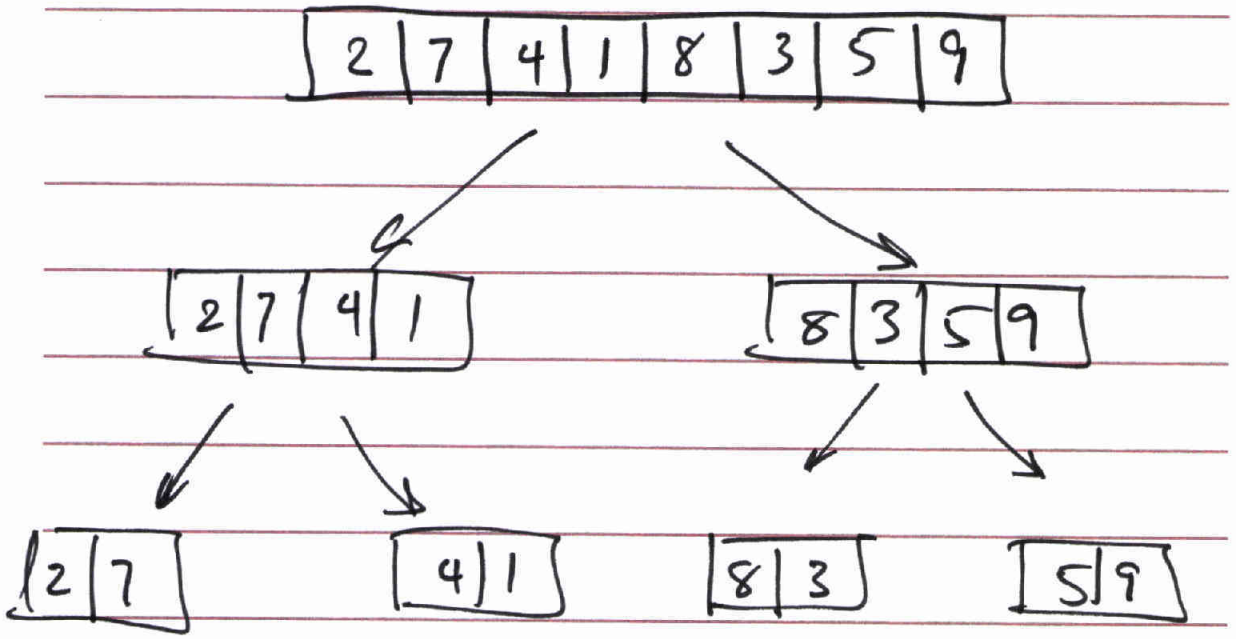
