

#고급반 6주차

기하

<Geometry>

4p / <CCW>

26p / <선분 교차>

55p / <볼록 껍질>

101p / <연습 문제>

#시작하기 전

<Geometry> - Introduction?

기하? 그거 누가 하나? ㄹㅇㅋㅋ



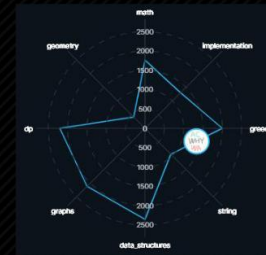
코포 레드



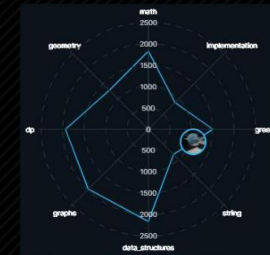
전 학술부장



전 학술부원 군인



코포 오렌지



현 학술부장



현 회장

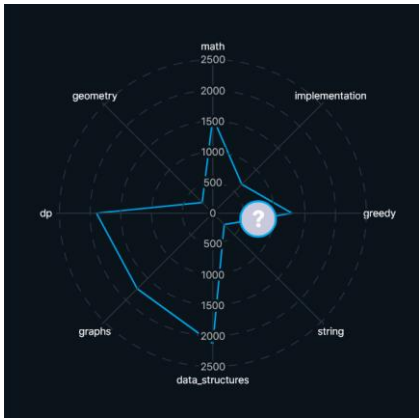
다들 북서쪽 방향에 신경 좀 써주세요 ㄹㅇㅇ

발표자 (전 학술부원)
심치어 플딱

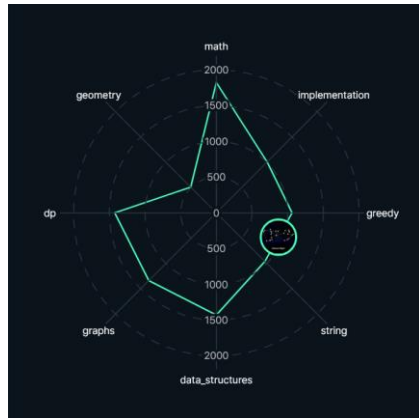
2022 ALOHA 알고리즘 멘토링

#시작하기 전

<Geometry> - Introduction?



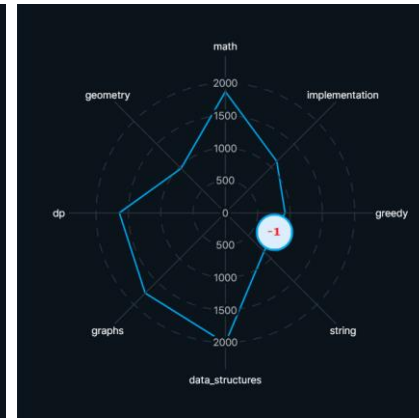
학술부원 1



학술부원 2



학술부원 3



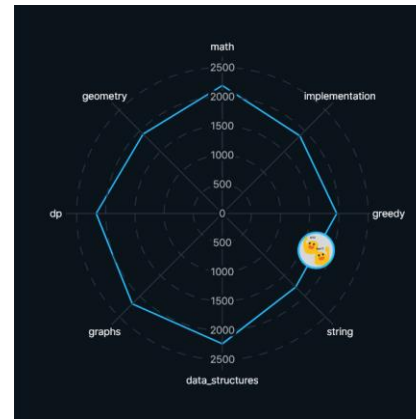
회장



익명의 오렌지



차기 학술부장



강의자 (학술부장)



#CCW

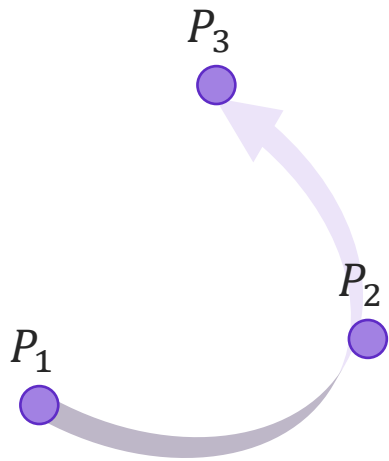
<Counter Clockwise>

3개의 점이 주어질 때, 이 3개의 점이 어떤 방식으로 나열되어 있는지 판별

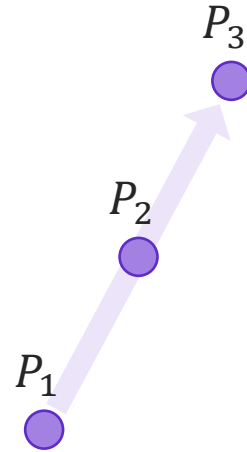
1. CCW는 Counter ClockWise의 약어입니다.
2. 약어에서 알 수 있듯이, 시계방향/반시계방향/일직선으로 있는지 판별해줍니다.

#CCW

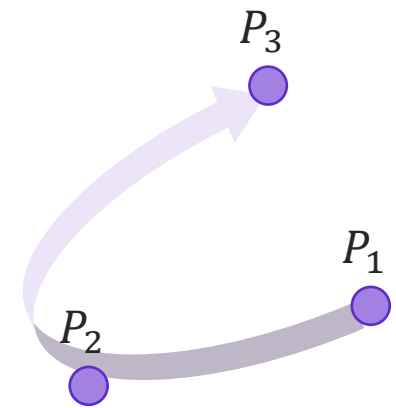
<Counter Clockwise>



반시계방향



일직선



시계방향



#CCW

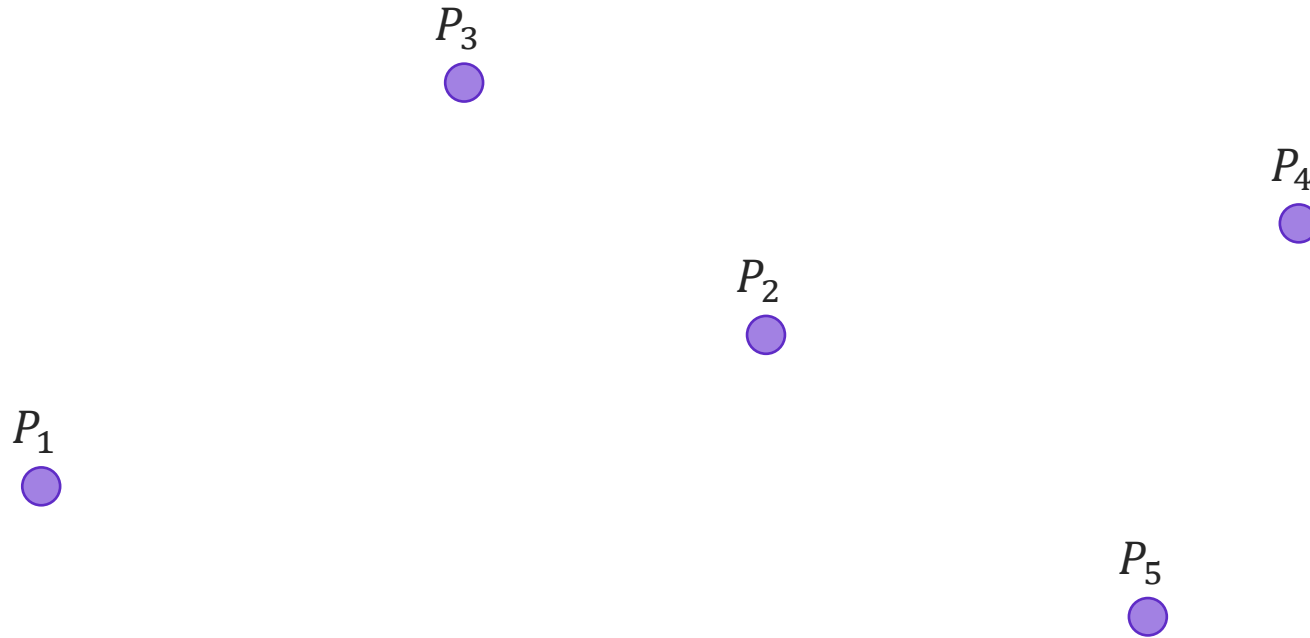
<Counter Clockwise>

직접 구해보자. 그런데 어떻게 구할까요?

이미 아시는 분들도 계시겠지만, 아무런 설명 없이 결과만 있으면 재미없잖아요?

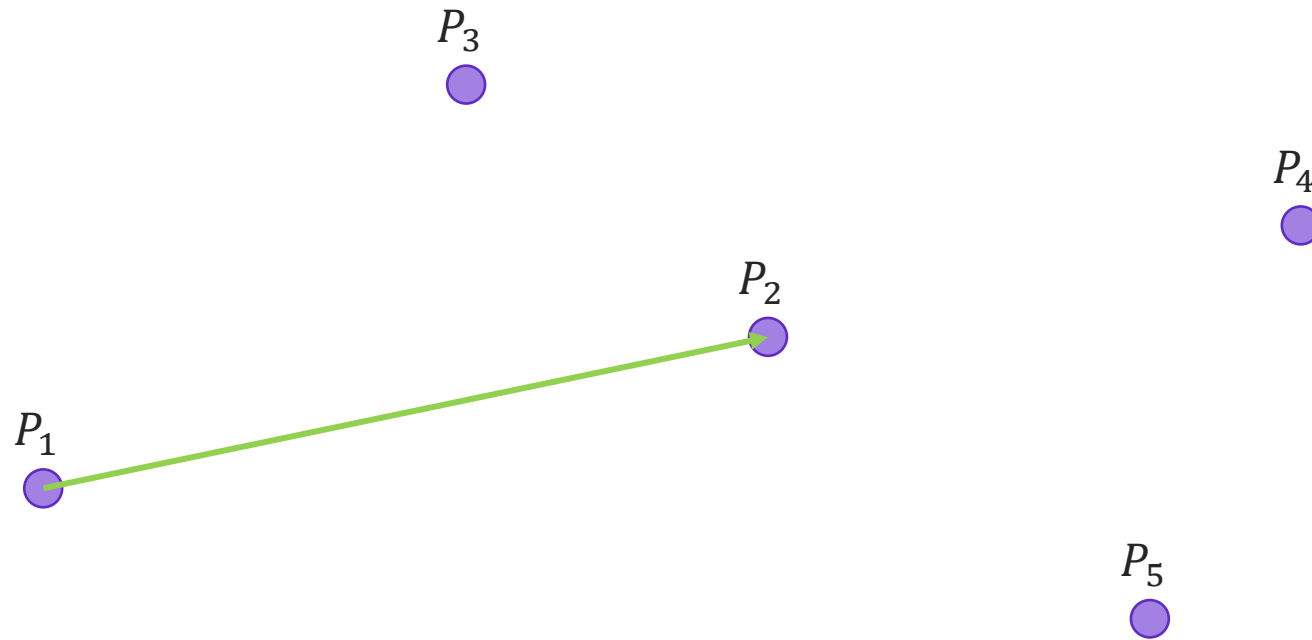
#CCW

<Counter Clockwise>



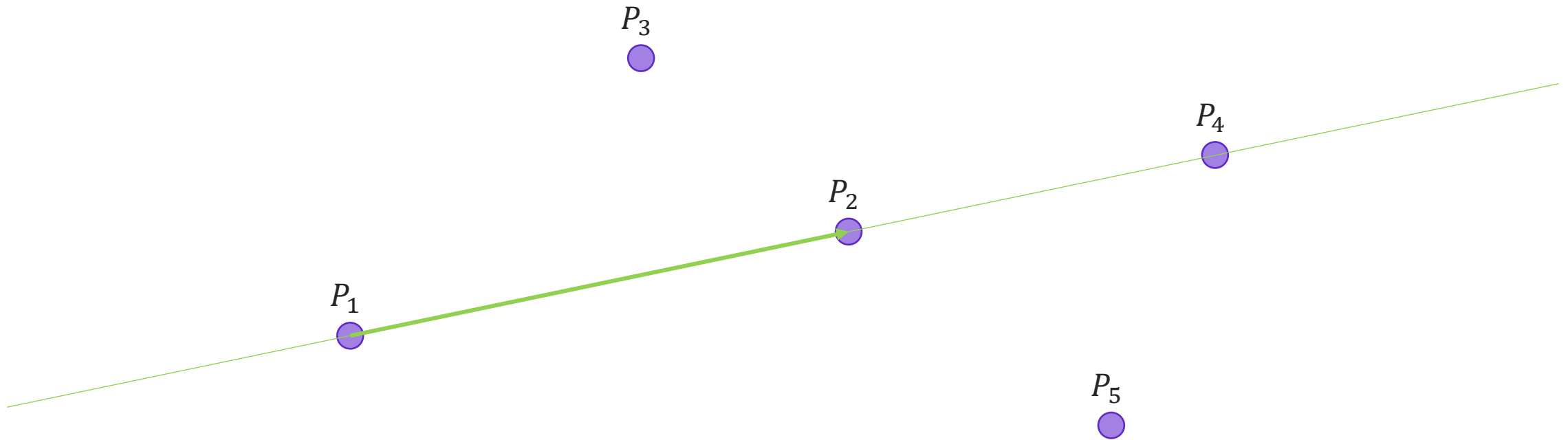
#CCW

<Counter Clockwise>



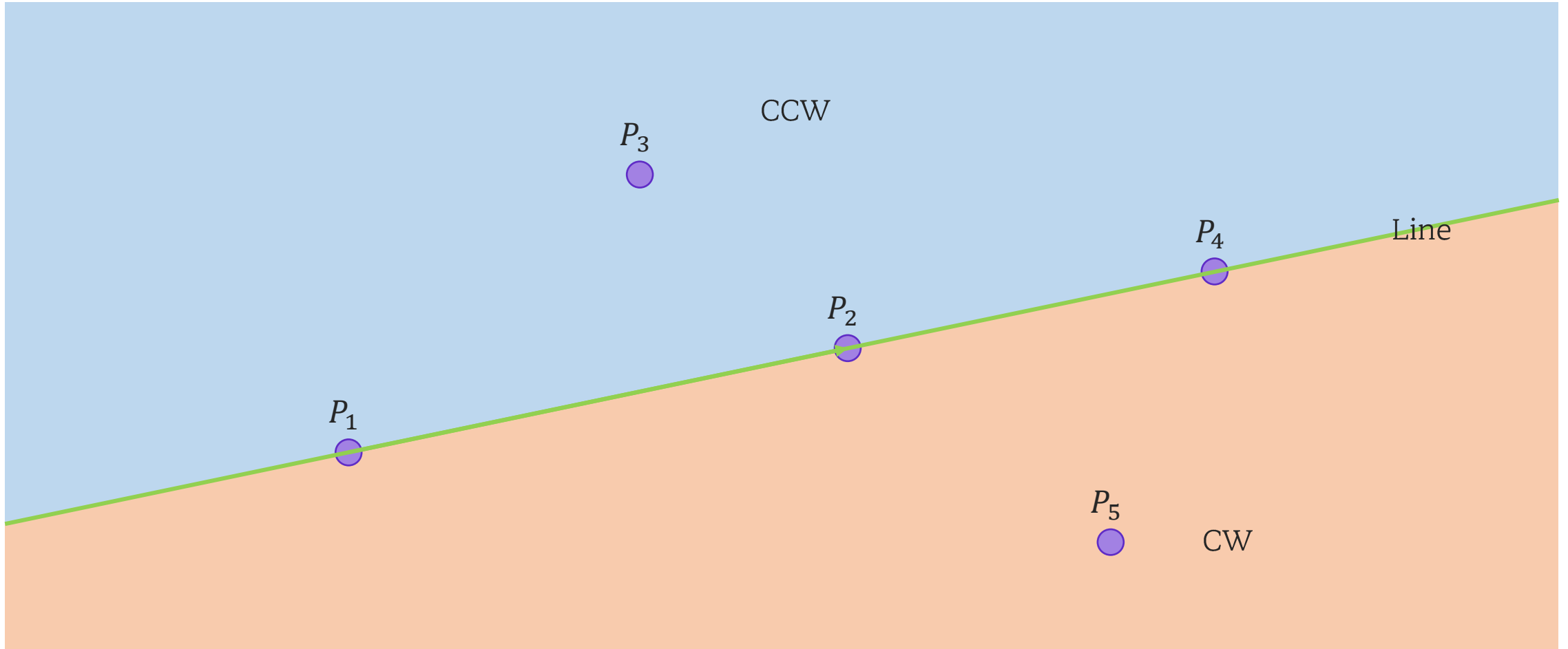
#CCW

<Counter Clockwise>



#CCW

<Counter Clockwise>





#CCW

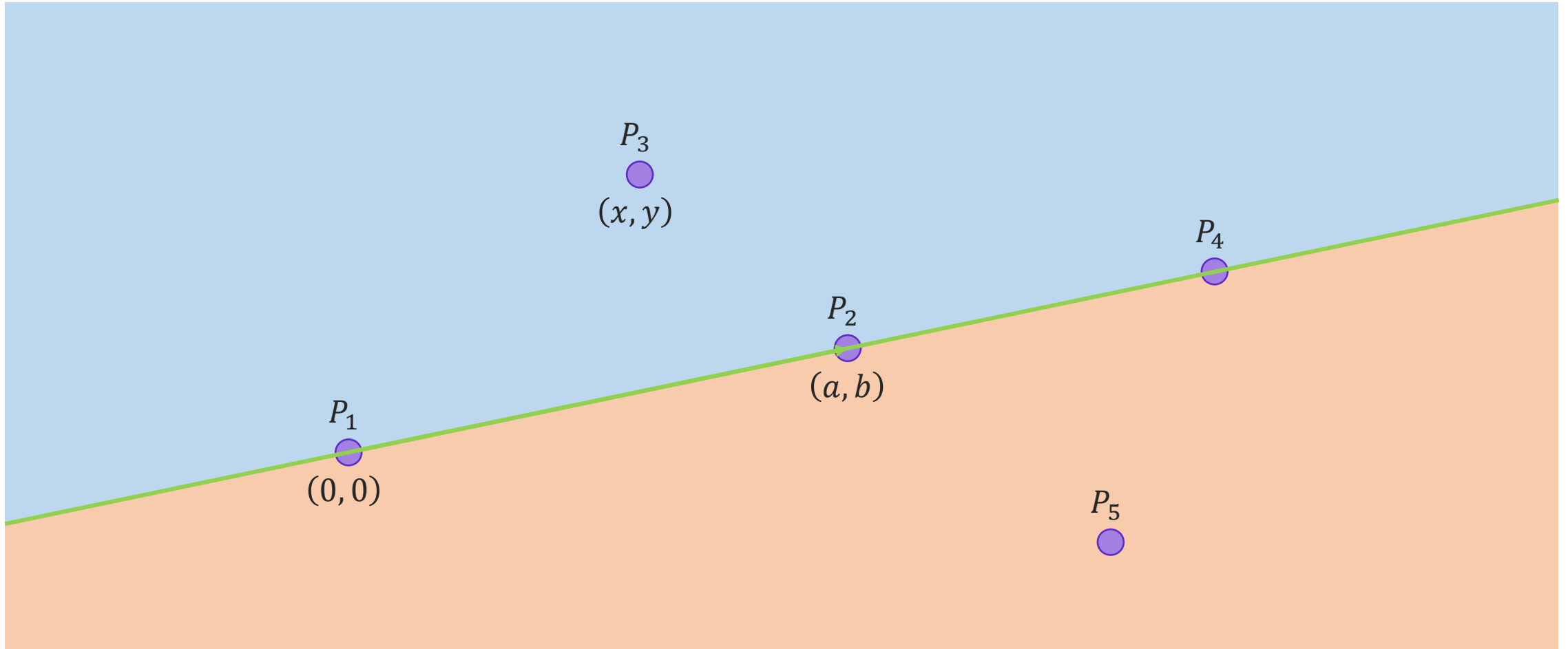
<Counter Clockwise>

직선을 기준으로 위/아래로 나뉘었으니, 이론상 위 그림을 잘 돌려서 문제를 풀 수 있습니다.

여기서부터 몇 페이지 동안은 엄밀한 증명을 위해 있는 슬라이드로,
실제 문제 풀이를 할 때는 그냥 넘어가도 됩니다.

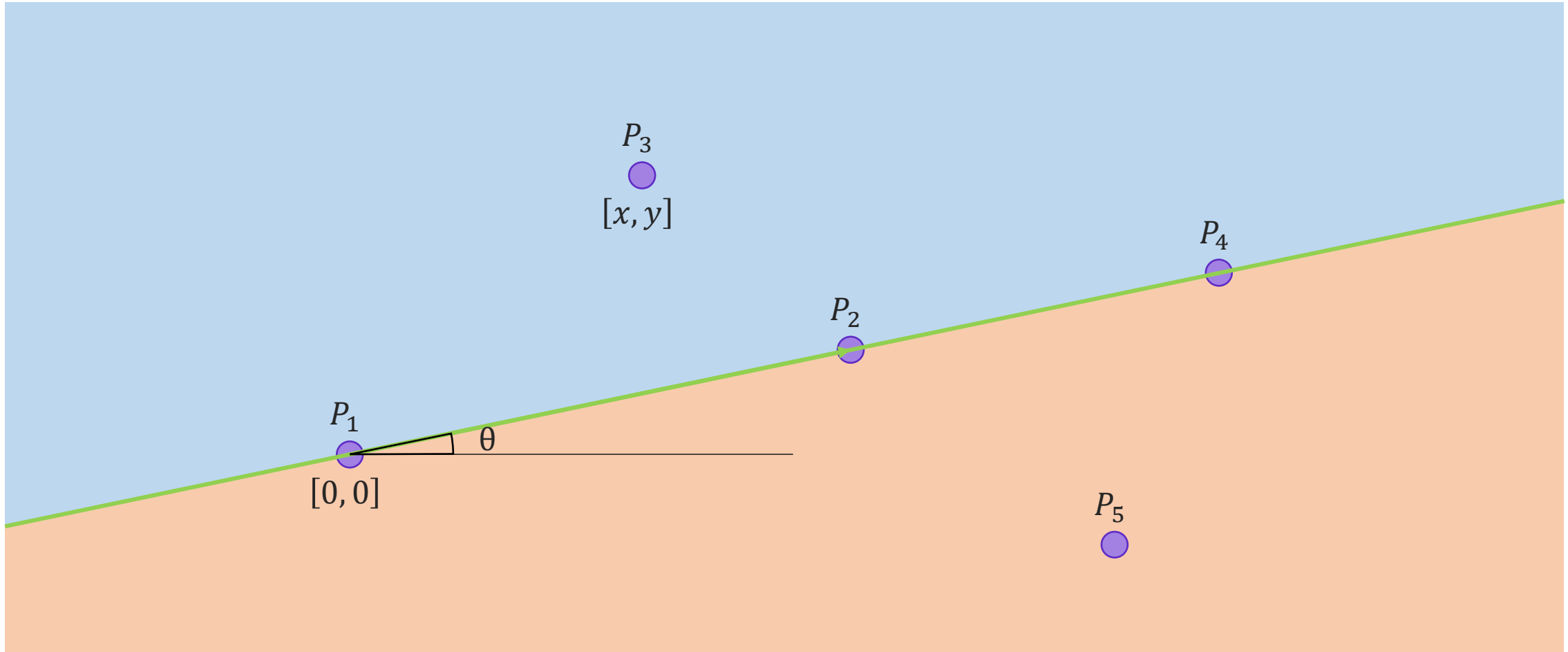
#CCW

<Counter Clockwise>



#CCW

<Counter Clockwise>

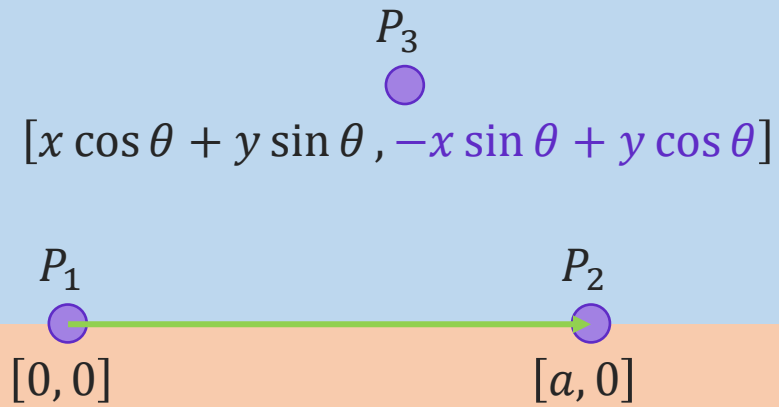


#CCW

<Counter Clockwise>

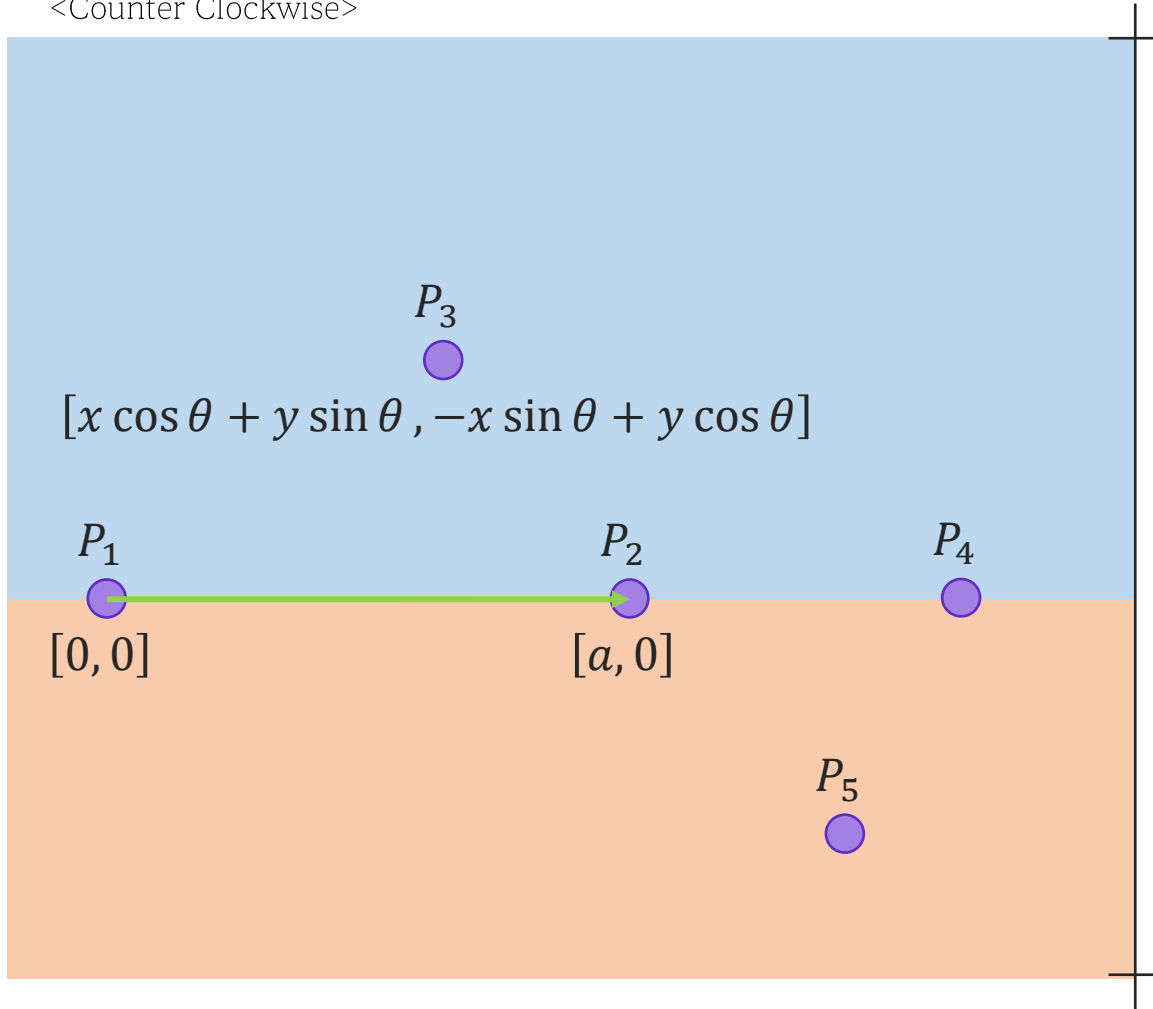
Rotation Matrix (with angle $-\theta$)

$$\begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$



#CCW

<Counter Clockwise>



$$[x \cos \theta + y \sin \theta, -x \sin \theta + y \cos \theta] \times [a, 0]$$

$$= \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x \cos \theta + y \sin \theta & -x \sin \theta + y \cos \theta & 0 \\ a & 0 & 0 \end{bmatrix}$$

$$= -ax \sin \theta + ay \cos \theta$$



#CCW

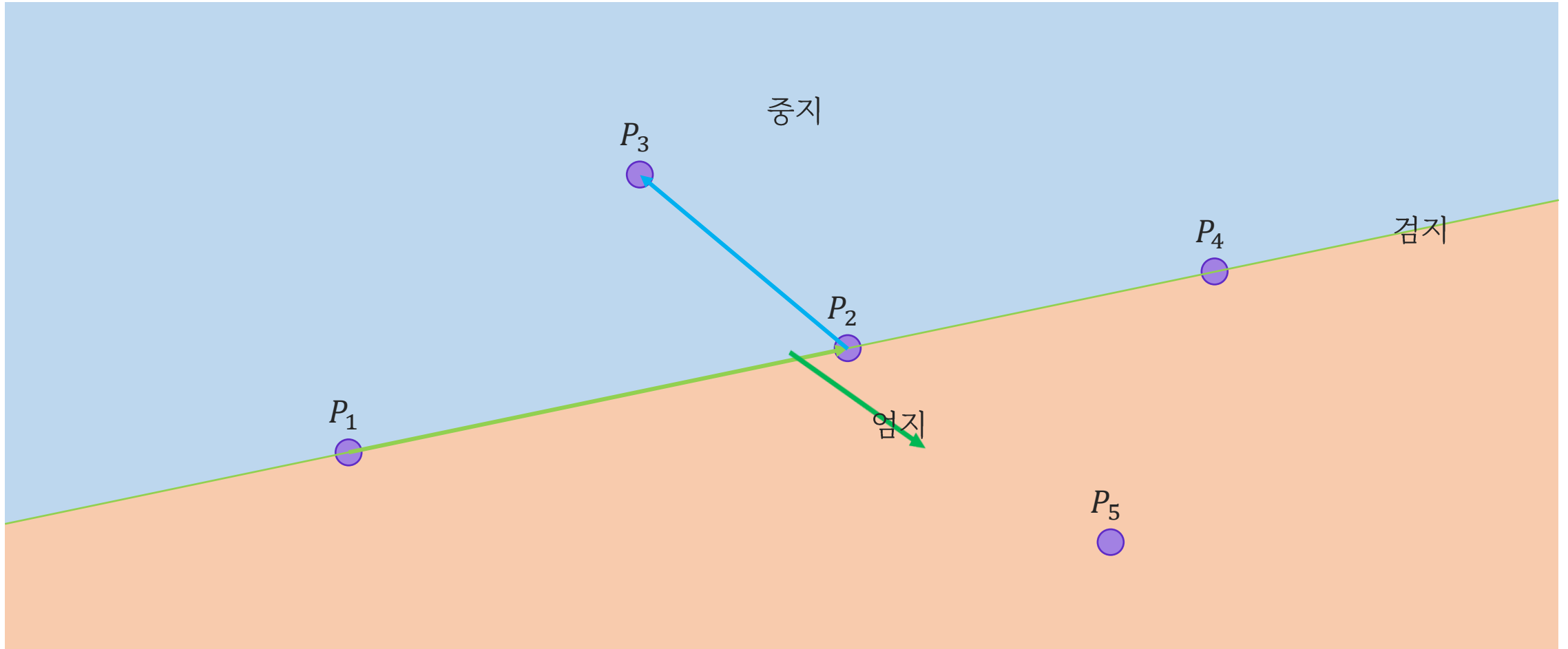
<Counter Clockwise>

결국 우리가 구하는 것을 **외적**으로 할 수 있음을 알아차렸습니다!
물론 위 과정을 매번 생각하는 건 귀찮으니, 좀 더 구현하기 편하게 고쳐봅시다.

