Convex Hull Problem Using Divide and Conquer

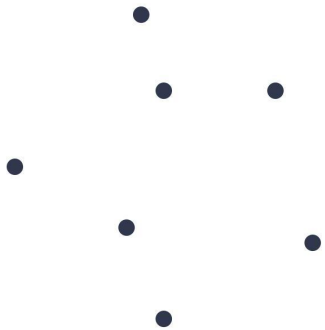
Team Members: Minyan Wang, Shreyas Kadekodi

Outline

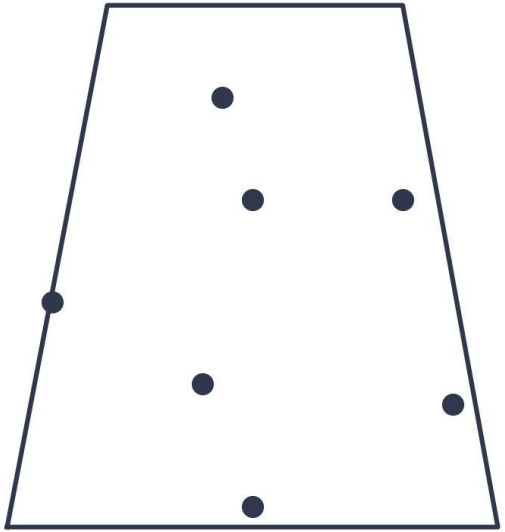
- Introduction and Motivation
- Brute Force Approach and Other Methods
- Divide and Conquer
- Code Demonstration
- Parallel Implementation

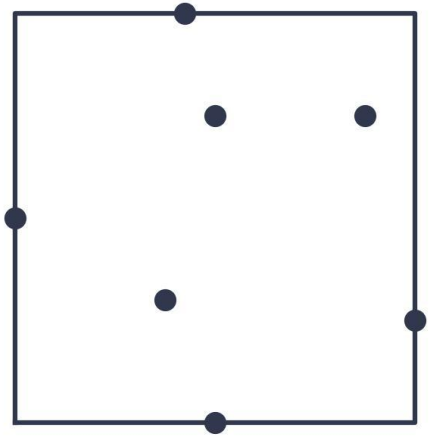
Introduction - What is a 2D convex hull?

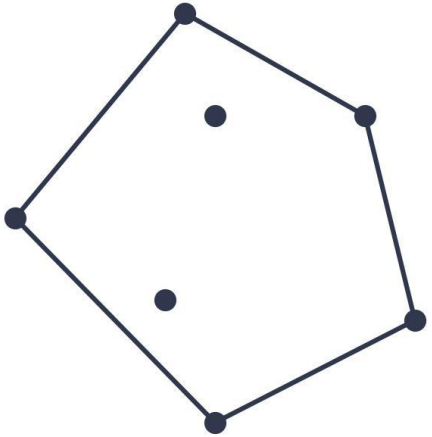
Suppose you have a set of points



Introduction - What is a 2D convex hull?

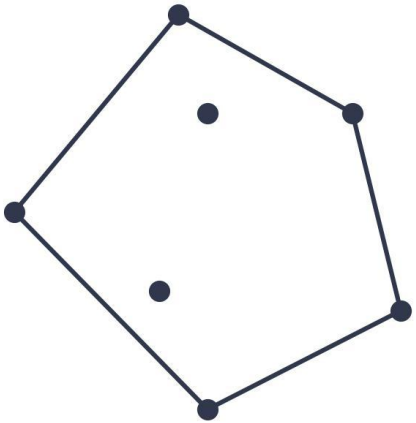


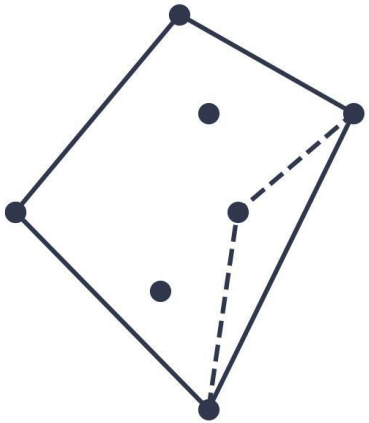


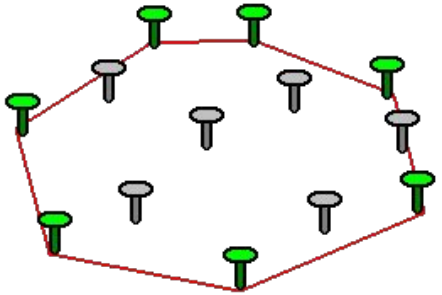


The smallest polygon that contains all the points

Introduction - What is a 2D convex hull?







The polygon has to be convex not concave.

