
# Bilkent University

## CS478 Computational Geometry

## Final Report

Implementation of Two-dimensional Convex Hull Algorithms and Comparing Their Performances

Zeynep Nur Öztürk

21501472

# Table of Contents

# 1. Introduction

## a. What is Convex Hull?

The convex hull is the smallest convex set which contains every point in a given point set. Figure 1 shows the convex hull of 1000 randomly created points.

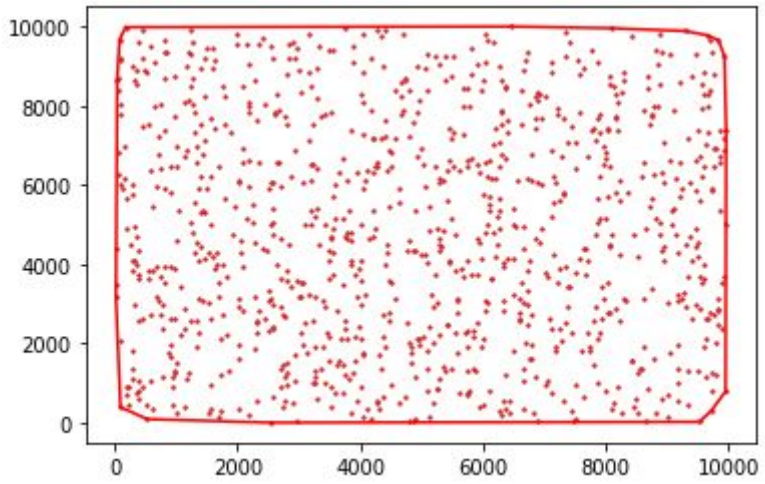



Figure 1: A picture of the convex hull which is the output of the implemented code which will be discussed in section 2.

Calculating the convex hull has various algorithms. Based on the algorithm, the time complexity or memory usage differs. The preference of the algorithm depends on the fundamental aim of the project.

## b. Usage of Convex Hulls

Convex hull algorithms have been used in many fields. Mathematics, statistics, economics, geometric modeling are some of the areas that use convex hulls[6].
For example, the asymptotic behavior of polynomials is calculating by convex hulls in mathematics[6]. In robot motions, the obstacles and paths that robots will move are determined by convex hulls[7]. Also, in research about tracking disease epidemics on animals, the epidemic area has been found by convex hulls as in figure 2.

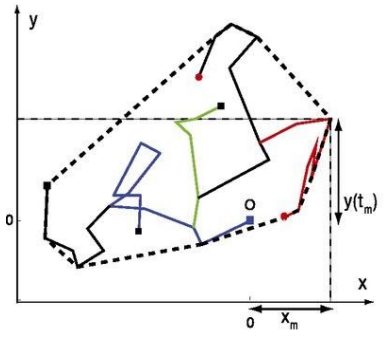



Figure 2 : A random walk of five individuals of animals. The graph is taken from article [8].

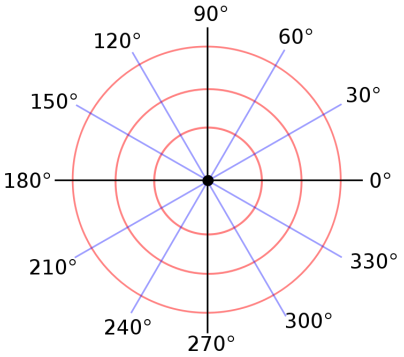
# 2. Background & Algorithms

In general most of the convex hull algorithms have O(nlogn) running time in 2D or 3D. Graham Scan, Gift Wrapping which is known as Jarvis March, Quickhull, Merge Hull which is known as Divide and Conquer Algorithm are investigated in this project.

## a. Graham Scan

This algorithm has been found by Ronald Graham in 1972 [9]. This has O(nlogn) running time. The steps of Graham's scan has been explained based on the algorithm that is written for this project.

i. **Preprocess**: The random points received as an array S. The point which has the minimum y-axis has been chosen as the pivot. All of the points have been sorted by their polar angle from smallest the larger. If two vertices have the same polar angle, then the distance is calculated. The ordered array is given to the function




.

