# *Determining Disease Outbreak Boundaries Using Divide and Conquer Convex Hull*

Emilio Justin - 13524043
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: emilio.justin54@gmail.com , 13524043@std.stei.itb.ac.id

***Abstract*— Describing where a disease outbreak is located, and not only how many cases occur, is an important part of disease surveillance. Given the geographic coordinates of reported cases, the spatial region an outbreak occupies can be summarized by its convex hull, the smallest convex polygon that encloses all the case points. This paper demonstrates, through a simple simulation, how the convex hull computed with a Divide and Conquer strategy can be used to determine the boundary of a disease outbreak. The set of case points is sorted and split recursively into halves whose hulls are solved independently, and adjacent hulls are merged by locating their upper and lower common tangents. The resulting hull is interpreted as the outbreak boundary, and its area and perimeter are reported as descriptive metrics; to handle several simultaneous outbreaks, the cases are first grouped using a distance-based clustering step so that each outbreak receives its own boundary. The method was implemented in Java with a simple graphical interface for entering case coordinates and visualizing the results. Results confirmed that the program produces the expected boundaries indicating that the Divide and Conquer convex hull is a simple, fast, and interpretable tool for determining outbreak regions.**

***Keywords—convex hull; divide and conquer; common tangent; disease outbreak; clustering***

## I. INTRODUCTION

Disease surveillance is concerned not only with *how many* cases occur, but also with *where* those cases are located. When the geographic coordinates of reported cases are plotted on a map, a recurring practical question arises, what is the spatial region that an outbreak currently occupies? Public-health practitioners use such regions to define containment zones, to allocate testing and vaccination resources, and to communicate risk to the public. A boundary that can be computed automatically and quickly from raw case coordinates is therefore a useful building block for any outbreak monitoring tool.

A natural and mathematically well-defined way to describe the extent of a set of points is the convex hull, the smallest convex polygon that encloses all the points. Intuitively, if one were to stretch an elastic band around all the case locations and release it, the band would settle onto the convex hull. For outbreak analysis, the hull is effective because it provides an easily interpretable boundary. Its enclosed area and perimeter directly map to the geographic scope of the affected region. Additionally, the method is non-parametric, eliminating the need for arbitrary tuning. This paper addresses the following problem: given a finite set of points $P = \{p_1, p_2, \ldots, p_n\}$ where each $p_i = (x_i, y_i)$ is the location of a reported case, determine the boundary polygon (or polygons) that represents the outbreak region.

The remaining question is how to compute the hull efficiently. A brute-force approach that inspects every pair of points to decide whether the segment between them is a hull edge costs $O(n^3)$, which becomes very slow as the number of cases grows. Several algorithms reach the optimal $O(n \log n)$ time, among them Graham's scan, Jarvis' march, and Divide and Conquer approach. This paper adopts the Divide and Conquer strategy because its structure is straightforward and familiar, much like Merge Sort. More importantly, it simplifies the problem by focusing the geometric logic into one key step, i.e. locating the common tangents to merge two adjacent hulls.

This paper gives an implementation of the convex hull using a Divide and Conquer strategy whose merge step finds the upper and lower common tangents between two sub-hulls; an application that interprets the resulting hull as an outbreak boundary, reports its area and perimeter, and supports multiple simultaneous outbreaks through distance-based clustering; and a simple graphical interface that accepts case coordinates.

## II. THEORETICAL FRAMEWORK

### *A. Divide and Conquer*

Divide and Conquer algorithm is a problem-solving technique used to solve problems by dividing the main problem into subproblems, solving them individually and then merging them to find solution to the original problem. Divide and Conquer is mainly useful when we divide a problem into independent subproblems.

Divide and Conquer Algorithm can be divided into three steps: **Divide**, **Conquer**, and **Merge**.

- *Divide:*
  - Break down the original problem into smaller subproblems.
  - Each subproblem should represent a part of the overall problem.
  - The goal is to divide the problem until no further division is possible.

- *Conquer:*
  - Solve each of the smaller subproblems individually.
  - If a subproblem is small enough, we solve it directly without further recursion.
  - The goal is to find solutions for these subproblems independently.
- *Merge:*
  - Combine the subproblems to get the final solution of the whole problem.
  - Once the smaller subproblems are solved, we recursively combine their solutions to get the solution of larger problem.
  - The goal is to formulate a solution for the original problem by merging the results from the subproblems.

