Divide and Conquer

- A general paradigm for algorithm design; inspired by emperors and colonizers.
- Three-step process:
  1. Divide the problem into smaller problems.
  2. Conquer by solving these problems.
  3. Combine these results together.
- Examples: Binary Search, Merge sort, Quicksort etc. Matrix multiplication, Selection, Convex Hulls.

Binary Search

- **Search for x in a sorted array A.**

Binary-Search (A, p, q, x)

1. if $p > q$ **return -1;**
2. $r = \lfloor (p + q)/2 \rfloor$
3. if $x = A[r]$ **return** r
4. else if $x < A[r]$ **Binary-Search**(A, p, r, x)
5. else Binary-Search$(A, r + 1, q, x)$

- **The initial call is Binary-Search$(A, 1, n, x)$.**

Binary Search

- **Let $T(n)$ denote the worst-case time to binary search in an array of length n.**
- **Recurrence is $T(n) = T(n/2) + O(1)$.**
- $T(n) = O(\log n)$.

Merge Sort

- **Sort an unordered array of numbers A.**

  Merge-Sort (A, p, q)

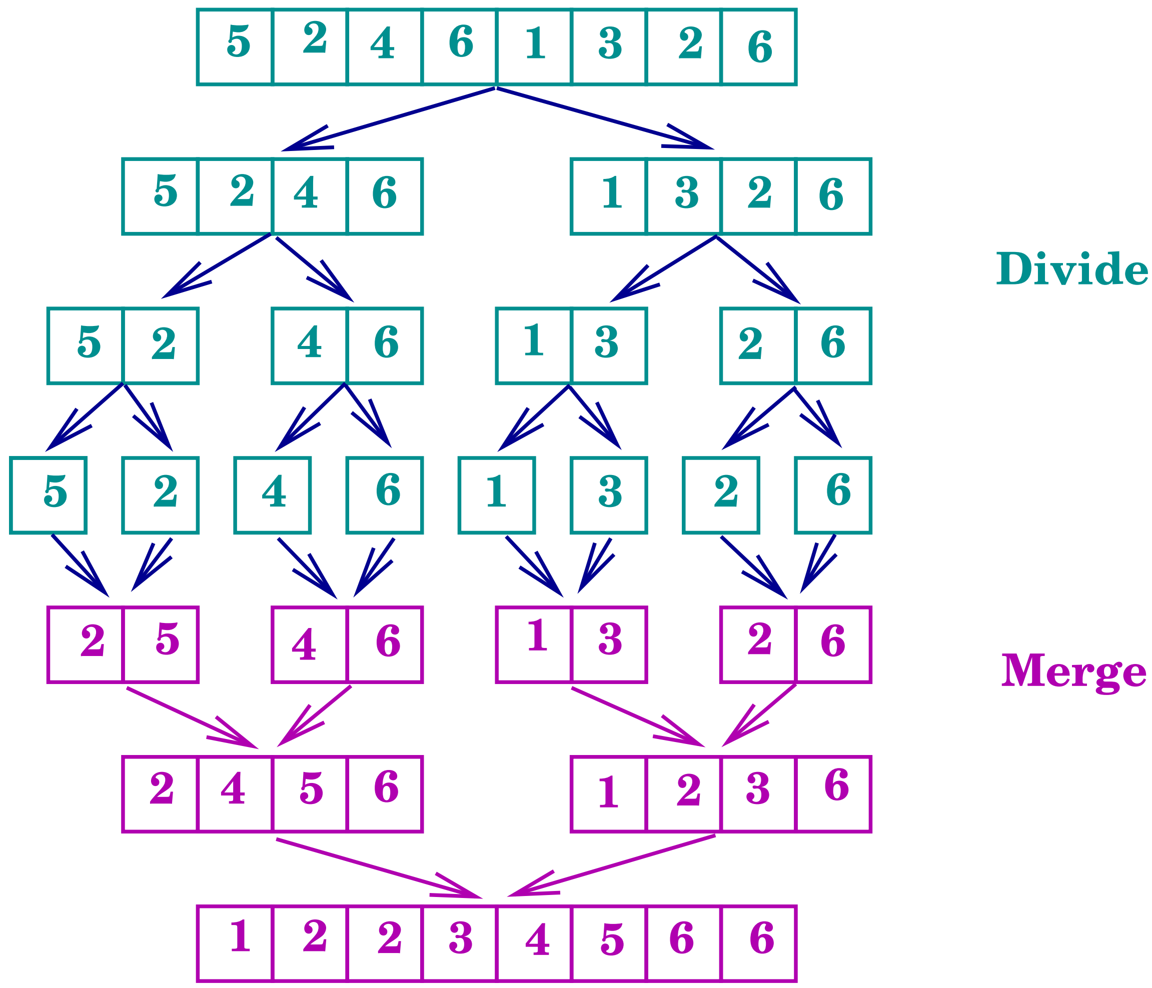
  1. **if** $p \geq q$ **return** A;
  2. $r = \lfloor \ (p+q)/2 \ \rfloor$
  3. **Merge-Sort** (A, p, r)
  4. **Merge-Sort** $(A, r+1, q)$
  5. **MERGE** (A, p, q, r)

- **The initial call is Merge-Sort** $(A, 1, n)$.

Merge Sort

- **Let $T(n)$ denote the worst-case time to merge sort an array of length n.**
- **Recurrence is $T(n) = 2T(n/2) + O(n)$.**
- $T(n) = O(n \log n)$.

Merge Sort: Illustration



Multiplying Numbers

- We want to multiply two n-bit numbers. Cost is number of elementary bit steps.
- Grade school method has $\Theta(n^2)$ cost.:

```
               xxxxxxxx
               xxxxxxxx
 -------------------------
              xxxxxxxxx
             xxxxxxxxx
                 .
                 .
                 .
       xxxxxxxxx
 -------------------------
       xxxxxxxxxxxxxxxx
```

- n^2 multiplies, $n^2/2$ additions, plus some carries.

Why Bother?

