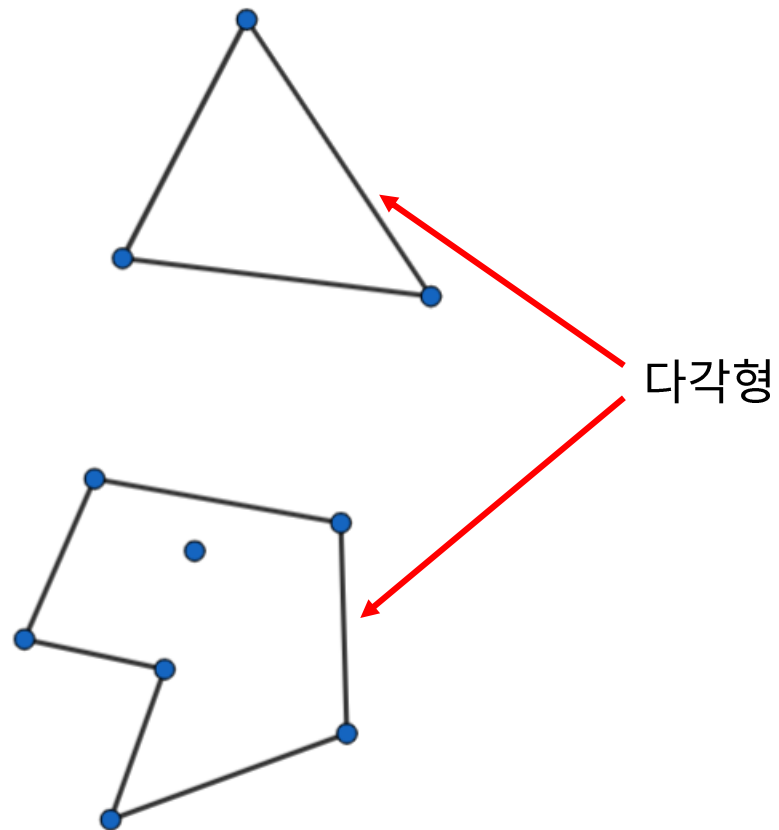
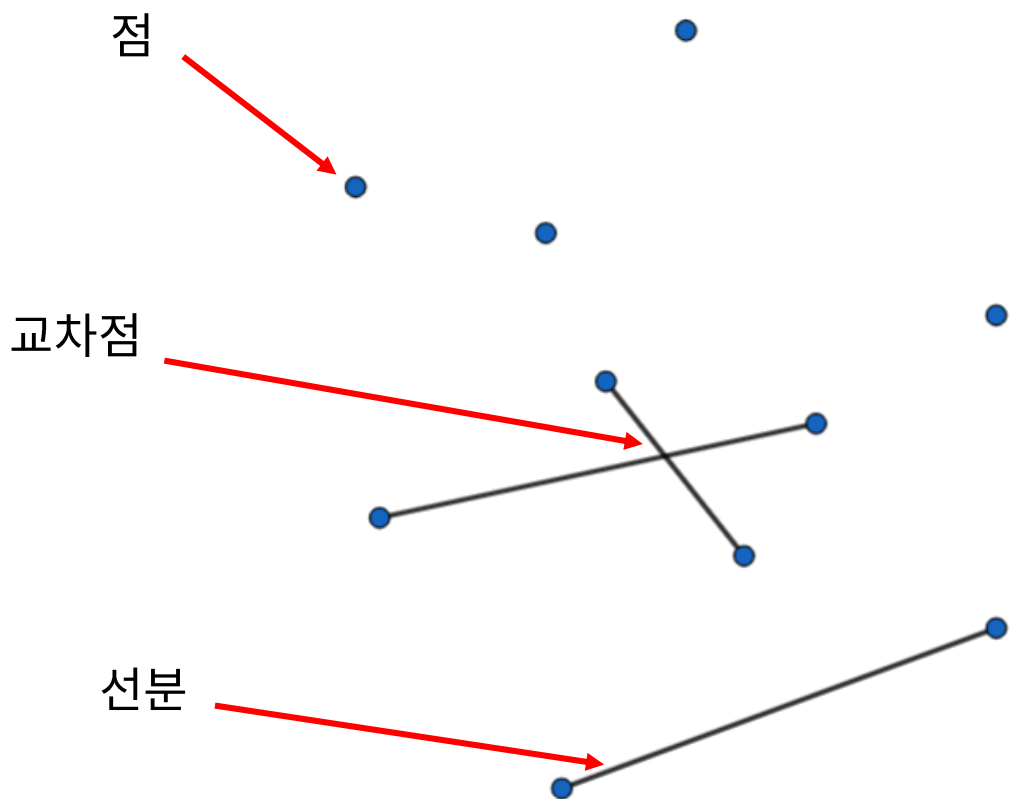


Geometry

2020 winter 중급

20141284 이기현

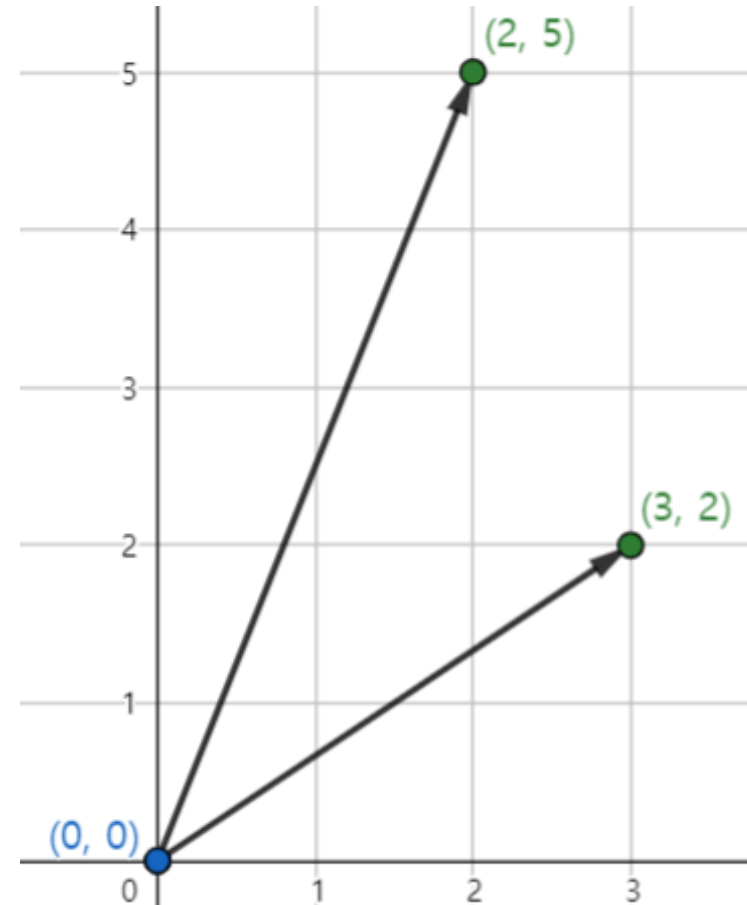






벡터 (vector)

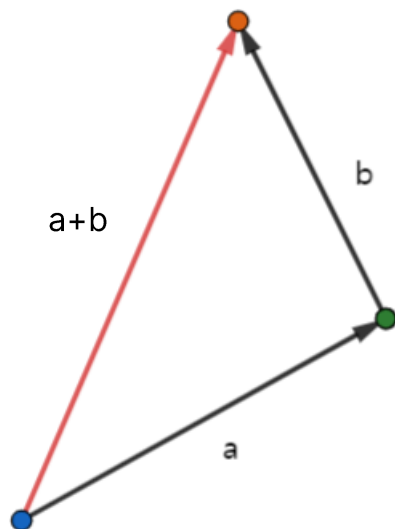
- 방향(시점 / 종점), 크기를 갖는다.
- 기하를 다루면서 점, 선 등을 표현할 기본 도구



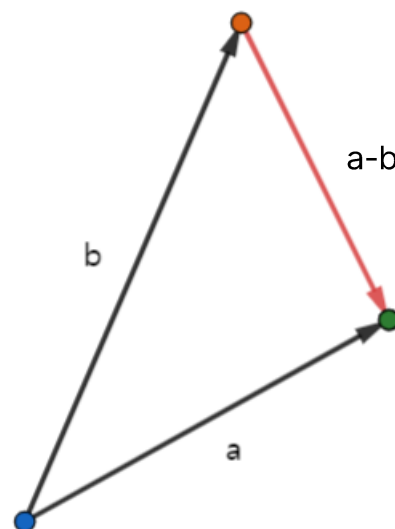


벡터 연산

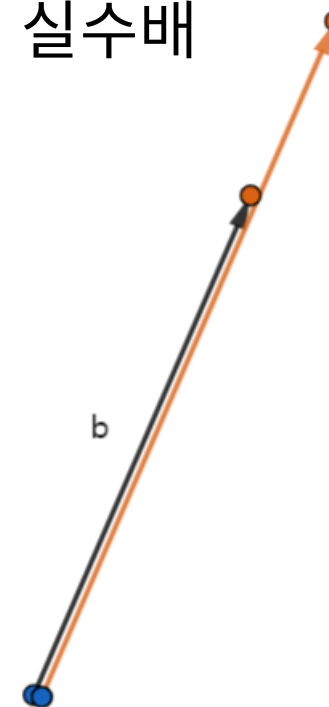
벡터의 합



벡터의 차



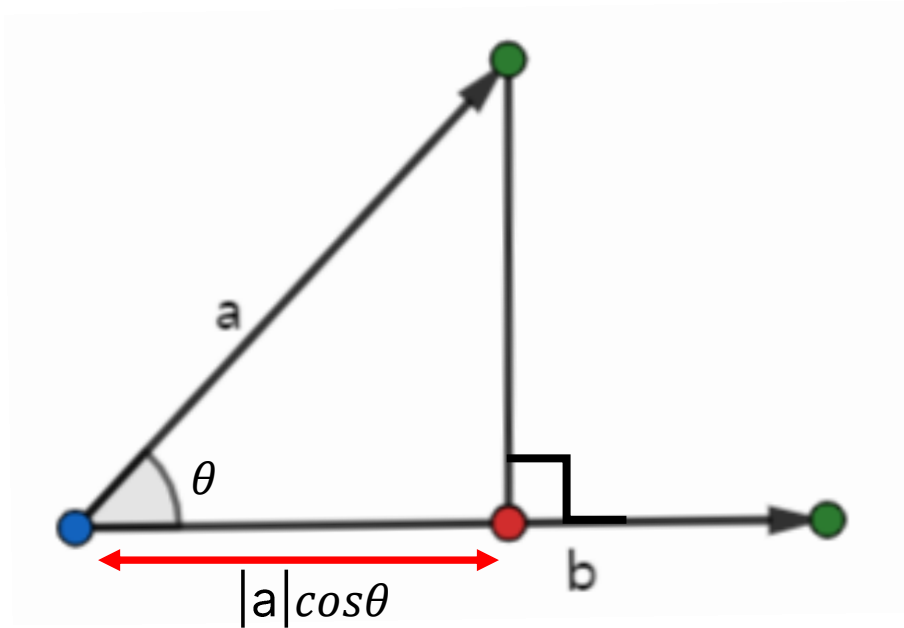
벡터의 실수배





벡터 연산

벡터의 내적

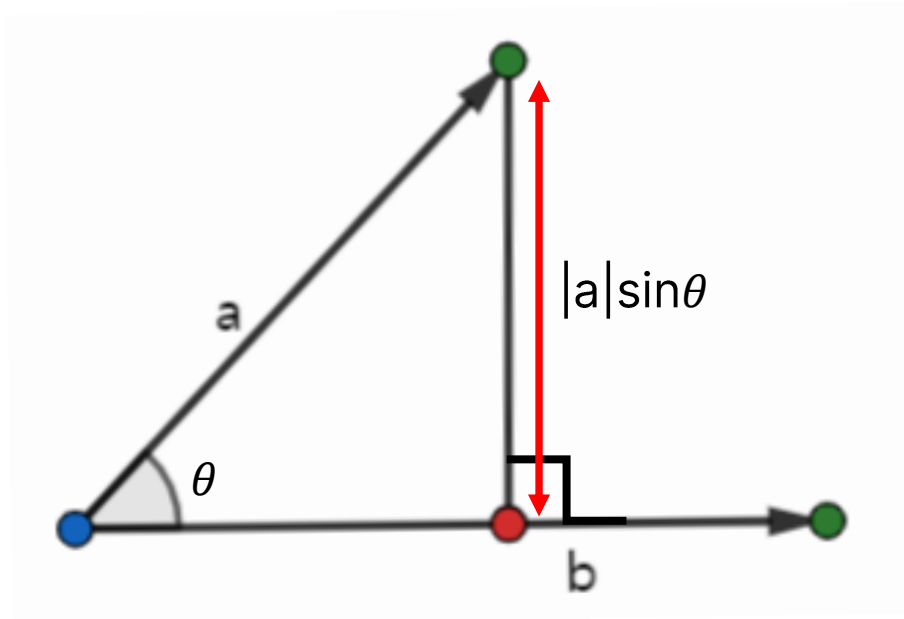


- $\vec{a} \cdot \vec{b} = a_x * b_x + a_y * b_y$
 $= |\vec{a}| * |\vec{b}| \cos \theta$
- 사이각 구하기
acos 함수 이용하여 θ 구하기
직각 판별 가능
- 사영 벡터 구하기
b 벡터 방향, $|a| \cos \theta$ 크기를 갖는 벡터



벡터 연산

벡터의 외적

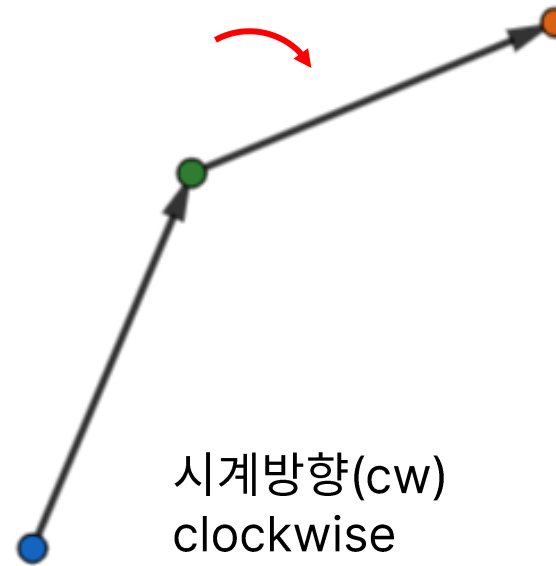
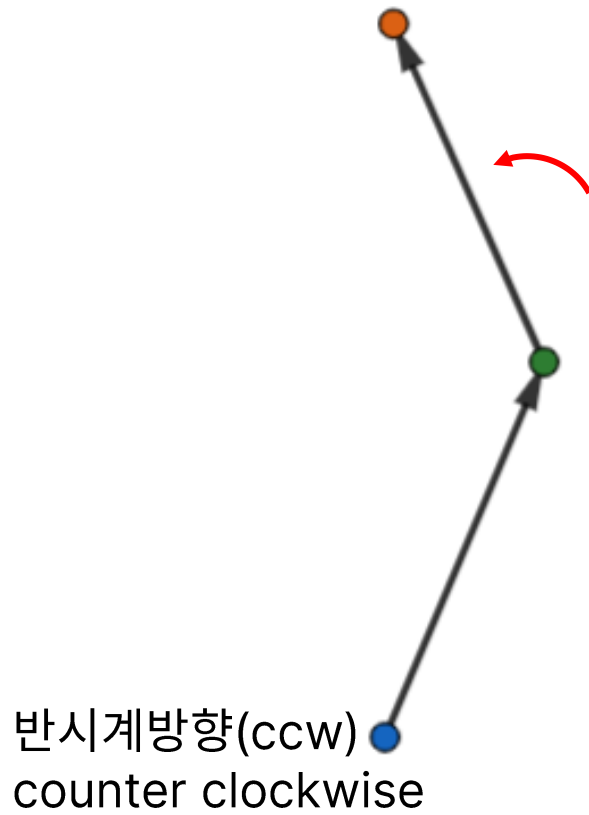


- $|\vec{a} \times \vec{b}| = a_x * b_y - a_y * b_x$
 $= |\vec{a}| * |\vec{b}| \sin \theta$
- 넓이 구하기
두 벡터가 만드는 삼각형/평행사변형의
넓이 구하기 가능
- 두 벡터의 방향 판별
오른손 법칙

CCW

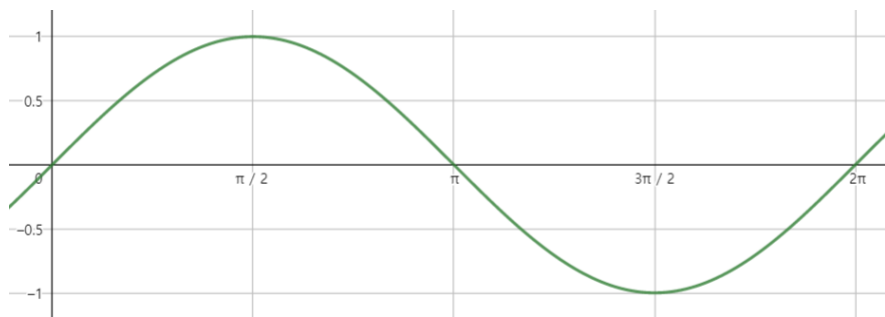
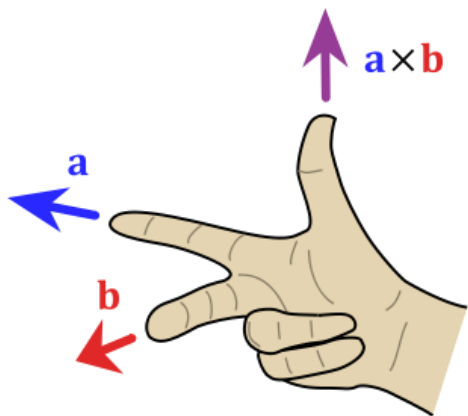


