CMPS 6610 – Fall 2018

Divide-and-Conquer

Carola Wenk

Slides courtesy of Charles Leiserson
with changes and additions by Carola Wenk

The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems of sizes that are fractions of the original problem size.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.

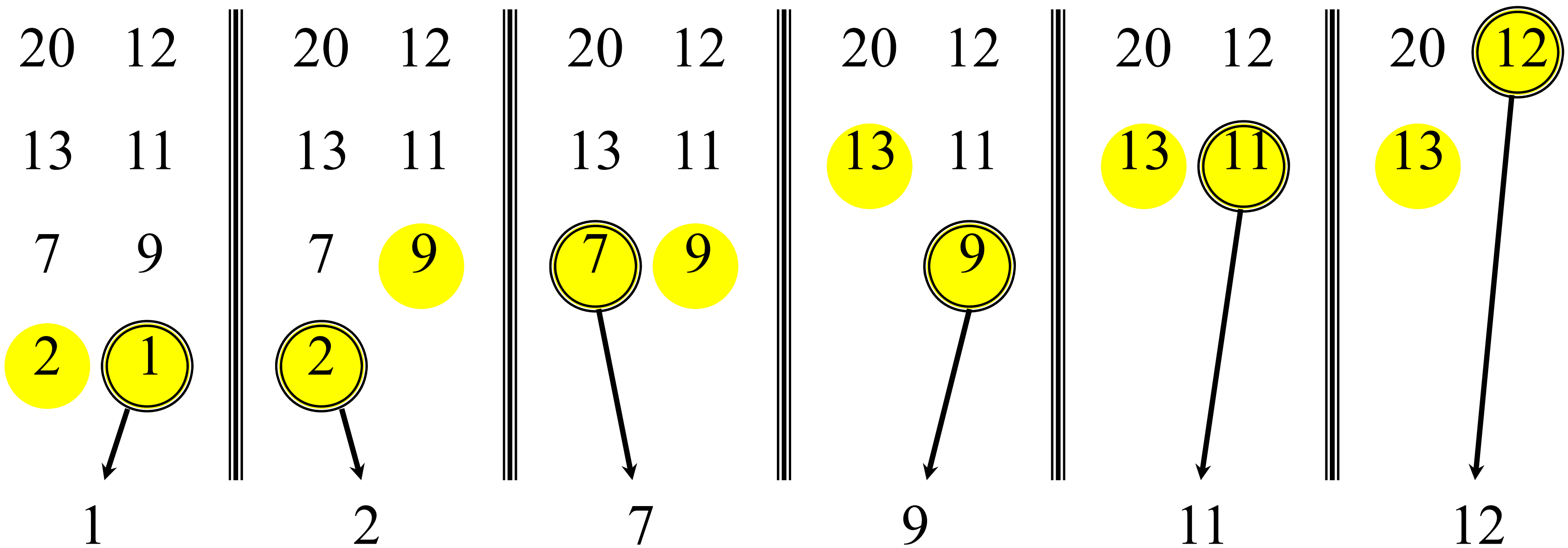
Merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays of size $n/2$
3. ***Combine:*** Linear-time key subroutine **MERGE**

MERGE-SORT ($A[0 \,.\,.\, n\text{-}1]$)

1. If $n = 1$, done.
2. **MERGE-SORT** ($A[\,0 \,.\,.\, \lceil n/2 \rceil \text{-}1]$)
3. **MERGE-SORT** ($A[\,\lceil n/2 \rceil \,.\,.\, n\text{-}1\,]$)
4. "***Merge***" the 2 sorted lists.

Merging two sorted arrays



Time $dn \in \Theta(n)$ to merge a total of n elements (linear time).

Analyzing merge sort

$T(n)$	MERGE-SORT ($A[0 \,.\,.\, n\text{-}1]$)
d_0	1. If $n = 1$, done.
$T(n/2)$	2. MERGE-SORT ($A[\,0 \,.\,.\, \lceil n/2 \rceil + 1]$)
$T(n/2)$	3. MERGE-SORT ($A[\,\lceil n/2 \rceil \,.\,.\, n\text{-}1\,]$)
dn	4. "***Merge***" the 2 sorted lists.

Sloppiness: Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$, but it turns out not to matter asymptotically.