- **Doesn't hardware provide multiply? It is fast, optimized, and free. So, why bother?**
- **True for numbers that fit in one computer word. But what if numbers are very large.**
- **Cryptography (encryption, digital signatures) uses big number "keys." Typically 256 to 1024 bits long!**
- n^2 **multiplication too slow for such large numbers.**
- **Karatsuba's (1962) divide-and-conquer scheme multiplies two** n **bit numbers in** $O(n^{1.59})$ **steps.**

Karatsuba's Algorithm

- **Let X and Y be two n-bit numbers. Write**

$$\begin{aligned} X &= a\ b \\ Y &= c\ d \end{aligned}$$

- a, b, c, d **are $n/2$ bit numbers. (Assume $n = 2^k$.)**

$$\begin{aligned} XY &= (a2^{n/2} + b)(c2^{n/2} + d) \\ &= ac2^n + (ad + bc)2^{n/2} + bd \end{aligned}$$

An Example

- $X = 4729 \qquad Y = 1326.$
- $a = 47; b = 29 \qquad c = 13; d = 26.$
- $ac = 47 * 13 = 611$
- $ad = 47 * 26 = 1222$
- $bc = 29 * 13 = 377$
- $bd = 29 * 26 = 754$
- $XY = 6110000 + 159900 + 754$
- $XY = 6270654$

Karatsuba's Algorithm

- **This is D&C: Solve 4 problems, each of size $n/2$; then perform $O(n)$ shifts to multiply the terms by 2^n and $2^{n/2}$.**
- **We can write the recurrence as**

$$T(n) = 4T(n/2) + O(n)$$

- **But this solves to $T(n) = O(n^2)$!**

Karatsuba's Algorithm

- $XY = ac2^n + (ad + bc)2^{n/2} + bd$.
- **Note that** $(a - b)(c - d) = (ac + bd) - (ad + bc)$.
- **Solve 3 subproblems:** $ac, \quad bd, \quad (a - b)(c - d)$.
- **We can get all the terms needed for** XY **by addition and subtraction!**
- **The recurrence for this algorithm is**

$$T(n) = 3T(n/2) + O(n) = O(n^{\log_2 3}).$$

- **The complexity is** $O(n^{\log_2 3}) \approx O(n^{1.59})$.

Recurrence Solving: Review

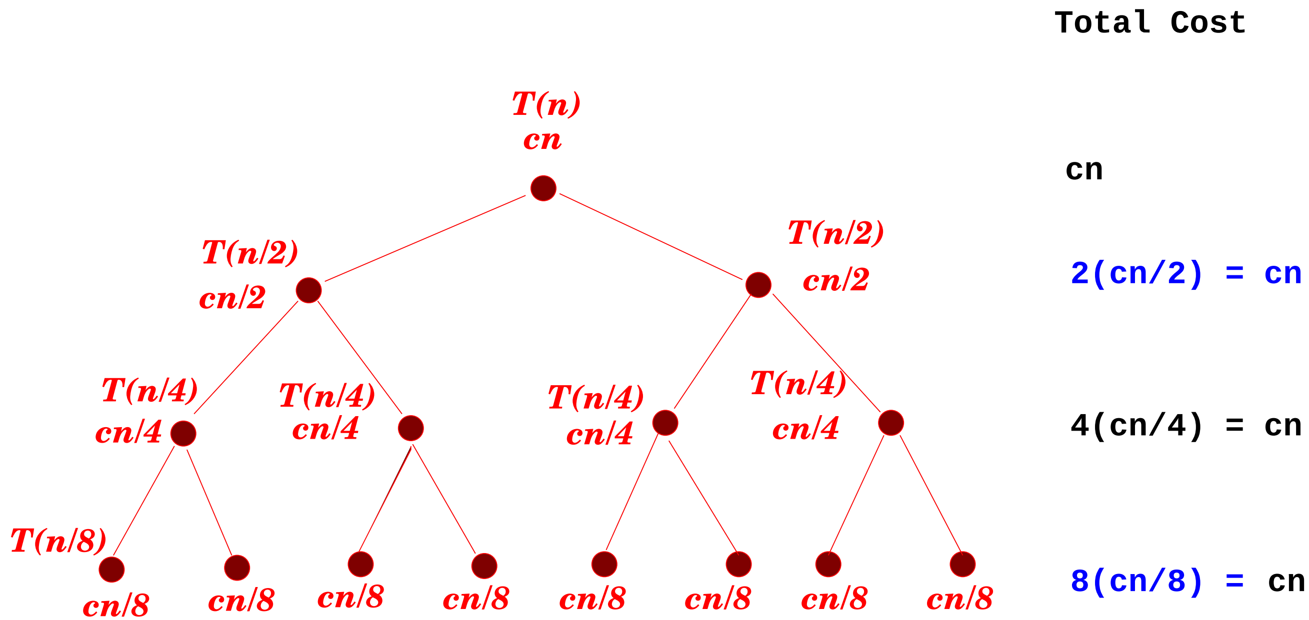
- $T(n) = 2T(n/2) + cn$, **with** $T(1) = 1$.
- **By term expansion.**

$$
\begin{aligned}
T(n) &= 2T(n/2) + cn \\
&= 2\left(2T(n/2^2) + cn/2\right) + cn = 2^2T(n/2^2) + 2cn \\
&= 2^2\left(2T(n/2^3) + cn/2^2\right) + 2cn = 2^3T(n/2^3) + 3cn \\
&\vdots \\
&= 2^iT(n/2^i) + icn
\end{aligned}
$$

- **Set** $i = \log_2 n$. **Use** $T(1) = 1$.
- **We get** $T(n) = n + cn(\log n) = O(n \log n)$.

The Tree View

- $T(n) = 2T(n/2) + cn$, **with** $T(1) = 1$.



