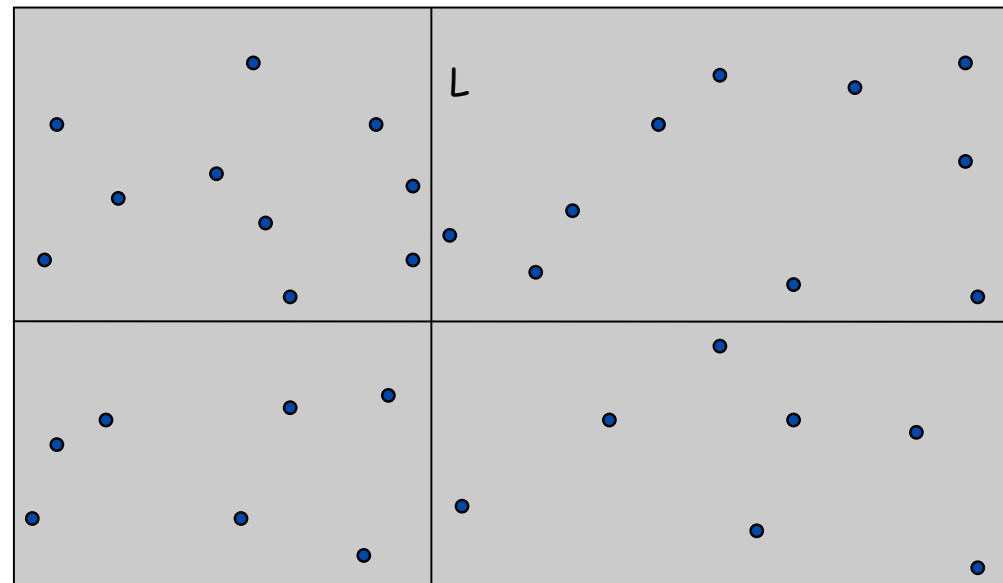
```
void merge_and_count( int data[], int s, int mid, int t )  
{  
    // Left as exercise  
}
```

Closest Pair of Points

Closest Pair of Points

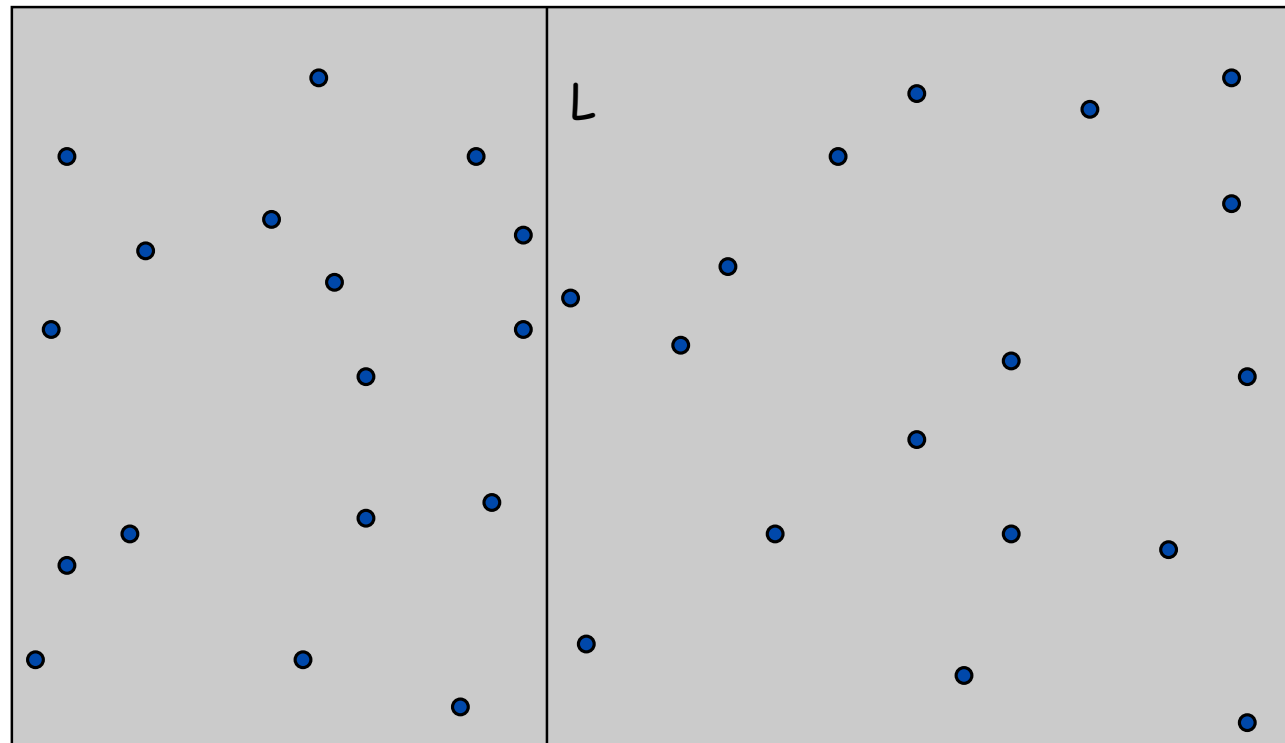
- 평면에 n 개의 점들.
- 거리가 가장 가까운 한 쌍의 점을 찾아라.
- Brute Force: 모든 쌍을 검사. $O(n^2)$
- 1차원 버전: $O(n \log_2 n)$. **How?**
- 가정: 어떤 두 점도 동일한 x 좌표를 갖지 않는다.

