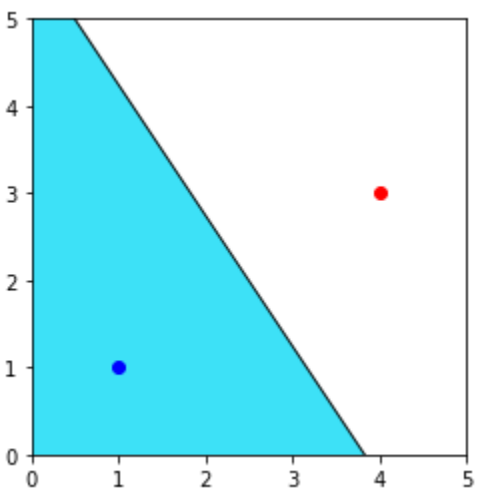
Project 3: Decision Boundary in k-NN Classification through Voronoi diagrams

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.spatial import Voronoi, voronoi_plot_2d

#vectorized Function to set color regions
def setRegionsFunc(j, colors):
    if not -1 in vorReg[vorPt[j]]:
        plt.fill(*zip(*[vor.vertices[i] for i in vorReg[vorPt[j]]]), colors)

setRegions = np.vectorize(setRegionsFunc)
```

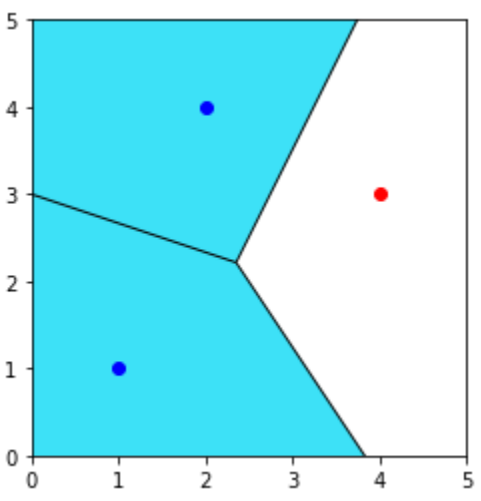
Plot 1: Red (4,3) | Blue (1,1)

In [2]:
```
#Points to plot
rpoints = np.array([[4, 3]])
bpoints = np.array([[1, 1]])

#Point Array creation (add 4 dummy points to extend the graph axes)
dummy = np.array([[999,999], [-999,999], [999,-999], [-999,-999]])
points = np.append(dummy, np.append(rpoints, bpoints, axis = 0), axis = 0)
ptColors = np.array(["red", "red", "red", "red", "red", "blue"])

#color region specifics
size = (len(points)) - (len(bpoints))
arr = np.arange(len(points))
colors = list(map(lambda t: '#3de1f7' if (t >= size) else 'w', arr))

#plot the voronoi diagram
vor = Voronoi(points)
fig = voronoi_plot_2d(vor, show_vertices = False)
vorPt = vor.point_region
vorReg = vor.regions

#color the regions
setRegions(np.arange(len(points)), colors)

#Plot points and constrict display
completePts = np.array(list(zip(points,ptColors)), dtype=object)
list(map(lambda t: plt.plot(t[0][0], t[0][1], 'o', color=t[1]), completePts))
plt.axis('square')
plt.xlim(0, 5), plt.ylim(0, 5)

#show plot
plt.show()
```



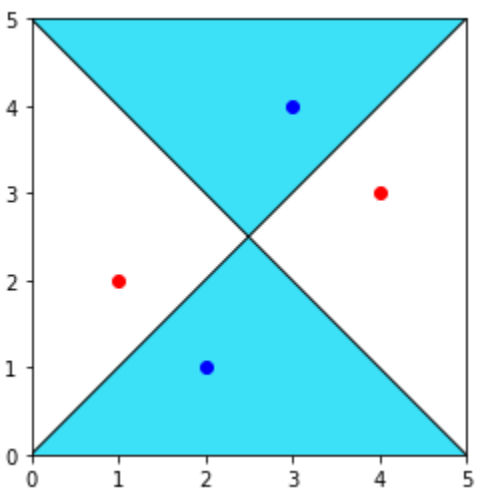
Plot 2: Red (4,3) | Blue (1,1), (2,4)

In [3]:
```
#Points to plot
rpoints = np.array([[4, 3]])
bpoints = np.array([[1, 1], [2, 4]])

#Point Array creation (add 4 dummy points to extend the graph axes)
dummy = np.array([[999,999], [-999,999], [999,-999], [-999,-999]])
points = np.append(dummy, np.append(rpoints, bpoints, axis = 0), axis = 0)
ptColors = np.array(["red", "red", "red", "red", "red", "blue", "blue"])

#color region specifics
size = (len(points)) - (len(bpoints))
arr = np.arange(len(points))
colors = list(map(lambda t: '#3de1f7' if (t >= size) else 'w', arr))

#plot the voronoi diagram
vor = Voronoi(points)
fig = voronoi_plot_2d(vor, show_vertices = False)
vorPt = vor.point_region
vorReg = vor.regions

#color the regions
setRegions(np.arange(len(points)), colors)

#Plot points and constrict display
completePts = np.array(list(zip(points,ptColors)), dtype=object)
list(map(lambda t: plt.plot(t[0][0], t[0][1], 'o', color=t[1]), completePts))
plt.axis('square')
plt.xlim(0, 5), plt.ylim(0, 5)

#show plot
plt.show()
```



Plot 3: Red (1,2), (4,3) | Blue (2,1), (3,4)

In [4]:
```
#Points to plot
rpoints = np.array([[1, 2], [4, 3]])
bpoints = np.array([[3, 4], [2, 1]])

#Point Array creation (add 4 dummy points to extend the graph axes)
dummy = np.array([[999,999], [-999,999], [999,-999], [-999,-999]])
points = np.append(dummy, np.append(rpoints, bpoints, axis = 0), axis = 0)
ptColors = np.array(["red", "red", "red", "red", "red", "red", "blue", "blue"])

#color region specifics
size = (len(points)) - (len(bpoints))
arr = np.arange(len(points))
colors = list(map(lambda t: '#3de1f7' if (t >= size) else 'w', arr))

#plot the voronoi diagram
vor = Voronoi(points)
fig = voronoi_plot_2d(vor, show_vertices = False)
vorPt = vor.point_region
vorReg = vor.regions

#color the regions
setRegions(np.arange(len(points)), colors)

#Plot points and constrict display
completePts = np.array(list(zip(points,ptColors)), dtype=object)
list(map(lambda t: plt.plot(t[0][0], t[0][1], 'o', color=t[1]), completePts))
plt.axis('square')
plt.xlim(0, 5), plt.ylim(0, 5)

#show plot
plt.show()
```



Decision Boundary Reasoning

The plots were created using the Voronoi function from the SciPy package. The package partitions the points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than to any other. The distance of the lines between the points was determined using the Euclidean distance. The polygons are creating using the equation: V (p_i) = {x s.t. k x − xi k≤k x − xj k for j 6= i, j ∈ In}. Simply put: The midpoint between any two points in question is found, and then a line that is perpendicular to the line formed by the two points in question and the midpoint is drawn, and that is the dividing boundary between the polygons.