Motivation

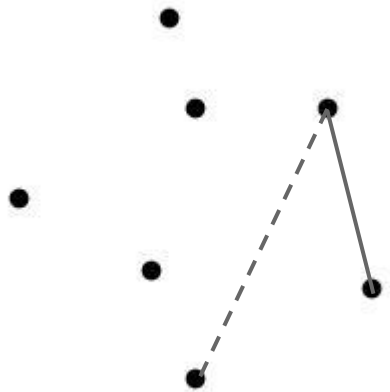
Applications:

- Image processing
- Robotics
- Path Planning
- Optimization

Outline

- ~~Introduction and Motivation~~
- Brute Force Approach and Other Methods
- Divide and Conquer
- Code Demonstration
- Parallel Implementation

Brute Force Approach



n points

$O(n^2)$ segments

O(n) test complexity

→ Overall $O(n^3)$

* Segment is the line connected by two points

Methods so far

Algorithm	Complexity	Proposed by
Graham Scan	$O(n^2)$	Graham, 1972
Jarvis March	O(nh)	Jarvis, 1973
Quick Hull	O(nh)	Eddy, 1977; Bykat, 1978
Divide and Conquer	**O(nlogn)**	**Preparata & Hong, 1977**
Monotone Chain	O(nlogn)	Andrew, 1979
Incremental	O(nlogn)	Kallay, 1984
Marriage before Conquest	O(nlogn)	Kirkpatrick & Seidel, 1986
n = number of points, h = number of vertices in the output hull		

Outline

- ~~Introduction and Motivation~~
- ~~Brute Force Approach and Other Methods~~
- Divide and Conquer
- Code Demonstration
- Parallel Implementation

Divide and Conquer

```
function convex_hull(points):
    if size(points) <= 4:
        return base_case(points)
    else:
        left_points, right_points = divide(points)
        left_hull = convex_hull(left_points)
        right_hull = convex_hull(right_points)
        return merge(left_hull, right_hull)


function base_case(points):
        Use graham scan method

function merge_hulls(left_hull, right_hull):
        Combine points
```

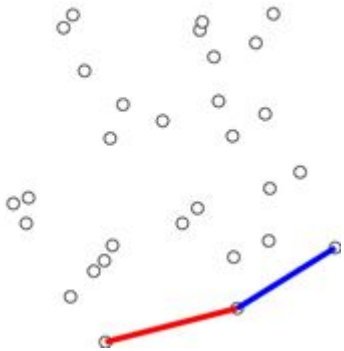
Divide and Conquer- Division

- Division is trivial
- Same divide-and-conquer strategy you've seen a million times

```
def convex_hull_2d(points):
    # Base Case
    if len(points) <= 5:
        return find_convex_hull_graham(points)
    #Divide (and Conquer)
    else:
        midpoint = int(len(points)/2)
        left_hull = convex_hull_2d(points[:midpoint])
        right_hull = convex_hull_2d(points[midpoint:])
    return merge_hull(left_hull, right_hull)
```

