

Chapter 4: Divide and Conquer

Chin Lung Lu

Department of Computer Science

National Tsing Hua University

Basic idea of divide and conquer strategy:

1. If the problem size is small enough, then solve the problem by some straightforward method; otherwise, divide a problem into two (or more) smaller sub-problems, preferably in equal size.
(Note that each sub-problem is identical to its original problem, except its input size is smaller.)
2. Solve these two sub-problems by using the divide and conquer strategy again (i.e., solving them recursively).
3. Merge two sub-solutions into the final solution.

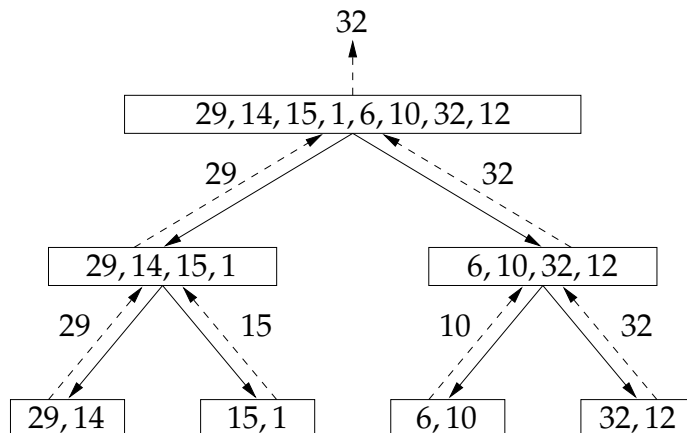
1

2

Divide and conquer strategy (cont'd)

Example (finding maximum problem):

Find the maximum among $S = \{29, 14, 15, 1, 6, 10, 32, 12\}$.



3

Time complexity of divide and conquer

► Let $T(n)$ be the time complexity of a problem with input size n .

Recursive formula of $T(n)$:

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + S(n) + M(n) & \text{if } n \geq c \\ b & \text{if } n < c \end{cases}$$

- $S(n)$: Time to divide the problem into two sub-problems
- $M(n)$: Time to merge two sub-solutions into the final solution
- b (a constant): Time to straightforwardly solve the sub-problem of size small enough

4

Time complexity of divide and conquer (cont'd)

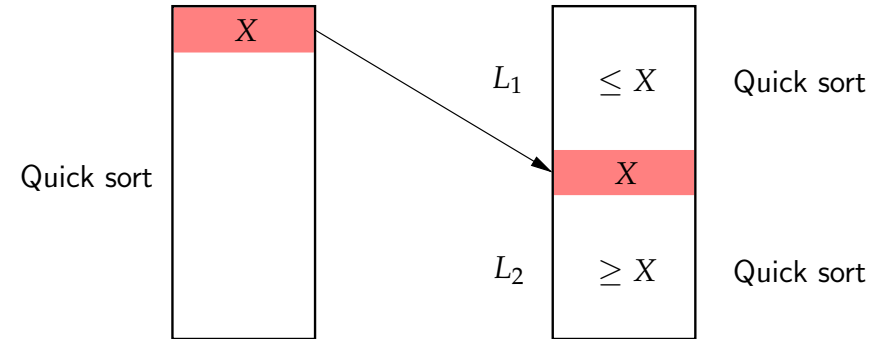
Example (finding maximum problem): let $n = 2^k$

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \mathcal{O}(1) & \text{if } n > 2 \\ 1 & \text{if } n = 2 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 1 \\ &= 2(2T(\frac{n}{4}) + 1) + 1 = 2^2T(\frac{n}{2^2}) + 2^1 + 2^0 \\ &\vdots \\ &= 2^{k-1}T(2) + 2^{k-2} + \dots + 2^0 \\ &= 2^{k-1} + 2^{k-2} + \dots + 2^0 \\ &= 2^k - 1 \\ &= n - 1 \\ &= \mathcal{O}(n) \end{aligned}$$

Quick sort

- Given a set of n numbers a_1, a_2, \dots, a_n , we choose a number X to divide a_1, a_2, \dots, a_n into two lists as shown below.



5

6

Quick sort algorithm

Algorithm: *Quicksort*(f, l)

Input: A sequence of $(l - f + 1)$ numbers a_f, a_{f+1}, \dots, a_l .

Output: Sorted sequence of a_f, a_{f+1}, \dots, a_l in non-decreasing order.

1. **if** $f \geq l$, **then** return
2. $X = a_f, i = f, j = l$
3. **while** $i < j$ **do**
4. **while** $a_j \geq X$ and $i < j$ **do**
5. $j = j - 1$
6. $a_i \leftrightarrow a_j$
7. **while** $a_i \leq X$ and $i < j$ **do**
8. $i = i + 1$
9. $a_i \leftrightarrow a_j$
10. **end while**
11. *Quicksort*($f, j - 1$), *Quicksort*($j + 1, l$)

7

Example of quick sort

Iteration 1: Let

$a_1 = 3, a_2 = 6, a_3 = 1, a_4 = 4, a_5 = 5, a_6 = 2$.

$X = 3$	a_1	a_2	a_3	a_4	a_5	a_6
$i = 1, j = 6$	3	6	1	4	5	2
$(a_j = a_6 < X)$	$\uparrow i$					$\uparrow j$
$a_1 \leftrightarrow a_6$	2	6	1	4	5	3
$(a_i = a_1 < X)$	$\uparrow i$					$\uparrow j$
$i = i + 1 = 2$	2	6	1	4	5	3
$(a_i = a_2 > X)$		$\uparrow i$				$\uparrow j$
$a_2 \leftrightarrow a_6$	2	3	1	4	5	6
$(a_j = a_6 > X)$		$\uparrow i$				$\uparrow j$
$j = j - 1 = 5$	2	3	1	4	5	6
$(a_j = a_5 > X)$		$\uparrow i$			$\uparrow j$	

8

Example of quick sort (cont'd)

Iteration 1 (cont'd):

$X = 3$	a_1	a_2	a_3	a_4	a_5	a_6
$j = j - 1 = 4$ ($a_j = a_4 > X$)	2	3	1	4	5	6
		$\uparrow i$		$\uparrow j$		
$j = j - 1 = 3$ ($a_j = a_3 < X$)	2	3	1	4	5	6
		$\uparrow i$	$\uparrow j$			
$a_2 \leftrightarrow a_3$ ($a_i = a_2 < X$)	2	1	3	4	5	6
		$\uparrow i$	$\uparrow j$			
$i = i + 1 = 3$ ($i = j = 3$) (end of iteration 1)	2	1	3	4	5	6
			$i \uparrow j$			
	≤ 3	≤ 3	$= 3$	≥ 3	≥ 3	≥ 3

Time complexity of quick sort

- ▶ Let $T(n)$ be the time complexity of quick sort with input size n .

Best case:

- ▶ The best case occurs when we can split the problem into two equal-size subproblems for each round.
- ▶ It means that $T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n)$.
- ▶ In this case, we have $T(n) = \mathcal{O}(n \log n)$.