- 평면을 4개의 영역으로 분할

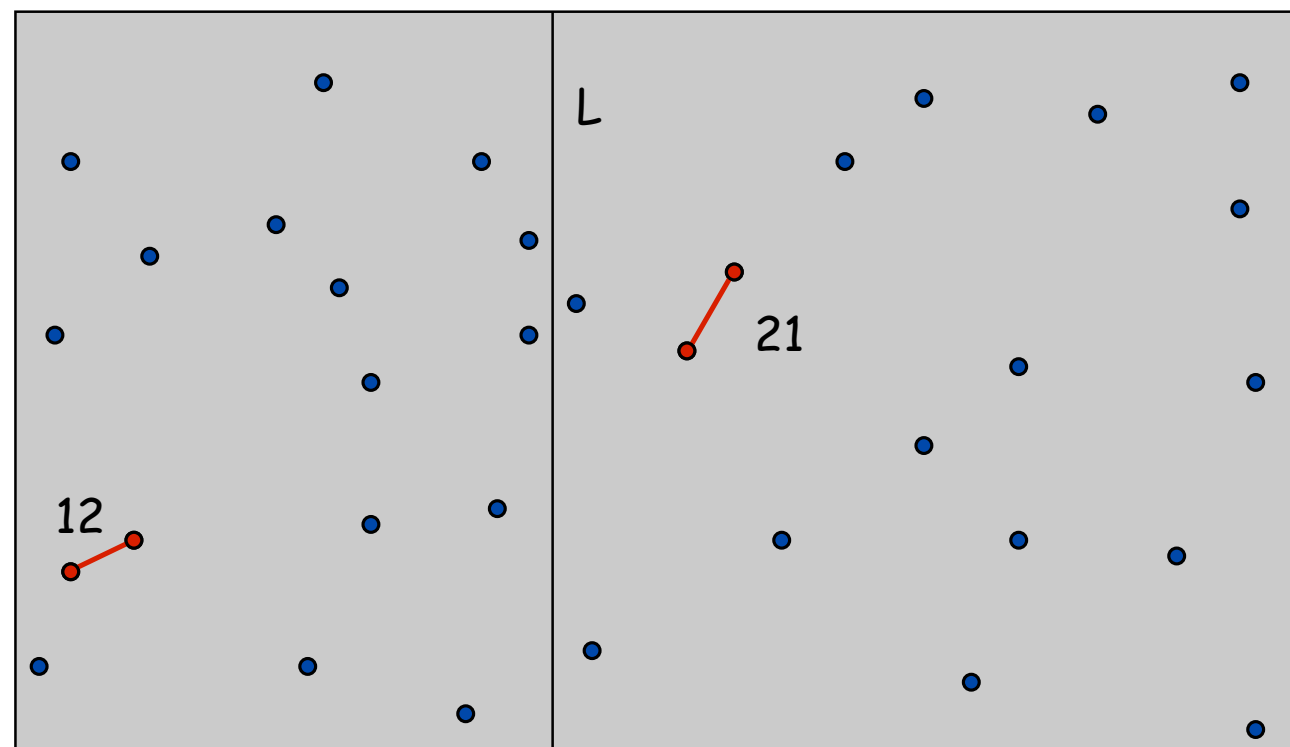


- 각 영역이 동일한 개수의 점들을 가지도록 만들수 없음

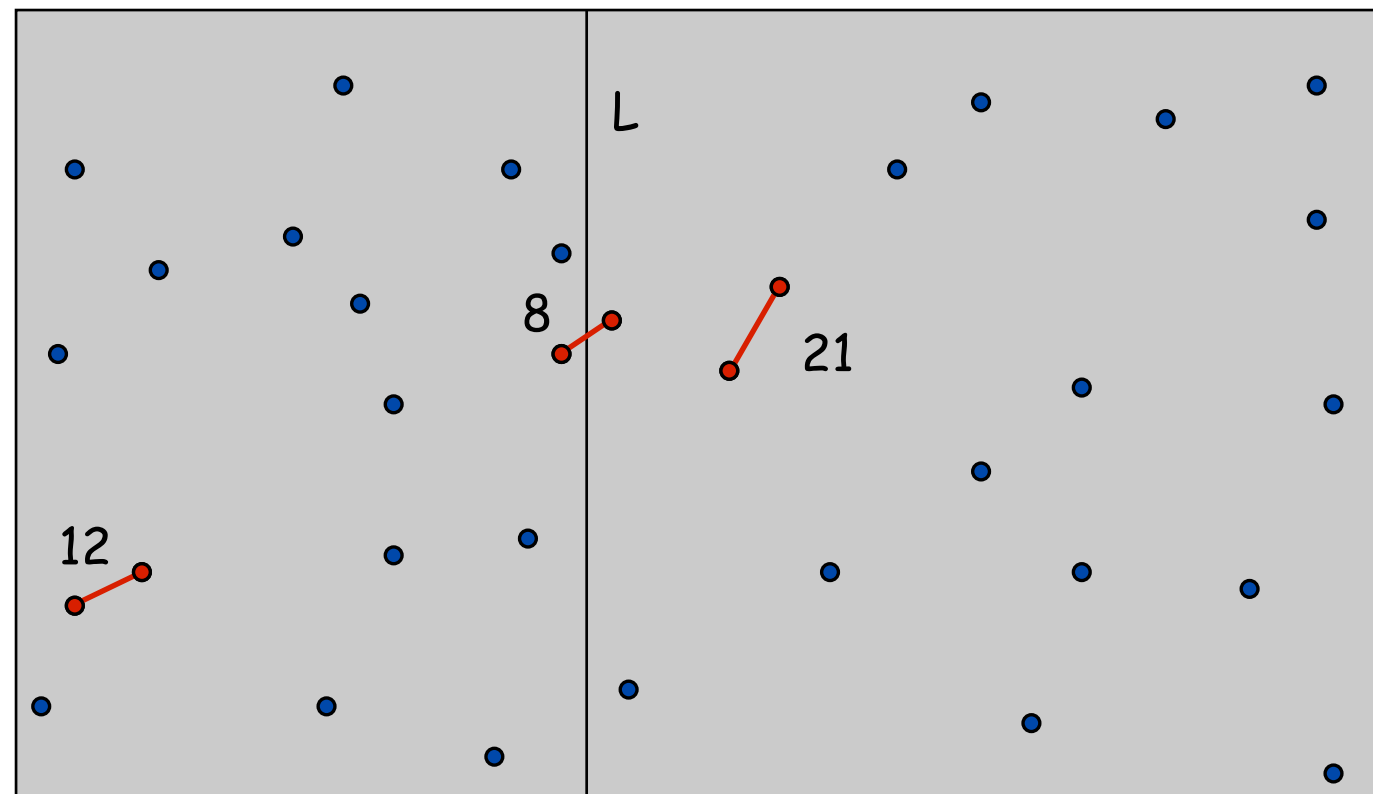
1. 분할: 절반의 점들이 포함되도록 양분



1. 분할: 절반의 점들이 포함되도록 양분
2. 정복: 각 영역에서 가장 가까운 쌍을 찾음



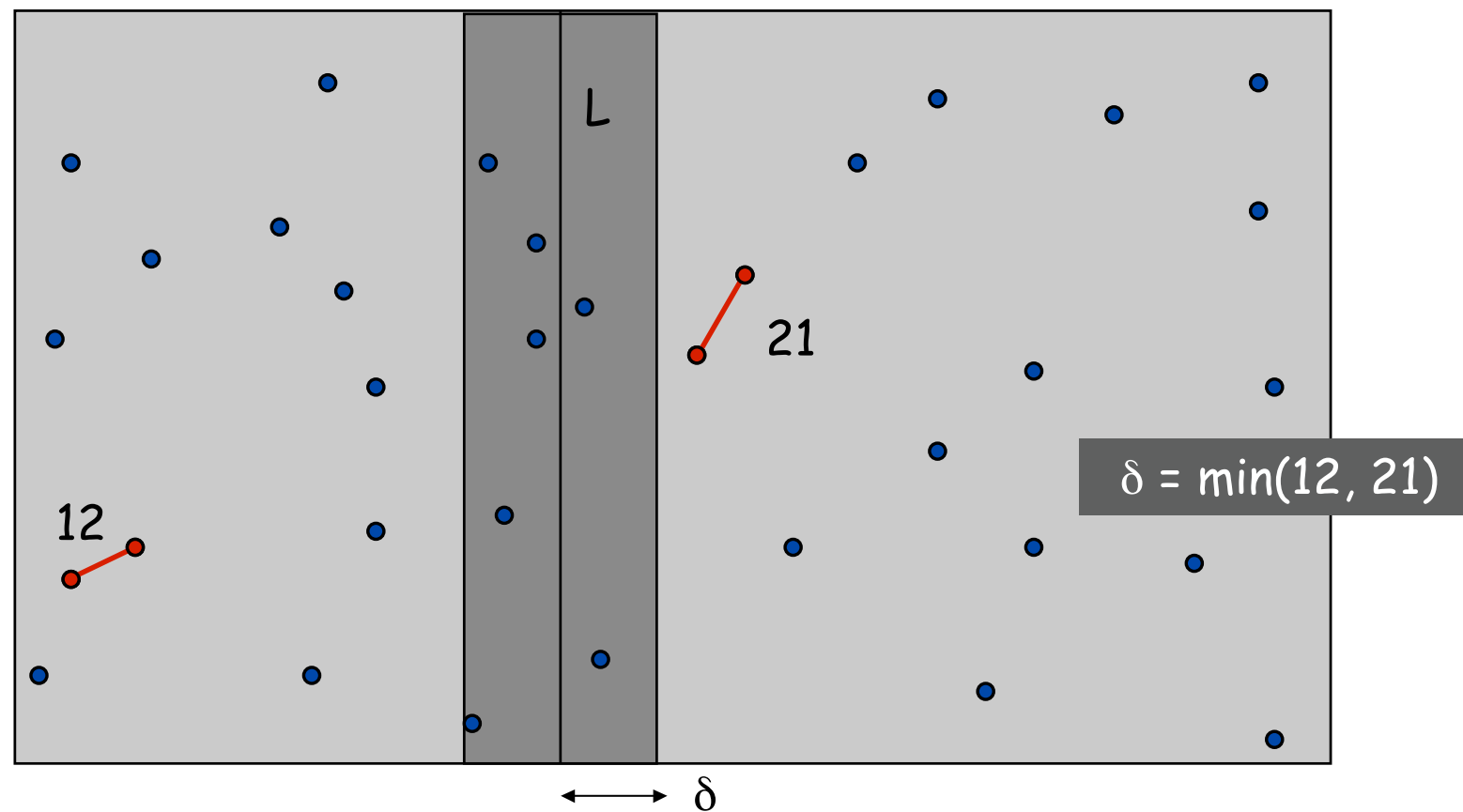
1. 분할: 절반의 점들이 포함되도록 양분
2. 정복: 각 영역에서 가장 가까운 쌍을 찾음
3. 합병: 양쪽을 연결하는 가장 가까운 쌍을 찾은 후 셋 중 최소를 선택