Recurrence for merge sort

$$T(n) = \begin{cases} d_0 \text{ if } n = 1; \\ 2T(n/2) + dn \text{ if } n > 1. \end{cases}$$

- But what does $T(n)$ solve to? I.e., is it $O(n)$ or $O(n^2)$ or $O(n^3)$ or …?

Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.

$$T(n)$$

Recursion tree

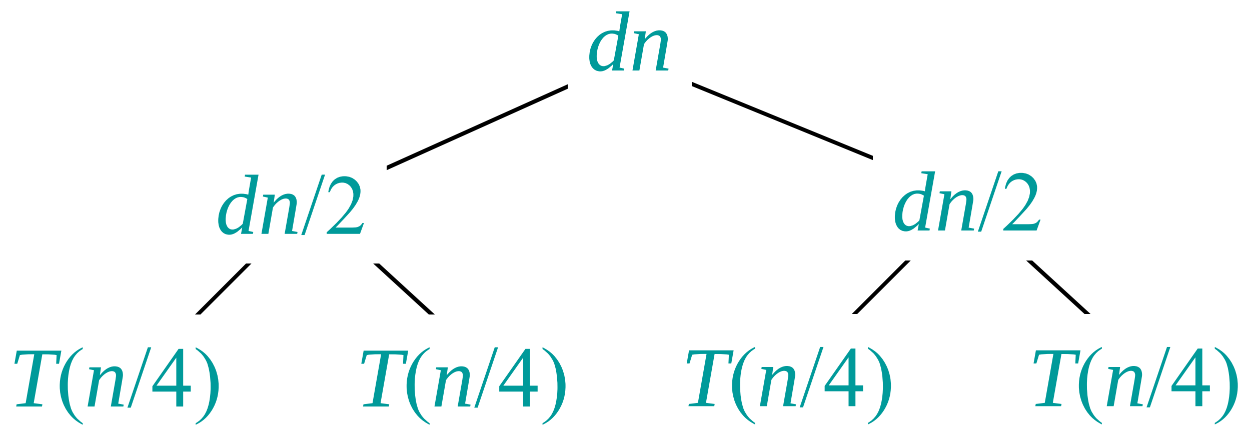
Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.

$$dn$$

$$T(n/2) \qquad\qquad T(n/2)$$

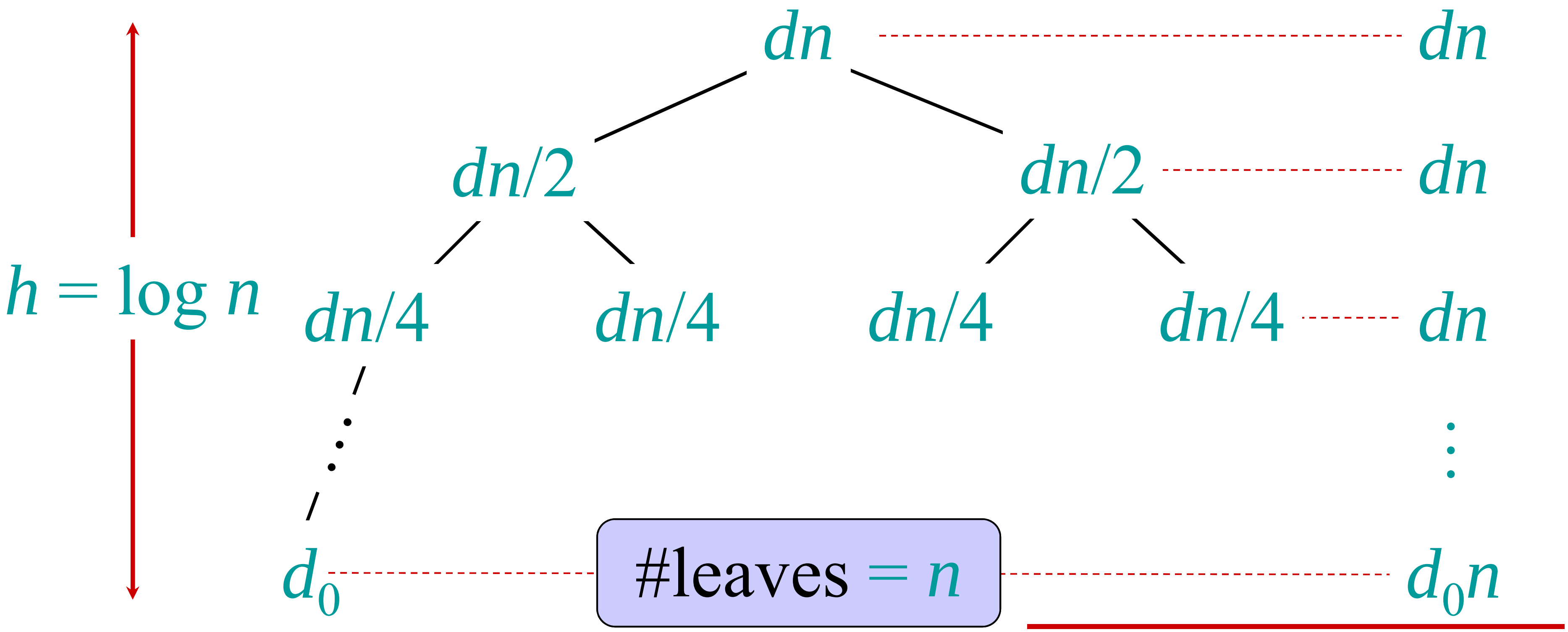
Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



