Part 1 (Jarvis march algorithm)

Computational Complexity:

The total run time is O(Nh),where N is the size of input data , h is the size of the output(the subset of points that lie on the convex hull). $h \leq N$

Random input case:

Theoretically: O(Nh), h depends on how the function ' random _data' define.

```
import random
#code for random data generation
def random_data(N):
    inputlista = [[random.randint(0, 32767) for j in range(1, 3)] for i in range(1,N)] #get N pairs random points
    return inputlista
```

Experimentally:

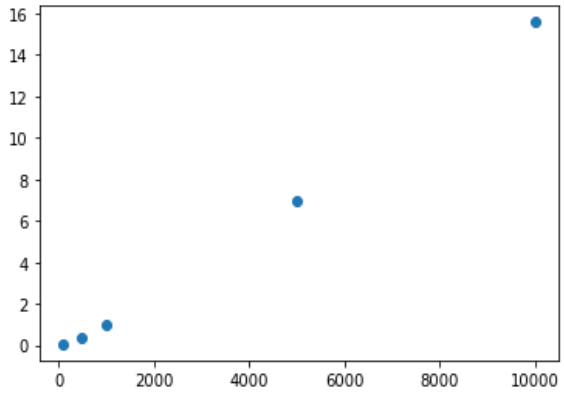
N = 100 , 500 , 1000, 5000, 10000

h = 10 ,10 ,21 ,24 ,17

(note:Because of the randomly generated number, when N is large, there will be more repetition, resulting in errors.)

execution time (s) = 0.05751770000642864, 0.388366600003792, 0.9850992000065162, 6.978307500001392, 15.574939099999028

Scatter plots (x-axis is N , y-axis is the measured execution time):



Conclusion:

With the range of x-y scale of input points unchanged ([0,32767]) , the value of h does not change much when N takes 100,500,1000,50000,10000. The value of h is much less than N, and h is not proportional to N.

The worst-case:

Theoretically: h = N $\rightarrow O(N^2)$

Experimentally: ' h = N ' means all of the input points lie on the convex hull. The code used here is an example which is a circle with center (32767/2 ,32767/2).

'input' N = 100,500,1000,5000,10000

Note: The actual value of N is always less than the 'input ' N. This is because we use ' random ' . It causes the larger 'input' N , the more repetitions. But in the actual algorithm, there is no repeated point by default.

However we use the code to measure the average execution time by running 100 times , that will reduce the error.

execution time : 0.45483109999622684, 9.413900899999135, 29.279578300003777, 288.7286435999995, 711.1791921999975

Scatter plots (x-axis is N , y-axis is the measured execution time):