고급 소프트웨어 실습(CSE4152)

2주차

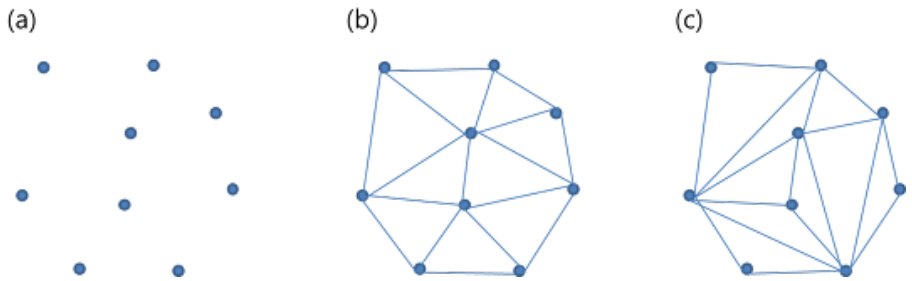
Delaunay triangulation (들로네 삼각분할)

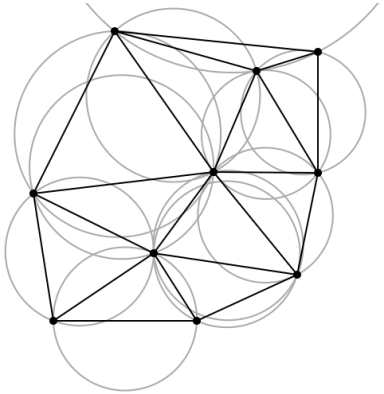
목차

- Delaunay triangulation
- Voronoi Diagram (Voronoi tessellation)
- 활용 분야
- 실습 문제
- 과제

Delaunay Triangulation (들로네 삼각분할)