- **# leaves** $= n$; **# levels** $= \log n$.
- **Work per level is** $O(n)$**, so total is** $O(n \log n)$**.**

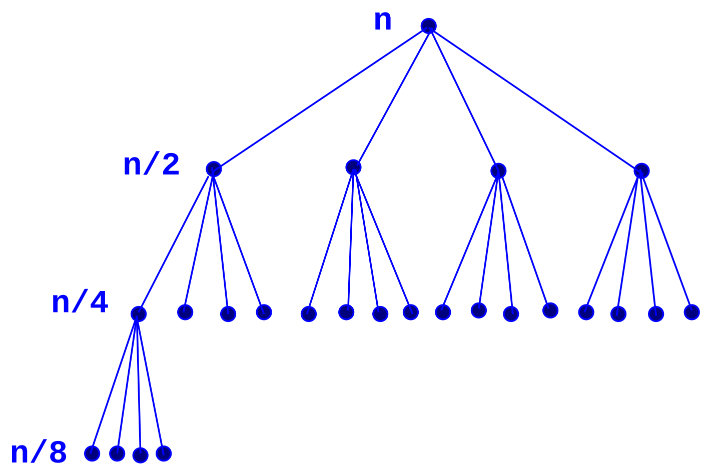
Solving By Induction

- **Recurrence:** $T(n) = 2T(n/2) + cn$.
- **Base case:** $T(1) = 1$.
- **Claim:** $T(n) = cn \log n + cn$.

$$
\begin{aligned}
T(n) &= 2T(n/2) + cn \\
&= 2\left(c(n/2)\log(n/2) + cn/2\right) + cn \\
&= cn\left(\log n - 1 + 1\right) + cn \\
&= cn \log n + cn
\end{aligned}
$$

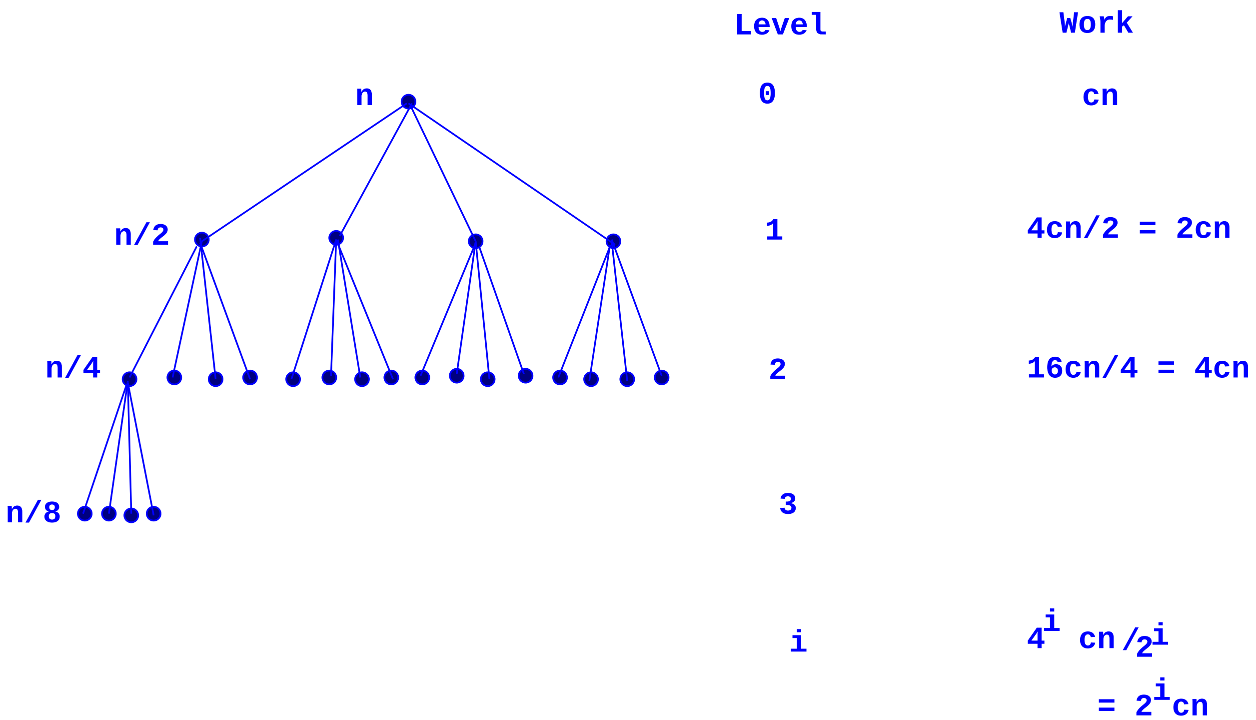
More Examples

- $T(n) \ = \ 4T(n/2) + cn, \qquad T(1) \ = \ 1.$



Level	Work
0	cn
1	4cn/2 = 2cn
2	16cn/4 = 4cn
3	
i	$4^i cn/2^i$ = $2^i cn$

More Examples



- **Stops when** $n/2^i = 1$**, and** $i = \log n$**.**
- **Recurrence solves to** $T(n) = O(n^2)$**.**

By Term Expansion

$$\begin{aligned} T(n) &= 4T(n/2) + cn \\ &= 4^2T(n/2^2) + 2cn + cn \\ &= 4^3T(n/2^3) + 2^2cn + 2cn + cn \\ &\vdots \\ &= 4^iT(n/2^i) + cn\left(2^{i-1} + 2^{i-2} + \ldots + 2 + 1\right) \\ &= 4^iT(n/2^i) + 2^icn \end{aligned}$$

- **Terminates when $2^i = n$, or $i = \log n$.**
- $4^i = 2^i \times 2^i = n \times n = n^2$.
- $T(n) = n^2 + cn^2 = O(n^2)$.

More Examples

$$T(n) = 2T(n/4) + \sqrt{n}, \qquad T(1) = 1.$$

$$\begin{aligned}
T(n) &= 2T(n/4) + \sqrt{n} \\
&= 2\left(2T(n/4^2) + \sqrt{n/4}\right) + \sqrt{n} \\
&= 2^2T(n/4^2) + 2\sqrt{n} \\
&= 2^2\left(2T(n/4^3) + \sqrt{n/4^2}\right) + 2\sqrt{n} \\
&= 2^3T(n/4^3) + 3\sqrt{n} \\
&\vdots \\
&= 2^iT(n/4^i) + i\sqrt{n}
\end{aligned}$$