세 점의 방향성





세 점의 방향성 – with 외적



벡터의 외적은 3차원벡터에서 정의
→ 2차원벡터의 z좌표를 0으로 설정해서 확장

$$\vec{a} : (a_x, a_y, 0), \vec{b} : (b_x, b_y, 0)$$

$$\begin{aligned}\vec{a} \times \vec{b} &= (0, 0, a_x * b_y - a_y * b_x) \\ &= \hat{n} |\vec{a}| * |\vec{b}| \sin\theta \quad (\hat{n} : (0, 0, 1))\end{aligned}$$

θ 가 180도 미만이면 $\sin\theta$ 가 양수

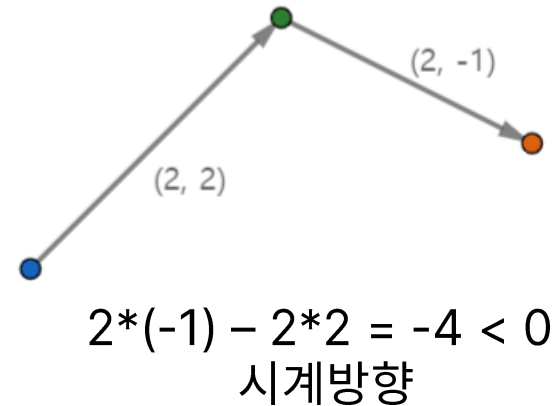
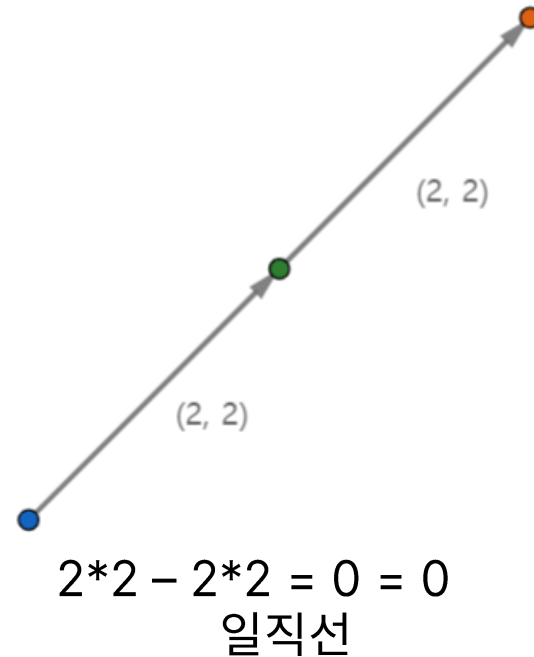
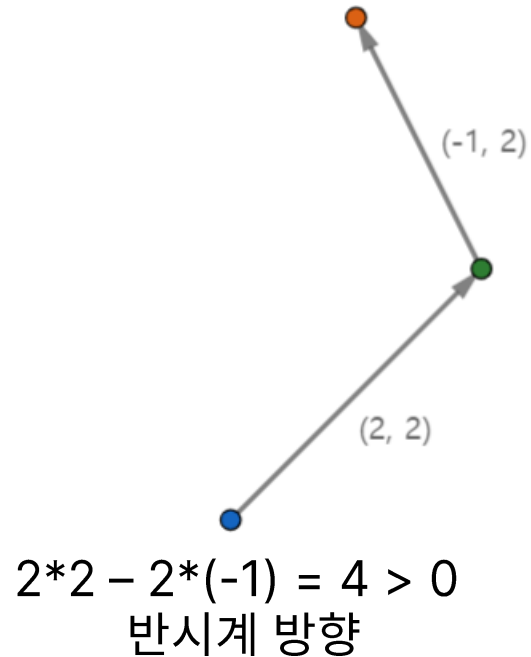
θ 가 180도면 $\sin\theta$ 가 0

θ 가 180도 초과면 $\sin\theta$ 가 음수



세 점의 방향성 - with 외적

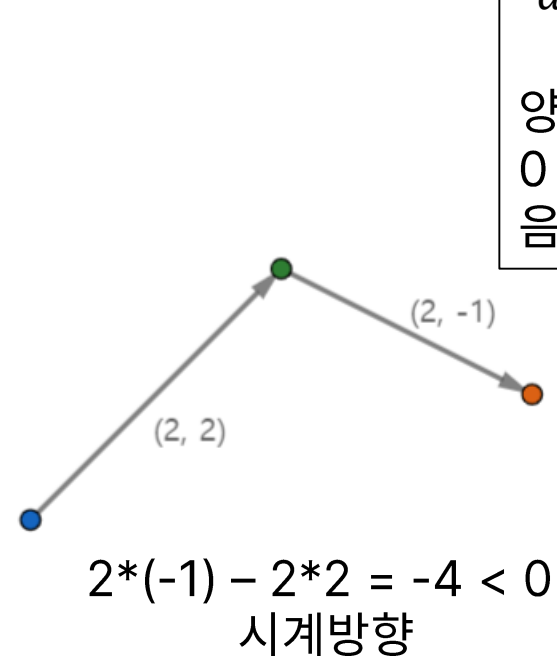
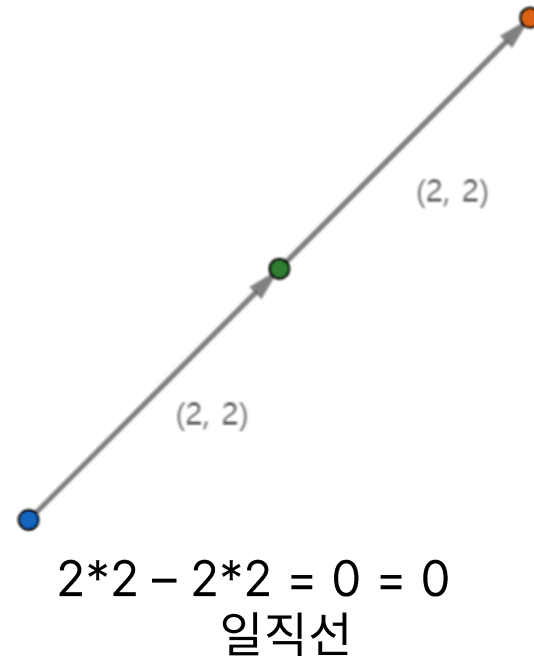
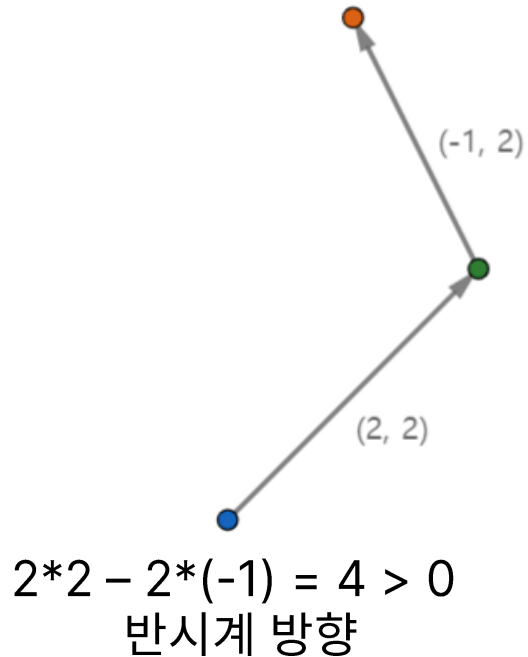
- $a_x * b_y - a_y * b_x$ 값의 부호를 통해 두 벡터의 방향을 알 수 있음
(= 세 점의 방향성을 알 수 있음)





세 점의 방향성 - with 외적

- $a_x * b_y - a_y * b_x$ 값의 부호를 통해 두 벡터의 방향을 알 수 있음
(= 세 점의 방향성을 알 수 있음)



$$a_x * b_y - a_y * b_x$$

양수 → CCW
0 → 일직선
음수 → CW



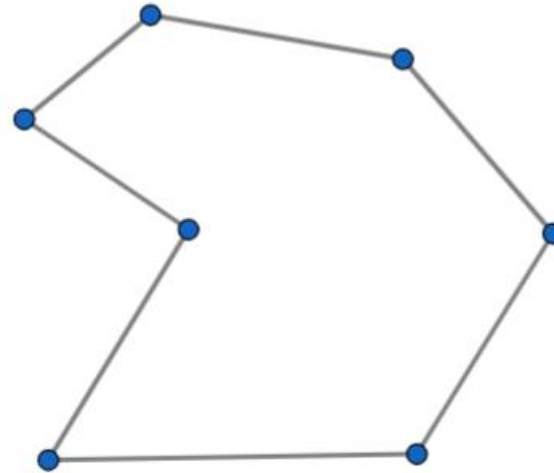
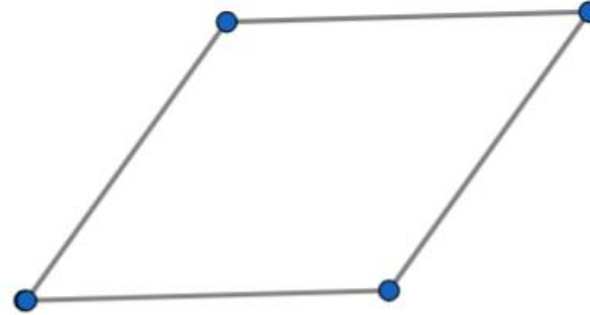
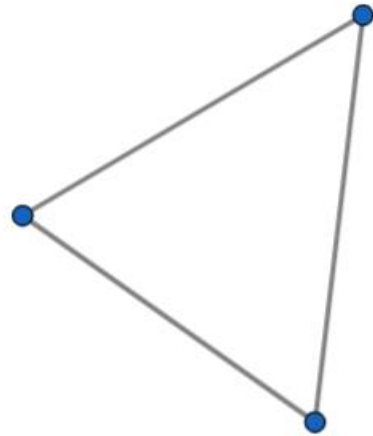
세 점의 방향성 - with 외적

```
9      using pii = pair<int, int>;
10     #define x first
11     #define y second
12
13     int ccw(pii a, pii b, pii c) {
14         b = { b.x - a.x , b.y - a.y };
15         c = { c.x - a.x, c.y - a.y };
16         int ret = b.x * c.y - b.y * c.x;
17         if (ret > 0) return 1;
18         else if (ret == 0) return 0;
19         else return -1;
20     }
```

CCW



다각형의 면적 - with ccw





다각형의 면적 - with ccw

- 사선공식

세 점의 좌표를 알 때, 세 점이 이루는 삼각형의 넓이를 구하는 공식 (평행사변형도 가능)

$$\begin{vmatrix} x1 & x2 & x3 & x1 \\ y1 & y2 & y3 & y1 \end{vmatrix} = \frac{1}{2} |(x1 * y2 + x2 * y3 + x3 * y1) \\ -(x2 * y1 + x3 * y2 + x1 * y3)|$$



다각형의 면적 - with ccw

- 사선공식

$$\begin{vmatrix} x_1 & x_2 & x_3 & x_1 \\ y_1 & y_2 & y_3 & y_1 \end{vmatrix} = \frac{1}{2} |(x_1 * y_2 + x_2 * y_3 + x_3 * y_1) - (x_2 * y_1 + x_3 * y_2 + x_1 * y_3)|$$

⇒ (x1, y1)을 (0, 0) 으로 평행이동 시키면?

$$\begin{vmatrix} 0 & x_2 - x_1 & x_3 - x_1 & 0 \\ 0 & y_2 - y_1 & y_3 - y_1 & 0 \end{vmatrix} = \frac{1}{2} |(x_2 - x_1) * (y_3 - y_1) - (x_3 - x_1) * (y_2 - y_1)|$$



다각형의 면적 - with ccw

- 사선공식

$$\begin{vmatrix} 0 & x_2 - x_1 & x_3 - x_1 & 0 \\ 0 & y_2 - y_1 & y_3 - y_1 & 0 \end{vmatrix} = \frac{1}{2} |(x_2 - x_1) * (y_3 - y_1) - (x_3 - x_1) * (y_2 - y_1)|$$

⇒ 각 값을 a_x, a_y, b_x, b_y 로 바꾸면?

$$\begin{vmatrix} 0 & a_x & b_x & 0 \\ 0 & a_y & b_y & 0 \end{vmatrix} = \frac{1}{2} |a_x * b_y - a_y * b_x|$$

⇒ 외적 식! (ccw 리턴 값으로 ret의 절대값을 반환하면 되겠다)



다각형의 면적 – with ccw

- 기본 아이디어

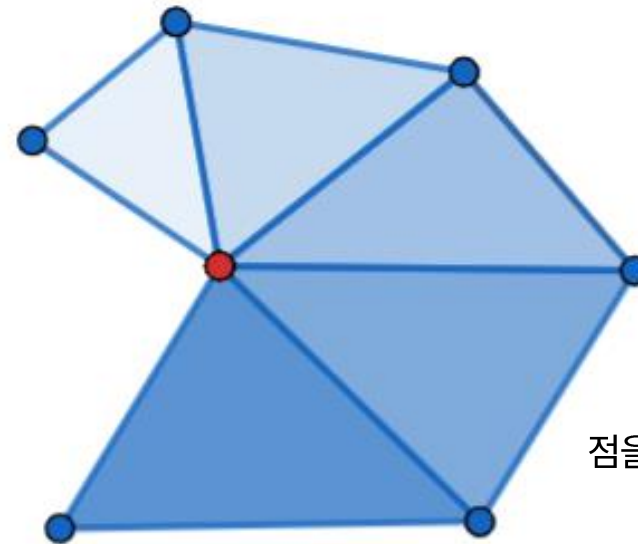
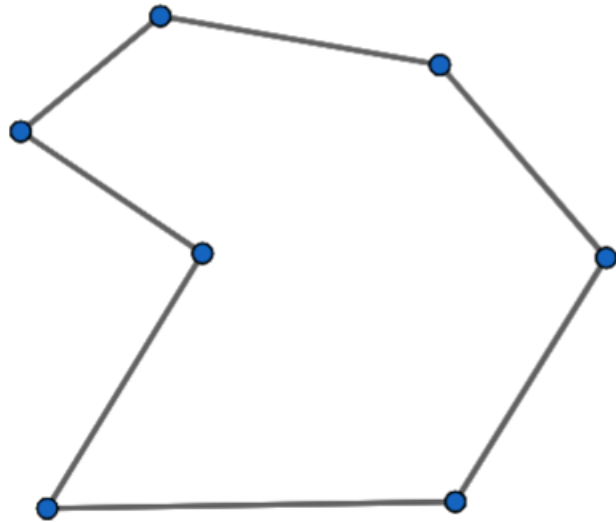
다각형의 한 꼭지점을 기준으로 잡고 삼각형 단위로 쪼개서 넓이를 더하자!



다각형의 면적 - with ccw

- 기본 아이디어

다각형의 한 꼭지점을 기준으로 잡고 삼각형 단위로 쪼개서 넓이를 더하자!



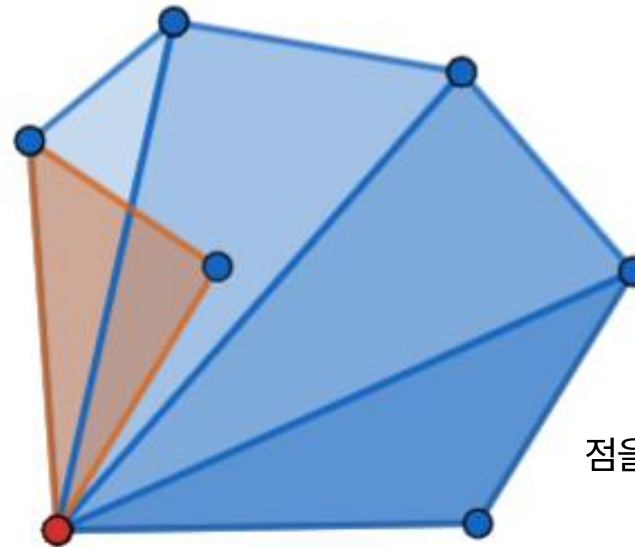
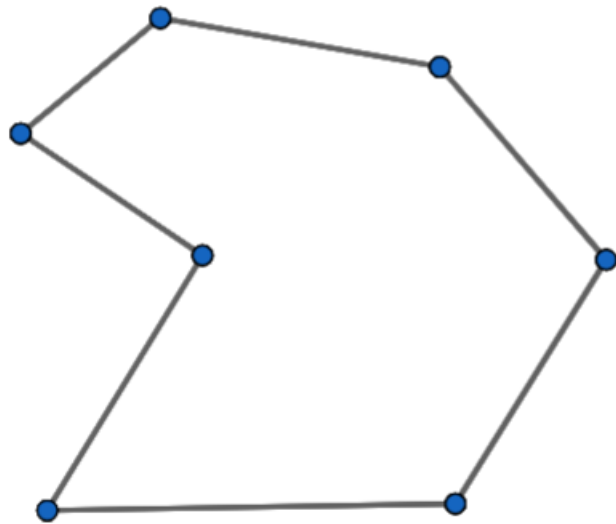
점을 잘 고르면 성공!?



다각형의 면적 - with ccw

- 기본 아이디어

다각형의 한 꼭지점을 기준으로 잡고 삼각형 단위로 쪼개서 넓이를 더하자!



점을 잘못 고르면 실패!? 주황색 부분 중복



다각형의 면적 – with ccw

- 기본 아이디어

다각형의 한 꼭지점을 기준으로 잡고 삼각형 단위로 쪼개서 넓이를 더하자!

