Divide and Conquer

▷ Divide and Conquer (DaC):

(1) Divide input instance into many smaller instances

(2) Conquer smaller inst. recursively → usually trivial.

(3) Combine the sols of smaller inst. to get a sol of the original input inst

▷ Compared with Greedy algs. DaC has the following difference.

- Alg correctness is usually easy to see.
- Main focus: running time.

(I) Exmp: MergeSort

Input: seq of n numbers $A=[a_1, a_2, \cdots, a_n]$

Output: a permutation $a_1', a_2', \cdots, a_n'$ s.t.

$$a_1' \le a_2' \le \cdots \le a_n'$$

Bubble Sort

$O(n^2)$

- Consider a simpler prob:
  - Suppose $A[1, \cdots, n/2]$ and $A[\frac{n}{2}+1, \cdots, n]$ already sorted.
  - How would you sort A?

Exmp: $A=[\underbrace{4\ 7\ 9\ 12}_{\uparrow}, \underbrace{1\ 3\ 17\ 25}_{\uparrow}]$

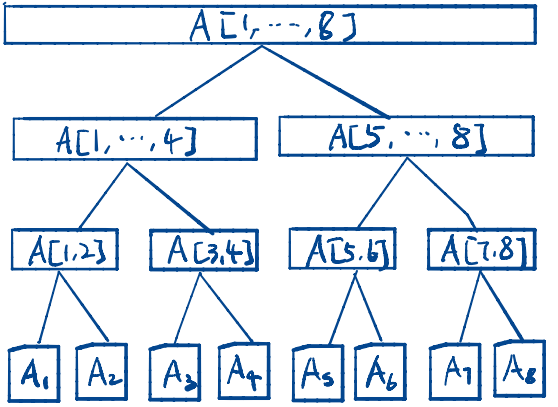
Merge: $A'=[1\ 3\ 4\ 7\ 9\ 12\ 17\ 25]$ $\Rightarrow O(n)$ time.

```
Alg-MergeSort(A, n)
    if n = 1 :
        return A
    else
        B ← MergeSort(A[1, ···, ⌊n/2⌋], ⌊n/2⌋)
        C ← MergeSort(A[⌊n/2⌋+1, ···, n], ⌈n/2⌉)
    return Merge(B, C)
```

divide

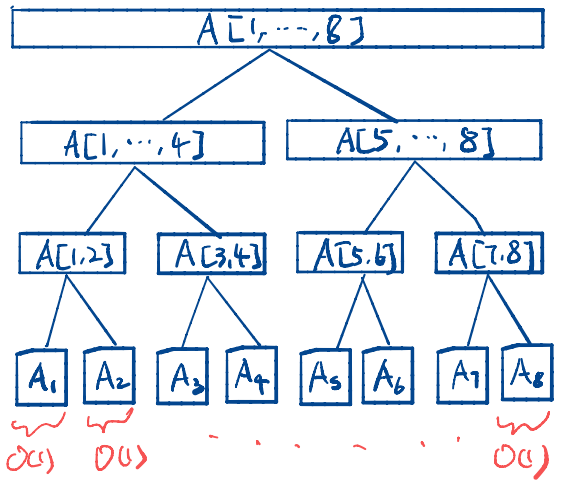
divide.

combine



- "Cheat" in the sense that we can assume the sol of the smaller instances is given
  - Achieved by recursion.

▷ Running Time:



$O(n)$ time.

$O(n)$ time

$\frac{n}{2} \cdot 2 \cdot O(1) = O(n)$ time

$O(n)$ time.

$O(n) \times \underbrace{\#\text{ of levels}}_{\log_2 n} = O(n \log n)$

$$T(n) = 2T(n/2) + O(n) = 2 \cdot \left(2T(n/4) + O\left(\frac{n}{2}\right)\right) + O(n)$$

$$= 4T(n/4) + O(n) + O(n) = \cdots = (\log_2 n)\, T(1) + \underbrace{O(n) + \cdots + O(n)}_{\log_2 n}$$

$$= O(n \log n)$$

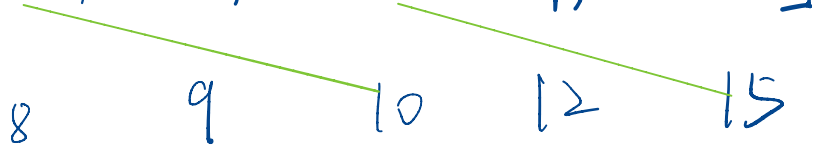
▷ Exmp (II): Counting Inversions

Def (Inversion): Given array A of n numbers, an inversion is a pair (i, j) of indices s.t. $i < j$ but $A[i] > A[j]$