Figure 3: Polar grid [10]

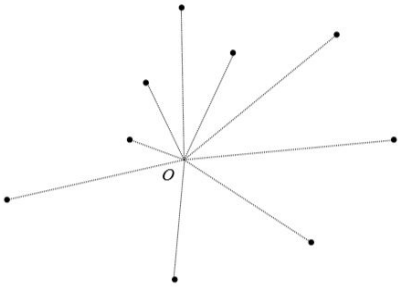



Figure 4: polar ordered points[3]

ii. **Scan:** The pivot which is chosen and the first element in the sorted list are pushed into the convex array, and in each loop, we get another point from the sorted list and apply the left test for last 2 elements in the convex array and the one in the sorted list. If the left test fails the points are pop out from the list until the left test gives a positive result. Then, return the convex hull array.

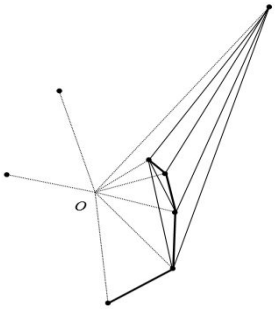



Figure 5: Scanning Process [3]

The preprocess dominated running complexity is O(nlogn), the " scanning process' complexity is O(nlogn).

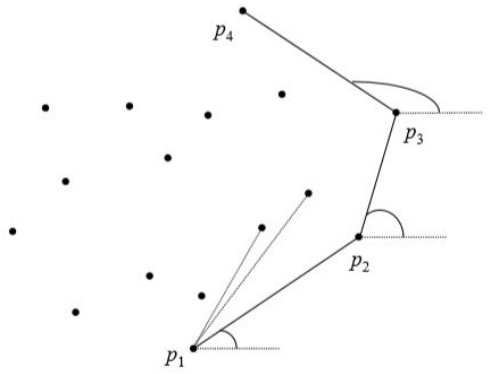
## b. Jarvis' March

This algorithm has been found by R.A. Jarvis in 1973 [11]. This algorithm has order O(nh) running time complexity. The steps of Jarvis March has been explained based on the algorithm that is written for this project.

i. **Preprocess:** The preprocessing of this algorithm is to find the smallest y coordinated point as the bottom pivot and largest as the top pivot.

ii. **Scan:** In this part we order points based on the bottom pivot. The bottom pivot is pushed to the convex array, then choose the smallest polar angled point for the array's last element. Then continue until the top pivot and repeat the same step from top to bottom. Figure 6 represents this step.

This algorithm calculates only the vertices on the convex hull. In other words, only the points of the convex hull change the running time. Preprocess has O(n) running time while the scan part has O(nh). This algorithm has O(nh) time complexity while h is the number of points in the convex hull.




. Figure 6: Jarvis March scanning [3]

## c. Quickhull

N-dimensional Quickhull is invented by C. B. Barber, D.P. Dobkin, and H.Huhdanpaa in 1996 [12]. The fundamental logic in this algorithm dividing the points into 2 subarrays. The steps of Quickhull has been explained based on the algorithm that is written for this project.

i. **Preprocess:** Finding the smallest and largest point in the x-axis and create another point which is slightly less than the smallest x point.

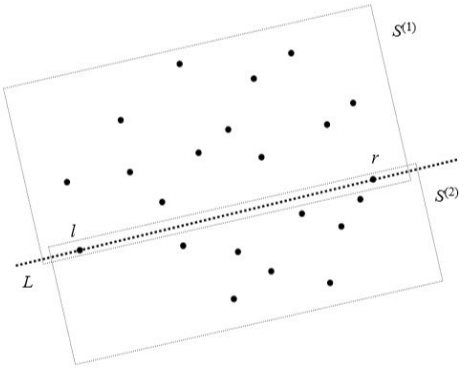



Figure 7: preprocess part of the quickhull[3]

ii. **Recursive method:** In this part two points and sub-array S are given to the function. The farthest point is found on the S by calculating the triangle area which is O(1). The points in the triangle are ignored and the recursive called for the two subarrays for left out and right out part of the triangle. This continues until one point is left in the subarray.

