分治法：

1. **import** random
2. **import** time
3. **from** tkinter.simpledialog **import** askstring, askinteger, askfloat
4. **import** numpy as np
5. **import** matplotlib.pyplot as plt
6. **from** tkinter **import** \*
7. **import** tkinter as tk


11. **class** Convex\_Hull():
12. **def** \_\_init\_\_(self):
13. **pass**
15. # 算面积
16. **def** calc\_area(self,a, b, c):
17. """
18. 判断三角形面积
19. """
20. x1, y1 = a
21. x2, y2 = b
22. x3, y3 = c
23. **return** x1 \* y2 + x3 \* y1 + x2 \* y3 - x3 \* y2 - x2 \* y1 - x1 \* y3
25. #  生成随机点
26. **def** rand\_point\_set(self,n, range\_min=0, range\_max=101):
27. **try**:
28. **return** list(zip([random.uniform(range\_min, range\_max) **for** \_ **in** range(n)],
29. [random.uniform(range\_min, range\_max) **for** \_ **in** range(n)]))
30. **except** IndexError as e:
31. **print**("\033[31m" + ''.join(e.args) + "\n输入范围有误！" + '\033[0m')
33. **def** AreaOfUp(self,left, right, lists, boundary):
34. area\_max = 0
35. max\_point = ()
36. **for** item **in** lists:
37. **if** item == left **or** item == right:
38. **continue**
39. **else**:
40. max\_point = item **if** Object.calc\_area(left, right, item) > area\_max **else** max\_point
41. area\_max = Object.calc\_area(left, right, item) **if** Object.calc\_area(left, right, item) > area\_max **else** area\_max
42. **if** area\_max != 0:
43. boundary.append(max\_point)
44. Object.AreaOfUp(left, max\_point, lists, boundary)
45. Object.AreaOfUp(max\_point, right, lists, boundary)
47. **def** AreaOfDown(self,left, right, lists, boundary):
48. area\_max = 0
49. max\_point = ()
50. **for** item **in** lists:
51. **if** item == left **or** item == right:
52. **continue**
53. **else**:
54. max\_point = item **if** Object.calc\_area(left, right, item) < area\_max **else** max\_point
55. area\_max = Object.calc\_area(left, right, item) **if** Object.calc\_area(left, right, item) < area\_max **else** area\_max
56. **if** area\_max != 0:
57. boundary.append(max\_point)
58. Object.AreaOfDown(left, max\_point, lists, boundary)
59. Object.AreaOfDown(max\_point, right, lists, boundary)
61. **def** order\_border(self,lists):
62. lists.sort()
63. first\_x, first\_y = lists[0]  # 最左边的点
64. last\_x, last\_y = lists[-1]  # 最右边的点
65. list\_border\_up = []  # 上半边界
66. **for** item **in** lists:
67. x, y = item
68. **if** y > max(first\_y, last\_y):
69. list\_border\_up.append(item)
70. **if** min(first\_y, last\_y) < y < max(first\_y, last\_y):
71. **if** Object.calc\_area(lists[0], lists[-1], item) > 0:
72. list\_border\_up.append(item)
73. **else**:
74. **continue**
75. list\_border\_down = [\_ **for** \_ **in** lists **if** \_ **not** **in** list\_border\_up]  # 下半边界
76. list\_end = list\_border\_up + list\_border\_down[::-1]  # 最终顺时针输出的边界点
77. **return** list\_end
79. """可视化"""
81. **def** print\_integer(self):
82. res = askinteger("Spam", "Egg count", initialvalue=12 \* 12)
83. **return** res
85. **def** display(self,list\_points, boundary):
86. root = Tk()
87. root.config(bg='#87CEEB')
88. root.title("202018526高树林的凸包可视化")
89. cv = Canvas(root, bg="white", width=800, height=800)
90. cv.pack()
91. # x轴
92. **for** i **in** range(11):
93. x = 100 + (i \* 60)
94. # y轴
95. **for** i **in** range(6):
96. y = 500 - (i \* 80)
97. scaled = []
98. all\_point = []
99. **for** x, y **in** boundary:
100. scaled.append((100 + 6 \* x, 500 - 8 \* y / 5))
101. **for** x, y **in** list\_points:
102. all\_point.append((100 + 6 \* x, 500 - 8 \* y / 5))
103. scaled.append((boundary[0][0] \* 6 + 100, 500 - 8 \* boundary[0][1] / 5))
104. cv.create\_line(scaled, fill='green')
105. **for** x, y **in** all\_point:
106. cv.create\_oval(x - 6, y - 6, x + 6, y + 6, width=1, outline='black', fill='red')
107. cv.create\_text(350, 560, text='\t高树林使用Tkinter做的可视化', fill='black', font='SimHei 20 bold')
108. root.mainloop()
110. **def** main(self):
111. """
112. :return: 所有点
113. """
114. root = tk.Tk()
115. tk.Button(root, text='取一个整数', command=Object.print\_integer).pack()
116. # inputs = list(map(int, input().split()))
117. inputs = Object.print\_integer()
118. **return** Object.rand\_point\_set(inputs)