- 평면위의 점들을 삼각형으로 연결하여 공간을 분할할 때, 이 삼각형들의 내각의 최소값이 최대가 되도록 하는 분할



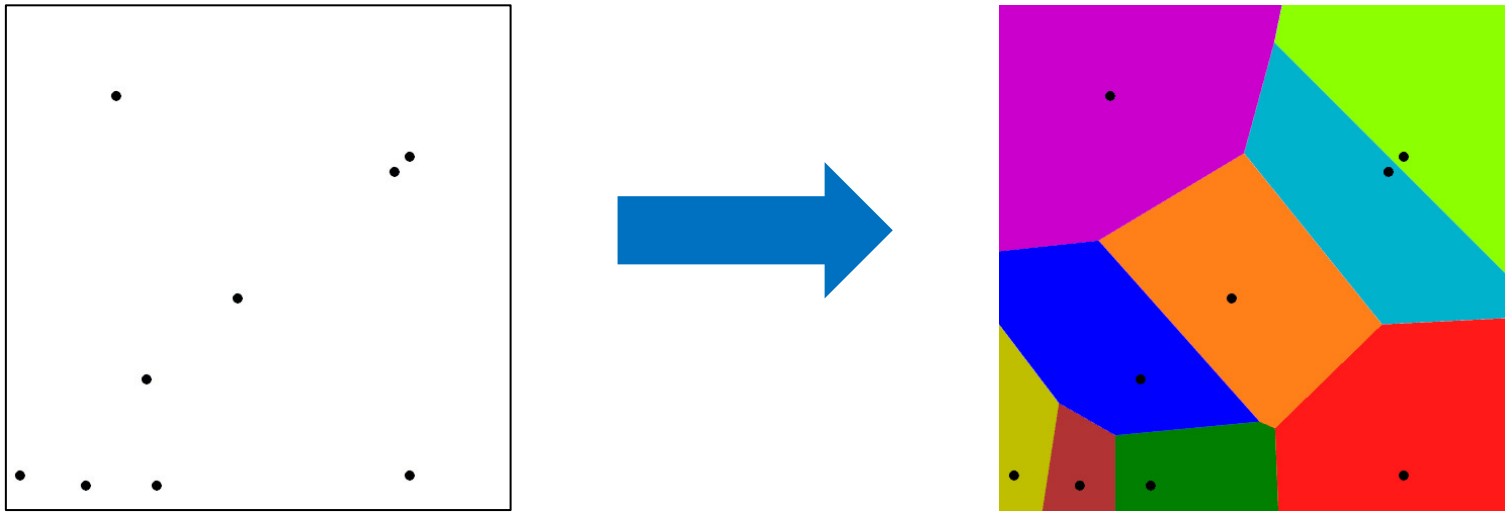


외접원이 표시된
들로네 삼각분할

- (a)와 같이 점들이 있을 때 이 점들을 연결하여 삼각형을 만드는 방법은 다양. 들로네 삼각분할은 여러 삼각분할 중에서 (b)와 같이 각각의 삼각형들이 최대한 정삼각형에 가까운 즉, (c)와 같이 길쭉하고 홀쭉한 삼각형이 나오지 않도록 하는 분할
- 들로네 삼각분할의 가장 중요한 특징은 “어떤 삼각형의 외접원도 그 삼각형의 세 꼭지점을 제외한 다른 어떤 점도 포함하지 않는다“ (empty circumcircle property)
- 삼각형이 홀쭉하고 길수록 외접원도 커짐. 위 그림에서 (b)는 이 조건을 만족하고 (c)는 만족하지 않음

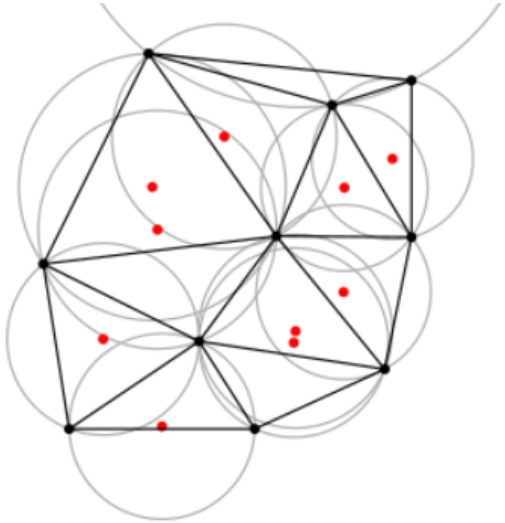
Voronoi Diagram (보로노이 다이어그램)

- 시드 점(seed point)들과의 거리에 따라서 평면을 분할한 다이어그램
- 평면 위에 주어진 시드 점 $p_1, p_2, \ldots, p_n$ 에 대해, 평면 위의 점들이 어떤 p_i 와 가장 가까운지에 따라 영역을 분할한 다이어그램

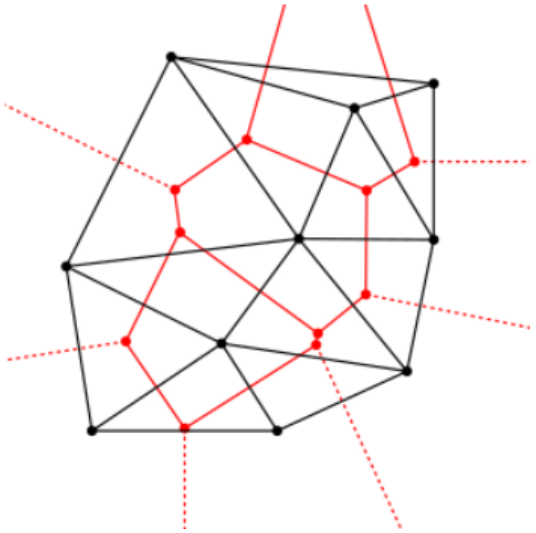


들로네 삼각분할과 보로노이 다이어그램의 관계

- 들로네 삼각분할과 보로노이 다이어그램은 서로 듀얼 (dual) 관계에 있으며 어느 하나를 알면 다른 하나도 구할 수 있음



모든 외접원과 그 중심(빨간색)이 표시된 들로네 삼각분할



외접원의 중심을 연결하면 보로노이 다이어그램 (빨간색 다각형)

- [들로네 -> 보로노이]

