**A Comprehensive Study of Quick Hull: Algorithm, Complexity and Applications**

Gargee Thakur, Riddhima Ghule

*Department of Computer Engineering. SVKM'S NMIMS, Navi Mumbai, India*email: gargeet16@gmail.com

riddhimaghule18@gmail.com

*Abstract*—In this paper we have conducted a research on quick hull, and conducted an comparison with other methods to find the convex hull namely Jarvis March, Graham’s scan and Monotone chain. We took a set of points in a 2D plane and found its convex hull. Then we calculated the time and space complexities of quick hull and other algorithms and made a comparison. Jarvis March had a best-case time complexity while the space complexities of other algorithms were the same. Then we took a different number of random points – 20, 50, 100, 150, 200…, 1000, to calculate the run time difference between them we found that Jarvis March was quicker for smaller number of points, Quick hull is better only if the points are well distributed, and overall Monotone Chain and Graham’s Scan are faster than other algorithms.

Keywords—Convex hull, Quick hull, Algorithm, Complexities

# introduction

## ConvexHull

The convex hull of a set of points is the smallest convex set that contains the points. The Convex Hull Algorithm is used to find the convex hull of a set of points in computational geometry. The convex hull is the smallest convex set that encloses all the points, forming a convex polygon. This algorithm is important in various applications such as image processing, route planning, and object modelling. This article presents a practical convex hull algorithm that combines the two-dimensional Quickhull algorithm with the general-dimension Beneath-Beyond Algorithm. It is similar to the randomized, incremental algorithms for convex hull and Delaunay triangulation. We provide empirical evidence that the algorithm runs faster when the input contains nonextreme points and that it used less memory. Computational geometry algorithms have traditionally assumed that input sets are well behaved. When an algorithm is implemented with floating-point arithmetic, this assumption can lead to serious errors. We briefly describe a solution to this problem when computing the convex hull in two, three, or four dimensions. The output is a set of “thick” facets that contain all possible exact convex hulls of the input. A variation is effective in five or more dimensions. The quick hull algorithm uses a divide and conquer strategy to compute the convex hull of a shape. A shape's convex hull (also referred to as the convex closure) is the smallest set of points encapsulating it. The following image illustrates the convex hull.

The Quickhull algorithm is an adaptation of the Quicksort algorithm, a recursive method which uses the divide-and-conquer approach. The original problem is divided into two subproblems, each subproblem is solved recursively, and the solutions are merged to produce the overall solution. Quickhull is a method of computing the convex hull of a finite set of points in the plane. It uses a divide and conquer approach similar to that of quicksort, from which its name derives. Its average case complexity is considered to be Θ(n \* log(n)), whereas in the worst case it takes O(n^2). Quick Hull was published by C. Barber and D. Dobkin in 1995. [1, 4]

We also have other different types of convex hull algorithms, which are as follows:

Graham’s Scan: Graham’s scan algorithm was introduced by Ronald Graham in 1972 [5]. The algorithm represents the working hull as a stack. It starts with an extreme point and sorts all the other points according to the angle they make with that initial extreme point. The Graham scan algorithm is a simple and efficient algorithm for computing the convex hull of a set of points. It works by iteratively adding points to the convex hull until all points have been added. The algorithm starts by finding the point with the smallest y-coordinate. This point is always on the convex hull. The algorithm then sorts the remaining points by their polar angle with respect to the starting point. The algorithm then iteratively adds points to the convex hull. At each step, the algorithm checks whether the last two points added to the convex hull form a right turn. If they do, then the last point is removed from the convex hull. Otherwise, the next point in the sorted list is added to the convex hull. The algorithm terminates when all points have been added to the convex hull. [1, 5, 6]

Jarvis’s March (Gift Wrapping): The gift-wrapping algorithm, also known as the Jarvis March algorithm, was discovered independently by Chand and Kapur in 1970 and R.A. Jarvis in 1973 [4]. The algorithm starts by finding the leftmost point and then iteratively choosing the next point such that all other points are to the right side of the line between the current and next point. The algorithm is called the gift-wrapping algorithm because the hull is formed by “wrapping” it around the point set. Jarvis’s March is similar to Selection Sort. In each iteration of Selection Sort, we pick the smallest element in the array and then move it to the front. Once we’re done, we’ll have a sorted array. In Jarvis’s algorithm, the smallest element in each iteration is the vertex that is furthest to the right from the last vertex that was added to the convex hull. The Jarvis March algorithm computes the convex hull of a set S of planar points by identifying the hull edges. There are two main steps.