The general structure of this algorithmic strategy can be modeled as follows:

**Divide and Conquer General Structure**

```
procedure DIVIDE_AND_CONQUER(P: problem, n: integer)
    if n ≤ n0 then
        // The problem size is small enough to be solved directly
        SOLVE problem P of size n
    else
        // Divide P into r subproblems: p1,p2,…,pr
        // of sizes n1,n2,…,nr respectively
        DIVIDE P into p1,p2,…,pr
        for each subproblem pi do
            DIVIDE_AND_CONQUER(pi, ni)
        // Combine the subproblem solutions into the final solution
        COMBINE solutions of p1,p2,…,pr
```

The time complexity of a standard Divide and Conquer algorithm is mathematically expressed using the following recurrence relation:

$$T(n) = \begin{cases} g(n), & if\ n \le n_0 \\ \Sigma_{i=1}^{r} T(n_i) + f(n), & if\ n > n_0 \end{cases}$$

Where the parameters are defined as follows:

1. $T(n)$ represents the total time complexity so solve a problem of size $n$.
2. $g(n)$ is the time required to directly solve the base case when the problem size is sufficiently small ($n \le n_0$).
3. $\Sigma_{i=1}^{r} T(n_i)$ denotes the total time needed to recursively process all $r$ subproblems, where $n_i$ is the size of each subproblem.
4. $f(n)$ is the time required to merge or combine the solutions of the subproblems into the final answer.
5. The time required to divide the problem is assumed to be $O(1)$.

## B. Convex Hull

Convex hull of a set of points in a Euclidean space is the smallest convex polygon that encloses all the points. In two dimensions (2D), convex hull is a convex polygon, and in three dimensions (3D), it is a convex polyhedron.

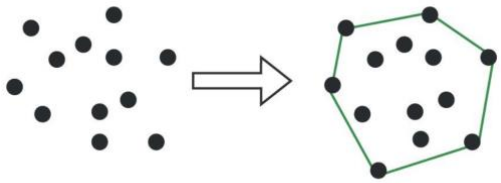

Fig. 1. Convex hull illustration
*source: https://www.geeksforgeeks.org/dsa/convex-hull-using-divide-and-conquer-algorithm/*

A region is said to be *convex* if, for any two points that lie within it, the entire straight-line segment connecting them also lies within the region. The convex hull of a point set $P$ is therefore the smallest convex region that contains every point of $P$, i.e. it is the intersection of all convex sets that contain $P$. A common and intuitive analogy is that of an elastic band. If a rubber band were stretched around all the points and then released, it would contract until it tightly wraps the outermost points, and the shape it settles into is precisely the convex hull.

Formally, the convex hull can be described as the set of all *convex combinations* of the points in $P$. Given points $p_1, p_2, \dots, p_n$ a convex combination is any point of the form

$$\Sigma_{i=1}^{n} \lambda_i p_i$$

where each coefficient satisfies $\lambda_i \ge 0$ and $\Sigma_{i=1}^{n} \lambda_i = 1$. The collection of all such points forms the convex hull.

## C. Tangent Lines Between Two Hulls

When two convex polygons are positioned side by side, there exist two special lines that touch both without passing between them, it is the upper tangent and the lower tangent. A tangent line, in this context, is a line that touches a convex polygon at one of its vertices while leaving the entire polygon on a single side of the line. A common tangent of two polygons is a line that is simultaneously tangent to both. The upper tangent rests on top of both polygons, so that both lie entirely below it, while the lower tangent sits beneath both, so that both lie entirely above it. These two tangents are important because, together, they form the outer "bridge" that joins two separate convex polygons into one.

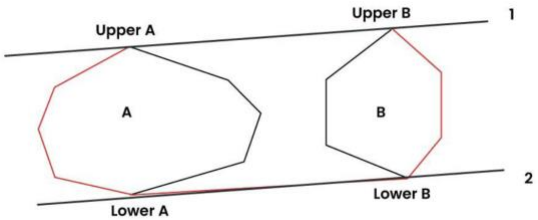

Fig. 2. Tangent line illustration
*source: https://www.geeksforgeeks.org/dsa/convex-hull-using-divide-and-conquer-algorithm/*

Finding the upper tangent can be described as a process of gradually lifting a line until it rests cleanly on top of both polygons. The process begins by drawing a line that joins the rightmost point of the left polygon to the leftmost point of the right polygon. In general, this initial line is not yet tangent, it cuts through one or both polygons. If the line still crosses the right polygon, the contact point on the right polygon is moved upward (in the anticlockwise direction) to the next vertex, which raises that end of the line. Whenever the line crosses the left polygon, the contact point on the left polygon is moved upward (in the clockwise direction) instead. These two adjustments are repeated alternately until the line no longer crosses either polygon, at which moment it has become the upper tangent.