양쪽을 연결하는 가장 가까운 쌍

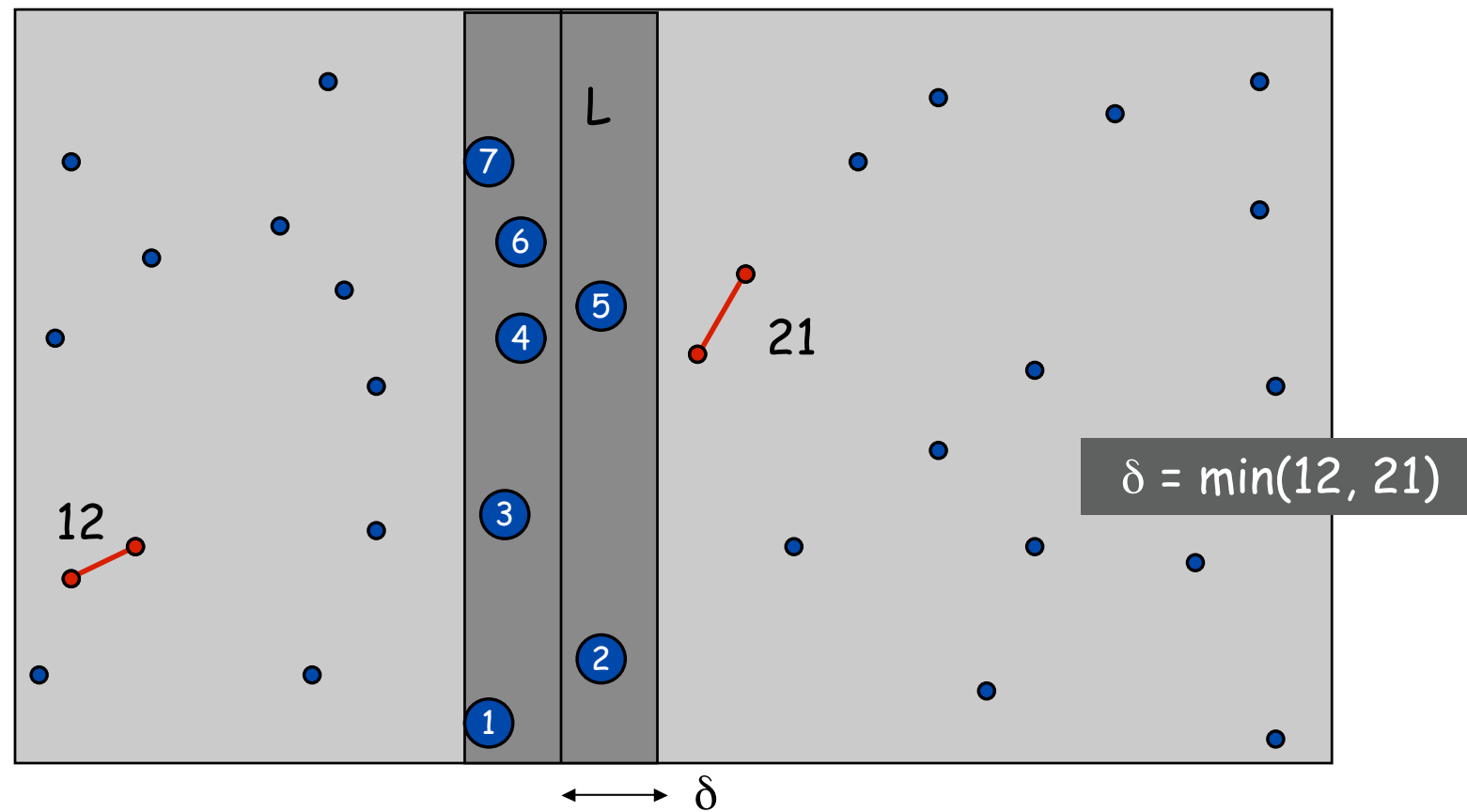
- 양쪽에서 찾은 가장 가까운 쌍들중 거리의 최소값을 δ

- 분할선의 양쪽으로 거리 δ 이내의 점들만 보면 됨
- 서로간의 거리가 δ 이내인 점들만 보면 됨



양쪽을 연결하는 가장 가까운 쌍

- δ -strip 내의 점들을 y좌표를 기준으로 정렬
- 동일 영역에 속한 점들끼리는 최소한 δ 만큼 떨어져 있음

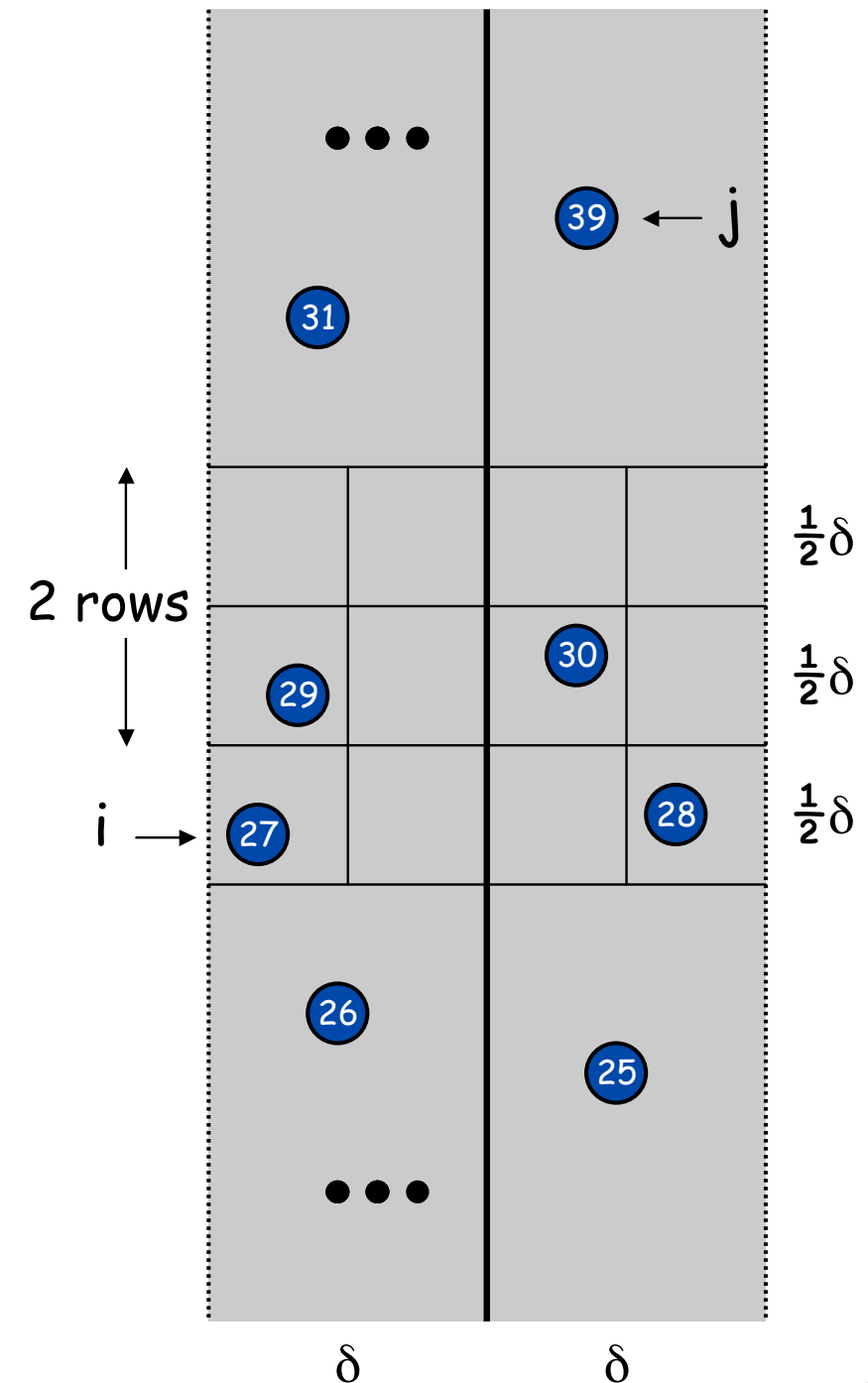


양쪽을 연결하는 가장 가까운 쌍

주장: 만약 $|i-j| \geq 12$ 이면, 두 점간의 거리는 δ 이상이다.

증명: $\delta/2 * \delta/2$ 사각형 내에 두 점이 있을 수 없다.

즉, y 좌표로 정렬된 순서상 서로 11칸 이내에 있는 점들간의 거리만 계산해서 비교하면 된다.



```
Closest-Pair( $p_1, \dots, p_n$ ) {  
  Compute separation line  $L$  such that half the points  
    are on one side and half on the other side.  $O(n \log n)$   
  
   $\delta_1 = \text{Closest-Pair}(\text{left half})$   
   $\delta_2 = \text{Closest-Pair}(\text{right half})$   $2T(n / 2)$   
   $\delta = \min(\delta_1, \delta_2)$   
  
  Delete all points further than  $\delta$  from separation line  $L$   $O(n)$   
  
  Sort remaining points by  $y$ -coordinate.  $O(n \log n)$   
  
  Scan points in  $y$ -order and compare distance between  
    each point and next 11 neighbors. If any of these  
    distances is less than  $\delta$ , update  $\delta$ .  $O(n)$   
  
  return  $\delta$ .  
}
```

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(n \log n) = O(n \log^2 n)$$

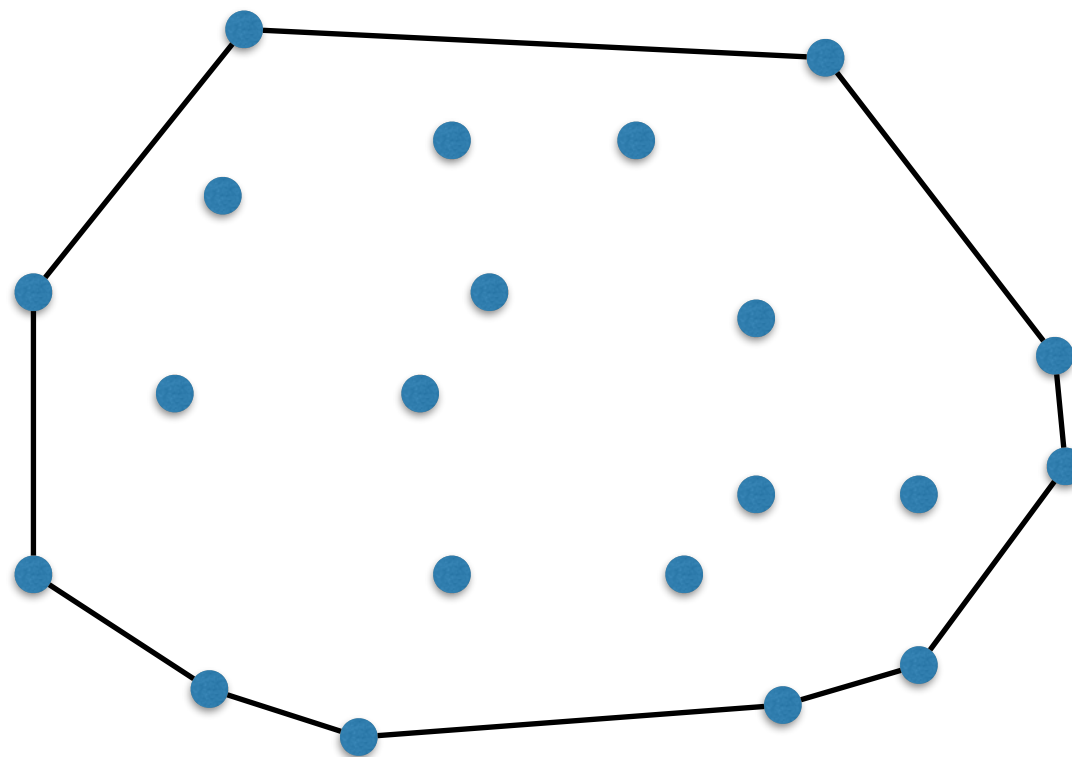
Q. $O(n \log_2 n)$ 으로 가능?