1. Find the left most vertex

2. Find the next vertex in the hull and repeat this until we're done. [7, 8] Use the enter key to start a new paragraph. The appropriate spacing and indent are automatically applied.

Monotone Chain (Andrew’s Algorithm**):** Andrew's Monotone Chain Algorithm is a method for finding the convex hull of a set of points in the plane (fig.1). It works by sorting the points lexicographically (first by x-coordinate, then by y-coordinate) and then constructing the upper and lower halves of the convex hull separately. It constructs the convex hull in O(n \* log(n)) time. We must sort the points first and then calculate the upper and lower hulls in O(n) time. The points will be sorted with respect to x-coordinates (with respect to y-coordinates in case of a tie in x-coordinates), we will then find the left most point and then try to rotate in clockwise direction and find the next point and then repeat the step until we reach the rightmost point and then again rotate in the clockwise direction and find the lower hull.[9]

Fig. 1: Convex hull

***B*** *Overview: What is Quickhull?*

Quickhull is a method of computing convex hull of a set of points in 2-dimensional or higher-dimensional space by recursively partitioning the dataset. Convex hull is the smallest convex polygon that encloses all the given points. It uses divide-and-conquer approach and is similar to quicksort. The algorithm starts by identifying the left-most and right-most points from the given set of points, and then we form partitions by joining these two points, and the remaining points in these two sets are based on their positions relative to this line. Then the points which are farthest from this line are selected as the next hull point, creating new line segments. This process is repeated recursively until no points remain outside the convex hull. This method is useful in finding the convex hull, especially when the points are randomly distributed. This algorithm is also widely used in three-dimensional or higher dimensional space, though its efficiency might decrease. It is widely preferred and has various applications such as computer visualization and image processing which is used in determining object boundaries, detect shapes and simplify complex spaces, path finding is used in robotic navigation or autonomous vehicles by identifying obstacles and mapping their convex boundaries, visual pattern matching by identifying objects by determining their convex shapes and in geometry where quick hull is used to solve problems including Voronoi diagrams, Delaunay triangulations. [10]

# problem statement

Given is a set of points in a two-dimensional plane, our objective is to find the quick hull of these points, determine its time complexity and space complexity along with its best case and worst case, and compare it with other algorithms of convex hull.

## Fig. 2: Problem statement

# algorithm

## QuickHull

If there is a set of points lying on a two-dimensional plane in such a way that they are composed of set of points and line segments. A n-dimensional convex hull is represented by its number of vertices and (n-1)-dimensional faces. Two points which are present at most extreme position from each other is joined together and they form an intersection, creating a partition between other points. After determining these two points lets name them A and B, we select a third farthest point from A and B, lets name it C, and join C with A and B, forming a triangle ABC. We continue this same process by joining the farthest most point from AC and CB, and so on. So, our algorithm keeps on running recursively until all points lie inside the closed polygon. All the points bounding the polygon define the convex hulls of the given set of points, the points inside the polygon do not affect the convex hull of the set of points. [10,11]

**Algorithm 1** Algorithm for Quick Hull

**Input**: in a set of n points

**Output**: out points forming the convex hull

***Initialization:***

1. Let the set consist of at least 2 points
2. QuickHull(set)
3. Convex\_Hull == {}
4. Find left and right most extreme points, say A and B, and join them
5. Segment AB divides the remaining (n-2) points into two partitions S1 and S2
6. Here, S1 are the points present in the set on the right side of line segment AB
7. And, S2 are the points present in the set on the left side of line segment BA
8. FindHull(S1, A, B)
9. FindHull(S2, B, A)}
10. FindHull(S, A, B)
11. If there are no points present in set S
12. **return**
13. From the given set of points S, find the farthest point from line segment AB, and name it C
14. Add point C to convex hull at the location between A and B
15. Three points A, B and C partition the remaining points in the subset S into more subsets, suppose S0, S1 and S2
16. Here, S0 are points inside the triangle ABC
17. S1 are the points on the right side of line segment AC
18. and, S2 are the points on the right side of line segment CB
19. FindHull(S1, A, C)
20. FindHull(S2,C,B)

