Noah Schill

CS 312 Spring 2021

Convex Hull

1. Partially functioning code in Appendix
2. Theoretical Analysis:

The algorithm finding the convex hull given n points consists of two main parts: a *convex_hull* function and a *merge* function. *convex_hull* calls *merge* and itself recursively.

Procedure *convex_hull* splits the list of sorted points in half and then calls itself on each of these halves. Therefore, each recursive call takes half as long as its calling stack, introducing an O(log n) complexity.

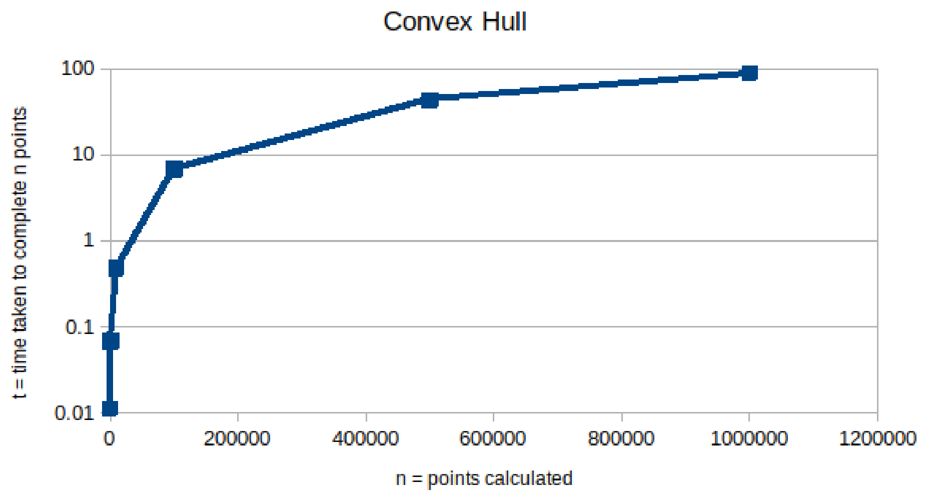
The main task of *merge* is to find an upper and lower tangent between two polygons and to join them together. The upper and lower tangent methods start from the inner points and work their ways outwards until slope decreases. In a worst-case scenario, these will take O(n/2) time or O(1) if the first guess is correct. Running both upper and lower tangent is equivalent to:
O(n/2) + O(n/2) = O(n).

Bringing this together with regards to the master theorem, we can say:
a = 2, as we are creating 2 sub-tasks with each recursive call
b = 2, as the time halves with each recursive call
d = 1, as it takes each recursive call is O(n).
T(n) = 2T(n/2)+O(n)
a/b^1 = 2/2 = 1 → O(n^d log n) = O(n log n)

This intuitively makes sense, as we are running our O (log n) function n times.

3. Empirical Analysis:

n	time(s)	mean time
10	1E-24	1E+025
100	0.011	9090.909091
1000	0.068	14705.88235
10000	0.481	20790.02079
100000	6.718	14885.38255
500000	43.276	11553.74804
1000000	88.378	11315.03315

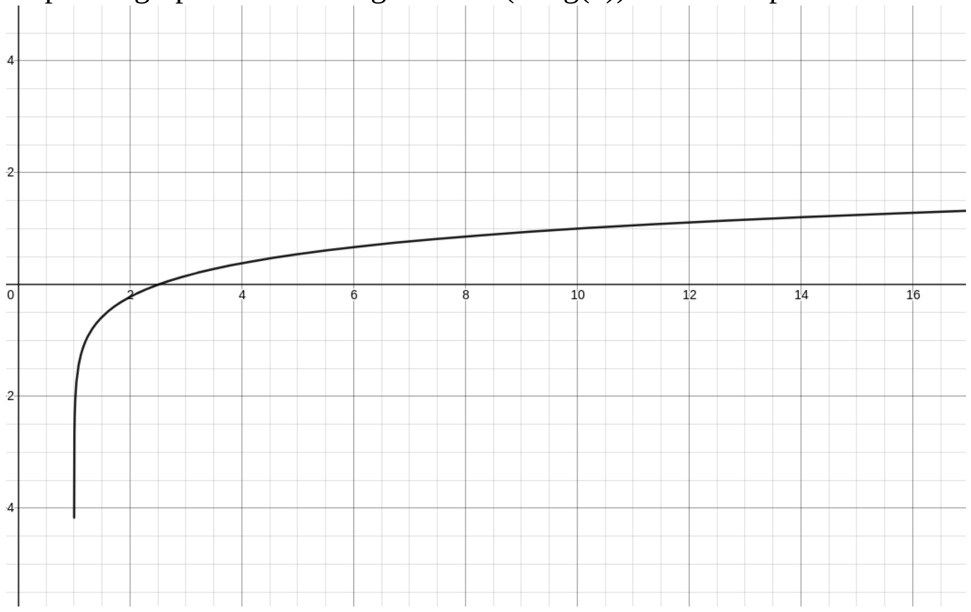


The order of growth which fits best is undoubtedly O (n log n). The reasoning behind the theoretical analysis would point to this being accurate.

By my estimate using empirical data, the constant of proportionality converged at about 8.838E-5.