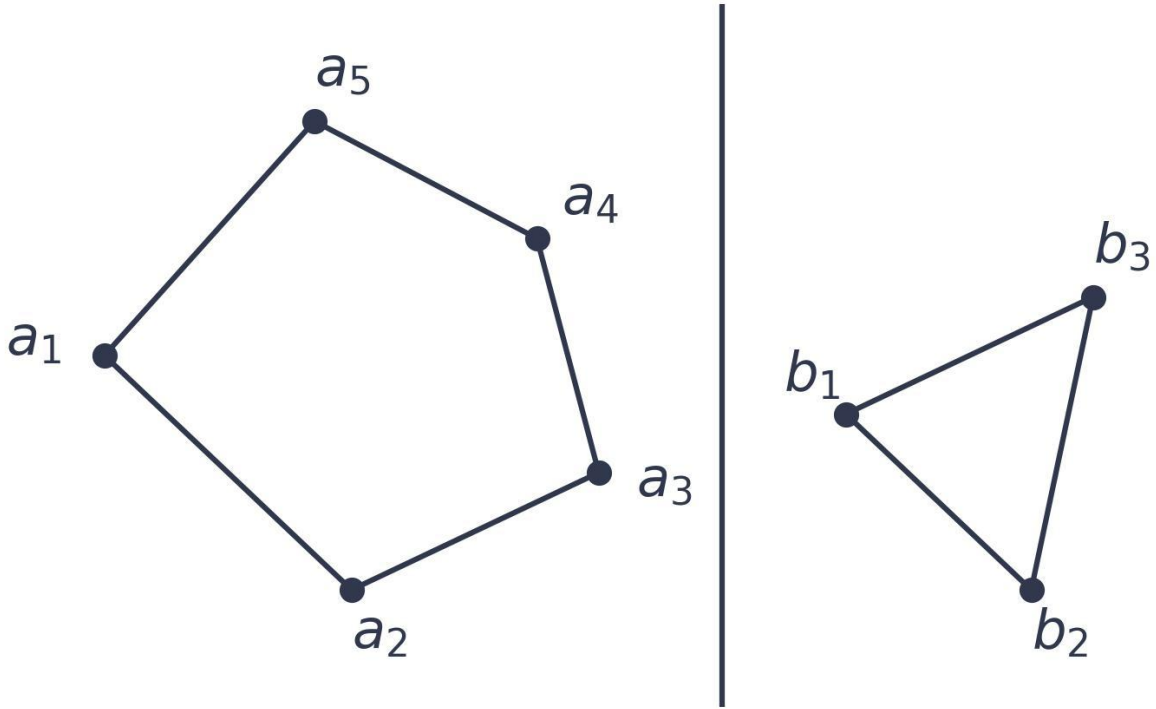
Divide and Conquer - Base Case

- Base case for 4 or less nodes — Graham scan
- Algorithm starts by selects lowest y-coordinate and designating it as the pivot.
- The sorted points are processed one by one, and those that form a counter-clockwise turn with the previous two points are added to the convex hull.
- If a point forms a clockwise turn, it is removed from consideration and the algorithm moves on to the next point.
- Algorithm terminates when it reaches the pivot point



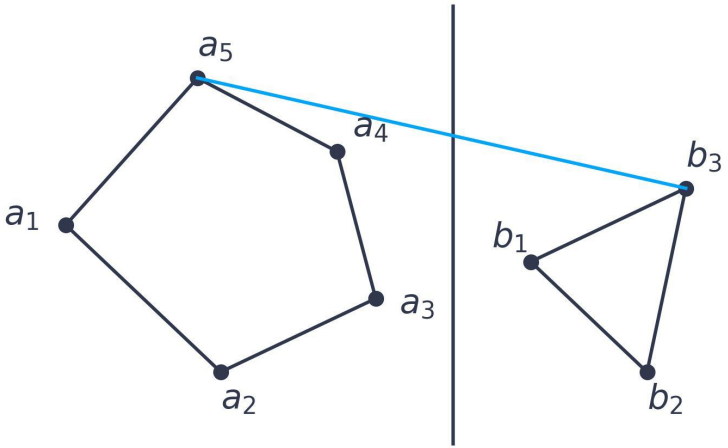
Credit: Wikipedia

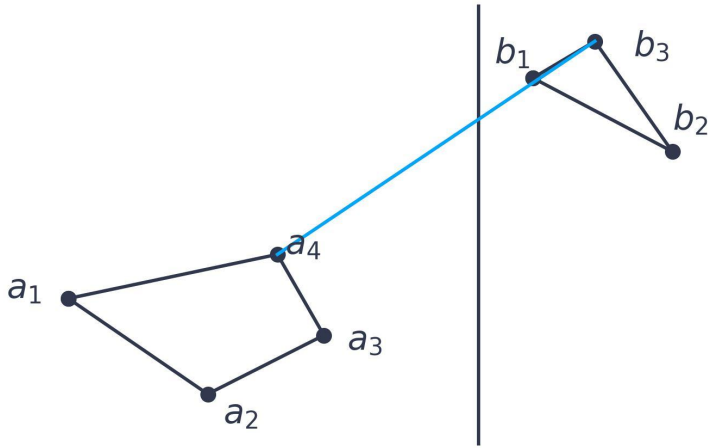
Divide and Conquer - How to Merge?



Divide and Conquer - How to Merge?

For the upper tangent, can we just pick the highest two points on the left and right respectively?

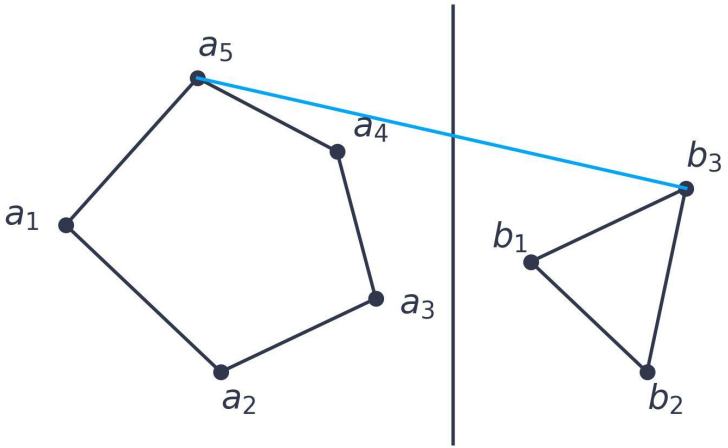


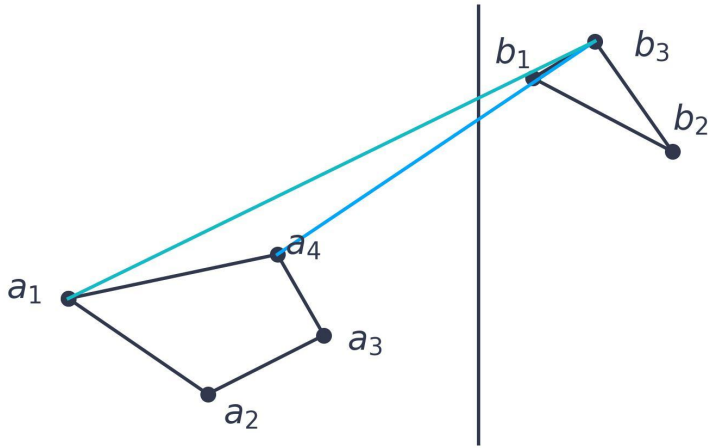


Divide and Conquer - How to Merge?