A. Yes, but 조금 더 복잡.

Convex Hull

Convex Hull

- 평면상에 n 개의 점이 주어짐
- Convex Hull: 모든 점을 포함하는 가장 작은 볼록 다각형
- 출력: 다각형을 이루는 꼭지점들을 시계방향 순으로 출력

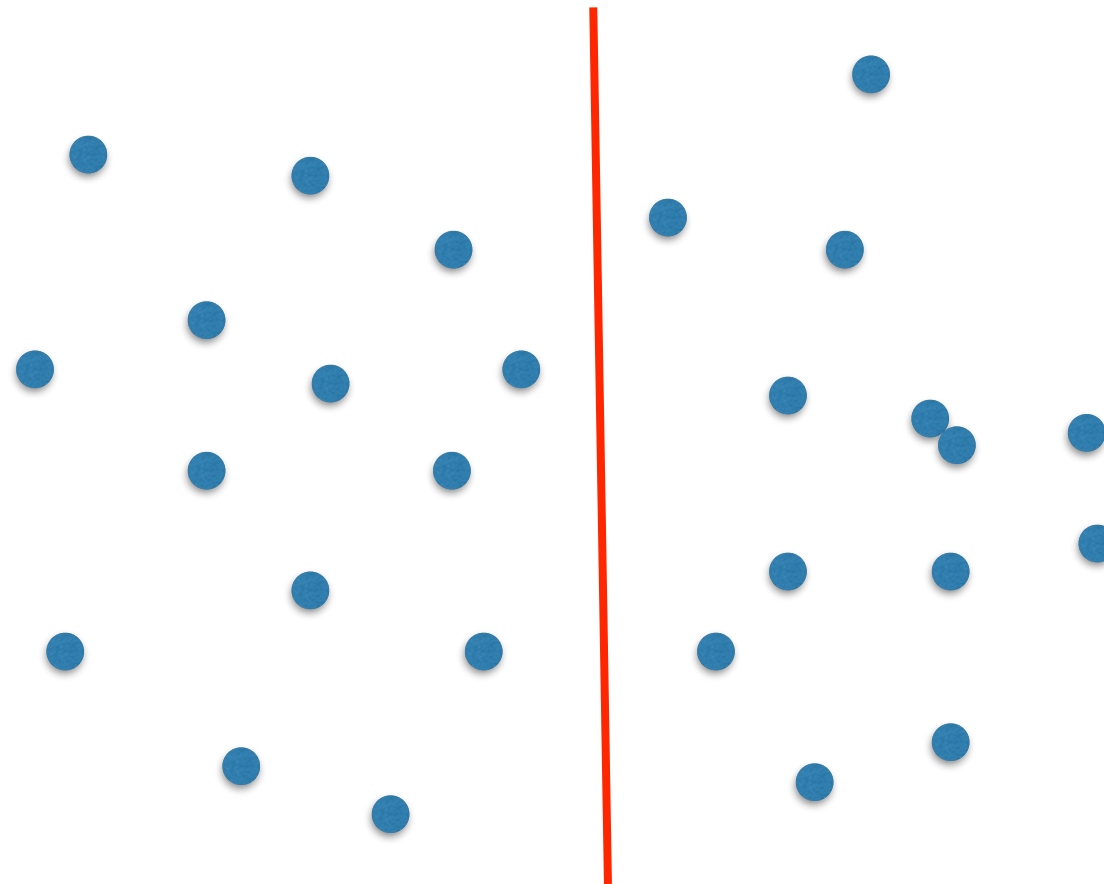


Convex Hull Algorithms

- Jarvis march — $O(nh)$
 - h : the number of points in the hull. Worst case $\Theta(n^2)$.
- Graham scan — $O(n \log n)$
- Quickhull
 - average case $O(n \log n)$, but $O(n^2)$ in the worst case.
- **Divide and conquer — $O(n \log n)$**
- The ultimate planar convex hull algorithm — $O(n \log h)$
 - The first optimal output-sensitive algorithm
- Chan's algorithm — $O(n \log h)$

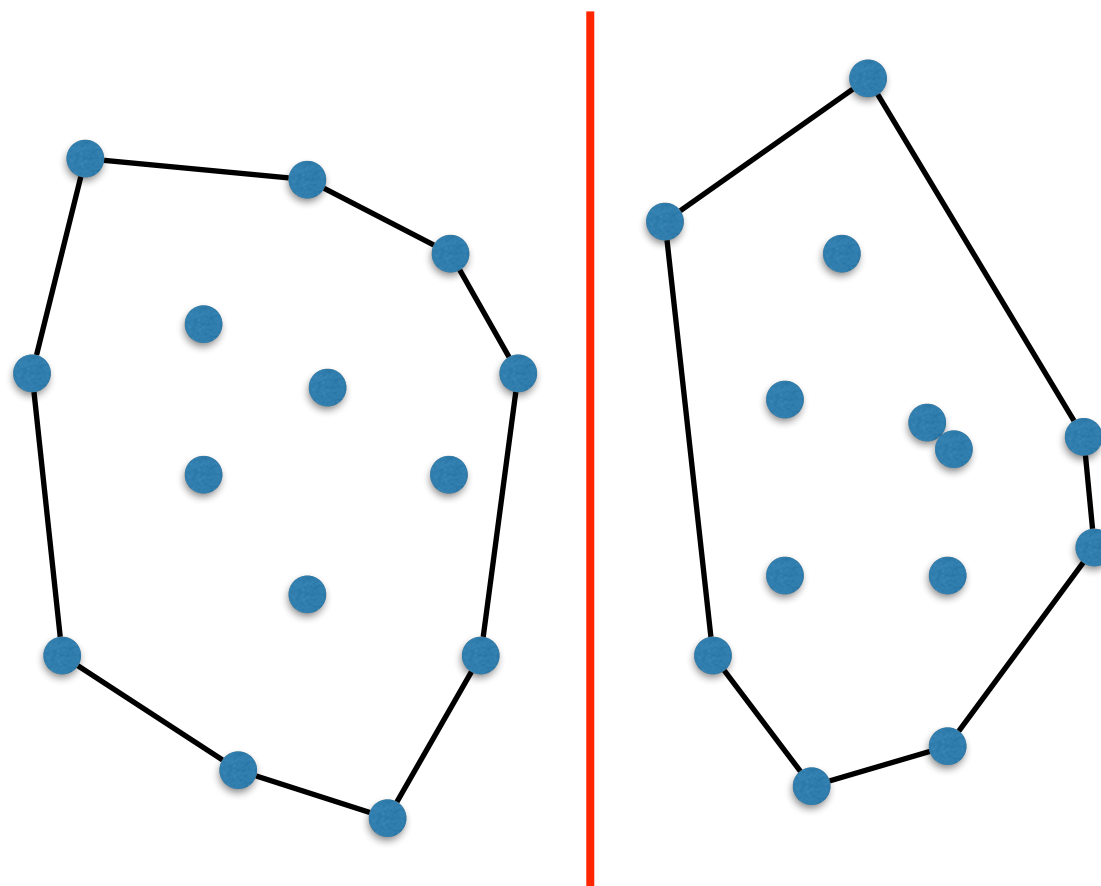
1. Divide

- x좌표순으로 정렬한 후 절반으로 분할



2. Conquer

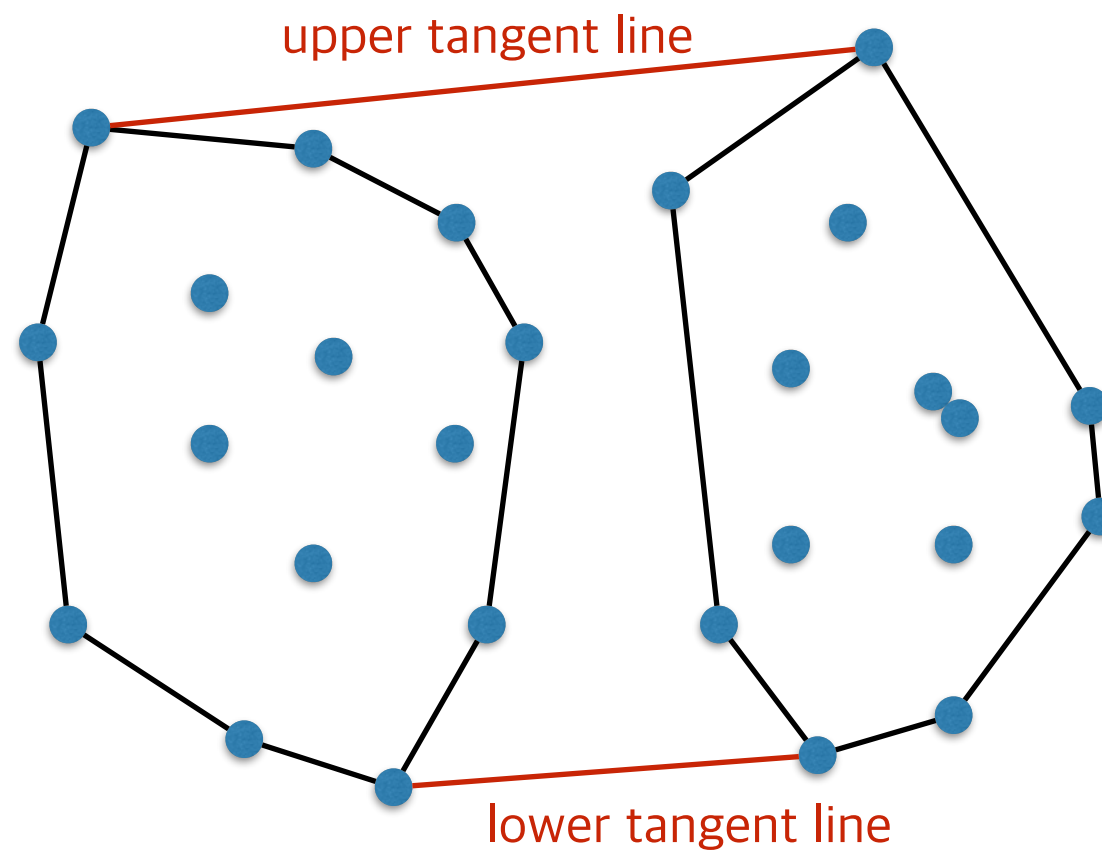
- 각각에 대해서 convex hull을 recursion으로 구함



양쪽 convex hull에 속하는 꼭지점들의
시계 방향 리스트를 가지고 있다.