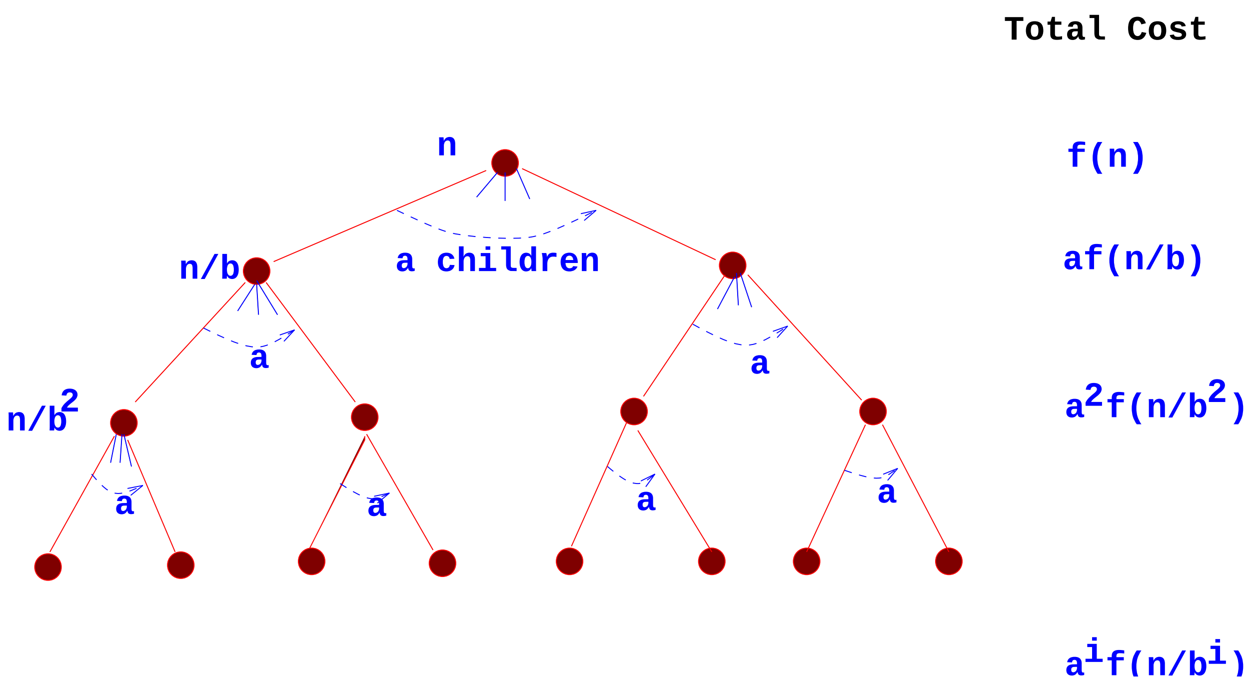
More Examples

- **Terminates when $4^i = n$, or when** $i = \log_4 n = \frac{\log_2 n}{\log_2 4} = \frac{1}{2}\log n$.

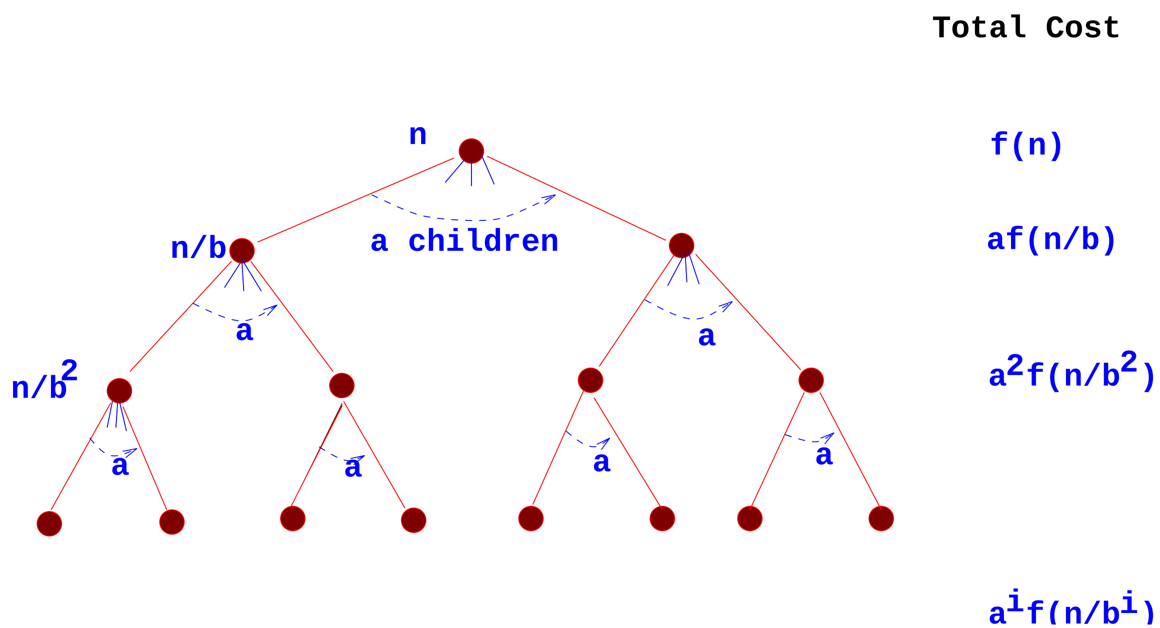
$$
\begin{aligned}
T(n) &= 2^{\frac{1}{2}\log n} + \sqrt{n}\log_4 n \\
&= \sqrt{n}(\log_4 n + 1) \\
&= O(\sqrt{n}\log n)
\end{aligned}
$$

Master Method

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



Master Method



- $\#$ **children multiply by factor** a **at each level.**
- **Number of leaves is** $a^{\log_b n} = n^{\log_b a}$**. Verify by taking logarithm on both sides.**

Master Method

- **By recursion tree, we get**

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

- **Let** $f(n) = \Theta(n^p \log^k n)$**, where** $p, k \geq 0$**.**
- **Important:** $a \geq 1$ **and** $b > 1$ **are constants.**

- **Case I:** $p < \log_b a$**.**

  $n^{\log_b a}$ **grows faster than** $f(n)$**.**

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

Master Method

- **By recursion tree, we get**

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

- **Let** $f(n) = \Theta(n^p \log^k n)$**, where** $p, k \geq 0$**.**

- **Case II:** $p = \log_b a$**.**

  Both terms have same growth rates.

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

Master Method

- **By recursion tree, we get**

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

- **Let** $f(n) = \Theta(n^p \log^k n)$**, where** $p, k \geq 0$**.**

- **Case III:** $p > \log_b a$**.**

  $n^{\log_b a}$ **is slower than** $f(n)$**.**

$$T(n) = \Theta\left(f(n)\right)$$

Applying Master Method

- **Merge Sort:** $T(n) = 2T(n/2) + \Theta(n)$.