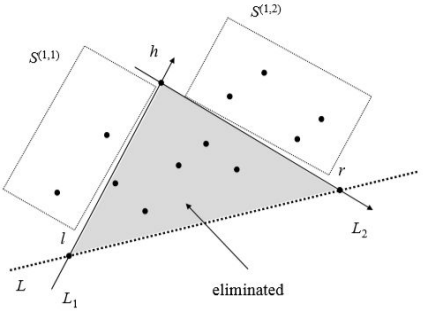



Figure 8: finding the farthest point and subarray the left and right part [3].

In the worst case, quickhull has O(n*n) running time which in case it is not equally divided in each recursive. However, it has O(nlogn) in the average running time. The preprocess part has O(n) running time while the recursive has O(n*n) in the worst case or O(nlogn) in the average case.

## d. Merge Hull

Merge hull has two different parts. The first part is calling quick hull recursively then merge them. Because the quick hull is explained in section 2.c only the merge hull will be discussed based on the algorithm which is implemented for this project. Assume P1 and P2 are given.

i. Find p internal to P1.

ii. Check whether or not p is internal to P2. If internal to P2 go to the step iii else go to step iv. To check it scan the P2 and apply left test even if one left test return negative it means it is not internal to the P2. This requires O(n) at most because of scan all the points in P2.

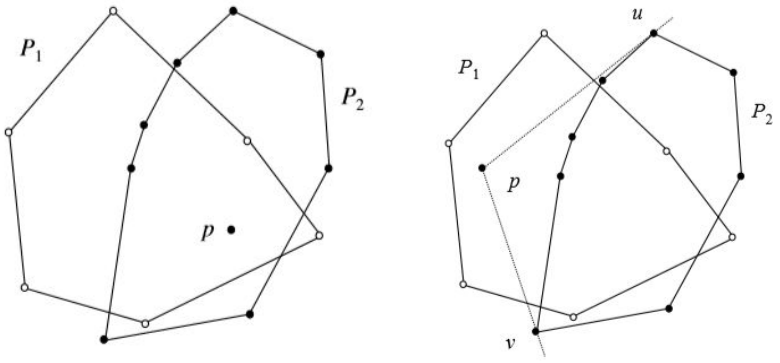



Figure 9: showing p inside P1 and P2 on the left and showing p inside P1 but not inside P2 on the right[3]

iii. Sort all points of P1 and P2 based on p with their polar angle. Sorting all points requires O(N) times

iv. Find u and v vertices on P2 to eliminate the middle chain. To find them, the supporting vectors are used by scanning all the points of P2 which has O(n) time complexity. Merge P2's outside chain and P1 and sort them by A polar angle is O(N) times.

v. Use a graham scan for points left. Because the list is sorted the graham scan's scan part will be used. Therefore, the time complexity is equal to O(n).

The merge hull has O(n) time complexity. However, divide and conquer has quickhull before merging them it has O(n*n) in the worst case and O(nlogn) time complexity in the average case.

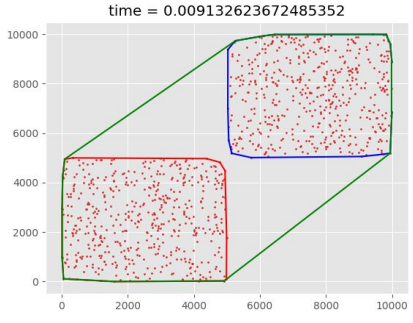



Figure 10: Example of merge sort

# 3. Progress

## a. Language and Libraries

i. **Language:** Python 2.7 and 3.6 has been used.

ii. **Libraries:**

1. Random library is used to create random points.
2. Math library is used to calculate arctan and pi.
3. Numpy library used to store the points.
4. Time library is used to calculate the time.
5. Matplotlib is used to visualize the results.
6. Sys library is used to increase the recursion limit of the terminal.
7. Tkinter library is used to create UI

## b. Features

There are some features in this project as:

i. Points are created from 1000 to 10,000,000 randomly.