이거 증명하는데 오래 걸렸어요 살려줘요

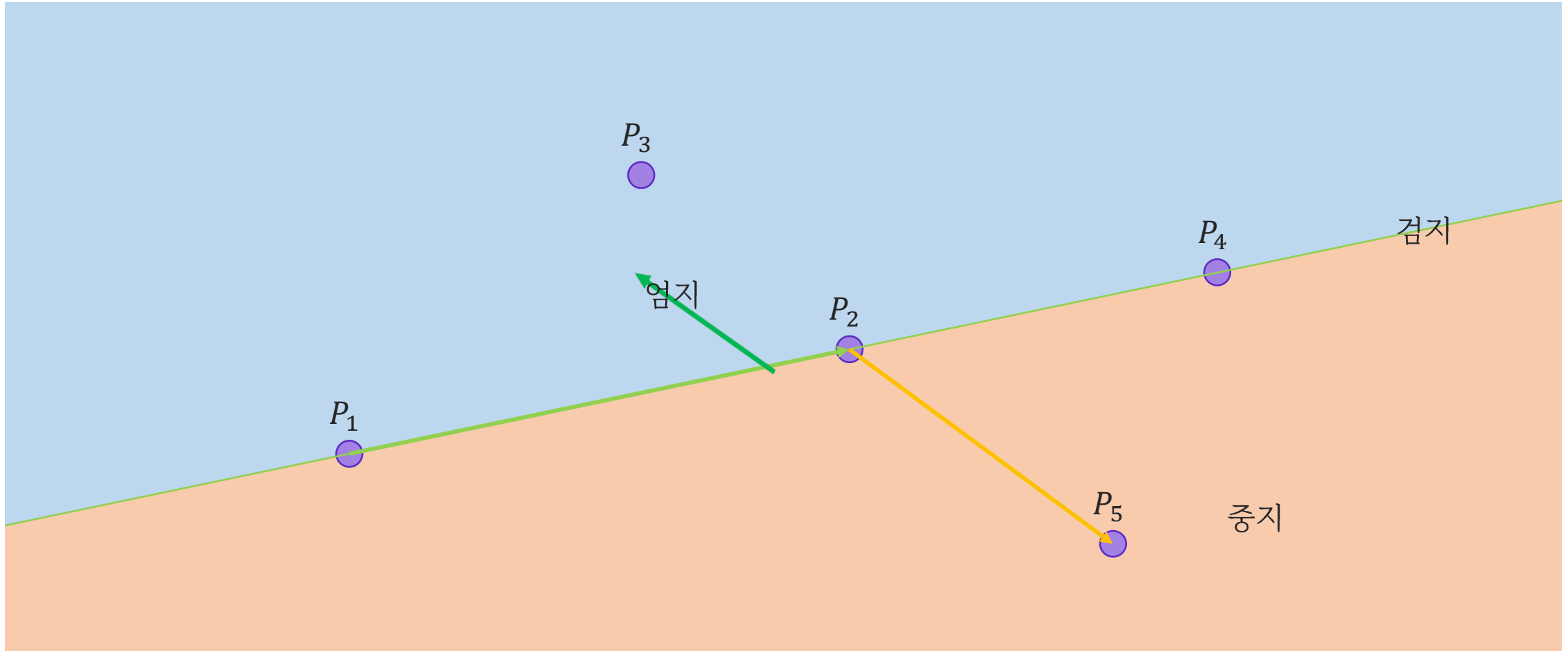
#CCW

<Counter Clockwise>



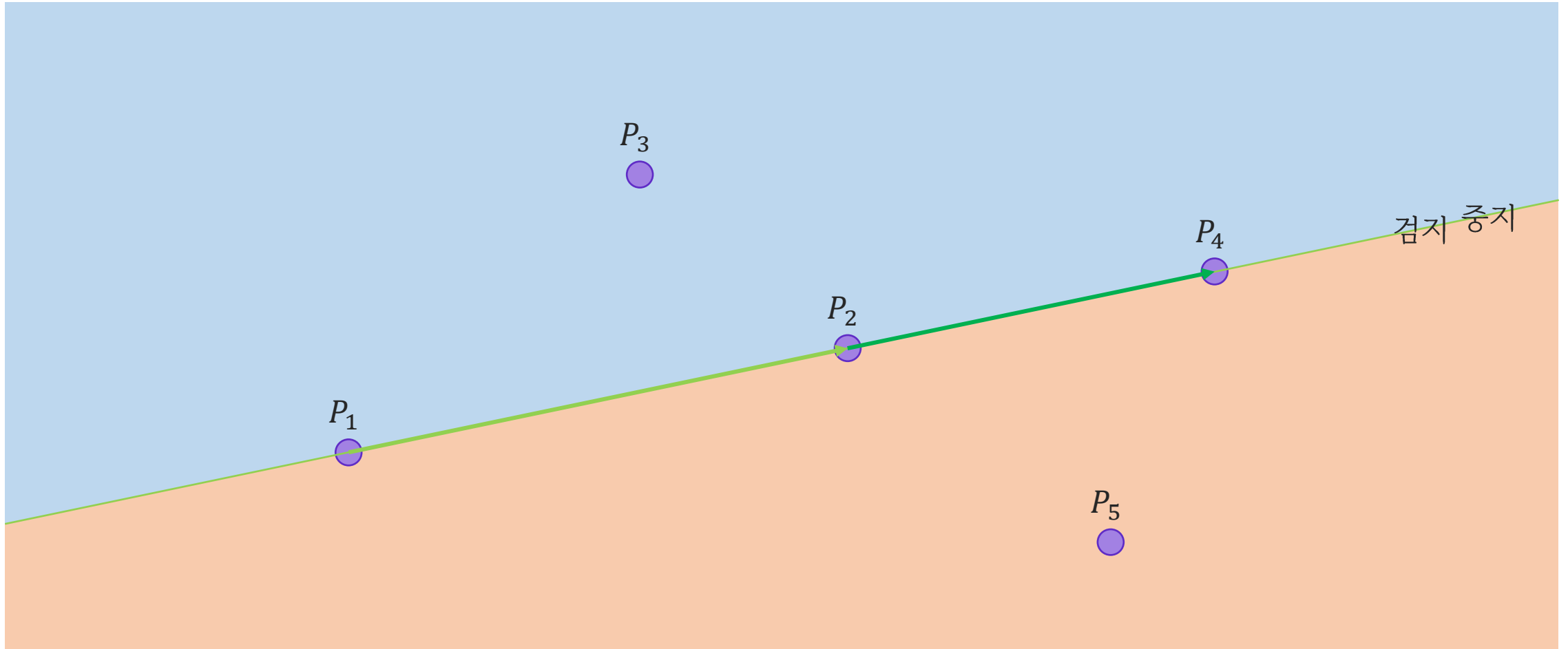
#CCW

<Counter Clockwise>



#CCW

<Counter Clockwise>




#CCW

<Counter Clockwise>

즉, P_1 과 P_2 를 잇는 벡터 \mathbf{v}_1 과 P_2 와 P_3 를 잇는 벡터 \mathbf{v}_2 의 외적을 사용해서 세 점의 배치 여부를 알아낼 수 있습니다!

조금 더 자세히 말하자면, 외적 결과의 부호를 사용할 수 있습니다.

근데 외적의 결과는 벡터 아닌가요? → 사용하는 벡터가 2차원이므로, 외적의 결과는 $(0, 0, ?)$ 가 되겠죠.

저희는 여기서 ?의 부호를 취할 것입니다.

다르게 말하자면, 벡터의 방향을 알아내는 것입니다.


#CCW

<Counter Clockwise>

이제 수식을 적어봅시다.

$$\mathbf{v}_1 \times \mathbf{v}_2$$

$$= [x_2 - x_1, y_2 - y_1] \times [x_3 - x_2, y_3 - y_2]$$

$$= \det \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_2 - x_1 & y_2 - y_1 & 0 \\ x_3 - x_2 & y_3 - y_2 & 0 \end{bmatrix}$$

$$= [0, 0, (x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1)]$$


#CCW

<Counter Clockwise>

$$\begin{aligned} & (x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1) \\ &= x_2y_3 - x_2y_2 - x_1y_3 + x_1y_2 - x_3y_2 + x_3y_1 + x_2y_2 - x_2y_1 \\ &= x_1y_2 + x_2y_3 + x_3y_1 - x_2y_1 - x_3y_2 - x_1y_3 \end{aligned}$$

벡터의 방향만 구하면 되니, 위 값의 부호만 알아내면 됩니다.

참고로, 반시계방향이라면 위 식은 양수가 나옵니다.

일직선이라면 0이 나오고, 시계방향이라면 음수가 나옵니다.

#CCW

<Counter Clockwise>

이 짧은 코드를 위해 열심히 달려왔습니다.

+ sgn(x)는 x의 부호에 맞춰서 -1, 0, +1을 반환하는 함수입니다.

```
using point2d = pair<int, int>;
#define x first
#define y second

inline int sgn(int x){ return (x > 0) - (x < 0); }

int ccw(point2d p1, point2d p2, point2d p3){
    int a = p1.x*p2.y + p2.x*p3.y + p3.x*p1.y;
    int b = p1.y*p2.x + p2.y*p3.x + p3.y*p1.x;
    return sgn(a-b);
}
```



#CCW

<Counter Clockwise>

 CCW (BOJ #11758)

<문제 설명>

- 세 점 P_1, P_2, P_3 가 주어진다.
- 세 점이 어떻게 배치되어 있는지 출력한다.


<제약 조건>

- $-10\,000 \leq \text{좌표} \leq 10\,000$
- 모든 좌표는 정수이다.




#CCW

<Counter Clockwise>

 CCW (BOJ #11758)

<문제 해설>

방금 배운 CCW를 그대로 구현해주면 됩니다.



#선분 교차

<Line Intersection>

두 선분이 주어질 때, 교차하는 점이 있는지 판별

1. 쉬운 버전과 어려운 버전이 있으며,
일단 쉬운 버전부터 풀어봅시다.

#선분 교차

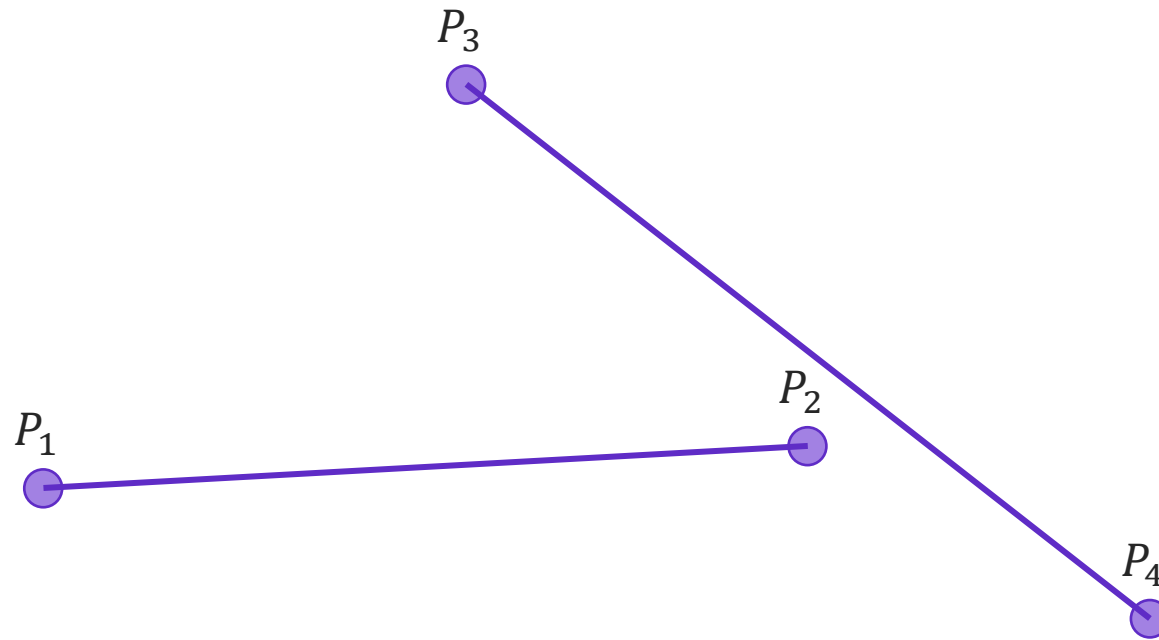
<Line Intersection> - Easy

두 선분이 주어질 때, 교차하는 점이 있는지 판별
단, 두 선분의 끝점 4개 중 세 점이 일직선 위에 있는 경우는 없다.

1. 저 조건이 왜 쉬움과 어려움을 가르는지는
어려운 버전을 풀면 알게 될 겁니다.

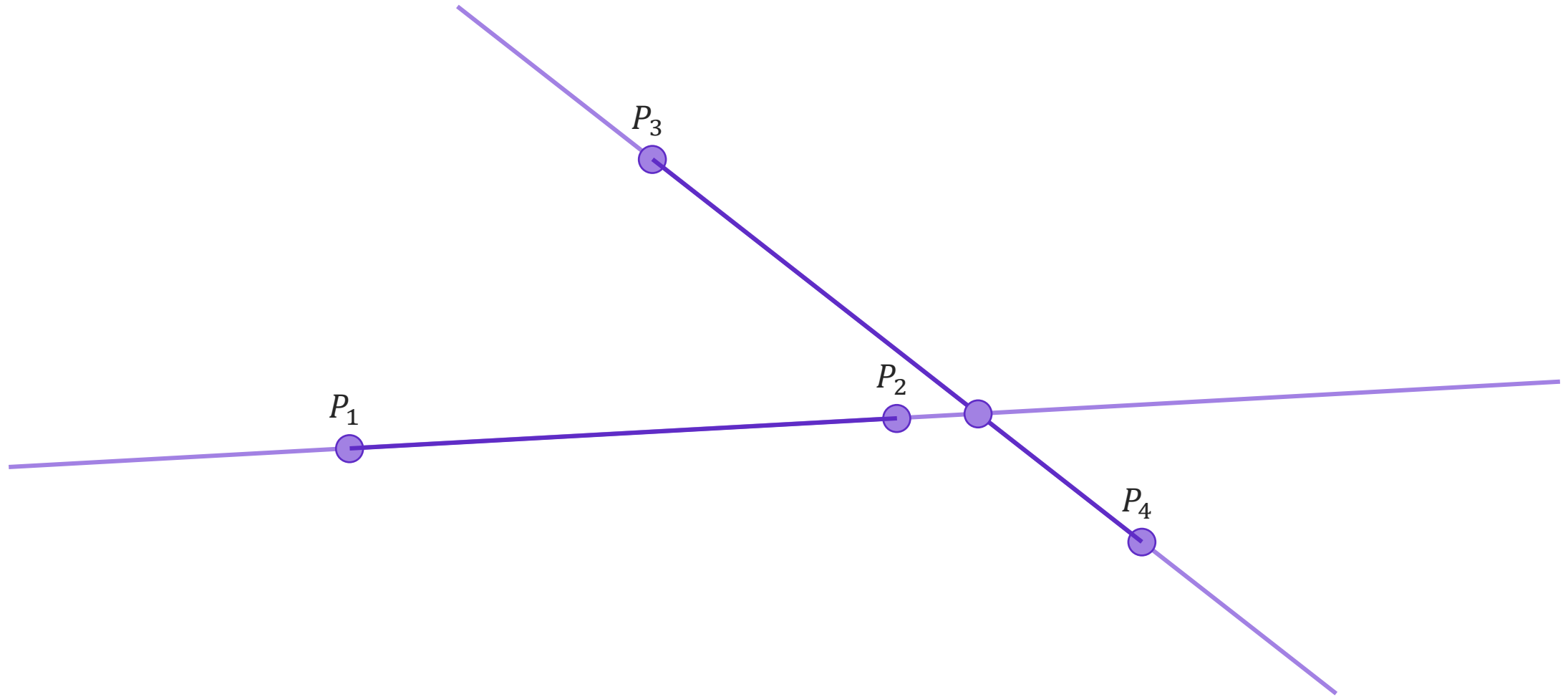
#선분 교차

<Line Intersection> - Easy



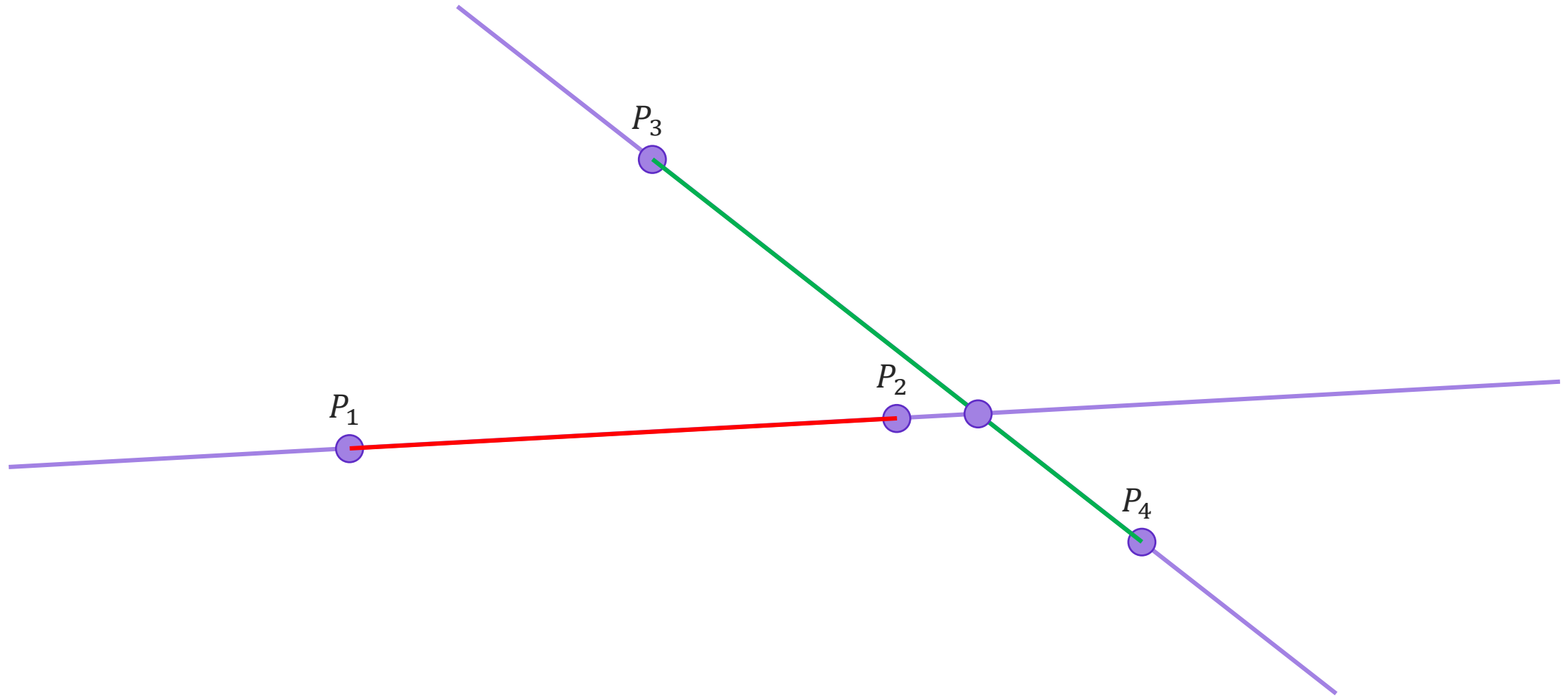
#선분 교차

<Line Intersection> - Easy



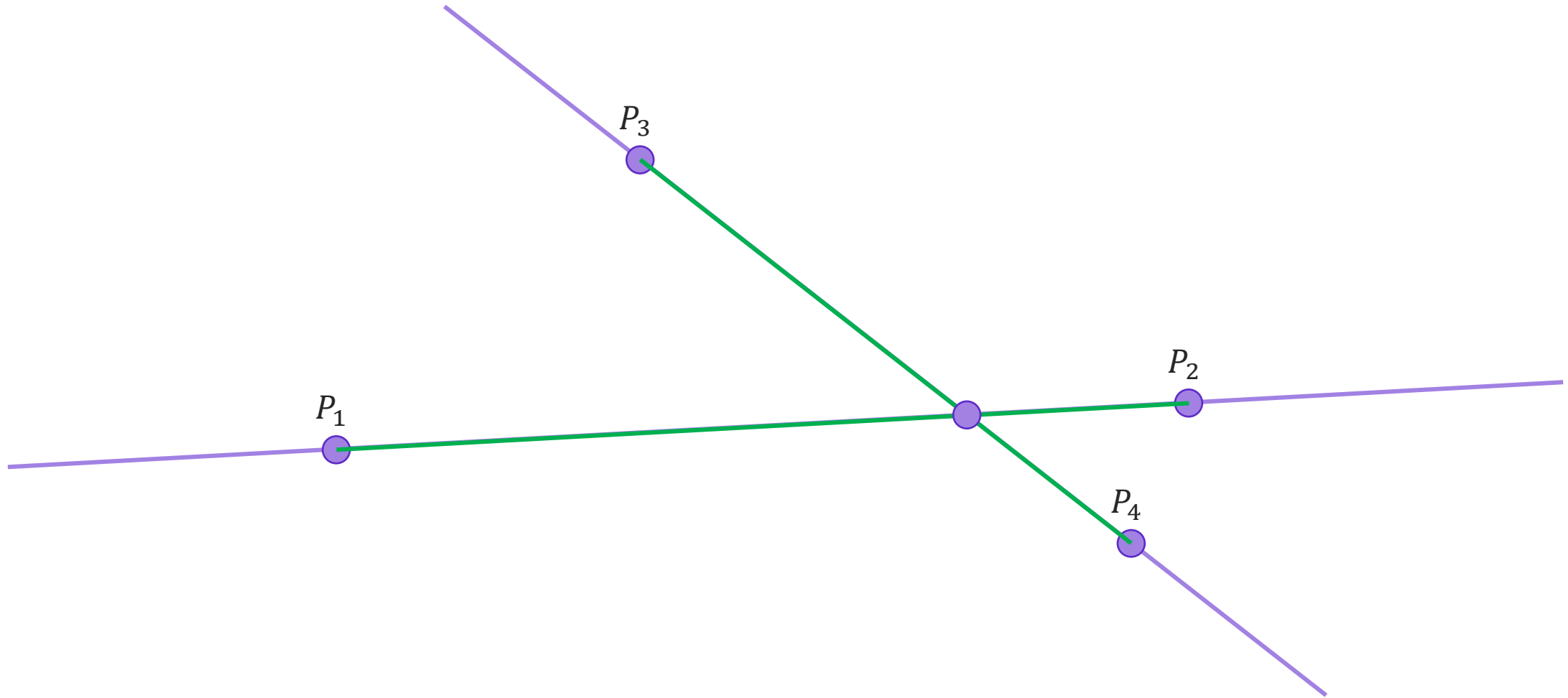
#선분 교차


<Line Intersection> - Easy



#선분 교차

<Line Intersection> - Easy





#선분 교차

<Line Intersection> - Easy

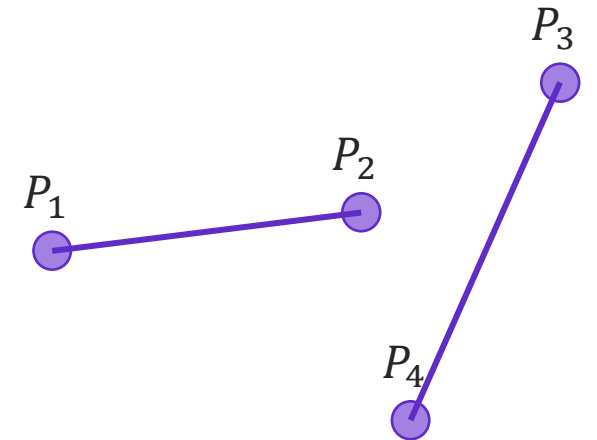
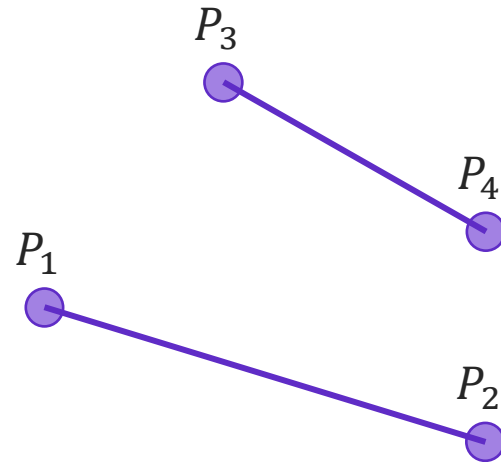
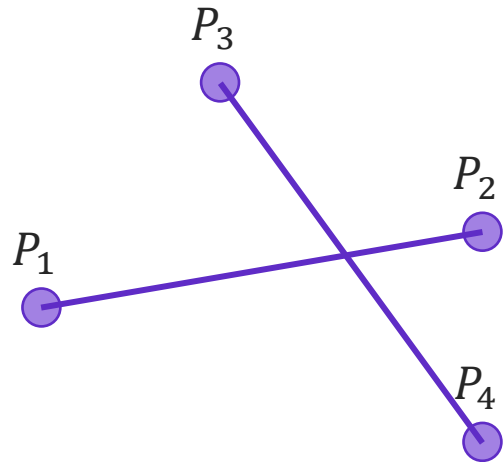
나누는 점을 직접 찾아서 계산할 수 있지만,
이런 **중간 과정 없이** 두 직선만으로 바로바로 찾아볼 수 있을까요?