- 확장 아이디어

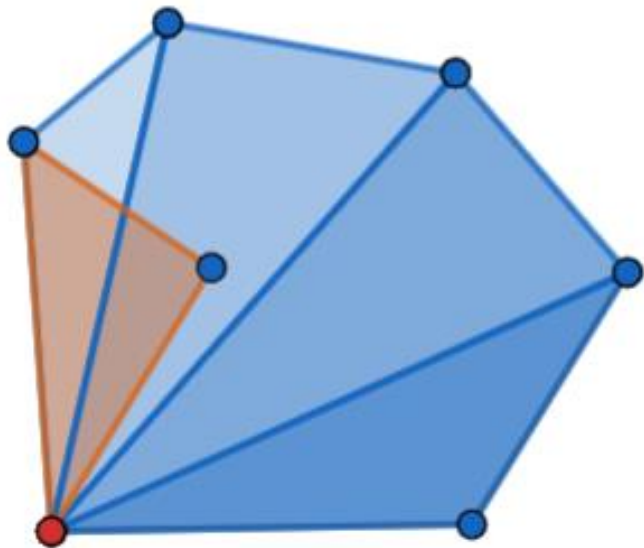
다각형의 한 꼭지점을 기준으로 잡고 반시계방향으로 돌며 기준점으로부터 절대값을 씌우지 않은 외적값을 더하자!



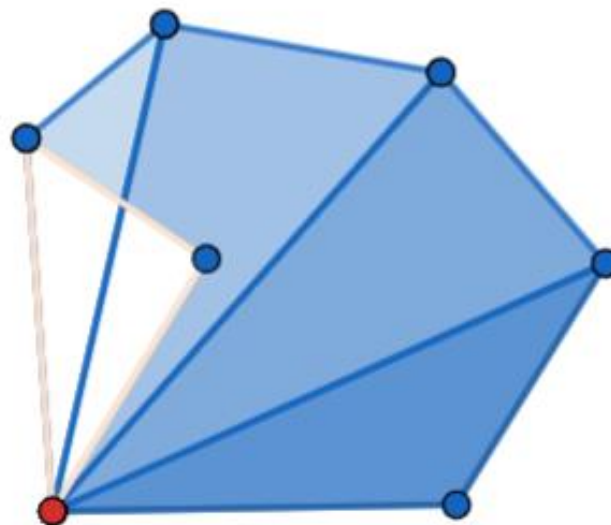
다각형의 면적 - with ccw

- 확장 아이디어

다각형의 한 꼭지점을 기준으로 잡고 반시계방향으로 돌며 기준점으로부터 절대값을 씌우지 않은 외적값을 더하자!



\Rightarrow



파란색 부분은 양수 넓이
주황색 부분은 음수 넓이 \rightarrow 중복 제거



#2166 다각형의 면적

- 다각형을 이루는 점이 순서대로 주어질 때,
- 다각형의 면적을 구하여라.

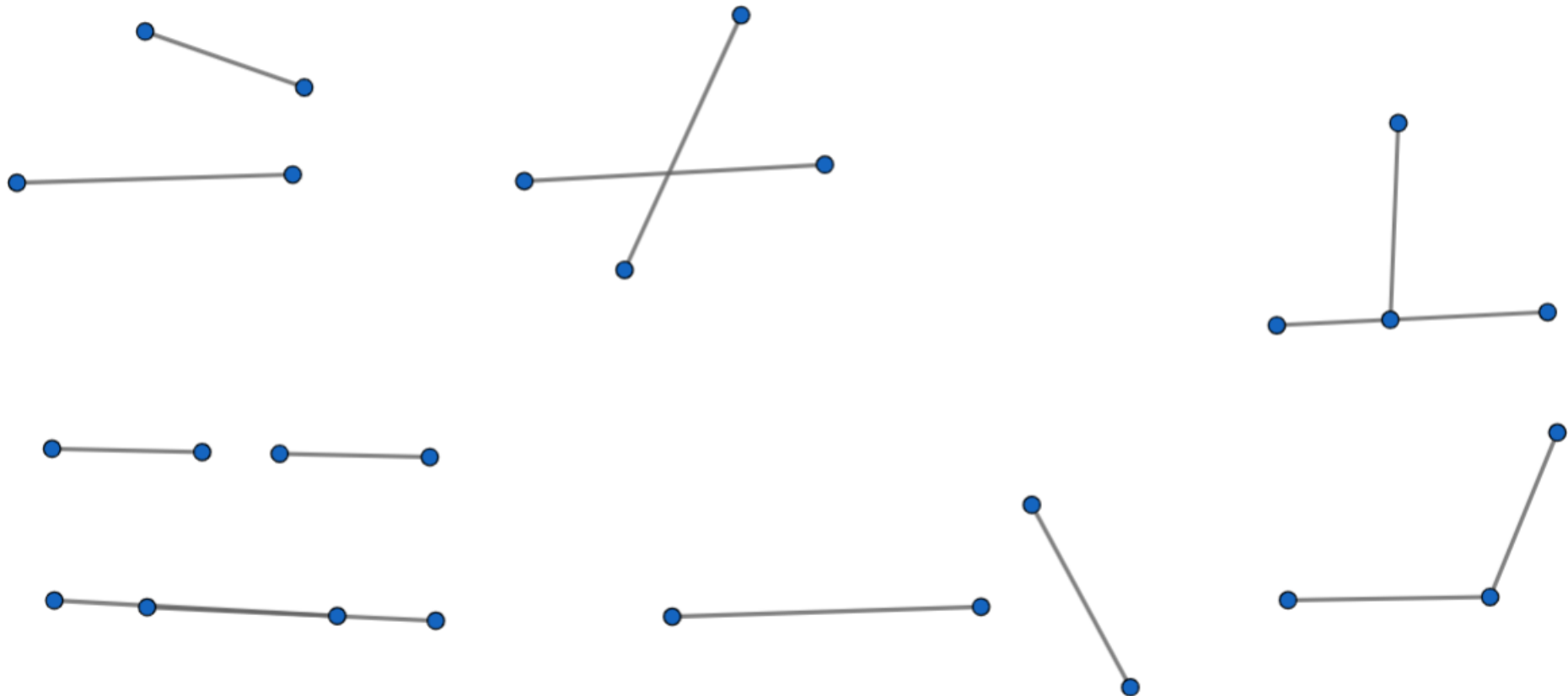


#2166 다각형의 면적

- 다각형을 이루는 점이 순서대로 주어질 때,
 - 다각형의 면적을 구하여라.
-
- 설명했던 것처럼 기준점을 잡고 순서대로 이동하며 ccw값을 더해주면 됩니다!
 - 순서대로 이동했을 때, 시계방향일 수도 있으므로 최종 넓이의 절대값을 출력!



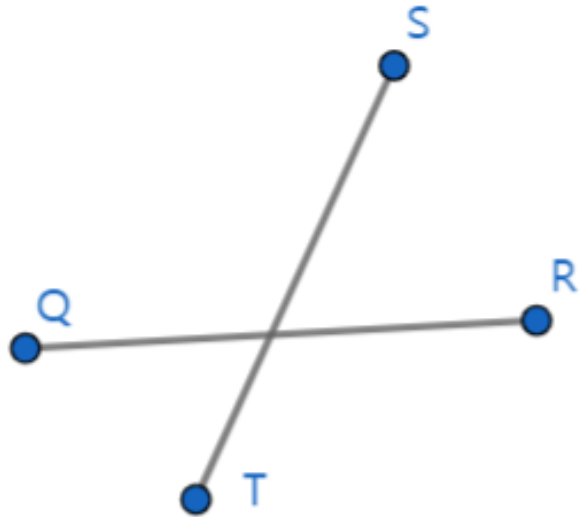
직선의 교차 판별 - with ccw





직선의 교차 판별 - with ccw

- 대놓고 교차 case

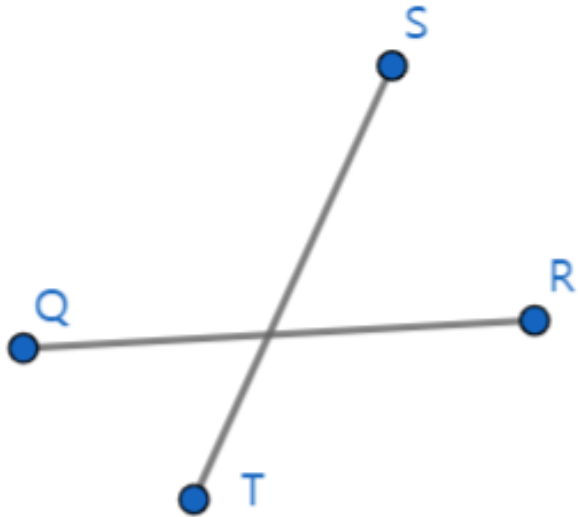


한 선분을 기준으로 다른 선분의 양 끝점이
다른 방향에 위치 \rightarrow ccw 곱이 0보다 작다



직선의 교차 판별 - with ccw

- 대놓고 교차 case



한 선분을 기준으로 다른 선분의 양 끝점이
다른 방향에 위치 → ccw 곱이 0보다 작다

$$\text{ccw}(Q, R, S) * \text{ccw}(Q, R, T) < 0$$



직선의 교차 판별 - with ccw

- 대놓고 교차 case



한 선분을 기준으로 다른 선분의 양 끝점이
다른 방향에 위치 → ccw 값이 0보다 작다
→ 가로선을 기준으로 세로선의 양 끝점이
다른 방향에 위치



직선의 교차 판별 - with ccw

- 대놓고 교차 case



한 선분을 기준으로 다른 선분의 양 끝점이
다른 방향에 위치 → ccw 곱이 0보다 작다

→ 가로선을 기준으로 세로선의 양 끝점이
다른 방향에 위치

→ 두 선분을 각각 기준으로 두 곱 모두 0보다 작아야함

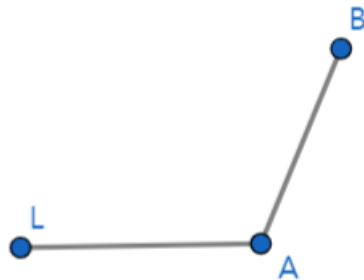
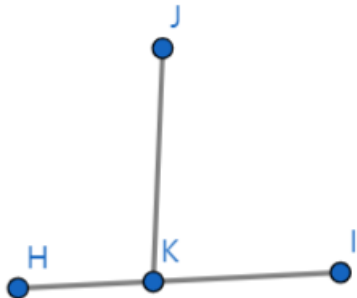
$$\text{ccw}(D, E, F) * \text{ccw}(D, E, G) < 0 \ \&\& \ \text{ccw}(F, G, D) * \text{ccw}(F, G, E) < 0$$

→ false!



직선의 교차 판별 – with ccw

- 선분의 끝이 걸치는 case



한 선분을 기준으로 다른 선분의 한 끝점이
일직선으로 위치

$$\text{ccw}(H, I, K) * \text{ccw}(H, I, J) = 0$$

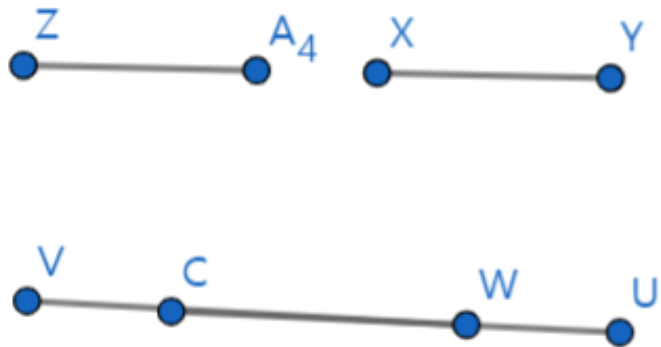
$$\text{ccw}(J, K, H) * \text{ccw}(J, K, I) < 0$$

ccw의 곱 중 하나가 0



직선의 교차 판별 – with ccw

- 무수히 많이 교차하는 case



두 선분이 일직선

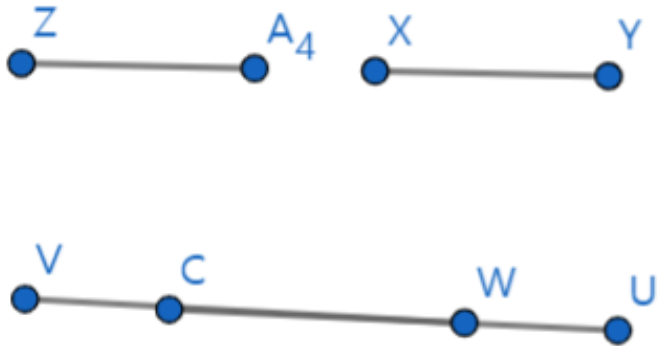
$$\text{ccw}(V, W, C) * \text{ccw}(V, W, U) = 0$$

$$\text{ccw}(C, U, V) * \text{ccw}(C, U, W) = 0$$



직선의 교차 판별 - with ccw

- 무수히 많이 교차하는 case



두 선분이 일직선

$$\text{ccw}(V, W, C) * \text{ccw}(V, W, U) = 0$$

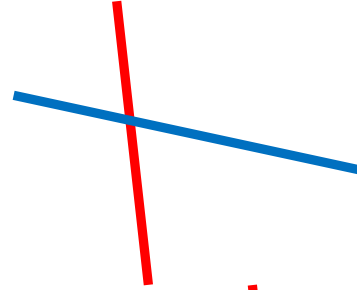
$$\text{ccw}(C, U, V) * \text{ccw}(C, U, W) = 0$$

일직선일 때, 교차하지 않으려면 x좌표가 중복되지 않음.
x좌표가 같을 때는 y좌표로 판단 가능

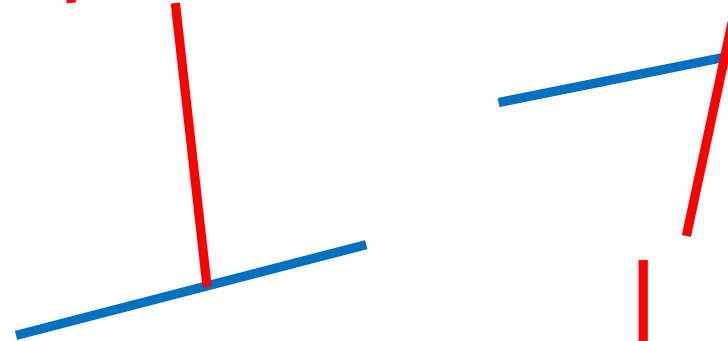


직선의 교차 판별 - with ccw

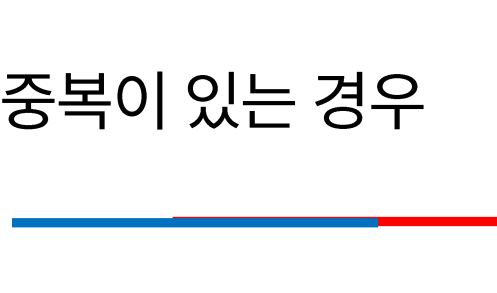
1. 두 ccw 곱이 모두 0보다 작은 경우



2. 둘 중, 하나의 ccw 곱이 0보다 작은 경우



3. 두 ccw 곱이 모두 0이고, x좌표와 y좌표의 중복이 있는 경우

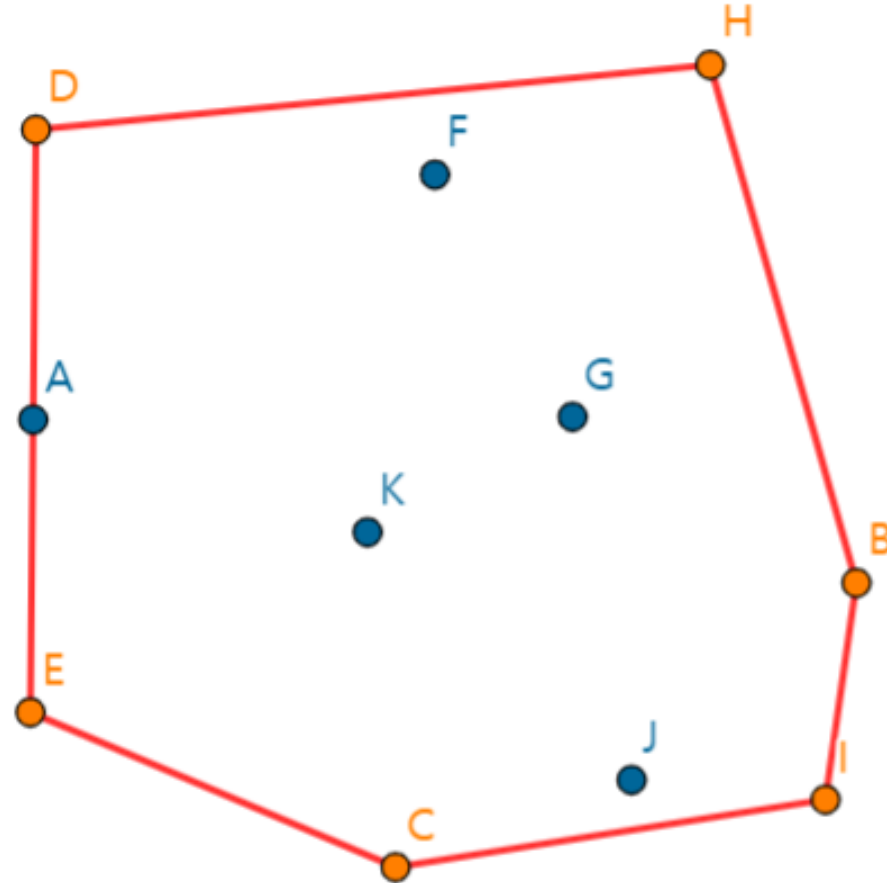




Convex Hull (볼록 껍질)