  $a = b = 2$, $p = 1$, **and** $k = 0$. **So** $\log_b a = 1$, **and** $p = \log_b a$. **Case II applies, giving us**

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

- **Binary Search:** $T(n) = T(n/2) + \Theta(1)$.

  $a = 1, b = 2, p = 0$, **and** $k = 0$. **So** $\log_b a = 0$, **and** $p = \log_b a$. **Case II applies, giving us**

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

Applying Master Method

- $T(n) = 2T(n/2) + \Theta(n \log n)$.

  $a = b = 2$, $p = 1$, **and** $k = 1$. $p = 1 = \log_b a$, **and Case II applies.**

  $$T(n) = \Theta(n \log^2 n)$$

- $T(n) = 7T(n/2) + \Theta(n^2)$.

  $a = 7, b = 2$, $p = 2$, **and** $\log_b 2 = \log 7 > 2$. **Case I applied, and we get**

  $$T(n) = \Theta(n^{\log 7})$$

Applying Master Method

- $T(n) = 4T(n/2) + \Theta(n^2\sqrt{n})$.

  $a = 4, b = 2$, $p = 2.5$, **and** $k = 0$. **So** $\log_b a = 2$, **and** $p > \log_b a$. **Case III applies, giving us**

  $$T(n) = \Theta(n^2\sqrt{n})$$

- $T(n) = 2T(n/2) + \Theta\left(\frac{n}{\log n}\right)$.

  $a = 2, b = 2, p = 1$. **But** $k = -1$, **and so the Master Method does not apply!**

Matrix Multiplication

- **Multiply two $n \times n$ matrices: $C = A \times B$.**
- **Standard method: $C_{ij} = \sum_{k=1}^{n} A_{ik} \times B_{kj}$.**
- **This takes $O(n)$ time per element of C, for the total cost of $O(n^3)$ to compute C.**
- **This method, known since Gauss's time, seems hard to improve.**
- **A very surprising discovery by Strassen (1969) broke the n^3 asymptotic barrier.**
- **Method is divide and conquer, with a clever choice of submatrices to multiply.**

Divide and Conquer

- **Let A, B be two $n \times n$ matrices. We want to compute the $n \times n$ matrix $C = AB$.**

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

- **Entries a_{11} are $n/2 \times n/2$ submatrices.**

Divide and Conquer

- **The product matrix can be written as:**

$$
\begin{aligned}
c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\
c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\
c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\
c_{22} &= a_{21}b_{12} + a_{22}b_{22}
\end{aligned}
$$

- **Recurrence for this D&C algorithm is** $T(n) = 8T(n/2) + O(n^2)$**.**
- **But this solves to** $T(n) = O(n^3)$**!**

Strassen's Algorithm

- **Strassen chose these submatrices to multiply:**

$$
\begin{aligned}
P_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) \\
P_2 &= (a_{21} + a_{22})b_{11} \\
P_3 &= a_{11}(b_{12} - b_{22}) \\
P_4 &= a_{22}(b_{21} - b_{11}) \\
P_5 &= (a_{11} + a_{12})b_{22} \\
P_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\
P_7 &= (a_{12} - a_{22})(b_{21} + b_{22})
\end{aligned}
$$

Strassen's Algorithm

- **Then,**

$$\begin{aligned} c_{11} &= P_1 + P_4 - P_5 + P_7 \\ c_{12} &= P_3 + P_5 \\ c_{21} &= P_2 + P_4 \\ c_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned}$$

- **Recurrence for this algorithm is** $T(n) = 7T(n/2) + O(n^2)$**.**

Strassen's Algorithm

- **The recurrence $T(n) = 7T(n/2) + O(n^2)$.**

  solves to $T(n) = O(n^{\log_2 7}) = O(n^{2.81})$.

- **Ever since other researchers have tried other products to beat this bound.**

- **E.g. Victor Pan discovered a way to multiply two 70×70 matrices using $143,640$ multiplications.**

- **Using more advanced methods, the current best algorithm for multiplying two $n \times n$ matrices runs in roughly $O(n^{2.376})$ time.**

Quick Sort Algorithm

- **Simple, fast, widely used in practice.**
- **Can be done "in place;" no extra space.**
- **General Form:**

1. **Partition: Divide into two subarrays, L and R; elements in L are all smaller than those in R.**
2. **Recurse: Sort L and R recursively.**
3. **Combine: Append R to the end of L.**

- **Partition (A, p, q, i) partitions A with pivot $A[i]$.**

Partition

- **Partition returns the index of the cell containing the pivot in the reorganized array.**

11	**4**	**9**	**7**	**3**	**10**	**2**	**6**	**13**	**21**	**8**

- **Example: Partition** $(A, 0, 10, 3)$**.**
- $4, 3, 2, 6, 7, 11, 9, 10, 13, 21, 8$

Quick Sort Algorithm

- **QuickSort** (A, p, q) **sorts the subarray** $A[p \cdots q]$**.**
- **Initial call with** $p = 0$ **and** $q = n - 1$**.**

QuickSort(A, p, q)

    if $p \geq q$ **then return**
    $i \leftarrow$ **random**(p, q)
    $r \leftarrow$ **Partition**(A, p, q, i)
    Quicksort $(A, p, r - 1)$
    Quicksort $(A, r + 1, q)$

Analysis of QuickSort

- **Lucky Case: Each Partition splits array in halves. We get $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$.**
- **Unlucky Case: Each partition gives unbalanced split. We get $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$.**
- **In worst case, Quick Sort as bad as BubbleSort. The worst-case occurs when the list is already sorted, and the last element chosen as pivot.**
- **But, while BubbleSort always performs poorly on certain inputs, because of random pivot, QuickSort has a chance of doing much better.**

Analyzing QuickSort

- $T(n)$: runtime of randomized QuickSort.
- Assume all elements are distinct.
- Recurrence for $T(n)$ depends on two subproblem sizes, which depend on random partition element.
- If pivot is i smallest element, then exactly $(i-1)$ items in L and $(n-i)$ in R. Call it an i-split.
- What's the probability of i-split?
- Each element equally likely to be chosen as pivot, so the answer is $\frac{1}{n}$.