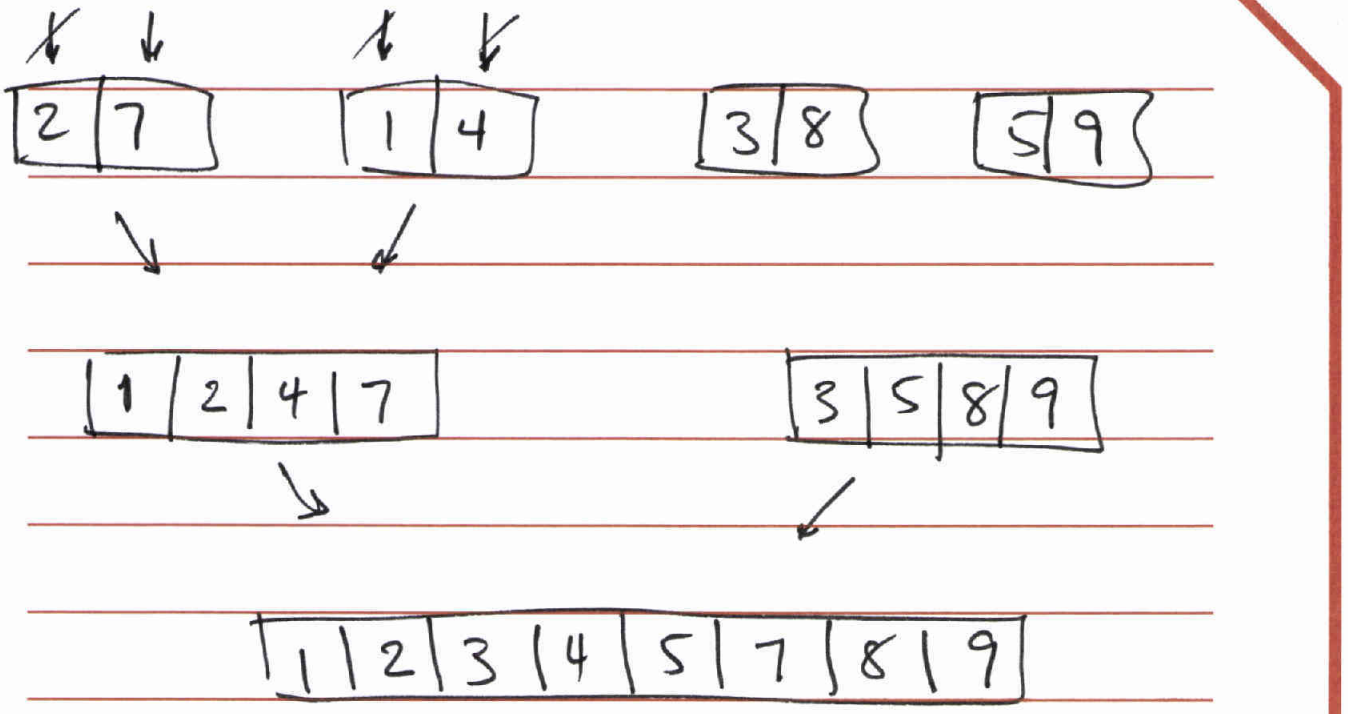
Divide & Conquer

