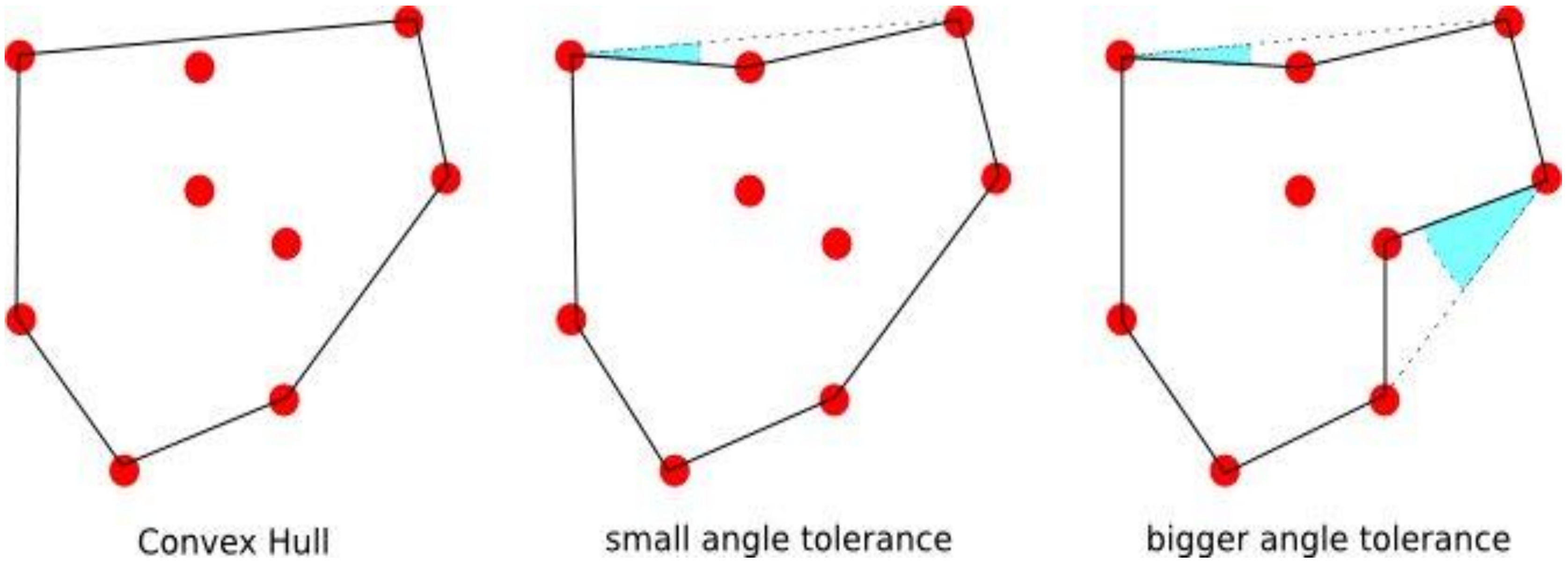
Divide and Conquer Algorithms

DIVIDE AND CONQUER THE PARADIGM

Given a problem of size n :

- Decompose it into smaller (similar) problems
- Solve the smaller (easier) problems
- Combine the partial solution into the overall solution to the initial problem

Convex Hull Problem: Given a set of n point in the plane , find the smallest convex polygon that encloses all the points



Convex Hull Problem: Given a set of n point in the plane , find the smallest convex polygon that encloses all the points

Brute force: Algorithm

Given two points, if all other points are on the same side of the segment joining those points, then that segment is a vertex of the convex hull

slido

What is the time complexity of the brute force algorithm to find the convex hull of n points?

ⓘ Start presenting to display the poll results on this slide.

Convex Hull Problem: Given a set of n point in the plane , find the smallest convex polygon that encloses all the points

- Sort the points on the x coordinates (Done once)
- Divide the Points in A on the left and B on the right based on the x coordinates
- (Recursively) Find the CH(A) and CH(B)
- Merge the two convex hulls

HOW TO MERGE TWO CONVEX HULLS ?

CHECK Every pair of points (a_i , b_j) to determine if the corresponding segment is part of the overall convex hull

Convex Hull Problem: Given a set of n point in the plane , find the smallest convex polygon that encloses all the points

- Sort the points on the x coordinates (Done once)
- Divide the Points in A on the left and B on the right based on the x coordinates
- (Recursively) Find the CH(A) and CH(B)
- Merge the two convex hulls

$n \log n$

O(1)

T(n/2) + T(n/2)

O(n)

slido

What is the time complexity of the brute merge ?

ⓘ Start presenting to display the poll results on this slide.

Two finger Algorithm for merging convex hulls

i=1
j= 1
While($y(i, j+1) > y(i, j)\ or\ \ y(i+1, j) > y(i, j)$):
    if($y(i, j+1) > y(i, j)$)
        j= j+1 mod q   //move the right finger clockwise
    else if ($y(i+1, j) > y(i, j)$)
        i=i+1 mod p   //move the left finger
counterclockwise
Return $(a_i, b_j)\ \ as\ the\ upper\ tangent$

Median finding: Given a set of n numbers find the median

Given a set of n numbers, the rank of a number x in the number of elements ≤ x

Lower median has rank = floor($\frac{n+1}{2}$)

Upper Median has rank=Ceil($\frac{n+1}{2}$)

Select(S,i) is a function that returns the element of rank i in S

```
Select (S,i) {
   Pick element x in S
   A={y in S such that y < x }
   B={y in S such that y >x}
   if ( k==i) return x
   else if (k >i ) return Select( A,i)
  else return Select(B,i-k)
}
```