Recursion tree

Solve $T(n) = 2T(n/2) + dn$, where $d > 0$ is constant.



Total $dn \log n + d_0 n$

Mergesort Conclusions

- Merge sort runs in $\Theta(n \log n)$ time.
- $\Theta(n \log n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so. (Why not earlier?)

Recursion-tree method

- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- It is good for generating **guesses** of what the runtime could be.

But: Need to **verify** that the guess is correct.
→ Induction (substitution method)

Substitution method

The most general method to solve a recurrence (prove O and Ω separately):

1. ***Guess*** the form of the solution:
   (e.g. using recursion trees, or expansion)
2. ***Verify*** by induction (inductive step).
3. ***Solve*** for O-constants n_0 and c (base case of induction)

Master Theorem

$$T(n) = a\,T(n/b) + f(n)$$

CASE 1:

$f(n) = O(n^{\log_b a - \varepsilon})$ $\Rightarrow T(n) = \Theta(n^{\log_b a})$

for some $\varepsilon > 0$

CASE 2:

$f(n) = \Theta(n^{\log_b a} \log^k n)$ $\Rightarrow T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

for some $k \geq 0$

CASE 3:

(i) $f(n) = \Omega(n^{\log_b a + \varepsilon})$

for some $\varepsilon > 0$

and (ii) $a\,f(n/b) \leq c\,f(n)$

for some $c < 1$

(i) and (ii) together $\Rightarrow T(n) = \Theta(f(n))$

Powering a number

Problem: Compute a^n, where $n \in \boldsymbol{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm: **(recursive squaring)**

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \;\Rightarrow\; T(n) = \Theta(\lg n)\,.$$

The master method

The master method applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n) \, ,$$

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

Master Theorem

$$T(n) = a\,T(n/b) + f(n)$$

CASE 1:

$f(n) = O(n^{\log_b a - \varepsilon}) \quad \Rightarrow T(n) = \Theta(n^{\log_b a})$

for some $\varepsilon > 0$

CASE 2:

$f(n) = \Theta(n^{\log_b a} \log^k n) \quad \Rightarrow T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

for some $k \geq 0$

CASE 3:

(i) $f(n) = \Omega(n^{\log_b a + \varepsilon})$

for some $\varepsilon > 0$

and (ii) $a\,f(n/b) \leq c\,f(n)$

for some $c < 1$

$\Rightarrow T(n) = \Theta(f(n))$

Example: merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + O(n)$$

 # subproblems — *subproblem size* — *work dividing and combining*

$$n^{\log_b a} = n^{\log_2 2} = n^1 = n \Rightarrow \text{CASE 2 } (k = 0)$$

$$\Rightarrow T(n) = \Theta(n \log n)\,.$$

Example: binary search

$$T(n) = 1\,T(n/2) + \Theta(1)$$

 # subproblems

subproblem size

work dividing and combining

$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$

$$\Rightarrow T(n) = \Theta(\log n)\ .$$

How to apply the theorem

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

   Solution: $T(n) = \Theta(n^{\log_b a})$.

2. $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.

How to apply the theorem

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that $a f(n/b) \le c f(n)$ for some constant $c < 1$.

   Solution: $T(n) = \Theta(f(n))$.