Input: Array A of n numbers

Output: # of inv in A

Exmp: $A = [10, 8, 15, 9, 12]$



Invs: $(10, 8), (10, 9), (15, 9), (15, 12)$

▷ DaC Alg for Count Inv.

$$\#inv(A) = \#inv(B) + \#inv(C) + m$$

$$m = \left| \{ (i,j) : B[i] > C[j] \} \right|$$

Obs: If B and C are sorted, then we can get m in $O(n)$ time.

B: [3, 8, 12, 20, 32, 48] C: [5, 7, 9, 25, 29]

Merge(B,C): 3 5 7 8 9 12 20 25 29 32 48

(+2, +3, +3, +5, +5)

merge-and-count(B, C, n_1, n_2)

1. $count \leftarrow 0$;
2. $A \leftarrow []$; $i \leftarrow 1$; $j \leftarrow 1$
3. while $i \leq n_1$ or $j \leq n_2$
4. if $j > n_2$ or ($i \leq n_1$ and $B[i] \leq C[j]$) then
5. append $B[i]$ to A; $i \leftarrow i + 1$
6. $count \leftarrow count + (j - 1)$
7. else
8. append $C[j]$ to A; $j \leftarrow j + 1$
9. return $(A, count)$

// B. C are sorted.

recursive call ↑ merge-n-count ↗

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

$$= O(n \log n)$$

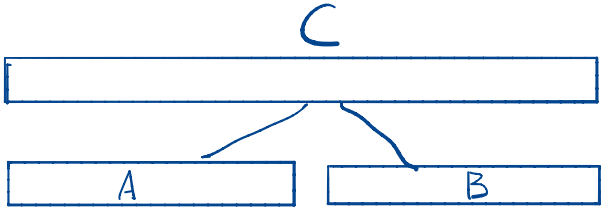
Alg Sort-and-Count(A, n)
If $n = 1$ return $(A, 0)$
$(B, m_1) \leftarrow$ Sort-and-Count$(A[1, \cdots, \lfloor \frac{n}{2} \rfloor], \lfloor n/2 \rfloor)$
$(C, m_2) \leftarrow$ Sort-and-Count$(A[\lfloor \frac{n}{2} \rfloor + 1, \cdots, n], \lceil n/2 \rceil)$
$(A, m) \leftarrow$ merge-and-count$(B, C, \lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil)$
return $(A, m_1 + m_2 + m)$

Exmp (II): Quick Sort

	Merge Sort	Quick Sort
Divide	Trivial	separate small & large numbers $O(n)$-time
Conquer	recursion	recursion
Combine	merge two sorted array $O(n)$ time	trivial

Assumption:

We can find the median of n numbers in $O(n)$ time.



$\forall a \in A < \forall b \in B$

smaller half larger half

quicksort(A, n)

1. if $n \leq 1$ then return A
2. $x \leftarrow$ lower median of A — $O(n)$-time
3. $A_L \leftarrow$ elements in A that are less than x \\ Divide
4. $A_R \leftarrow$ elements in A that are greater than x \\ Divide
5. $B_L \leftarrow$ quicksort$(A_L, A_L.\text{size})$ \\ Conquer
6. $B_R \leftarrow$ quicksort$(A_R, A_R.\text{size})$ \\ Conquer
7. $t \leftarrow$ number of times x appear A
8. return the array obtained by concatenating B_L, the array containing t copies of x, and B_R

- $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$

▷ Implementing QuickSort

- How to find the median in $O(n)$ time?
  - $\exists$ $O(n)$-time median-selection alg
  - very complicated & impractical.