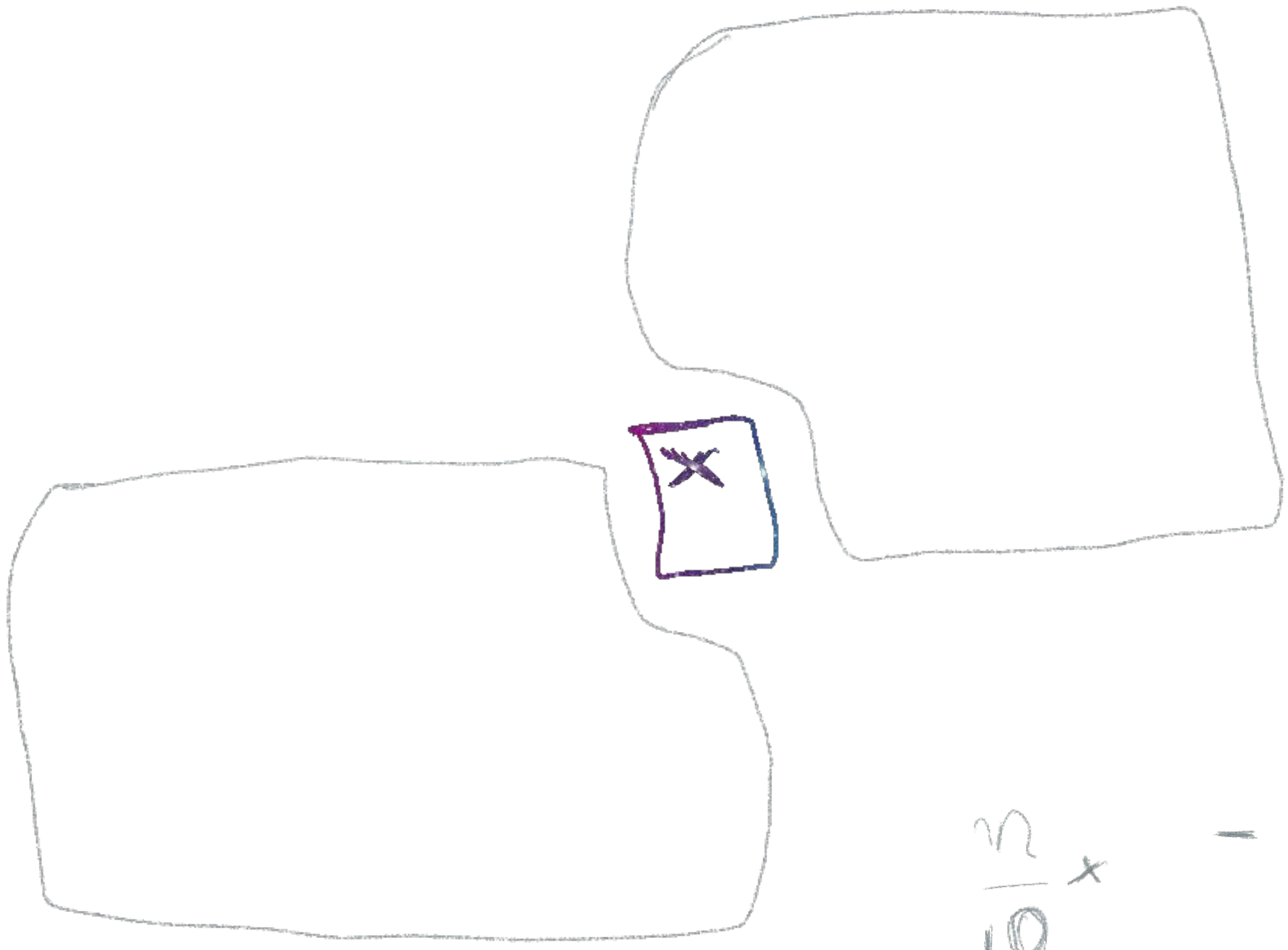
slido

What is the worst case time complexity of the Select function?

ⓘ Start presenting to display the poll results on this slide.

How to pick x in a clever, deterministic way ?

Divide the Set into columns of size 5=>
Ceil(n/5) columns
Sort each column with biggest element at the top
Return the median of the medians of the columns



$\frac{n}{10} \times \quad -2$

The recurrence relation

$$T(n) = \begin{cases} O(1) \ \ if \ \ n \leq 140 \\ T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + \theta(n) \end{cases}$$

Inversion Counting Algorithms

Given an Array A , we have an inversion every time A[i] >A[j] and i<j

Given an Array A of size n, what is the maximum possible number of inversions?

Given an Array A of size n, How can we effiently count the number of inversions?

Inversion Counting Algorithms

- Divide the Array in two subarrays (equal)
- Recursively count the number of inversions in the left subarray
- Recursively count the number of inversions in the right subarray
- Count the number of inversions across the subarray

Inversion Counting Algorithms

$$T(n) = 2\,T\left(\frac{n}{2}\right) + O(n) \implies T(n) = n \log n$$

Average Case of Quick Sort

Worst Case

$$T(n) = T(n-1) + O(n)$$
$$\Rightarrow T(n) = O(n^2)$$

Best Case

$$T(n) = 2\,T\left(\frac{n}{2}\right) = O(n)$$
$$\Rightarrow T(n) = O(n \log n)$$

Average Case of Quick Sort

- Depends on the possible choices of the pivot
- For Simplicity, Assume that elements are unique all have the same chance of being selected as the pivot => Randomized choice of the pivot.
- What are the possible partitions => 0: n-1,1: n-2,...,n-1,0
- The expected time complexity is the average time complexity :

$$T(n) = 1/n(\sum_{k}^{n-1}(T(k) + T(n-k-1)) + O(n))$$

Closest pair of points