Spoiler: 네. CCW를 써서 할 수 있습니다.



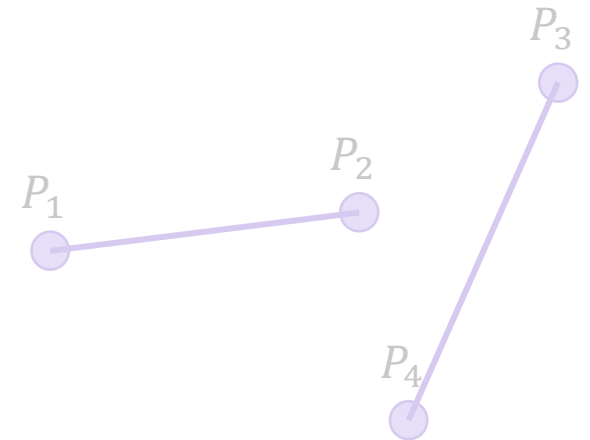
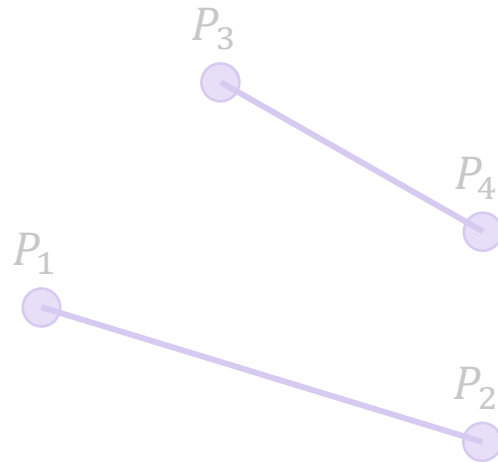
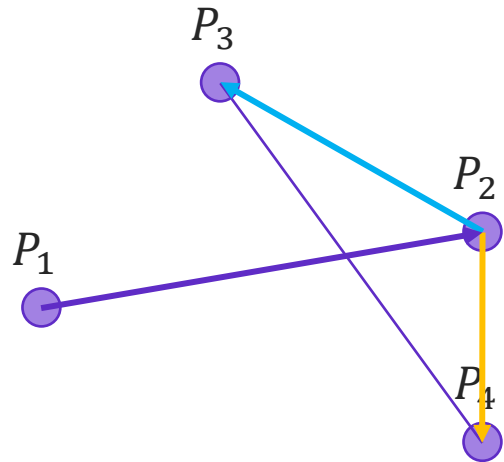
#선분 교차

<Line Intersection> - Easy



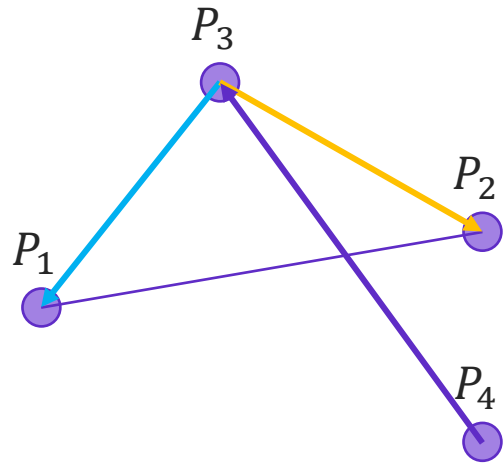
#선분 교차

<Line Intersection> - Easy

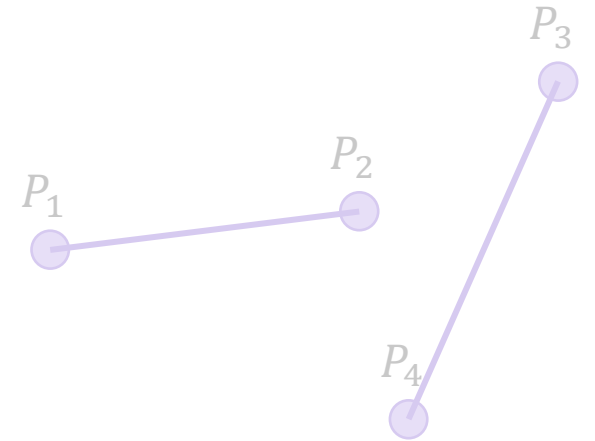
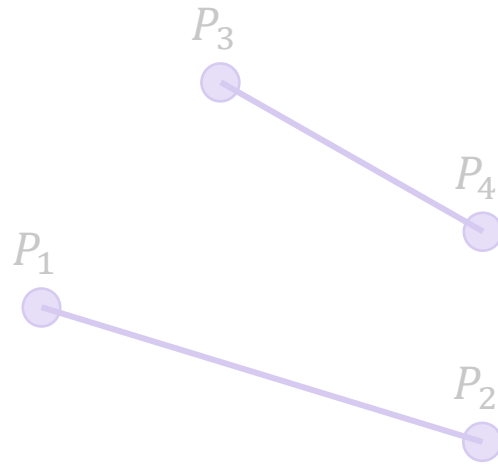


#선분 교차

<Line Intersection> - Easy

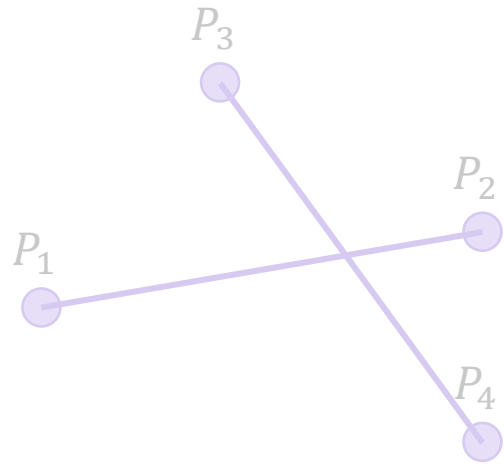


교차 O

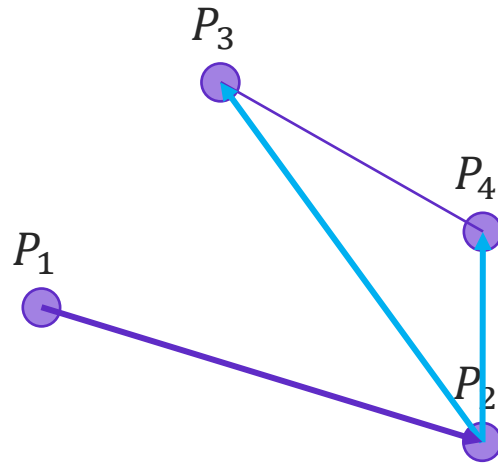


#선분 교차

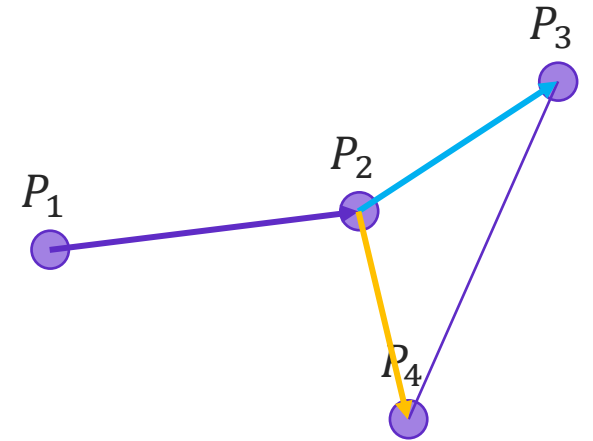
<Line Intersection> - Easy



교차 O

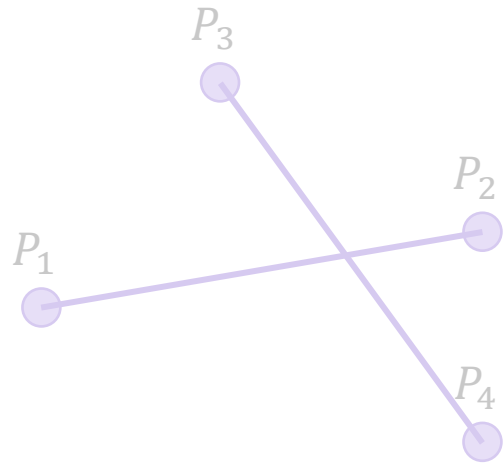


교차 X

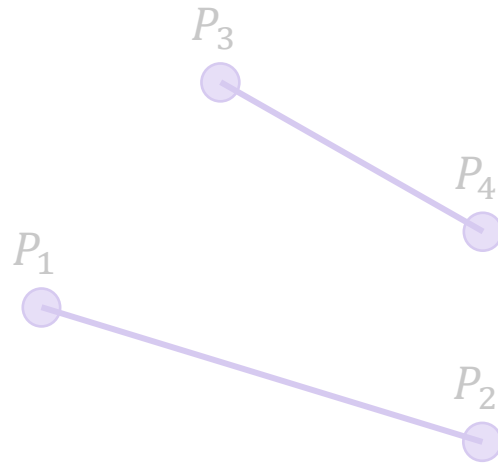


#선분 교차

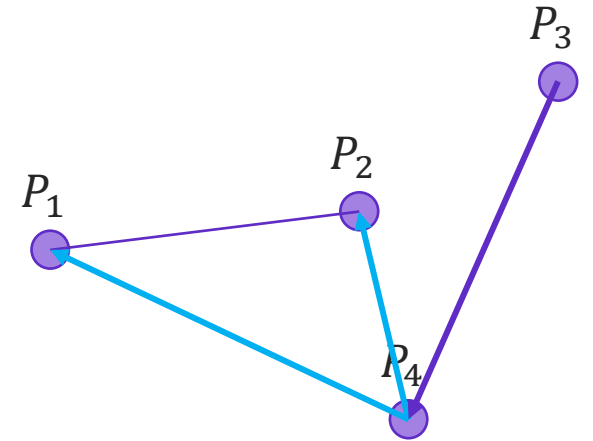
<Line Intersection> - Easy



교차 O



교차 X



교차 X

#선분 교차

<Line Intersection> - Easy

이 짧은 코드를 위해 열심히 달려왔습니다 2.

+ a와 b가 다른 부호임을 간단히 $a \times b < 0$ 으로 표현했습니다.

```
using line2d = pair<point2d, point2d>;

bool intersect(line2d l1, line2d l2){
    point2d p1 = l1.first, p2 = l1.second;
    point2d p3 = l2.first, p4 = l2.second;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    return c123*c124 < 0 && c341*c342 < 0;
}
```

#선분 교차

<Line Intersection> - Easy

3 선분 교차 1 (BOJ #17386)

<문제 설명>

- 선분 2개 (또는 점 4개)가 주어진다.
- 두 선분이 **교차하는지** 출력한다.

<제약 조건>

- $-1\,000\,000 \leq \text{좌표} \leq 1\,000\,000$
- 모든 좌표는 정수이다.
- 세 점이 일직선 위에 있는 경우는 없다.

#선분 교차

<Line Intersection> - Easy

3 선분 교차 1 (BOJ #17386)

<문제 해설>

방금 배운 선분 교차를 그대로 구현해주면 됩니다.

#선분 교차

<Line Intersection> - Hard

두 선분이 주어질 때, 교차하는 점이 있는지 판별

1. 세 점이 일직선이어야 한다는 조건이 왜 난이도를 바꿨을까요?

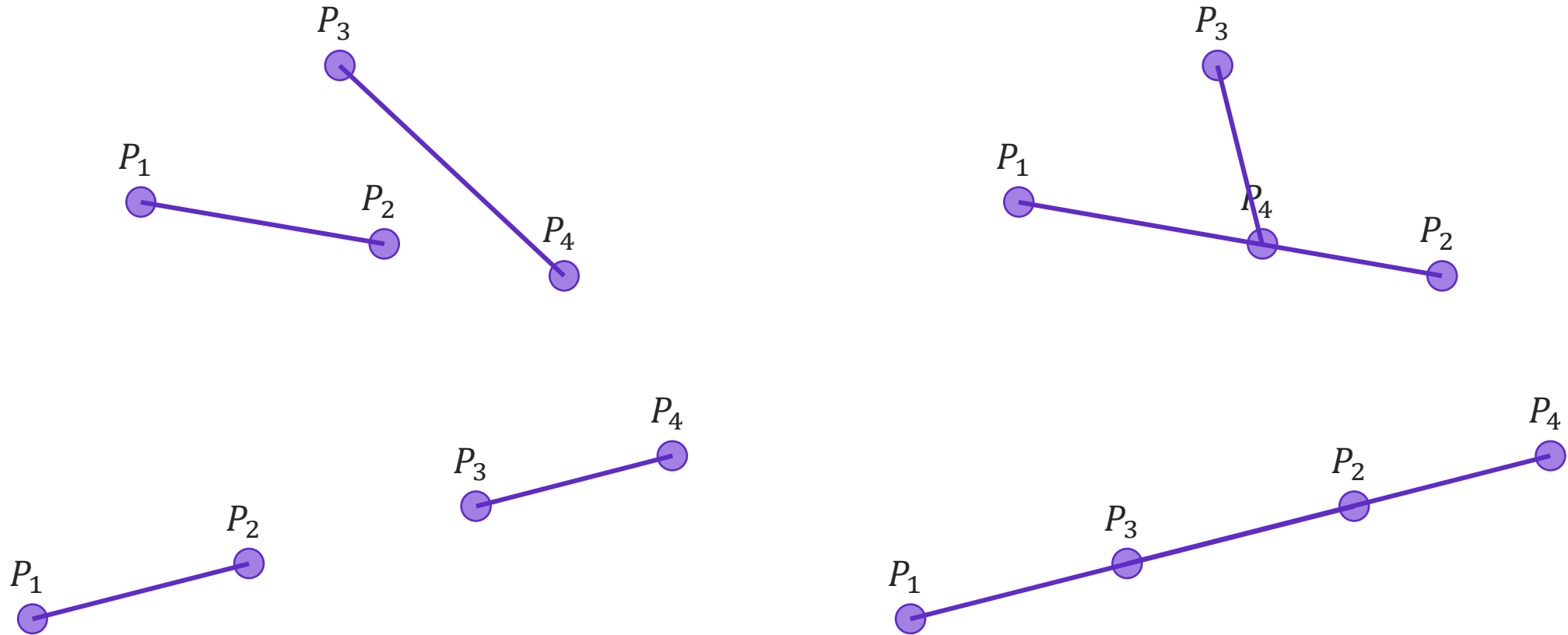
#선분 교차

<Line Intersection> - Hard

세 점이 일직선이라는 건, 세 점으로 CCW를 구하면 0이 나온다는 것을 의미합니다.
그러므로, 이 경우들만 따로 빼서 고민해봅시다.

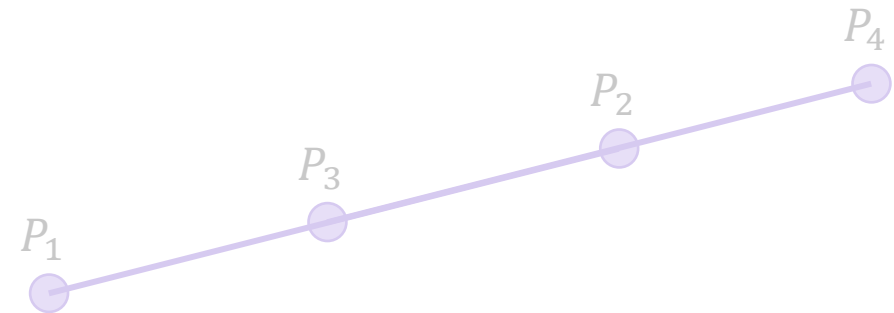
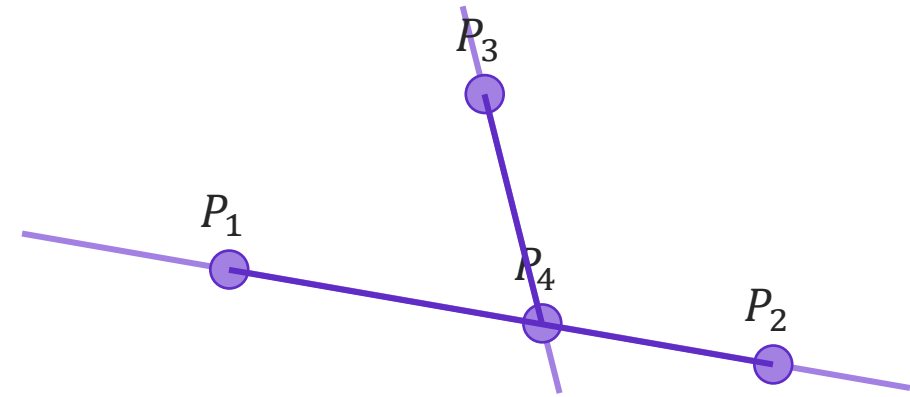
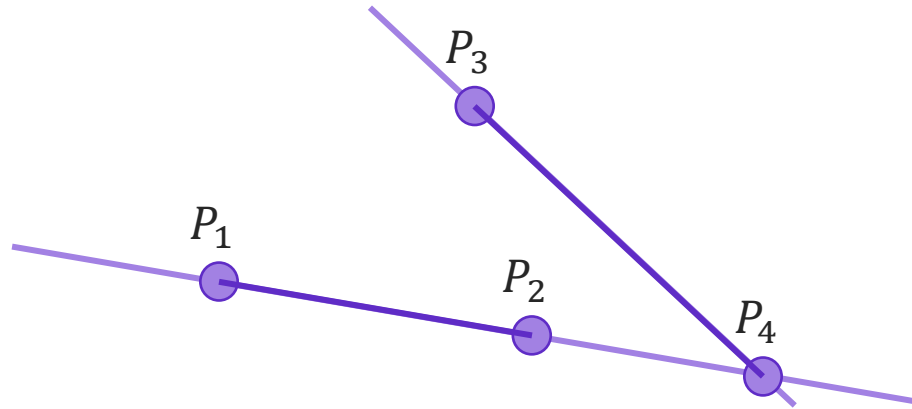
#선분 교차

<Line Intersection> - Hard



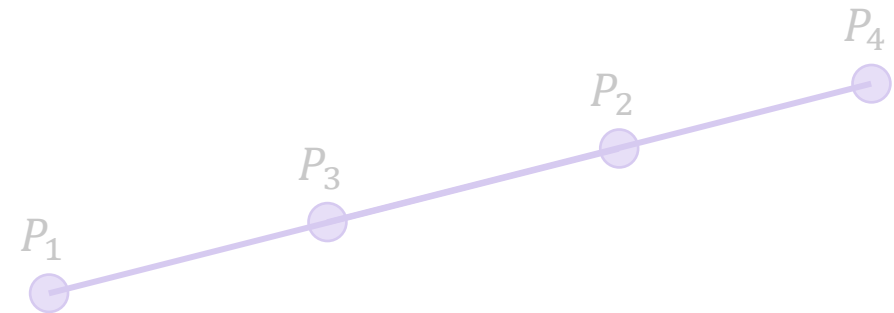
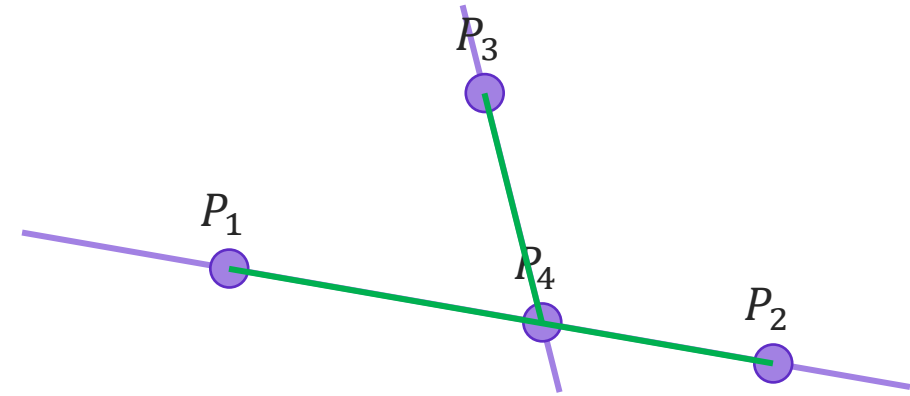
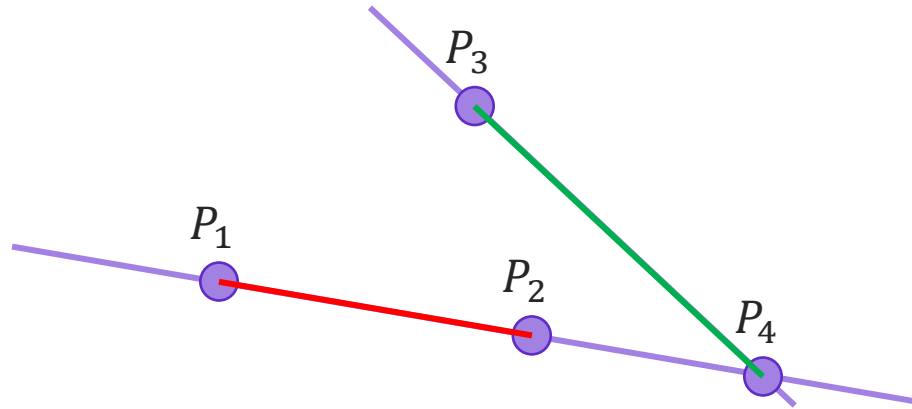
#선분 교차

<Line Intersection> - Hard



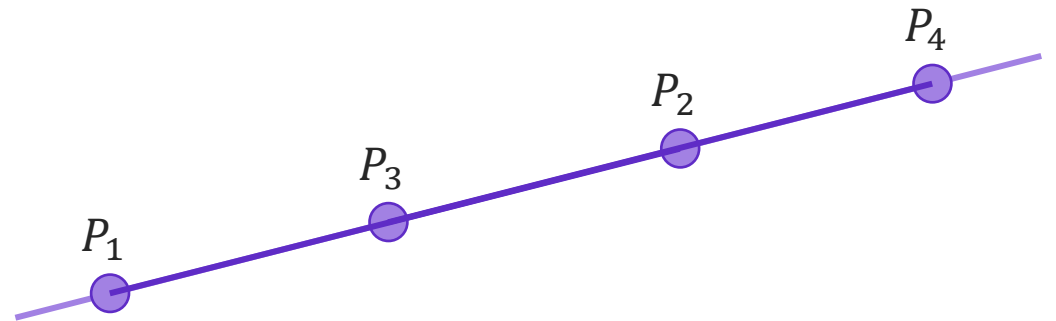
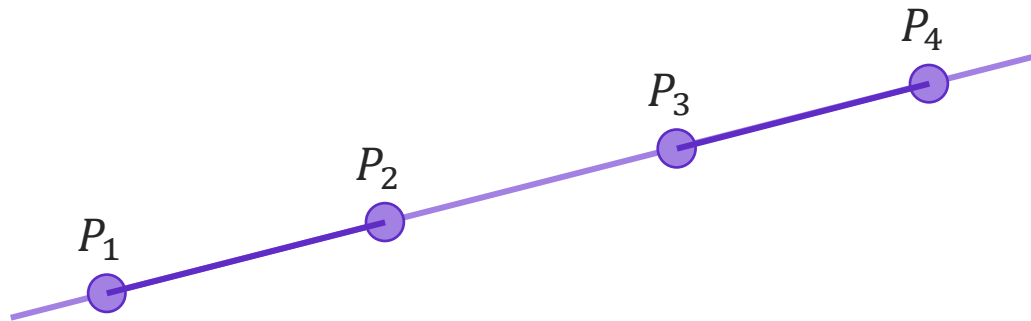
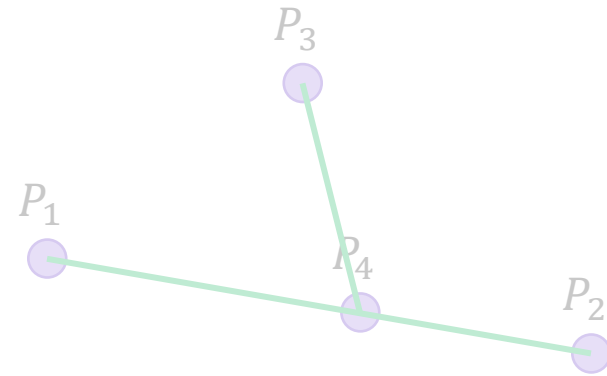
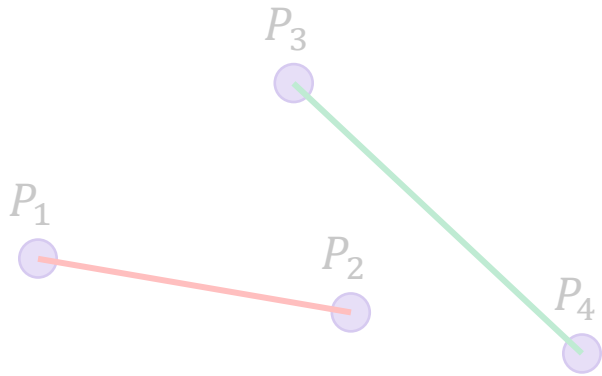
#선분 교차

<Line Intersection> - Hard



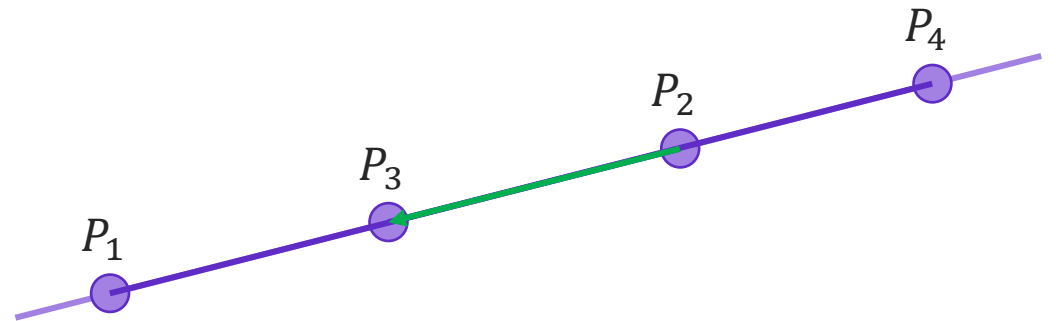
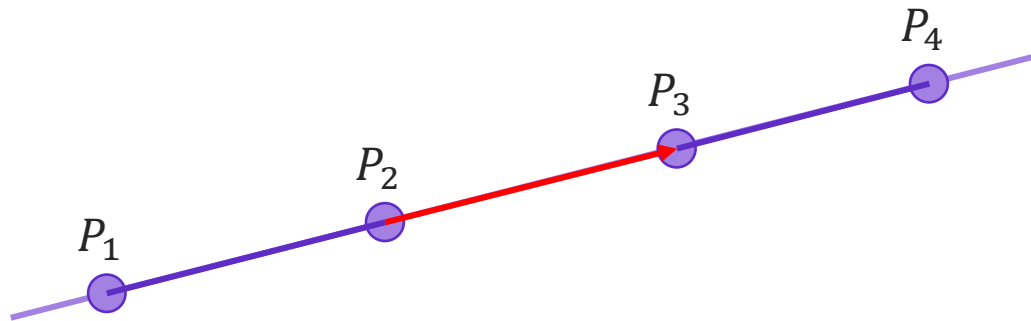
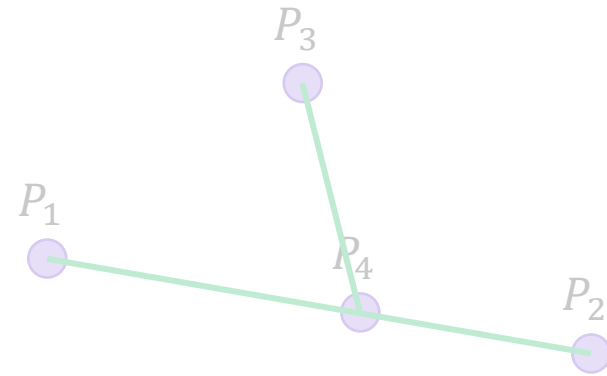
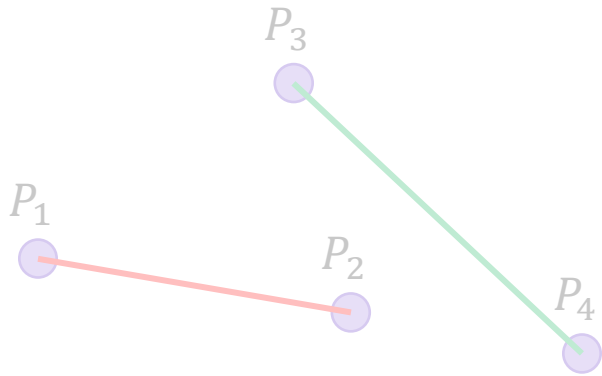
#선분 교차

<Line Intersection> - Hard



#선분 교차

<Line Intersection> - Hard



#선분 교차

<Line Intersection> - Hard

이제 조건문이 들어왔습니다. 그러니 하나하나 살펴보죠.

```
bool intersect(line2d l1, line2d l2){
    point2d p1 = l1.first, p2 = l1.second;
    point2d p3 = l2.first, p4 = l2.second;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    if (c123 == 0 && c124 == 0){
        if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
        return p2 >= p3 && p4 >= p1;
    }
    return c123*c124 <= 0 && c341*c342 <= 0;
}
```


#선분 교차

<Line Intersection> - Hard

우선 4개의 점이 일직선으로 나열되어 있는지
확인합니다.

```
bool intersect(line2d l1, line2d l2){
    point2d p1 = l1.first, p2 = l1.second;
    point2d p3 = l2.first, p4 = l2.second;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    if (c123 == 0 && c124 == 0){
        if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
        return p2 >= p3 && p4 >= p1;
    }
    return c123*c124 <= 0 && c341*c342 <= 0;
}
```

#선분 교차

<Line Intersection> - Hard

그 뒤로 일단 정규화를 거칩니다.