Worst case:

- ▶ The worst case occurs when the input data is already a sorted or reversely sorted sequence.
- ▶ In this case, we have $T(n) = T(n - 1) + \mathcal{O}(n) = \mathcal{O}(n^2)$.

9

10

Time complexity of quick sort (cont'd)

Given $T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n)$, let $n = 2^k$ (i.e., $k = \log_2 n$).

$$\begin{aligned}
 T(n) &\leq 2T(\frac{n}{2}) + cn \\
 &\leq 2(2T(\frac{n}{4}) + c\frac{n}{2}) + cn \\
 &= 4T(\frac{n}{4}) + 2c\frac{n}{2} + cn \\
 &\vdots \\
 &= 2^k T(\frac{n}{2^k}) + 2^{k-1} c \frac{n}{2^{k-1}} + 2^{k-2} c \frac{n}{2^{k-2}} + \dots + 2^0 c \frac{n}{2^0} \\
 &= nT(1) + cn + cn + \dots + cn \\
 &= nT(1) + cnk \\
 &= n + cn \log_2 n
 \end{aligned}$$

Therefore, $T(n) = \mathcal{O}(n \log n)$.

Merge sort algorithm

Algorithm: *MergeSort*(C)

Input: A list C of n elements

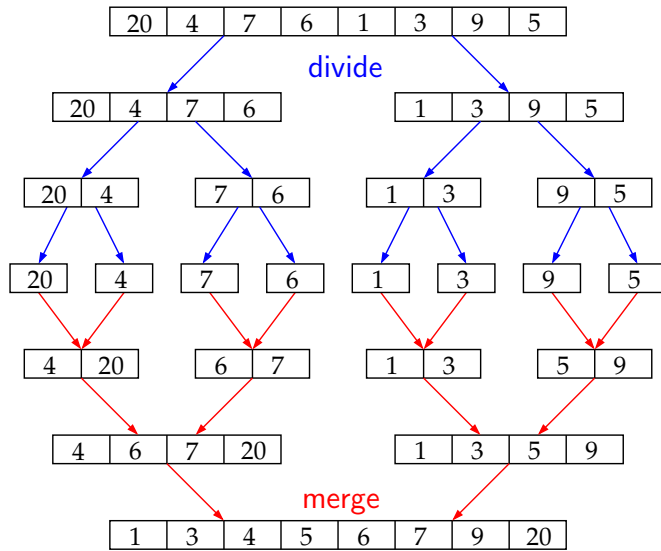
Output: A sorted list of these n elements in non-decreasing order

1. **if** $n = 1$ **then** return C.
 2. Let C_L be the list of first $\frac{n}{2}$ elements of c .
 3. Let C_R be the list of last $n - \frac{n}{2}$ elements of c .
 4. $S_L = \text{MergeSort}(C_L)$.
 5. $S_R = \text{MergeSort}(C_R)$.
 6. $\text{Merge}(S_L, S_R)$.
-

11

12

Example of merge sort



13

Time complexity of merge sort

- Note that the cost of merging two sorted sequences is $\mathcal{O}(n)$.

Time complexity of the merge sort $T(n)$:

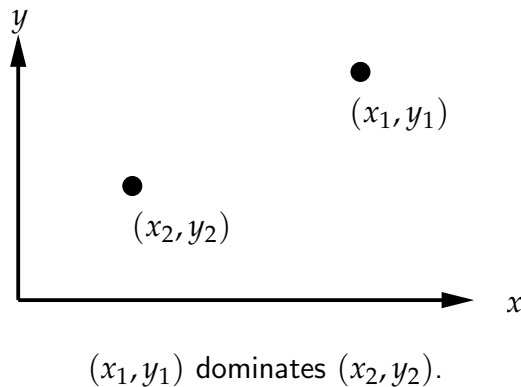
$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + \mathcal{O}(n) & \text{if } n > 2 \\ 1 & \text{if } n = 2 \end{cases}$$

- Therefore, we have $T(n) = \mathcal{O}(n \log n)$.

14

Point domination

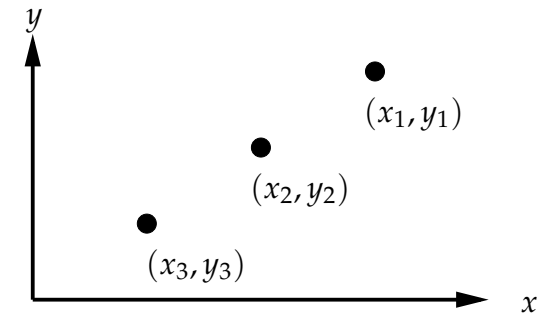
- In the 2D space, a point (x_1, y_1) dominates a point (x_2, y_2) if $x_1 > x_2$ and $y_1 > y_2$.



15

Maximal point

- A point is called a maximal point if no other point dominates it.

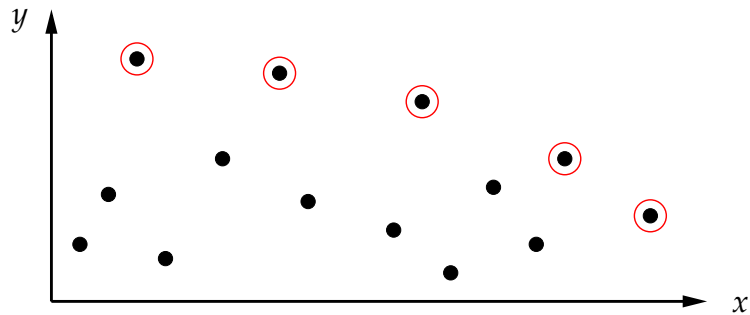


(x_1, y_1) is a maximal point, but both (x_2, y_2) and (x_3, y_3) are not.

16

2D maxima finding problem

- ▶ Given a set of n points, the 2D maxima finding problem is to find all of the maximal points among these n points.



The circle points are maximal points.

17

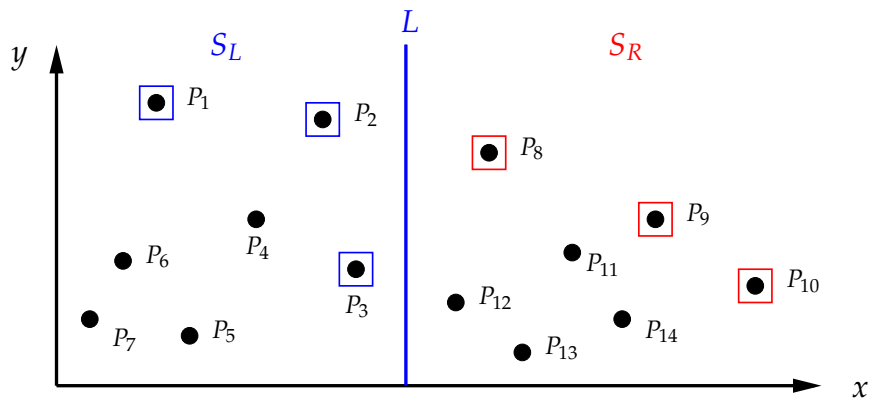
2D maxima finding problem (cont'd)

- ▶ A straightforward method we can use to solve the 2D maxima finding problem is to compare every pair of points.
- ▶ This method requires $\mathcal{O}(n^2)$ comparison of points.
- ▶ In fact, the divide and conquer strategy can solve the problem in $\mathcal{O}(n \log n)$ steps.

