

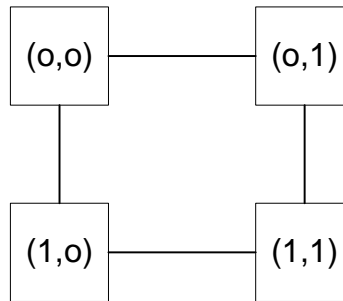
Lecture #6:

0.0.1 Divide & Conquer Method (Continued – More Examples):

We conclude this section with a few more examples.

Example 1 *Hamiltonian Circuit in a Hypercube:*

Consider the following graph: Its vertices correspond to the set of 2^n vectors of dimension n all of whose components are 0 or 1. For example, for $n = 2$, these are $\{(0, 0), (0, 1), (1, 0), (1, 1)\}$. Two vertices are connected if their vectors differ by a unit vector. For $n = 2$, the graph looks like:



Definition 2 A *Hamiltonian Circuit* (also known as a *Traveling Salesperson Tour*) is circuit that passes through every node exactly once.

We want to produce a Hamiltonian circuit for a hypercube for all n . For $n = 2$, one order in which the nodes are visited is

(0, 0)
(0, 1)
(1, 1)
(1, 0)

From this we can create the solution for $n = 3$ by the following process:

(0, 0, 0)
(0, 1, 0)
(1, 1, 0)
(1, 0, 0)
(1, 0, 1)
(1, 1, 1)
(0, 1, 1)
(0, 0, 1)

The first four are obtained by simply appending a zero the four in $n = 2$ diagram; the last four are obtained by having a 1 in the last position and writing the solution for $n = 2$ in the reverse order!

Example 3 *Scheduling Tournaments among $n = 2^k$ teams to complete the schedule in $(n - 1)$ days. (Courtesy: Professor Sudborough).*

1	2
2	1

1	2		3	4
2	1		4	3
3	4		1	2
4	3		2	1

1	2		3	4		5	6		7	8
2	1		4	3		6	5		8	7
3	4		1	2		7	8		5	6
4	3		2	1		8	7		6	5
5	6		7	8		1	2		3	4
6	5		8	7		2	1		4	3
7	8		5	6		3	4		1	2
8	7		6	5		4	3		2	1

Example 4 (Chapter 10.3) *Selecting the k^{th} smallest element in an unsorted array $A[i, \dots, j]$:*

```

SELECT( $A, i, j, k$ )
  if  $j - i < 80$ , then sort  $A[i, \dots, j]$  and choose the  $k^{\text{th}}$  smallest element.
  else
    begin
      break up  $A[i, \dots, j]$  into sets of 5 elements;
      choose the median of each set;
      form the array of medians  $B[1, \dots, t]$ , for appropriate value of  $t$ ;
       $p \leftarrow$  SELECT( $B, 1, t, \lceil \frac{t}{2} \rceil$ ) /*  $p$  is the median-of-medians */
      PARTITION ( $A[i, \dots, j], p$ ) /* partition the array  $A[i, \dots, j]$  using the
pivot  $p$  */
      Let the LOW side have  $m$  elements and the high side have all the
remaining elements;
      /* LOW side is all elements in  $A[i, \dots, j] < p$  and HIGH side is all elements
in  $A[i, \dots, j] \geq p$ . */
      if  $k \leq m$       then SELECT( $A, i, i + m - 1, k$ )
                        else SELECT( $A, i + m, j, k - m$ )
    end

```

```

PARTITION( $A[i, \dots, j], x$ )
   $u \leftarrow i - 1$ 
   $v \leftarrow j + 1$ 
  while TRUE
    do repeat  $v \leftarrow v - 1$ 
      until  $A[v] \leq x$ 
    repeat  $u \leftarrow u + 1$ 
      until  $A[u] \geq x$ 
    if  $u < v$ 
      then exchange  $A[u] \leftrightarrow A[v]$ 
    else return  $v$ 

```

The LOW side is $A[i, \dots, q]$ and the HIGH side is $A[q + 1, \dots, j]$ where q is value of v returned at the end of the algorithm. See page 154-155 of your book. The run time of this algorithm is $\Theta(|(j - i)|)$.