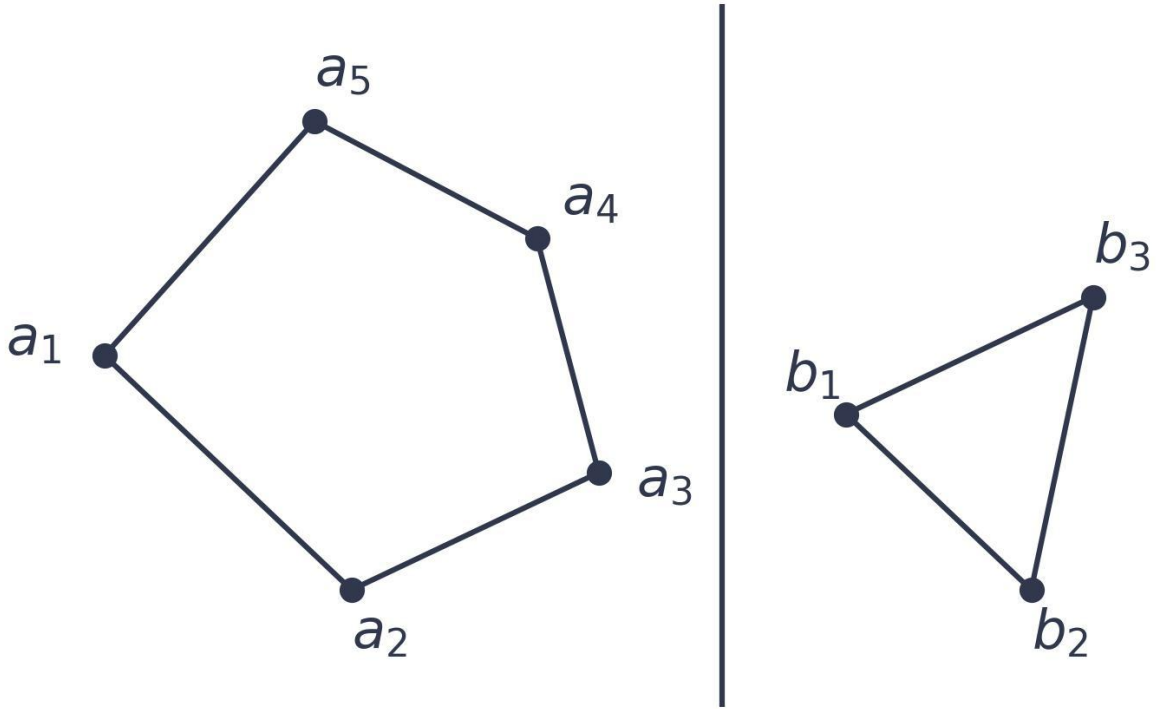
For the upper tangent, can we just pick the highest two points on the left and right respectively?

NO



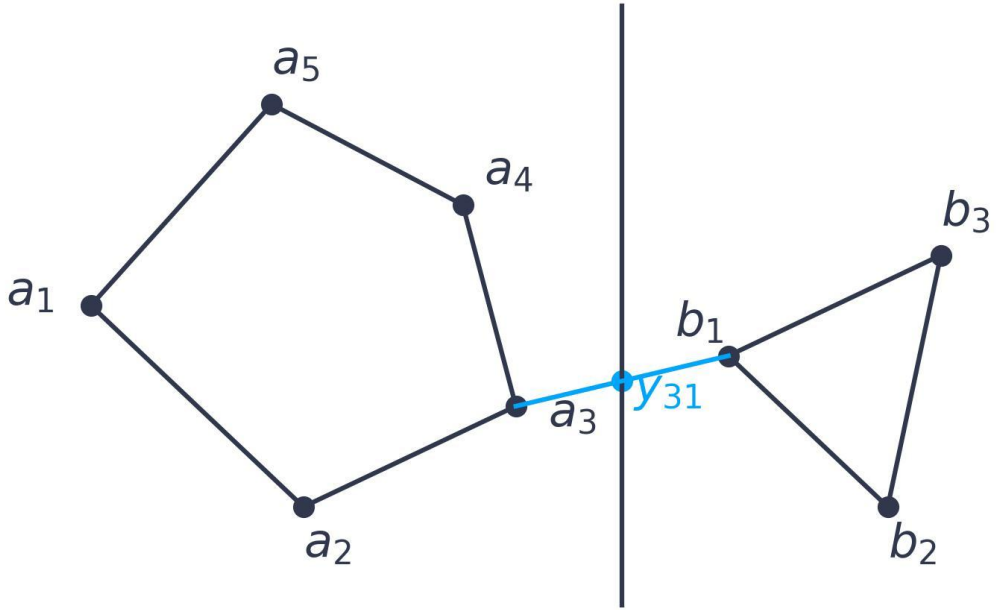


Divide and Conquer - How to Merge?