After going through the pseudocode carefully, it can be observed that FindHull() function is called recursively after forming new partitions within the set. This function will be called recursively until there are no points left outside the polygon, and hence the convex hull of the given set of points can be determined.[11]

## Graham Scan

Graham’s Scan algorithm finds the corner points of the convex hull. First, the lowest point is chosen, and it is the starting point of the convex hull. The remaining vertices are sorted based on anti-clockwise direction from the starting point. If the two or more points form the same angle, then all the points are removed except the farthest point from start. The remaining points are pushed into the stack, and then popped one by one, when the orientation is not anti-clockwise for the top point in the stack, the second top point and the newly selected points are pushed into the stack.

**Algorithm 2** Algorithm for Graham Scan

**Input:** A set of n points in the plane, P = {p1, p2, . . . , pn}

**Output:** The convex hull H, a sequence of points ordered counterclockwise

***Initialization:***

1. Find the point pmin with the smallest y-coordinate in P
2. Sort the remaining points in increasing order of the angle with respect to pmin
3. Initialize an empty stack S to store the points on the convex hull
4. **For** each point p ∈ P, starting from the sorted list do
5. **While** the size of S is greater than 1 and the turn from the second-to-last point in S to the last point in S and p is a right turn
6. **Pop** the last point from S
7. **Push** p onto the stack S
8. **End for**
9. **Return** the points in S, which represent the convex hull H

## Jarvis March

Jarvis March is used to detect the corner points of convex hull from a given set of points. It starts from leftmost point of the dataset, the points are kept in convex hull by anti-clockwise rotation. From the current point, we can choose the next point by checking the orientations of those points from the current point. When the angle is the largest that point is chosen, all points go through this algorithm, and hence the convex hull is chosen.

**Algorithm 3** Algorithm for Jarvis March

**Input:** A set of n points in the plane, P = {p1, p2, . . . , pn}

**Output:** The convex hull H, a sequence of points ordered counterclockwise

***Initialization***

1. Initialize an empty list H to store the convex hull points
2. Find the leftmost point pleft ∈ P. Set pcurrent = pleft
3. **while** pcurrent ≠ pleft do
4. Add pcurrent to H
5. Set pnext = pcurrent
6. **for** each point p ∈ P **do**
7. If p ≠ pcurrent and p is more counterclockwise than pnext with respect to pcurrent
8. Set pnext = p
9. **End** **if**
10. **End** **for**
11. Set pcurrent = pnext
12. **End** **while**
13. **Return** H

## Monotone Chain

The points are sorted with respect to x-coordinates into lower and upper hulls. Then find the leftmost point and repeat the algorithm until we reach rightmost point. Then again the points are rotated in clockwise direction.

**Algorithm 4** Algorithm for Monotone Chain [7, 9]

**Input:** set of n points,

**Output:** set H = the output convex hull of set P, k number of output convex points in set H.

***Initialization***

1. Sort points set P based on their X coordinate in counter-clockwise order
2. make P as the sorted array of N points.
3. Let P1 is leftmost point, P2 is the rightmost point, k = 3
4. Add both points p1, p2 to output array of convex points
5. H as H[ 1 ] = p1,
6. H[ 2 ] = p2
7. Divides the set of points into upper and lower hull based on the line p1p2.
8. Build upper hull as follow
9. **for** I = 1 to n - 1
10. begin
11. **while** (k >= 3) and (cross\_Direction (H[ k - 2 ], H[ k - 1 ], p[ I ]) <= 0) do
12. k = k - 1
13. H[ k ] = P[ I ]
14. k = k + 1
15. **End**