- 모든 점 또는 모든 영역을 감싸는 가장 작은 볼록 다각형
- 다각형과 점의 관계
- 다각형과 직선의 관계

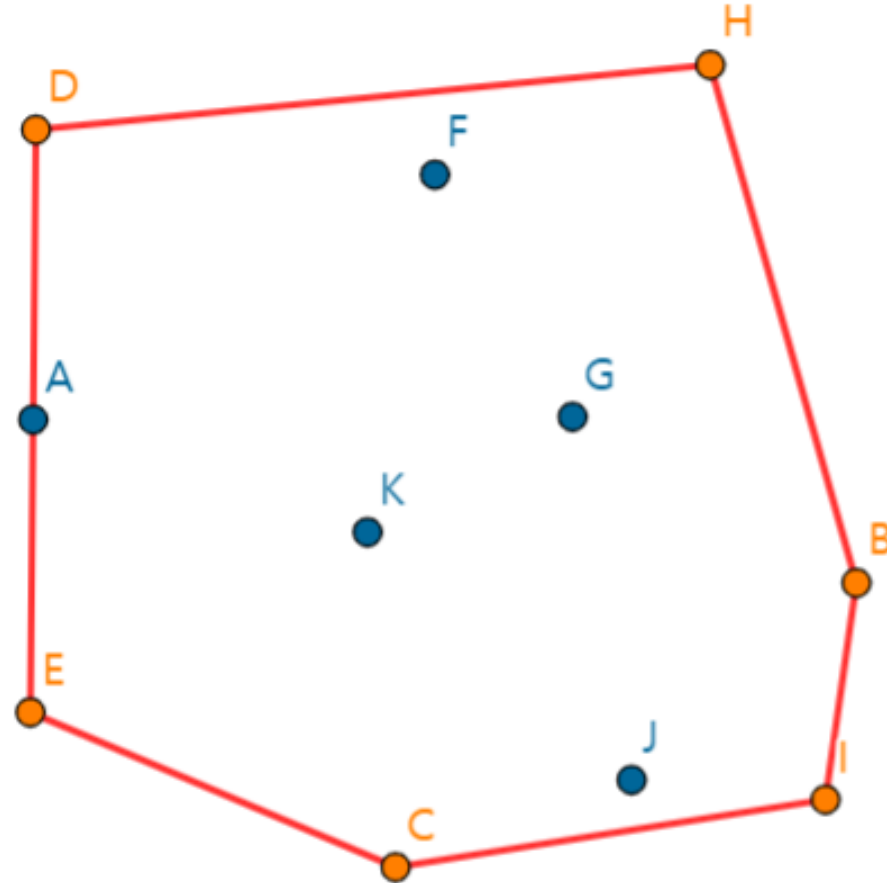
⇒ 그림 만들어보자!





Graham scan algorithm

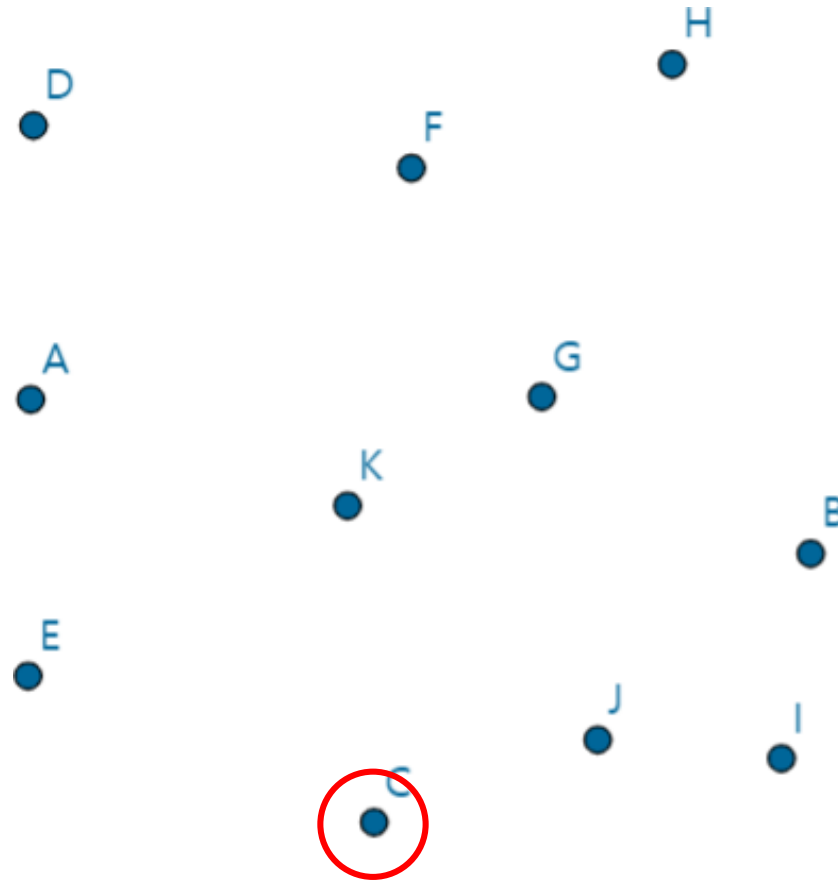
1. 기준점 뽑기 – 가장 좌측 점 or 가장 하단 점
2. 기준점을 기준으로 각도 정렬
+ 각도가 같을 땐 거리 정렬
3. 스택에 쌓아가며 top의 두 점과
새로운 점의 관계가 ccw를 이루도록 한다.





Graham scan algorithm

기준점 선택 - C

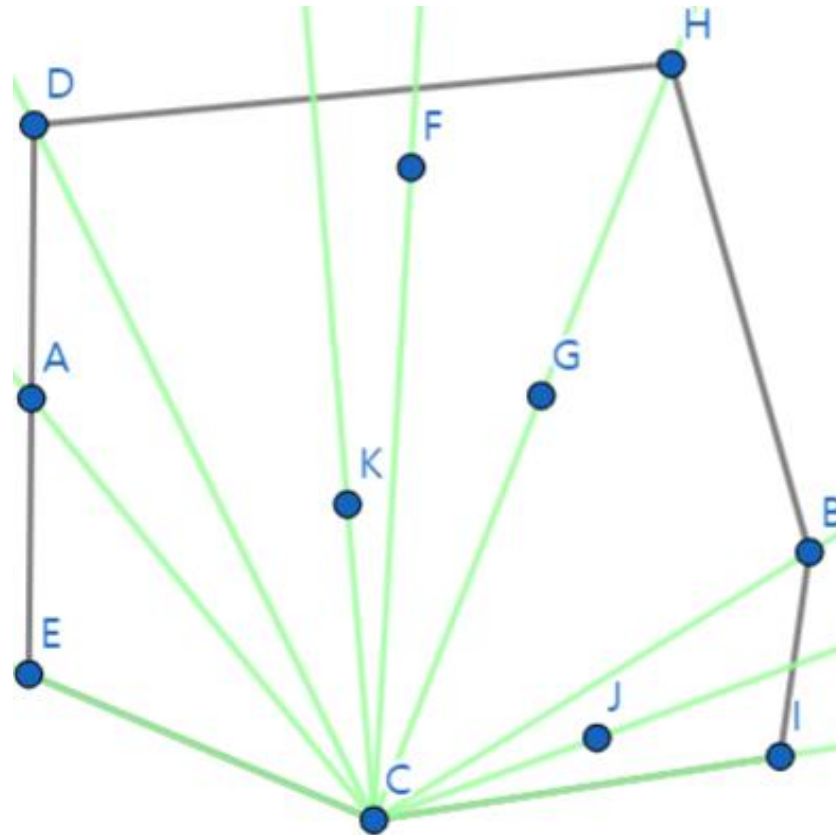




Graham scan algorithm

좌표들을 각도 정렬

C-I-J-B-G-H
-F-K-D-A-E

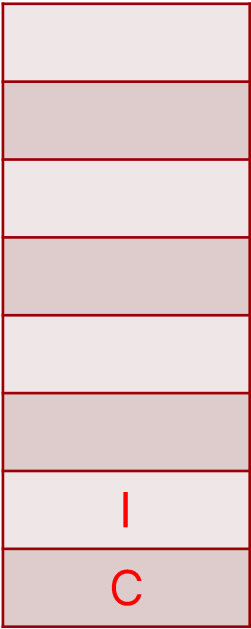
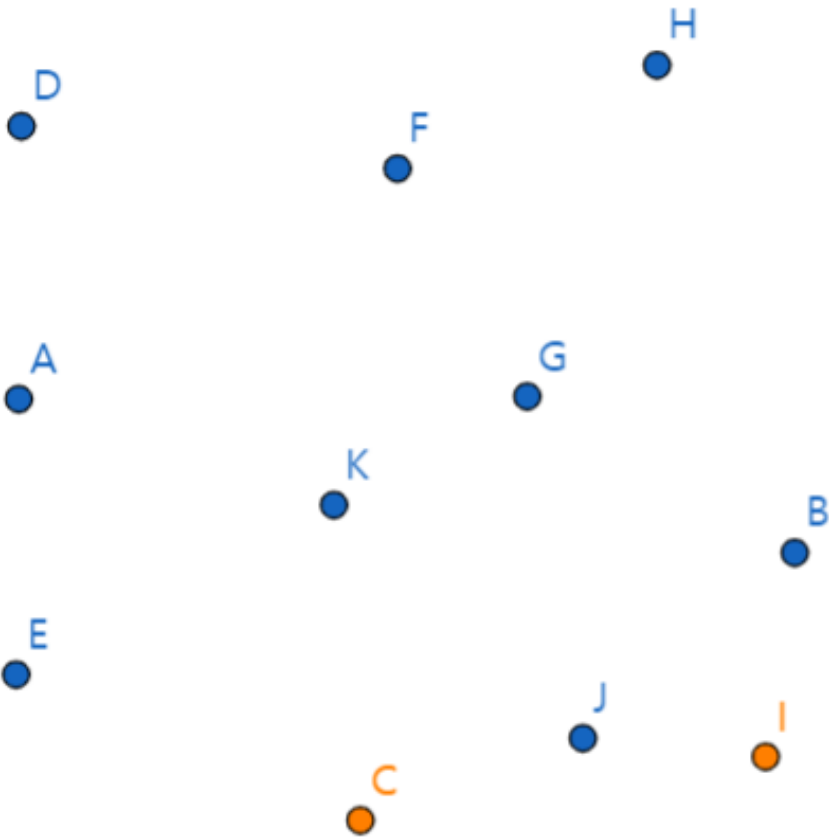




Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

두 점을 스택에 넣어놓기



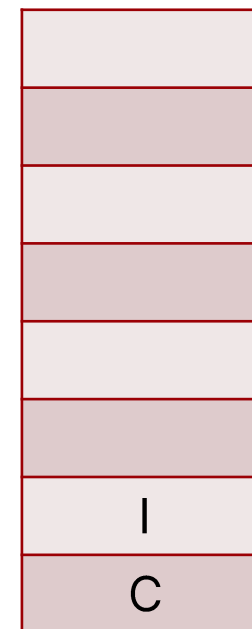
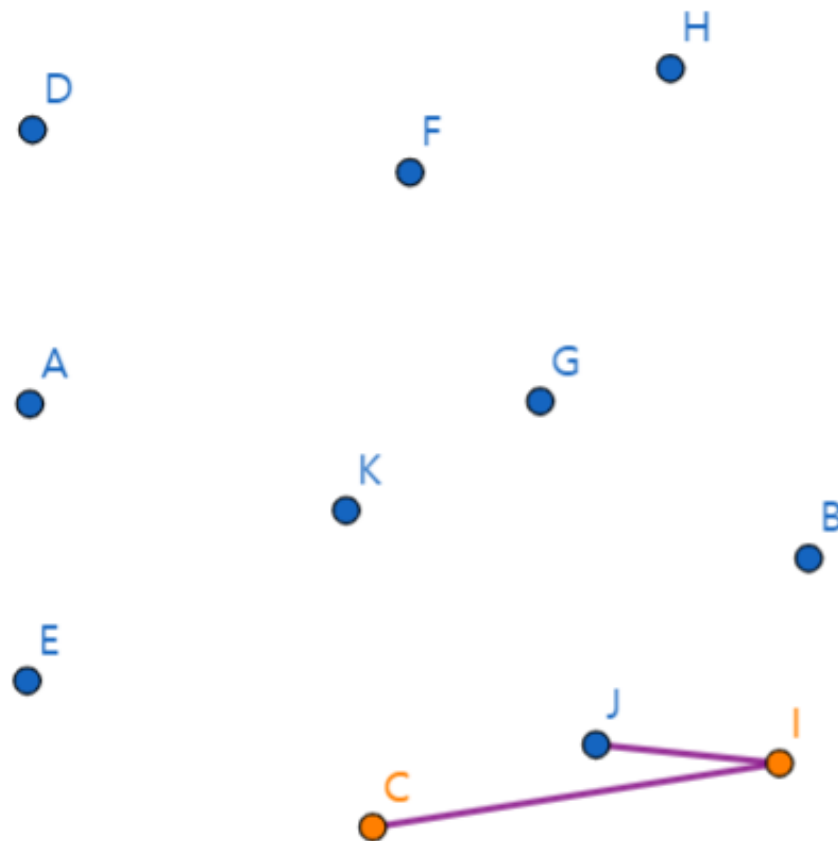
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 있다면 push



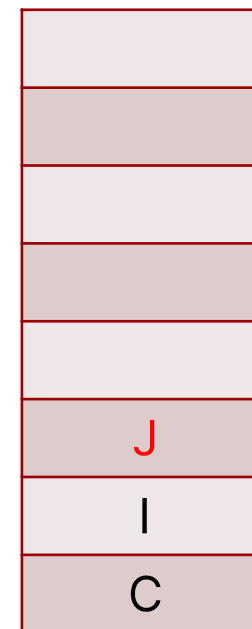
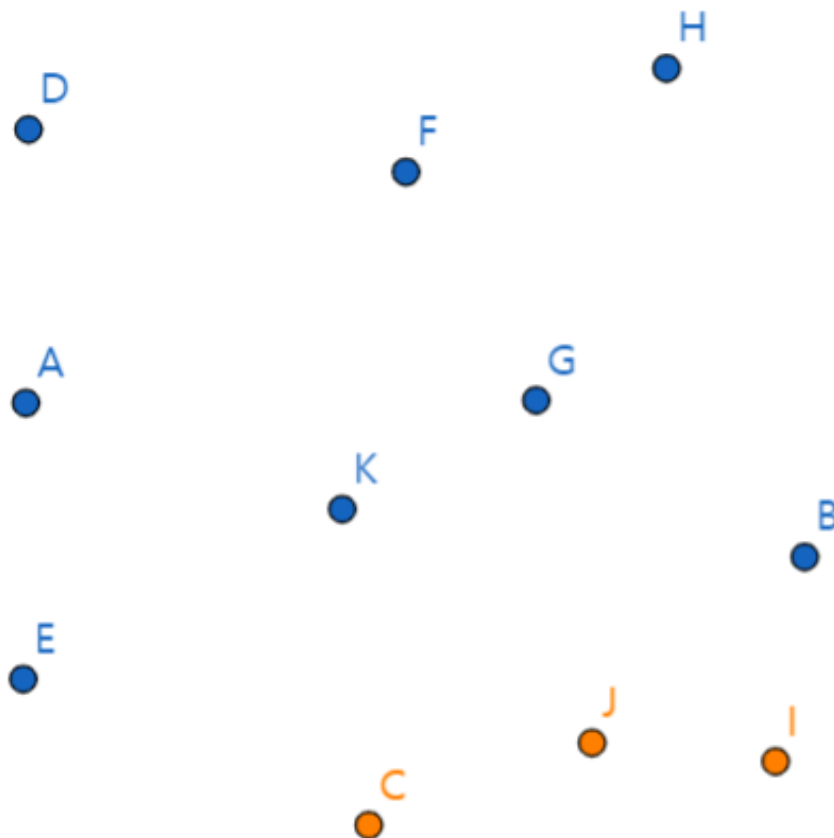
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 있다면 push



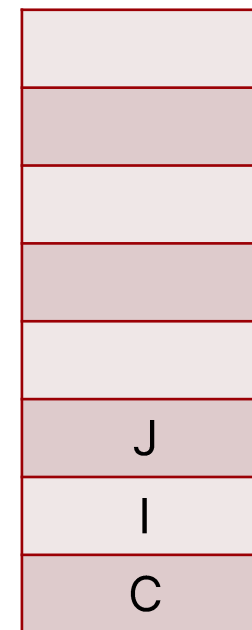
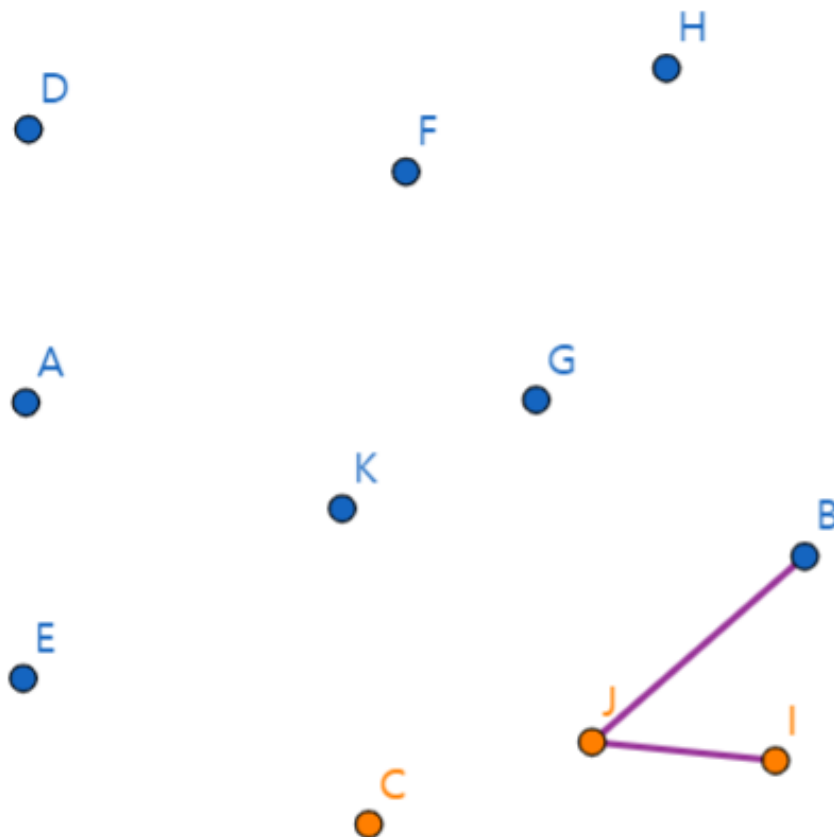
스택