18

Divide and conquer method

Dividing process of 2D maxima finding problem



19

Divide and conquer method

Merging process of 2D maxima finding problem

- ▶ Note that the x -value of a point in S_R is always larger than the x -value of every point in S_L .

Observation:

A point in S_L is a maximal point if and only if its y -value is not less than the y -value of a maximal point in S_R .

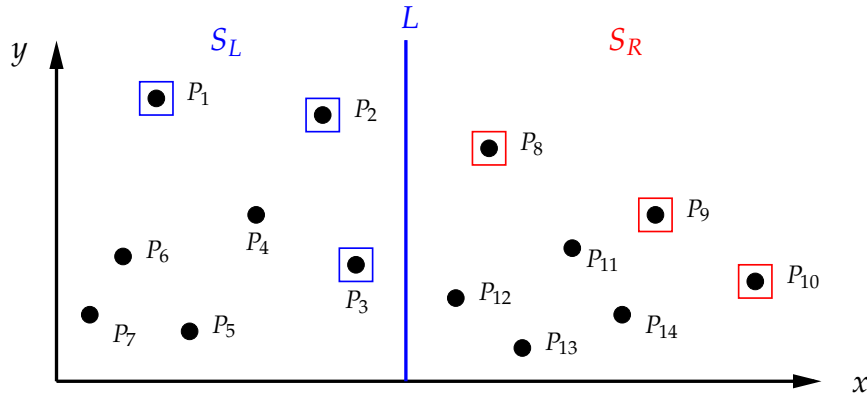
20

Divide and conquer method

Merging process of 2D maxima finding problem (cont'd)

Example:

P_3 is not a maximal point in S_L because its y -value is less than the y -values of P_8 and P_9 in S_R .



21

Divide and conquer algorithm

2D maxima finding problem

Algorithm: $FindMax(S)$

Input: A set S of n planar points.

Output: A set $MP(S)$ of all maximal points of S .

1. **if** S contains only one point **then** return S .
2. Find L perpendicular to the x -axis that separates S into S_L and S_R with $|S_L| = |S_R| = n/2$.
3. $FindMax(S_L)$ and $FindMax(S_R)$.
/* Recursively find maximal points of S_L and S_R */
4. Project the maximal points of S_L and S_R onto L and sort them by their y -values.
5. Conduct a linear scan on the projections and discard each of the maximal points of S_L if its y -value is less than the y -value of some maximal point of S_R .

22

Time complexity of $FindMax(S)$

- Let $T(n)$ be the time complexity of $FindMax(S)$.
- In fact, L can be found in $\mathcal{O}(n)$ time by finding the median of n numbers (will be discussed in Chapter 6 on prune and search).
- Step 4 costs $\mathcal{O}(n \log n)$ time using heap sort.

Recursive formula of $T(n)$:

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \mathcal{O}(n) + \mathcal{O}(n \log n) & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

- It can be proved that $T(n) = \mathcal{O}(n \log^2 n)$.

23

Time complexity of $FindMax(S)$ (cont'd)

Let $n = 2^k$ (i.e., $k = \log_2 n$).

$$\begin{aligned} T(n) &\leq 2T(\frac{n}{2}) + cn \log_2 n \\ &\leq 2(2T(\frac{n}{4}) + c\frac{n}{2} \log_2 \frac{n}{2}) + cn \log_2 n \\ &= 2^2 T(\frac{n}{2^2}) + cn(\log_2 \frac{n}{2^1} + \log_2 \frac{n}{2^0}) \\ &\vdots \\ &= 2^k T(\frac{n}{2^k}) + cn(\log_2 \frac{n}{2^{k-1}} + \log_2 \frac{n}{2^{k-2}} + \dots + \log_2 \frac{n}{2^0}) \\ &= nT(1) + cn(\log_2 2 + \log_2 4 + \dots + \log_2 n) \\ &= nT(1) + cn \left(\frac{(1 + \log_2 n) \log_2 n}{2} \right) \\ &= n + \frac{cn \log_2^2 n}{2} + \frac{cn \log_2 n}{2} \end{aligned}$$

Therefore, $T(n) = \mathcal{O}(n \log^2 n)$.

24

Time complexity of $FindMax(S)$ (cont'd)

- ▶ The divide and conquer algorithm is dominated by sorting in the merging step.
- ▶ The sorting actually can be done only once by a pre-sorting.
- ▶ Using the pre-sorting, the merging complexity is $\mathcal{O}(n)$.

Modified recursive formula of $T(n)$:

The total time complexity is $T(n) + \mathcal{O}(n \log n)$, where

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \mathcal{O}(n) + \mathcal{O}(n) & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$

- ▶ As shown before, $T(n) = \mathcal{O}(n \log n)$.
- ▶ Hence, the time complexity of $FindMax(S)$ is $\mathcal{O}(n \log n)$.

25

1D closest pair problem

Definition:

Given a set S of n points in one-dimensional space, the 1D closest pair problem is to find a pair of points which are closest together.

- ▶ **Example** The closest pair in the set $\{4, 11, 1, 7, 3\}$ is $(3, 4)$.

A simple algorithm to solve the 1D closest pair problem:

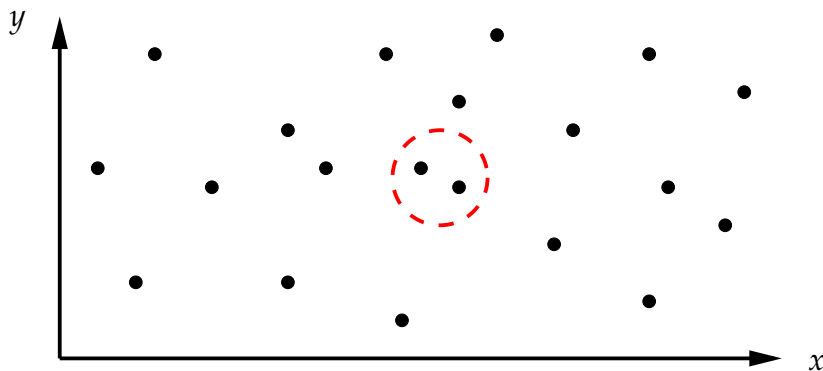
1. Sort n numbers.
 2. Linearly scan the sorted n numbers and find two consecutive numbers whose distance is minimum.
- ▶ The time complexity of this simple algorithm is $\mathcal{O}(n \log n)$.