Build lower hull as follow:

1. t = k + 1
2. **for** I = n - 2 down to 1 do
3. begin
4. **while** (k >= t ) and (cross\_Direction(H[k-2], H[k-1], P[ I ]) <= 0) **do**
5. k = k - 1
6. H[k] = P [ I ]
7. k = k + 1
8. End
9. **return** H and k

# time complexity

## Quick Hull

## The time complexity of Quickhull is similar to QuickSort due to the division into two sub-problems.In general, the time complexity of geometric algorithms is difficult to analyse as characterization of the data is not simple. Note that the searching and splitting operations of algorithm Hull will always have a time complexity of O(n).

**Case 1:** (almost) every point a primary vertex: This is the case where time-complexity is O(n2). This is the worst case for any convex hull algorithm, since all primary vertices must be output. This happens when the points are distributed in such a way that every recursive call processes almost all points. For example, suppose the input points are arranged in a way that forces the algorithm to perform many recursive calls without significantly reducing the number of points. In that case, the complexity degrades to O(N²). [4,14]

#### 

**Fig. 3:** Worst case point distribution

**Case 2 Average Case:** In the average case, the QuickHull algorithm behaves similarly to QuickSort and runs in O(N log N) time. This happens when the recursive partitioning effectively reduces the number of points processed at each step.

This results in a recurrence relation similar to **QuickSort**: T(n) = 2 T(n/2) + O(n).

This expands to become T(n) = O(n log(n)).

Example: When points are randomly distributed, such as scattered points forming an irregular shape (e.g., a star-like or polygonal spread).

**Case 3 (reasonably) uniformly distributed points:** Here, we assume that our partitioning of the data within Quick Hull is (reasonably) even. It is similar to Average-case. By, the divide-and-conquer nature of the algorithm, we have a time complexity of O(n log n) for this case. If most of the given points are already on or very close to the convex hull, the algorithm quickly eliminates interior points without unnecessary recursive calls. The algorithm efficiently divides the set into two balanced subsets, reducing the problem size logarithmically at each step.

The best case is when the line divides the two balanced parts, resulting in the recurrence relation of T(n) = 2 T(n/2) + O(n).

Solving this gives T(n) = O(n log(n)).

Example: Points forming a well-distributed convex shape (e.g., a rectangle or a convex polygon). [4,12]

## Graham’s Scan

Graham’s scan has O (n log n) complexity. The calculation of the lowest, leftmost point has θ(n) operation. This can be safely claimed because it must, by necessity, iterate through all the points in the input set. It is a simple heap sort and has the complexity class of O(n log n). It has the complexity of O(n). The two nested loops give the impression of great complexity but, in fact, pose little complication.

Thus, the total complexity of the algorithm is O(1) + θ(n) + O(n log n) + O(n) + O(1) + O(1) + O(1) + O(n) + O(n) = O(n log n). It is interesting to note that the complexity of the algorithm does not depend on the part of the algorithm that does the calculation of the hull itself, but instead on the sorting step, which is the dominant member, complexity-wise.

Unlike QuickHull, which has a worst-case of O(N²), Graham’s Scan is always O(N log N) in all cases.

**Case 1 Worst Case:** In the worst-case scenario (e.g., all points forming a large convex shape), sorting still dominates,resulting in O(N log N) complexity.

**Case 2 Average Case**: In a general distribution of points, sorting takes O(N log N), and hull construction takes O(N), leading to O(N log N) overall.

**Case 3 Best Case:** If the points are already sorted by angle, the sorting step can be skipped, but sorting still dominates. The hull construction takes O(N), so the final complexity remains O(N log N).[6,13]

## Jarvis’s March (Gift Wrapping)

The Jarvis’s March algorithm is a simple and intuitive approach to constructing the convex hull. While its time complexity is (𝑛ℎ) where *n* is the number of input points and *h* is the number of points on the convex hull, it is particularly efficient for smaller datasets or cases where the convex hull contains only a small number of points.

Unlike Quickhull, it iterates through all points to find the next convex hull point, leading to O(Nh) to O(N²) complexity.