The lower tangent is found by the mirror image of the same procedure. Starting again from the line joining the rightmost point of the left polygon and the leftmost point of the right polygon, the directions of movement are simply reversed: whenever the line crosses the right polygon its contact point is moved downward, and whenever it crosses the left polygon its contact point is moved downward on that side. The process terminates when the line clears both polygons from below, yielding the lower tangent. The decision of whether a line "crosses" a polygon, and of which direction counts as upward or downward, is made using the orientation of point triples described earlier, so the entire tangent search reduces to a sequence of simple geometric comparisons. Because each contact point only ever advances around its own polygon, the search visits each vertex a bounded number of times and therefore runs in time linear in the number of vertices.

### D. Convex Hull With Divide and Conquer

The Divide and Conquer strategy compute the convex hull of a point set by breaking the problem into smaller instances, solving each independently, and then merging the partial results. Suppose the convex hull of the left half of the points and the convex hull of the right half are already known. The remaining task is to merge these two hulls into a single convex hull that covers the complete set, and this is achieved precisely by finding the upper and lower tangents between the two hulls, as described in the previous subsection. The two tangents identify where the outer boundaries of the left and right hulls connect; the portions of each hull that face inward, between the tangent contact points, are discarded, and the remaining outer chains are joined through the two tangents to form the final hull.

This naturally leads to a recursive formulation. The set of points is first sorted by x-coordinate and then repeatedly split into a left half and a right half. The splitting continues until each subset contains only a small number of points, at which stage the hull of that subset is found directly by a simple brute-force method. The partial hulls are then merged back together, level by level, until a single hull for the entire set remains. The brute-force base case is straightforward but costly, with a time complexity of $O(n^3)$. However, because it is applied only to very small subsets, it does not affect the overall complexity of the algorithm.

A subtle but important detail concerns the size of the base case. One might expect that the hull of three or fewer points could simply be taken as the points themselves, and this is indeed correct in isolation. In practice, though, merging a hull of two points with a hull of three points can drive the tangent search into an infinite loop in certain special configurations. To avoid this problem, the convex hull of any subset of five or fewer points is computed directly with the brute-force method. This slightly enlarges the base case but eliminates the degenerate behavior without changing the asymptotic cost.

The efficiency of the approach follows from how the work is distributed across the recursion. Sorting the points once at the beginning takes $O(n \log n)$ time. Each merge step, being a linear-time tangent search, costs $O(n)$, and the recursion splits the problem in half at every level. This yields the recurrence $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, whose solution is $O(n \log n)$. The Divide and Conquer convex hull therefore match the optimal time bound for the problem and is faster than the naive $O(n^3)$ approach as the number of points grows.

## III. METHODS

### A. Scope and Limitations

This paper assumes a planar, two-dimensional setting. Therefore, geographic projection effects are not modeled. Furthermore, the generated boundaries are strictly convex. While this suits compact outbreaks, it may overestimate the affected area for highly irregular or concave spreads. Also, the current clustering method relies entirely on distance thresholds set by the user.

### B. Program Overview

The program operates on a straightforward pipeline. It takes raw case coordinates, optionally groups them into clusters, computes Divide and Conquer convex hull for each cluster, calculates spatial metrics, and displays the results.

The application is implemented in Java and structured into four primary packages: *model* (data types), *algorithm* (geometric processing and clustering), *io* (file handling), and *gui* (the visual interface). Users can input coordinate data in three ways: manual typing, loading from a csv/txt file, or random generation.

### C. Convex Hull and Outbreak Metrics

The boundary is computed using Divide and Conquer algorithm detailed in Section 2. First, the input points are deduplicated and sorted by their x-coordinates. The dataset is then recursively split into two halves until the base cases are reached. Finally, the adjacent sub-hulls are merged by locating their upper and lower common tangents. Each merge runs in linear time, so the overall procedure achieves $O(n \log n)$ time.

To ensure the polygon boundary only consists of genuine corners, collinear vertices are removed. Once the final hull is formed, the program calculates the enclosed area using shoelace formula and the perimeter by summing the edge lengths.

### D. *Multi-Cluster Outbreaks*

Real-world surveillance data often contains multiple, separated outbreaks. To prevent the algorithm from generating a single massive hull that includes empty spaces, the program implements a distance-based clustering feature.