1- Divide problem into n subproblems

2- Conquer: i.e. solve the subproblems recursively, or if trivial solve the problem itself

3- Combine the solution to the subproblems





MERGE-SORT (A, p, r)

```
MERGE-SORT(A, p, r)
    if p < r then
        q = ⌊(p+r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q+1, r)
        MERGE(A, p, q, r)
    end if
```

Analyzing Merge-Sort

Divide - Takes $O(1)$

Conquer - if the original problem takes $T(n)$ time, the 2 subproblems takes $2 \cdot T(n/2)$

Combine - Takes $O(n)$ on array size of n.

Recurrence equation for merge sort

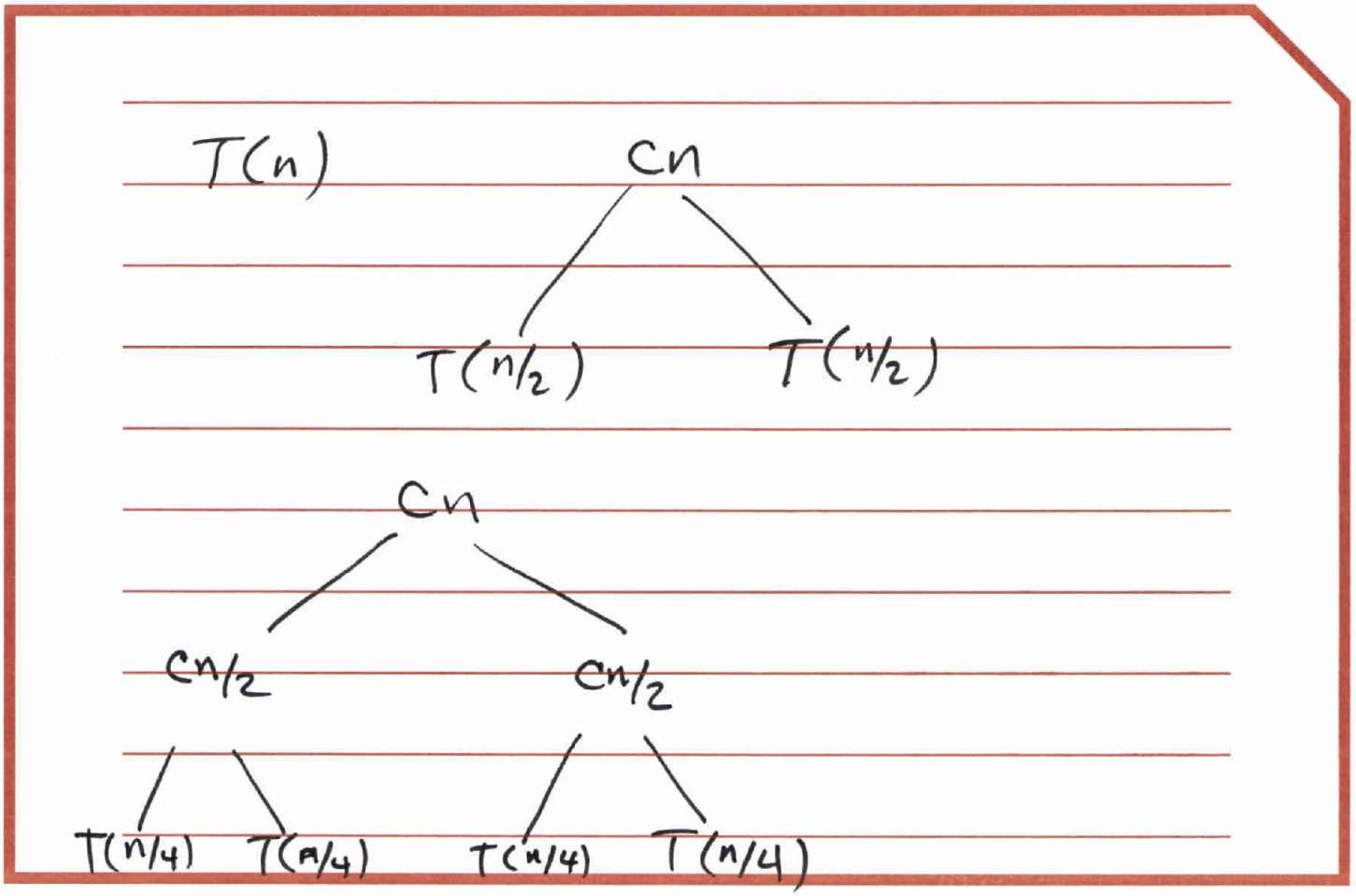
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2 \cdot T(n/2) + \Theta(n) + \Theta(1) & \end{cases}$$