**Case 1 Worst Case:** If every point is on the convex hull (h ≈ N), then for each point, the algorithm checks all other N points, leading to O(N²) complexity.

**Case 2 Average Case**: Where h is the number of points on the convex hull. Typically, h << N, making this slower than O(N log N) algorithms, which comes down to O(Nh) complexity.

**Case 3 Best Case:** If all points are already part of the convex hull (e.g., forming a convex polygon), the algorithm only scans them once, leading to O(N) complexity.[12]

## Monotone Chain (Andrew’s Algorithm)

Andrew’s Monotone Chain algorithm is a method of finding the convex hull of a finite set of points in the 2D plane with time complexity O(n logn). The Monotone Chain algorithm constructs the convex hull in O(N log N) time by first sorting the points and then constructing the upper and lower hulls separately. This algorithm is also known as Andrew's algorithm.

Unlike Quickhull, it uses sorting and a two-pass approach to construct the convex hull in O(N log N) consistently.

**Case 1 Worst Case:** Sorting still dominates, ensuring O(N log N) complexity.

**Case 2 Average Case**: Sorting step O(N log N) and linear hull construction O(N).

**Case 3 Best Case:** Sorting dominates, and hull construction takes O(N). [12]

# space complexity

## Quick Hull

Consider a convex hull problem consisting of n data points, m of which are hull vertices. Every non-empty call to QuickHull (p, q, R) results in a partitioning of R into three sets (one for each of the two recursive calls to QuickHull, as well as one for the internal points), without duplication. Furthermore, since O(m) calls are required for the complete algorithm, no more than O(m) activation records will exist at any one time. Therefore, the activation records collectively contain no more than O(n) pieces of information, and an efficient implementation of the algorithm will require only O(n) memory. The implementation of listing 1 depends upon the automatic reclamation and re-use of storage space (released by the removal of points from a set) to achieve the above space complexity. [2]

## Graham’s Scan

The Graham scan algorithm is used to find the convex hull of a finite set of points in a plane. The algorithm uses a stack to remove concavities in the boundary efficiently. The algorithm finds all vertices of the convex hull ordered along its boundary. The algorithms run in O(n) time and O(n) space complexity. [15]

## Jarvis’s March (Gift Wrapping)

The space complexity of Jarvis’s March depends on how many points are stored in the convex hull. The algorithm does not use extra data structures like recursion stacks or additional arrays. It only requires a few constant variables for computations (like tracking the next hull point), making auxiliary space O(1).

If all points are part of the convex hull, we store all **N** points.

**Worst Case:**

If all points are part of the convex hull, we store all **N** points.

O(N)

**Average/Best Case:** If only a few points are on the convex hull, we store only those.

O(h)

**Auxiliary Case:** Uses a few extra variables, but no additional large data structures.

O(1) [12]

## Monotone Chain (Andrew’s Algorithm)

The runtime complexity is O(nlogn), with n being the number of input points. If the point set is already sorted (by x-coordinate), the runtime complexity is O(n).

**Best case:** when all points form the hull O(n)

**Worst case:** when few points form the hull O(n)

**Average case:** O(n). [16, 17]

# Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Best Case Time Complexity | Worst Case Time Complexity | Average Case Time Complexity | Space Complexity |
| QuickHull | O(nlogn) | O(n2) | O(nlogn) | O(n) |
| Graham’s Scan | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| Monotone Chain | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| Jarvis March | O(n) | O(n2) | O(nh) | O(n) |

1. Comparative chart for various algorithms of convex hull

## Quick Hull

Uses divide-and-conquer to find extreme points and recursively partition. Worst case occurs when all points lie on the hull, leading to deep recursion. Efficient for randomly distributed points.

## Graham’s Scan

Sorting dominates the runtime. Always requires sorting of points, followed by a linear scan. Works well for large datasets but needs sorting even if unnecessary.

## Jarvis March

Iteratively finds the next hull point, making it efficient for small convex hulls. Worst case occurs when almost all points are on the hull, leading to quadratic time complexity

## Monotone Chain

Similar to Graham’s Scan, but constructs the upper and lower hull separately. Sorting step remains the bottleneck. Slightly simpler to implement than Graham’s Scan.