Using a Union-Find data structure, cases are grouped together if their Euclidean distance falls below a user-defined threshold. A larger threshold merges nearby cases, while a smaller one isolates them into distinct outbreaks. Each identified cluster is then processed individually and receives its own-colored boundary.

### E. *Visualization and Verification*

The graphical interface displays each case as a colored dot and the resulting boundaries as filled polygons. A status bar provides real-time reporting on the point count, cluster count, total area, perimeter, and the core hull computation time (measured in milliseconds).

To ensure absolute correctness across numerous random inputs, the output of Divide and Conquer approach was compared against an independent brute-force approach. This confirms that both approaches consistently produced identical boundaries.

## IV. Implementation and Experiment

### A. *Implementation*

The Divide and Conquer convex hull are implemented exactly as described in the theoretical framework. Points are deduplicated and sorted by x-coordinate, the set is split recursively until subsets of at most five points are reached (Figure 4), those small subsets are solved by an $O(n^3)$ brute-force hull, and adjacent hulls are merged by locating their upper and lower common tangents (Figure 5) through the orientation test (Figure 6). Collinear vertices are removed from the merged boundary, and degenerate configurations fall back to the brute-force hull, which guarantees that the procedure always terminates with a correct, well-formed boundary.

The graphical interface, shown in Figure 3, is divided into a control panel on the left, a canvas in the center, and a status bar along the bottom. The control panel provides a text area for entering coordinates, buttons to parse the text, to load a csv/txt file, and to generate random points, a checkbox to enable multi-cluster mode together with a distance-threshold field, and buttons to compute the hull and to clear the canvas. When the hull is computed, the canvas plots each case as a colored dot and draws each outbreak boundary as a filled, outlined polygon, automatically scaling the data so that it fits the canvas area. The status bar then reports the number of points, the number of clusters, the total area the total perimeter, and the computation time.

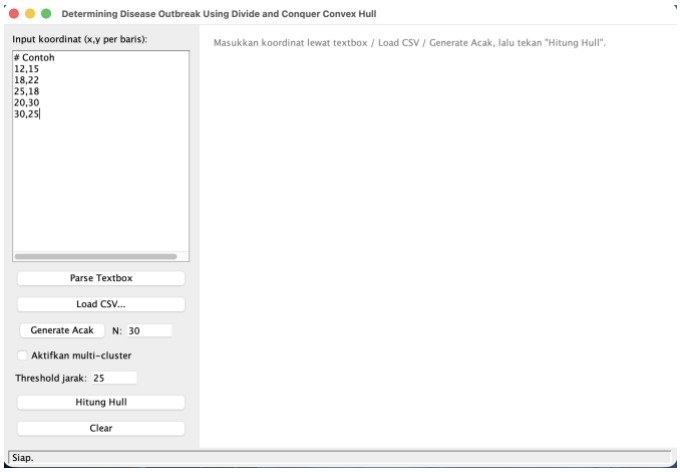


Fig. 3. Main window of the application
*source: Author's archive*

```
private static List<Point> divide(List<Point> a) {
    if (a.size() <= 5) {
        return bruteHull(a);
    }
    int mid = a.size() / 2;
    List<Point> leftHull = divide(new ArrayList<>(a.subList(fromIndex: 0, mid)));
    List<Point> rightHull = divide(new ArrayList<>(a.subList(mid, a.size())));
    return merge(leftHull, rightHull);
}
```

Fig. 4. Divide and conquer structure source code
*source: Author's archive*

```
int inda = ia, indb = ib;
boolean done = false;
int guard = 0;
while (!done) {
    done = true;
    while (orient(b.get(indb), a.get(inda), a.get((inda + 1) % n1)) >= 0) {
        inda = (inda + 1) % n1;
        if (++guard > guardLimit) return bruteHull(unionOf(a, b));
    }
    while (orient(a.get(inda), b.get(indb), b.get((n2 + indb - 1) % n2)) <= 0) {
        indb = (n2 + indb - 1) % n2;
        done = false;
        if (++guard > guardLimit) return bruteHull(unionOf(a, b));
    }
}
int upperA = inda, upperB = indb;
```

Fig. 5. Upper common tangent partial source code
*source: Author's archive*

```
private static int orient(Point a, Point b, Point c) {
    double res = (b.y - a.y) * (c.x - b.x) - (c.y - b.y) * (b.x - a.x);
    if (res > EPS) return 1;
    if (res < -EPS) return -1;
    return 0;
}
```

Fig. 6. Orientation test used in tangent search source code
source: Author's archive

### B. *Testing*

The program was tested using a set of test cases, each chosen to examine a particular behavior of the algorithm. For every case the coordinates were entered into the program, the hull was computed, and the resulting boundary together with its area and

perimeter was captured as a screenshot. The cases and their results are summarized in Table 1 and discussed afterward.

