分治法:

```
import random
import time
from tkinter.simpledialog import askstring, askinteger, askfloat
import numpy as np
import matplotlib.pyplot as plt
from tkinter import *
import tkinter as tk


class Convex_Hull():
    def __init__(self):
        pass

    # 算面积
    def calc_area(self,a, b, c):
        """
        判断三角形面积
        """
        x1, y1 = a
        x2, y2 = b
        x3, y3 = c
        return x1 * y2 + x3 * y1 + x2 * y3 - x3 * y2 - x2 * y1 - x1 * y3

    #  生成随机点
    def rand_point_set(self,n, range_min=0, range_max=101):
        try:
            return list(zip([random.uniform(range_min, range_max) for _ in range(n)],
                            [random.uniform(range_min, range_max) for _ in range(n)]))
        except IndexError as e:
            print("\033[31m" + ''.join(e.args) + "\n输入范围有误!" + '\033[0m')

    def AreaOfUp(self,left, right, lists, boundary):
        area_max = 0
        max_point = ()
        for item in lists:
            if item == left or item == right:
                continue
            else:
```

```
                max_point = item if Object.calc_area(left, right, item) > area_max else max_point
                area_max = Object.calc_area(left, right, item) if Object.calc_area(left, right, item) > area_max else area_max
        if area_max != 0:
            boundary.append(max_point)
            Object.AreaOfUp(left, max_point, lists, boundary)
            Object.AreaOfUp(max_point, right, lists, boundary)

    def AreaOfDown(self,left, right, lists, boundary):
        area_max = 0
        max_point = ()
        for item in lists:
            if item == left or item == right:
                continue
            else:
                max_point = item if Object.calc_area(left, right, item) < area_max else max_point
                area_max = Object.calc_area(left, right, item) if Object.calc_area(left, right, item) < area_max else area_max
        if area_max != 0:
            boundary.append(max_point)
            Object.AreaOfDown(left, max_point, lists, boundary)
            Object.AreaOfDown(max_point, right, lists, boundary)

    def order_border(self,lists):
        lists.sort()
        first_x, first_y = lists[0]  # 最左边的点
        last_x, last_y = lists[-1]  # 最右边的点
        list_border_up = []  # 上半边界
        for item in lists:
            x, y = item
            if y > max(first_y, last_y):
                list_border_up.append(item)
            if min(first_y, last_y) < y < max(first_y, last_y):
                if Object.calc_area(lists[0], lists[-1], item) > 0:
                    list_border_up.append(item)
                else:
                    continue
        list_border_down = [_ for _ in lists if _ not in list_border_up]  # 下半边界
        list_end = list_border_up + list_border_down[::-1]  # 最终顺时针输出的边界点
        return list_end
```

```
78.
79.     """可视化"""
80.
81.     def print_integer(self):
82.         res = askinteger("Spam", "Egg count", initialvalue=12 * 12)
83.         return res
84.
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 * boundar
    y[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, ou
    tline='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()
109.
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()
```

```
            return Object.rand_point_set(inputs)


    if __name__ == "__main__":
        Object = Convex_Hull()
        list_points = Object.main()  # 所有点
        # print(list_points)
        list_points.sort()
        border_points = []  # 边界点集
        Object.AreaOfUp(list_points[0], list_points[-1], list_points, border_points)  # 上边界点集
        Object.AreaOfDown(list_points[0], list_points[-1], list_points, border_points)  # 下边界点集
        border_points.append(list_points[0])
        border_points.append(list_points[-1])  # 将首尾两个点添加到边界点集中
        print(Object.order_border(border_points))  # 顺时针输出边界点
        Object.display(list_points, Object.order_border(border_points))
```

Graham 算法：

```
import math
import matplotlib.pyplot as plt
import numpy as np


class Convex_Hull():
    def __init__(self):
        pass

    def atan(self, point, y, x):
        x = x - point[0]
        y = y - point[1]
        if x == 0 and y == 0:
            return 0
        point = (5, 0)  # 表示 x 轴的向量，随便取
        cos = (point[0] * x + point[1] * y) / (math.sqrt(point[0] ** 2 + point[1] ** 2) * math.sqrt(x ** 2 + y ** 2))
        return np.arccos(cos) * (180 / math.pi)

    def angle_sort(self, p0, points):
        dic = {}
        for point in points:
            angle = self.atan(p0, point[1], point[0])
            dic[point] = angle
        points = [k[0] for k in sorted(dic.items(), key=lambda x: x[1])]  # ,reverse=True
        return points

    def cross_product(self, a, b, c):
        '''''判断点 c 在由点 a,b 构成的向量的那一侧'''
        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]
        if result < 0:
            return False  # 点 c 在向量 ab 右边 返回 False
        else:
            return True  # 点 c 在向量 ab 左边  返回 True

    def draw(self, x, y, x0, y0):
        plt.figure(figsize=(10, 10))
        plt.scatter(x, y)
        plt.plot(x0, y0)
        plt.xlim(-1, 11)
        plt.ylim(-1, 6)
```

```
        i = 0
        plt.show()


if __name__ == '__main__':
    Object = Convex_Hull()
    key = 3
    num = 300
    level = 10
    vertical = 5
    seed = np.random.RandomState(key)
    seed2 = np.random.RandomState(key + 1)
    Z1 = seed.rand(num, 1) * level  # 生成点集
    Z2 = seed2.rand(num, 1) * vertical
    Z = np.concatenate([Z1, Z2], axis=1)
    lists_points = [tuple(i) for i in Z]

    # 起点为 y 坐标最小的点
    ymin = min(lists_points, key=lambda x: x[1])[1]
    start = min([i for i in lists_points if i[1] == ymin], key=lambda
     x: x[0])
    boundary = []
    lists_points = Object.angle_sort(start, lists_points)
    boundary.append(lists_points[0])
    boundary.append(lists_points[1])
    i = 2
    while len(boundary) != 0 and i != len(lists_points):
        if Object.cross_product(boundary[len(boundary) - 2], boundary
   [len(boundary) - 1], lists_points[i]):
            boundary.append(lists_points[i])
            i += 1
        else:
            boundary.pop()
            if len(boundary) < 2:
                boundary.append(lists_points[i])
                i += 1
            continue
    boundary.append(boundary[0])
    x = [i[0] for i in lists_points]
    y = [i[1] for i in lists_points]
    x0 = [i[0] for i in boundary]
    y0 = [i[1] for i in boundary]
    Object.draw(x, y, x0, y0)
```