# Time taken to compute quick hull at different number of points

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Points | Runtime | | | |
| QuickHull | Graham’s Scan | Monotone Chain | Jarvis March |
| 20 | 1.306 s | 0.739 s | 0.251 s | 0.179 s |
| 100 | 1.034 s | 0.569 s | 0.173 s | 0.172 s |
| 150 | 0.712 s | 0.314 s | 0.274 s | 0.164 s |
| 200 | 0.343 s | 0.518 s | 0.226 s | 0.266 s |
| 300 | 0.298 s | 0.393 s | 0.237 s | 0.214 s |
| 400 | 0.631 s | 0.248 s | 0.271 s | 0.222 s |
| 500 | 0.403 s | 0.266 s | 0.099 s | 0.253 s |
| 1000 | 0.596 s | 0.467 s | 0.492 s | 0.219 s |

1. TIME TAKEN TO COMPUTE QUICK HULL AT DIFFERENT NUMBER OF POINTS

From the above analysis of runtimes of different algorithms used to find convex hull on set of different number of points, it can be concluded that Jarvis March is quicker for small number of points. Monotone chain and Graham’s scan are faster than other algorithms; Quick hull is comparatively slower for small number of points, but if the points are well distributed and recursion is acceptable then Quick hull is better.

# Limitations

The Quickhull algorithm, while efficient on average, has a worst-case time complexity of O(n^2) and struggles with certain point distributions, particularly when points are arranged along a circle's border.

Poor Performance with Circular Point Distributions: When points are arranged along the border of a circle, the algorithm can fail to eliminate points effectively in each step, leading to the worst-case complexity.

Not Suitable for All Convex Hull Problems: While Quickhull is a good algorithm for computing the convex hull of a finite set of points in n-dimensional space, it may not be the best choice for all convex hull problems, especially those with specific constraints or requirements.

Floating-Point Arithmetic Issues: Like many computational geometry algorithms, Quickhull can be sensitive to floating-point arithmetic errors, which can lead to inaccuracies in the computed convex hull, especially in higher dimensions.[2, 4, 14, 18]

# Applications

Computer Visualization and Image Processing: Quick Hull is applied in computer vision and image processing to find the boundaries of the objects, their outlines, so it finds its application extensively in video games and in medical imaging, face detection where it is very crucial to define the outer boundary.

Path Finding: Quick Hull is applied to robotic navigation and autonomous vehicle systems through the detection of obstacles by mapping the convex boundaries surrounding them. This is very crucial when finding the shortest collision-free path since if the convex hull of a vehicle clears all the collisions ahead, so will the vehicle, and it is therefore mostly applied to path planning.

Visual Pattern Matching: Quick Hull is applied in clustering and feature extraction algorithms in pattern recognition and machine learning. It recognizes and classifies the object by finding its convex shapes, enhances its accuracy.

Geometry: Quick Hull is employed to find solutions to several issues such as Voronoi diagrams and Delaunay triangulation, it has an important role in geometric structure analysis in many fields such as game development and architecture.

Geographical Information System: Quickhull is used for terrain-mapping to estimate landforms and surface reconstruction, and boundary detection, navigation and route planning. [15]