Graham scan algorithm

스택 위 두 점과 새로운 점이
ccw를 이룰 수 없다면 pop

정렬 순서 : C I J B G H K F D A E



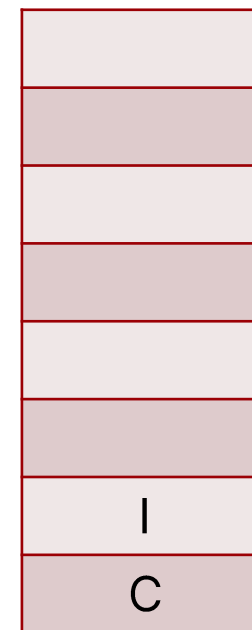
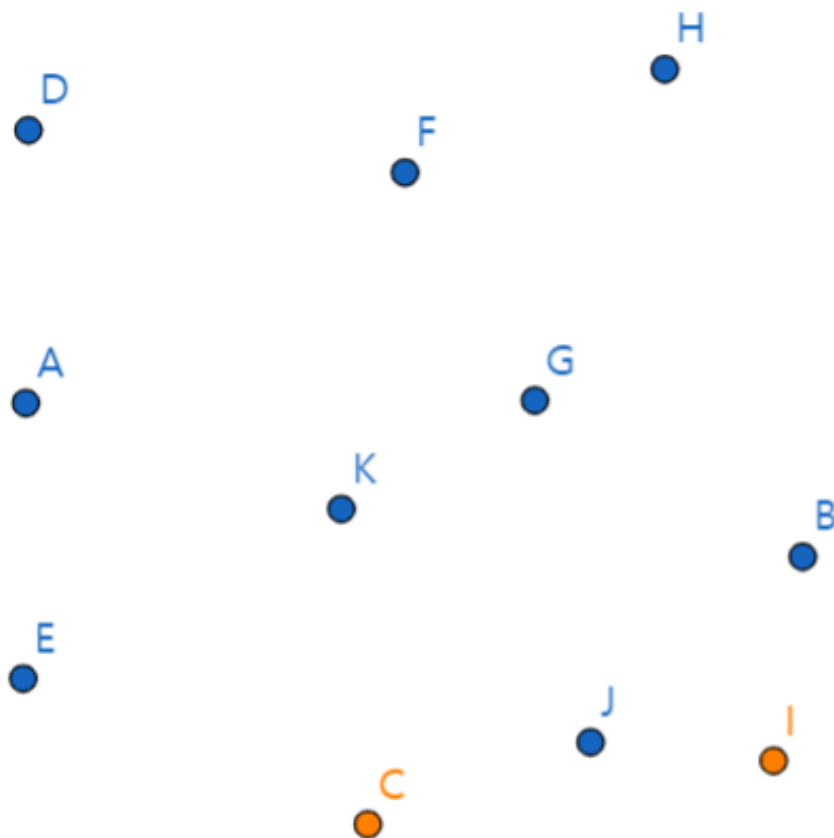
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 없다면 pop



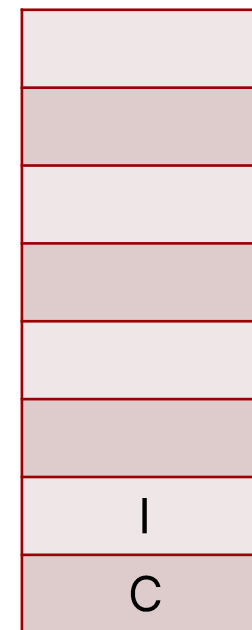
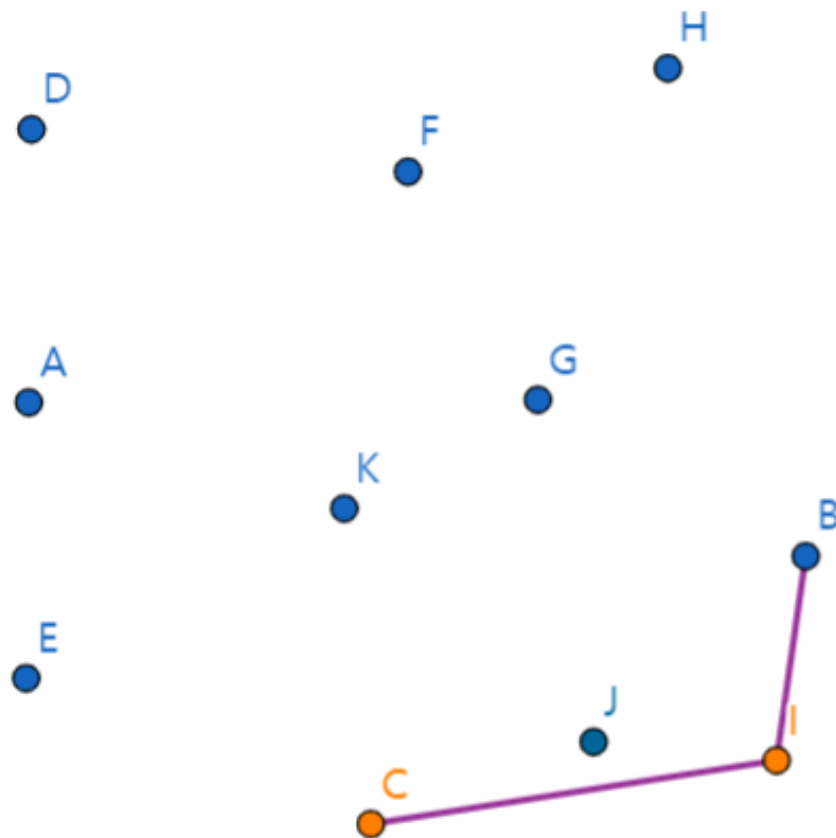
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 있다면 push



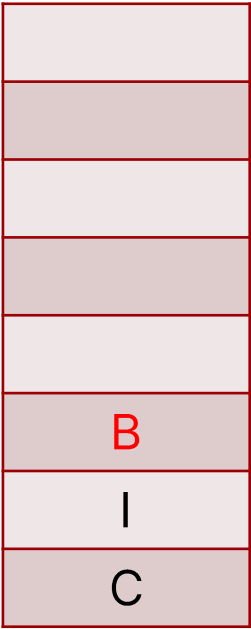
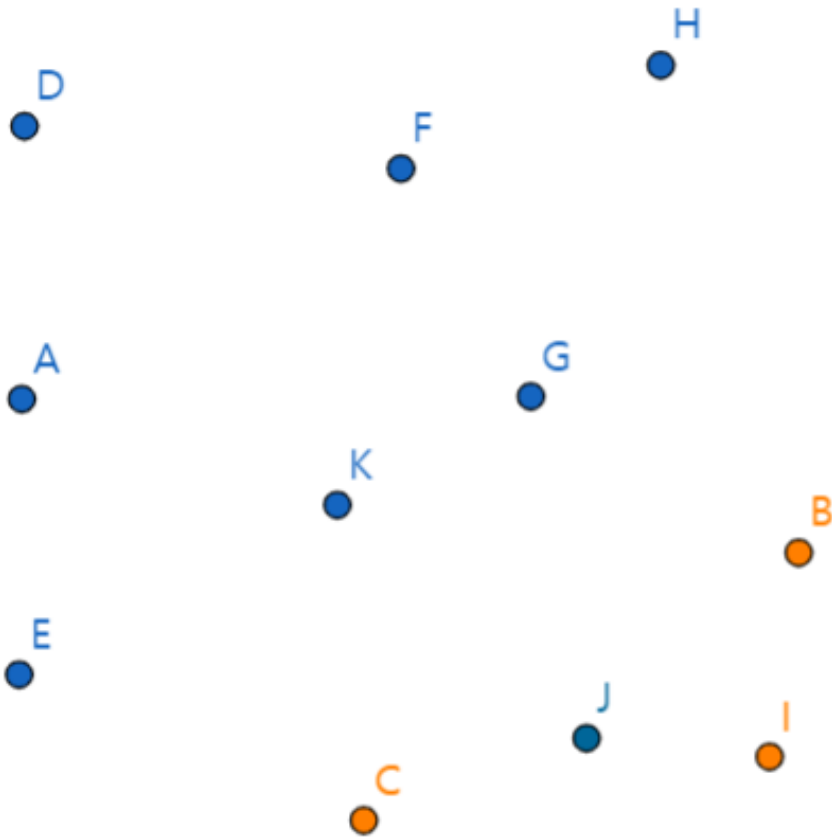
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 있다면 push



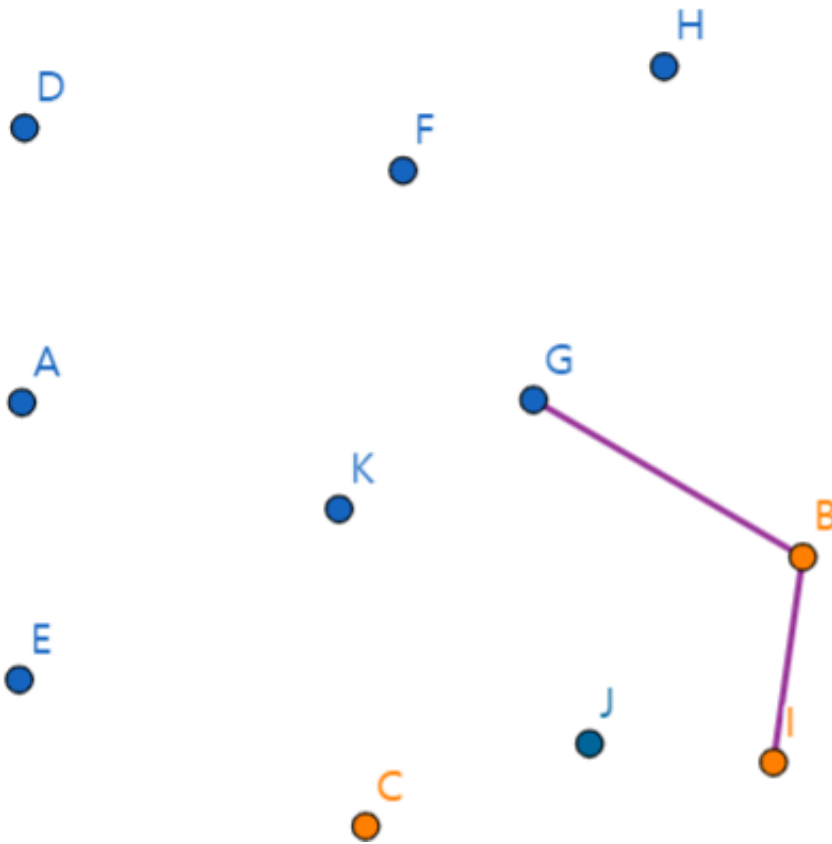
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 있다면 push



B
I
C

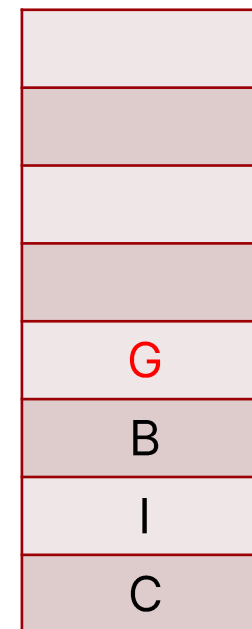
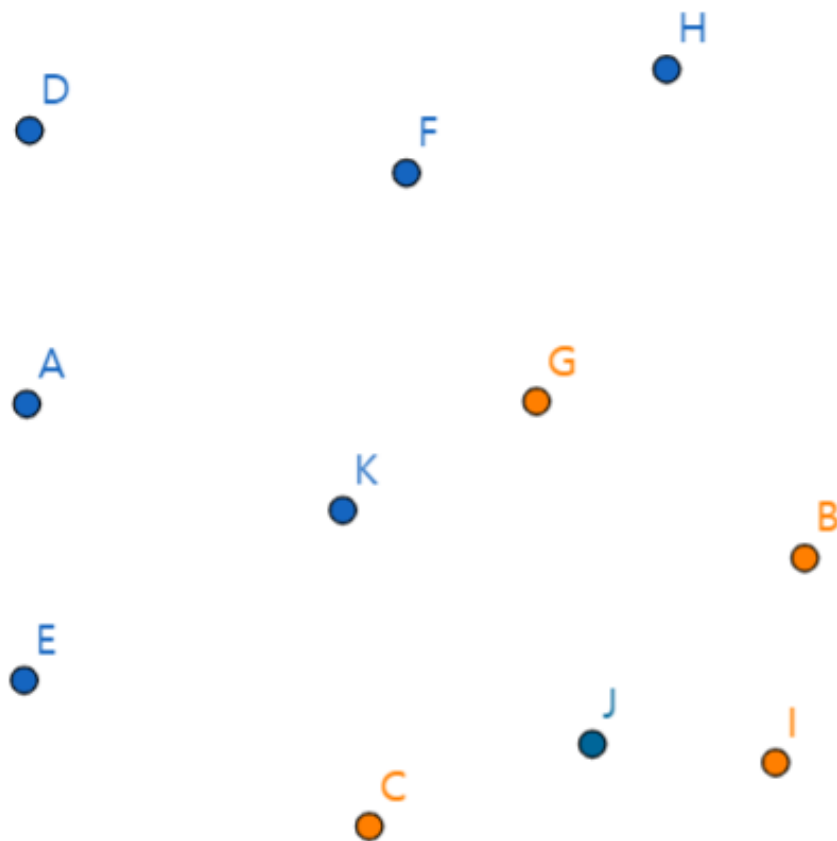
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