- In practice:
  - randomly pick a number (p) from the array

  - $O(n \log n)$-time in expectation

▷ QuickSort: Separate large & small numbers in place

partition(A, ℓ, r)

1. $p \leftarrow$ random integer between ℓ and r, swap $A[p]$ and $A[\ell]$
2. $i \leftarrow \ell, j \leftarrow r$
3. while **true** do
4. while $i < j$ and $A[i] < A[j]$ do $j \leftarrow j - 1$
5. **if** $i = j$ **then** break
6. swap $A[i]$ and $A[j]$; $i \leftarrow i + 1$
7. while $i < j$ and $A[i] < A[j]$ do $i \leftarrow i + 1$
8. **if** $i = j$ **then** break
9. swap $A[i]$ and $A[j]$; $j \leftarrow j - 1$
10. return i

quicksort(A, ℓ, r)

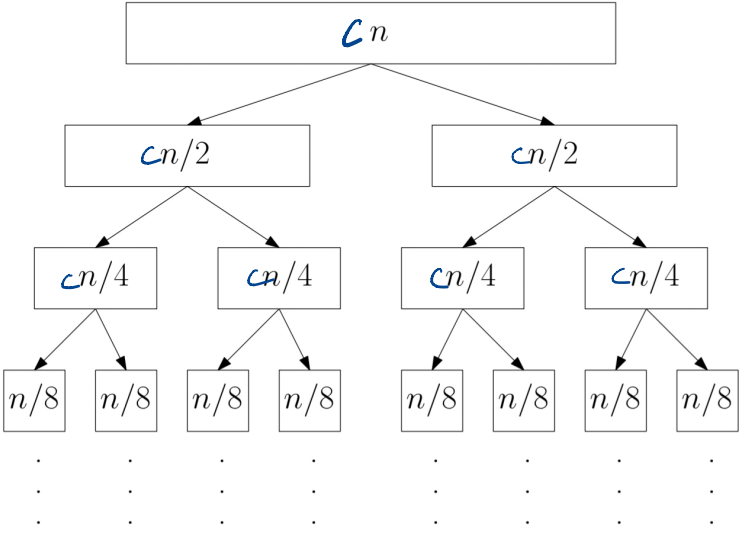
1. **if** $\ell \geq r$ **then return**
2. $m \leftarrow$ patition(A, ℓ, r)
3. quicksort$(A, \ell, m - 1)$
4. quicksort$(A, m + 1, r)$

- quicksort (A, 1, n)

(IV) Solving Recurrence

- Recursion-tree method.

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



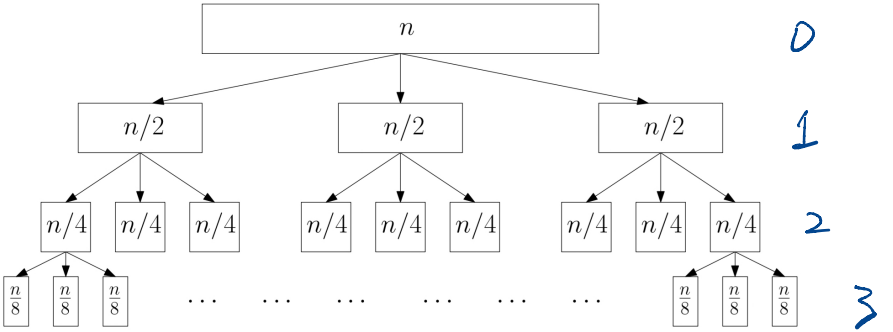
cn

Each level: $O(n)$ time

level: $O(\log n)$ levels

$\Rightarrow$ Running time $O(n \log n)$

- $T(n) = 3T(n/2) + O(n)$



- Time of each level ?

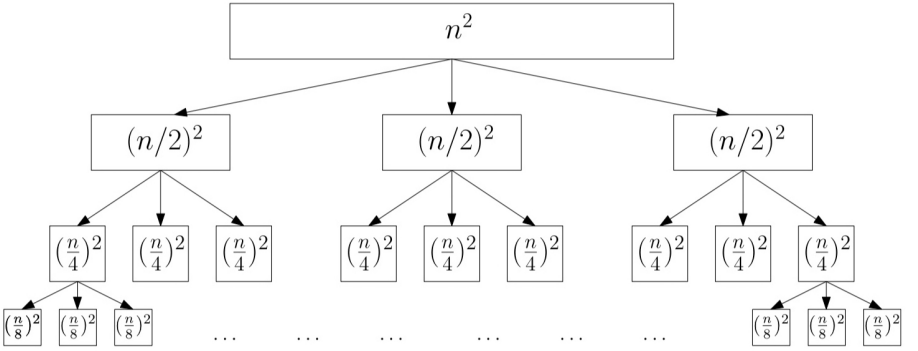
$$\frac{n}{2^i} \times 3^i = \left(\frac{3}{2}\right)^i n$$