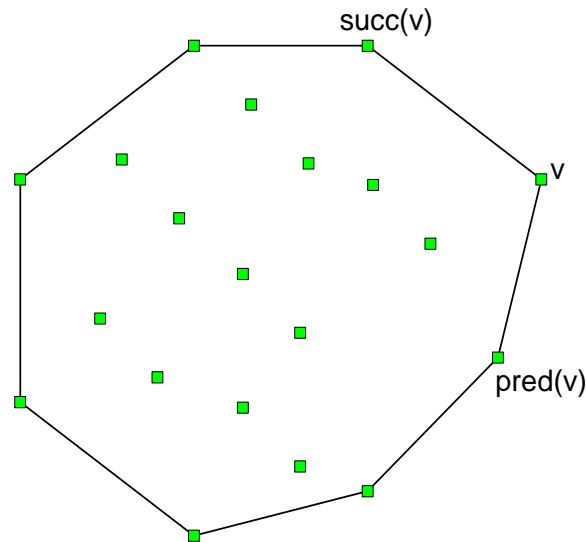
For each set with a median smaller than $p = \text{median-of-medians}$, there are at least three elements smaller than p . So we obtain a lower bound for the number of elements in the LOW side when we partition using p as the pivot as follows; $|LOW| = 3(\lceil \frac{1}{2} \rceil \lceil \frac{n}{5} \rceil - 2) \geq \frac{3}{10}n - 6$. Similar logic shows that the number of elements in the HIGH side is also at least $\frac{3}{10}n - 6$. Hence each side has no more than $(n - \frac{3}{10}n + 6) = \frac{7}{10}n + 6$ elements. So we obtain the recurrence:

$$t(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 80 \\ t(\lceil \frac{n}{5} \rceil) + t(\frac{7}{10}n + 6) + cn & \text{if } n > 80 \end{cases}$$

The first term is time needed for **SELECT** to find the median-of-medians; the second is an upper bound for **SELECT** to find the k^{th} rank element on the LOW or HIGH side as needed; the third is time for breaking up the array, partitioning, getting array of medians etc. One can show by substitution, or master theorem that $t(n) \leq dn$, for some constant $d > 0$, and all n .

Instead of the number 5 in the above discussion, we may choose any odd number greater than or equal to 5 – for example 7 will also do.

Example 5 *Convex hulls in 2 – D*



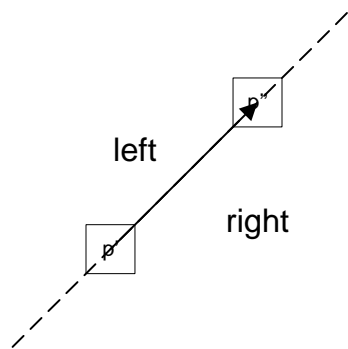
Given a set of points in the plane, the convex hull is the smallest convex polygon that contains all of them. We denote the convex hull of a set Q of points by $CH(Q)$. We want an algorithm which computes $CH(Q)$ given Q . The out put of the algorithm is the set of vertices of the polygon in counterclockwise order. There are several algorithms for this and our purpose here is to show the use of divide-and-conquer method. Even with this restriction, there are many algorithms.

We illustrate one of them below. For a vertex v of the convex polygon P ,

$succ(v)$ [$pred(v)$] is the point that follows [comes before] v in counterclockwise order. These are shown in the above diagram.

The **basic operation** that is allowed here is the following: Given three points p' , p'' , and p , we want to check whether p lies to the *left* (=above), *right*(=below), or *on* the *directed line segment* from p' to p'' . See diagram

below:



If we let $p = (x, y)$; $p' = (x', y')$; and $p'' = (x'', y'')$ then $left \iff \Delta > 0$; $right \iff \Delta < 0$; $on \iff \Delta = 0$ where

$$\Delta = \det \begin{vmatrix} x & y & 1 \\ x' & y' & 1 \\ x'' & y'' & 1 \end{vmatrix}$$

Thus, this takes a constant amount of time.