스택 위 두 점과 새로운 점이
ccw를 이룰 수 있다면 push

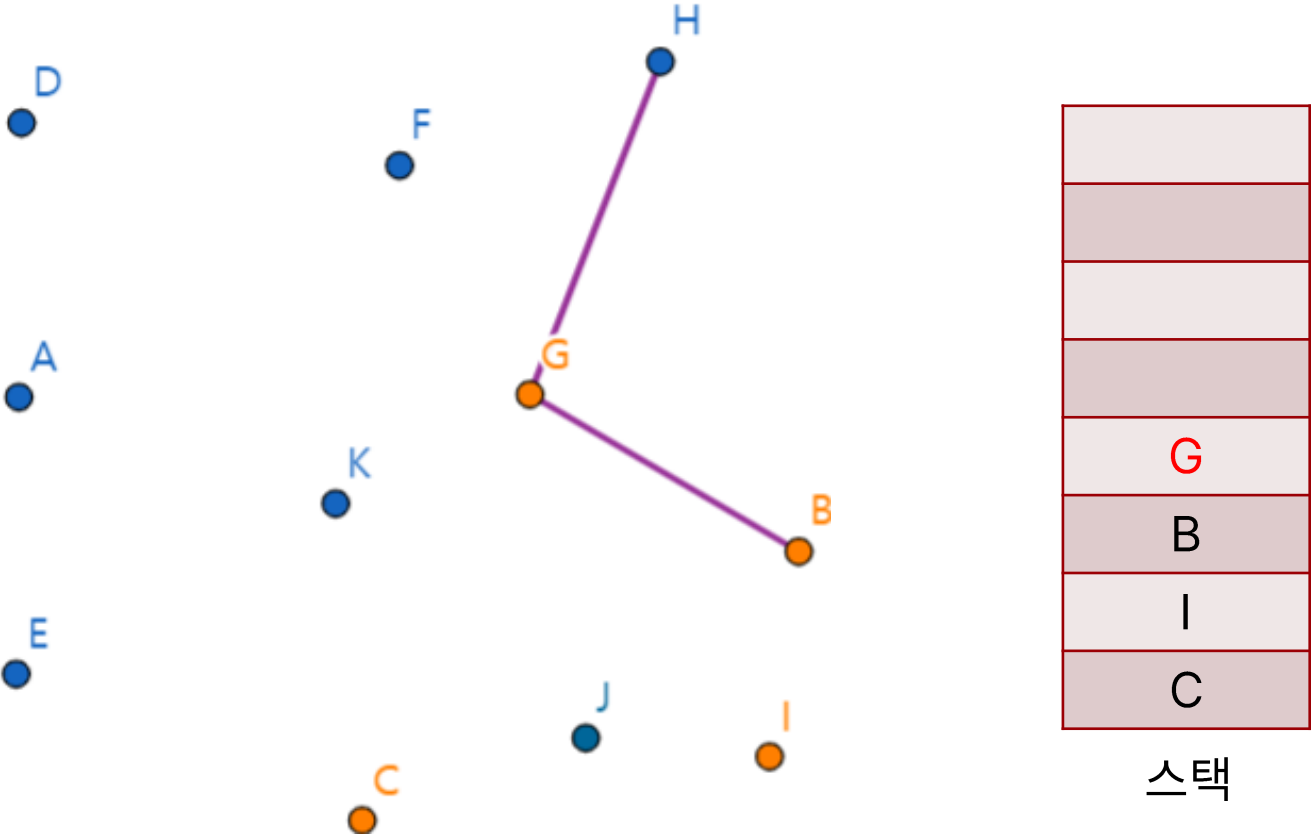


스택



Graham scan algorithm

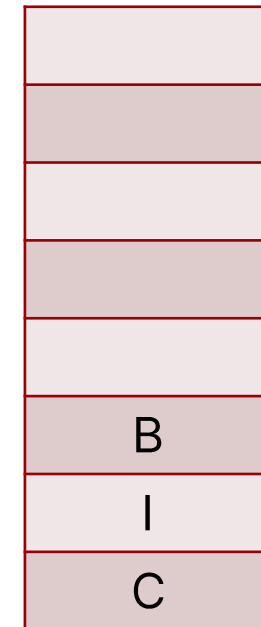
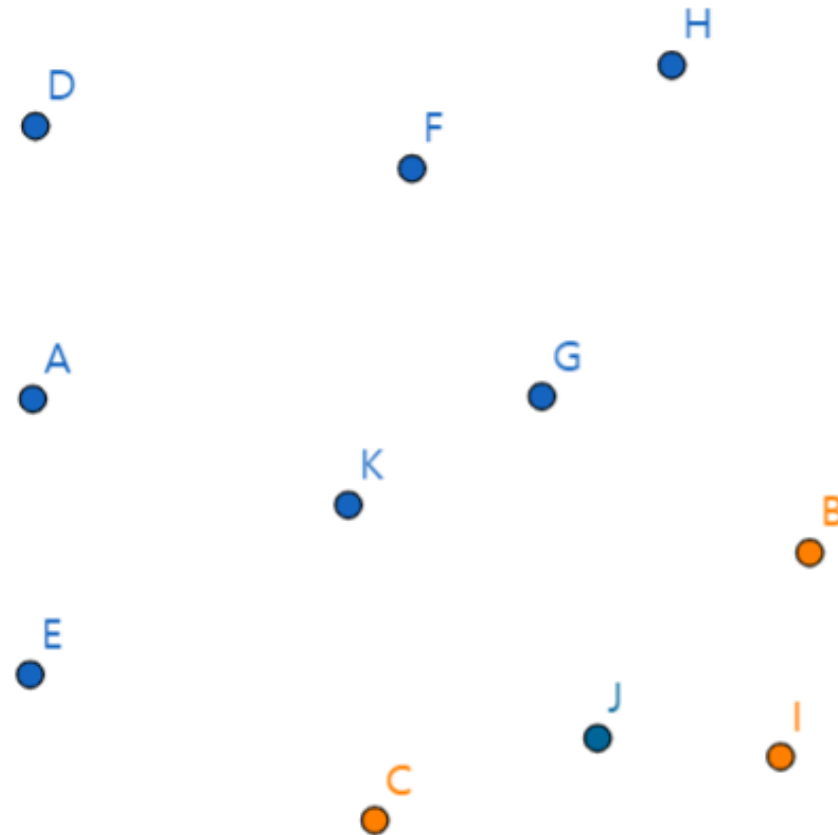
정렬 순서 : C I J B G H K F D A E





Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

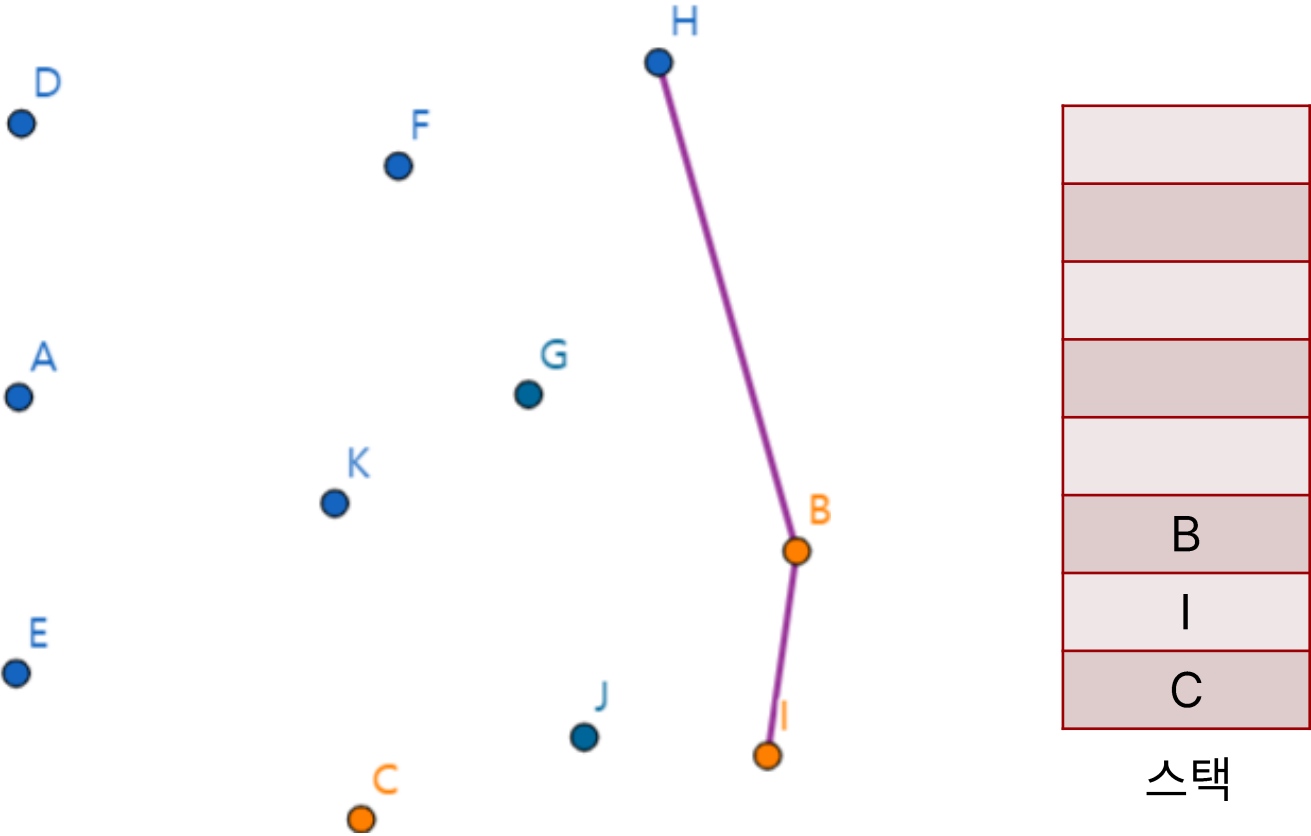


스택



Graham scan algorithm

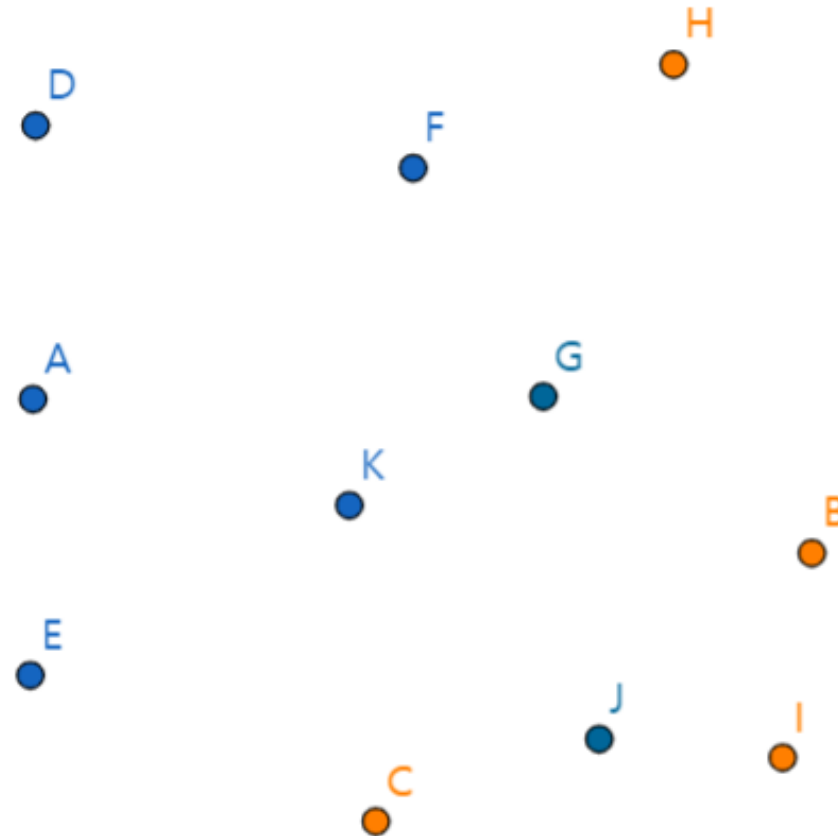
정렬 순서 : C I J B G H K F D A E





Graham scan algorithm

정렬 순서 : C I J B G H K F D A E



H
B
I
C

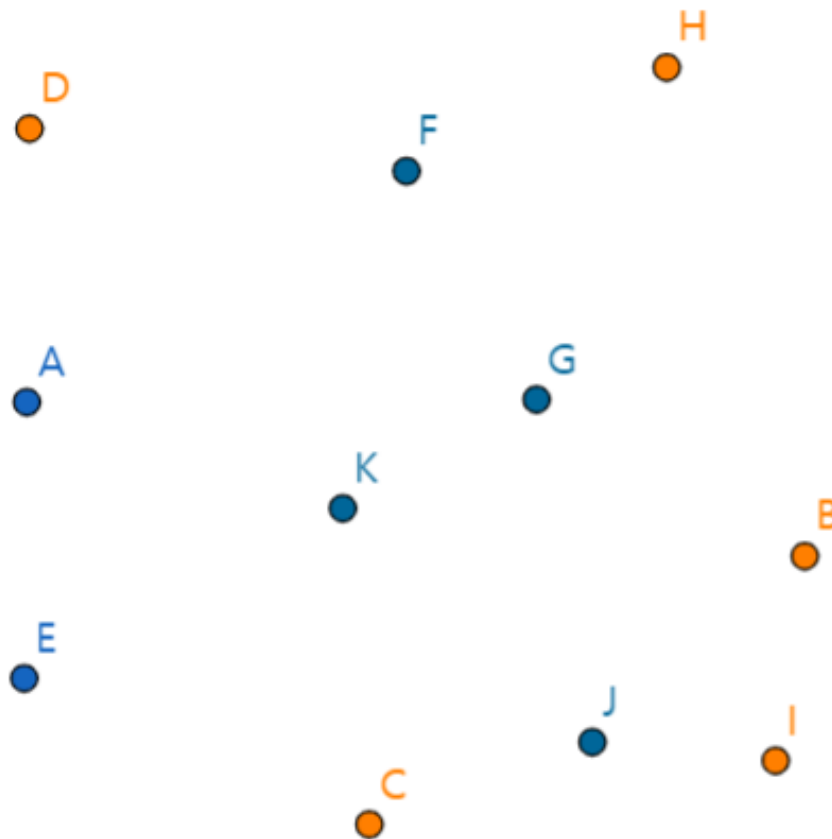
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

몇 단계 건너뛰어서,,,



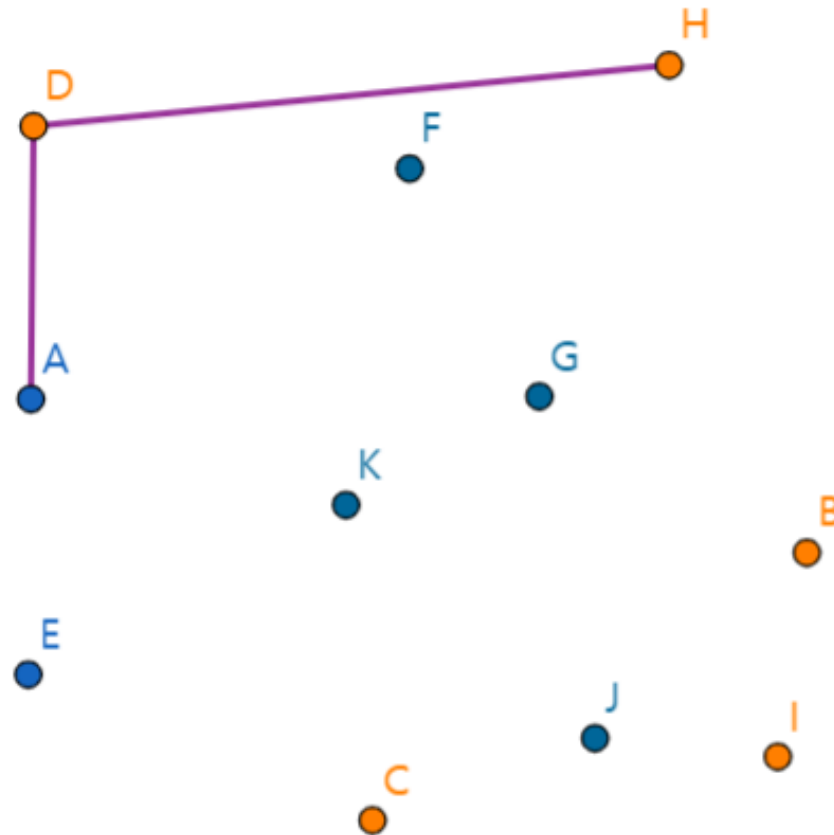
D
H
B
I
C

스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E



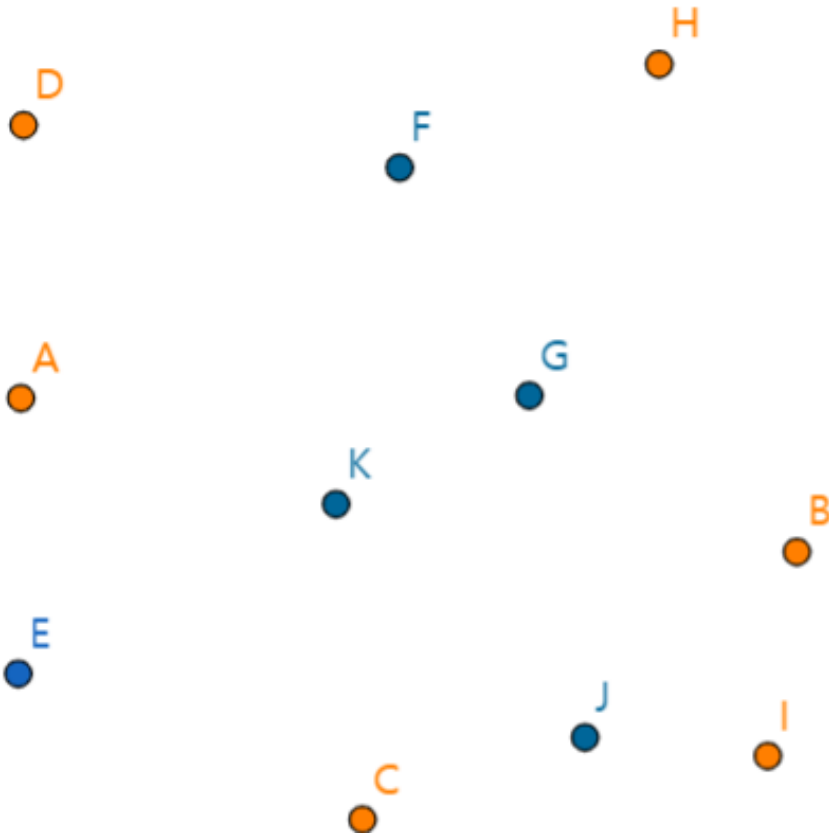
D
H
B
I
C

스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E



A
D
H
B
I
C

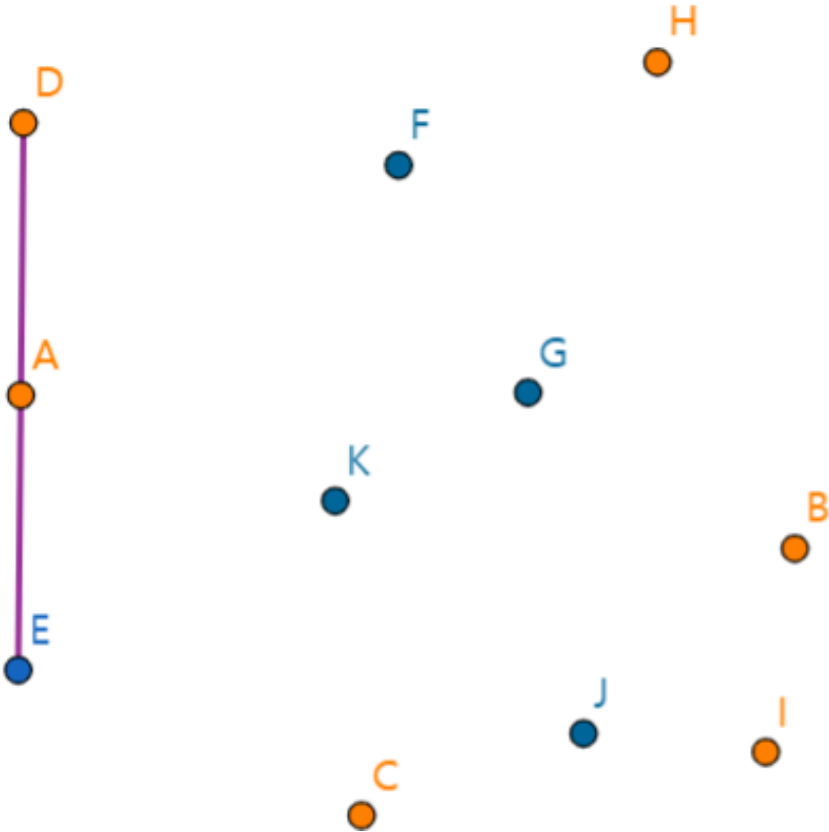
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

일직선 위에 3개 이상의
점이 올 때?