직선별로 먼저 나오는 점을 각각 P1과 P3로 고정시킵니다.

```
bool intersect(line2d l1, line2d l2){
    point2d p1 = l1.first, p2 = l1.second;
    point2d p3 = l2.first, p4 = l2.second;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    if (c123 == 0 && c124 == 0){
        if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
        return p2 >= p3 && p4 >= p1;
    }
    return c123*c124 <= 0 && c341*c342 <= 0;
}
```

#선분 교차

<Line Intersection> - Hard

이제 두 직선에 **겹치는 부분이 있는지** 확인합니다.

이는 한 직선이 끝난 뒤에 다른 직선이 나오는지 확인하면 됩니다.

옆의 코드에서는 그 경우가 발생하지 않음을 판별 중입니다.


```
bool intersect(line2d l1, line2d l2){
    point2d p1 = l1.first, p2 = l1.second;
    point2d p3 = l2.first, p4 = l2.second;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    if (c123 == 0 && c124 == 0){
        if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
        return p2 >= p3 && p4 >= p1;
    }
    return c123*c124 <= 0 && c341*c342 <= 0;
}
```

#선분 교차

<Line Intersection> - Hard

남은 건 세 점 일직선 조건이 추가된 일반적인 경우로, 일직선인 경우 역시 가능함으로 처리하면 됩니다.

```
bool intersect(line2d l1, line2d l2){
    point2d p1 = l1.first, p2 = l1.second;
    point2d p3 = l2.first, p4 = l2.second;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    if (c123 == 0 && c124 == 0){
        if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
        return p2 >= p3 && p4 >= p1;
    }
    return c123*c124 <= 0 && c341*c342 <= 0;
}
```



#선분 교차

<Line Intersection> - Hard

선분 교차 2 (BOJ #17387)

<문제 설명>

- 선분 2개 (또는 점 4개)가 주어진다.
- 두 선분이 **교차**하는지 출력한다.

<제약 조건>

- $-1\,000\,000 \leq \text{좌표} \leq 1\,000\,000$
- 모든 좌표는 정수이다.



#선분 교차

<Line Intersection> - Hard

2 선분 교차 2 (BOJ #17387)

<문제 해설>

방금 배운 선분 교차를 그대로 구현해주면 됩니다.

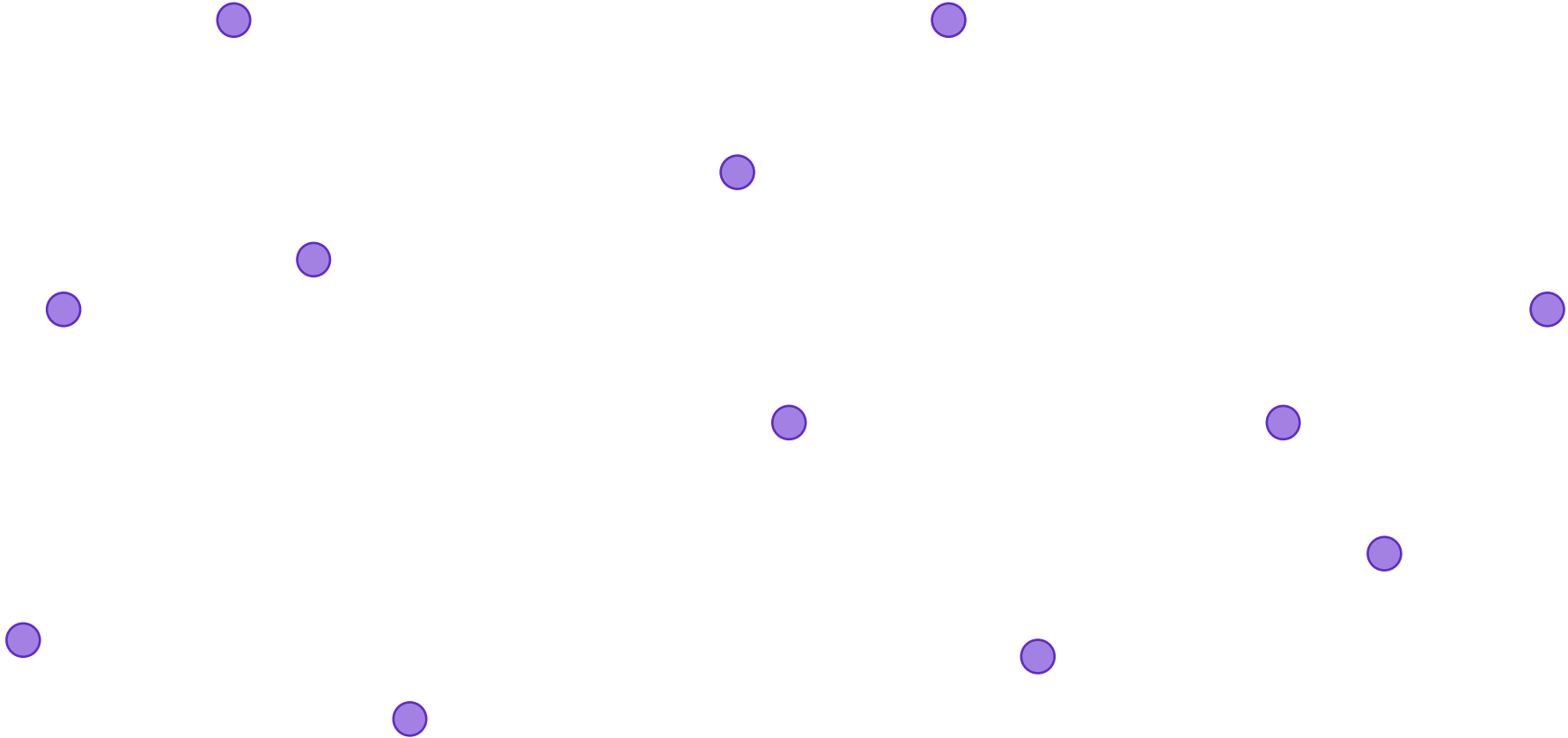
#볼록 껍질

<Convex Hull>

N 개의 점이 주어질 때, 모든 점을 포함하는 가장 작은 볼록한 다각형

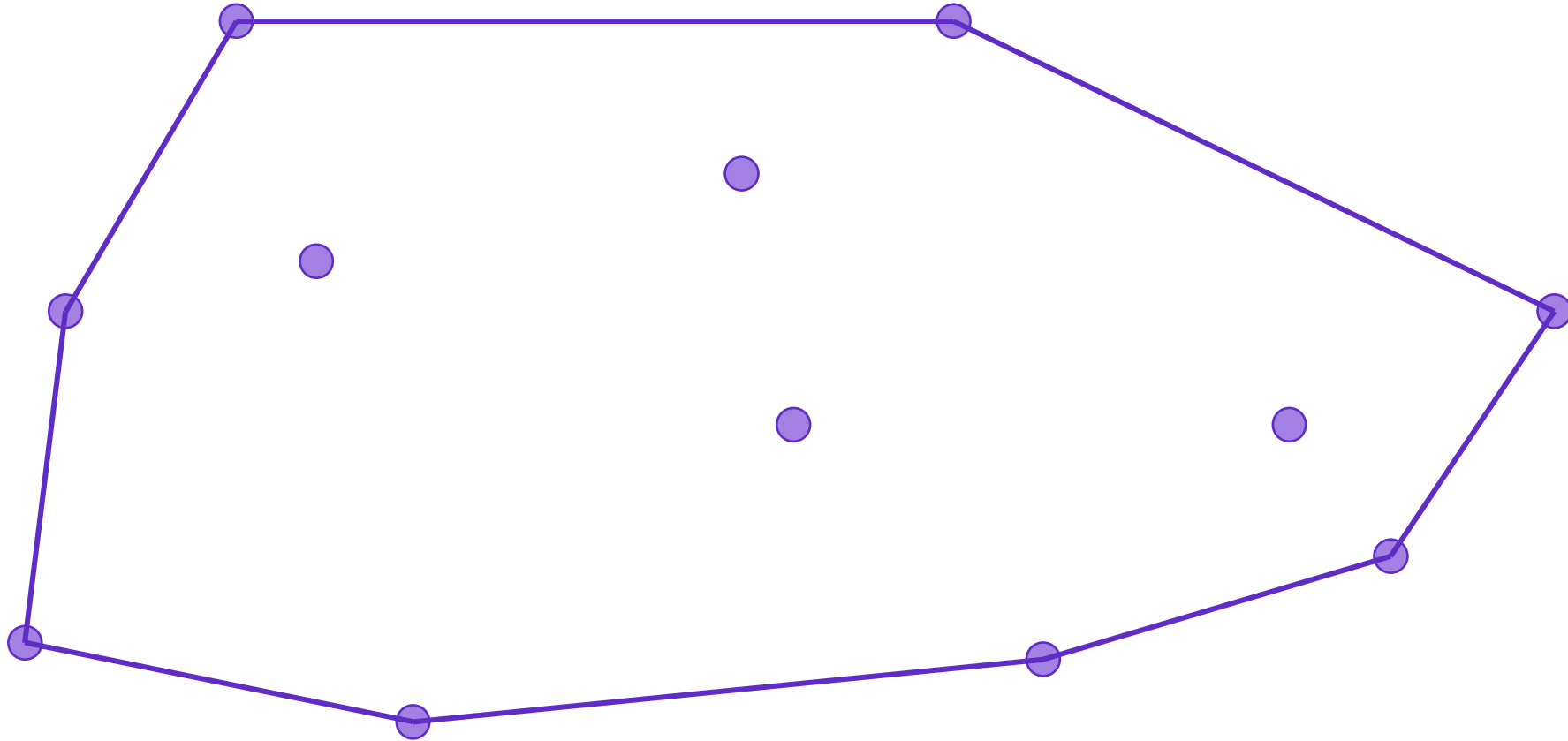
#볼록 껍질

<Convex Hull>



#볼록 껍질

<Convex Hull>



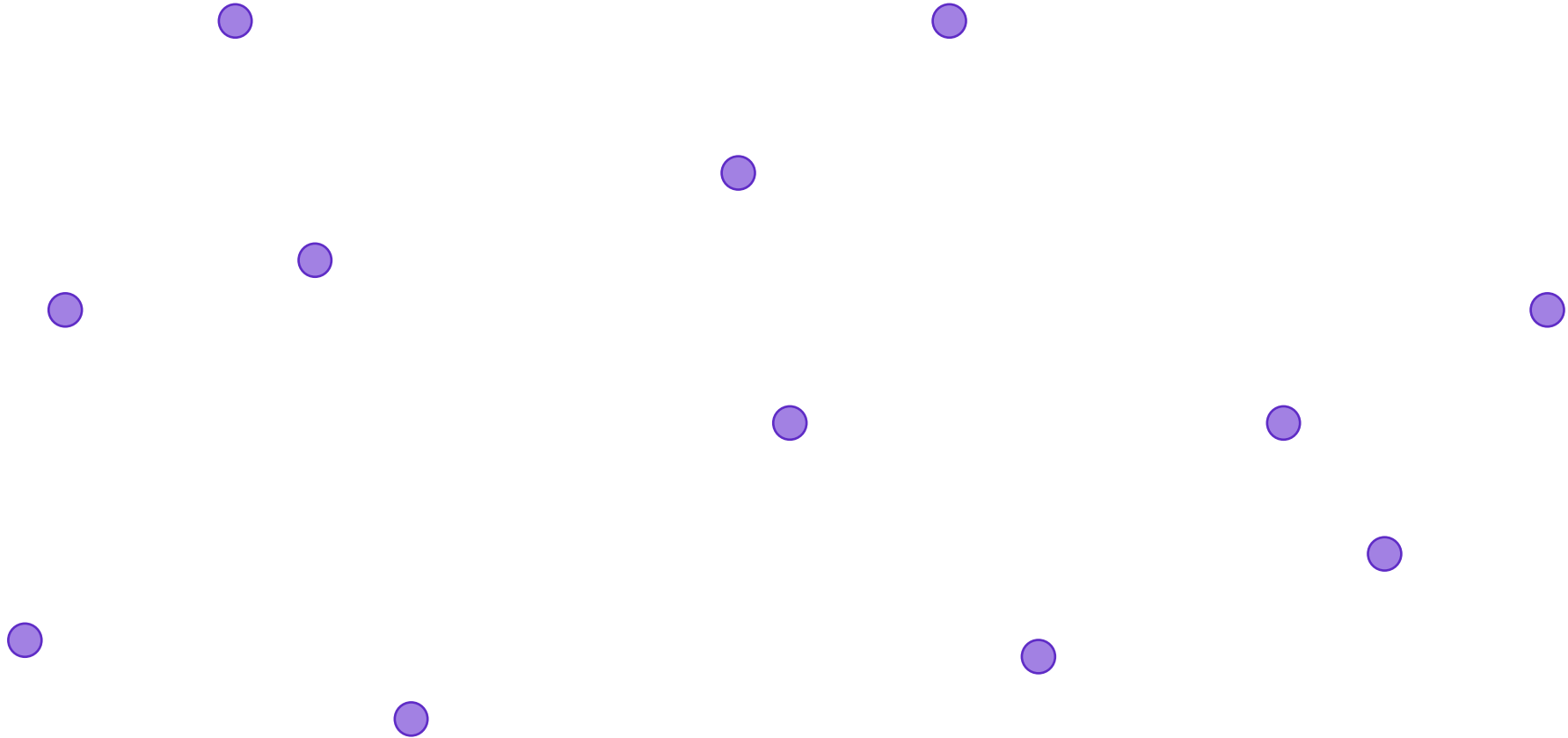
#볼록 껍질

<Convex Hull>

눈으로는 잘 보이는데, 어떻게 구할까요?

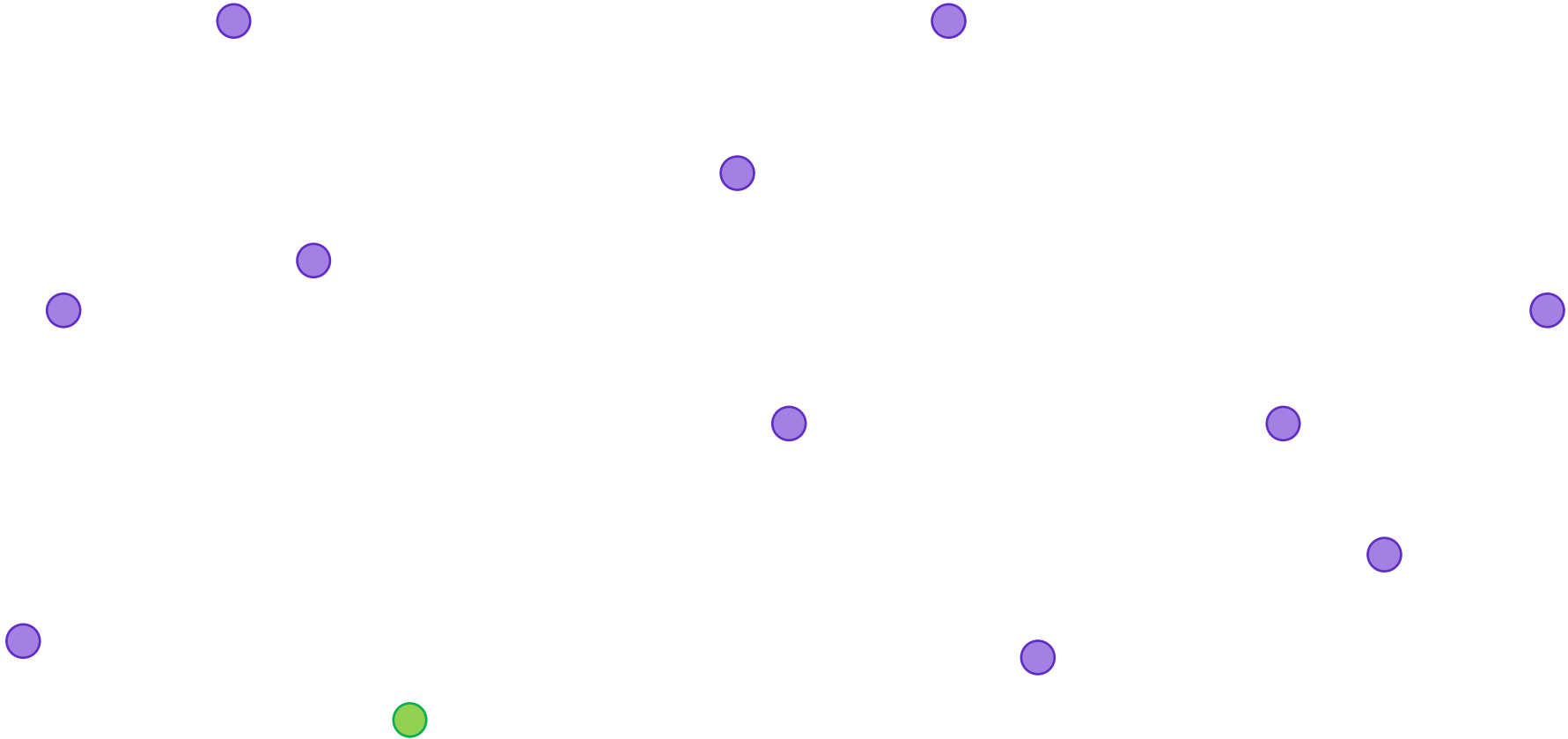
#볼록 껍질

<Convex Hull>



#볼록 껍질

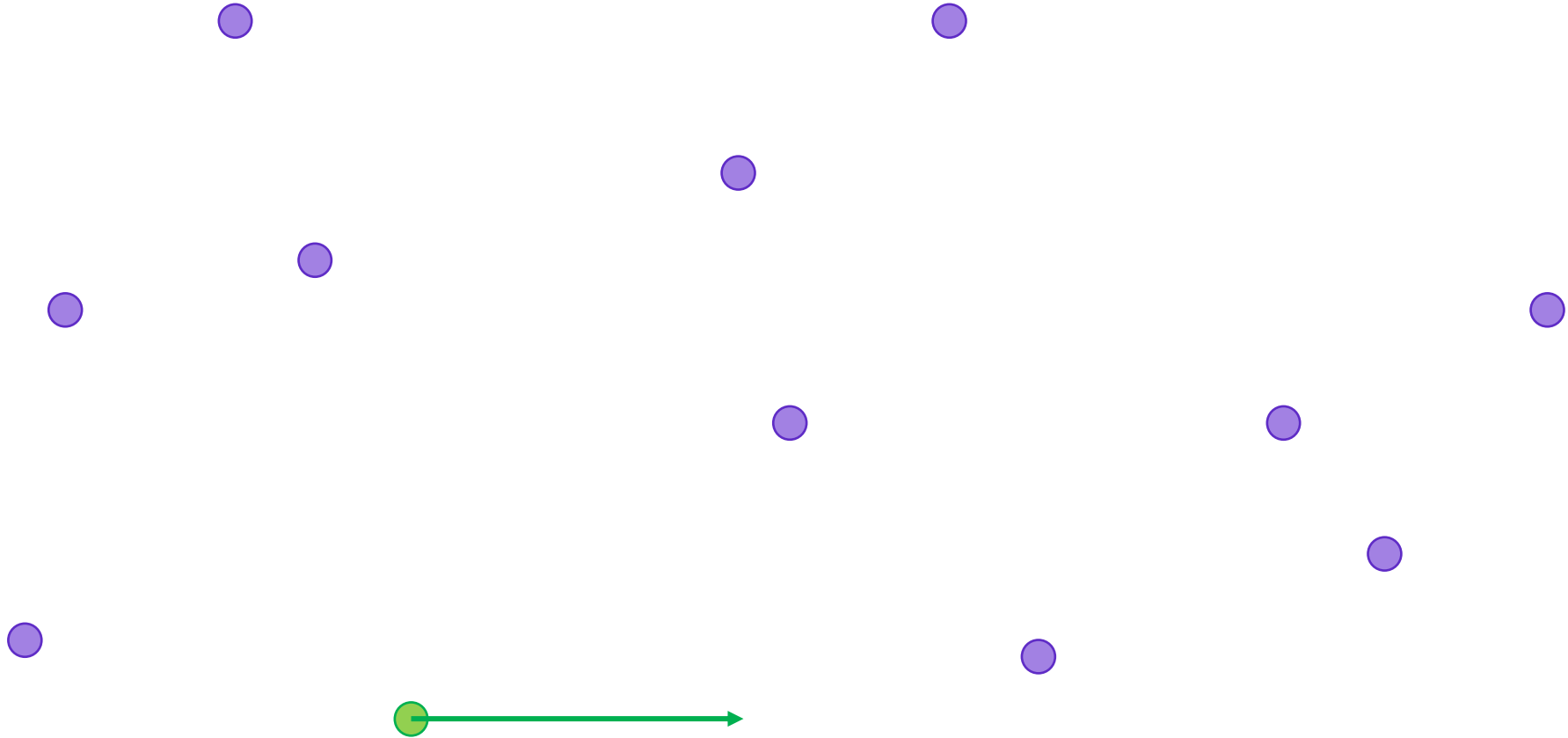
<Convex Hull>



1. 가장 낮은 위치에 있는 점을 찾는다.

#볼록 껍질

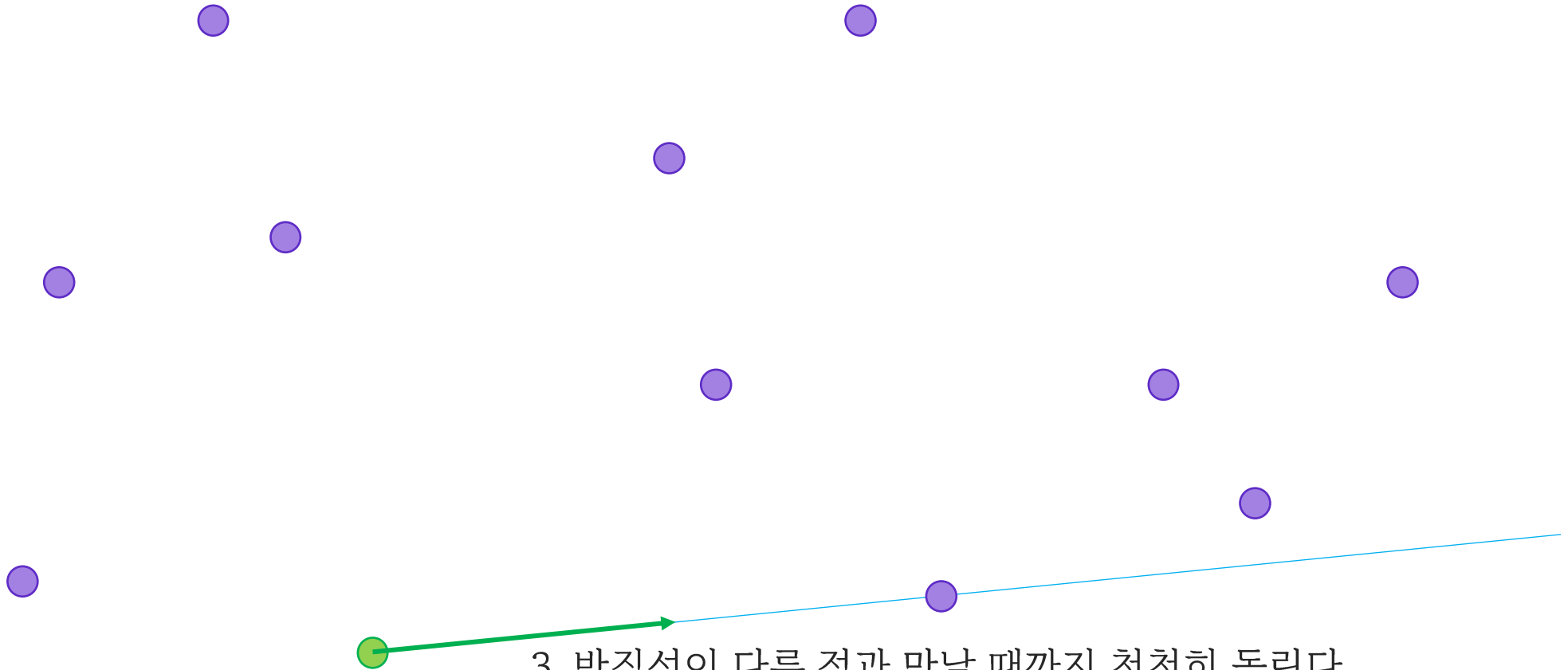
<Convex Hull>



2. x축에 평행한 반직선을 긋는다.