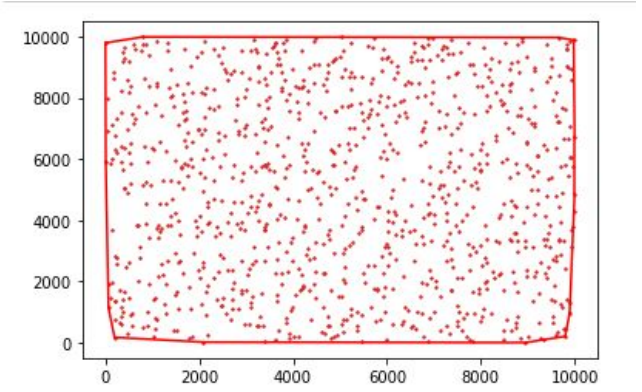



Figure 11: The uniformly distributed 1000 points from the code

ii. Points can be created as uniformly distributed or clustered.

iii. Time is calculated from the beginning of the function until the end.

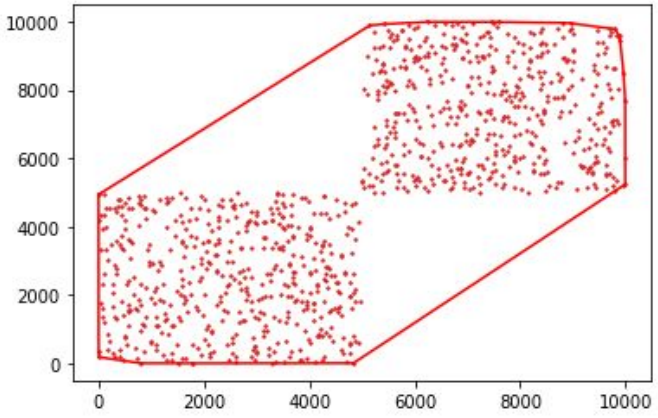



Figure 12: The clustered example of 1000 points from the code

## c. UI Design

Because time limit UI is implemented as below.

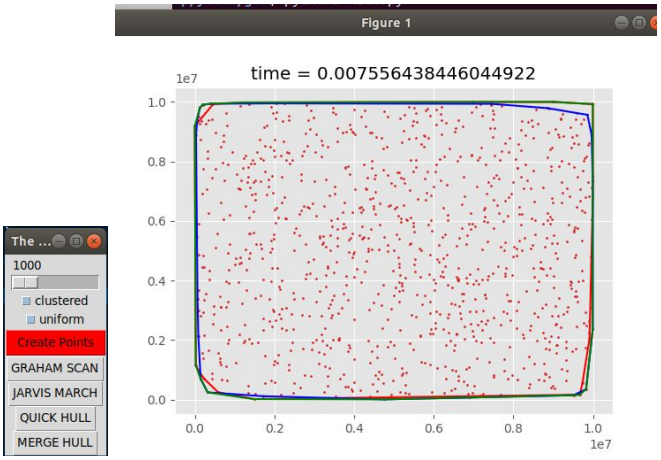



Figure 13: UI example

# 4. Results

## a. Graham Scan's Results

Results are showing that a graham scan doesn't differ based on the clustered or uniform points until 5,000,000 point size. This algorithm is independent of the shape of the points, because it visits ever point and order based on the polar angle. After 5,000,000 point size the reason the time differs because of the computer tired from long hour usage.

Also, the rate of the expected running time can be compared. Point size 34,000,000 has expected time O(nlogn)=26,408,240 and point size 4,000,000 has expected time O(nlogn)=19,431,363. the rate of their expected time is 0.73 and real-time rate is 0.76.