Solving the Recurrence

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{n} \frac{1}{n}(\textbf{runtime with } i\textbf{-split}) + n + 1 \\
&= \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) \; + \; n + 1 \\
&= \frac{2}{n}\sum_{i=1}^{n} T(i-1) \; + \; n + 1 \\
&= \frac{2}{n}\sum_{i=0}^{n-1} T(i) \; + \; n + 1
\end{aligned}
$$

Solving the Recurrence

- **Multiply both sides by n. Subtract the same formula for $n-1$.**

$$nT(n) = 2\sum_{i=0}^{n-1} T(i) + n^2 + n$$

$$(n-1)T(n-1) = 2\sum_{i=0}^{n-2} T(i) + (n-1)^2 + (n-1)$$

Solving the Recurrence

$$
\begin{aligned}
nT(n) &= (n+1)T(n-1) + 2n \\
\frac{T(n)}{n+1} &= \frac{T(n-1)}{n} + \frac{2}{n+1} \\
&= \frac{T(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\
&\vdots \\
&= \frac{T(2)}{3} + \sum_{i=3}^{n} \frac{2}{i} \\
&= \Theta(1) + 2\ln n
\end{aligned}
$$

- **Thus,** $T(n) \leq 2(n+1)\ln n$**.**

Median Finding

- **Median of n items is the item with rank $n/2$.**
- **Rank of an item is its position in the list if the items were sorted in ascending order.**
- **Rank i item also called ith statistic.**
- **Example: $\{16, 5, 30, 8, 55\}$.**
- **Popular statistics are quantiles: items of rank $n/4,\ n/2,\ 3n/4$.**
- **SAT/GRE: which score value forms 95th percentile? Item of rank $0.95n$.**

Median Finding

- **After spending $O(n \log n)$ time on sorting, any rank can be found in $O(n)$ time.**
- **Can we find a rank without sorting?**

Min and Max Finding

- **We can find items of rank 1 or n in $O(n)$ time.**

MINIMUM (A)

> min $\leftarrow A[0]$
> **for** $i = 1$ **to** $n - 1$ **do**
>   **if** min $> A[i]$ **then** min $\leftarrow A[i]$**;**
> **return** min

- **The algorithm** MINIMUM **finds the smallest (rank 1) item in $O(n)$ time.**
- **A similar algorithm finds maximum item.**

Both Min and Max

- **Find both min and max using $3n/2$ comparisons.**

MIN-MAX (A)

if $|A| = 1$**, then return** $\min = \max = A[0]$
Divide A **into two equal subsets** A_1, A_2
$(\min_1, \max_1) :=$ **MIN-MAX** (A_1)
$(\min_2, \max_2) :=$ **MIN-MAX** (A_2)
if $\min_1 \leq \min_2$ **then return** $\min = \min_1$
else return $\min = \min_2$
if $\max_1 \geq \max_2$ **then return** $\max = \max_1$
else return $\max = \max_2$

Both Min and Max

- **The recurrence for this algorithm is $T(n) = 2T(n/2) + 2$.**
- **Verify this solves to $T(n) = 3n/2 - 2$.**

Finding Item of Rank k

- **Direct extension of min/max finding to rank k item will take $\Theta(kn)$ time.**
- **In particular, finding the median will take $\Omega(n^2)$ time, which is worse than sorting.**
- **Median can be used as a perfect pivot for (deterministic) quick sort.**
- **But only if found faster than sorting itself.**
- **We present a linear time algorithm for selecting rank k item [BFPRT 1973].**

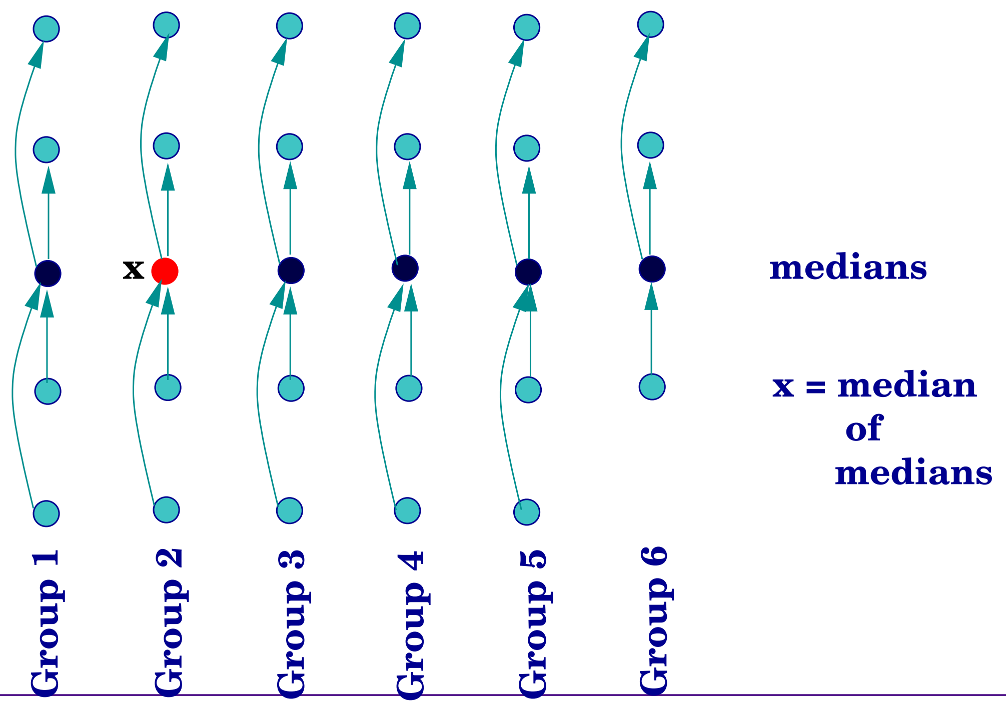
Linear Time Selection

SELECT (k)

1. **Divide items into $\lfloor n/5 \rfloor$ groups of 5 each.**
2. **Find the median of each group (using sorting).**
3. **Recursively find median of $\lfloor n/5 \rfloor$ group medians.**
4. **Partition using median-of-median as pivot.**
5. **Let low side have s, and high side have $n - s$ items.**
6. **If $k \leq s$, call** SELECT(k) **on low side; otherwise, call** SELECT$(k - s)$ **on high side.**