121. **if** \_\_name\_\_ == "\_\_main\_\_":
122. Object = Convex\_Hull()
123. list\_points = Object.main()  # 所有点
124. # print(list\_points)
125. list\_points.sort()
126. border\_points = []  # 边界点集
127. Object.AreaOfUp(list\_points[0], list\_points[-1], list\_points, border\_points)  # 上边界点集
128. Object.AreaOfDown(list\_points[0], list\_points[-1], list\_points, border\_points)  # 下边界点集
129. border\_points.append(list\_points[0])
130. border\_points.append(list\_points[-1])  # 将首尾两个点添加到边界点集中
131. **print**(Object.order\_border(border\_points))  # 顺时针输出边界点
132. Object.display(list\_points, Object.order\_border(border\_points))

Graham算法：

1. **import** math
2. **import** matplotlib.pyplot as plt
3. **import** numpy as np

6. **class** Convex\_Hull():
7. **def** \_\_init\_\_(self):
8. **pass**
10. **def** atan(self, point, y, x):
11. x = x - point[0]
12. y = y - point[1]
13. **if** x == 0 **and** y == 0:
14. **return** 0
15. point = (5, 0)  # 表示x轴的向量，随便取
16. cos = (point[0] \* x + point[1] \* y) / (math.sqrt(point[0] \*\* 2 + point[1] \*\* 2) \* math.sqrt(x \*\* 2 + y \*\* 2))
17. **return** np.arccos(cos) \* (180 / math.pi)
19. **def** angle\_sort(self, p0, points):
20. dic = {}
21. **for** point **in** points:
22. angle = self.atan(p0, point[1], point[0])
23. dic[point] = angle
24. points = [k[0] **for** k **in** sorted(dic.items(), key=**lambda** x: x[1])]  # ,reverse=True
25. **return** points
27. **def** cross\_product(self, a, b, c):
28. '''''判断点c在由点a,b构成的向量的那一侧'''
29. result = a[0] \* b[1] - a[1] \* b[0] + b[0] \* c[1] - b[1] \* c[0] + c[0] \* a[1] - c[1] \* a[0]
30. **if** result < 0:
31. **return** False  # 点c在向量ab右边 返回False
32. **else**:
33. **return** True  # 点c在向量ab左边  返回True
35. **def** draw(self, x, y, x0, y0):
36. plt.figure(figsize=(10, 10))
37. plt.scatter(x, y)
38. plt.plot(x0, y0)
39. plt.xlim(-1, 11)
40. plt.ylim(-1, 6)
41. i = 0
42. plt.show()

45. **if** \_\_name\_\_ == '\_\_main\_\_':
46. Object = Convex\_Hull()
47. key = 3
48. num = 300
49. level = 10
50. vertical = 5
51. seed = np.random.RandomState(key)
52. seed2 = np.random.RandomState(key + 1)
53. Z1 = seed.rand(num, 1) \* level  # 生成点集
54. Z2 = seed2.rand(num, 1) \* vertical
55. Z = np.concatenate([Z1, Z2], axis=1)
56. lists\_points = [tuple(i) **for** i **in** Z]
58. # 起点为y坐标最小的点
59. ymin = min(lists\_points, key=**lambda** x: x[1])[1]
60. start = min([i **for** i **in** lists\_points **if** i[1] == ymin], key=**lambda** x: x[0])
61. boundary = []
62. lists\_points = Object.angle\_sort(start, lists\_points)
63. boundary.append(lists\_points[0])
64. boundary.append(lists\_points[1])
65. i = 2
66. **while** len(boundary) != 0 **and** i != len(lists\_points):
67. **if** Object.cross\_product(boundary[len(boundary) - 2], boundary[len(boundary) - 1], lists\_points[i]):
68. boundary.append(lists\_points[i])
69. i += 1
70. **else**:
71. boundary.pop()
72. **if** len(boundary) < 2:
73. boundary.append(lists\_points[i])
74. i += 1
75. **continue**
76. boundary.append(boundary[0])
77. x = [i[0] **for** i **in** lists\_points]
78. y = [i[1] **for** i **in** lists\_points]
79. x0 = [i[0] **for** i **in** boundary]
80. y0 = [i[1] **for** i **in** boundary]
81. Object.draw(x, y, x0, y0)