| Graham Scan | Uniform(s) | Clustered(s) | Expected |
|---|---|---|---|
| 1,000 | 0.00348 | 0.00343 | O(nlogn)=3,000 |
| 10,000 | 0.04643 | 0.04083 | O(nlogn)=40,000 |
| 100,000 | 1.26024 | 0.48540 | O(nlogn)=500,000 |
| 1,000,000 | 7.10247 | 7.25312 | O(nlogn)=6,000,000 |
| 2,000,000 | 14.91750 | 14.30834 | O(nlogn)=12,602,060 |
| 3,000,000 | 23.81629 | 23.92488 | O(nlogn)=19,431,363 |
| 4,000,000 | 31.19035 | 30.92443 | O(nlogn)=26,408,240 |
| 5,000,000 | 40.74763 | 38.05435 | O(nlogn)=33,494,850 |
| 10,000,000 | 87.29779 | 80.77861 | O(nlogn)=70,000,000 |

## b. Jarvis' March's Result

Results are showing that Jarvis March differs based on the number of convex hull's vertices. In the uniform part rate of the point size 1,000,000 is O(nh)=33,000,000 and for point size 2,000,000 is O(nh)=94,000,000. their ratio is 0.35 and the real-time ratio is 0.37. The same thing apply for the Clustered part.

| Jarvis' March | Uniform(s) | Clustered(s) | Expected |
|---|---|---|---|
| 1,000 | 0.01776<br>h = 23 | 0.01535<br>h = 18 | u->O(nh)=23,000<br>c->O(nh)=18,000 |
| 10,000 | 0.23531<br>h = 26 | 0.25829<br>h = 26 | u->O(nh)=260,000<br>c->O(nh)=260,000 |
| 100,000 | 5.52423<br>h = 37 | 4.01435<br>h = 35 | u->O(nh)=3,700,000<br>c->O(nh)=3,500,000 |
| 1,000,000 | 66.89897<br>h = 33 | 69.90606<br>h = 41 | u->O(nh)=33,000,000<br>c->O(nh)=41,000,000 |
| 2,000,000 | 174.11069<br>h = 47 | 115.76362<br>h = 44 | u->O(nh)=94,000,000<br>c->O(nh)=88,000,000 |
| 3,000,000 | 244.66016<br>h = 45 | 295.58773<br>h = 48 | u->O(nh)=135,000,000<br>c->O(nh)=144,000,000 |
| 4,000,000 | 340.58623<br>h = 40 | 414.12098<br>h = 48 | u->O(nh)=160,000,000<br>c->O(nh)=192,000,000 |
| 5,000,000 | 353.88278<br>h = 47 | 469.60794<br>h = 44 | u->O(nh)=235,000,000<br>c->O(nh)=220,000,000 |
| 10,000,000 | 356.32224<br>h=50 | 1145.9719<br>h=66 | u->O(nh)=500,000,000<br>c->O(nh)=660,000,000 |

u-> represents the uniform's expected time

c-> represents the clustered's expected time

## c. Quickhull's Result

The X's in the table means that recursive reach to the depth. Results are showing that Quickhull differs based on the number of convex hull's vertices. In the uniform part rate of the point size 1,000,000 is O(nlogn)=6,000,000 and for point size 2,000,000 is O(nlogn)=12,602,060. their ratio is 0.47 and the real time ratio is 0.45. Same thing apply for the Clustered part. If we look for O(n*n) the ratio between expected time 0.25 which shows us the algorithm most likely run in the average time.

| Quickhull | Uniform(s) | Clustered(s) | Expected |
|---|---|---|---|
| 1,000 | 0.00512 | 0.00384 | O(nlogn)=3,000<br>O(n*n)=10^6 |
| 10,000 | 0.05689 | 0.03930 | O(nlogn)=40,000<br>O(n*n)= 10^8 |
| 100,000 | 0.71644 | 0.55157 | O(nlogn)=500,000<br>O(n*n)=10^10 |
| 1,000,000 | 5.52458 | 5.60349 | O(nlogn)=6,000,000<br>O(n*n)=10^12 |
| 2,000,000 | 11.87036 | 11.54690 | O(nlogn)=12,602,060<br>O(n*n)=4*10^12 |
| 3,000,000 | X | 17.12558 | O(nlogn)=19,431,363<br>O(n*n)=9*10^12 |
| 4,000,000 | X | X | O(nlogn)=26,408,240<br>O(n*n)=16*10^12 |
| 5,000,000 | X | X | O(nlogn)=33,494,850<br>O(n*n)=25*10^12 |
| 10,000,000 | X | X | O(nlogn)=70,000,000<br>O(n*n)=10^14 |