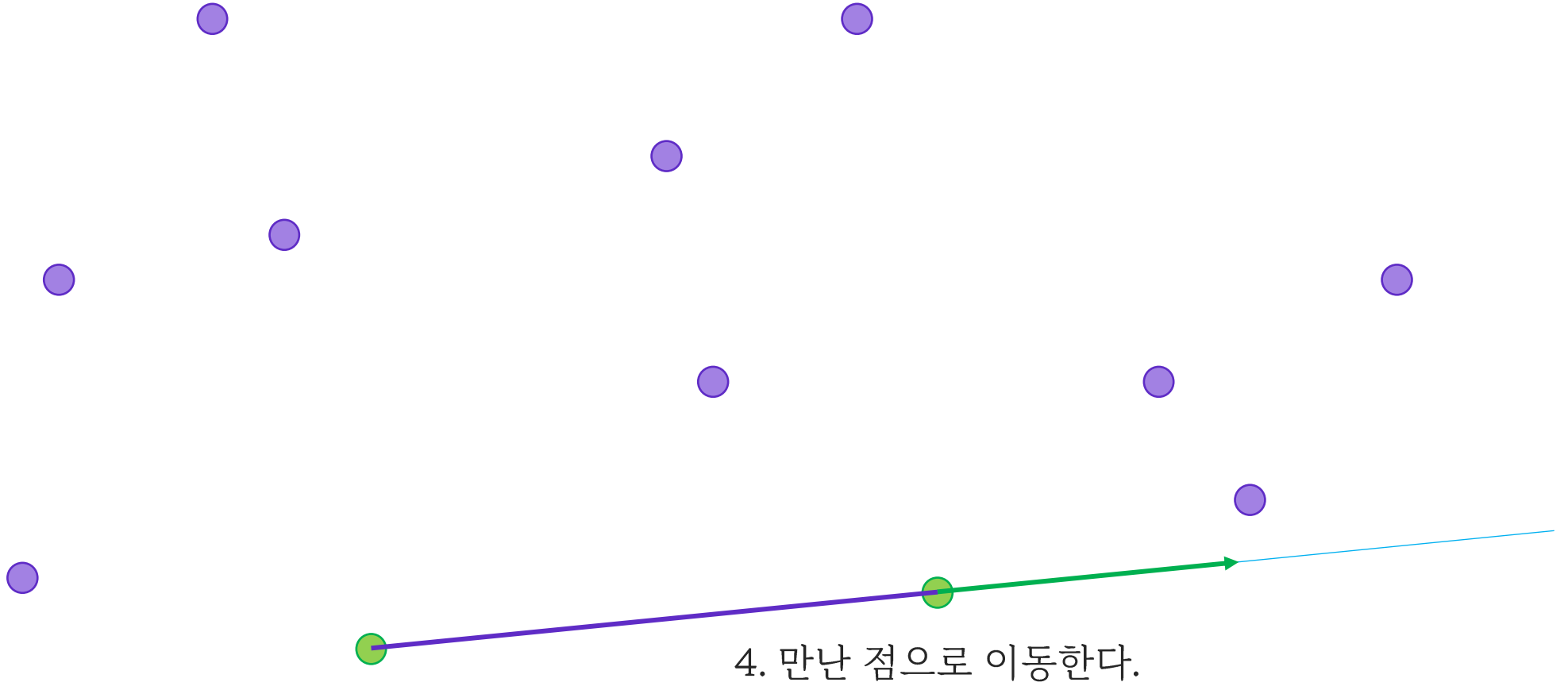
#볼록 껍질

<Convex Hull>



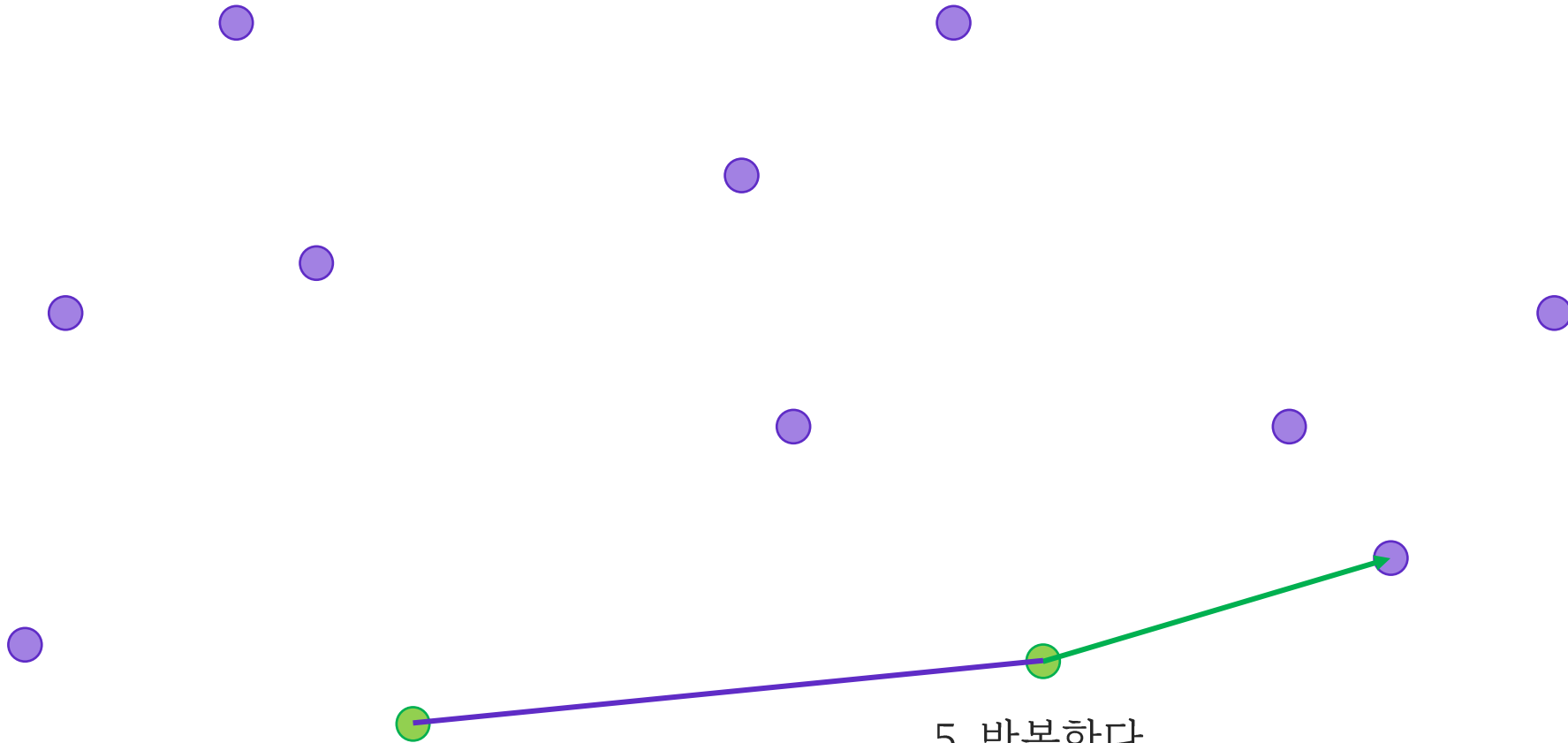
#볼록 껍질

<Convex Hull>



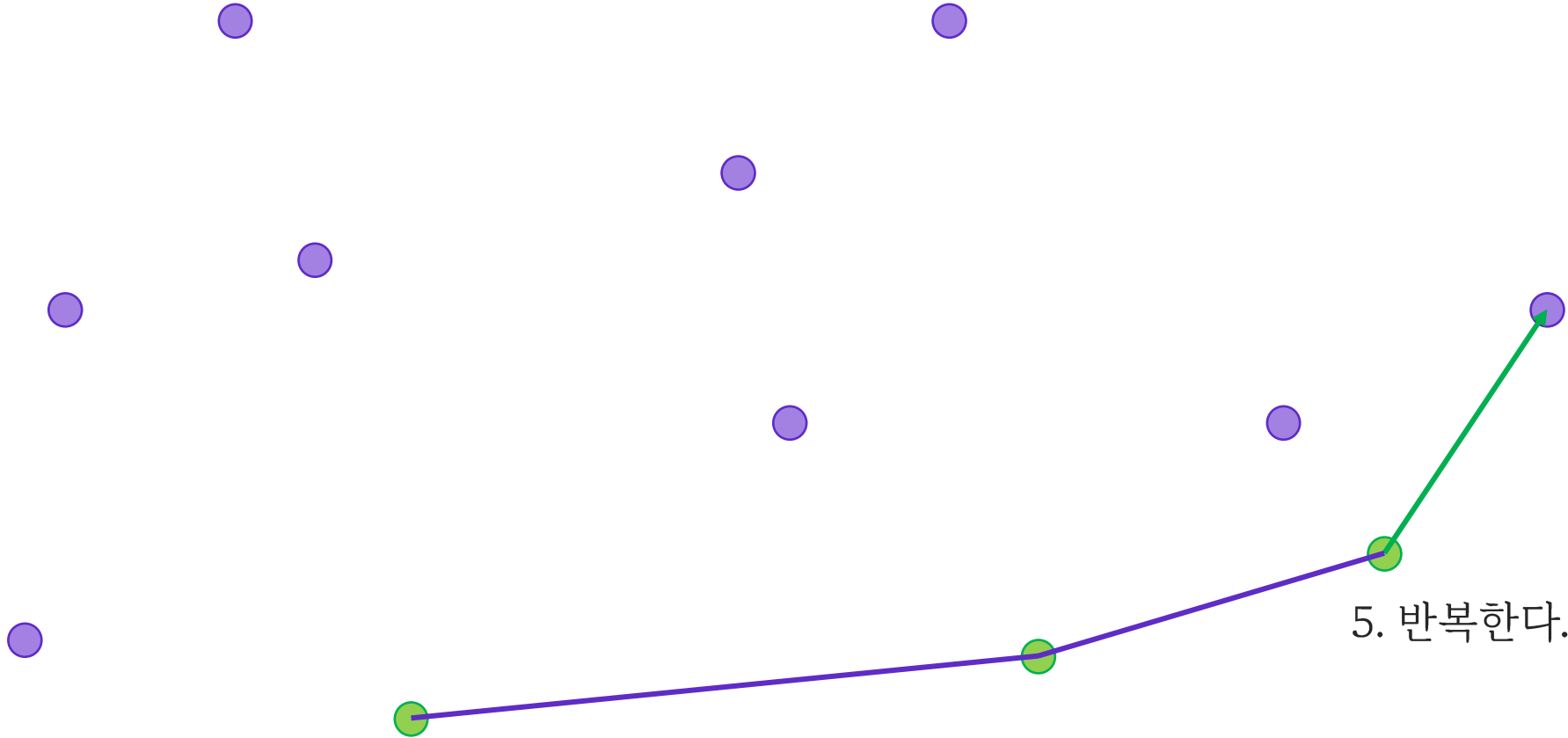
#볼록 껍질

<Convex Hull>



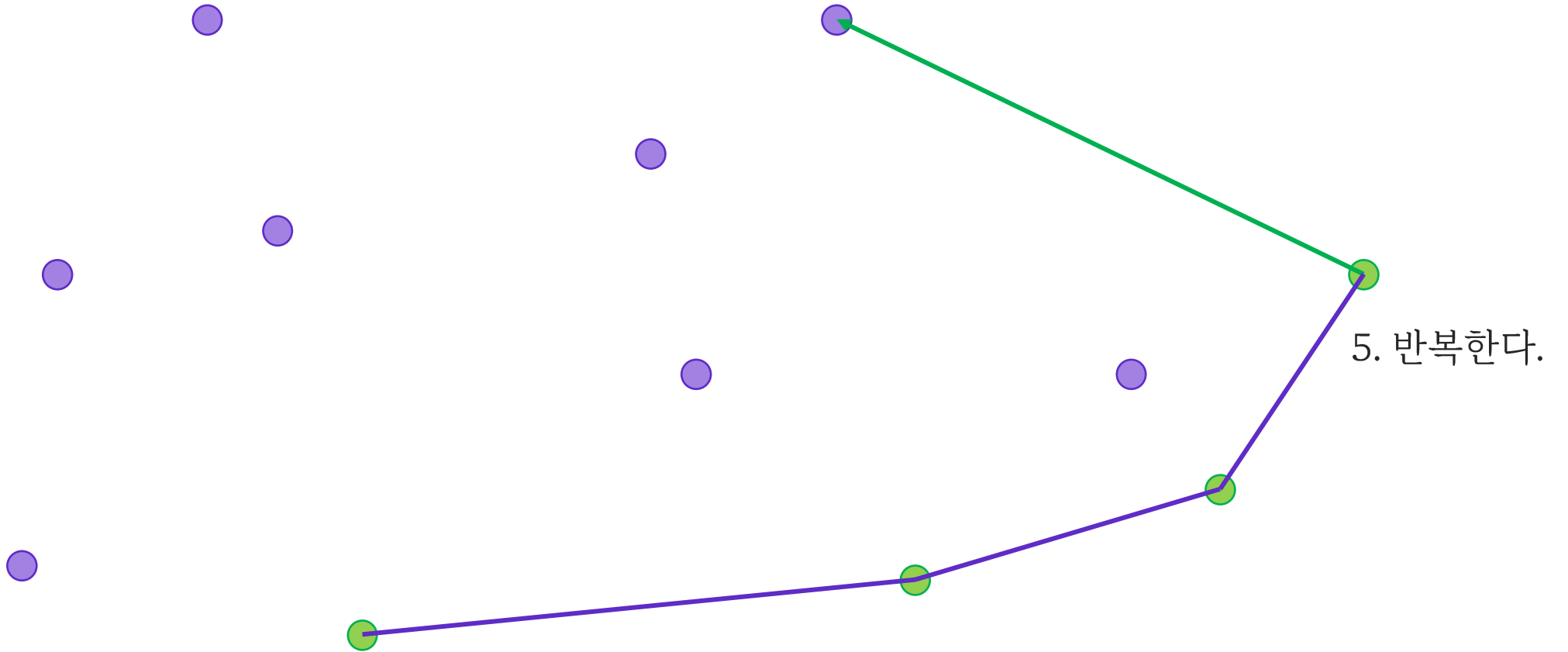
#볼록 껍질

<Convex Hull>



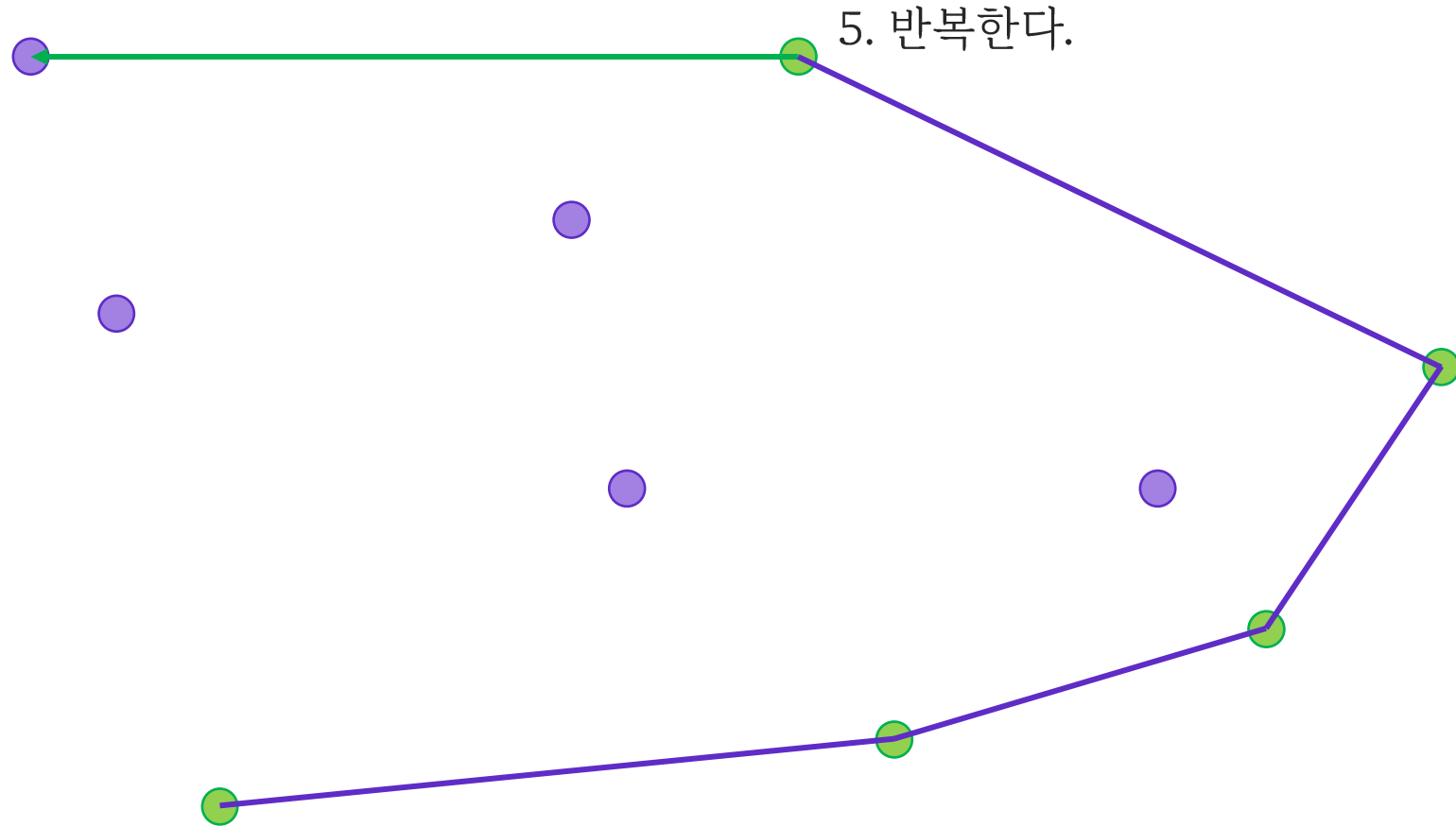
#볼록 껍질

<Convex Hull>



#볼록 껍질

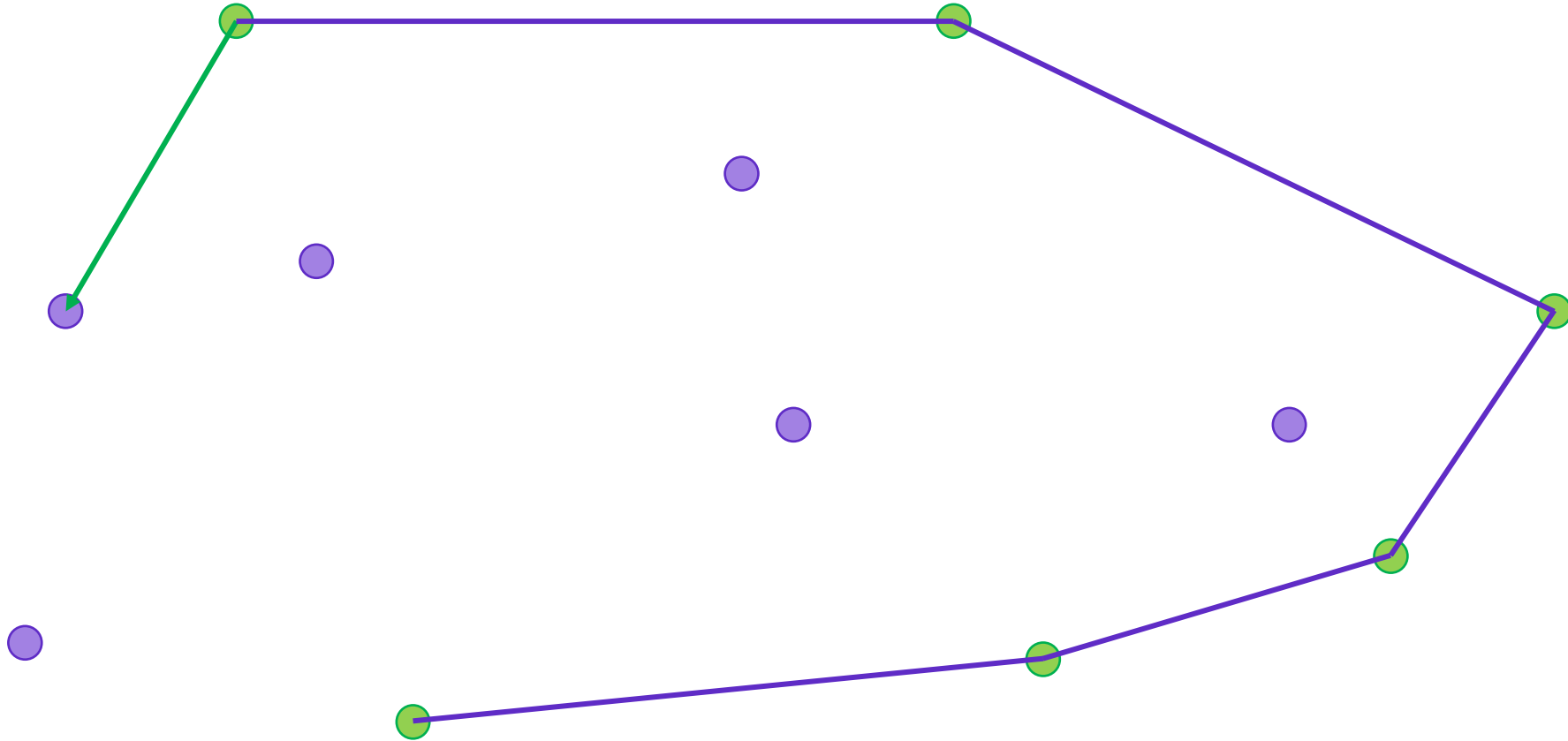
<Convex Hull>



#볼록 껍질

<Convex Hull>

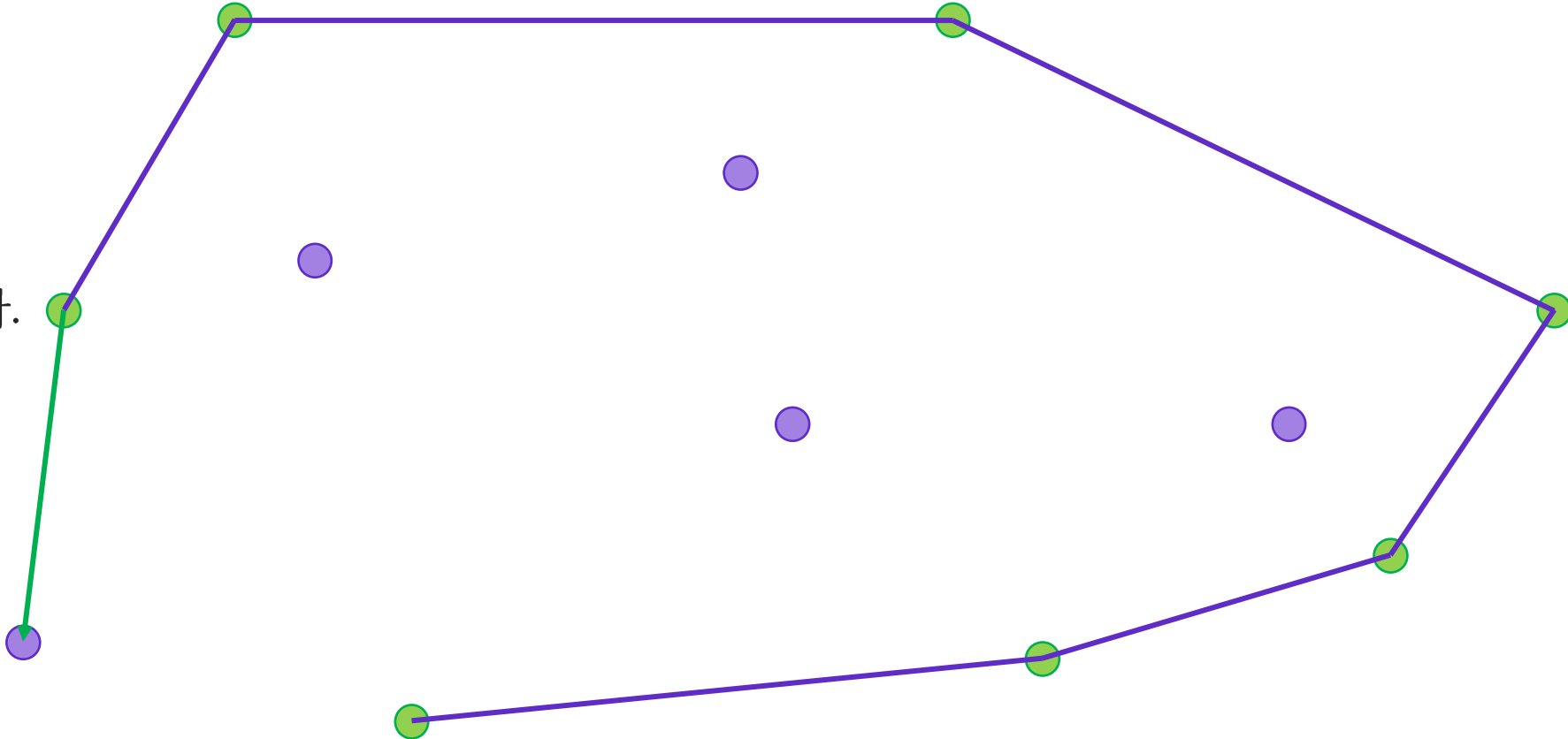
5. 반복한다.



#볼록 껍질

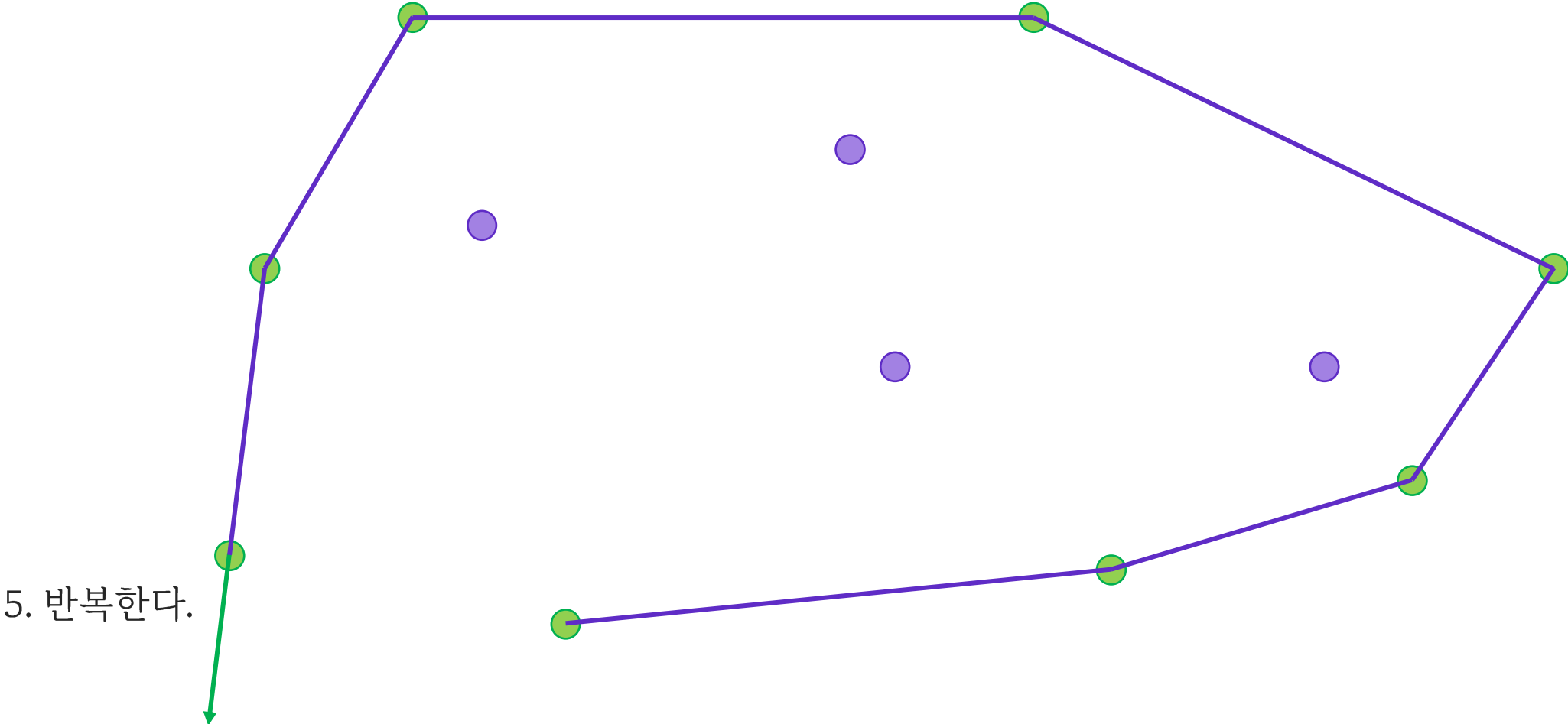
<Convex Hull>

5. 반복한다.