A
D
H
B
I
C

스택

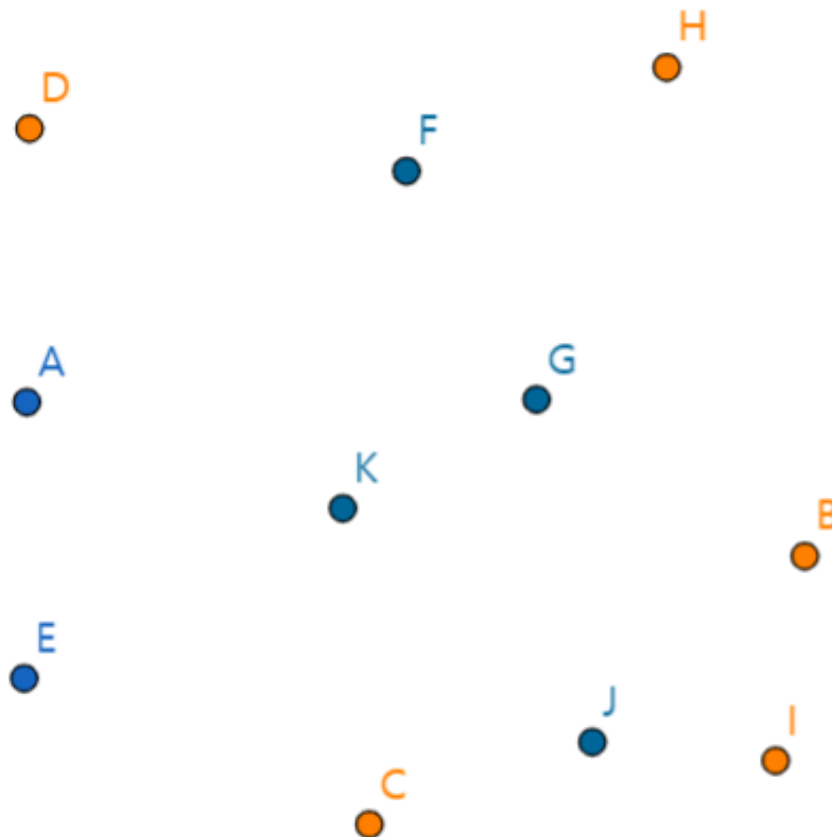


Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

일직선 위에 3개 이상의
점이 올 때?

⇒ 내부에 있는 점은 무시됨



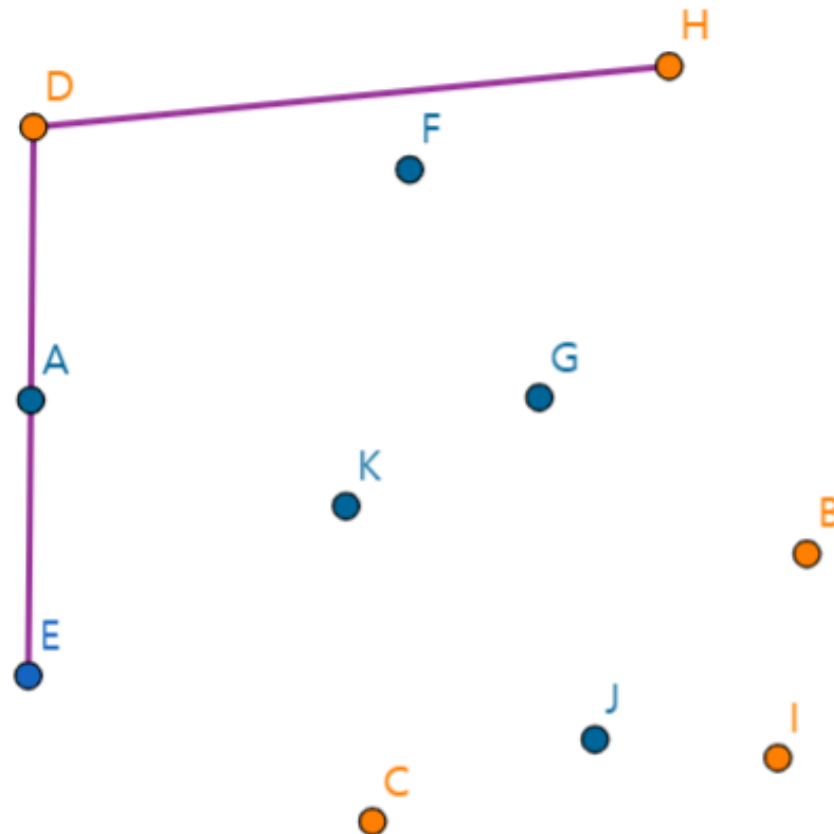
D
H
B
I
C

스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E



D
H
B
I
C

스택

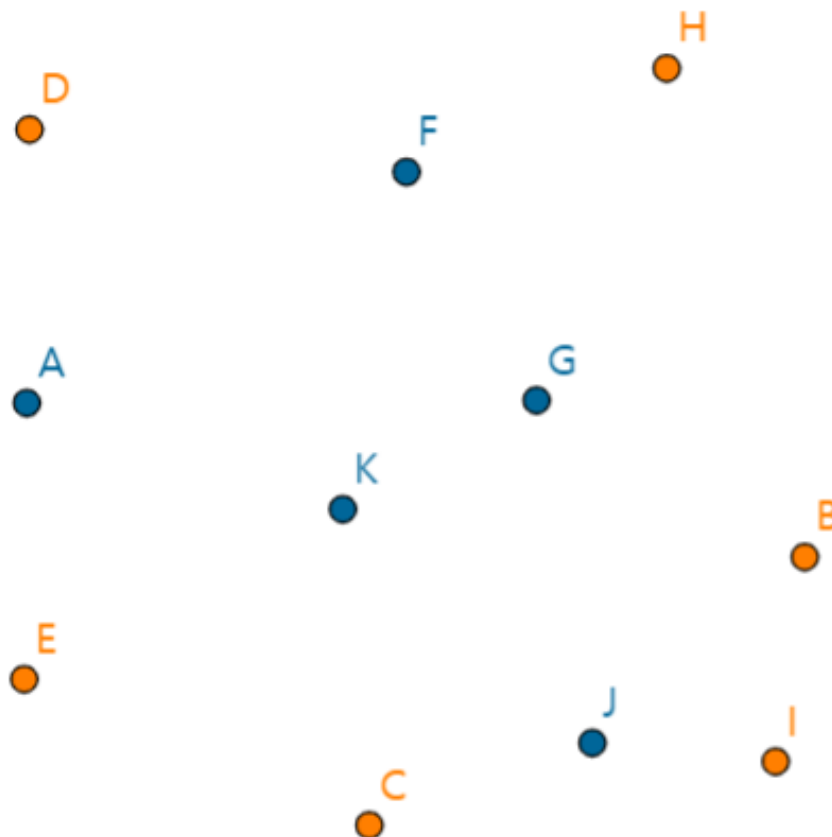


Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

일직선 위에 3개 이상의
점이 올 때?

⇒ 내부에 있는 점은 무시됨



E
D
H
B
I
C

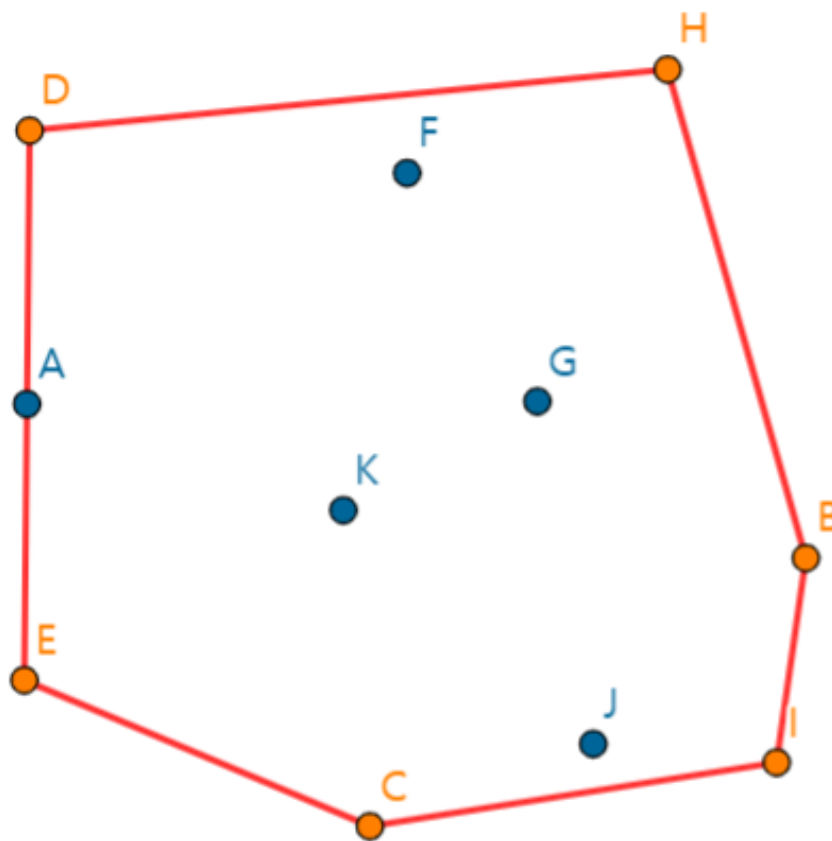
스택



Graham scan algorithm

정렬 순서 : C I J B G H K F D A E

반복을 통해 다음을
구할 수 있다!



E
D
H
B
I
C

스택



Graham scan algorithm

- 93 ~ 95 line
pair 를 벡터로 쓰기 위한 처리
- 98 ~ 99 line
P : point 보관,
CH : 볼록 껍질 정점 보관
- 101 ~ 108 line
ccw구현

```
93     using pii = pair<int, int>;
94     #define x first
95     #define y second
96
97     int n;
98     vector<pii> P;
99     vector<int> CH;
100
101     int ccw(pii a, pii b, pii c) {
102         b = { b.x - a.x, b.y - a.y };
103         c = { c.x - a.x, c.y - a.y };
104         ll ret = 1LL * b.x * c.y - 1LL * b.y * c.x;
105         if (ret > 0) return 1;
106         else if (ret == 0) return 0;
107         else return -1;
108     }
```



Graham scan algorithm

- 109 ~ 113 line
각도 정렬, 같은 각 거리 정렬
- 135 ~ 138 line
인풋 처리
- 139 line
자동으로 좌측하단 점을 찾아 줌
- 140 ~ 141 line
정렬
 $\Rightarrow O(n \log n)$

```
109 bool cmp(pii& a, pii& b) {  
110     if (ccw(P[0], a, b) == 0)  
111         return a.x - P[0].x < b.x - P[0].x;  
112     else return ccw(P[0], a, b) == 1;  
113 }
```

```
134 cin >> n;  
135 for (int i = 0; i < n; ++i) {  
136     int x, y; cin >> x >> y;  
137     P.push_back({ x, y });  
138 }  
139 swap(P[0], *min_element(P.begin(), P.end()));  
140 sort(P.begin() + 1, P.end(), cmp);  
141 graham_scan();  
142 cout << CH.size();  
143 }
```



Graham scan algorithm

- 115 line
기본 점 2개 박아놓기
- 119 ~ 120 line
스택 top 두 점 뽑기
- 121 ~ 123 line
스택에서 top 두 점 + 새로운 점의
ccw값이 1인지 체크
→ p2 점을 다시 CH에 넣어주기

$\Rightarrow O(n)$

```
114 void graham_scan() {  
115     CH.push_back(0); CH.push_back(1);  
116  
117     for (int i = 2; i < n; ++i) {  
118         while (CH.size() >= 2) {  
119             int p2 = CH.back(); CH.pop_back();  
120             int p1 = CH.back();  
121             if (ccw(P[p1], P[p2], P[i]) == 1) {  
122                 CH.push_back(p2);  
123                 break;  
124             }  
125         }  
126         CH.push_back(i);  
127     }  
128 }
```

$O(n \log n + n) \Rightarrow O(n \log n)$



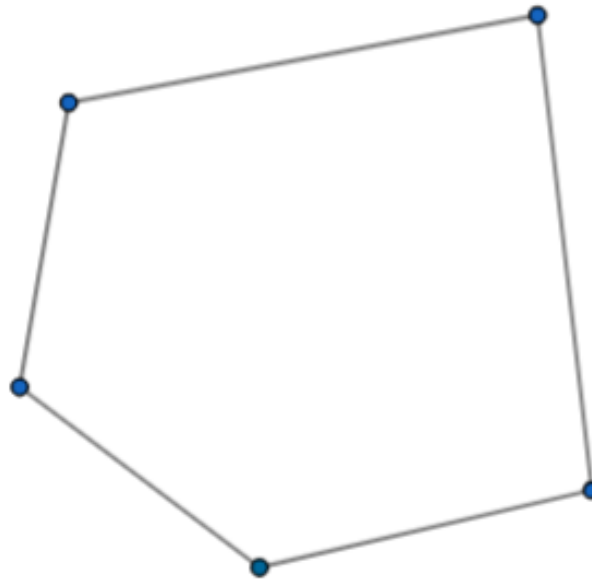
#10903 Wall construction

- 홍준이는 돈이 차고 넘쳐서 미술관을 짓는다
- 원기둥을 N 개 만들었고, 아주 얇은 유리를 이용해 외벽을 세운다.
- 모든 기둥은 외벽내부에 있고 외벽이 폐곡선 형태일 때, 외벽의 둘레는?
- $2 \leq N \leq 1000$, $1 \leq$ 원기둥의 반지름 ≤ 100
- $-10^4 \leq$ 각 기둥의 x, y 좌표 $\leq 10^4$