## d. Merge Hull's Result

The X's in the table shows that function reach the maximum recursive depth. Also, the results show the ratio between 5,000,000 point size and 3,000,000 point sṣze. Expected ratio is 0.57 and the real time ratio is 0.56 which also apply in the clustered points.

| Merge Hull | Uniform(s) | Clustered(s) | Expected |
|---|---|---|---|
| 1,000 | 0.00414 | 0.00391 | O(nlogn)=3,000 |
| 10,000 | 0.04171 | 0.03664 | O(nlogn)=40,000 |
| 100,000 | 0.52161 | 0.44887 | O(nlogn)=500,000 |
| 1,000,000 | 6.18138 | 6.00312 | O(nlogn)=6,000,000 |
| 2,000,000 | 12.24851 | 13.15488 | O(nlogn)=12,602,060 |
| 3,000,000 | 17.88851 | 26.40263 | O(nlogn)=19,431,363 |
| 4,000,000 | 25.77283 | 31.05477 | O(nlogn)=26,408,240 |
| 5,000,000 | 32.26809 | 42.65405 | O(nlogn)=33,494,850 |
| 10,000,000 | X | X | O(nlogn)=70,000,000 |

## e. Clustered Results

The reason that table shows only the point size from 1,000,000 to 10,000,000 is the results are more clear in this. If we check for O(nlogn) and O(nh) the results of O(nh) is always bigger than O(nlogn) which shows the Jarvis' March should be larger than the others. While the point size is 2,000,000 the ratio between expected times of O(nlogn)/O(nh) has the same ratio with real-time results of Graham and Jarvis.
Also, Until 3,000,000 Quickhull is faster than the Graham Scan. Merge hull is worst than quick hull however, graham scan and merge sort are close to each other. This table shows that, Until 3,000,000 point size result quickhull has the

best performance. However the recursive calls were hard to handle for computer after a certain point size. Graham and merge sort is preferable rather than jarvis, but Merge hull has quickhull function in it. Because it has two sub quickhull inside the quick hull calls 2,500,000 pointsize while merge sort has 5,000,000 points. Therefore, Merge hull will be eliminated from the list after 6,000,000 point size. Graham scan has a advantage on large point sized results

| Clustered | Graham | Jarvis | Quickhull | Merge | Running times |
|---|---|---|---|---|---|
| 1,000,000 | 7.25312 | 69.90606<br>h = 41 | 5.60349 | 6.00312 | O(nlogn)=6,000,000<br>O(nh)=41,000,000 |
| 2,000,000 | 14.30834 | 115.76362<br>h = 44 | 11.54690 | 13.15488 | O(nlogn)=12,602,060<br>O(nh)=88,000,000 |
| 3,000,000 | 23.92488 | 295.58773<br>h = 48 | 17.12558 | 26.40263 | O(nlogn)=19,431,363<br>O(nh)=144,000,000 |
| 4,000,000 | 30.92443 | 414.12098<br>h = 48 | X | 31.05477 | O(nlogn)=26,408,240<br>>O(nh)=192,000,000 |
| 5,000,000 | 38.05435 | 469.60794<br>h = 44 | X | 42.65405 | O(nlogn)=33,494,850<br>O(nh)=220,000,000 |
| 10,000,000 | 80.77861 | 1145.9719<br>h=66 | X | X | O(nlogn)=70,000,000<br>O(nh)=660,000,000 |

## f. Uniform Results

The results of O(nh) is always bigger than O(nlogn) which shows the Jarvis' March should be larger than the others. While the point size is 2,000,000 the ratio between expected times of O(nlogn)/O(nh) has not the same ratio with real-time results of Graham and Jarvis which is unpredictable.