3. Merge

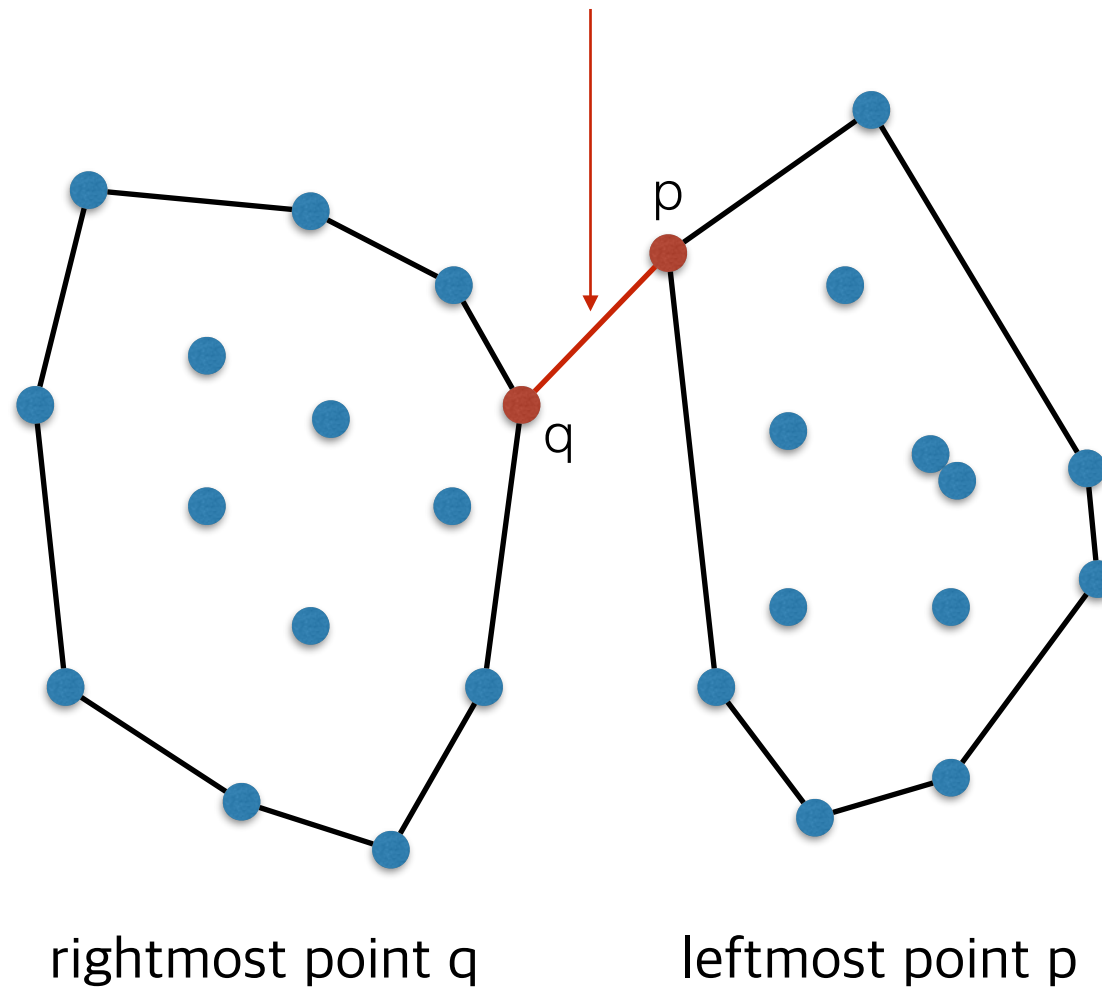
두 convex hull의 upper tangent line과 lower tangent line을 찾는다.



두 tangent line의 양 끝 꼭지점을 알면 전체 convex hull의 꼭지점을 시계방향으로 출력할 수 있다.

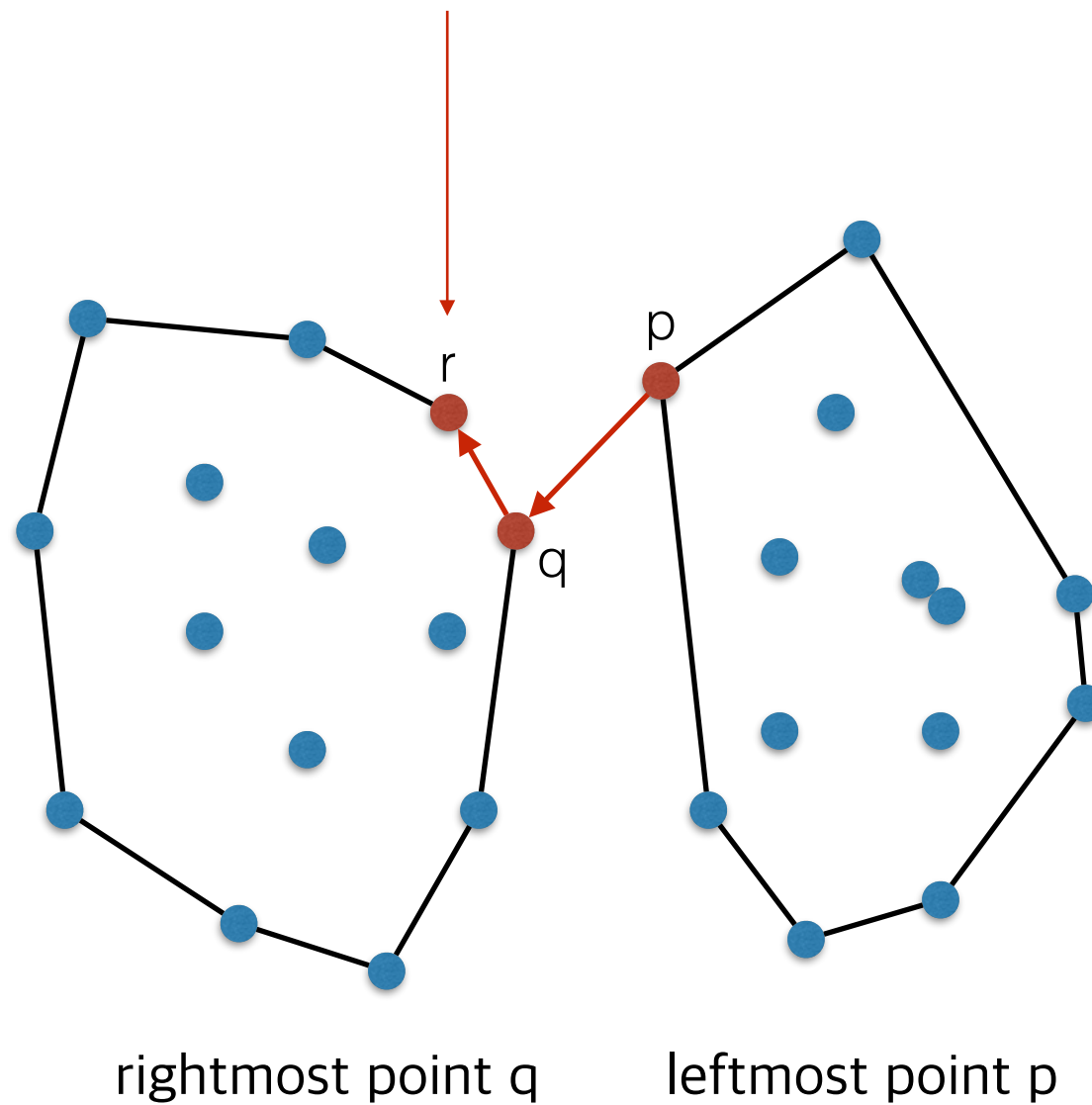
3. Merge

왼쪽 convex hull의 가장 오른쪽 꼭지점 q와
오른쪽 convex hull의 가장 왼쪽 꼭지점 p를 연결하는 선분을
생각해본다.



3. Merge

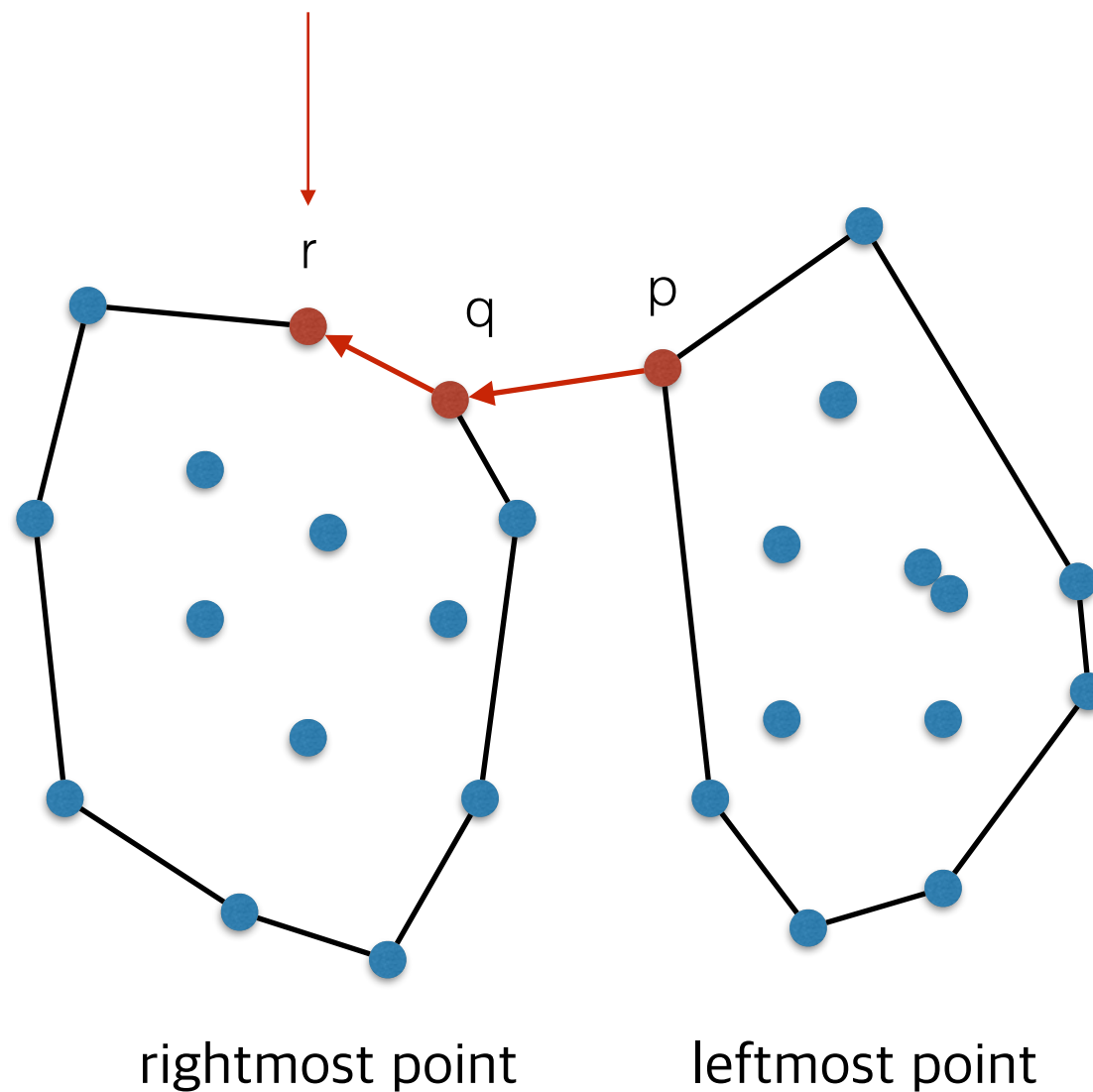
r은 왼쪽 convex hull에서 반시계방향으로 q 다음 꼭지점이다.