26

2D closest pair problem

Definition:

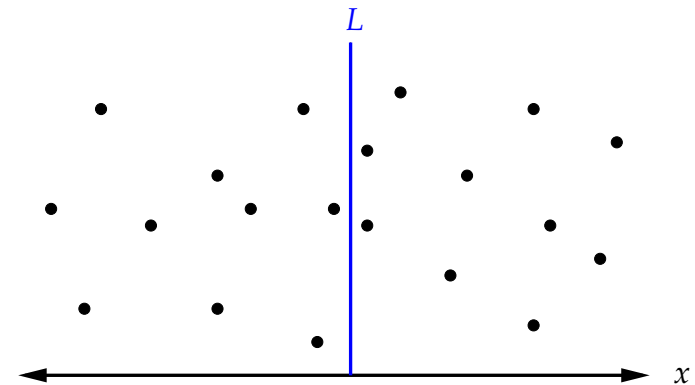
Given a set S of n points in two-dimensional space, find a pair of points which are closest together.



27

Divide and conquer for 2D closest pair

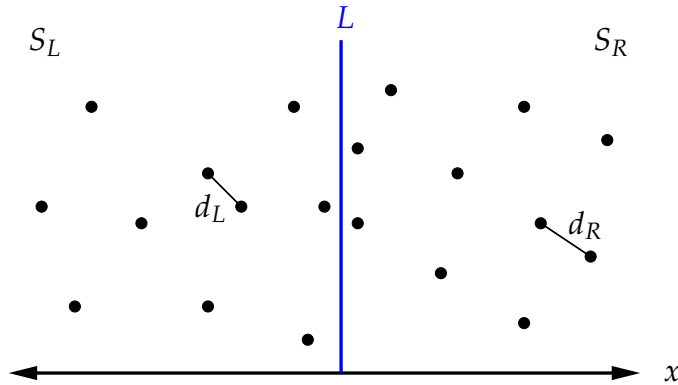
- ▶ Find a line L perpendicular to the x -axis to partition S into S_L and S_R such that $|S_L| \simeq |S_R|$.



28

Divide and conquer for 2D closest pair (cont'd)

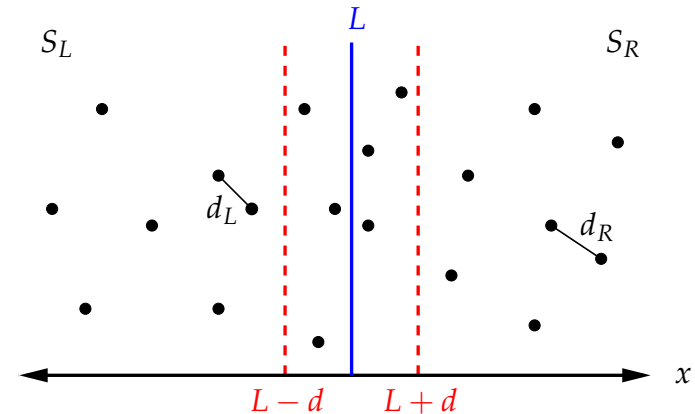
- By recursively solving the 2D closest pair problem in S_L and S_R , we obtain d_L and d_R , which denote the distances of the closest pairs in S_L and S_R , respectively.



29

Divide and conquer for 2D closest pair (cont'd)

- Let $d = \min(d_L, d_R)$.
- If the closest pair (P_a, P_b) of S consists of a point in S_L and a point in S_R , then P_a and P_b have to lie within a slab centered at line L and bounded by lines $L - d$ and $L + d$.



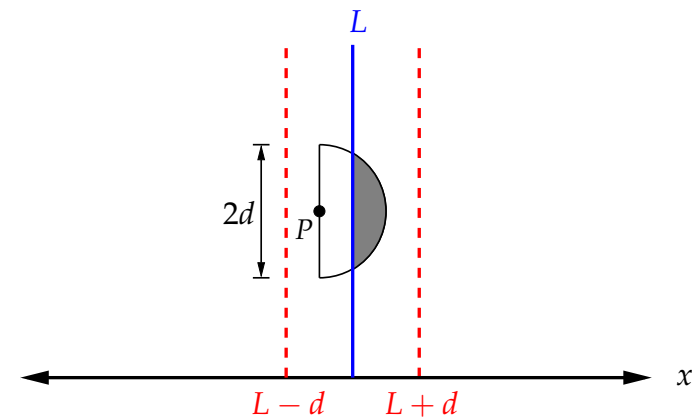
30

Divide and conquer for 2D closest pair (cont'd)

- In other words, we may examine only points in the slab during the merging step in the divide and conquer algorithm.
- In the worst case, there can be n points within the slab.
- Hence the brute force method to find the closest pair in this slab need to calculate $n^2/4$ distances and comparisons.
- This kind of merging step will not be good for our divide and conquer algorithm.

Divide and conquer for 2D closest pair (cont'd)

- If a point P in S_L and a point Q in S_R constitute a closest pair, the distance between P and Q must be less than d .
- It means that we do not consider a point too far away from P .
- Actually, we only have to examine the shaded area as follows.

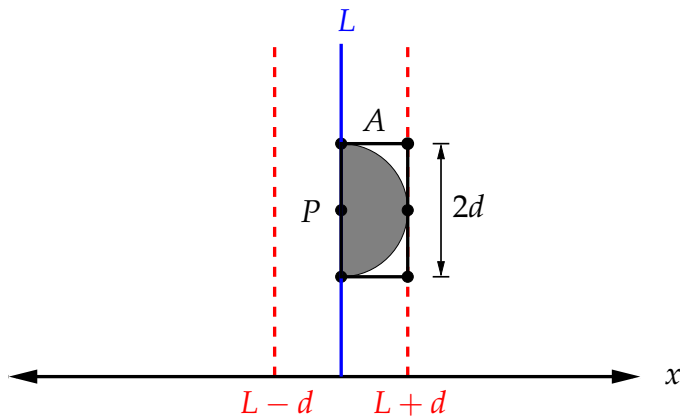


31

32

Divide and conquer for 2D closest pair (cont'd)

- If P is exactly on L , then we only have to examine points within the rectangle A .



There are at most 6 points in the rectangle A .

33

Divide and conquer algorithm

2D closest pair problem

Algorithm: $2dClosestPair(S)$

Input: A set S of n points in the plane.

Output: The distance of the closest pair in S .

Preprocessing: Sort the points of S by their y -values and x -values, respectively.

1. **if** S contains only one point **then** return ∞ as its distance.
 2. Find a median line L perpendicular to the x -axis to divide S into two equal sized subsets S_L and S_R .
 3. $2dClosestPair(S_L)$ and $2dClosestPair(S_R)$.
/* Recursively solve the problems of S_L and S_R */
 4. Let $d = \min\{d_L, d_R\}$.
-

34

Divide and conquer algorithm (cont'd)

2D closest pair problem

Algorithm: $2dClosestPair(S)$ (cont'd)

5. Project all points within the area bounded by $L - d$ and $L + d$ onto L .
 6. **for** each point P in the half slab bounded by $L - d$ and L with descending y -value **do**
 7. Let the y -value of P be y_p .
 8. Find all points in the half slab bounded by L and $L + d$ whose y -values fall within $y_p + d$ and $y_p - d$.
 9. **if** the shortest distance d' between P and a point in the other half slab is less than d **then**
 10. $d = d'$.
 11. **end if**
 12. **end for**
-

35

Divide and conquer algorithm (cont'd)

2D closest pair problem

Time complexity of $2dClosestPair(S)$:

The time complexity of this algorithm is $\mathcal{O}(n \log n) + T(n)$ and

$$T(n) = 2T\left(\frac{n}{2}\right) + Split(n) + Merge(n)$$

where $Split(n) = \mathcal{O}(n)$ and $Merge(n) = \mathcal{O}(n)$.