Also, Until 3,000,000 Quickhull is faster than the Graham Scan. Merge hull is worst than quick hull however, graham scan and merge sort are distinguishable in real time. This is caused by the merge hull algorithm does not eliminates the middle chain which gains some time when the points are uniform. This table shows that, Until 3,000,000 point size result quickhull has the best performance. However the recursive calls were hard to handle for computer after a certain point size. Graham and merge sort is preferable rather than jarvis, but Merge hull has quickhull function in it. Because it has two sub quickhull inside the quick hull calls 2,500,000 pointsize while merge sort has 5,000,000 points. Therefore, Merge hull will be eliminated from the list after 6,000,000 point size. Graham scan has a advantage on large point sized results

| Uniform | Graham | Jarvis | Quickhull | Merge | Running times |
|---|---|---|---|---|---|
| 1,000,000 | 7.10247 | 66.89897<br>h = 33 | 5.52458 | 6.18138 | O(nlogn)=6,000,000<br>O(nh)=33,000,000 |
| 2,000,000 | 14.91750 | 174.11069<br>h = 47 | 11.87036 | 12.24851 | O(nlogn)=12,602,060<br>O(nh)=94,000,000 |
| 3,000,000 | 23.81629 | 244.66016<br>h = 45 | X | 17.88851 | O(nlogn)=19,431,363<br>O(nh)=135,000,000 |
| 4,000,000 | 31.19035 | 340.58623 | X | 25.77283 | O(nlogn)=26, |

| | | | | | |
|---|---|---|---|---|---|
| | | h = 40 | | | 408,240 >O(nh)=160,000,000 |
| 5,000,000 | 40.74763 | 353.88278 h = 47 | X | 32.26809 | O(nlogn)=33,494,850 O(nh)=235,000,000 |
| 10,000,000 | 87.29779 | 356.32224 h=50 | X | X | O(nlogn)=70,000,000 O(nh)=500,000,000 |

# 5. References

[1] X. Kong, H. Everett, and G. Toussaint, “The Graham scan triangulates simple polygons,” *Pattern Recognition Letters*, vol. 11, no. 11, pp. 713–716, 1990.

[2] S.-L. Chen, J.-Y. Zhang, X.-G. Lu, K.-C. Chou, and Y. A. Chang, “Application of Graham Scan Algorithm in Binary Phase Diagram Calculation,” *Journal of Phase Equilibria & Diffusion*, vol. 27, no. 2, pp. 121–125, Jan. 2006.

[3]Computational Geometry: An Introduction, F. P. Preparata and M.I. Shamos, SpringerVerlag, 1985.

[5]Maher M. Atwah and Johnnie W. Baker and Selim Akl, “An Associative Implementation Of Classical Convex Hull Algorithms ” , http://www.cs.kent.edu/~jbaker/PDC-Sp12/references/atwah96.pdf Accessed 30.03.2020.

[6] Convex hull. (2020, April 18). Retrieved May 25, 2020, from https://en.wikipedia.org/wiki/Convex_hull

[7] Tao, Tan, & Juan. (2018, October 04). Path Planning with Obstacle Avoidance Based on Normalized R-Functions. Retrieved May 25, 2020, from https://www.hindawi.com/journals/jr/2018/5868915/

[8] Dumonteil, E., Majumdar, S., Rosso, A., & Zoia, A. (2013, March 12). Spatial extent of an outbreak in animal epidemics. Retrieved May 25, 2020, from https://www.pnas.org/content/110/11/4239.full

[9] Graham scan. (2020, April 21). Retrieved May 25, 2020, from https://en.wikipedia.org/wiki/Graham_scan

[10] Dumonteil, E., Majumdar, S., Rosso, A., & Zoia, A. (2013, March 12). Spatial extent of an outbreak in animal epidemics. Retrieved May 25, 2020, from https://www.pnas.org/content/110/11/4239.full

[11] Dumonteil, E., Majumdar, S., Rosso, A., & Zoia, A. (2013, March 12). Spatial extent of an outbreak in animal epidemics. Retrieved May 25, 2020, from https://www.pnas.org/content/110/11/4239.full

[12]Dumonteil, E., Majumdar, S., Rosso, A., & Zoia, A. (2013, March 12). Spatial extent of an outbreak in animal epidemics. Retrieved May 25, 2020, from https://www.pnas.org/content/110/11/4239.full