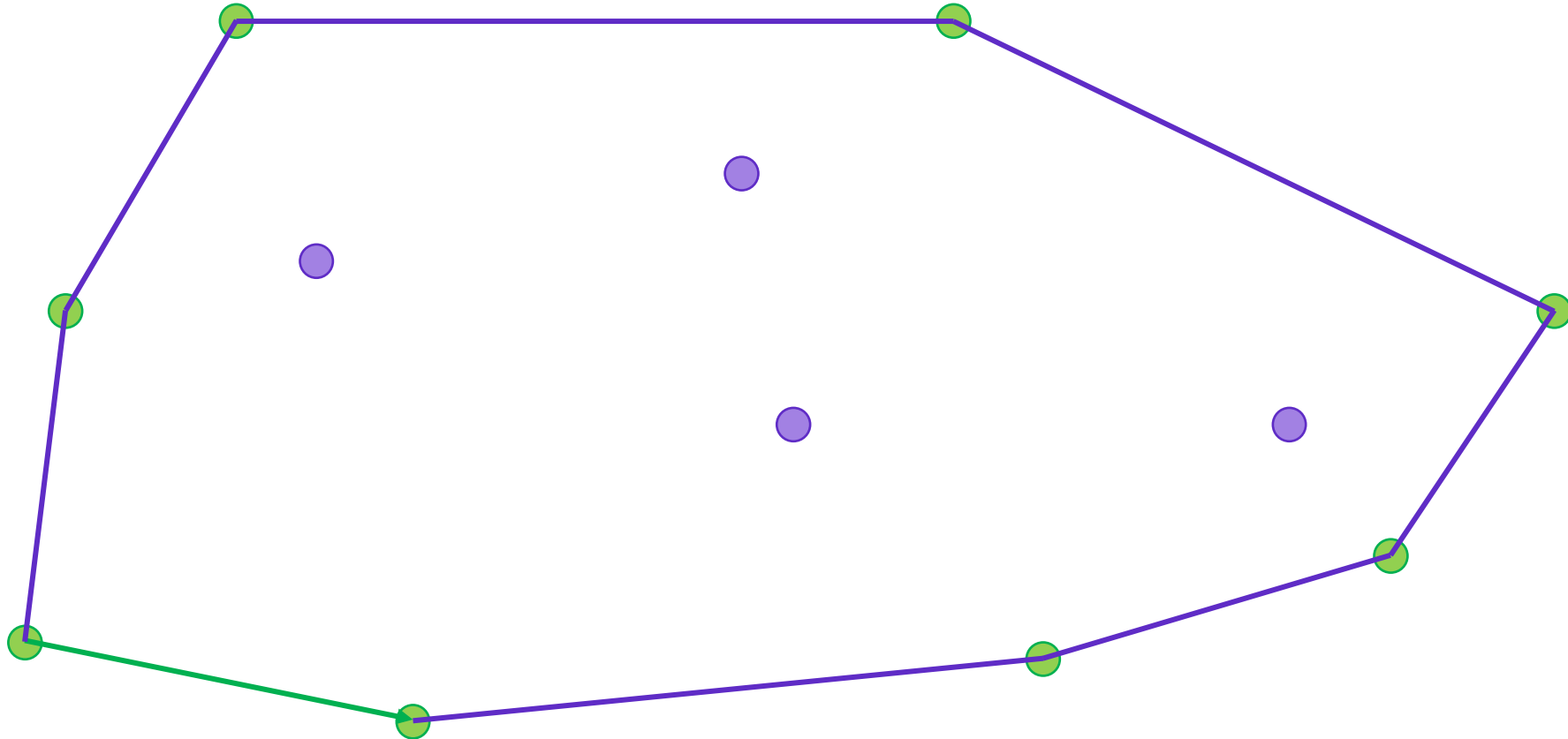
#볼록 껍질

<Convex Hull>




#볼록 껍질

<Convex Hull>



6. 시작점으로 돌아오면 알고리즘을 종료한다.



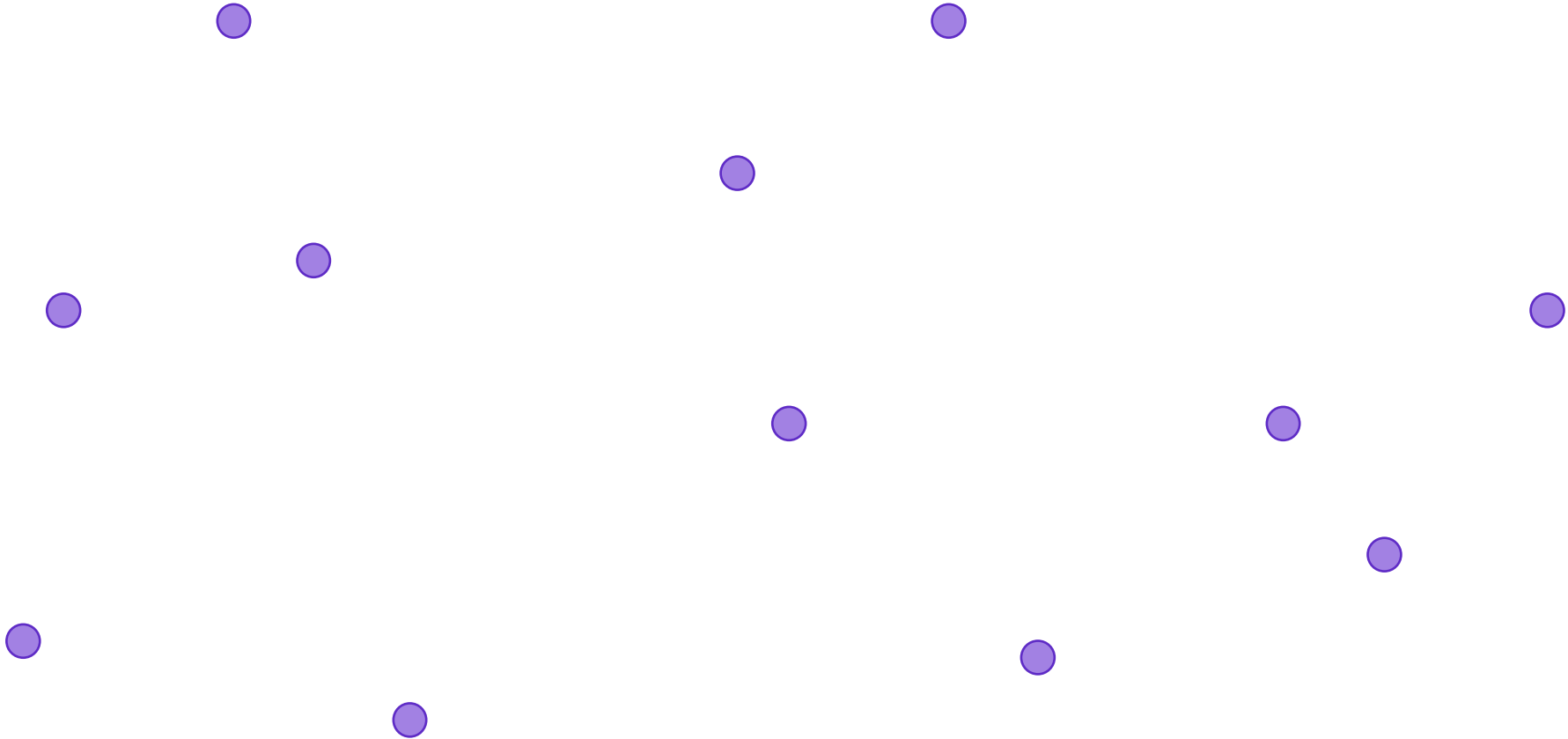
#볼록 껍질

<Convex Hull>

하지만 이대로 구현하기에는 구현할 것도 많고, 귀찮을 것이 분명해 보입니다.
그러니, 조금 다른 방법을 생각해봅시다.

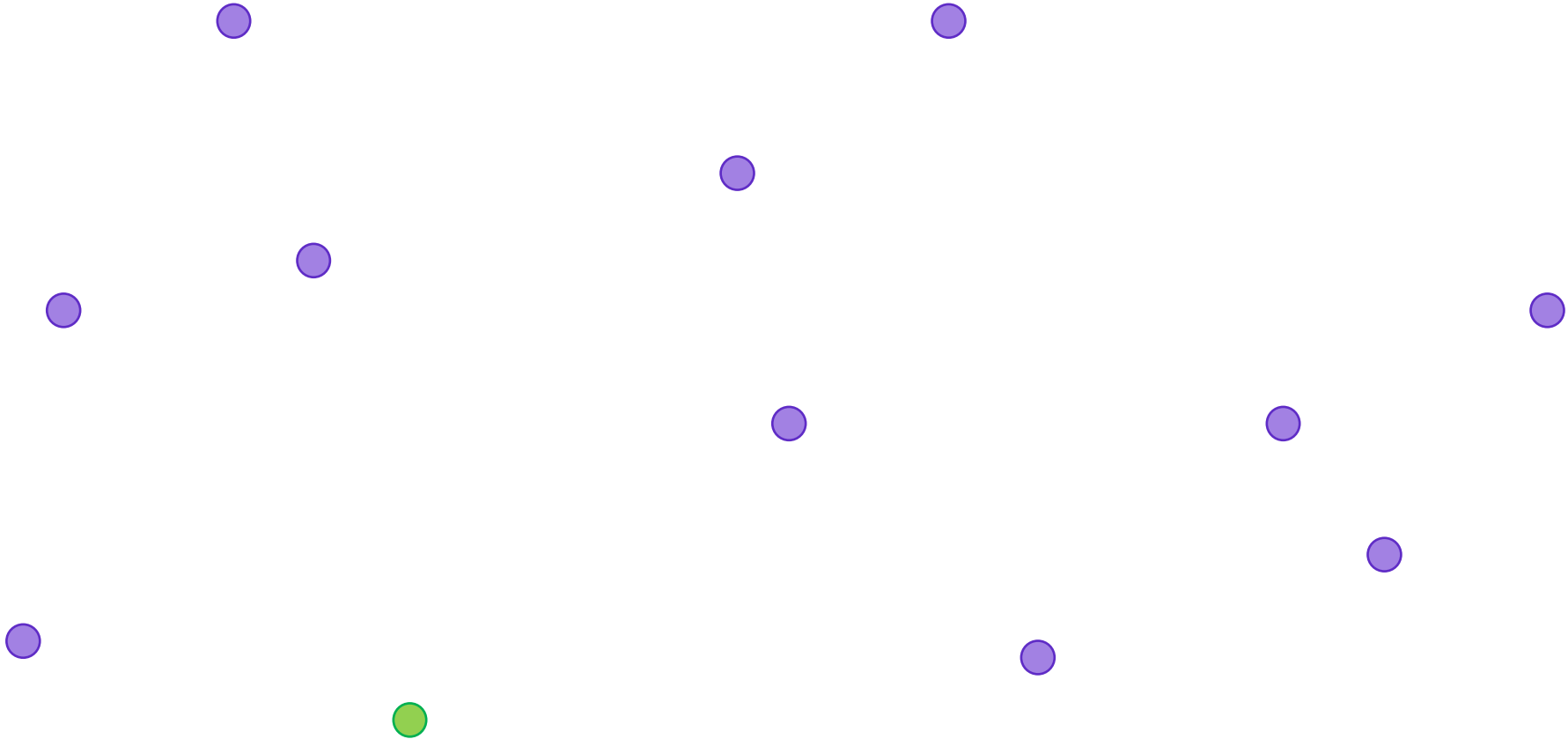
#볼록 껍질

<Convex Hull>



#볼록 껍질

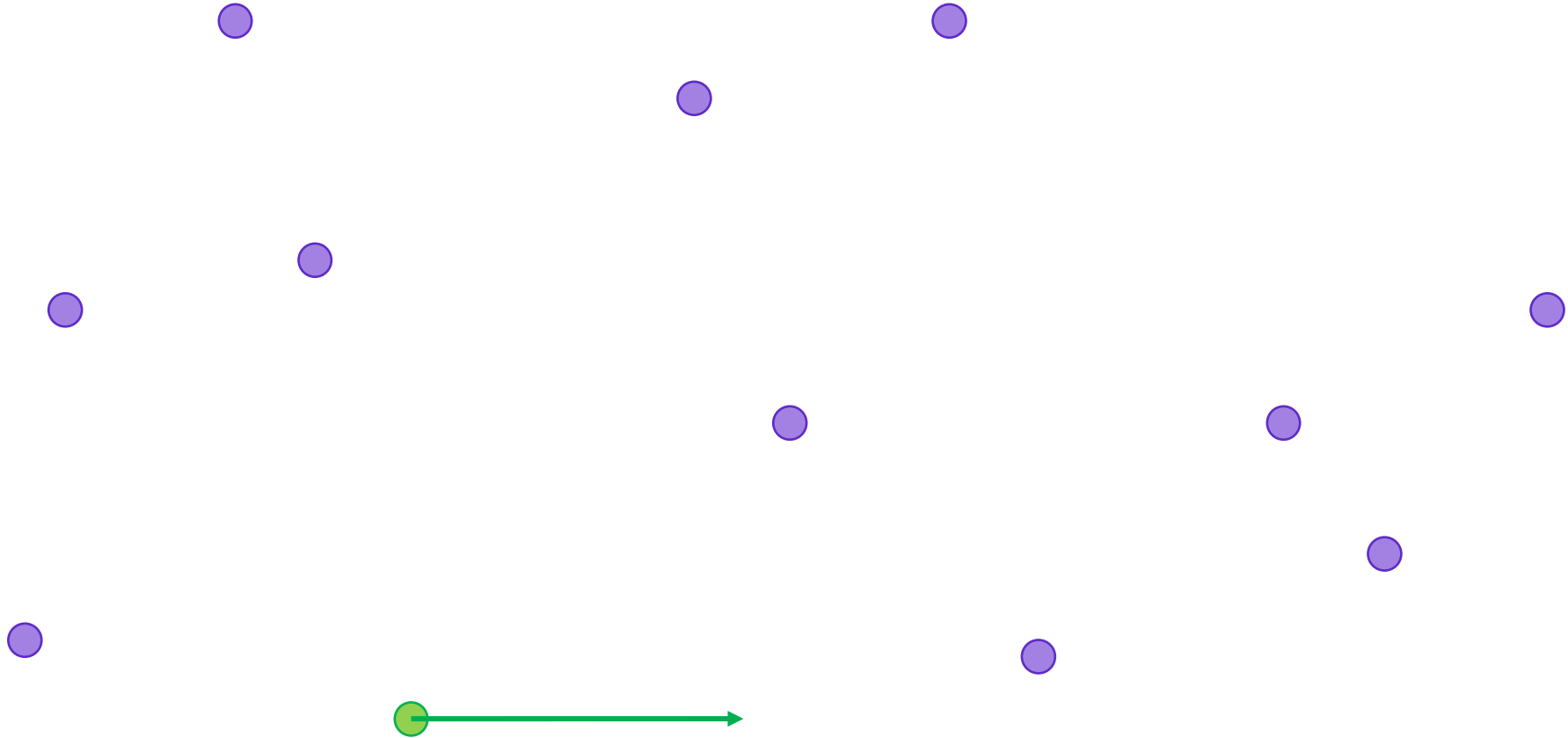
<Convex Hull>



1. 가장 낮은 위치에 있는 점을 찾는다.

#볼록 껍질

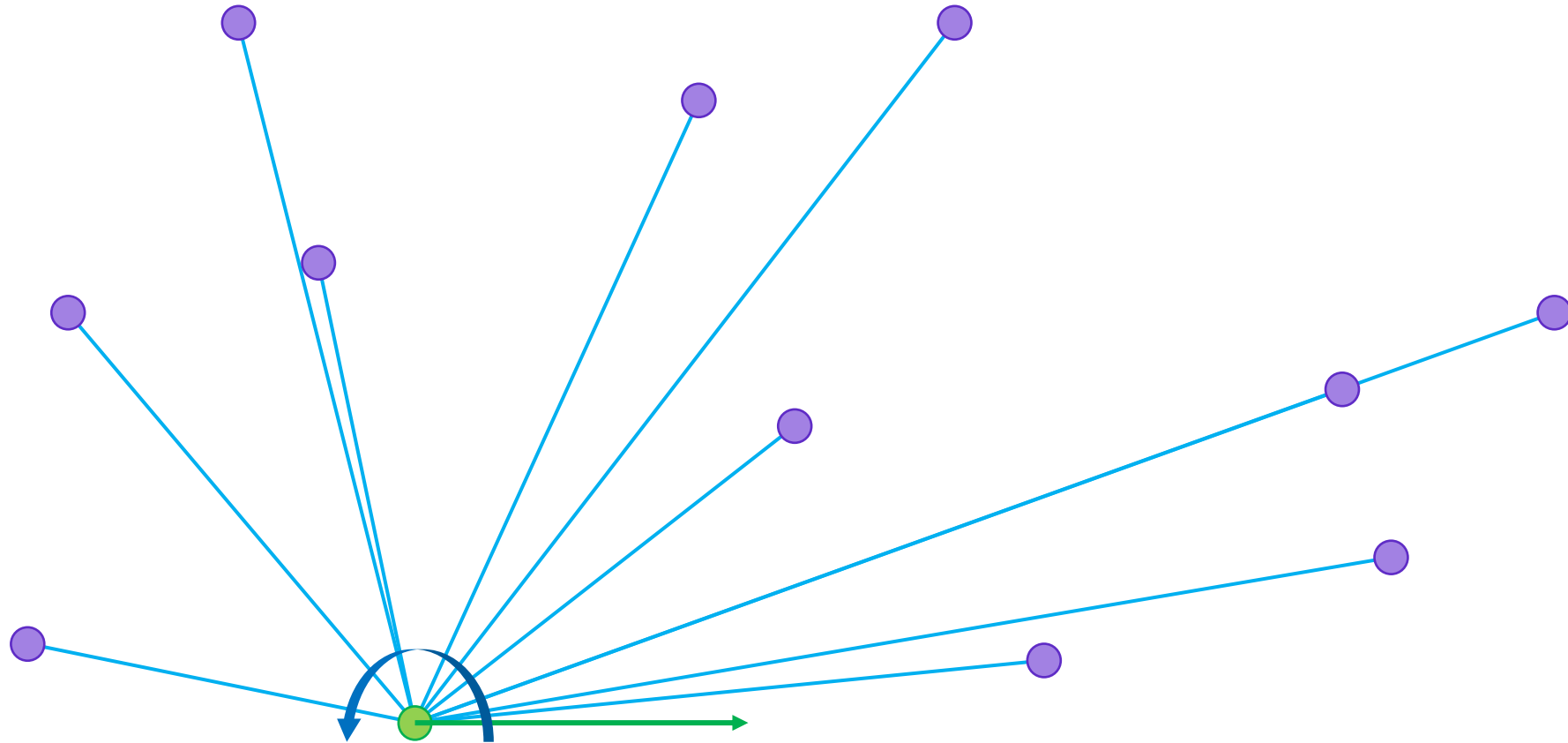
<Convex Hull>



2. x축에 평행한 반직선을 긋는다.

#볼록 껍질

<Convex Hull>

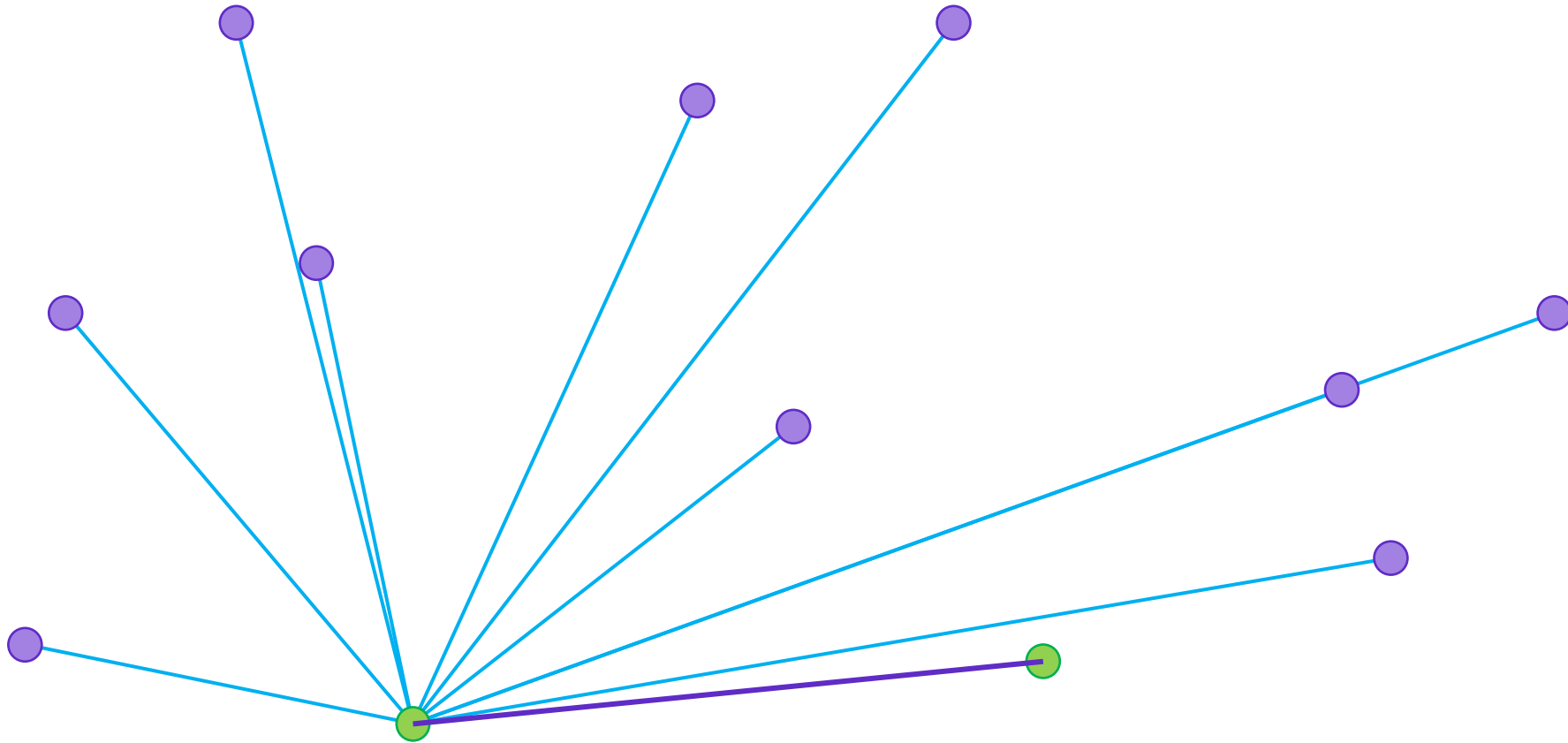


3. 이를 기준으로 반시계방향 각도 정렬을 한다.

같은 각도의 경우, 거리가 짧은 점을 우선한다.

