Report on Convex Hull

Sabal Subedi
Master's in Computer Science
Idaho State University
Pocatello, ID, 83209 USA
sabalsubedi@isu.edu

February 8, 2024

Abstract

In this report, I will summarize the use of divide and conquer algorithm to obtain the convex hull of a set of points. I will give a pseudocode and analyze the asymptotic notation.

1 Introduction

The convex hull of a set of points is defined as the smallest convex polygon, that encloses all of the points in the set. Convex means that the polygon has no corner that is bent inwards.
The divide and conquer algorithm is a strategy of solving a large problem by breaking the problem into smaller sub-problems, solving the sub-problems and finally combining them to get the desired output. To implement this algorithm, we use recursion.
Here, we use divide and conquer algorithm to compute the convex hull of a set of points. First, I divide the set of points into equals halfs recursively and then merge those halfs using the logics that gives the final convex hull of the given set of points.

2 Pseudocode and Asymptotic Analysis

2.1 Pseudocode for divide and conquer

Algorithm 1 Divide and Conquer

```
1: function DIVIDEANDCONQUER(points)
2:     if len(points) <= 3 then
3:         return points
4:     mid = len(points)//2
5:     left_hull ←DIVIDEANDCONQUER(points[: mid])
6:     right_hull ←DIVIDEANDCONQUER(points[mid :])
7:     final_hull ←CONVEXHULL(left_hull, right_hull)
8:     return final_hull
```

Time complexity and space complexity
Given,
points: a list of tuples(x,y)

Space Complexity:

$$S(n) = O(n) + O(\log n) \approx O(n) \tag{1}$$

Time Complexity:

$$\begin{aligned} T(n) &= O(1) + O(1) + O(\log n) + O(n^2) \\ &\approx O(n^2 * \log n) \end{aligned} \tag{2}$$

Analysis of algorithm:
The function takes set of points and output a single set. It uses divide and conquer algorithm to divide the given set of points into two halfs recursively. Finally, it merges the hull points to get the final convex hull.

2.2 Pseudocode for convex hull

Algorithm 2 Convex Hull

```
1:  function CONVEXHULL(left_hull, right_hull)
2:      points = left_hull + right_hull
3:      hull = []
4:      current = min(points)
5:      while True do
6:          hull.append(current)
7:          next_point = points[0]
8:          for point ← point[0] to points[len(points) − 1] do
9:              orient ← ORIENTATION(current, next_point, point)
10:             check_distance ←DISTANCE(current, point) >DISTANCE(current, next_point)
11:             if next_point = current or orient = 1 or (orient =
    0 and check_distance then
12:                 next_point = point
13:             current = next_point
14:             if current = hull[0] then
15:                 Break
16:     return hull
```

Time complexity and space complexity
Given,
left_hull: a list of tuples(x,y)
right_hull: a list of tuples(x,y)

Space Complexity:

$$\begin{aligned} S(n) &= O(n) + O(1) + O(1) + O(n) + O(1) + O(1) + O(n) * (O(1) + O(1) + O(1)) \\ &= O(3 * n) \approx O(n) \end{aligned} \tag{3}$$

Space Complexity:

$$\begin{aligned} T(n) = {} & O(n) + O(1) + \\ & O(n) * (O(1) + O(1) + O(n) * (O(1) + O(1) + O(1) + O(1))) + \\ & O(1) \approx O(n^2) \end{aligned} \tag{4}$$

Analysis of algorithm:
The function takes two set of points and output a single set. It checks orientation and distance between each points on the sets and computes the set of hull points and returns it.

2.3 Pseudocode to compute orientation

Algorithm 3 Orientation

```
function ORIENTATION(point1, point2, point3)
    upper = (point2[1] − point1[1])*(point3[0] − point2[0])
    lower = (point2[0] − point1[0])*(point3[1] − point2[1])
    orient = upper-lower
    if orient = 0 then
        return 0
    else if orient > 0 then
        return 1
    else
        return −1
```

Time complexity and space complexity
Given,
point1: a valid tuple(x,y)
point2: a valid tuple(x,y)
point3: a valid tuple(x,y)
Space Complexity:

$$S(n) = O(1) + O(1) + O(1) = O(3) \approx O(1) \tag{5}$$

Time Complexity:

$$\begin{aligned} S(n) &= O(1) + O(1) + O(1) + O(1) + O(1) + O(1) \\ &= O(6) \approx O(1) \end{aligned} \tag{6}$$

Analysis of algorithm:
The function takes three points as input and output a single value. It helps to determine the orientaion of the points3 relative to point1 and point2. It determines whether the point3 is collinear, clockwise or counterclockwise relative to point1 and point2. It returns 0 if point is collinear, 1 if it is clockwise and -1 if it is counterclockwise.