- $Split(n) = \mathcal{O}(n)$ because points are sorted by their x -values.
- $Merge(n) = \mathcal{O}(n)$ since the area bounded by $L - d$ and $L + d$ contains at most n points and for each point from the left half slab, at most 6 points need to be examined.

36

Divide and conquer algorithm (cont'd)

2D closest pair problem

Recursive formula of $T(n)$:

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \mathcal{O}(n) + \mathcal{O}(n) & \text{if } n > 1 \\ 1 & \text{if } n = 1 \end{cases}$$
$$= \mathcal{O}(n \log n)$$

- ▶ As a result, the time complexity of the divide and conquer algorithm for the 2D closest pair problem is $\mathcal{O}(n \log n)$.

37

Master theorem (cont'd)

Case 1:

$T(n) = \Theta(n^{\log_b a})$, if $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.

Example 1:

Let $T(n) = 9T(\frac{n}{3}) + n$. Then $T(n) = \Theta(n^2)$.

- ▶ In this case, we have $a = 9, b = 3$ and $f(n) = n$.
- ▶ We then have $n^{\log_b a} = n^{\log_3 9} = n^2$.
- ▶ By letting $\epsilon = 1$, we have $f(n) = n = \mathcal{O}(n) = \mathcal{O}(n^{\log_b a - \epsilon})$.
- ▶ Hence, $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.

39

Master theorem

- ▶ Let $T(n)$ be the recurrence defined below:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is a positive function.

- ▶ The above recurrence of $T(n)$ describes the running time of an algorithm that divides a problem of size n into a subproblems, each of size n/b .
- ▶ The a subproblems are solved recursively, each in $T(\frac{n}{b})$ time.
- ▶ The function $f(n)$ denotes the cost of dividing the problem and combining the results of the subproblems.

38

Master theorem (cont'd)

Case 2:

$T(n) = \Theta(n^{\log_b a} \log_2 n)$, if $f(n) = \Theta(n^{\log_b a})$.

Example 2:

Let $T(n) = T(\frac{2n}{3}) + 1$. Then $T(n) = \Theta(\log_2 n)$.

- ▶ In this case, we have $a = 1, b = \frac{3}{2}$ and $f(n) = 1$.
- ▶ Thus, we have $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.
- ▶ Clearly, $f(n) = 1 = \Theta(1) = \Theta(n^{\log_b a})$.
- ▶ Hence, $T(n) = \Theta(n^{\log_b a} \log_2 n) = \Theta(\log_2 n)$.

40

Master theorem (cont'd)

Case 3:

$T(n) = \Theta(f(n))$, if $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n .

Example 3:

Let $T(n) = 3T(\frac{n}{4}) + n \log_2 n$. Then $T(n) = \Theta(n \log_2 n)$.

- ▶ In this case, $n^{\log_b a} = n^{\log_4 3} \approx O(n^{0.793})$ and $f(n) = n \log_2 n$.
- ▶ By letting $\epsilon \approx 0.2$, we have $f(n) = n \log_2 n = \Omega(n^{\log_b a + \epsilon})$.
- ▶ Also, $af(\frac{n}{b}) = 3(\frac{n}{4}) \log_2(\frac{n}{4}) \leq (\frac{3}{4})n \log_2 n = cf(n)$ for $c = \frac{3}{4}$.

41

Master theorem (cont'd)

- ▶ To easily prove the correctness of the master theorem, we assume that $T(n)$ is defined on exact power of b (i.e., $n = 1, b, b^2, \dots$):

$$T(n) = \begin{cases} aT(\frac{n}{b}) + f(n) & \text{if } n = b^i \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

- ▶ Actually, the proof below can be applied to all positive integers n with a little modification.

43

Master theorem (cont'd)

Example 4:

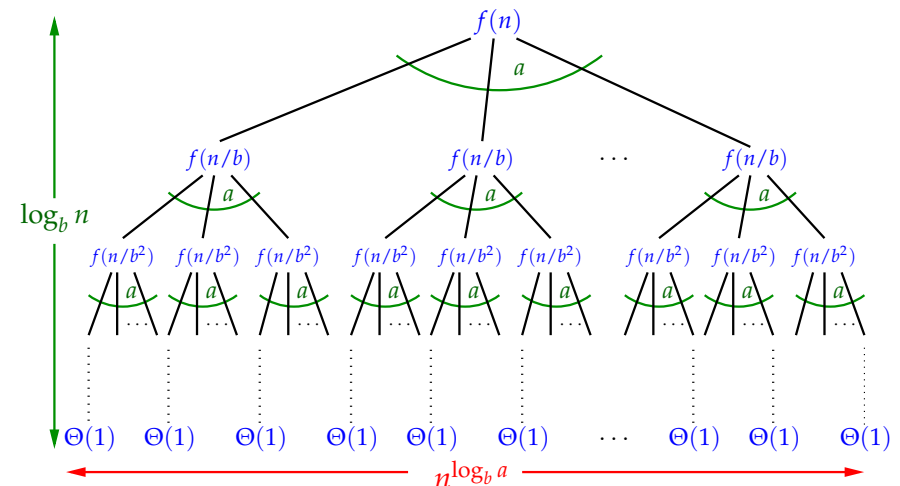
Let $T(n) = 2T(\frac{n}{2}) + n \log_2 n$. Then $T(n) = \Theta(n \log_2^2 n)$.

- ▶ In this case, $n^{\log_b a} = n^{\log_2 2} = n$ and $f(n) = n \log_2 n$.
- ▶ Note that case 3 of the master method cannot apply to this case, since $\log_2 n = O(n^\epsilon)$ for any positive constant ϵ .