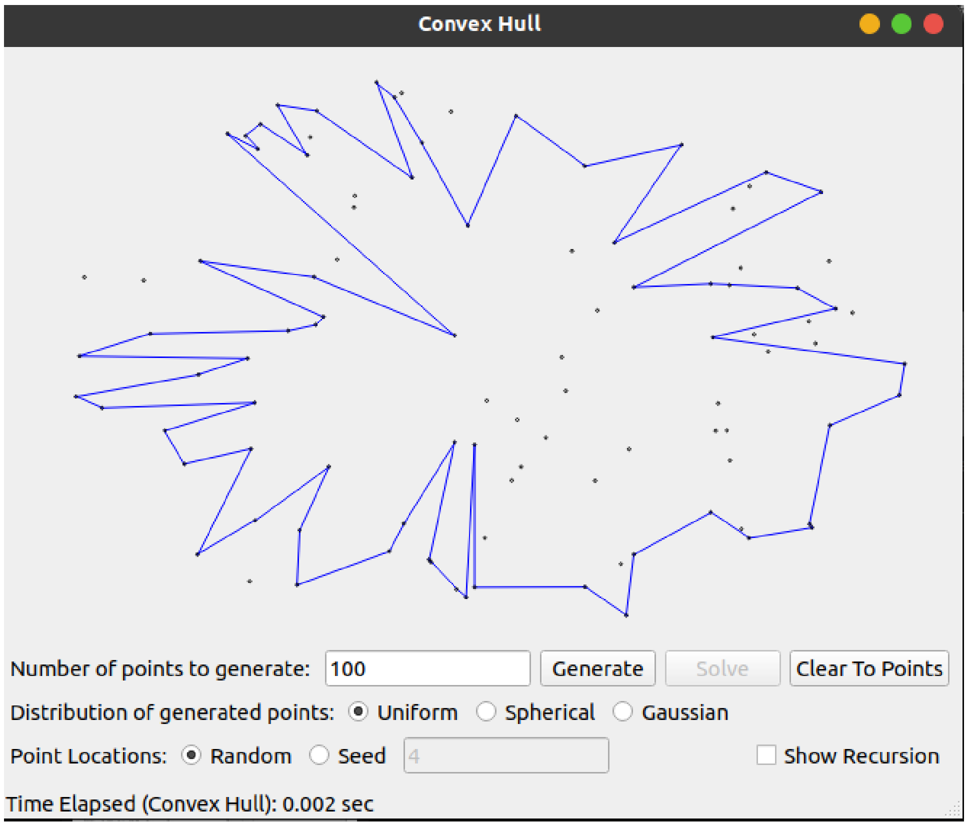
4. Observations

It is fascinating how closely we can predict the behavior and demands of an algorithm using asymptotic notation, the master theorem, among other tools. No major differences.   The above empirical graph looks like a generic O (n log(n)) function I plotted in Desmos below:



5. Screenshots

These are obviously not completely functional, but hopefully the code demonstrates knowledge of divide-and-conquer concepts even if the implementation is poor.



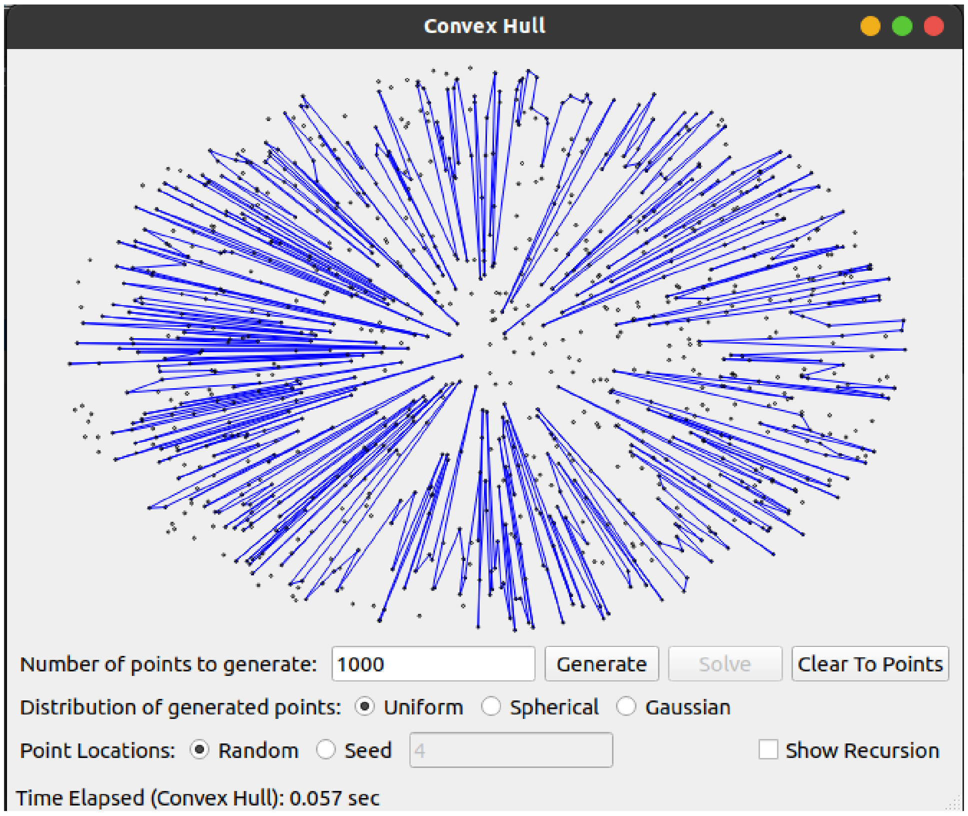
Number of points to generate: 100

Distribution of generated points: [✓ Uniform] Spherical Gaussian

Point Locations: [✓ Random] Seed 4

Show Recursion

Time Elapsed (Convex Hull): 0.002 sec



Number of points to generate: 1000

Distribution of generated points: [✓ Uniform] Spherical Gaussian

Point Locations: [✓ Random] Seed 4

Show Recursion

Time Elapsed (Convex Hull): 0.057 sec

Appendix A -- Python Code: