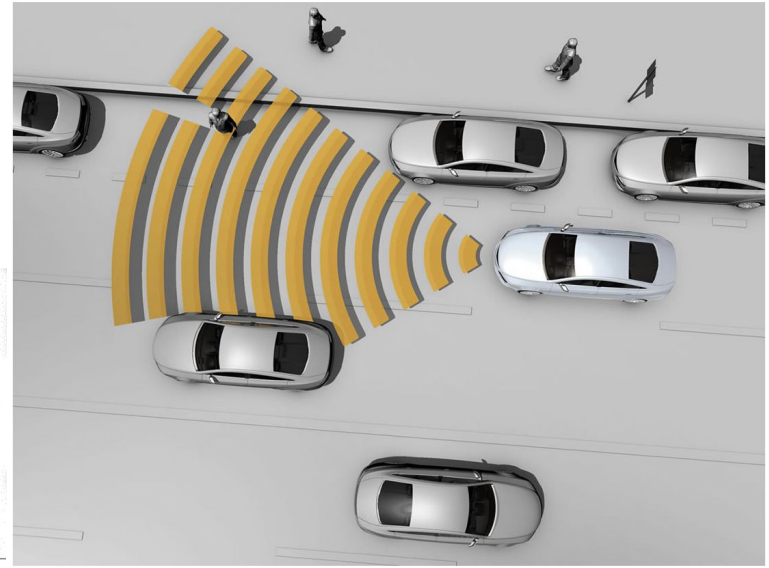
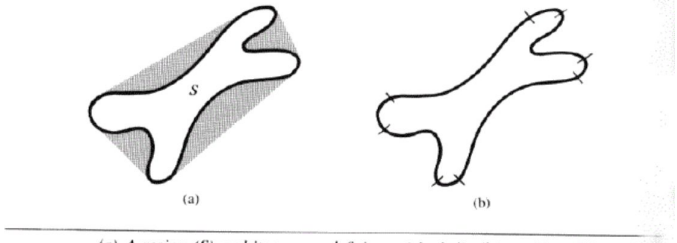
Divide & Conquer

L2. Convex Hull & Median Finding

Soyoung Chung

Real Life Applications Convex Hull





(a) A region (S) and its convex deficiency (shaded); (b) partitioned boundary.

Applications

A few of the applications of the convex hull are:

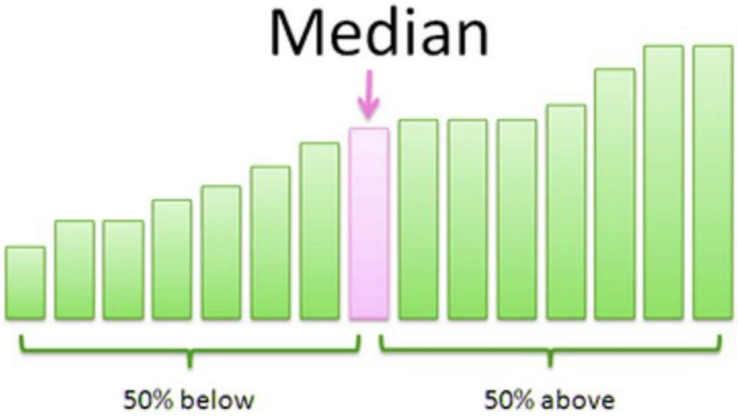
- **Collision avoidance**: If the convex hull of a car avoids collision with obstacles then so does the car. Since the computation of paths that avoid collision is much easier with a convex car, then it is often used to plan paths.
- **Smallest box**: The smallest area rectangle that encloses a polygon has at least one side flush with the convex hull of the polygon, and so the hull is computed at the first step of minimum rectangle algorithms. Similarly, finding the smallest three-dimensional box surrounding an object depends on the 3D-convex hull.
- **Shape analysis**: Shapes may be classified for the purposes of matching by their "convex deficiency trees", structures that depend for their computation on convex hulls.

WHERE CONVEX HULL IS USED

It is used in parallel computing, computational geometry, discrete mathematics, and computer science. A convex hull algorithm can be used for collision avoidance. As it helps particles avoid collision, it can be translated to real-life scenarios to avoid vehicle collisions in cars and airplanes. It is also getting used in image processing.

Real Life Applications Find Median

Index of Article (Click to Jump)



Paradigm

Given a problem of size n divide it into subproblems of size $\frac{n}{b}$, $a \geq 1$, $b > 1$. Solve each subproblem recursively. Combine solutions of subproblems to get overall solution.

$$T(n) = aT(\frac{n}{b}) + [\text{work for merge}]$$

Divide **Conquer**

a: # of subproblems
T(n/b): recursion
[work for merge]: combining solutions of subproblems

Convex Hull

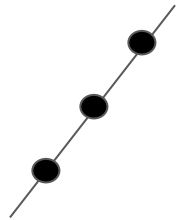
Given n points in plane

$$S = \{(x_i, y_i) | i = 1, 2, \ldots, n\}$$

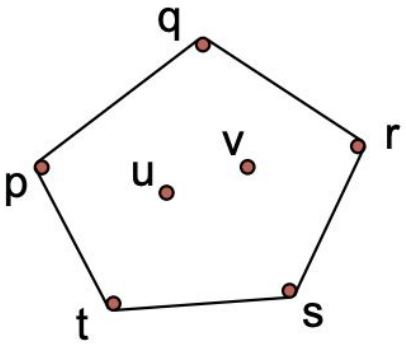
assume no two have same x coordinate, no two have same y coordinate, and no three in a line for convenience.

Convex Hull (CH(S)): smallest polygon containing all points in S. **Goal!**

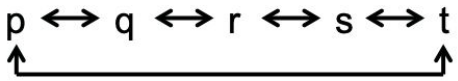
No!



Smallest Convex Polygon



CH(S): the sequence of points on the boundary in order clockwise as doubly linked list



1. Brute force for Convex Hull

Test each line segment to see if it makes up an edge of the convex hull

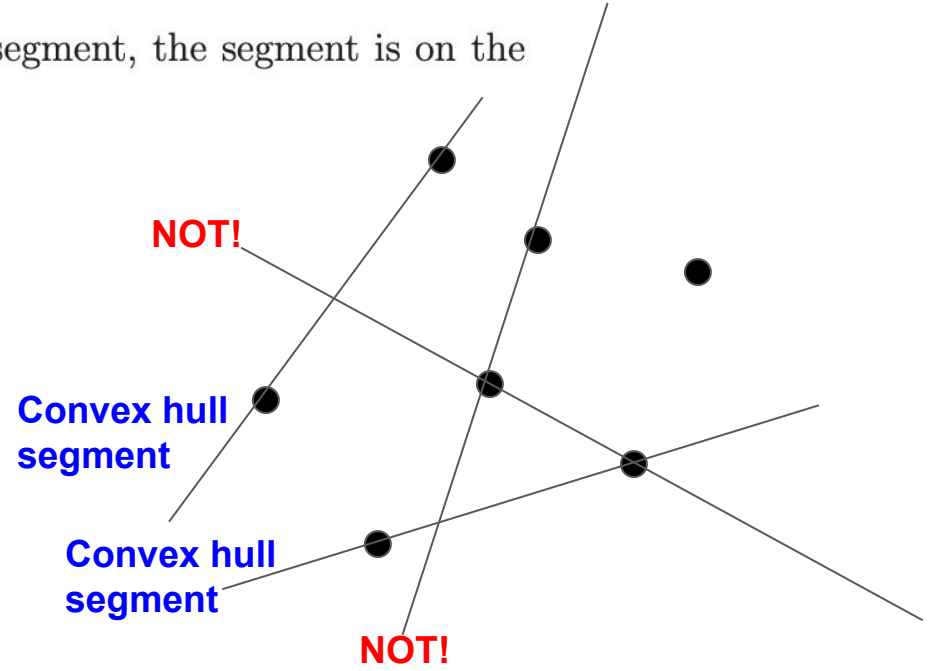
- If the rest of the points are on one side of the segment, the segment is on the convex hull.
- else the segment is not.

$O(n^2)$ edges, $O(n)$ tests $\Rightarrow O(n^3)$ complexity

Can we do better?

n points

O(n^2): n^2 possible # of edges
O(n): test all n points



2. Divide and Conquer Convex Hull

Divide and Conquer Convex Hull

Sort points by x coord (once and for all, $O(n \log n)$)

For input set S of points:

- Divide into left half A and right half B by x coords **By drawing a line**
- Compute $CH(A)$ and $CH(B)$
- Combine CH's of two halves (merge step)

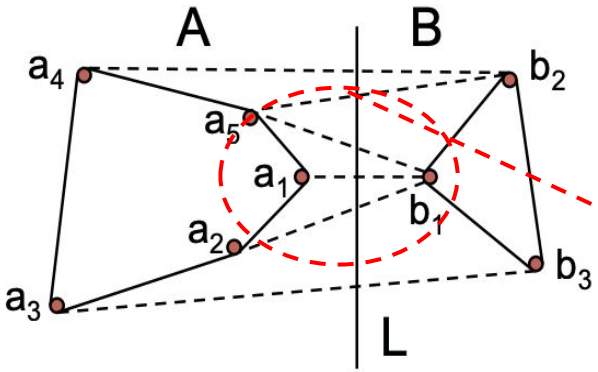
Recursively divide.
Apply "brote force" on each segment (tangent) in O(1) time

How to Merge? By merging upper segment and lower segment

How to Merge?

(a4, b2) upper tangent

(a3, b3) lower tangent



Why **NOT** a convex hull segment?
Both sides have some points when divided by line

y(i,j): y coords of segment at ai and bj

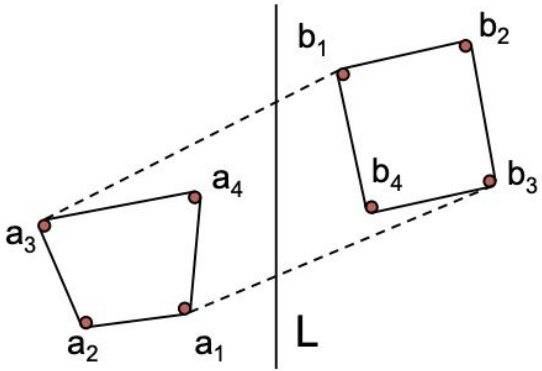
- Find upper tangent (a_i, b_j). In example, (a_4, b_2) is U.T. **Highest y(i,j)**
- Find lower tangent (a_k, b_m). In example, (a_3, b_3) is L.T. **Lowest y(i,j)**

How to Merge? By merging upper segment and lower segment

- Cut and paste in time $\Theta(n)$.

Check every nodes

First link a_i to b_j, go down b ilst till you see b_m and link b_m to a_k, continue along the a list until you return to a_i. In the example, this gives (a_4, b_2, b_3, a_3).



(a1,a2,a3,a4)
(b1,b2,b3,b4)
Upper in y(i, j): (a3, b1)
Lower in j(i, j): (a1, b3)

Convex Hull: a3 - b1 - b2 - b3 - a1 - a2 - a3

a_3, b_1 is upper tangent. $a_4 > a_3$, $b_2 > b_1$ in terms of Y coordinates.
a_1, b_3 is lower tangent, $a_2 < a_1$, $b_4 < b_3$ in terms of Y coordinates.

a_i, b_j is an upper tangent. Similarly, for lower tangent.

Does not mean that a_i or b_j is the highest point.

How to Find Upper/Lower Segment (Tangent) in O(n^2)

Finding Tangents

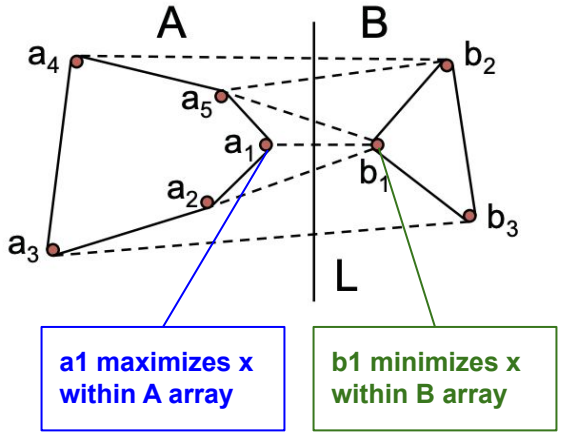
Assume a_i maximizes x within $CH(A)$ $(a_1, a_2, \ldots, a_p)$. b_1 minimizes x within $CH(B)$ $(b_1, b_2, \ldots, b_q)$

L is the vertical line separating A and B. Define $y(i, j)$ as y-coordinate of intersection between L and segment (a_i, b_j).

Claim: (a_i, b_j) is uppertangent iff it maximizes $y(i, j)$.

If $y(i, j)$ is not maximum, there will be points on both sides of (a_i, b_j) and it cannot be a tangent.

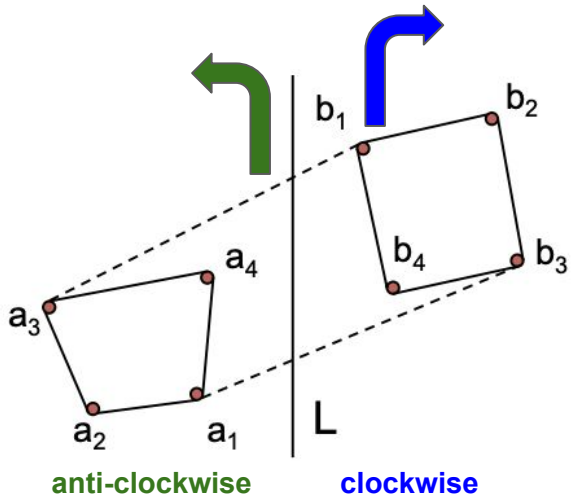
Algorithm: Obvious $O(n^2)$ algorithm looks at all a_i, b_j pairs. $T(n) = 2T(n/2) + \Theta(n^2) = \Theta(n^2)$.



How to Find Upper/Lower Segment (Tangent) in O(n)

By two fingers algorithm for merging 2 convex hulls

```
1  i = 1
2  j = 1                                   2 operations per loop in O(1)
3  while (y(i, j+1) > y(i, j) or y(i−1, j) > y(i, j))
4        if (y(i, j+1) > y(i, j)) ▷ move right finger clockwise
5              j = j + 1( mod q)                    p + q = n
6        else
7              i = i − 1( mod p) ▷ move left finger anti-clockwise
8        return (a_i, b_j) as upper tangent
```

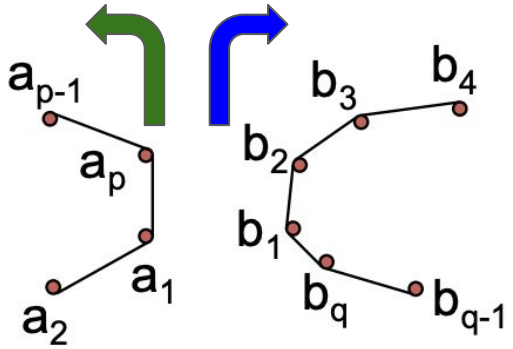


Similarly for lower tangent.

$$T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n)$$

Find upper/lower segment + cut & paste in O(n)

Intuition for Why Merge Works



a_1, b_1 are right most and left most points. We move anti clockwise from a_1, clockwise from b_1. $a_1, a_2, \ldots, a_q$ is a convex hull, as is $b_1, b_2, \ldots, b_q$. If a_i, b_j is such that moving from either a_i or b_j decreases $y(i, j)$ there are no points above the (a_i, b_j) line.

Median Finding

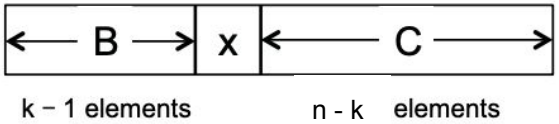
Given set of n numbers, define $rank(x)$ as number of numbers in the set that are $\leq x$.
Find element of rank $\lfloor \frac{n+1}{2} \rfloor$ (lower median) and $\lceil \frac{n+1}{2} \rceil$ (upper median).

Clearly, sorting works in time $\Theta(n \log n)$.

Can we do better?

We can do in O(n) linear

i = Rank.
rank(n/2)
= median




```
SELECT(S, i)
1  Pick x ∈ S ▷ cleverly
2  Compute k = rank(x)
3  B = {y ∈ S | y < x}
4  C = {y ∈ S | y > x}
5  if k = i
6        return x
7  else if k > i
8        return Select(B, i)
9  else if k < i
10       return Select(C, i − k)
```

subproblems

i < k: i exists in left of k: B

i > k: i exists in right of k: C

What if **arbitrary selection**?

Worst case analysis:
If we choose x at the end of the sequence, two sub-sequences are extremely balanced.
If left sub-sequence has n-1 elements, n-1 recursion = **O(n^2)**

Randomized vs Deterministic Algorithms

Randomized

Problem - given a probability distribution,

It'll be an expected time.

Deterministic

Guaranteed to run in worse case time

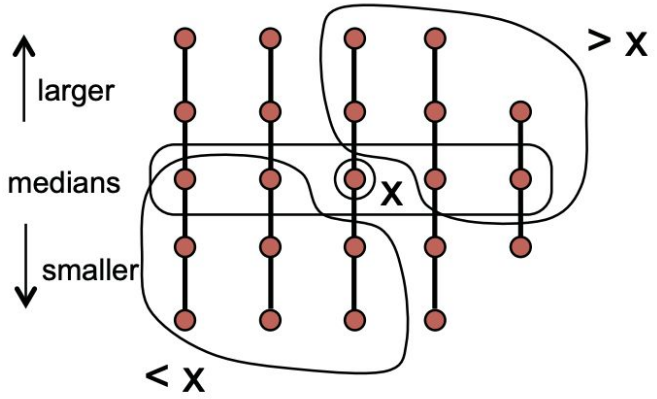
-> **deterministic way of picking x in S**

-> solve it by fairly balanced partitions

-> shrinking down geometrically

We do this here!

Picking x Cleverly



Need to pick x so $rank(x)$ is not extreme.

- Arrange S into columns of size 5 ($\lceil \frac{n}{5} \rceil$ cols)
- Sort each column (bigger elements on top) (linear time)
- Find "median of medians" as x

How many elements are guaranteed to be $> x$?

Half of the $\lceil \frac{n}{5} \rceil$ groups contribute at least 3 elements $> x$ except for 1 group with less than 5 elements and 1 group that contains x.

At lease $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $> x$, and at least $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $< x$

Recurrence:

$$T(n) = \begin{cases} O(1), & \text{for } n \leq 140 \\ T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6), \Theta(n), & \text{for } n > 140 \end{cases} \quad (1)$$