42

Proof of master theorem

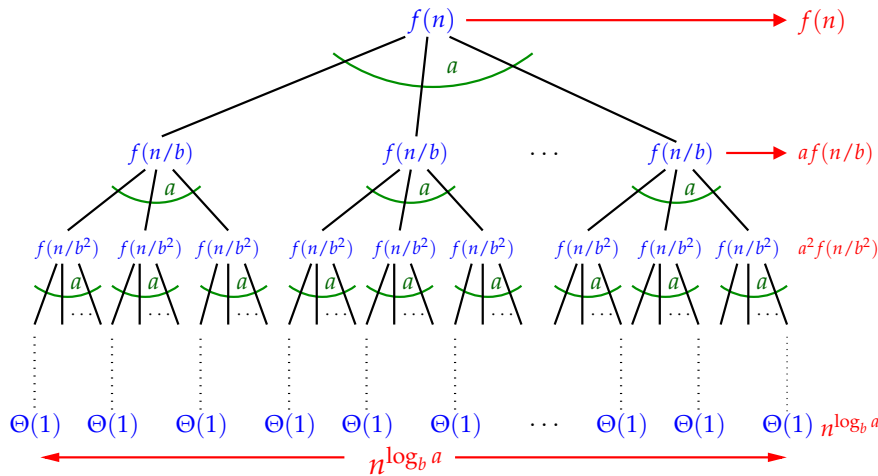
Recursive tree $T(n) = aT(n/b) + f(n)$



44

Proof of master theorem (cont'd)

Recursive tree $T(n) = aT(n/b) + f(n)$



45

Proof of master theorem (cont'd)

Lemma:

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right).$$

- ▶ In general, there are a^j nodes at depth j (starting from 0) and each such node has the cost $f\left(\frac{n}{b^j}\right)$.
- ▶ There are $a^{\log_b n} = n^{\log_b a}$ leaves in the tree, because each leaf is at depth $\log_b n$.
- ▶ Since the cost of each leaf is $T(1) = \Theta(1)$, the total cost of all leaves is $n^{\log_b a}$.
- ▶ This is the cost of doing all $n^{\log_b a}$ subproblems of size 1.

46

Proof of master theorem (cont'd)

- ▶ The cost for all internal nodes at depth j is $a^j f\left(\frac{n}{b^j}\right)$.
- ▶ Therefore, the total cost of all internal nodes is:

$$\sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

- ▶ This sum is the cost of dividing problems into subproblems and then recombining the subproblems.
- ▶ As a result, we have;

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

47

Proof of master theorem (cont'd)

- ▶ Let $g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$.

Case 1:

If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, $g(n) = \mathcal{O}(n^{\log_b a})$.

- ▶ Since $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$, $f\left(\frac{n}{b^j}\right) = \mathcal{O}\left(\left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$.
- ▶ Then we have $g(n) = \mathcal{O}\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$.

48

Proof of master theorem (cont'd)

$$\begin{aligned}
 \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} &= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j \\
 &= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j \\
 &= n^{\log_b a - \epsilon} \left(\frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}\right) \\
 &= n^{\log_b a - \epsilon} \left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right)
 \end{aligned}$$

Since b and ϵ are constants, $n^{\log_b a - \epsilon} \left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right) = \mathcal{O}(n^{\log_b a})$.

Therefore, we have $g(n) = \mathcal{O}(n^{\log_b a})$.

Proof of master theorem (cont'd)

Case 2:

If $f(n) = \Theta(n^{\log_b a})$, then $g(n) = \Theta(n^{\log_b a} \log_2 n)$.

- ▶ Assume that $f(n) = \Theta(n^{\log_b a})$.
- ▶ Then $f\left(\frac{n}{b^j}\right) = \Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a}\right)$ and

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) = \Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

49

50

Proof of master theorem (cont'd)

$$\begin{aligned}
 \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} &= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left(\frac{a}{b^{\log_b a}}\right)^j \\
 &= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1 \\
 &= n^{\log_b a} \log_b n
 \end{aligned}$$

Therefore, $g(n) = \Theta(n^{\log_b a} \log_b n) = \Theta(n^{\log_b a} \log_2 n)$

Proof of master theorem (cont'd)

Case 3:

If $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $g(n) = \Theta(f(n))$.

- ▶ Since $f(n)$ appears in the definition of $g(n)$ and all terms of $g(n)$ are nonnegative, we have $g(n) = \Omega(f(n))$.
- ▶ Assume that $af\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n .
- ▶ We can rewrite this assumption as $f\left(\frac{n}{b}\right) \leq \left(\frac{c}{a}\right)f(n)$.

51

52

Proof of master theorem (cont'd)

- By iterating the inequality $f\left(\frac{n}{b^j}\right) \leq \left(\frac{c}{a}\right)f(n)$ j times, we have (assume $\frac{n}{b^{j-1}}$ is still sufficiently large):

$$\begin{aligned} f\left(\frac{n}{b^j}\right) &\leq \left(\frac{c}{a}\right)^1 f\left(\frac{n}{b^{j-1}}\right) \\ &\leq \left(\frac{c}{a}\right)^2 f\left(\frac{n}{b^{j-2}}\right) \\ &\vdots \\ &\leq \left(\frac{c}{a}\right)^j f\left(\frac{n}{b^0}\right) \end{aligned}$$

- Therefore, $f\left(\frac{n}{b^j}\right) \leq \left(\frac{c}{a}\right)^j f(n)$ or equivalently $a^j f\left(\frac{n}{b^j}\right) \leq c^j f(n)$.

53

Proof of master theorem (cont'd)

- We use $\mathcal{O}(1)$ to denote the terms in $g(n)$ that are not covered by the assumption that n is sufficiently large.

$$\begin{aligned} g(n) &= \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right) \\ &\leq \sum_{j=0}^{\log_b n - 1} c^j f(n) + \mathcal{O}(1) \\ &\leq f(n) \sum_{j=0}^{\infty} c^j + \mathcal{O}(1) \\ &= f(n) \left(\frac{1}{1-c} \right) + \mathcal{O}(1) \\ &= \mathcal{O}(f(n)) \end{aligned}$$

- Therefore, $g(n) = \Theta(f(n))$.

54

Proof of master theorem (cont'd)

Case 1: Total cost is dominated by cost of leaves

If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

$$\because T(n) = \Theta(n^{\log_b a}) + \mathcal{O}(n^{\log_b a}) = \Theta(n^{\log_b a}).$$

Case 2: Total cost is evenly distributed among levels

If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log_2 n)$.

$$\because T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_b a} \log_2 n).$$

55

Proof of master theorem (cont'd)

Case 3: Total cost is dominated by cost of root

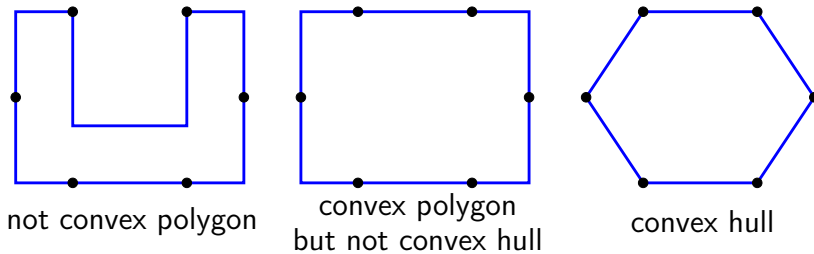
If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $a f\left(\frac{n}{b}\right) \leq c f(n)$ for some constant $c < 1$ and for all sufficiently large n , $T(n) = \Theta(f(n))$.

$$\because T(n) = \Theta(n^{\log_b a}) + \Theta(f(n)) = \Theta(f(n)).$$

56

Convex hull

- ▶ Convex polygon is a polygon P such that any line connecting any two points inside P must lie in P .
- ▶ The convex hull of a given set of planar points is the smallest convex polygon containing all of the points

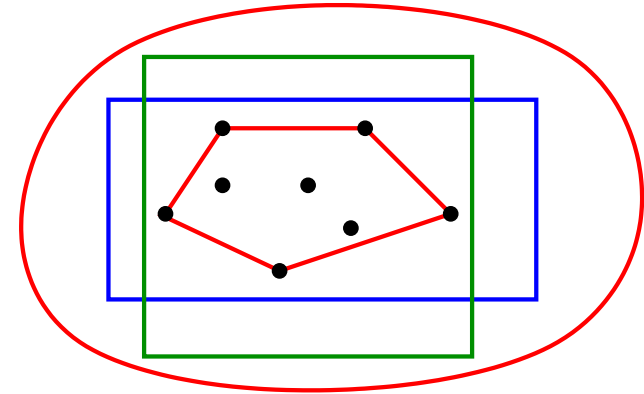


57

Convex hull problem

Definition:

Given a set S of planar points, find a convex hull for S , i.e., obtain the vertices of the convex hull in counterclockwise (clockwise) order.