- # levels ? $= \log_2 n = O(\log n)$

- Total running time $T(n) = \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i n = O\left(n \left(\frac{3}{2}\right)^{\log_2 n}\right)$

$$= O(3^{\log_2 n}) = O(n^{\log_2 3})$$

- $T(n) = 3T(n/2) + O(n^2)$



- Time on each level

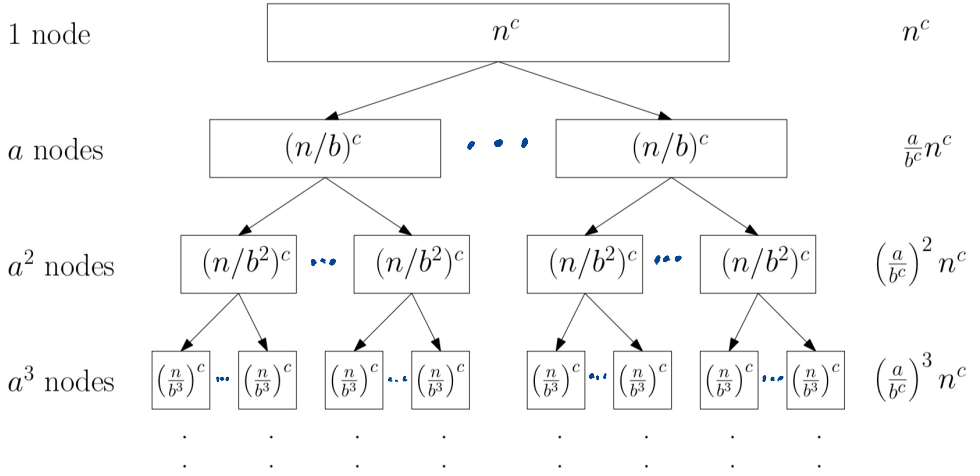
$$\left(\frac{n}{2^i}\right)^2 \times 3^i = \left(\frac{3}{4}\right)^i n^2$$

- #levels $= \log_2 n$
  $= O(\log n)$

- $$T(n) = \sum_{i=0}^{\log_2 n} \left(\frac{3}{4}\right)^i \cdot n^2 = n^2 \left(\frac{1 - \left(\frac{3}{4}\right)^{\log_2 n}}{1 - \frac{3}{4}}\right) = O(n^2)$$

▷ Master Thm

If $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1$, $b \geq 1$, $c \geq 0$



- # level $= \log_b n$
- level i takes time $\underbrace{\left(\frac{a}{b^c}\right)^i n^c}_{T_i}$

① c is large, $a < b^c$, i.e. $c > \log_b a$. $\alpha := \frac{a}{b^c}$

$\Rightarrow T_i = \alpha^i \cdot n^c$, $\alpha < 1$

$\Rightarrow T(n) = \sum_{i=0}^{\log_b n} T_i = n^c \sum_{i=0}^{\log_b n} \alpha^i = O(n^c)$

② $c = \log_b a$, i.e. $a = b^c$. $\Rightarrow T_i = n^c$

$\Rightarrow T(n) = n^c \cdot \#\text{levels} = O(n^c \log n)$

③ $a > b^c$, ie $c < \log_b a$. $\alpha > 1$.

$\Rightarrow T(n) = n^c \sum_{i=0}^{\log_b n} \alpha^i = O(n^c \cdot \alpha^{\log_b n})$

$= O\left(n^c \cdot \left(\frac{a}{b^c}\right)^{\log_b n}\right) = O\left(a^{\log_b n}\right) = O\left(n^{\log_b a}\right)$

Thm $T(n) = aT(n/b) + O(n^c)$, where $a \geq 1$, $b \geq 1$, $c \geq 0$ are constants. Then

Master Thm

$$T(n) = \begin{cases} O(n^{\log_b a}), & \text{if } c < \log_b a \\ O(n^c \log n), & \text{if } c = \log_b a \\ O(n^c), & \text{if } c > \log_b a \end{cases}$$

Exmp:

$T(n) = 4T(n/2) + O(n^2)$ $\quad a=4, b=2, c=2 \Rightarrow T(n) = O(n^2 \log n)$

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

$T(n) = T(n/2) + O(1) \Rightarrow T(n) = O(\log n)$

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

Exmp (IV) Polynomial Multiplication

> Input: two polyns of deg $n-1$
> Output: product of the two polyns

Eg: $n = 4$

$A(x) = 3x^3 + 2x^2 - 5x + 4$

$B(x) = 2x^3 - 3x^2 + 6x - 5$

$4 \cdot x^0 \quad -5 \cdot x^1 \quad 2x^2 \quad 3x^3$

$\Rightarrow$ Input: array $A = [4, -5, 2, 3]$

$B = [-5, 6, -3, 2]$

$C(x) = A(x) \cdot B(x)$

$= 6x^6 - 9x^5 + 18x^4 - 15x^3$
$+ 4x^5 - 6x^4 + 12x^3 - 10x^2$
$- 10x^4 + 15x^3 - 30x^2 + 25x$
$+ 8x^3 - 12x^2 + 24x - 20$

$= 6x^6 - 5x^5 + 2x^4 + 20x^3 - 52x^2 + 49x - 20$

$\Rightarrow$ Output: $C = [-20, 49, -52, 20, 2, -5, 6]$

polynomial-multiplication(A, B, n)

1. let $C[k] = 0$ for every $k = 0, 1, 2, \cdots, 2n-2$
2. for $i \leftarrow 0$ to $n-1$
3. for $j \leftarrow 0$ to $n-1$
4. $C[i+j] \leftarrow C[i+j] + A[i] \times B[j]$
5. return C

$O(n^2)$

- Divide: $A \Rightarrow [A_L, A_H]$, $B \Rightarrow [B_L, B_H]$ (Assume n is some power of 2)
- $P(x) = 3x^3 + 2x^2 - 5x + 4 = \underbrace{(3x+2)}_{P_H} x^2 + \underbrace{(-5x+4)}_{P_L}$
  $Q(x) = 2x^3 - 3x^2 + 6x - 5 = \underbrace{(2x-3)}_{Q_H} x^2 + \underbrace{(6x-5)}_{Q_L}$
- $P(x) = P_H(x)\, x^{n/2} + P_L(x)$

- $PQ = (P_H x^{n/2} + P_L)(Q_H x^{n/2} + Q_L)$

$= P_H Q_H x^n + (P_H Q_L + P_L Q_H) x^{n/2} + P_L Q_L$

$multp(P, Q) = multp(P_H, Q_H) \times x^n +$

$(multp(P_H, Q_L) + multp(P_L, Q_H)) \times x^{n/2} +$

$multp(P_L, Q_L)$

- $T(n) = 4T(n/2) + O(n)$ $\quad a = 4,\ b = 2,\ c = 1 \Rightarrow c < \log_b a = 2$

$\Rightarrow T(n) = O(n^{\log_b a}) = O(n^2)$

▷ Reduce the # of recursive calls

$$PQ = P_H Q_H x^n + (P_H Q_L + P_L Q_H) x^{n/2} + P_L Q_L$$

- Key observation:

$$P_H Q_L + P_L Q_H = (P_H + P_L)(Q_H + Q_L) - P_H Q_H - P_L Q_L$$

- $r_H = \text{multip}(P_H, Q_H)$, $r_L = \text{multip}(P_L, Q_L)$

$$\text{multip}(P, Q) = r_H x^n + (\text{multip}(P_H + P_L, Q_H + Q_L) - r_H - r_L) x^{n/2} + r_L$$

- $T(n) = 3T(n/2) + O(n) \Rightarrow T(n) = O(n^{\log_2 3}) \simeq O(n^{1.585})$

▷ Implementation.

- Assume: n is a power of 2.

multiply(A, B, n)