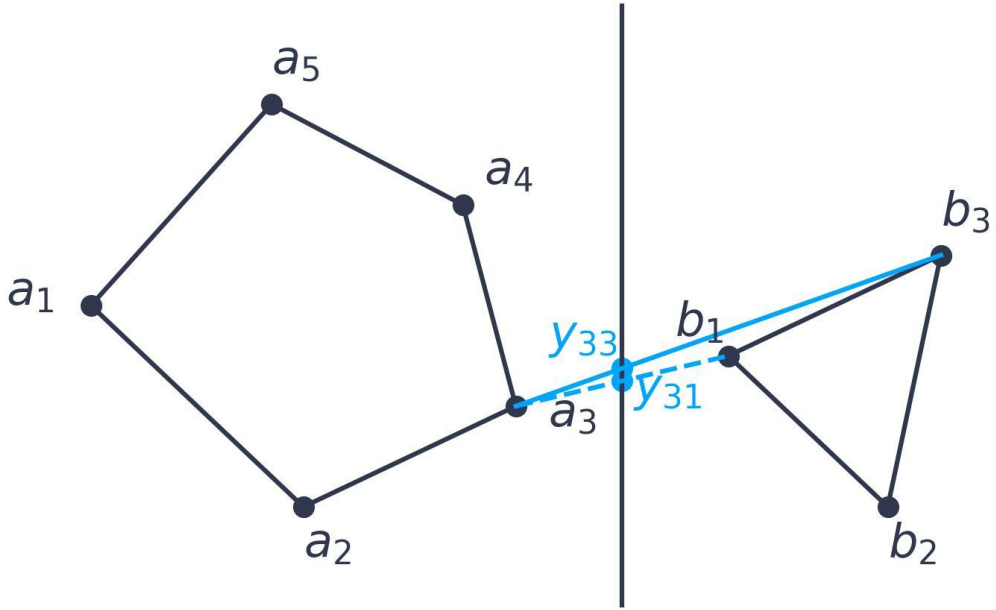
Divide and Conquer - How to Merge?

- Look at the segment connected by the two points closest to the middle line
- Record the y intercept with the middle line



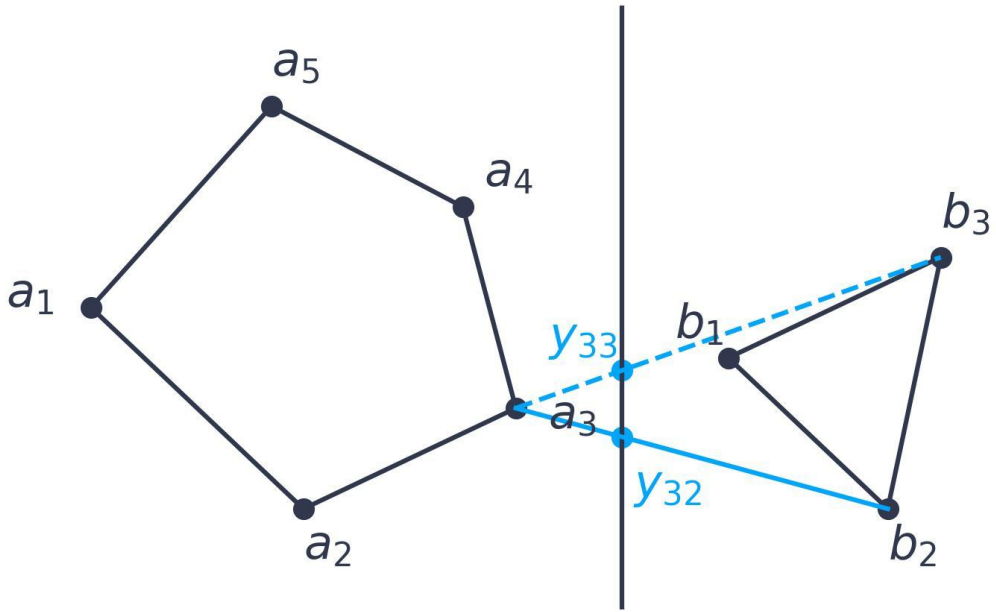
Divide and Conquer - How to Merge?