$p \rightarrow q \rightarrow r$ 이 좌회전인지 우회전인지 판단한다. 이 예에서는 **우회전**이다. 만약 우회전이면 꼭지점 q를 포기하고 r이 새로운 q가 된다.

3. Merge

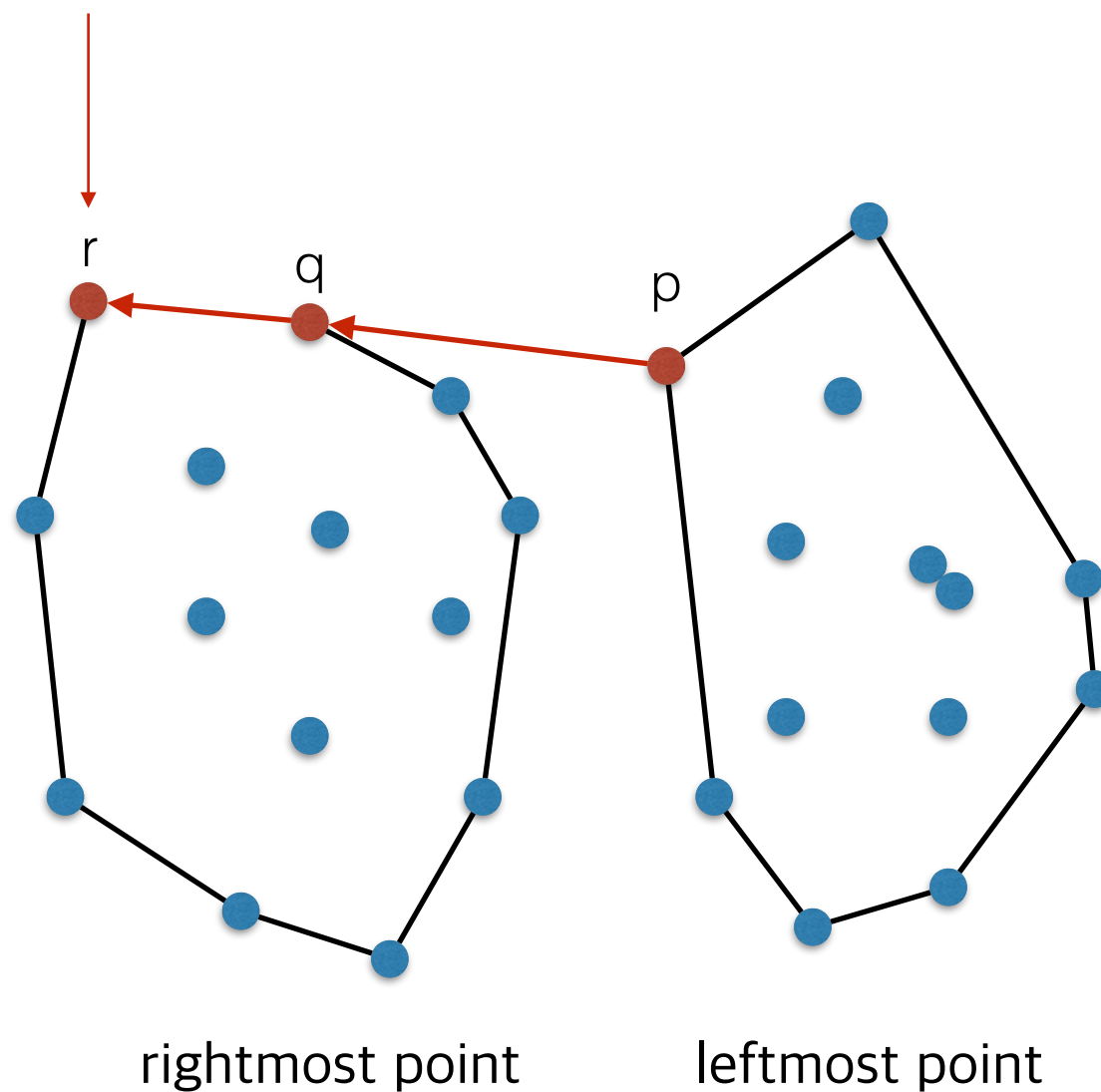
왼쪽 convex hull에서 r은 항상 반시계방향으로 q 다음 꼭지점이다.



다시 $p \rightarrow q \rightarrow r$ 이 좌회전인지 우회전인지 판단한다. 이 예에서는 여전히 **우회전**이다. 만약 우회전이면 꼭지점 q 를 포기하고 r 이 새로운 q 가 된다.

3. Merge

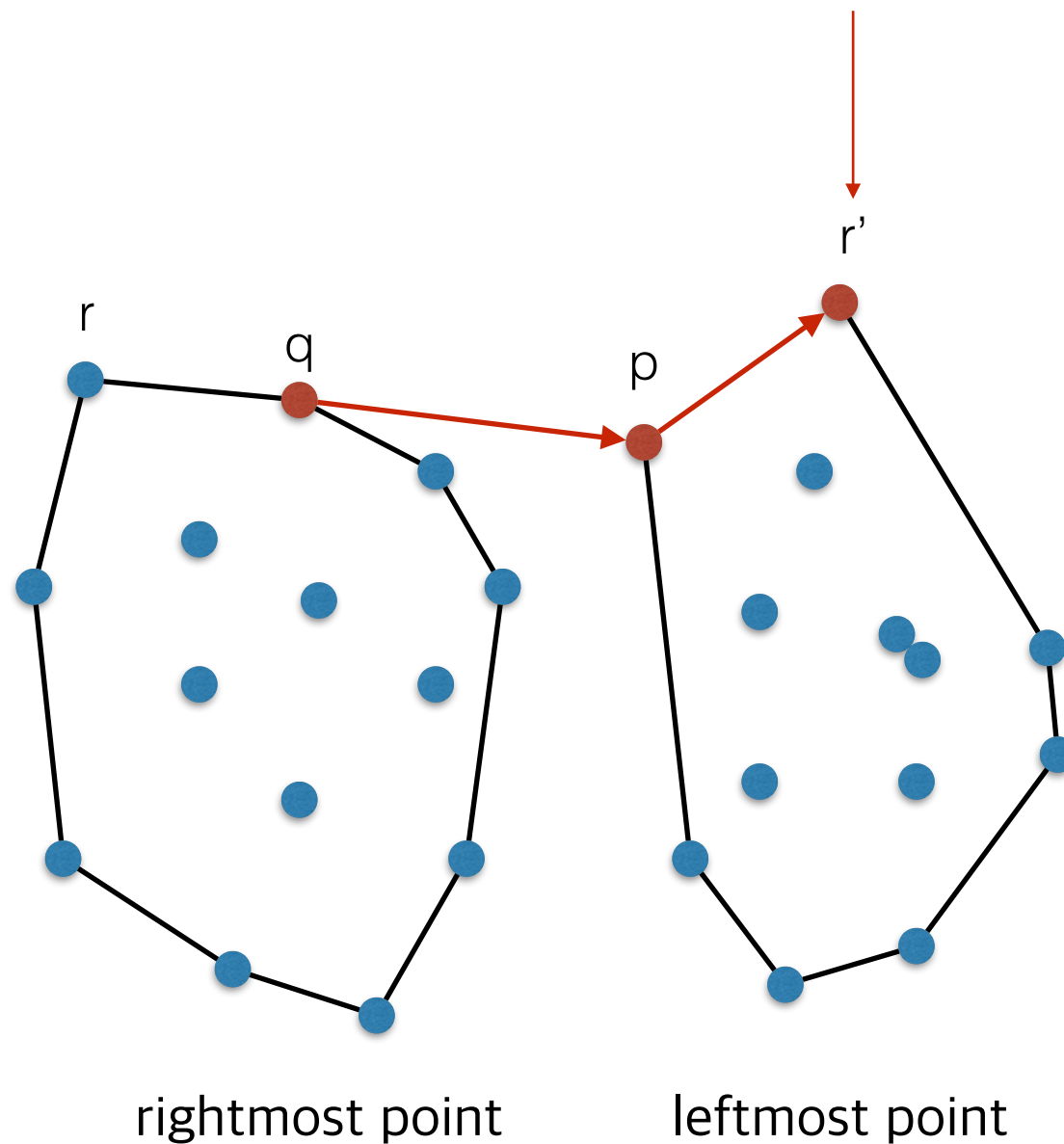
왼쪽 convex hull에서 r은 항상 반시계방향으로 q 다음 꼭지점이다.



다시 $p \rightarrow q \rightarrow r$ 이 좌회전인지 우회전인지 판단한다. 이 예에서는 미세하지만 **좌회전**이다. 그러면 OK.