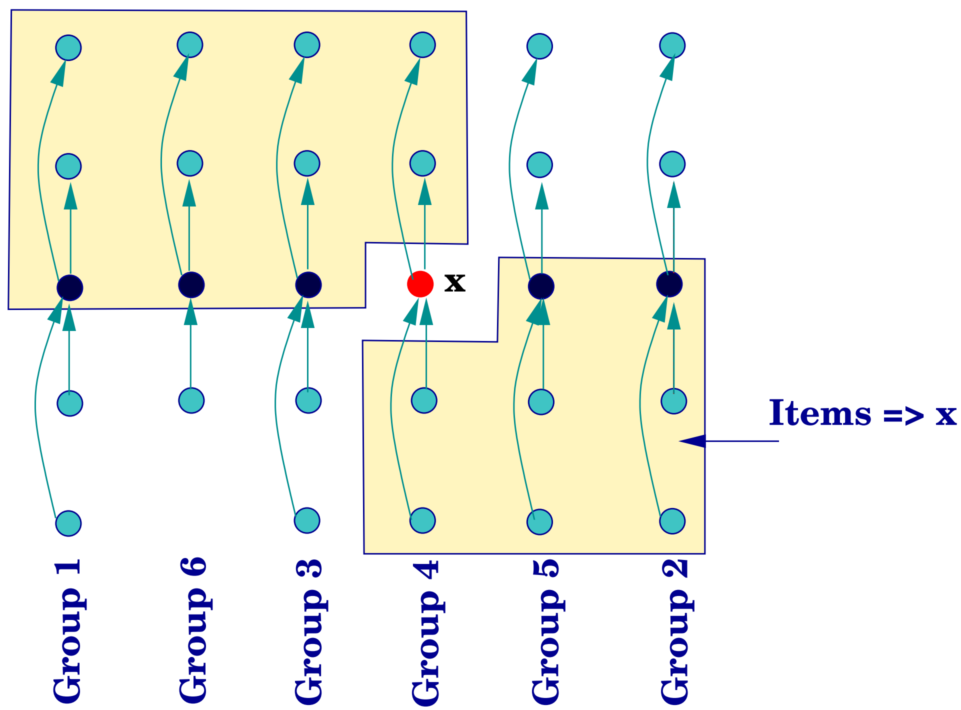
Illustration

- **Divide items into $\lfloor n/5 \rfloor$ groups of 5 items each.**
- **Find the median of each group (using sorting).**
- **Use SELECT to recursively find the median of the $\lfloor n/5 \rfloor$ group medians.**



Illustration

- Partition the input by using this median-of-median as pivot.
- Suppose low side of the partition has s elements, and high side has $n - s$ elements.
- If $k \leq s$, recursively call SELECT(k) on low side; otherwise, recursively call SELECT$(k - s)$ on high side.



Recurrence

- **For runtime analysis, we bound the number of items $\geq x$, the median of medians.**
- **At least half the medians are $\geq x$.**
- **At least half of the $\lfloor n/5 \rfloor$ groups contribute at least 3 items to the high side. (Only the last group can contribute fewer.**
- **Thus, items $\geq x$ are at least**

$$3\left(\frac{n}{10} - 2\right) \geq \frac{3n}{10} - 6.$$

- **Similarly, items $\leq x$ is also $3n/10 - 6$.**

Recurrence

- **Recursive call to SELECT is on size** $\leq 7n/10 + 6$**.**
- **Let** $T(n) =$ **worst-case complexity of SELECT.**
- **Group medians, and partition take** $O(n)$ **time.**
- **Step 3 has a recursive call** $T(n/5)$**, and Step 5 has a recursive call** $T(7n/10 + 6)$**.**
- **Thus, we have the recurrence:**

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7n}{10} + 6) + O(n).$$

- **Assume** $T(n) = O(1)$ **for small** $n \leq 80$**.**

Recurrence

$$T(n) \quad \leq \quad T(\frac{n}{5}) + T(\frac{7n}{10} + 6) + O(n)$$

- **Inductively verify that $T(n) \leq cn$ for some constant c.**

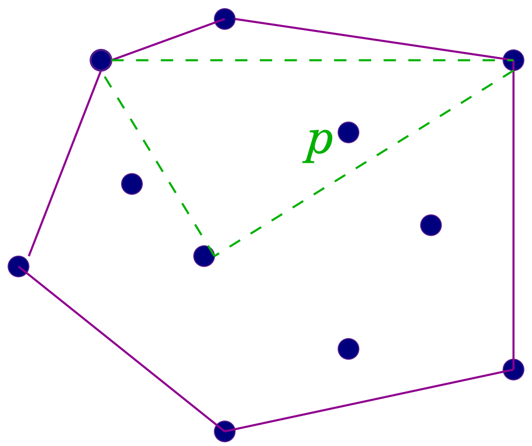
$$\begin{aligned} T(n) &\leq c(n/5) + c(7n/10 + 6) + O(n) \\ &\leq 9cn/10 + 6c + O(n) \\ &\leq cn \end{aligned}$$

- **In above, choose c so that $c(n/10 - 6)$ beats the function $O(n)$ for all n.**

Convex Hulls

1. **Convex hulls** are to CG what sorting is to discrete algorithms.

2. First order shape approximation. Invariant under rotation and translation.

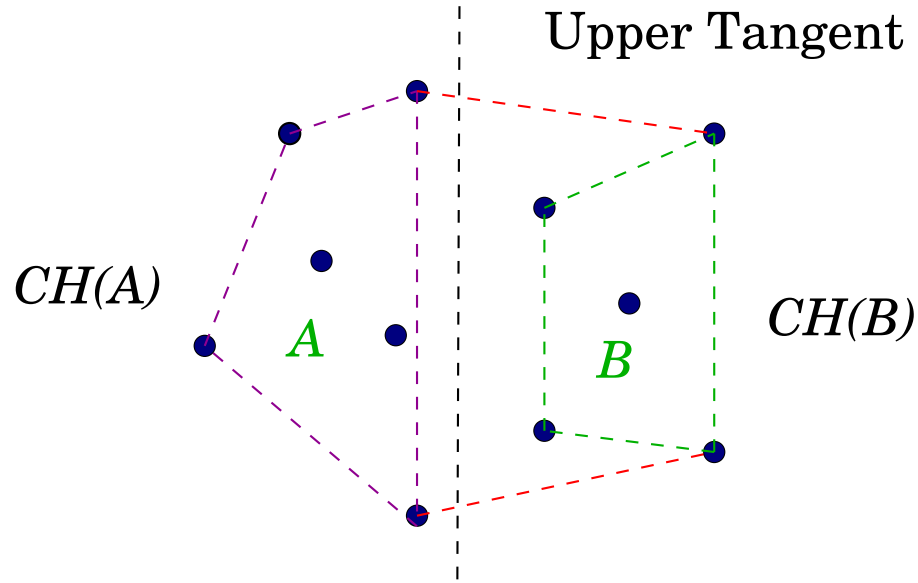


3. Rubber-band analogy.

Convex Hulls

- **Many aplications in robotics, shape analysis, line fitting etc.**
- **Example: if $CH(P_1) \cap CH(P_2) = \emptyset$, then objects P_1 and P_2 do not intersect.**
- **Convex Hull Problem:**
  Given a finite set of points S, compute its convex hull $CH(S)$. (Ordered vertex list.)

