**Algorithm Implementation**

Giftwrap algorithm – Problem encountered

The function gives error of 'index out of range' or return inaccurate convex hull when the points are collinear. This is resolved by having elif statement to check whether the calculated angle is the same with the previous angle. If it is the same, the point with maximum distance from the current point of convex hull is chosen.

Graham-scan algorithm – Useful Structure

The sorting step of the Graham-scan algorithm takes O(nlogn) time. This is implemented using merge sort which has the worst case performance of O(nlogn).

Grahamscan algorithm - Problem encountered

When the first three points are collinear, the function has error of 'list index out of range'. To prevent this error, the sorted list is processed for the points with the same angles. If the points have same angle, the point with the maximum distance from the current point of convex hull is chosen.

When the angle is sorted, the points are not sorted according to their angle. The resolution is to store the angle to the list of that point as the third argument in the list then sort it according to their angle.

Both algorithm – Useful Python function + Problem encountered

randrange() function from random module is useful to generate random values for points within the specific range. However sometimes, especially when the number of points become large, the function generates duplicated points. To avoid this, the while loop is used to ensure the list has a unique set of values before appending it to the tested list, and regenerate if it is not.

- Division gives integer value. This is resolved by multiplied the numerator by 1.0.

**Algorithm analysis**

*Graph1 : Time taken to produce convex hull with rectangular distribution condition for given size of points*

*Graph2 : Time taken to produce convex hull with circular distribution condition for given size of points*

In theory, grahamscan algorithm complexity is O(nlogn) where m is number of vertices on convex hull and n is number of set of points in the list. In Graph1 and Graph2, the graphs from using grahamscan algorithm(red) look more like graph for O(n2) algorithm. This is likely due to the resolution used to resolve collinear points problem. As when the code to resolve that problem is removed, the graph look more like the graph of y = xlogx function.

In theory, the gift wrap algorithm complexity is O(mn) where m is number of vertices on convex hull and n is number of set of points in the list. In rectangular distribution, most points are not on the convex hull. This means the complexity above or equal to O(n) and less than O(mn) is expected. Example, O(n) or O(nlogn). This prediction is consistent with the experiment as shown in Graph1. In Graph1, the graph using giftwrap algorithm looks like a linear graph. In circular distribution, most points are on convex hull. This means O(nlogn) or O(n2) is expected. This matches with the graph using giftwrap algorithm in Graph2.

**Improvement**

Grahamscan – using Python sort function and preprocessing data points

The sorting step in Graham-scan algorithm can be improved by using python sort function on preprocessing data points. The best case performance of Python sort function is O(n). This is better than merge sort best case performance which is O(nlogn). So if the input data points are already sorted according to their angle, the Python sort function can perform its best case performance of O(n) instead of O(nlogn). This affects the whole algorithm complexity to reduce to O(n).

**Reference**

- <http://en.wikipedia.org/wiki/Timsort>

- http://interactivepython.org/runestone/default/user/login?\_next=/runestone/default/index