- Fix a_i, start to iterate over b_j clockwise
- If the new intercept is greater, keep it. Otherwise, stop iteration



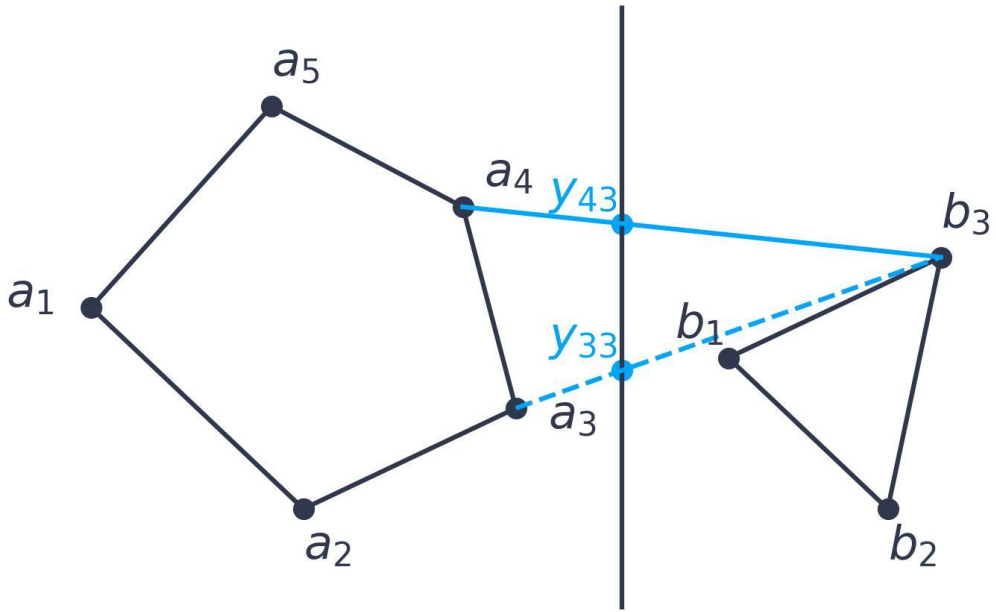
Divide and Conquer - How to Merge?

- Fix a_i, start to iterate over b_j clockwise
- If the new intercept is greater, keep it. Otherwise, stop iteration



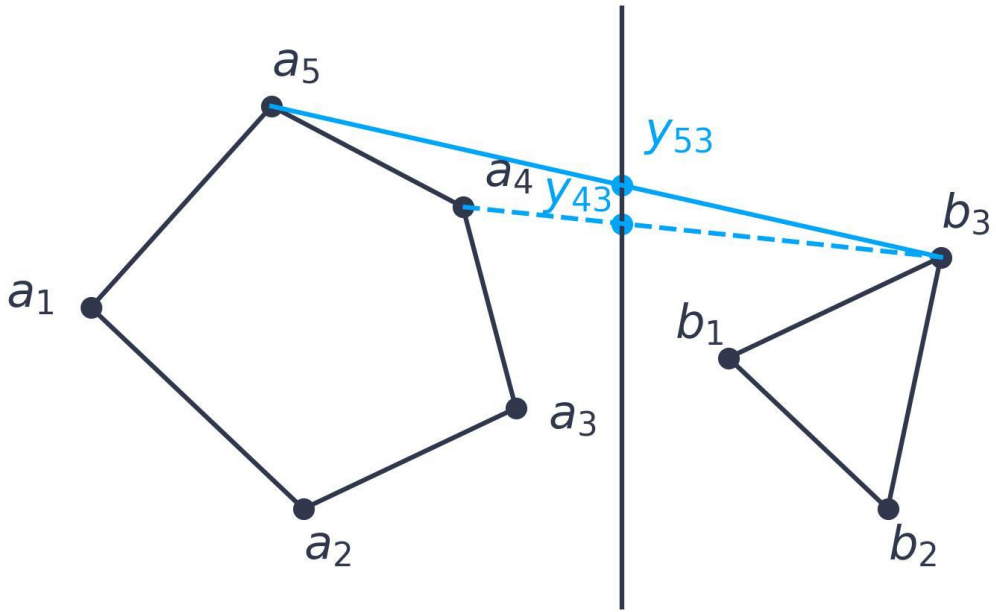
Divide and Conquer - How to Merge?