3. Merge

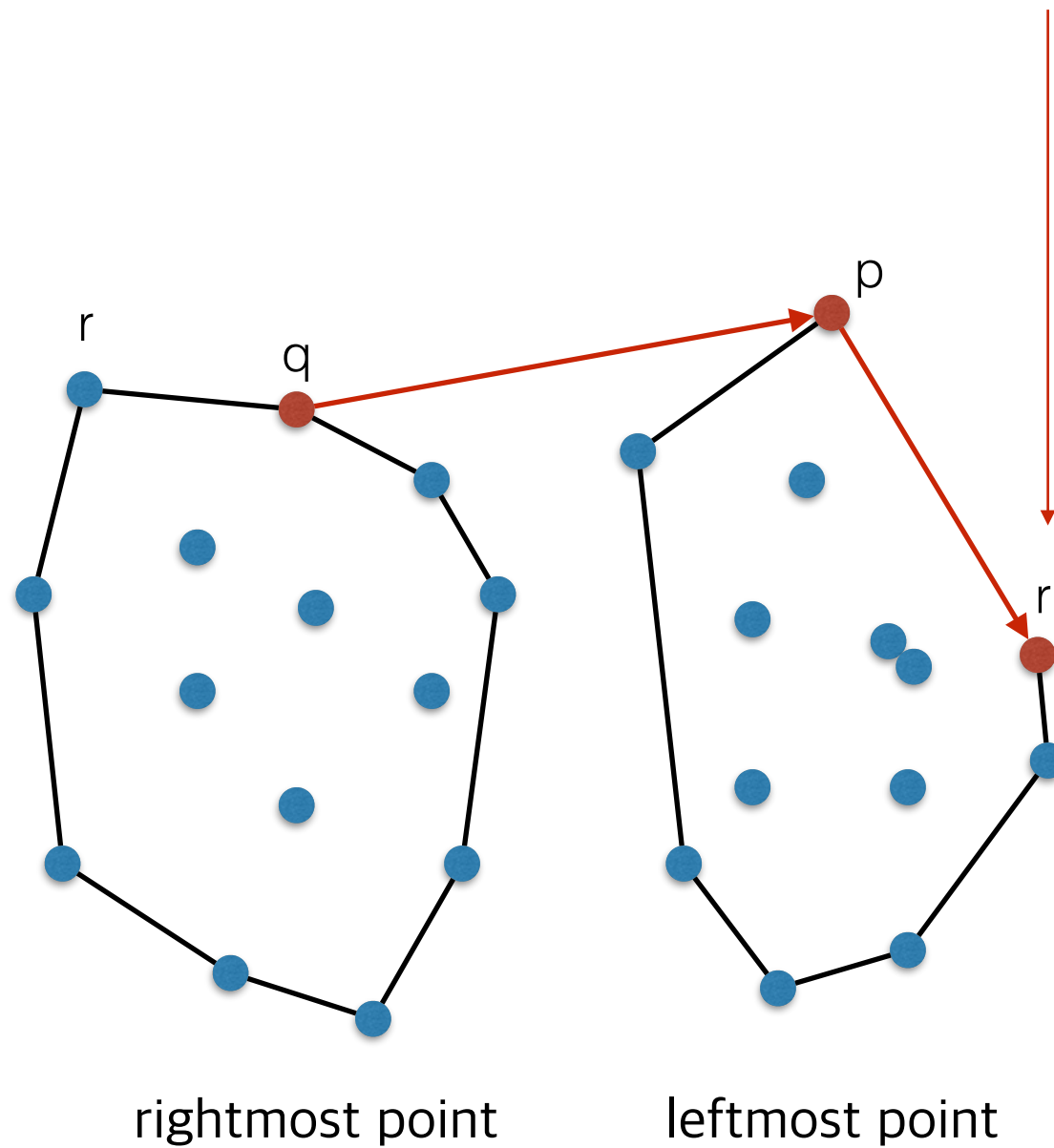
오른쪽 convex hull에서 r은 항상 시계방향으로 p 다음 꼭지점이다.



$q \rightarrow p \rightarrow r'$ 이 좌회전인지 우회전인지 판단한다. 이 예에서는 **좌회전**이다. 만약 좌회전이면 꼭지점 p 를 포기하고 r' 가 새로운 p 가 된다.

3. Merge

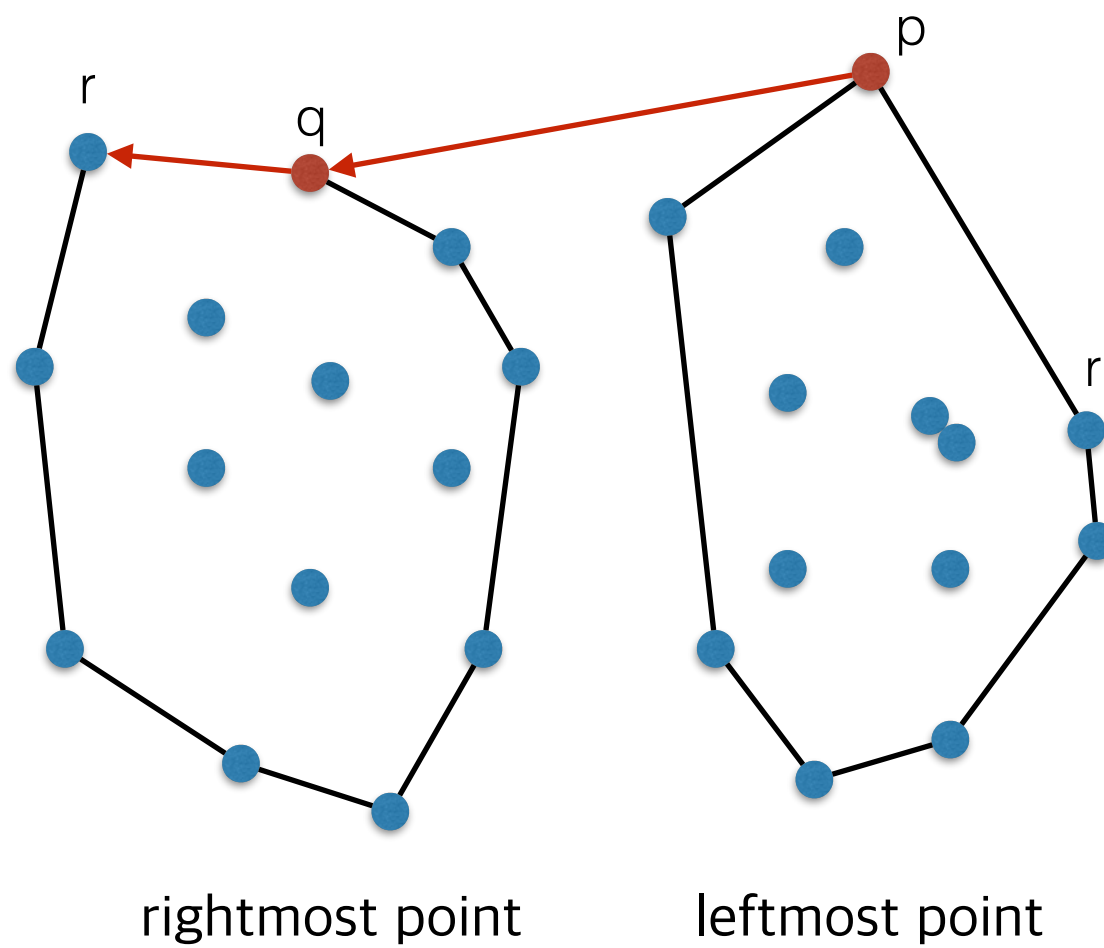
오른쪽 convex hull에서 r' 는 항상 시계방향으로 p 다음 꼭지점이다.



$q \rightarrow p \rightarrow r'$ 가 좌회전인지 우회전인지 판단한다. 이 예에서는 **우회전**이다. 그러면 OK.

Divide and Conquer

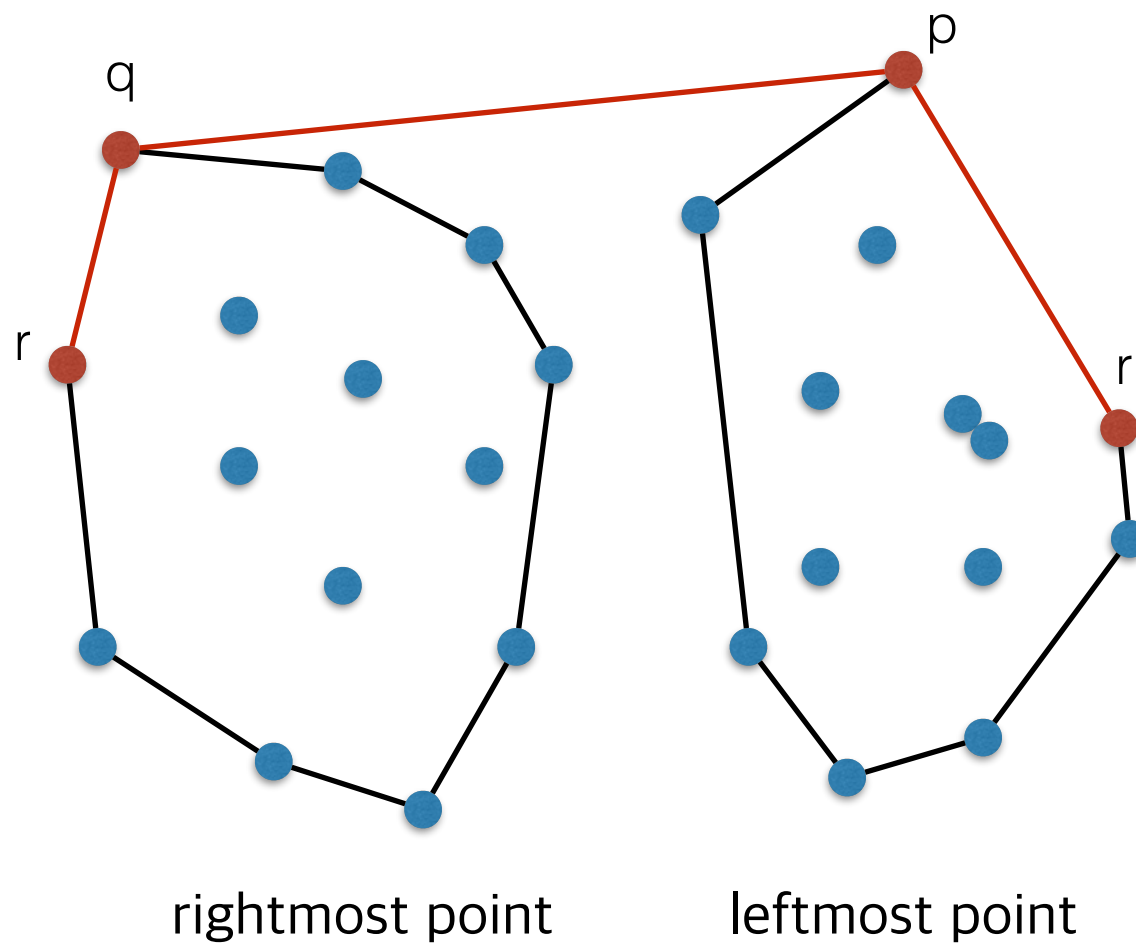
3. Merge



하지만 p 가 바뀌는 바람에 $p \rightarrow q \rightarrow r$ 이
다시 우회전이 되어버렸다.
다시 이전 과정을 반복한다.