#볼록 껍질

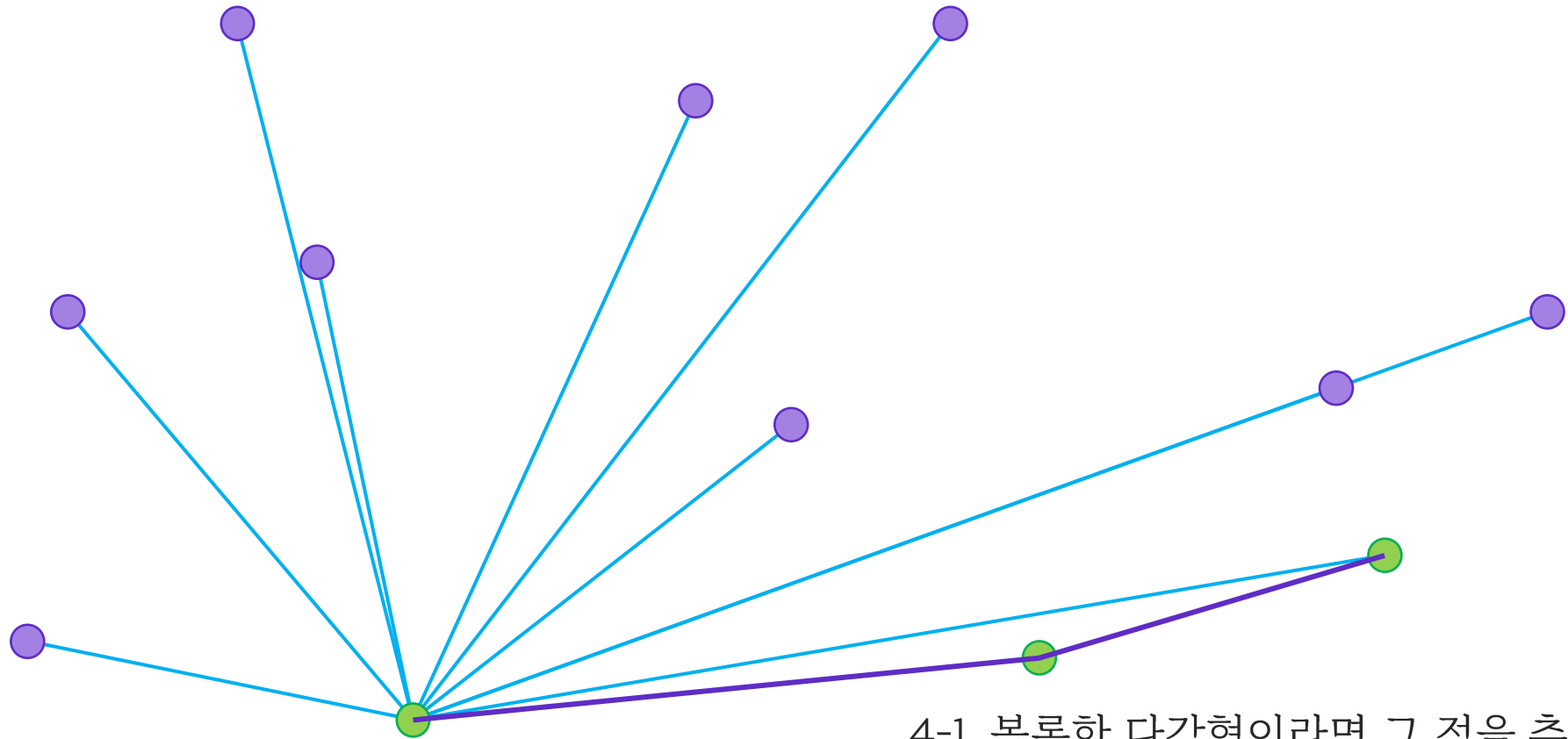
<Convex Hull>



4-1. 정렬된 순서대로 하나씩 보면서, 볼록한 다각형이라면 **그 점을 추가한다.**

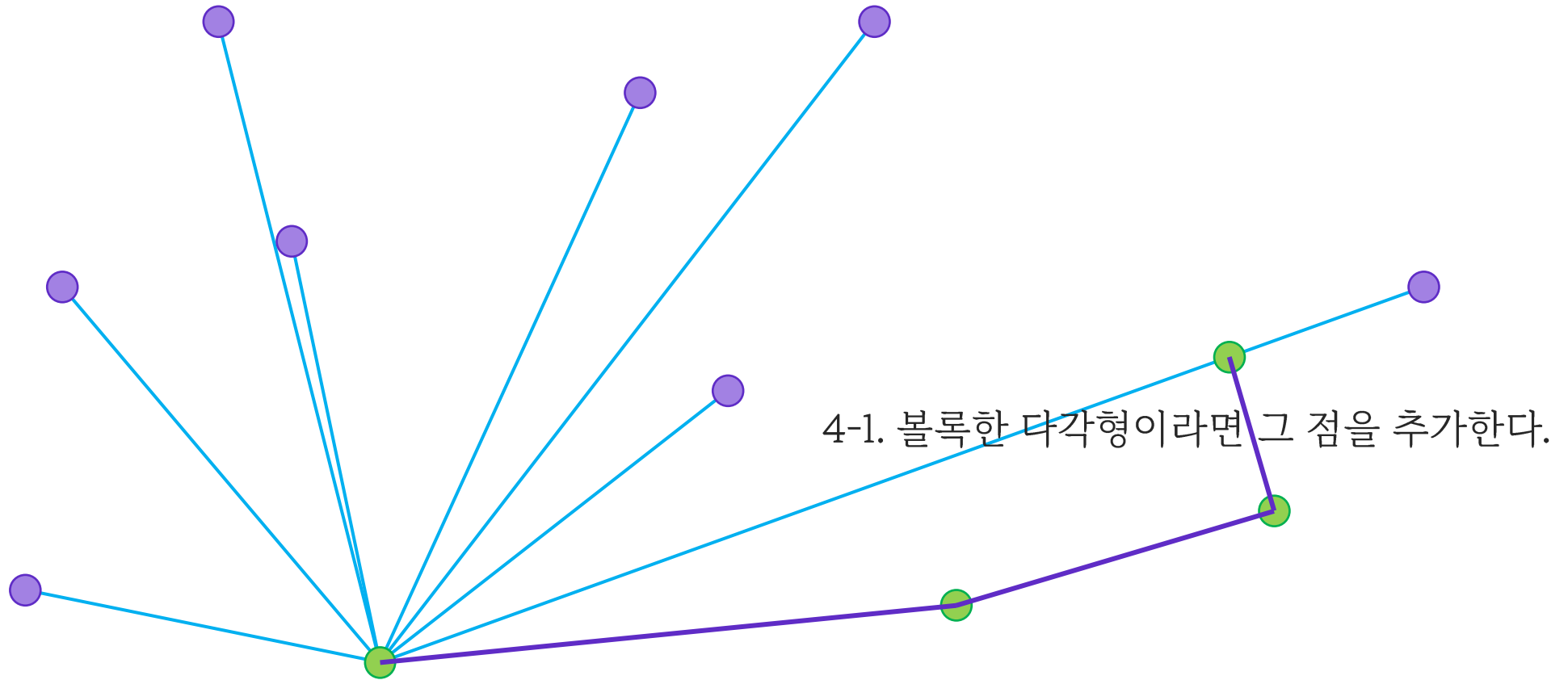
#볼록 껍질

<Convex Hull>



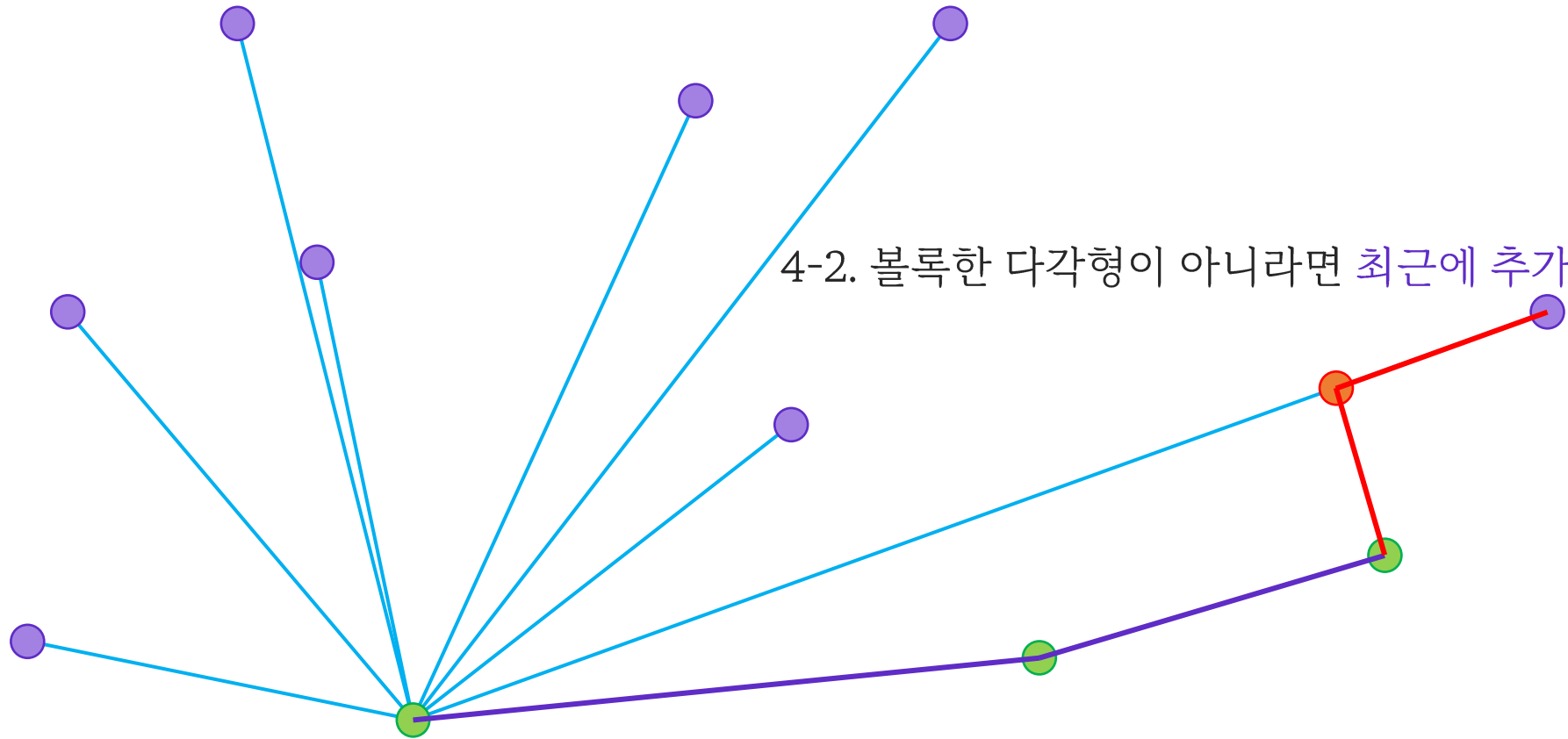
#볼록 껍질

<Convex Hull>



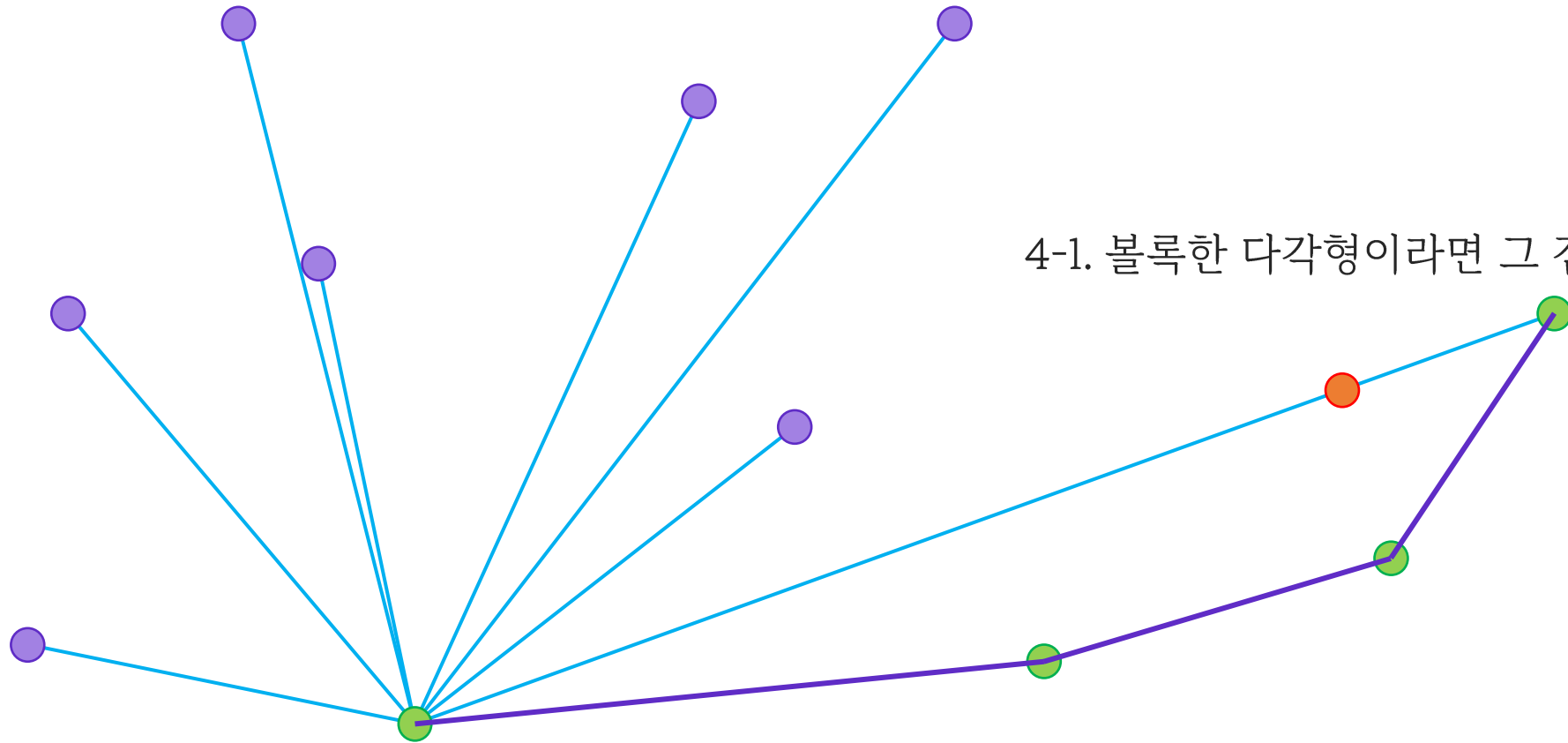
#볼록 껍질

<Convex Hull>



#볼록 껍질

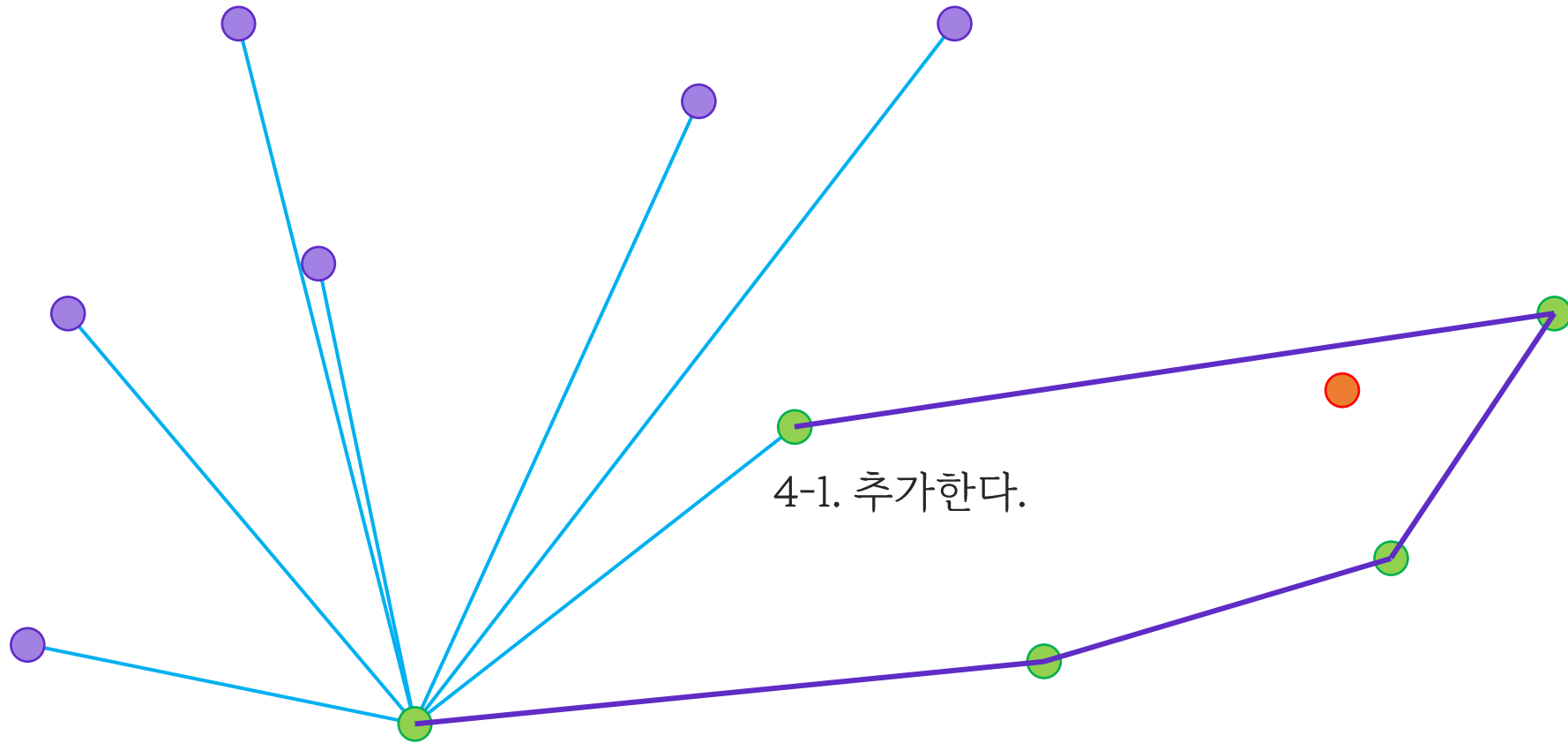
<Convex Hull>



4-1. 볼록한 다각형이라면 그 점을 추가한다.

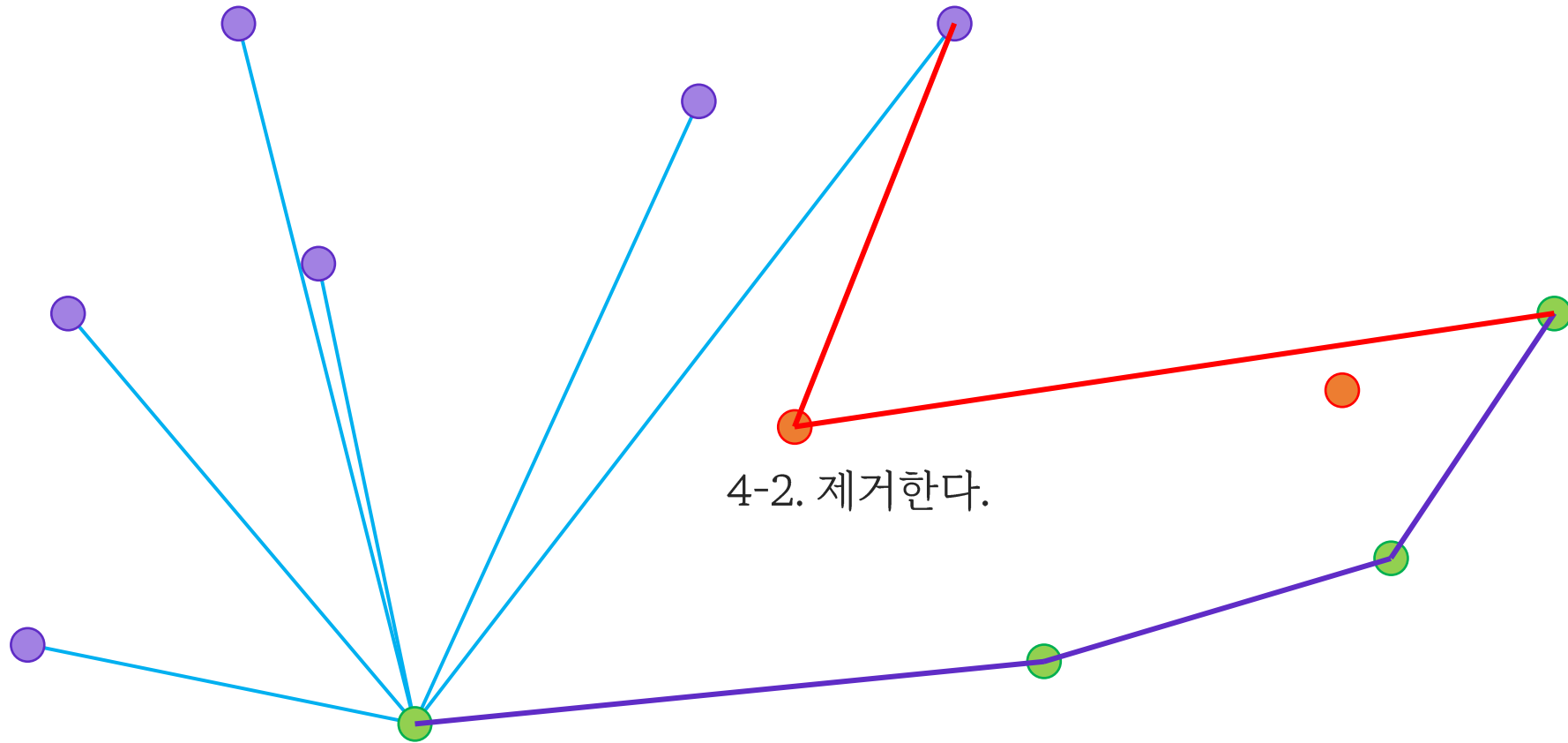
#볼록 껍질

<Convex Hull>



#볼록 껍질

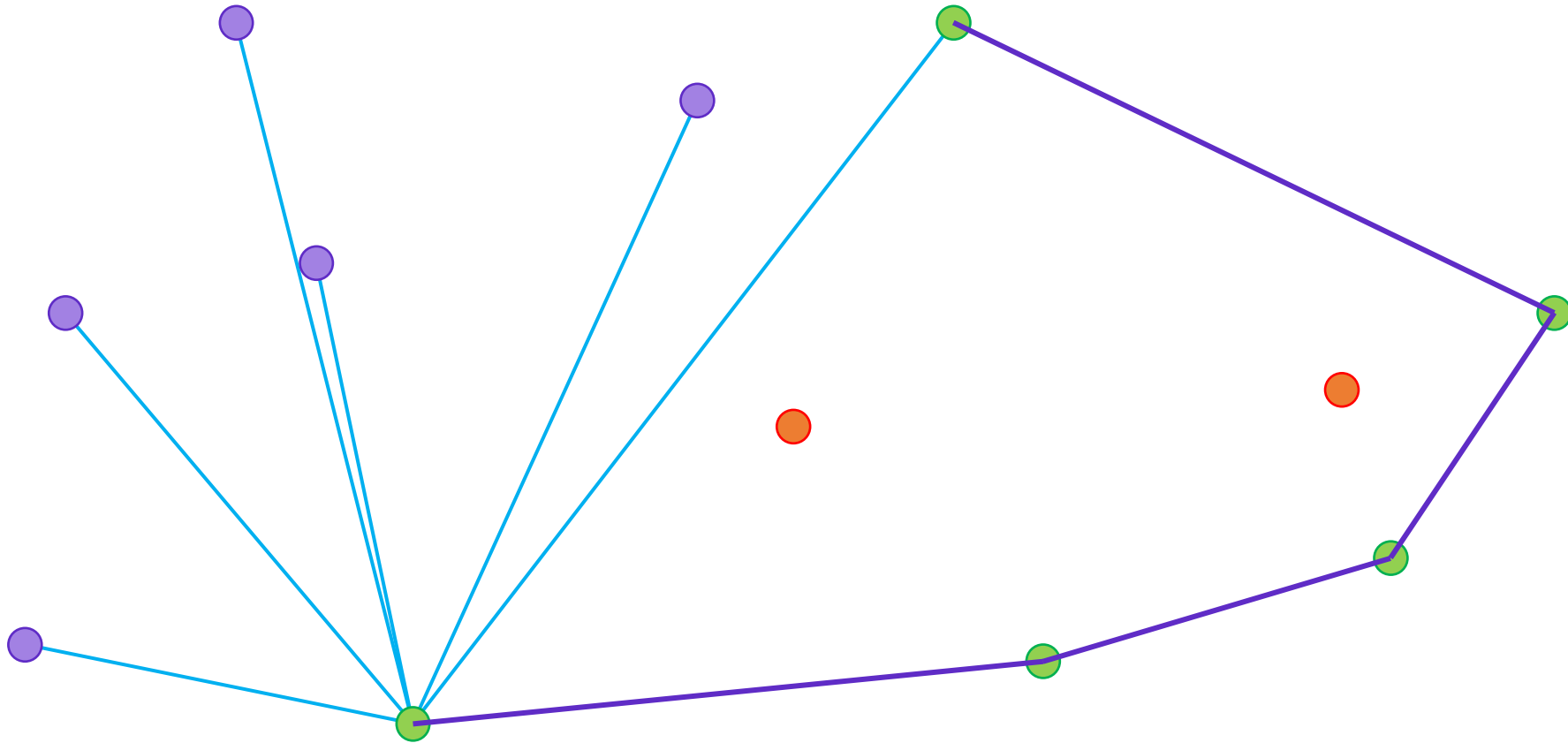
<Convex Hull>



#볼록 껍질

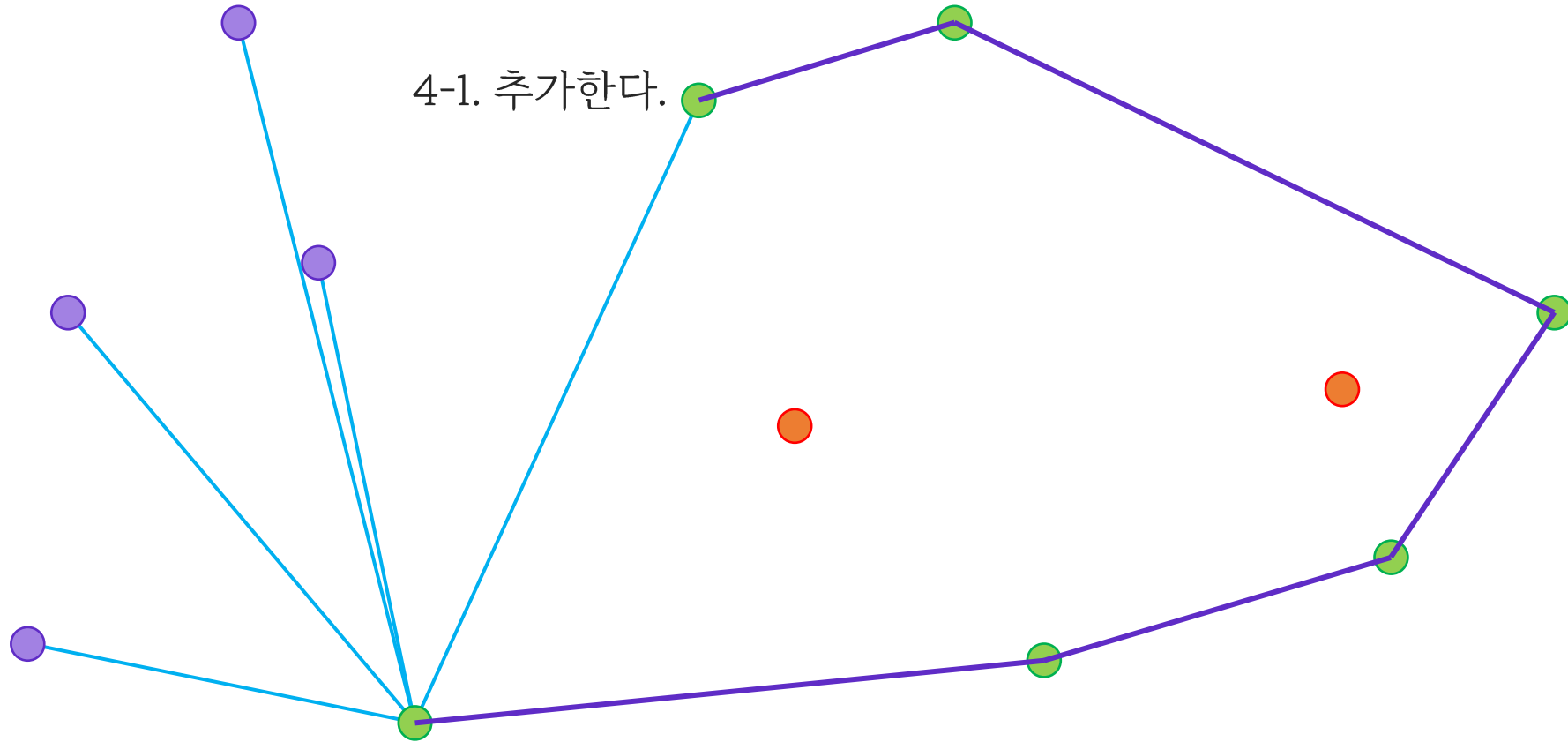
<Convex Hull>

4-1. 추가한다.