- Fix b_j, start to iterate over a_i counterclockwise
- If the new intercept is greater, keep it. Otherwise, stop iteration



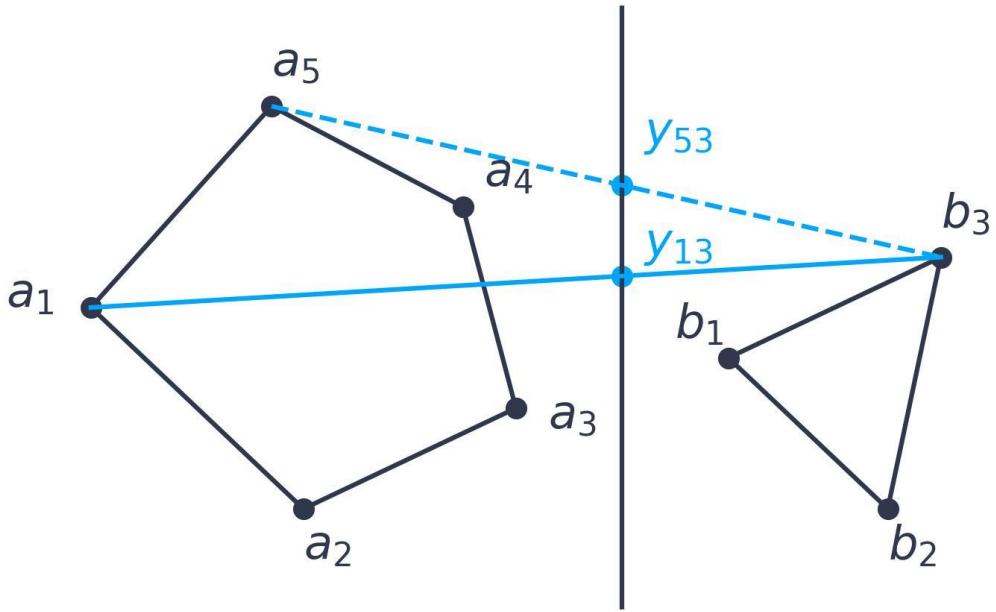
Divide and Conquer - How to Merge?

- Fix b_j, start to iterate over a_i counterclockwise
- If the new intercept is greater, keep it. Otherwise, stop iteration



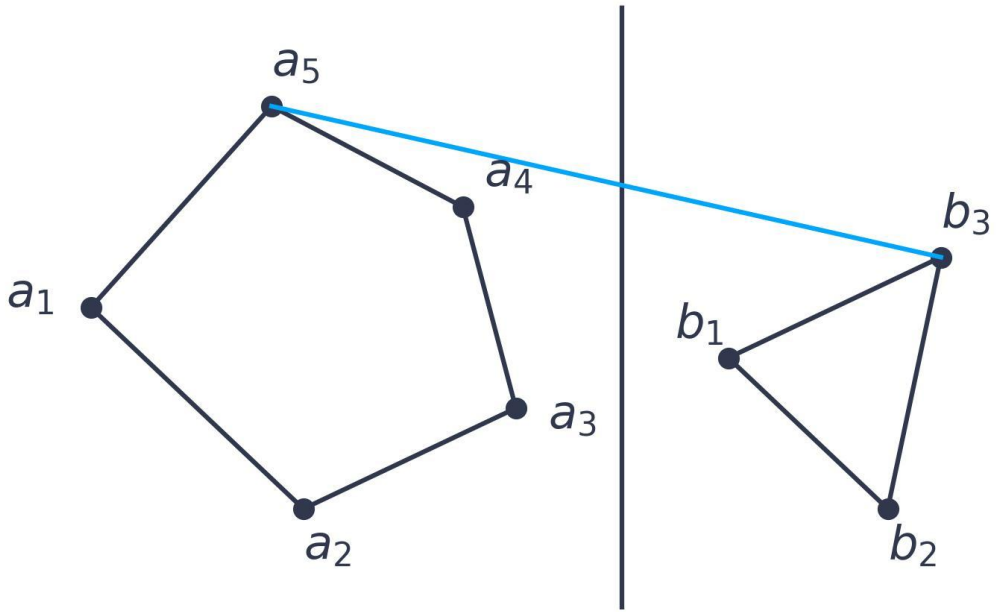
Divide and Conquer - How to Merge?

- Fix b_j, start to iterate over a_i counterclockwise
- If the new intercept is greater, keep it. Otherwise, stop iteration



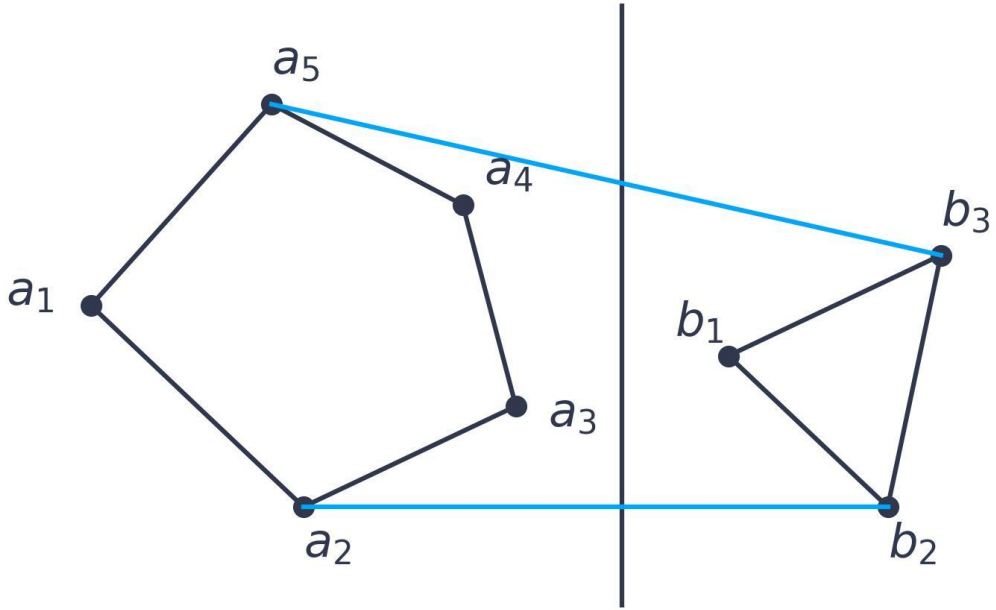
Divide and Conquer - How to Merge?