각 seed 점들을 가지고 들로네 삼각분할 후, 구해진 삼각형들의 외접원들의 중심을 연결하면 보로노이 다이어그램을 얻을 수 있음

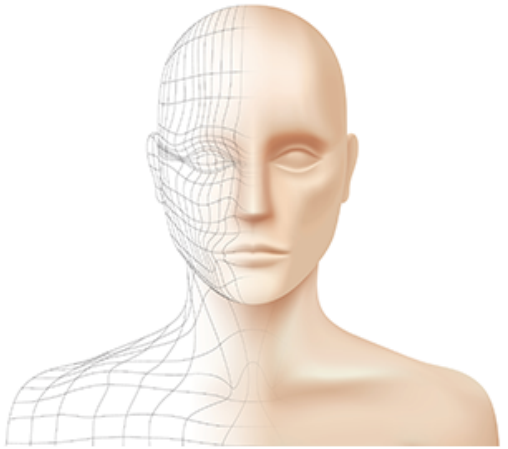
- [보로노이 -> 들로네]

보로노이 영역들 간의 seed 점들을 연결하면, 이 seed 점들에 대한 들로네 삼각분할을 얻을 수 있음

들로네 삼각분할 활용

- 입체의 표면을 다각형으로 모델링 (3D 모델링)

물체의 표면으로부터 많은 점을 얻은 후, 이 점들을 기준으로 삼각분할하면 입체적인 모양을 가장 그럴듯하게 표현할 수 있음. 들로네 삼각분할은 정삼각형에 가깝도록 세 점을 잡아 다각형 평면을 표현할 수 있으며, 이러한 원리 때문에 일정한 크기로 면적을 나눌 수 있어 3D 모델링에 활용됨.

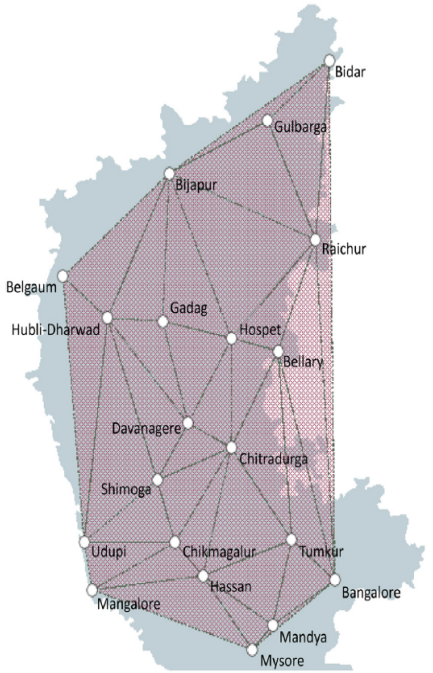


3D 모델링

- 도로망 설계

n개의 도시를 연결할 때, 모든 도시와 도시를 직접 직선도로로 연결할 경우 모두 nC2=n(n-1)/2개의 도로를 건설해야 함. 실제로는 일부 도로만을 연결하여 전체 도시들이 연결되도록 할 수 있음. 도시 A에서 도시 B로 이동할 때, 직접 연결된 도로가 없다면 다른 도시들을 우회해서 이동 가능.

이때 도시 A에서 도시 B로 이동하기 위해 우회한 총 이동거리가 A-B 사이의 직선거리와 비슷하게 하려면, 들로네 삼각분할을 이용 가능. 들로네 삼각분할은 길고 홀쭉한 삼각형이 나오지 않도록 최대한 가까운 세 점을 잡아 평면을 나누므로 우회에 따른 이동거리의 증가를 최소화할 수 있다



도로망 설계

보로노이 다이어그램 활용

- 보로노이 다이어그램의 가장 중요한 조건은 최대한 가까운 두 점을 수직 이등분선을 이용해 점이 꼭 하나씩 포함되도록 평면을 분할해야 한다는 것임. 이러한 특징을 활용한 사례는 아래와 같음.

- 편의시설 찾아주는 프로그램

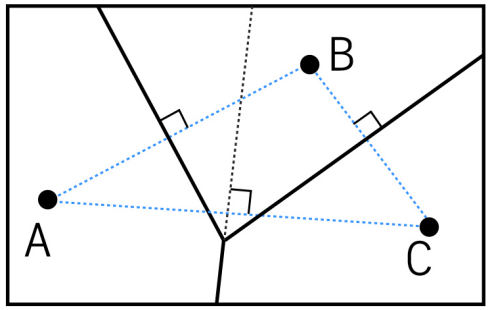
  내가 위치한 곳에서 가장 가까운 병원, 약국과 같은 편의 시설을 찾아주는 프로그램에도 활용됨. 지리적으로 내가 있는 위치를 기준점으로 하고, 각각의 점들은 생활에 필요한 서비스를 제공해주는 장소의 위치로 생각할 수 있음.

