**1. 임의의 개수의 포인트 셋에 대한 알고리즘**

**Pseudo-code**

function delaunayTriangulation(P):

**// 모든 점을 포함하는 큰 삼각형을 생성함**

T = createSuperTriangle(P) 🡨 ***모든 점을 포함하는 큰 삼각형을 생성***

**// 모든 점을 순차적으로 추가하며, 현재 점을 포함하는 모든 삼각형을 찾음**

for each point p in P:

badTriangles = []

for each triangle t in T:

if pointInCircumcircle(p, t): **🡨 *점 p가 주어진 삼각형의 외접원 안에 있는지 확인***

badTriangles.append(t)

**// 조건에 안 맞는 삼각형을 제거하고 경계 다각형을 구성**

polygon = []

for each triangle t in badTriangles:

for each edge e of t:

if e is not shared by any other triangle in badTriangles:

polygon.append(e)

**// 여기서 조건에 안 맞는 삼각형을 제거**

for each triangle t in badTriangles:

T.remove(t)

**// 현재 점과 경계 다각형의 각 변을 연결하여 새로운 삼각형을 생성**

for each edge e in polygon:

newTriangle = createTriangle(p, e) 🡨***점 p를 주어진 가장자리에 연결하여 새 삼각형을 만듦***

T.add(newTriangle)

**// 초기의 super triangle에 포함된 삼각형을 제거하여 최종 Delaunay Triangulation을 반환**

for each triangle t in T:

if t has a vertex in superTriangle:

T.remove(t)

**// Delaunay triangulation을 반환**

return T

**2. Voronoi diagram으로 변환하는 알고리즘**

function voronoiDiagram(P):

**// 주어진 점 집합 P에 대해 Delaunay Triangulation을 수행**

delaunayTriangles = delaunayTriangulation(P) ***🡨 점 집합 P의 Delaunay Triangulation을 반환***

**// 각 점에 대한 Voronoi Cell을 저장할 데이터 구조를 초기화**

voronoiCells = initializeVoronoiCells(P) ***🡨 P의 각 점에 대한* VoronoiCell*을 저장하도록 목록 초기화***

**// 각 Delaunay 삼각형에 대해 외심(circumcenter)을 계산**

for each triangle t in delaunayTriangles:

**// 삼각형의 외심을 계산**

circumcenter = calculateCircumcenter(t) ***🡨 주어진 삼각형의 외심을 계산하고 반환***

**// 삼각형의 각 정점에 대해 외심을 이웃으로 추가하여 Voronoi Cell을 구성**

for each vertex p in t:

voronoiCells[p].addNeighbor(circumcenter)

**// 각 Voronoi Cell의 외심 이웃들 간의 연결을 통해 Voronoi 다이어그램의 모서리를 만듦**

voronoiEdges = []

for each cell in voronoiCells:

for each pair of neighboring circumcenters (c1, c2) in cell:

voronoiEdges.append(createEdge(c1, c2)) 🡨 ***외심 c1과 c2 사이에 모서리를 만듦***

**// Voronoi diagram을 간선 목록 또는 셀로 반환**

return voronoiEdges, voronoiCells