How to Pick x Cleverly

25 24 33 39 3 18 19 31 23 49 45 16 1 29 40 22 15 20 24 4 13 34

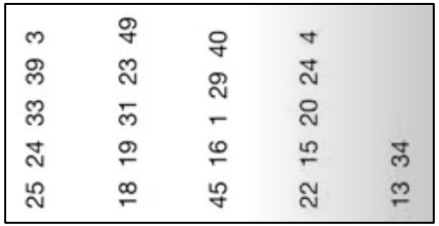
Divide array into chunks

Calculate Median directly for each chunk

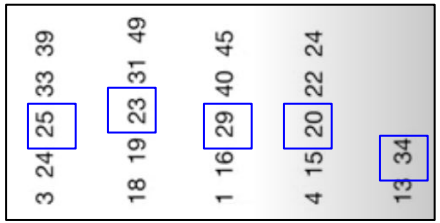
Recursively calculate median of set of medians from previous step

Use final Median of medians as the pivot element

5 rows
n/5 cols

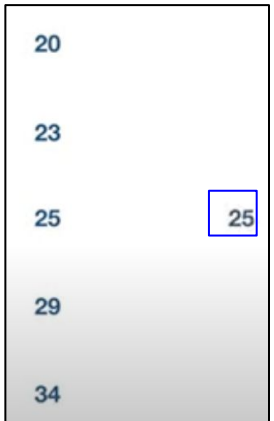


Sort in each col



25 23 29 20 34

medians



Median of medians

How to Pick x Cleverly (con't)

How many elements are guaranteed to be $> x$?

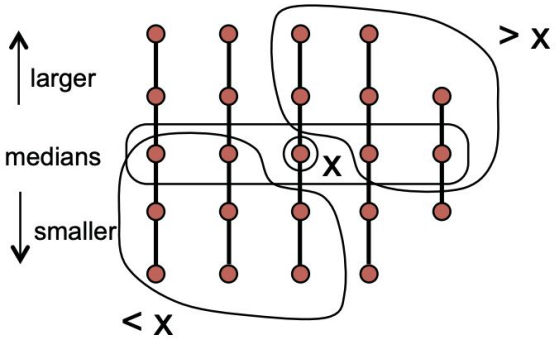
Half of the $\lceil \frac{n}{5} \rceil$ groups contribute at least 3 elements $> x$ except for 1 group with less than 5 elements and 1 group that contains x.

At lease $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $> x$, and at least $3(\lceil \frac{n}{10} \rceil - 2)$ elements are $< x$

Recurrence:

$$T(n) = \begin{cases} O(1), & \text{for } n \leq 140 \\ T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) + \Theta(n), & \text{for } n > 140 \end{cases} \quad (1)$$

T(n/5): time for divide by 5 rows
T(7n/10+6): time for finding x as median of medians
O(n): sorting all cols (each col in O(1))



How many elements we need to consider?

Here 5 cols - 1 col with less than 5 - 1 col with x = 3 cols

Half of n/5 groups = n/10 groups
So at least 3(n/10 - 2) = 3n/10 - 6 elements we can ignore.
So we can consider only **7n/10 + 6 elements** (max size partition that we recurse)

Solving the Recurrence

Master theorem does not apply. Intuition $\frac{n}{5} + \frac{7n}{10} < n$.

Prove $T(n) \leq cn$ by induction, for some large enough c.

True for $n \leq 140$ by choosing large c

$$T(n) \leq c\lceil\frac{n}{5}\rceil + c(\frac{7n}{10} + 6) + an \quad (2)$$

$$\leq \frac{cn}{5} + c + \frac{7nc}{10} + 6c + an \quad (3)$$

$$= cn + (-\frac{cn}{10} + 7c + an) \quad (4)$$

Linear time

If $c \geq \frac{70c}{n} + 10a$, we are done. This is true for $n \geq 140$ and $c \geq 20a$.

Linear Time Median Finding by D&C

Python code: https://rcoh.me/posts/linear-time-median-finding/

Find x as pivot for Median Finding:
https://www.youtube.com/watch?v=sNtu2oGDRvU

Median Finding Algorithm:
https://www.cs.cornell.edu/courses/cs2110/2009su/Lectures/examples/MedianFinding.pdf

END.