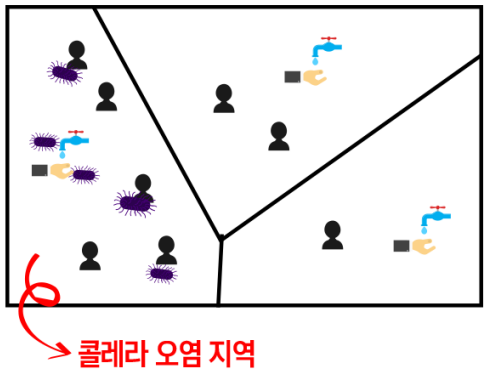
- 코로나 19 전파

  최초의 감염자를 찾고 그와 가장 가까운 접촉자를 찾는 역학조사에서 '가장 가까운 접촉자(밀접 접촉자)'를 찾을 때 활용.

- 질병 감염지역 분석

  식수 펌프를 기준으로 해당 식수 펌프를 이용하는 사람들을 나누는 영역 구성. 만일 어떤 우물이 오염되었다면 그 영역에 사는 사람은 모두 오염된 물을 먹었을 가능성이 큼. 이러한 원리를 이용해 감염 원인을 추적함.





문제 1. 들로네 삼각분할 구현

주어진 4개의 점에 대해 들로네 삼각 분할을 수행한다.

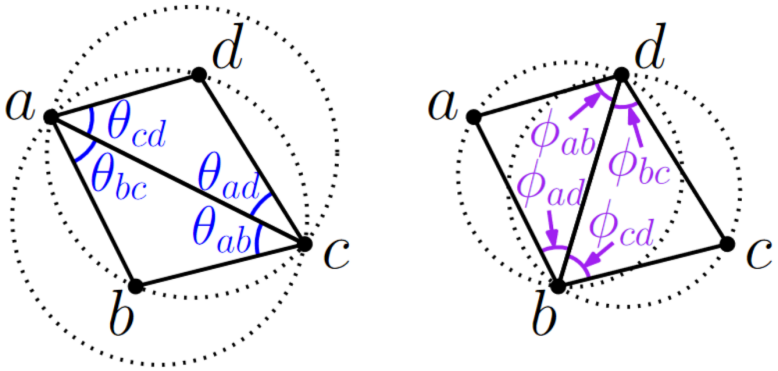
조건1. 어떤 삼각형의 외접원도 다른 점을 포함하지 않는다 (지나갈 수는 있음).

들로네 삼각분할 코드를 완성하기.

- 외접원을 계산하는 circumcircle 함수 구현
- 입력으로 들어온 점이 외접원 내에 있는지 확인하는 isInCircumcircle 함수 구현
- 입력으로 들어온 여러개의 점으로 만들 수 있는 삼각형 조합을 반환하는 combination 함수 구현

- 좌표

A (120,240)
B (370,180)
C (550,460)
D (260,540)