TABLE I. TEST CASES RESULTS SUMMARIZED

| Test | N | Multi-cluster | Threshold | Clusters | Total area | Perimeter | Time (ms) |
|---|---|---|---|---|---|---|---|
| TC1 | 20 | off | - | 1 | 2639.39 | 219.77 | 2.446 |
| TC2 | 100 | off | - | 1 | 5922.18 | 301.97 | 0.799 |
| TC3 | 60 | on | 20 | 2 | 2950.74 | 300.99 | 0.396 |

For the first test case involving a single outbreak with a small dataset, twenty random points were generated with the multi-cluster mode disabled. The program successfully produced a single boundary that tightly enclosed all the points, where the inner points remained inside and only the outermost points formed the corners, confirming the basic geometric behavior of the convex hull on scattered data as shown in Figure 7.

The second test case repeated the process with one hundred random points while keeping the multi-cluster mode disable. As shown in Figure 8, the program again generated a single correct boundary because only the outermost points appeared on the polygon while the much large number of interior points had no effect on its final shape, demonstrating that the algorithm handles a larger input size correctly.

The third test case evaluated multiple outbreaks by generating sixty random points distributed around two separate centers, creating two visually distinct groups. With the multi-cluster mode enabled and the distance threshold set to 20, the program successfully identified two separate clusters and drew an individual-colored boundary for each group rather than a single hull spanning the empty space between them, as shown in Figure 9.

Comparing the results of this multi-cluster setup with the previous single-boundary test directly highlights why clustering is useful. Without this feature, the two separate outbreaks would be merged into one heavily overstated region, whereas clustering allows each outbreak to be delineated independently. Furthermore, adjusting the threshold changed the grouping as expected, where a large threshold merged the groups together and a smaller threshold split them into even smaller fragments.

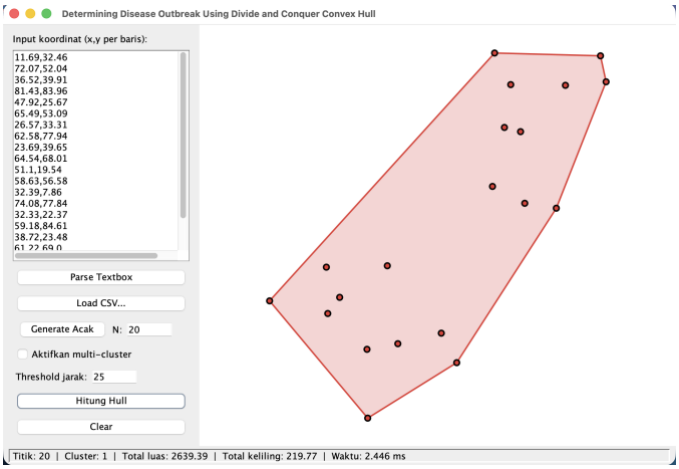


Fig. 7. Test case 1 result
*source: Author's archive*

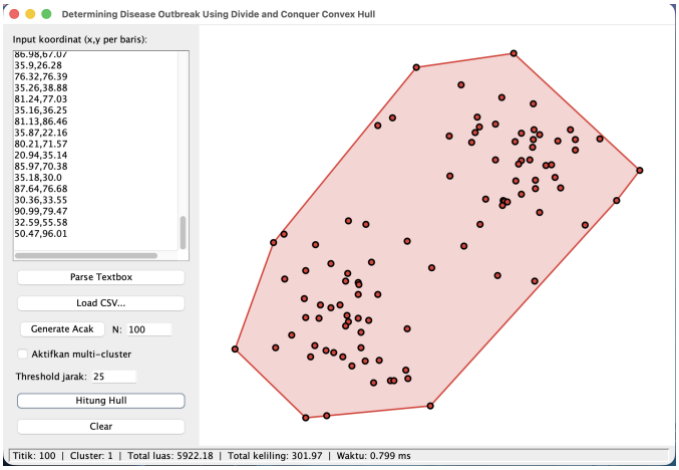


Fig. 8. Test case 2 result
*source: Author's archive*


Fig. 9. Test case 3 result
*source: Author's archive*

## V. Conclusion

This paper demonstrated how the convex hull computed with Divide and Conquer strategy can be applied to determine the boundaries of a disease outbreak from the coordinates of reported cases. The set of case points is sorted by x-coordinate and split recursively into halves whose hulls are solved independently, and adjacent hulls are combined by locating their upper and lower common tangents. The resulting hull is interpreted as the boundary of the affected region, and its area and perimeter are reported as descriptive metrics of the outbreak. To handle several simultaneous outbreaks, a distance-based clustering step groups the case before the hull is computed, so that each separate outbreak receives its own boundary. The whole method was packaged in a simple graphical interface that visualizes the computed boundaries.