58

Convex hull problem

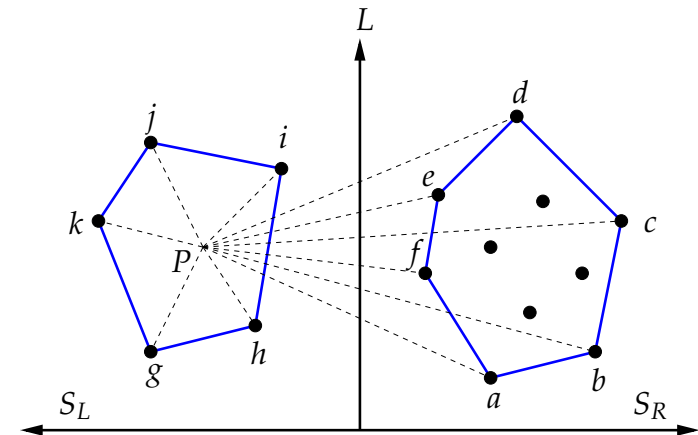
Divide and conquer algorithm

Algorithm: $CH(S)$ for computing a convex hull for S

1. **if** $|S| \leq 3$ **then** $CH(S) = S$ and return.
 2. Find a median line perpendicular to the x -axis to divide S into S_L and S_R .
 3. Recursively find $CH(S_L)$ and $CH(S_R)$.
 4. Find an interior point P of $CH(S_L)$. /* Merging step */
 5. Find v_1 and v_2 of $CH(S_R)$ such that $CH(S_R)$ is divided into two sequences of vertices that have increasing polar angles with respect to P , and let $y(v_1) > y(v_2)$.
 S_1 : $V(CH(S_L))$ in counterclockwise direction.
 S_2 : $V(CH(S_R))$ from v_2 to v_1 in counterclockwise.
 S_3 : $V(CH(S_R))$ from v_2 to v_1 in clockwise order.
 6. Merge S_1, S_2, S_3 and conduct the Graham scan.
-

59

Convex hull problem (cont'd)



- ▶ $CH(S_L) = (g, h, i, j, k)$ and $CH(S_R) = (a, b, c, d, e, f)$
- ▶ $v_1 = d$ and $v_2 = a$
- ▶ $S_1 = (g, h, i, j, k)$, $S_2 = (a, b, c, d)$ and $S_3 = (f, e)$

60

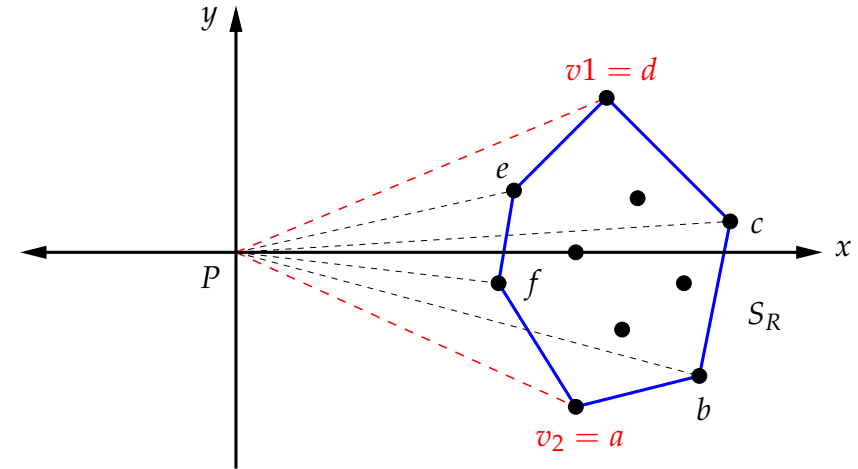
Convex hull problem (cont'd)

How to find v_1 and v_2 ?

- Construct a horizontal line through P .
- If this line intersects $CH(S_R)$, then v_1 is the vertex of $CH(S_R)$ with the greatest polar angle $< \frac{\pi}{2}$ and v_2 is the vertex with the least polar angle $> \frac{3\pi}{2}$.
- If this line does not intersect $CH(S_R)$, then v_1 is the vertex of $CH(S_R)$ with the largest polar angle and v_2 is the vertex with the smallest polar angle.

Convex hull problem (cont'd)

How to find v_1 and v_2 ?

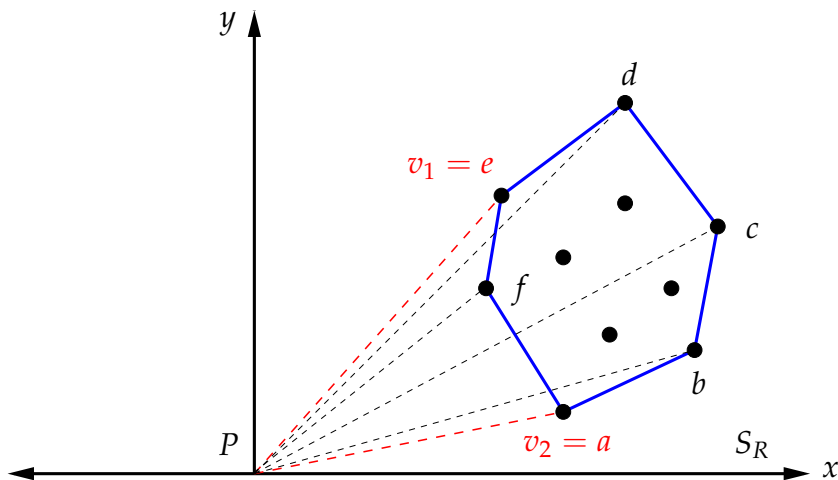


61

62

Convex hull problem (cont'd)

How to find v_1 and v_2 ?