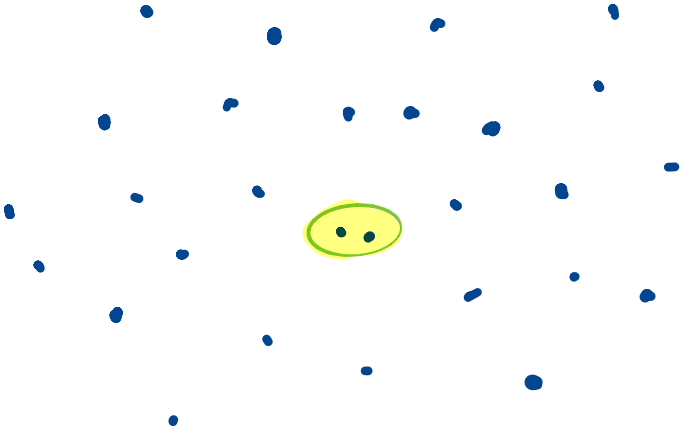
1. if $n = 1$ then return $(A[0]B[0])$
2. $A_L \leftarrow A[0 \,..\, n/2 - 1], A_H \leftarrow A[n/2 \,..\, n - 1]$
3. $B_L \leftarrow B[0 \,..\, n/2 - 1], B_H \leftarrow B[n/2 \,..\, n - 1]$
4. $C_L \leftarrow$ multiply$(A_L, B_L, n/2)$
5. $C_H \leftarrow$ multiply$(A_H, B_H, n/2)$
6. $C_M \leftarrow$ multiply$(A_L + A_H, B_L + B_H, n/2)$ $\quad (P_H + P_L)(Q_H + Q_L)$
7. $C \leftarrow$ array of $(2n - 1)$ 0's
8. for $i \leftarrow 0$ to $n - 2$ do
9. $\quad C[i] \leftarrow C[i] + C_L[i]$
10. $\quad C[i + n] \leftarrow C[i + n] + C_H[i]$
11. $\quad C[i + n/2] \leftarrow C[i + n/2] + C_M[i] - C_L[i] - C_H[i]$
12. return C

(V) Closest pair

Input: A seq of points on $\mathbb{R}^2$

$(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$

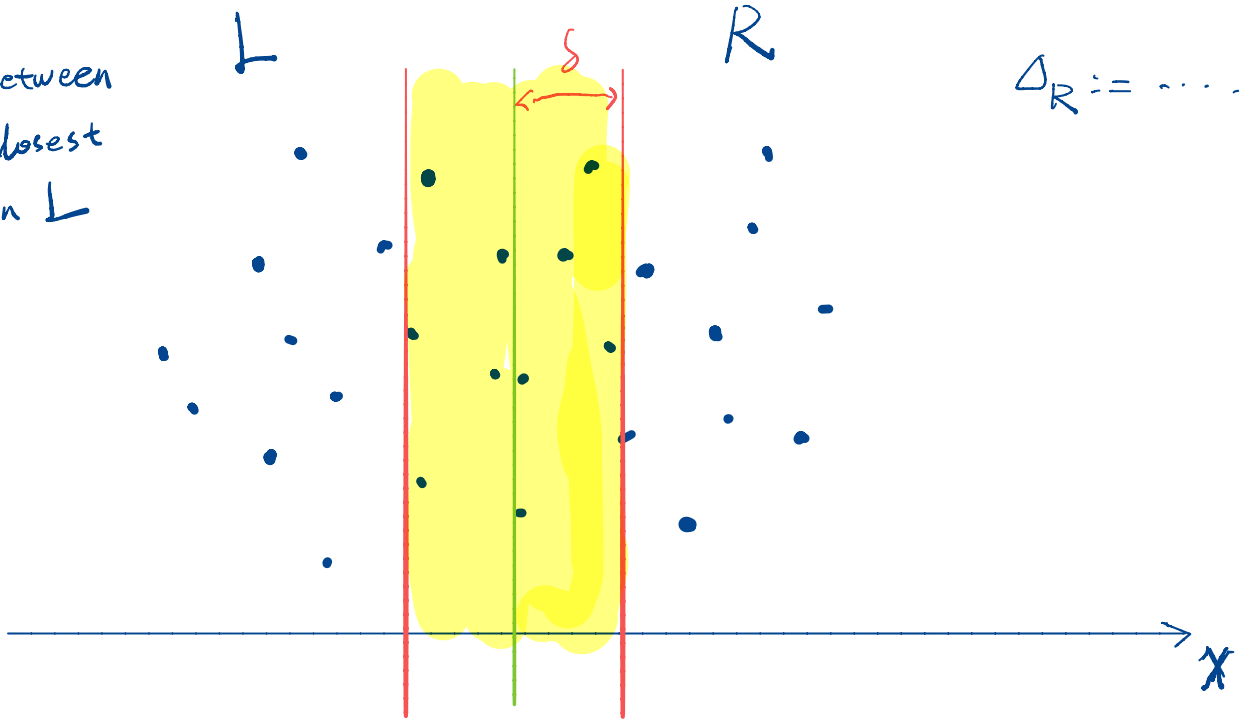
Output: The closest two points.



- Trivial to solve in $O(n^2)$ time.

▷▷ DaC Alg for Closest Pair.

Δ_L := dist between the two closest points in L



Δ_R := ....