The experiments showed that the implementation behaves correctly. The test cases confirmed that the program produces the expected boundaries, correctly excluding interior points and separating distinct outbreaks, and an automated comparison against an independent brute-force implementation agreed on every random input. The computation completed within a few milliseconds in all the test cases, and the analysis in Section 2 establishes that the algorithm runs in $O(n \log n)$ time. These results confirm that convex hull computed by Divide and Conquer approach is a simple, fast, and interpretable tool for determining outbreak regions.

The program developed in this paper is still a simple simulation, and several of its limitations point naturally to future development. Because the boundary is always convex, it overestimates the affected area for outbreaks that spread in a strongly concave or fragmented pattern, so concave boundaries such as alpha shapes could be explored to obtain a tighter fit. The clustering currently relies on a single distance threshold chosen by the user. A density-aware method such as DBSCAN could remove this manual tuning and adapt better to uneven case densities. The planar assumption also means the metrics are expressed in abstract coordinate units rather than true geographic distances, so incorporating real map projections and overlaying the boundaries on an actual map would make the results more realistic. Beyond these, the simulation could be extended with larger and real-world datasets and with an animation of how an outbreak evolves over time, developing the present prototype into a more practical tool.

## Appendix

1. Source Code : https://github.com/Valz0504/Determining-outbreak-boundaries-using-convexhull.git
2. Video : https://youtu.be/ey03e4F0-Nw

## Acknowledgment

The author would like to express sincere gratitude to Dr. Ir. Rinaldi, M.T., lecturer of IF2211 Algorithm Strategy course, for his guidance, encouragement, and support throughout the semester. Thanks to his support, the author was able to gain valuable insights and learn many new things throughout the course. Next, the author would thank to family and friends who have been keep motivating and supporting. Finally, the author acknowledges the use of artificial intelligence tools to assist in language refinement and brainstorming.

## References

[1] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," Communications of the ACM, vol. 20, no. 2, pp. 87–93, 1977.

[2] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, Computational Geometry: Algorithms and Applications, 3rd ed. Berlin, Germany: Springer-Verlag, 2008.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA, USA: MIT Press, 2022.

[4] GeeksforGeeks, "Introduction to Divide and Conquer Algorithm," GeeksforGeeks, 2024. [Online]. Available: https://www.geeksforgeeks.org/dsa/introduction-to-divide-and-conquer-algorithm/. [Accessed: 16-Jun-2026].

[5] GeeksforGeeks, "Convex Hull Algorithm," GeeksforGeeks, 2024. [Online]. Available: https://www.geeksforgeeks.org/dsa/convex-hull-algorithm/. [Accessed: 16-Jun-2026].

[6] GeeksforGeeks, "Tangents between two Convex Polygons," GeeksforGeeks, 2024. [Online]. Available: https://www.geeksforgeeks.org/dsa/tangents-two-convex-polygons/. [Accessed: 16-Jun-2026].

[7] GeeksforGeeks, "Convex Hull using Divide and Conquer Algorithm," GeeksforGeeks, 2024. [Online]. Available: https://www.geeksforgeeks.org/dsa/convex-hull-using-divide-and-conquer-algorithm/. [Accessed: 16-Jun-2026].

[8] R. Munir, "Algoritma Divide and Conquer (Bagian 1)", [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/07-Algoritma-Divide-and-Conquer-(2026)-Bagian1.pdf, 2025, [Accessed: 16-Jun-2026]

[9] R. Munir, "Algoritma Divide and Conquer (Bagian 4)", [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/10-Algoritma-Divide-and-Conquer-(2026)-Bagian4.pdf ,2025, [Accessed: 16-Jun-2026]

## Statement

Hereby, I declare that this paper I have written is my own work, not a reproduction or translation of someone else's paper, and not plagiarized.

Bandung, 19 Juni 2026

Emilio Justin [13524043]