- Upper tangent determined



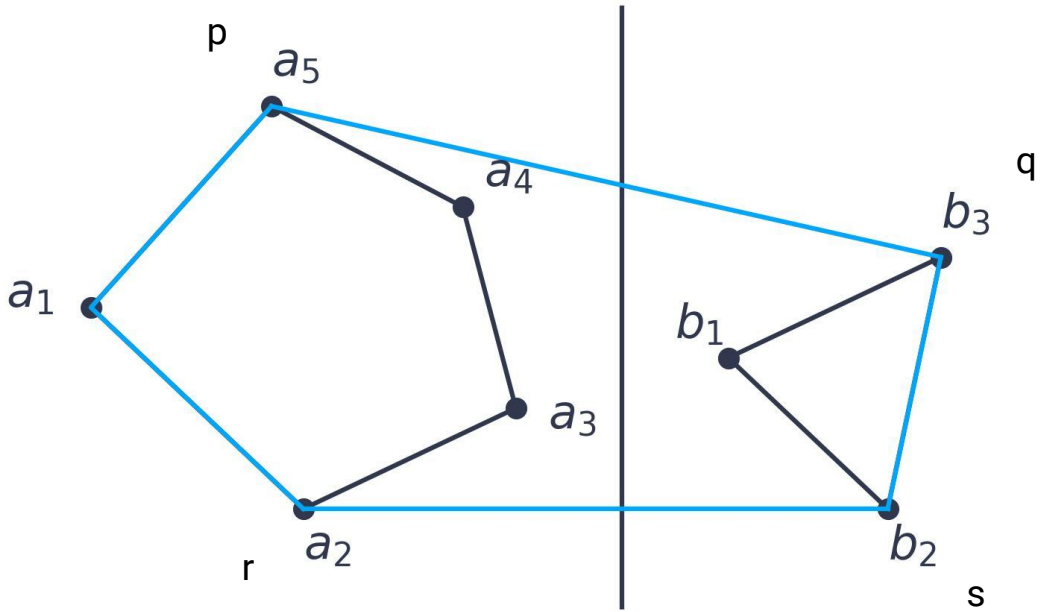
Divide and Conquer - How to Merge?

- Determine lower tangent in the same way



Divide and Conquer - How to Merge?

- Connect other points in the overall hull
- $a_p \rightarrow b_q \rightarrow \ldots \rightarrow b_s \rightarrow a_r \rightarrow \ldots \rightarrow a_p$



Outline

- ~~Introduction and Motivation~~
- ~~Brute Force Approach and Other Methods~~
- ~~Divide and Conquer~~
- Code Demonstration
- Parallel Implementation

Code Demonstration

Outline

- ~~Introduction and Motivation~~
- ~~Brute Force Approach and Other Methods~~
- ~~Divide and Conquer~~
- ~~Code Demonstration~~
- Parallel Implementation

Parallel Implementation

- The multiprocessing module in Python provides a way to spawn multiple processes to perform tasks in parallel.
- It allows for the efficient use of multiple CPU cores and can greatly speed up programs that perform CPU-bound tasks.
- The multiprocessing module provides a Process class and a Pool class that can be used to create a pool of worker processes that can be used to parallelize tasks.

Time Savings

- 100,000,000 points in seconds

	DnQ	DnQ - Parallel	Jarvis
Dataset 1	603	502	1120
Dataset 2	588	556	579
Dataset 3	655	532	1092

References

- "Shape Analysis and Classification: Theory and Practice" by Luciano Da Fona Costa and Roberto Marcondes Cesar Jr. (Springer, 2001).
- "Robotics: Modelling, Planning and Control" by Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo (Springer, 2010).
- "Geographic Information Systems and Science" by Paul A. Longley, Michael F. Goodchild, David J. Maguire, and David W. Rhind (Wiley, 2015).
- "Convex Optimization" by Stephen Boyd and Lieven Vandenberghe (Cambridge University Press, 2004).
- An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set", Information Processing Letters, volume 1, issue 4, 1972, pages 132-133

Thank you!