Algorithm:

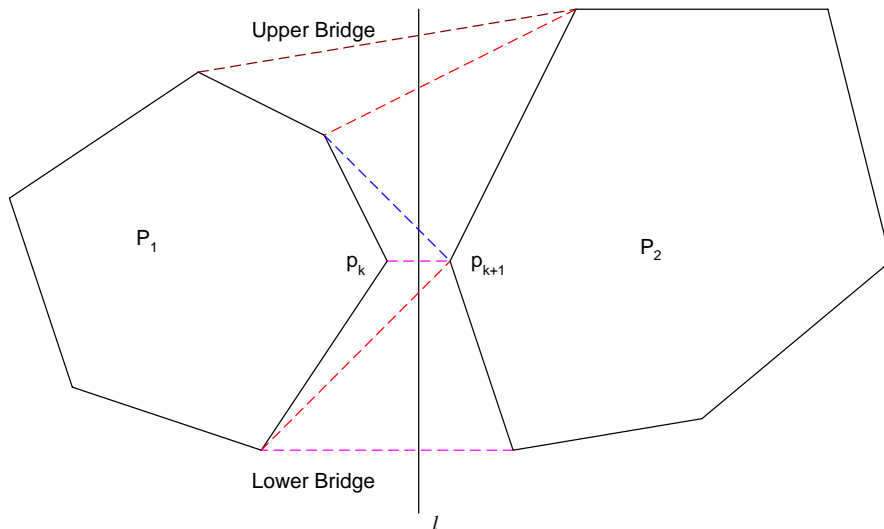
Pre-sort the points in increasing order of the x -coordinate so that $x_1 \leq x_2 \leq \dots \leq x_n$.

If $n \leq 3$, one basic operation will find the convex hull.

Else, define sets $S_1 = \{p_1, p_2, \dots, p_{\lfloor \frac{n}{2} \rfloor}\}$ and $S_2 = \{p_{\lfloor \frac{n}{2} \rfloor + 1}, \dots, p_n\}$

Find $CH(S_1)$ and $CH(S_2)$ recursively.

Using this, we merge to find $CH(S)$ as follows:



We find the upper and lower bridges as follows: Let p_k be the right most point in $P_1 = CH(S_1)$ and p_{k+1} be the left most point in $P_2 = CH(S_2)$. Let $s = [\overrightarrow{p_k, p_{k+1}}]$ be the directed line segment from p_k to p_{k+1} .

Finding Upper bridge:

```

 $v \leftarrow p_k; w \leftarrow p_{k+1}; h = [\overrightarrow{v, w}]$ 
while one of the points  $succ(v)$  and  $pred(w)$  lies above  $h$ 
    do if  $succ(v)$  lies above  $h$ , then  $v \leftarrow succ(v)$ 
    else  $w \leftarrow pred(w)$ 
endif
endwhile

```

What happens in the above discussion if p_k and p_{k+1} coincide?

Example 6 *Closest pair of points : 2 - D*

Given n points $p_i; i = 1, 2, \dots, n$ in two dimension, (i.e. $p_i = (x_i, y_i)$), we want the pair (k, l) such that $d(p_k, p_l)$ is minimum where distance is the usual euclidean distance. A straight forward algorithm takes $\Theta(n^2)$ computations. We want an algorithm that takes less work. We describe below an algorithm that takes $O(n \lg n)$ time.

Some notation first: P denotes the set of points. X denotes a permutation of $\{1, 2, \dots, n\}$ that is obtained by sorting the points according to their x -coordinates in increasing order. Y denotes a permutation of $\{1, 2, \dots, n\}$ that is obtained by sorting the points according to their y -coordinates in increasing order. These sorts are done once in the beginning of the algorithm and are often