Master theorem: Examples

Ex. $T(n) = 4T(n/2) + \sqrt{n}$
$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = \sqrt{n}$.
CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1.5$.
$\therefore T(n) = \Theta(n^2)$.

Ex. $T(n) = 4T(n/2) + n^2$
$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2$.
CASE 2: $f(n) = \Theta(n^2 \log^0 n)$, that is, $k = 0$.
$\therefore T(n) = \Theta(n^2 \log n)$.

Master theorem: Examples

Ex. $T(n) = 4T(n/2) + n^3$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$

Case 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(n/2)^3 \leq cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore T(n) = \Theta(n^3)$.

Ex. $T(n) = 4T(n/2) + n^2/\log n$

$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2/\log n.$

Master method does not apply. In particular, for every constant $\varepsilon > 0$, we have $\log n \in o(n^\varepsilon)$.

Conclusion

- Divide and conquer is just one of several powerful techniques for algorithm design.
- Divide-and-conquer algorithms can be analyzed using recurrences and the master method .
- Can lead to more efficient algorithms

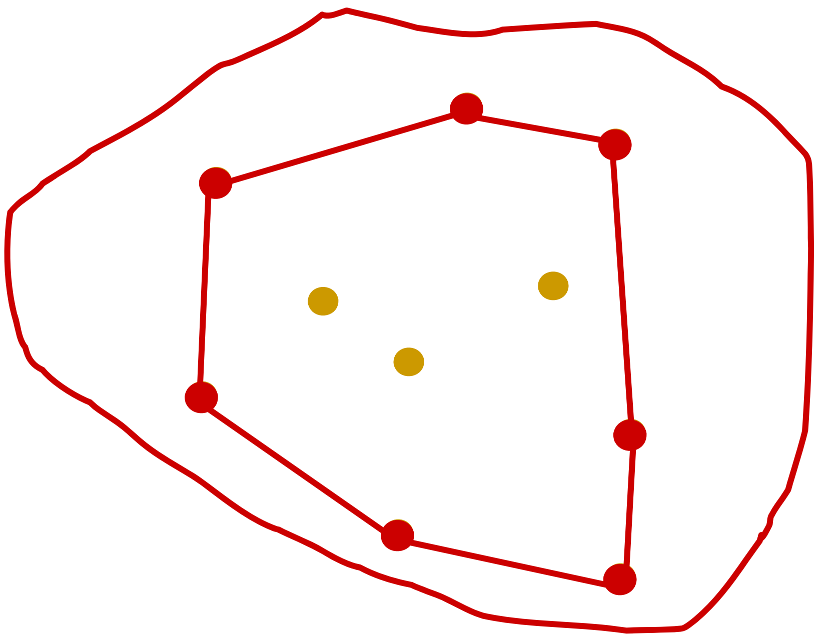
[redacted]

Convex Hull Problem

- Given a set of pins on a pinboard

  and a rubber band around them.

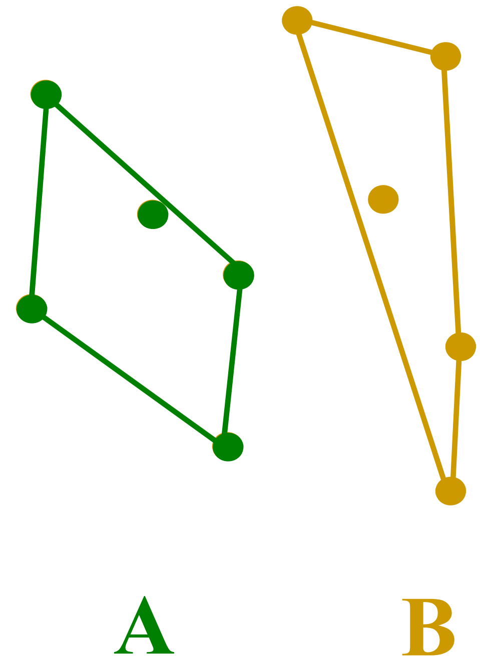
  How does the rubber band look when it snaps tight?



- The convex hull of a point set is one of the simplest shape approximations for a set of points.