Divide and Conquer

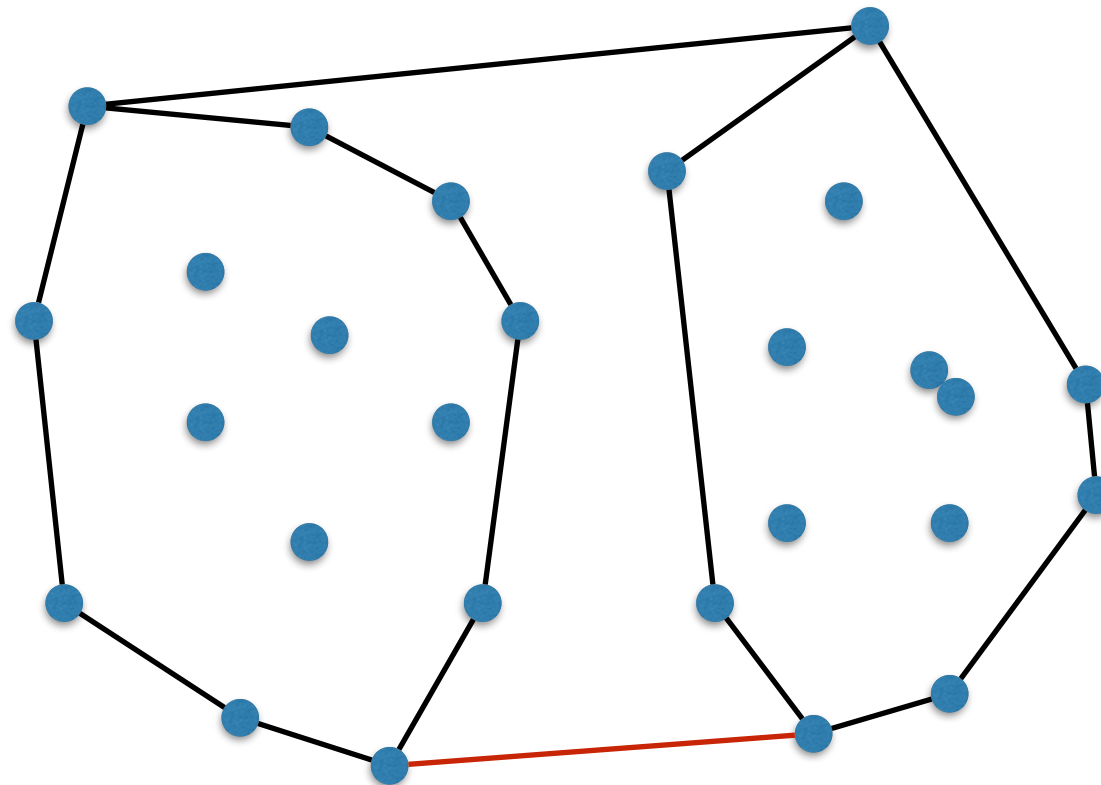
3. Merge



이런 과정을 좌우를 번갈아 가면서
 $p \rightarrow q \rightarrow r$ 은 좌회전,
 $q \rightarrow p \rightarrow r'$ 은 우회전이 될때 까지
반복한다.

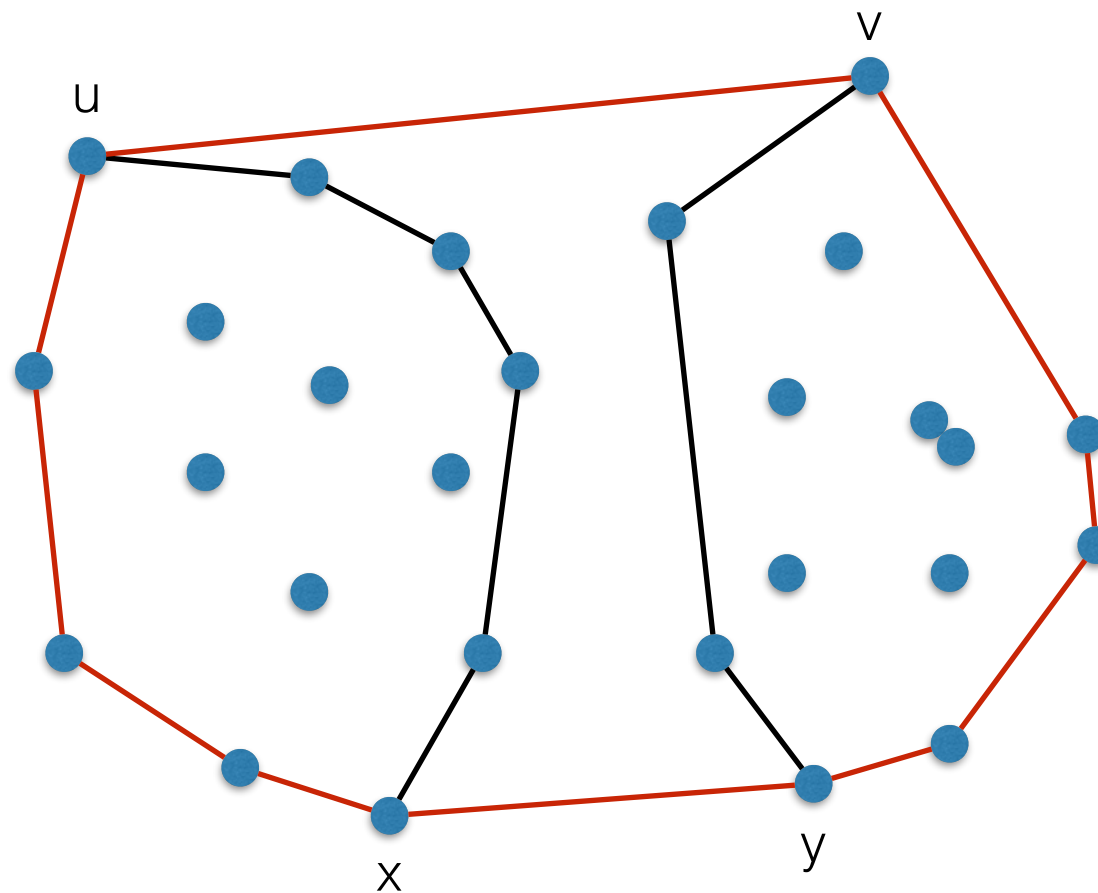
이렇게 해서 찾아진 q 와 p 를 연결
하는 선분이 두 convex hull의
upper tangent line이 된다.

3. Merge




Lower tangent line을 찾는 과정은
upper tangent line을 찾는 방법에서
좌·우, 그리고 시계방향·반시계방향만
반대로 바꾸면 된다.

3. Merge

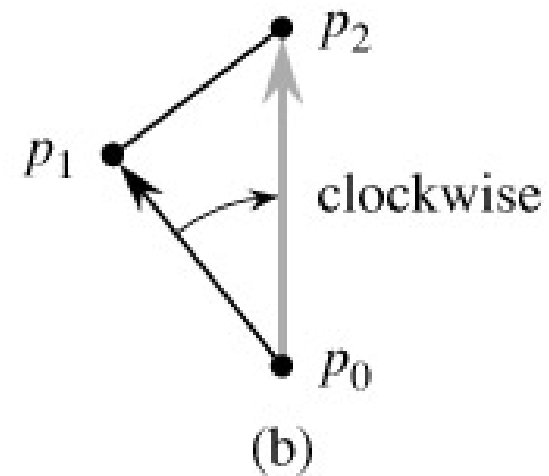
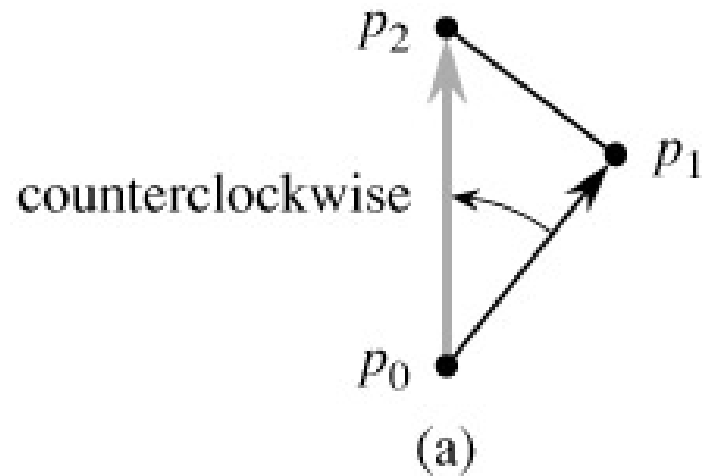


Convex hull을 구성하는 꼭지점은 왼쪽 convex hull에서 시계방향으로 x 에서 u 까지, 그리고 오른쪽 convex hull에서 시계방향으로 v 에서 y 까지이다.

2개의 tangent line을 찾아서 합병하는 일들이
 $O(n)$ 시간이 되도록 구현하면 된다.


$$T(n) = 2T\left(\frac{n}{2}\right) + n = O(n \log n)$$

좌회전? 우회전?



($p_2 - p_0$)가 ($p_1 - p_0$)로부터 시계방향인지 반시계방향인지 검사, 즉,

$$(p_1 - p_0) \times (p_2 - p_1) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

$$= \begin{cases} > 0 & \text{left turn} \\ < 0 & \text{right turn} \\ 0 & \text{colinear} \end{cases}$$