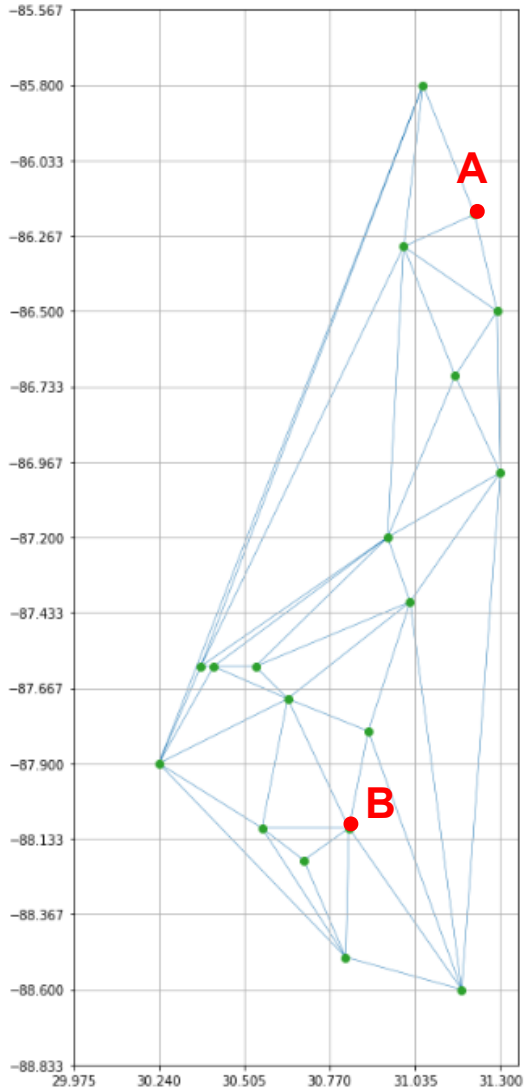
4개의 점에 대해 만들어질 수 있는 삼각 분할

문제 2. 들로네 삼각분할 활용

1) 제공된 삼각분할 알고리즘을 이용하여서 주어진 점들을 잇는 path를 생성하고 이미지로 보여주시오. (오른쪽 이미지)
2) 생성된 path를 이용하여서 A, B 간의 최소 이동경로를 조교에게 보이시오.
3) A, B 간의 최단 거리를 (변위) 조교에게 보이시오.

```
# A : [31.22, -86.2]
# B : [30.83, -88.1]

points = np.array([
[31.3, -87], [31.29, -86.5], [31.22, -86.2], [31.18, -88.6],
[31.16, -86.7], [31.06, -85.8], [31.02, -87.4], [31, -86.3],
[30.95, -87.2], [30.89, -87.8], [30.83, -88.1], [30.82, -88.5],
[30.69, -88.2], [30.64, -87.7], [30.56, -88.1], [30.54, -87.6],
[30.41, -87.6], [30.37, -87.6], [30.24, -87.9]
])
```



문제 3. 보로노이 다이어그램 구현

3.1. voronoid.py의 voronoid 함수 구현

```
#### 구현할 함수 1 ####
def voronoid(points, xmin=None, xmax=None, ymin=None, ymax=None):

    if xmin==None:

        xygraph = Xygraph(vl=points)
        xmin = xygraph.xmin - 1
        xmax = xygraph.xmax + 3
        ymin = xygraph.ymin - 1
        ymax = xygraph.ymax + 1


    points = list(set(points))
    n = len(points)

    v = (xmin, ymin)

    points.sort(key=lambda p: ((p[0]-v[0])**2 +
                               (p[1]-v[1])**2)**(1/2), reverse=True)

    cur_points = []

    p = points.pop()

    cur_points.append(p)
    cur_points = cur_points
```

1) 현재 cur_points 정보를 이용하여서 Nearest Neighbors 학습.
2) 학습된 Nearest Neighbors를 이용하여서 가까운 점의 인덱스를 얻기
3) voronoid[0:5] 출력

답안

```
(-1, 12, -1, 9, array([[0, 1],
       [1, 8],
       [9, 0],
       [9, 4]]))
```

문제 4. 보로노이 다이어그램 활용

4.1. KNN 에서 전체 train set 저장하는 경우, Voronoi 생성 후 nearest neighbor editing 하고 inference 하는 경우, 성능이 동일하게 나오는지 확인

[진행과정]

1) 2개의 클래스로 구성된 랜덤 데이터 생성
   - sklearn 라이브러리의 make_classification 함수 사용

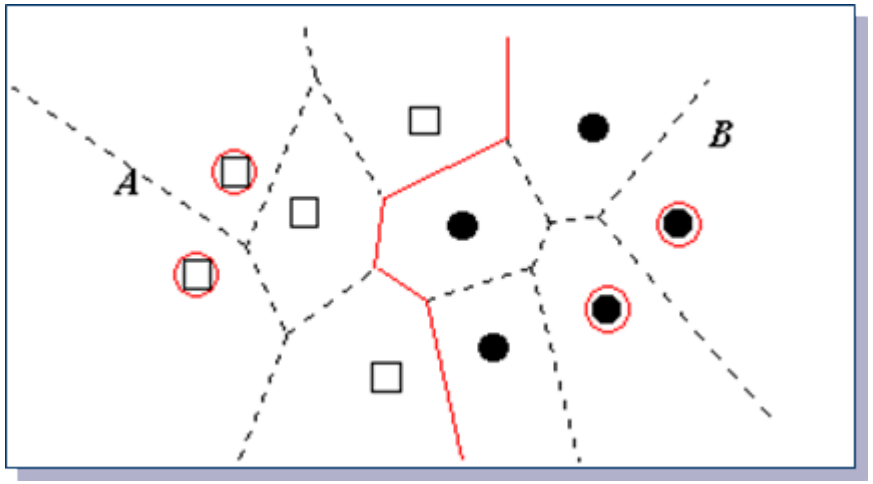
2) 생성한 데이터를 k=1인 knn 분류기에 학습(Classifier1)
   - sklearn 라이브러리의 KNeighborsClassifier 함수 사용