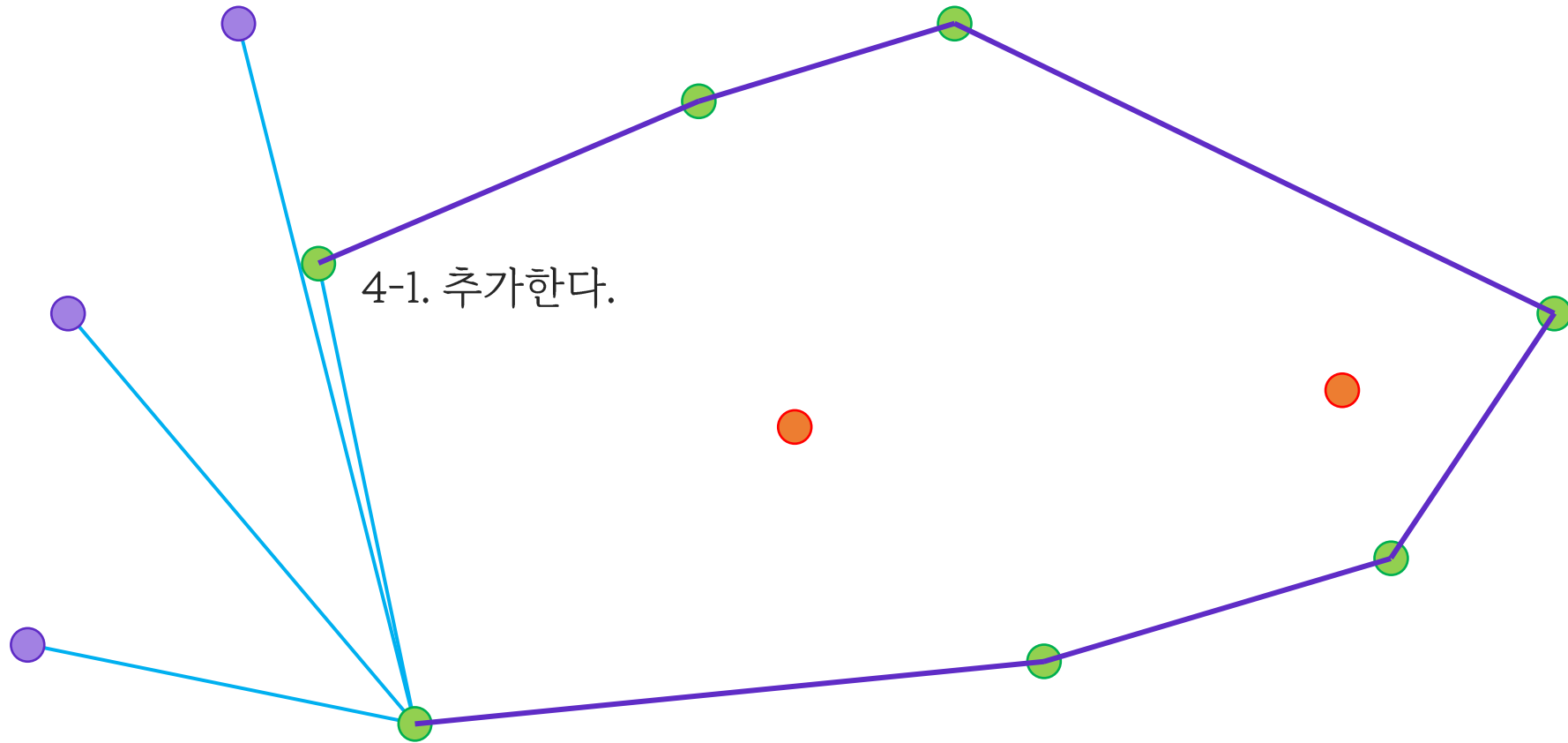
#볼록 껍질

<Convex Hull>



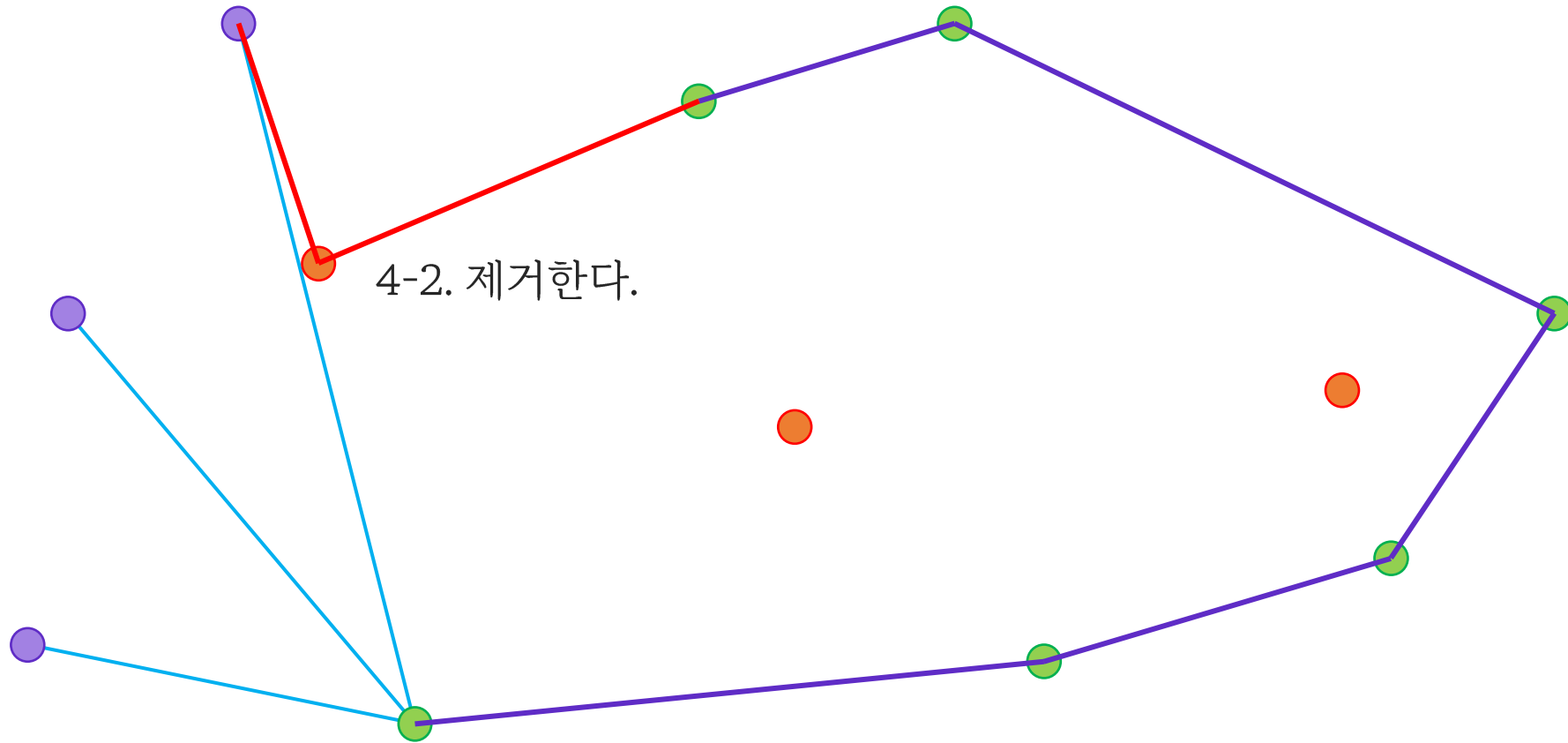
#볼록 껍질

<Convex Hull>



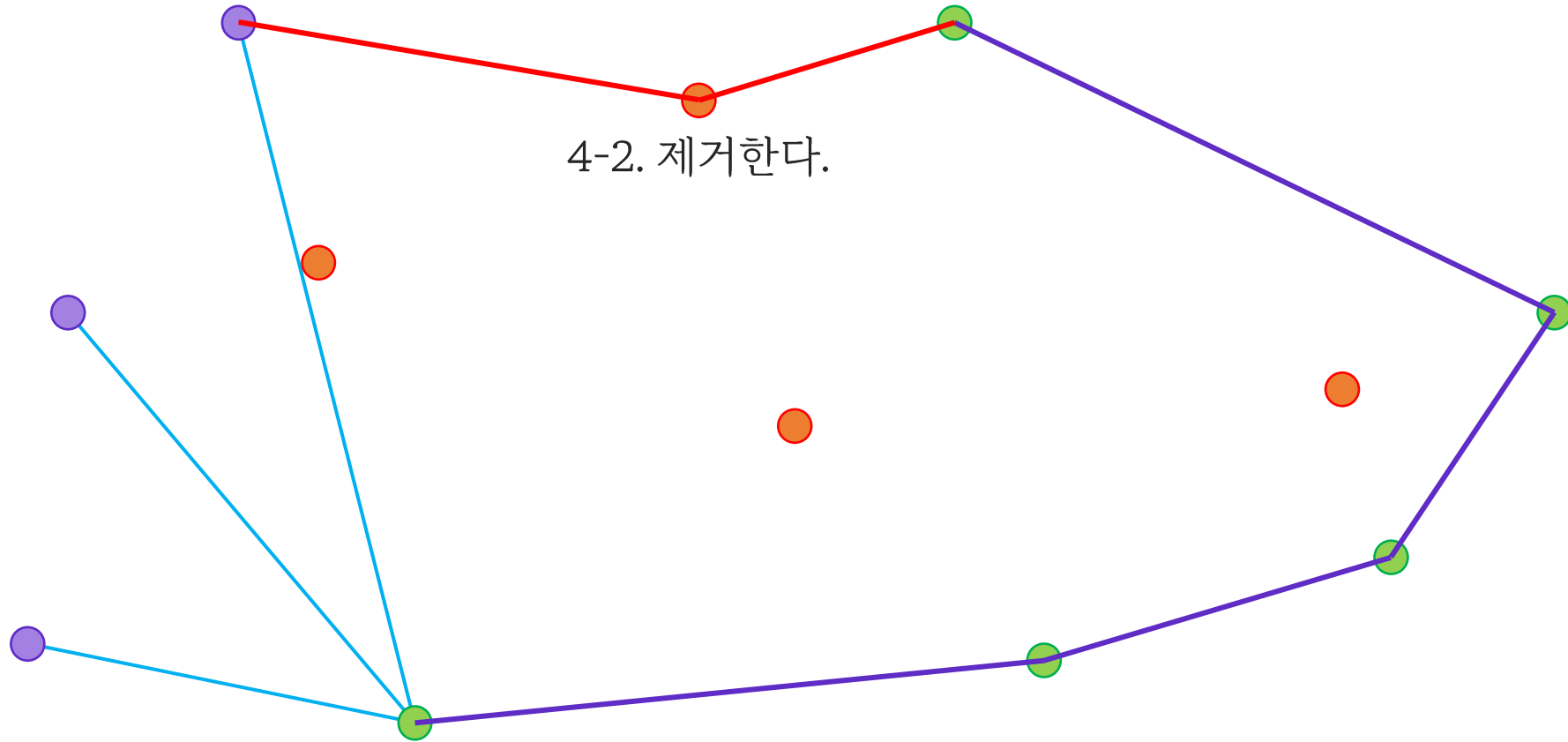
#볼록 껍질

<Convex Hull>



#볼록 껍질

<Convex Hull>

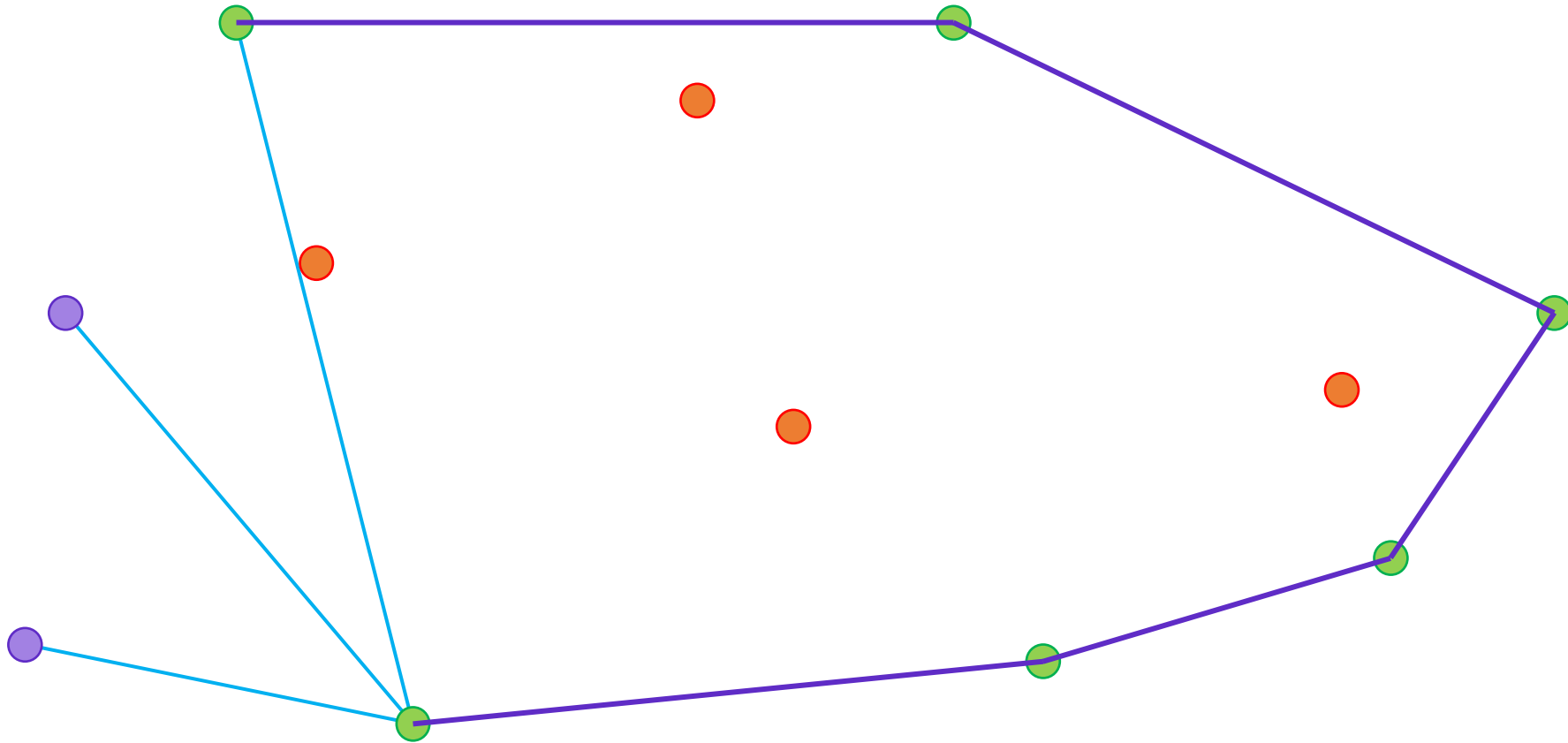


4-2. 제거한다.

#볼록 껍질

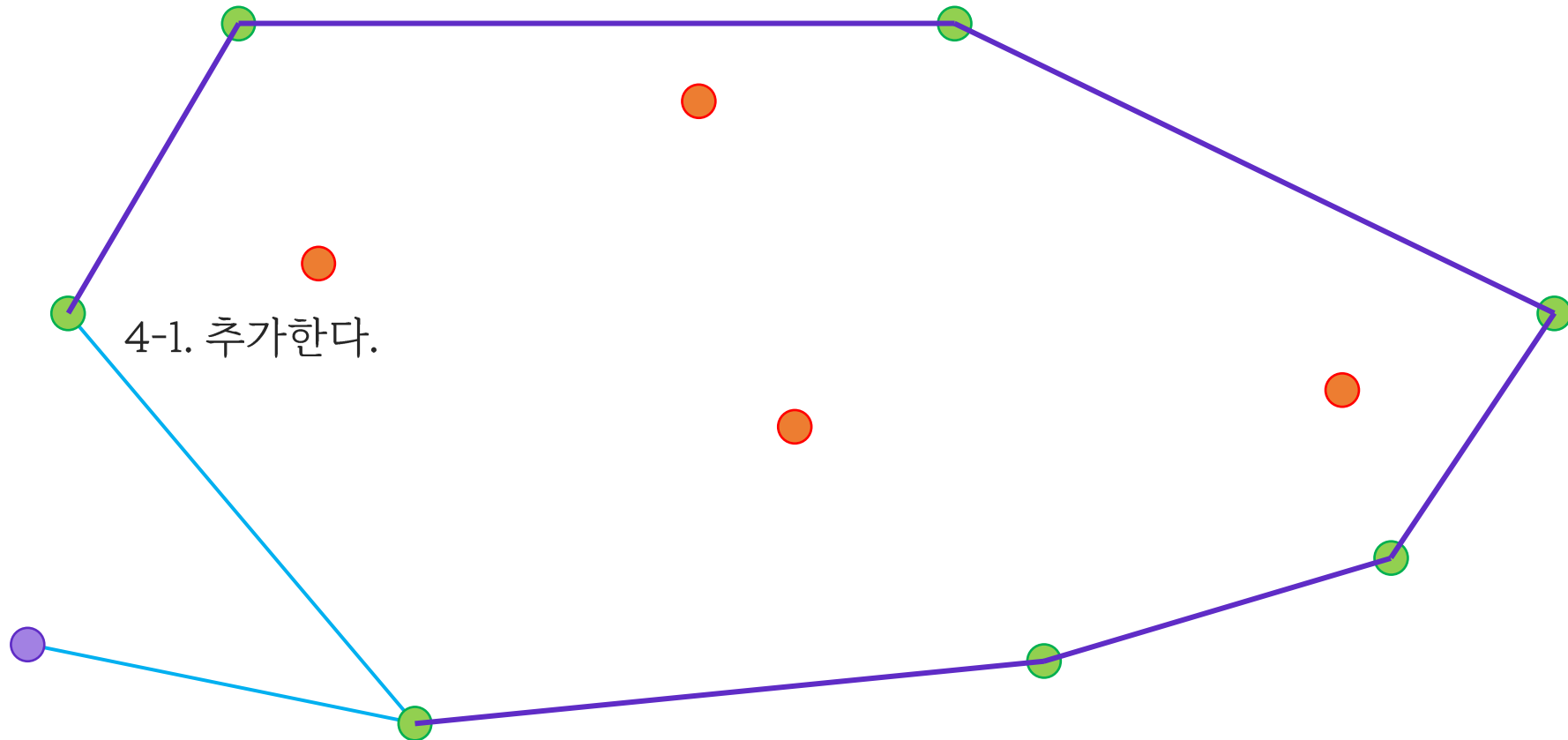
<Convex Hull>

4-1. 추가한다.



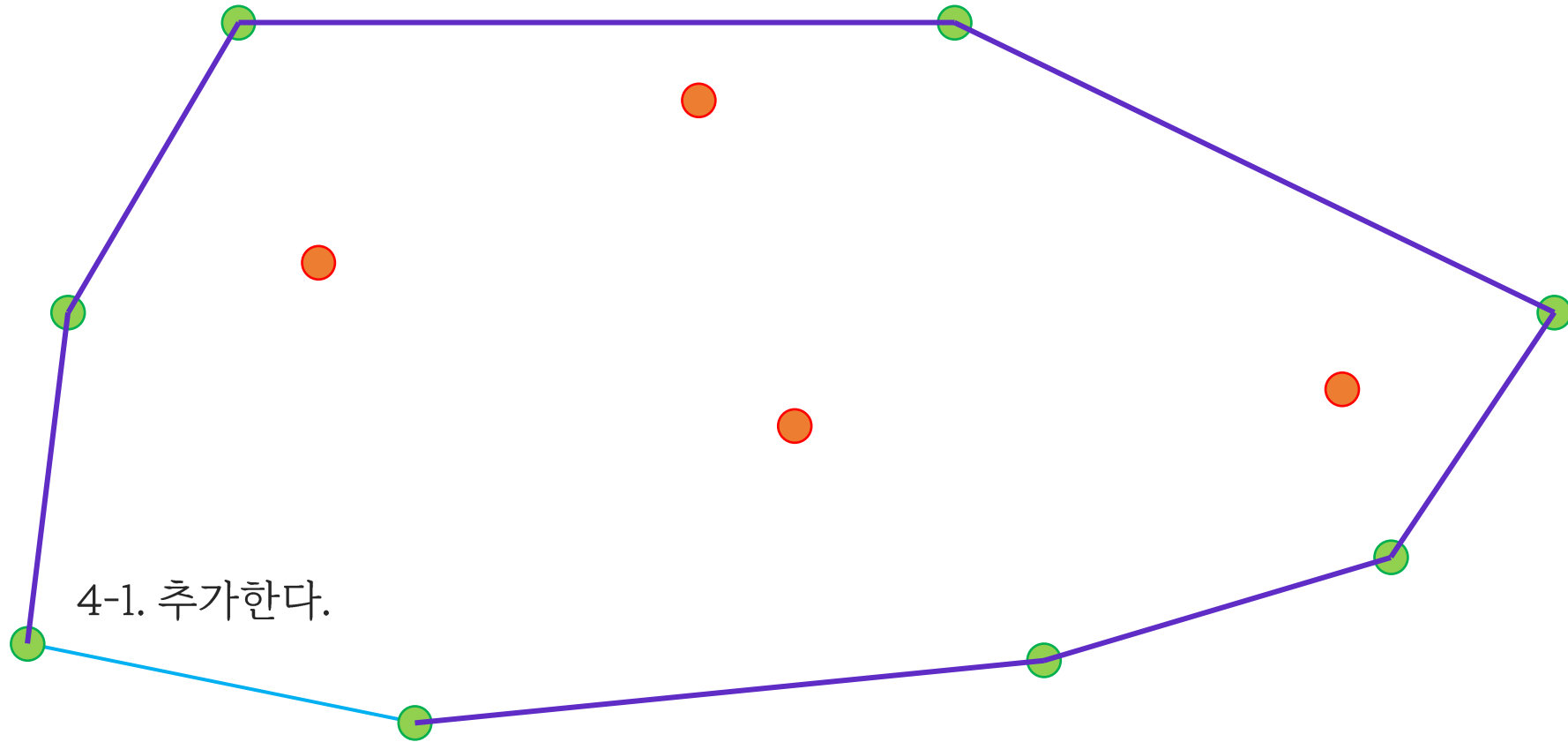
#볼록 껍질

<Convex Hull>



#볼록 껍질

<Convex Hull>





#볼록 껍질

<Convex Hull>

물론 이렇게 해도 질문은 남아있습니다.

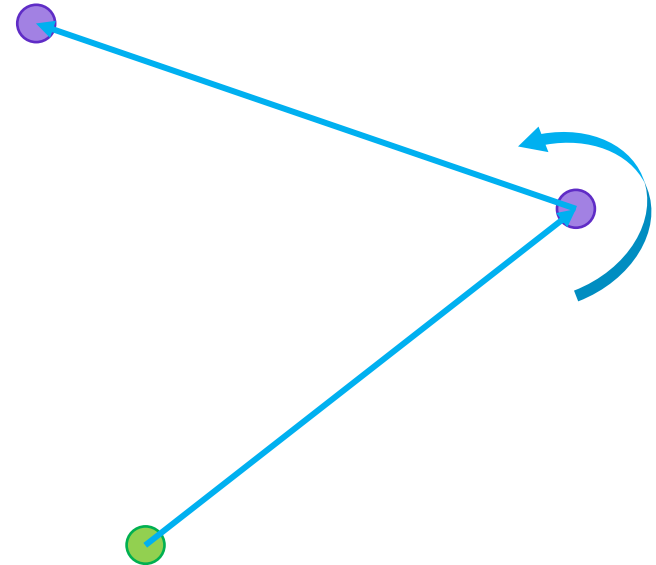
1. 각도 정렬은 어떻게 해요?
2. 다각형이 볼록한 지 어떻게 확인해요?

#볼록 껍질

<Convex Hull>

각도 정렬은 어떻게 하나요?

옆의 그림을 보면 알겠지만, 결국 CCW입니다.

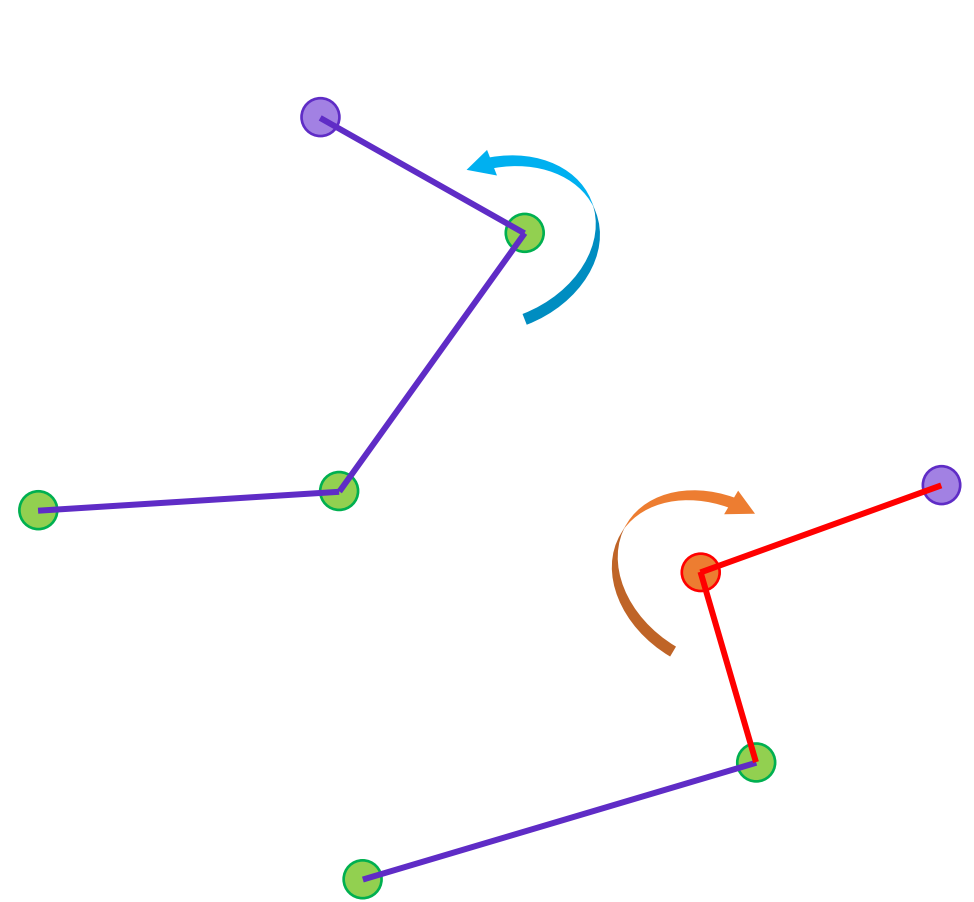


#볼록 껍질

<Convex Hull>

볼록한 지는 어떻게 확인하나요?

옆의 그림을 보면 알겠지만, 결국 CCW입니다.



#볼록 껍질

<Convex Hull>

길어졌지만, 겁먹지 마세요.
아까 본 과정들일 뿐입니다.

```

inline int dis(point2d p1, point2d p2){
    int dy = p1.y - p2.y, dx = p1.x - p2.x;
    return dy*dy + dx*dx;
}

vector<point2d> stk;
inline void psh(point2d p){ stk.push_back(p); }
inline void pop(){ stk.pop_back(); }

void convexHull(vector<point2d> arr){
    for (point2d& p : arr){
        if (p < arr[0]){ swap(p, arr[0]); }
    }
    point2d p0 = arr[0];
    sort(arr.begin(), arr.end(), [p0](point2d p1, point2d p2){
        int res = ccw(p0, p1, p2);
        if (res != 0){ return res > 0; }
        else{ return dis(p0, p1) < dis(p0, p2); }
    });
    for (point2d p : arr){
        while (stk.size() >= 2){
            if (ccw(arr[stk.size()-2], arr[stk.size()-1], p) <= 0){ pop(); }
            else{ break; }
        }
        psh(p);
    }
}

```

#볼록 껍질

<Convex Hull>

우선 가장 밑에 있는 점을 찾고, 0번 인덱스로 옮깁니다.

사실 x가 first라서 가장 왼쪽에 있는 점을 찾고 있긴 합니다.

여담으로 전 평소에 y를 first로 잡아요.

```
inline int dis(point2d p1, point2d p2){
    int dy = p1.y - p2.y, dx = p1.x - p2.x;
    return dy*dy + dx*dx;
}

vector<point2d> stk;
inline void psh(point2d p){ stk.push_back(p); }
inline void pop(){ stk.pop_back(); }

void convexHull(vector<point2d> arr){
    for (point2d& p : arr){
        if (p < arr[0]){ swap(p, arr[0]); }
    }
    point2d p0 = arr[0];
    sort(arr.begin(), arr.end(), [p0](point2d p1, point2d p2){
        int res = ccw(p0, p1, p2);
        if (res != 0){ return res > 0; }
        else{ return dis(p0, p1) < dis(p0, p2); }
    });
    for (point2d p : arr){
        while (stk.size() >= 2){
            if (ccw(arr[stk.size()-2], arr[stk.size()-1], p) <= 0){ pop(); }
            else{ break; }
        }
        psh(p);
    }
}
```


#볼록 껍질

<Convex Hull>

그 뒤로, 맨 밑의 점을 기준으로 정렬합니다.

정렬 과정에서는 아까 본 대로 CCW를 사용합니다.

물론 일직선에 놓였을 때의 처리에 주의하세요!

```
inline int dis(point2d p1, point2d p2){
    int dy = p1.y - p2.y, dx = p1.x - p2.x;
    return dy*dy + dx*dx;
}

vector<point2d> stk;
inline void psh(point2d p){ stk.push_back(p); }
inline void pop(){ stk.pop_back(); }

void convexHull(vector<point2d> arr){
    for (point2d& p : arr){
        if (p < arr[0]){ swap(p, arr[0]); }
    }
    point2d p0 = arr[0];
    sort(arr.begin(), arr.end(), [p0](point2d p1, point2d p2){
        int res = ccw(p0, p1, p2);
        if (res != 0){ return res > 0; }
        else{ return dis(p0, p1) < dis(p0, p2); }
    });
    for (point2d p : arr){
        while (stk.size() >= 2){
            if (ccw(arr[stk.size()-2], arr[stk.size()-1], p) <= 0){ pop(); }
            else{ break; }
        }
        psh(p);
    }
}
```

#볼록 껍질

<Convex Hull>

그 뒤로, 정렬된 순서대로 하나씩 보면서
볼록한 다각형이 만들어지지 않을 때마다 최근
에 넣은 점부터 **하나씩 제거**합니다.

최근에 넣은 것부터 제거하기 때문에 stack을 사용합니다.


물론 인덱스 접근도 필요하므로 vector로 구현했습니다.

볼록한 다각형이 아닐 때를 판별하는 것 역시 CCW로 구현됩니다.

```
inline int dis(point2d p1, point2d p2){
    int dy = p1.y - p2.y, dx = p1.x - p2.x;
    return dy*dy + dx*dx;
}

vector<point2d> stk;
inline void psh(point2d p){ stk.push_back(p); }
inline void pop(){ stk.pop_back(); }

void convexHull(vector<point2d> arr){
    for (point2d& p : arr){
        if (p < arr[0]){ swap(p, arr[0]); }
    }
    point2d p0 = arr[0];
    sort(arr.begin(), arr.end(), [p0](point2d p1, point2d p2){
        int res = ccw(p0, p1, p2);
        if (res != 0){ return res > 0; }
        else{ return dis(p0, p1) < dis(p0, p2); }
    });
    for (point2d p : arr){
        while (stk.size() >= 2){
            if (ccw(arr[stk.size()-2], arr[stk.size()-1], p) <= 0){ pop(); }
            else{ break; }
        }
        psh(p);
    }
}
```



#볼록 껍질

<Convex Hull>

볼록 껍질 (BOJ #1708)

<문제 설명>

- N 개의 점이 주어진다.
- N 개의 점을 모두 포함하는 볼록 껍질을 찾으려 한다.

<제약 조건>

- $3 \leq N \leq 100\,000$
- $-40\,000 \leq \text{좌표} \leq 40\,000$
- 모든 좌표는 정수이다.

#볼록 껍질

<Convex Hull>

5 볼록 껍질 (BOJ #1708)

<문제 해설>

방금 배운 볼록 껍질을 그대로 구현해주면 됩니다.

#연습 문제 도전

1 N수매화검법 (BOJ #25315)

UCPC 기출

1 울타리 넘기 (BOJ #2144)

울타리를 넘어봅시다.

5 Wall Construction (BOJ #10903)

헉 아까 문제에 소수점이라니

1 나는 가르친다 스위핑을 (BOJ #14266)

나는 낸다 문제를

5 맹독 방벽 (BOJ #7420)

헉 원이라니

5 선분 교차 EX (BOJ #27718)

The Ultimate Line Intersection Problem (TULIP for short)

#연습 문제 도전

4 민코프스키 합 (BOJ #2244)

볼록 다각형 + 볼록 다각형

3 Prison Break (BOJ #24458)

작년 기하 강의자가 추천해줬던 문제

2 Paris by Night (BOJ #17559)

ICPC 기출 2

4 Flatland Olympics (BOJ #23772)

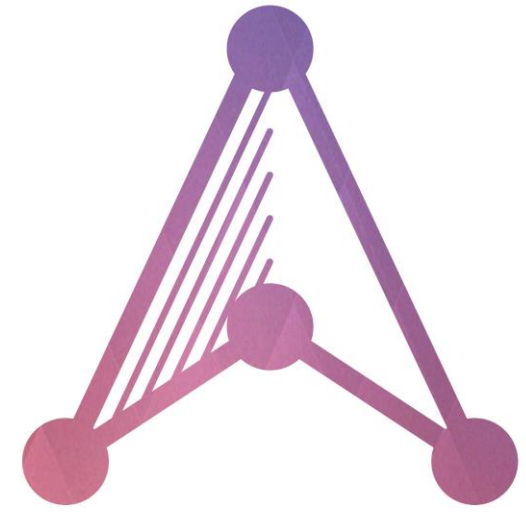
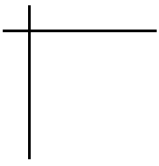
ICPC 기출

2 점 분리 (BOJ #3878)

지문에 있는 그림 왜 AI 배우면서 보게 될 거 같은 그림이지

5 보물 찾기 (BOJ #26160)

HCPC 기출. 볼록 다각형 내부 점 판별에 대해서도 배워보세요!



A L O H A
The algorithm club.