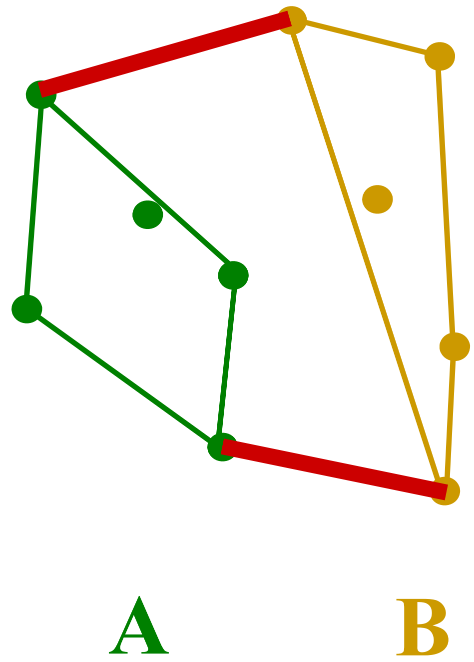
Convex Hull: Divide & Conquer

- Preprocessing: sort the points by x-coordinate
- Divide the set of points into two sets **A** and **B**:
  - **A** contains the left $\lfloor n/2 \rfloor$ points,
  - **B** contains the right $\lceil n/2 \rceil$ points
- Recursively compute the convex hull of **A**
- Recursively compute the convex hull of **B**
- Merge the two convex hulls



Merging

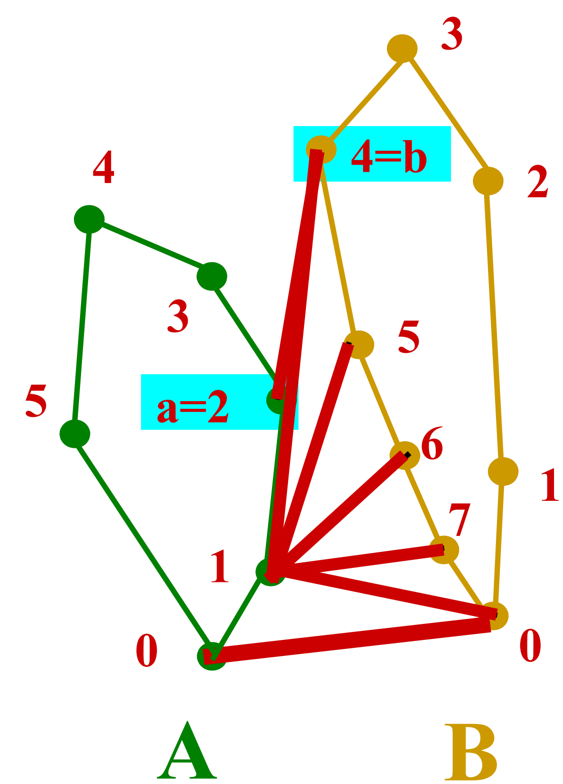
- **Find upper and lower tangent**
- With those tangents the convex hull of A∪B can be computed from the convex hulls of A and the convex hull of B in O(n) linear time

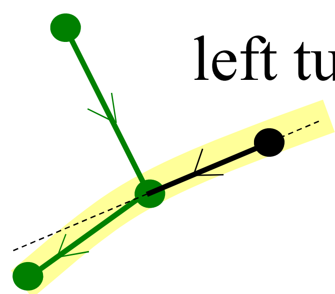


Finding the lower tangent

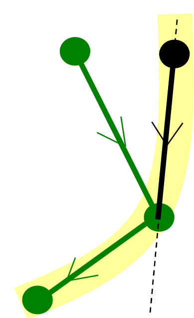
```
a = rightmost point of A
b = leftmost point of B
while T=ab not lower tangent to both
      convex hulls of A and B do{
    while T not lower tangent to
     convex hull of A do{
       a=a-1
    }
    while T not lower tangent to
     convex hull of B do{
       b=b+1
     }
}
```

check with orientation test





left turn



right turn

Convex Hull: Runtime

- Preprocessing: sort the points by x-coordinate — $O(n \log n)$ just once
- Divide the set of points into two sets **A** and **B**: — $O(1)$
  - **A** contains the left $\lfloor n/2 \rfloor$ points,
  - **B** contains the right $\lceil n/2 \rceil$ points
- Recursively compute the convex hull of **A** — $T(n/2)$
- Recursively compute the convex hull of **B** — $T(n/2)$
- Merge the two convex hulls — $O(n)$

Convex Hull: Runtime

- Runtime Recurrence:

  $T(n) = 2\,T(n/2) + cn$

- Solves to $T(n) = \Theta(n \log n)$