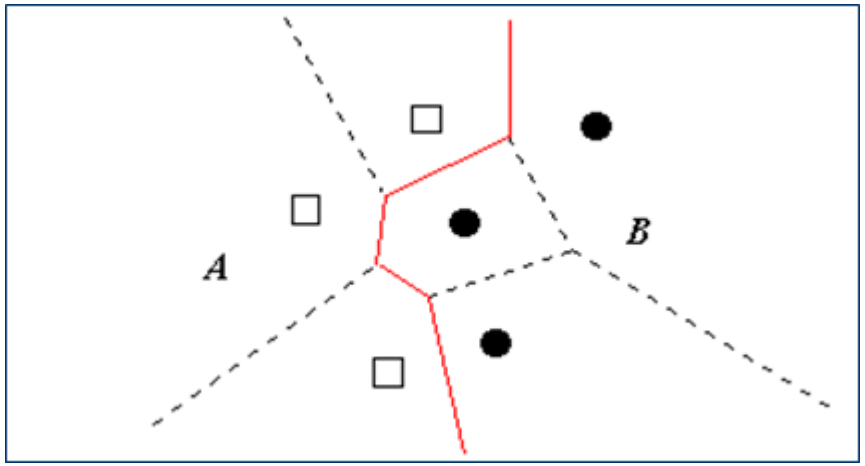
3) 데이터를 사용해 보로노이 다이어그램 생성

4) 보로노이 다이어그램에 대해 nearest neighbor editing 수행

5) 편집된 데이터에 대해 k=1인 knn 분류기에 학습(Classifier2)

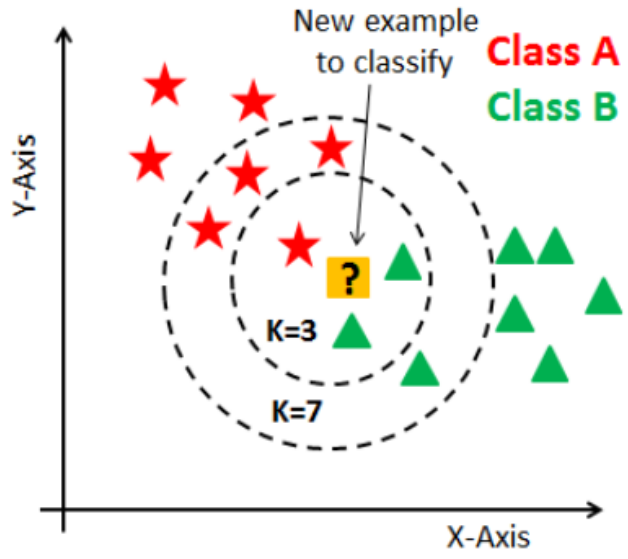
6) 테스트 데이터를 생성해 2개의 분류기에 대한 성능 비교
   - sklearn 라이브러리의 accuracy_score 함수 사용





KNN (K-Nearest Neighbor)

- 정의 : 새로운 데이터가 입력되었을 때, 기존의 데이터와 새로운 데이터를 비교함으로써 새로운 데이터와 가장 인접한 데이터 k개를 선정하여, k개 데이터의 category에 의해 입력 데이터를 분류. 보통 유클리드 거리를 사용



- 장점

1) 단순하고 효율적
2) 학습 데이터의 노이즈에 크게 영향을 받지 않음
3) 훈련 단계가 빠름

- 단점

1) 최적 이웃의 수(K)와 사용할 거리 척도를 데이터 각각의 특성에 맞게 연구자가 임의로 설정
2) 새로운 관측치와 각 학습 데이터 사이의 거리를 전부 측정해야 하므로, 계산이 오래 걸림

과제

- 실습 문제 1에서 간단한 Delaunay triangulation 에 대하여 살펴보았다. 임의의 개수의 포인트 셋에 대한 알고리즘을 수도 코드 형태로 작성해 보시오.