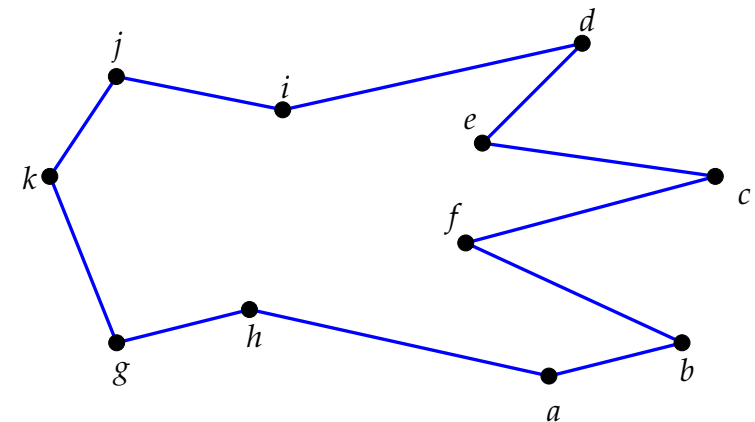


63

Convex hull problem (cont'd)

Merging step in the divide and conquer algorithm

- Merge S_1, S_2, S_3 to $(g, h, a, b, f, c, e, d, i, j, k)$.

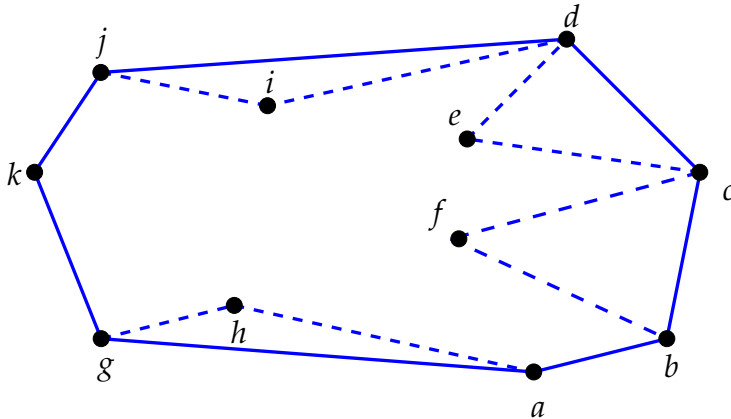


64

Convex hull problem (cont'd)

Merging step in the divide and conquer algorithm

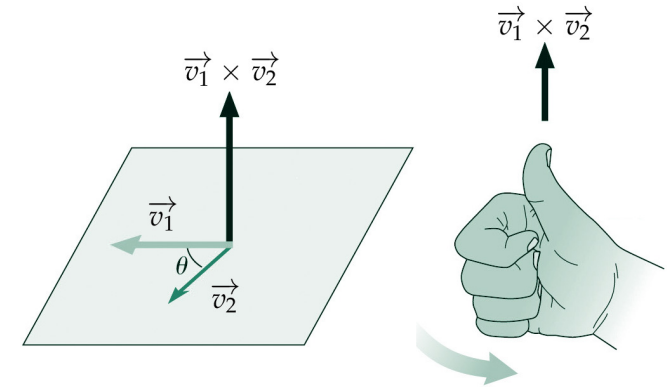
- Graham scan examines the points of the new sequence one by one and eliminates those points h, f, e, i with reflexive angles.



65

Cross product

- Given two vectors $\vec{v}_1 = (x_1, y_1)$ and $\vec{v}_2 = (x_2, y_2)$, the cross product $\vec{v}_1 \times \vec{v}_2$ is a vector, which is perpendicular to both \vec{v}_1 and \vec{v}_2 , with a direction given by the right-hand rule and a magnitude equal to $|x_1y_2 - x_2y_1|$.



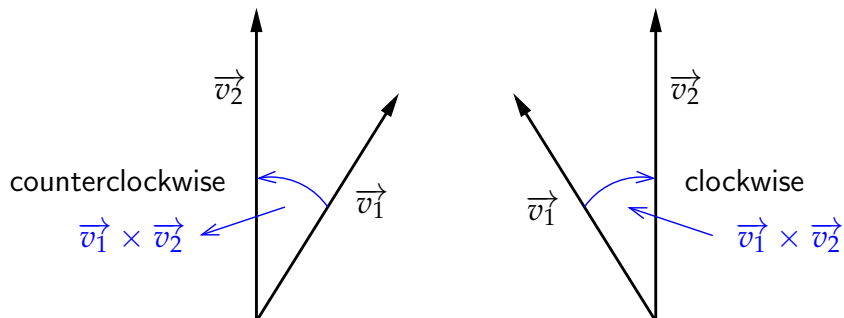
66

Cross product (cont'd)

- To determine the direction of $\vec{v}_1 \times \vec{v}_2$ in a convenient way, we treat $\vec{v}_1 \times \vec{v}_2$ simply as the value of $x_1y_2 - x_2y_1$.

$$\vec{v}_1 \times \vec{v}_2 = x_1y_2 - x_2y_1$$

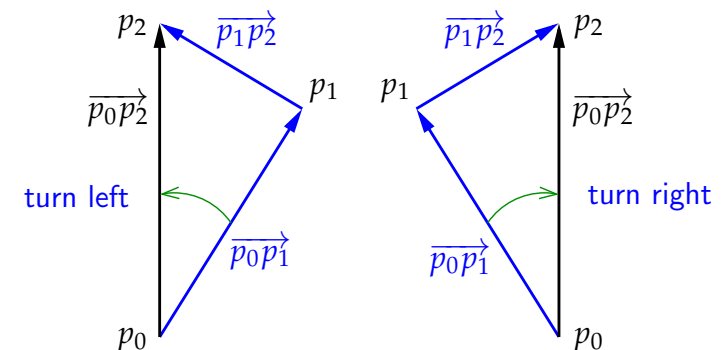
- If $\vec{v}_1 \times \vec{v}_2$ is positive, \vec{v}_2 is counterclockwise from \vec{v}_1 .
- If $\vec{v}_1 \times \vec{v}_2$ is negative, \vec{v}_2 is clockwise from \vec{v}_1 .



67

Turn left or right?

- We can use cross product to determine whether two consecutive line segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$ turn left or right at point p_1 .
- If the sign of $\overrightarrow{p_0p_1} \times \overrightarrow{p_1p_2}$ is positive, $\overrightarrow{p_0p_2}$ is counterclockwise from $\overrightarrow{p_0p_1}$ and hence we make a left turn at p_1 .
- If the sign of $\overrightarrow{p_0p_1} \times \overrightarrow{p_1p_2}$ is negative, $\overrightarrow{p_0p_2}$ is clockwise from $\overrightarrow{p_0p_1}$ and hence we make a right turn at p_1 .



68

Convex hull problem (cont'd)

Divide and conquer algorithm

Time complexity of divide and conquer algorithm:

$$T(n) = 2T\left(\frac{n}{2}\right) + \text{Split}(n) + \text{Merge}(n)$$

- ▶ $\text{Split}(n) = \mathcal{O}(n)$ for median finding process
- ▶ $\text{Merge}(n) = \mathcal{O}(n)$ for finding P , determination of v_1 and v_2 , merging of S_1, S_2, S_3 , and Graham scan
- ▶ In other words, we have $T(n) = 2T\left(\frac{n}{2}\right) + \mathcal{O}(n) = \mathcal{O}(n \log n)$.
- ▶ Note that it is optimal because the lower bound of the convex hull problem is $\Omega(n \log n)$.