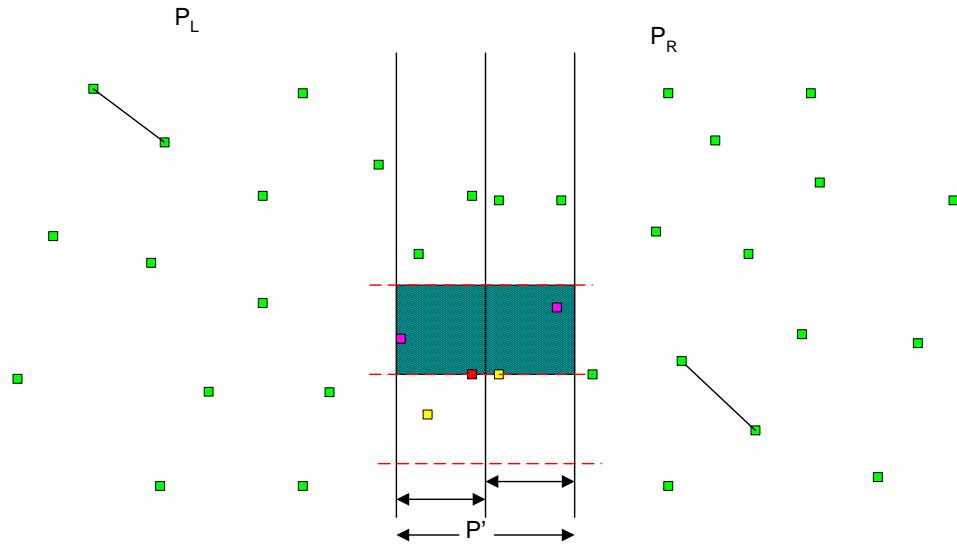
referred to as pre-sorts. The time taken for this is known to be $\Theta(n \lg n)$. From now on we think of the input to the algorithm as the combination $[P, X, Y]$. So, when we divide the problem, we must give the sub-problems in the same manner. We will also assume without loss of generality that the points are numbered as per X .

DIVIDE: We divide the problem into roughly two equal parts by taking the points in the two halves as per the sorting X . So we can imagine a vertical line l separating the two halves: P_L is the set of points on the left of the line l and P_R is the set of points on the right of the line l . It is conceivable that in degenerate cases there may be points on the line l which are distributed among the two sides. The x -coordinate corresponding to l is the median of the x -coordinates of all points in P . By using the presort X , we can get X_L and X_R which correspond to the sorting of the points P_L and P_R respectively **without sorting again**. So also we can get Y_L and Y_R that do sorting of these subsets by their y -coordinate. All this takes $\Theta(n)$ time. We now solve the sub-problems: $[P_L, X_L, Y_L]$ and $[P_R, X_R, Y_R]$ recursively. This gives the pair of points in P_L that have minimum distance δ_L and a similar pair for P_R with distance δ_R . We now let $\delta = \min[\delta_L, \delta_R]$.

COMBINE: Now we need to combine all this to get the results for P . We need to check if a pair of points of which one is in P_L and the other is in P_R might have a distance lower than δ and if so find the smallest of these. And we need to do this in time equal to $\Theta(n)$ since, otherwise the total time would exceed the bound we want. First some observations:

- If there is a pair of points of which one is in P_L and the other is in P_R with a distance lower than δ , then the x -coordinate of the point in P_L must be greater than $x_l - \delta$ and that of the point in P_R must be less than $x_l + \delta$. Thus, such pairs of points must be in a strip of width 2δ centered at l . Since points are numbered as per X , it is easy to locate all points in this strip in time equal to $\Theta(\lg n)$. Let P' be the subset of P that is in this strip. Further, let X' and Y' be the corresponding ordering according to x and y coordinates. These can be obtained as subsets of X and Y in time equal to $\Theta(n)$. Now we need to check if for pairs of points in P' distance is less than δ and if so find the minimum of these. A brute force approach at this point could take $\Theta(n^2)$ and all will be lost. We have to

be clever about this.



- For this we note that for any point in P' , any other point p that is within a distance of δ , must be inside not only the vertical strip of width 2δ centered at l , but also within a horizontal strip of width 2δ centered at p . Moreover, no two points in this section from P_L can be closer than δ and the same applies to two points in P_R . There can be at most eight such points in a rectangle of size $\delta \times 2\delta$. This restricts our search to at most 7 points for each point p in P' . Thus, this process takes $\Theta(n)$ time.

Thus, the recurrence relation for the overall problem is

$$t(n) = 2t\left(\frac{n}{2}\right) + \Theta(n)$$

and the solution to this is $t(n) = \Theta(n \lg n)$.