2.4 Pseudocode to compute distance

Algorithm 4 Distance

1: **function** DISTANCE($point1, point2$)
2: $x_1, y_1, x_2, y_2 = point1[0], point1[1], point2[0], point2[1]$
3: **return** $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Time complexity and space complexity
Given,
point1: a valid tuple(x,y)
point2: a valid tuple(x,y)
Space Complexity:

$$S(n) = O(1) + O(1) + O(1) + O(1) + O(1) = O(5) \approx O(1) \tag{7}$$

Time Complexity:

$$T(n) = O(1) + O(1) \approx O(1) \tag{8}$$

Analysis of algorithm:
The function takes two points as input and output a single value. Basically, it uses the Euclidean distance formula to compute distance between two points and returns the result as output.

3 Observations and Results

3.1 Observation Table

Sample Size	Distribution	Time
10	Gaussian	0.000 sec
100	Gaussian	0.001 sec
1000	Gaussian	0.010 sec
10000	Gaussian	0.121 sec
100000	Gaussian	1.201 sec
500000	Gaussian	5.918 sec
1000000	Gaussian	10.417 sec

Here, I have chosen Gaussian distribution to collect the elapsed time (i.e. time taken to get the convex hull) of seven different sample size. The elapsed time increases as the size of the sample increases.
To give more insight on this, I have collected 5 different samples for each sample size and get the data to compute the mean time needed for each sample size. Using the obtained data, I have plotted a graph shown in figure 1.

.

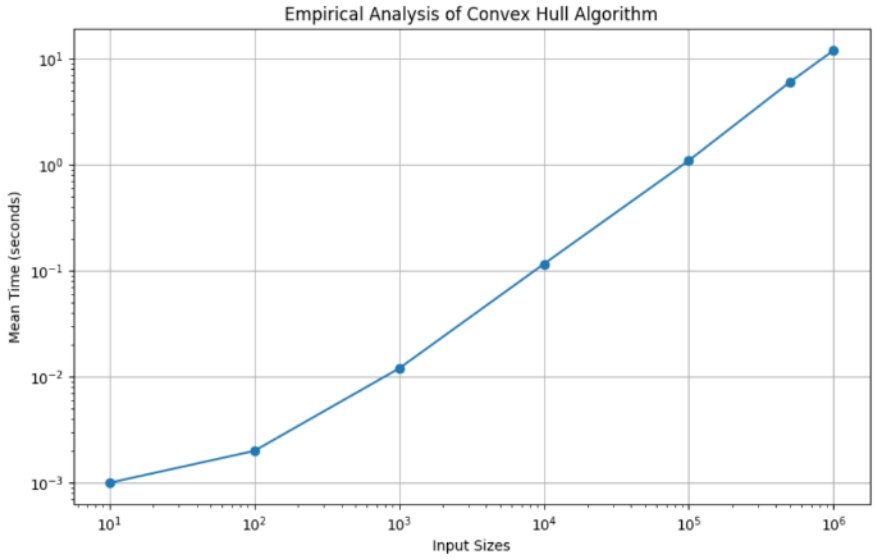

Figure 1: sample size vs mean time required

In the graph (in figure 1), we can see a increasing curve i.e. the mean time required to compute the convex hull with smaller sample size takes less time and so on. The change in size and the mean time follows logarithmic pattern. This mean that the changes either increasing or decreasing in sample size changes the mean time by increasing or decreasing rate.
According to my analysis on time complexity of CONVEXHULL algorithm, it takes quadradic time. I believe the graph (in figure 1) matches my analysis as the curve is identical to the curve of quadradic equation.
Since, I can observe my graph matches the n^2, I can conclude that g(n) = n^2 such that CH(Q) = k*g(n). Thus, the constant of proportionality k = 1.

3.2 Results

.