$\delta = \min(\Delta_L, \Delta_R)$

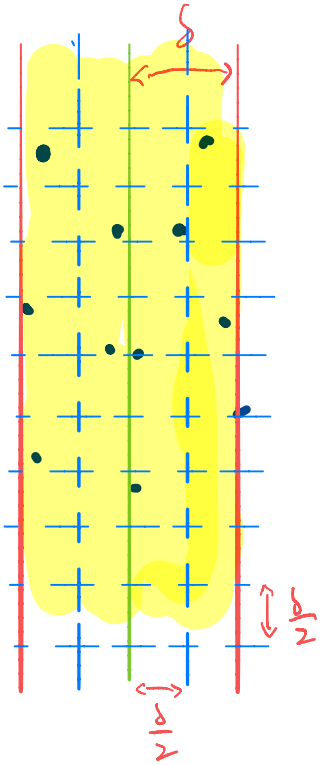
Finding the dividing line: $O(n)$

(via median selection)

▷ Find the closest pair crossing the dividing line.

- Divide the strip into $\frac{\delta}{2} \times \frac{\delta}{2}$ squares
  - Each square contain ≤ 1 point

- Checking for closest pairs
  - For each points, only need to check 15 squares around it



- Implementation
  - HashMap: $M\left[\left\lfloor \frac{x}{\delta/2}, \frac{y}{\delta/2} \right\rfloor\right] = (x, y)$

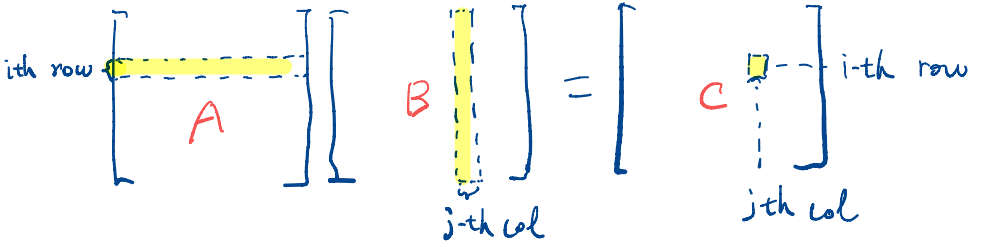
$\Rightarrow$ Running time: $T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$

(VI) Matrix Multiplication

Input: Two $n \times n$ matrices A & B

Output: $C = AB$

Def: $C = AB$ iff $C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$



Exmp:

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 0 \\ 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 3 \\ 5 & 4 & 0 \\ 2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 25 & 18 & 13 \\ 20 & 18 & 6 \\ 2 & 4 & 11 \end{bmatrix}$$

Trivial alg:

$O(n^3)$ time

▷ DaC Alg for Matrix Multip

$$A = \begin{array}{|c|c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array} \qquad B = \begin{array}{|c|c|}\hline B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline \end{array}$$

$$\Rightarrow \quad C = \begin{array}{|c|c|}\hline C_{11} & C_{12} \\ \hline C_{13} & C_{14} \\ \hline \end{array} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

- $T(n) = 8T(n/2) + O(n^2)$

  $\Rightarrow T(n) = O(n^3)$

running time:

$\Omega(n^2)$

▷ Strassen's Alg

$$C = \begin{array}{|c|c|} \hline C_{11} & C_{12} \\ \hline C_{21} & C_{22} \\ \hline \end{array}$$

- $$\begin{cases} M_1 = (A_{11} + A_{22})(B_{11} + B_{22}) \\ M_2 = (A_{21} + A_{22})\, B_{11} \\ M_3 = A_{11}(B_{12} - B_{22}) \\ M_4 = A_{22}(B_{21} - B_{11}) \\ M_5 = (A_{11} + A_{12})\, B_{22} \\ M_6 = (A_{21} - A_{11})(B_{11} + B_{12}) \\ M_7 = (A_{12} - A_{22})(B_{21} + B_{22}) \end{cases}$$

$$\Rightarrow \begin{cases} C_{11} = M_1 + M_4 - M_5 + M_7 \\ C_{12} = M_3 + M_5 \\ C_{21} = M_2 + M_4 \\ C_{22} = M_1 - M_2 + M_3 + M_6 \end{cases}$$

$$T(n) = 7\,T(n/2) + O(n^2) \Rightarrow T(n) = O(n^{\log_2 7}) = O(n^{2.808})$$