Divide and Conquer



- **Sort points by X-coordinates.**
- **Divide points into equal halves A and B.**
- **Recursively compute $CH(A)$ and $CH(B)$.**
- **Merge $CH(A)$ and $CH(B)$ to obtain $CH(S)$.**

Merging Convex Hulls

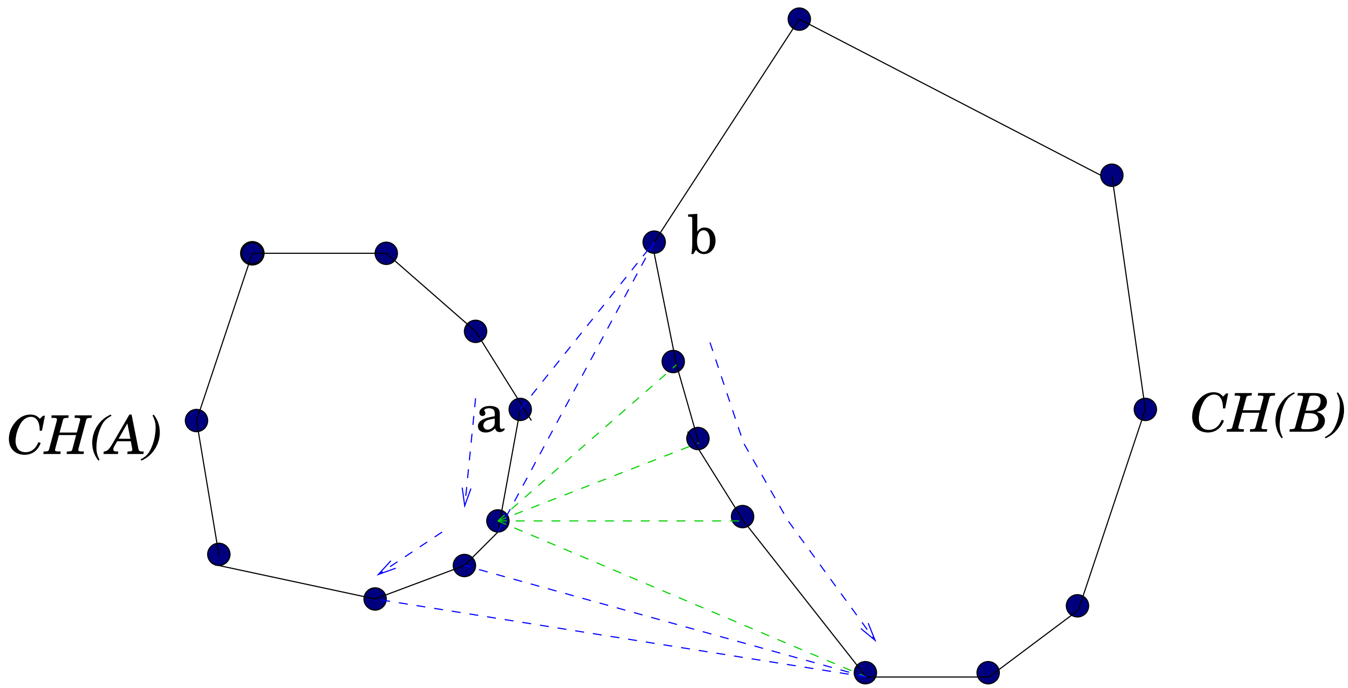
Lower Tangent

- $a =$ **rightmost point of** $CH(A)$.
- $b =$ **leftmost point of** $CH(B)$.
- **while** ab **not lower tangent of** $CH(A)$ **and** $CH(B)$ **do**

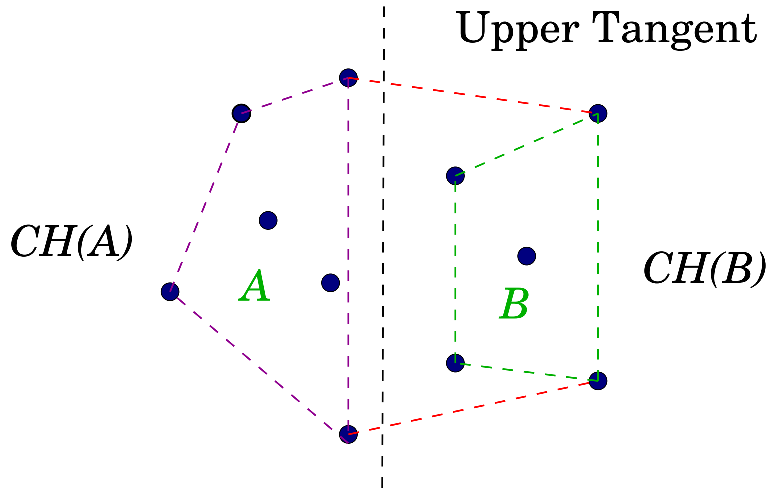
  1. **while** ab **not lower tangent to** $CH(A)$
     set $a = a - 1$ **(move** a **CW);**
  2. **while** ab **not lower tangent to** $CH(B)$
     set $b = b + 1$ **(move** b **CCW);**

- **Return** ab

Tangent Finding



Analysis of D&C



- **Initial sorting takes $O(N \log N)$ time.**
- **Recurrence $T(N) = 2T(N/2) + O(N)$**
- **$O(N)$ for merging (computing tangents).**
- **Recurrence solves to $T(N) = O(N \log N)$.**