# Conclusion

In this study, we implemented various algorithms to calculate convex hulls and confirmed their principles. In particular, there is a significant gap between the theoretical time complexity of the seven 2D convex hull algorithms (Graham Scan, Divide and Conquer, Jarvis’s March, QuickHull, Monotone Chain) and the actual performance. This was confirmed. (This seems to be due to the role of constant values that are not expressed in the 𝐵𝑖𝑔𝑂. BigO function and the use of complex functions for theoretical optimization.) In addition, the performance depends on the result value for algorithms that are sensitive to the result value (output-sensitive). Statistically significant differences were observed between the groups. This means that when we use the convex hull algorithm in an application, simply using only a theoretically optimized algorithm may decrease performance. Therefore, in each expected situation and environment, variables include how quickly the most frequently called function or command in the algorithm can be called and executed, expected result value, increase in constant value, etc. The algorithm that shows statistically optimal performance through simulation should be selected. In the case of the 3D convex hull algorithm, we looked at how the 2D convex hull algorithm can be applied to 3D cases. Compared with the 2D case, in the 3D case, the polyhedron can be uniquely determined from the points. Several differences, such as examining the direction between the surfaces and points instead of lines and points, were confirmed. It was also confirmed that thetime complexities of (𝑛ℎ)O(nh) and 𝑂(𝑛𝑙𝑜𝑔𝑛)O(nlogn) remained the same despite the additional steps, owing to this difference. Convex hull algorithms are commonly used to process and extract information from point clouds. Although Graham’s scan is stable and widely used in many fields, the deletion of a concave point requires the processing almost half of the points. [19]

##### References

1. E. Horowitz, S. Sahni and S. Rajasekaran “Computer Algorithms” 2nd-edition, Computer Science Press, 1998.
2. https://how.dev/answers/what-is-the-quick-hull-algorithm
3. https://www.geeksforgeeks.org/convex-hull-algorithm/
4. C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. “The quickhull algorithm for convex hulls” *ACM Trans. Math. Softw.* vol. 22, no. 4 pp. 469–483, Dec. 1996,. https://doi.org/10.1145/235815.235821
5. M. M. Atwah, J.W. Baker, and S.G. Akl, (1995). “An Associative Implementation of Graham's Convex Hull Algorithm”. *Parallel and Distributed Computing and Systems,* 1995.
6. V. B. Petrović, and D. Ivetic, “Education and out of the box thinking – linearization of Graham's scan algorithm complexity as fruit of education policy”, *UbiCC Journal (Special Issue of ICIT Conference)* 2011.
7. H. Kwon, S. Oh, and J.-W. Baek, “Algorithmic Efficiency in Convex Hull Computation: Insights from 2D and 3D Implementations” *Symmetry*, 16, 1590. 2024. https://doi.org/10.3390/sym16121590.
8. Z. Havens and D. P. Bates, “Convex Hulls: Comparing Theoretical and Practical Runtimes”, December 10, 2021.
9. A. Mahdi N. Alzubaidi, M. Saad and A-Saoud, “Gilbert-Johnson-Keerthi Algorithm For Computing The Shortest Distance Between Two 2d Convex Hull Polygons Based On Andrew’s Monotone Chain Hull Algorithm”, *European Academic Research* Vol. I, Issue 10/ January 2014.
10. The quickhull algorithm for convex hulls | ACM Transactions on Mathematical Software
11. "A Proof for a QuickHull Algorithm" by Jonathan Scott Greenfield
12. https://stackoverflow.com/questions/13524344/complexity-of-the-quickhull-algorithm
13. V. Tereshchenko, Y. Tereshchenko and D. Kotsur, "Point triangulation using Graham’s scan," *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, Galcia, Spain, 2015, pp. 148-151, doi: 10.1109/INTECH.2015.7173370.
14. https://www.geeksforgeeks.org/convex-hull-monotone-chain-algorithm/
15. A. N. Gamby and J. Katajainen. "Convex-Hull Algorithms: Implementation, Testing, and Experimentation" *Algorithms* vol. 11, no. 12: pp 195, 2018. https://doi.org/10.3390/a11120195
16. “Algorithm Implementation/Geometry/Convex hull/Monotone chain”, Wikibook, 16 May 2017.

<https://en.wikibooks.org/wiki/Algorithm_Implementation/Geometry/Convex_hull/Monotone_chain>

1. <https://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/org/apache/commons/math3/geometry/euclidean/twod/hull/MonotoneChain.html>
2. Y. Sabri and N. El Kamoun “Large-Scale Sensor Network Localization Based on Gift Wrapping algorithm.” Int. J. Comp. Sc. Inform.Sec. Vol. 15 No. 4, pp 35-43, April 2017

**Make sure to remove all placeholder and explanatory text from the template when you add your own text. This text should not be here in the final version!**