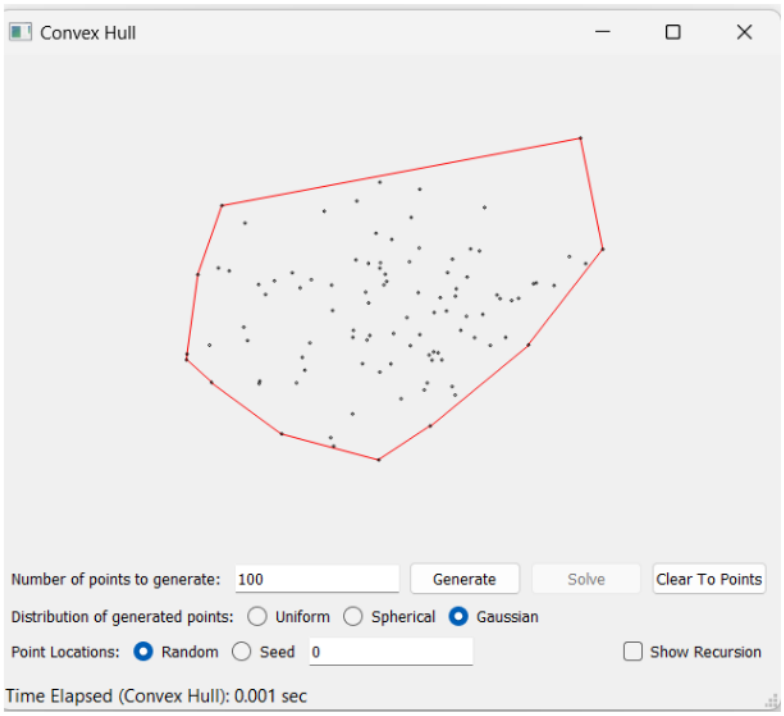

Figure 2: Convex hull for 100 points

I used 100 sample size (in figure 2) to compute hull. As expected the program was able to compute and display convex hull. The program took 0.001 second to compute the convex hull under Gaussian distribution and random seed.

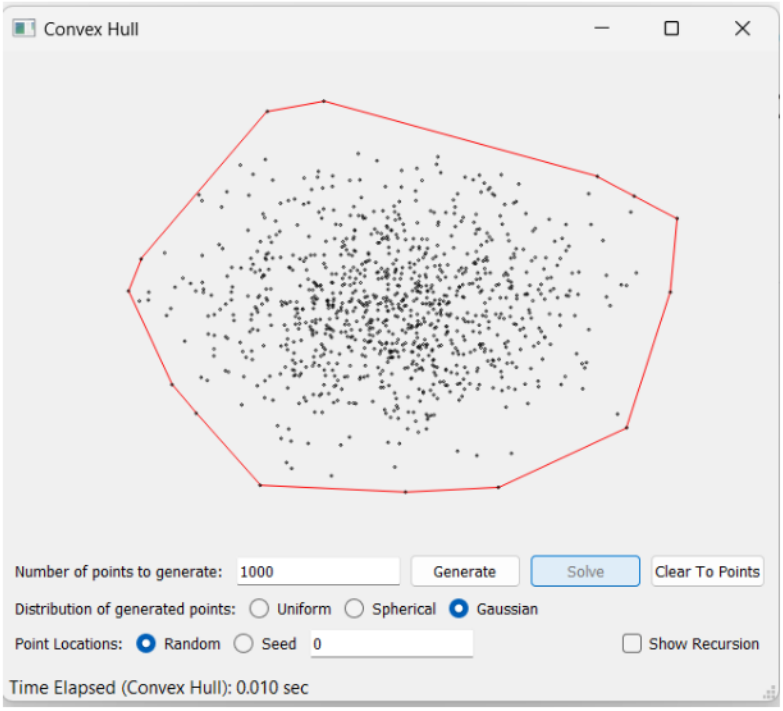

Figure 3: Convex hull for 1000 points

Again, I used 1000 random points to test the program under same setup and the program was able to generate the convex hull as expected. It took 0.010 seconds. The change in elpased time (required time) is significant as the sample size increases.

4 Conclusion

In summary, I was able to compute a convex hull using divide and conquer algorithm along with orientation and distance. I was able to analyze the pseudocode and get the time and space complexity.