#10903 Wall construction

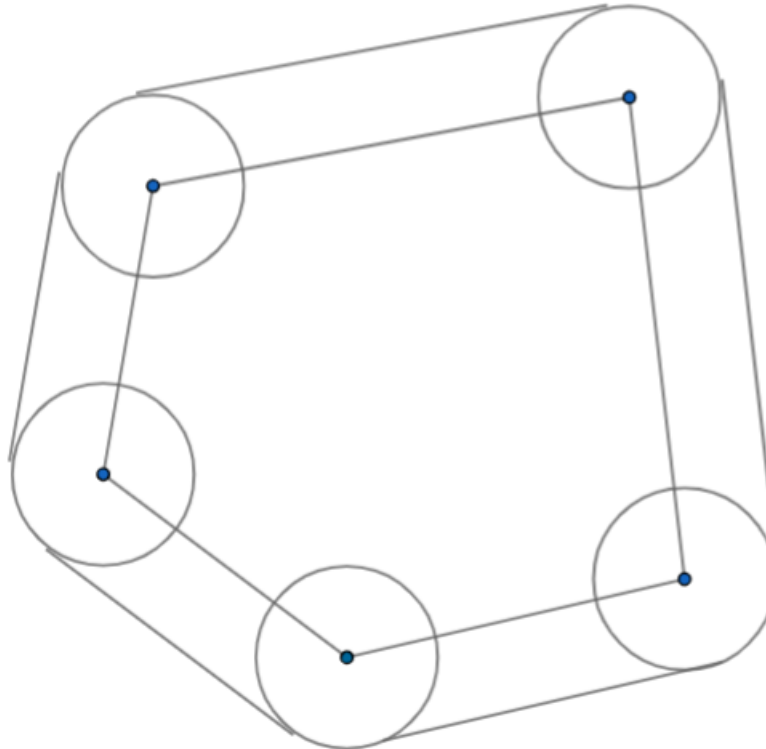
- 기둥의 좌표들을 이용해 convex hull 생성 가능





#10903 Wall construction

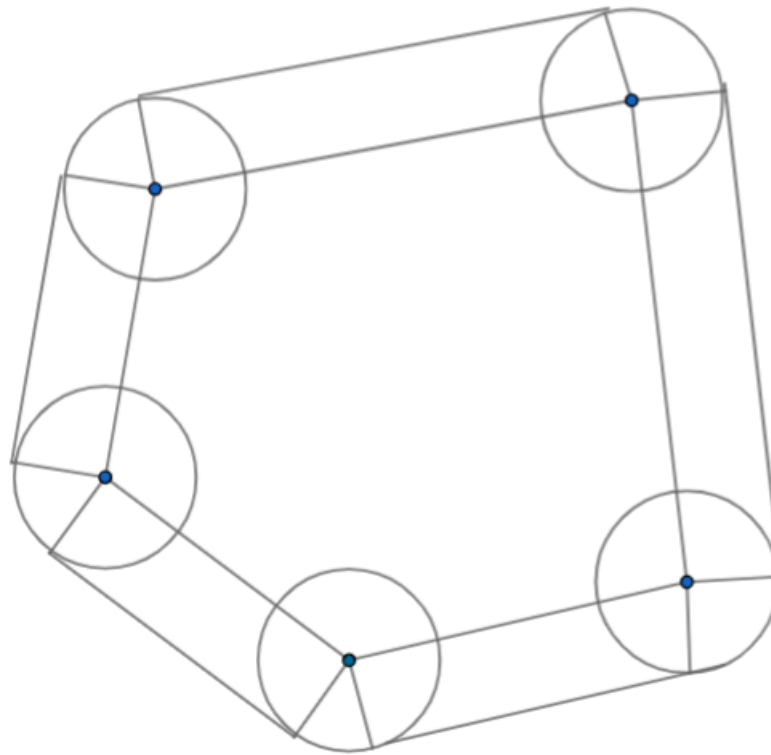
- 각 기둥은 반지름이 동일하다





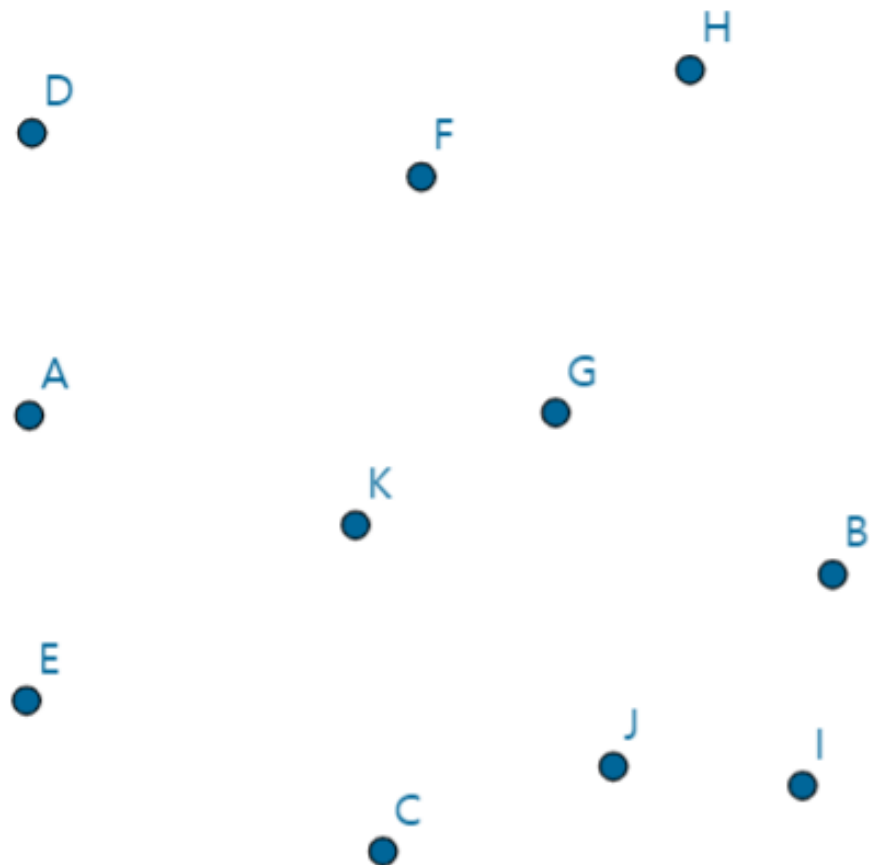
#10903 Wall construction

- 다각형의 외각의 합은 2π 이다.





가장 먼 두 점 고르기



눈에 보이는 후보

D - I

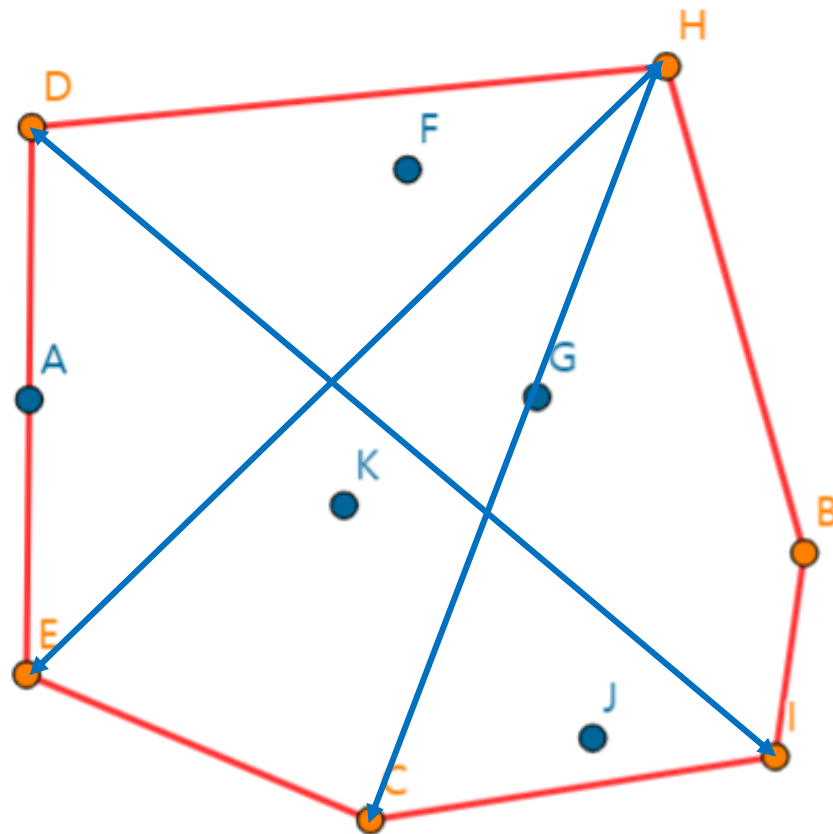
H - E

H - C



가장 먼 두 점 고르기

두 점이 Convex Hull 위에
존재 해야 하는 것은 자명해 보임.

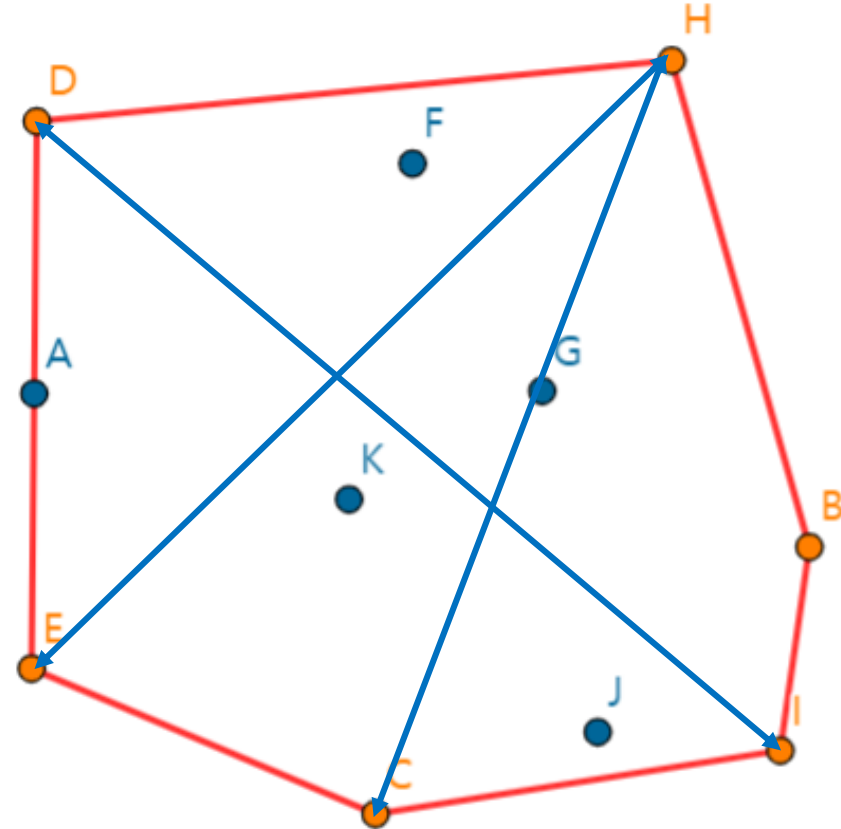




가장 먼 두 점 고르기

두 점이 Convex Hull 위에
존재 해야 하는 것은 자명해 보임.

⇒ 그렇다면 Convex Hull을
잘 돌아보면 되지 않을까?





Rotating Calipers



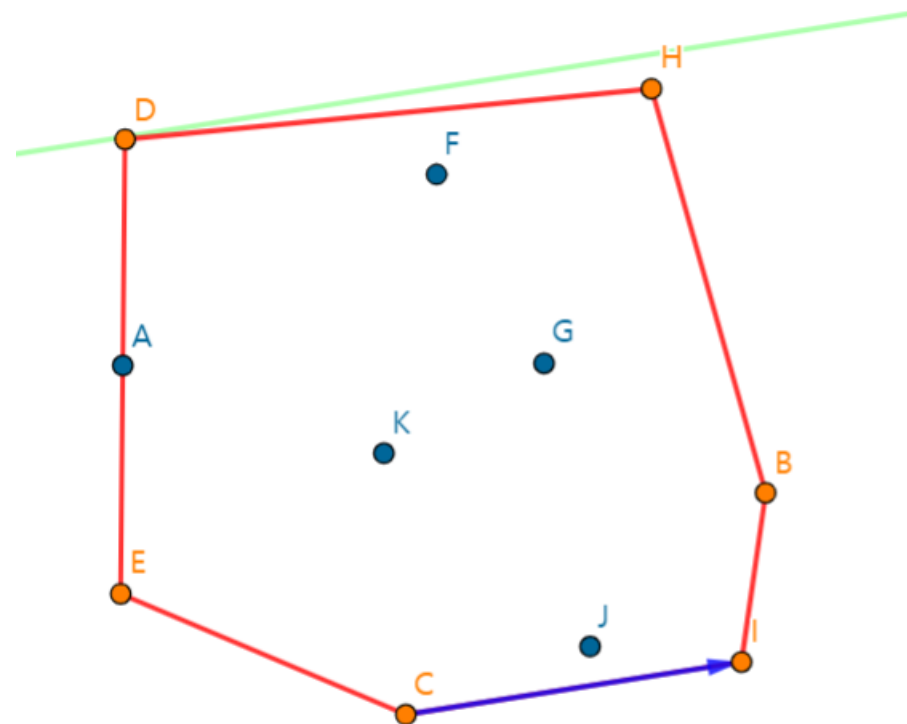
Calipers ...?



Rotating Calipers

convex hull 위의 한 선분에 대해 평행한 선을
convex hull 맞은편에 접하도록 그려보자.

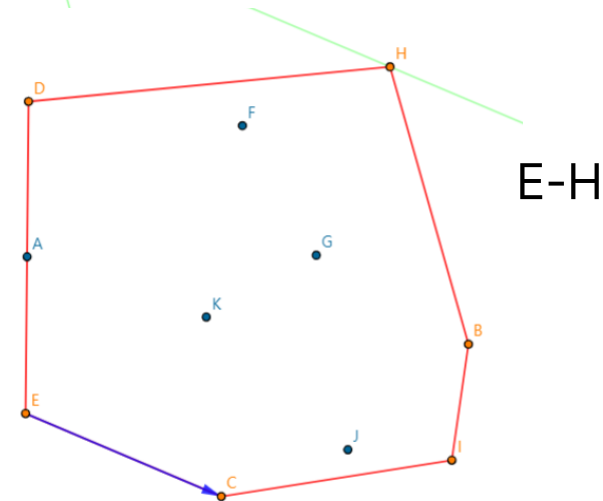
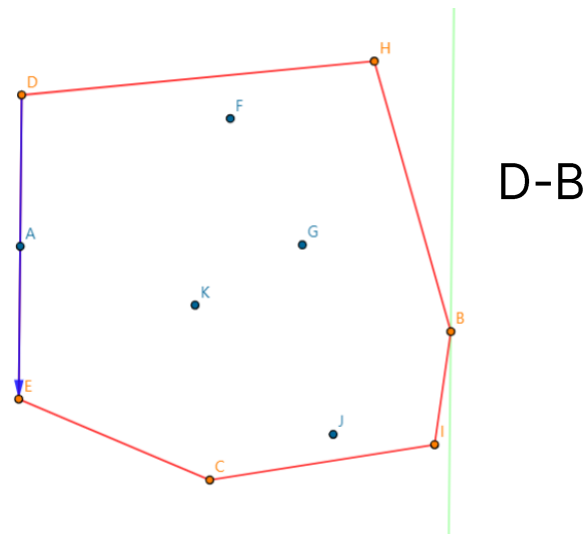
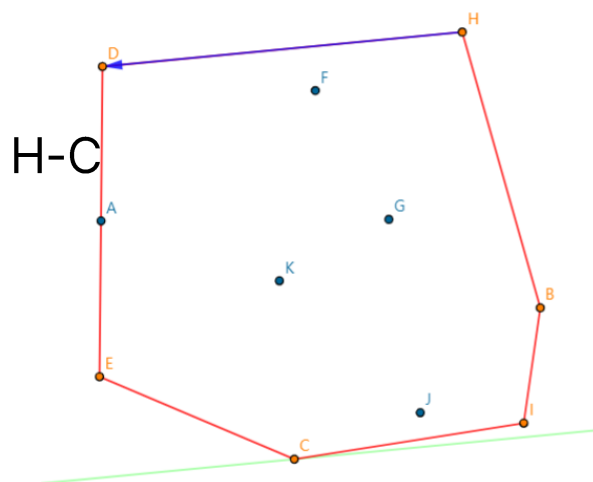
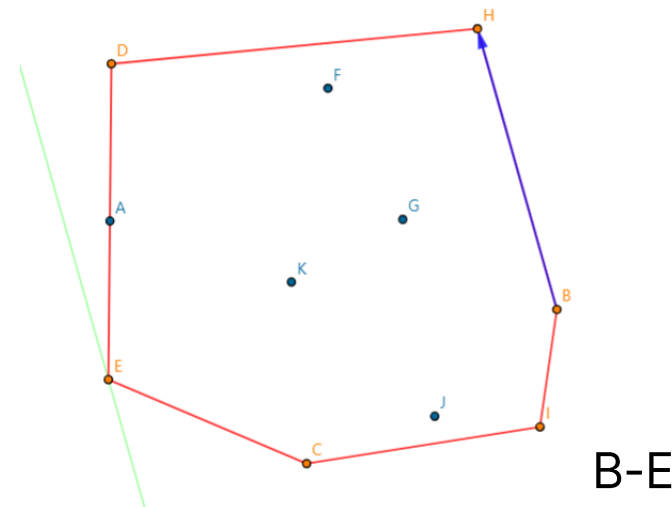
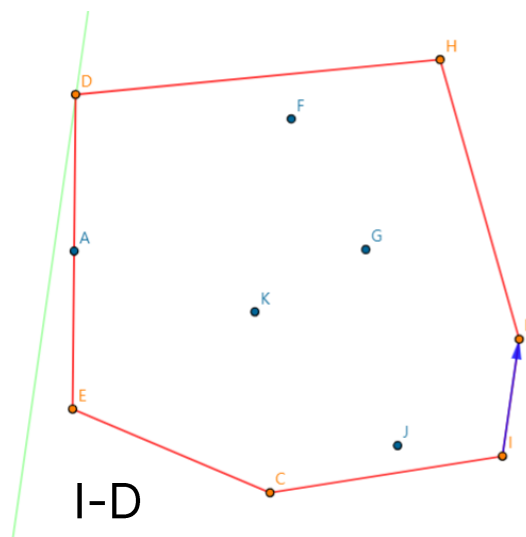
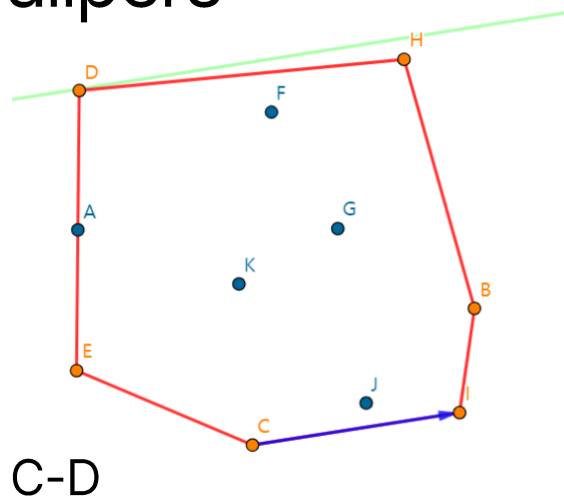
만나는 점이 있거나 평행한 선이 있을 경우,
만나는 선이 있겠다.





Rotating Calipers

요놈들 중에 있다





Rotating Calipers

- 모든 점에서 “한 쪽 방향”으로 돌면서 맞은편의 평행선과 만나는 점을 찾는다.

= 현재 점에서 나가는 벡터와 그 이후의 점들의 벡터의
ccw값이 양 \rightarrow 0 or 음이 되는 점 찾기
- 투 포인터 느낌으로 한바퀴 쪽 돌면 된다.



Rotating Calipers

- 60 ~ 62 line

현재 벡터와 이후의 벡터의 ccw가 양수면 이후 벡터를 계속 이동

- 64 line

최대 거리 갱신

⇒ $O(n)$ 으로 해결 가능

(하지만 convex hull 이 $O(n \log n)$ 이었음)


```
55 void rotating_calipers() {
56     int m = CH.size();
57     ll ret = -1;
58     for (int i = 0, j = 1; i < m; ++i) {
59         // ccw_(a, b, c, d) : a->b 벡터와 c->d 벡터의 방향성
60         while (ccw_(P[CH[i]], P[CH[(i + 1) % m]],
61                     P[CH[j]], P[CH[(j + 1) % m]]) > 0) {
62             j = (j + 1) % m;
63         }
64         ret = max(ret, dist(P[CH[i]], P[CH[j]]));
65     }
66 }
```




[ccw]

 11758 ccw

 2166 다각형의 면적


 2162 선분 그룹


 10255 교차점


[Convex Hull]

 1708 볼록 껍질


 10903 Wall construction

 2254 감옥건설

 3679 단순 다각형

 3878 점 분리

[Rotating Calipers]

 9240 로버트 후드