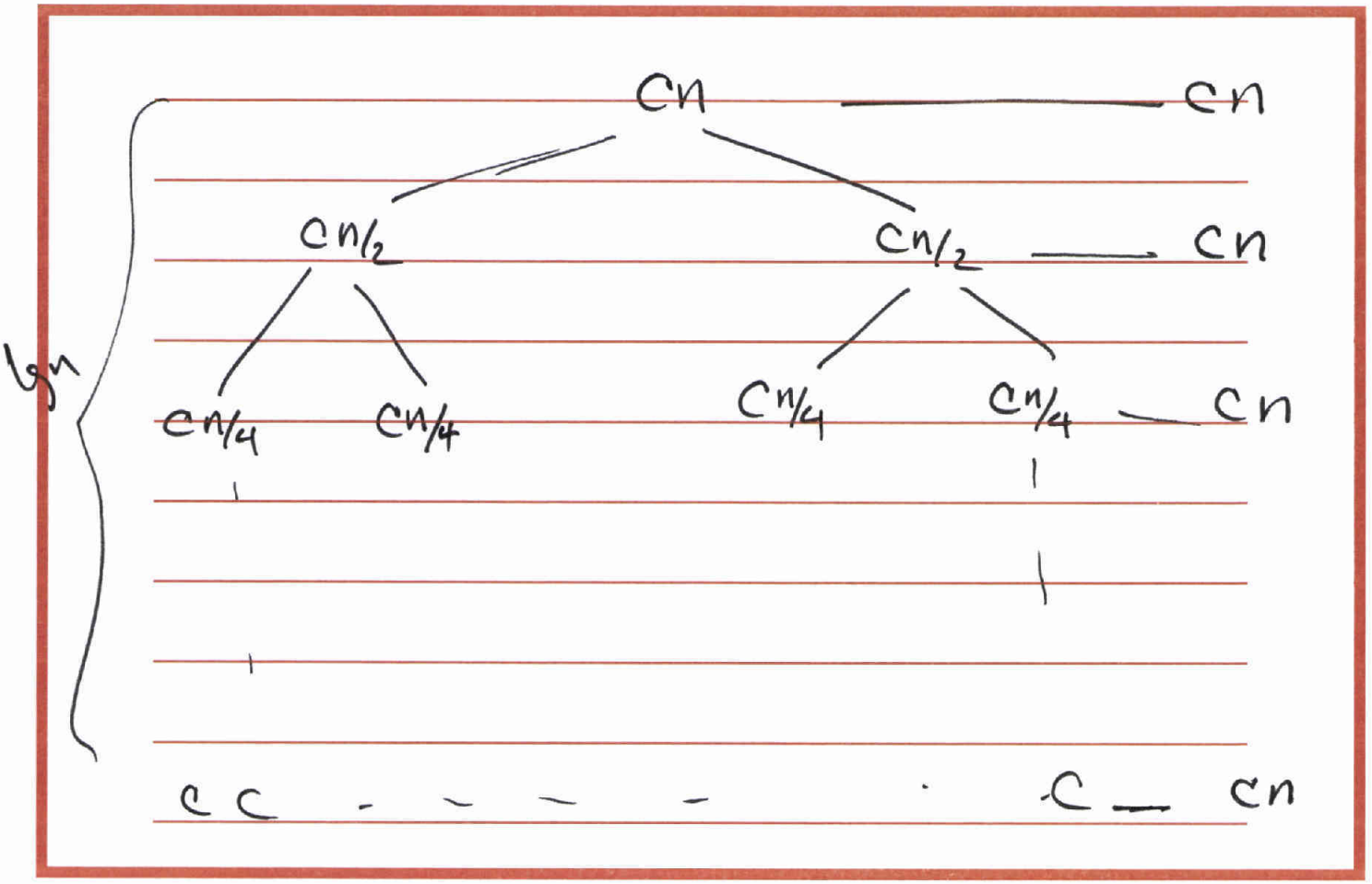
Conquer — Combine — Divide

in general, our recurrence equation for a D&C solution will look like:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \end{cases}$$

- a: no of subproblems
- n/b: size of the subproblems
- $D(n)$: Divide
- $C(n)$: Combine


T(n)
cn
T(n/2)
T(n/2)
cn
cn/2
cn/2
T(n/4)
T(n/4)
T(n/4)
T(n/4)


cn
cn
cn/2
cn/2
cn
cn/4
cn/4
cn/4
cn/4
cn
log n
cn

Overall Complexity of Merge_sort:

$O(n \log n)$

Master Method

It is a cookbook method for solving recurrences of the form

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

- where $a \geq 1$, $b \geq 1$ are constants

- $f(n)$ is an asymptotically positive function.

Master Theorem

- Given the above definition of the recurrence relation, $T(n)$ can be bounded asymptotically as follows:

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

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

2- If $f(n) = \Theta(n^{\log_b a})$ then

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

3- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a f(n/b) \leq c f(n)$ for some constant $c < 1$ and all sufficiently large n, then

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

$$n^{\log_b a} \; ? \; f(n)$$

$$n^{\log_b a} = n \quad , \quad f(n) = n \lg n$$

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

in general if $f(n) = \Theta(n^{\log_b a} \lg^k n)$
where $k \geq 0$

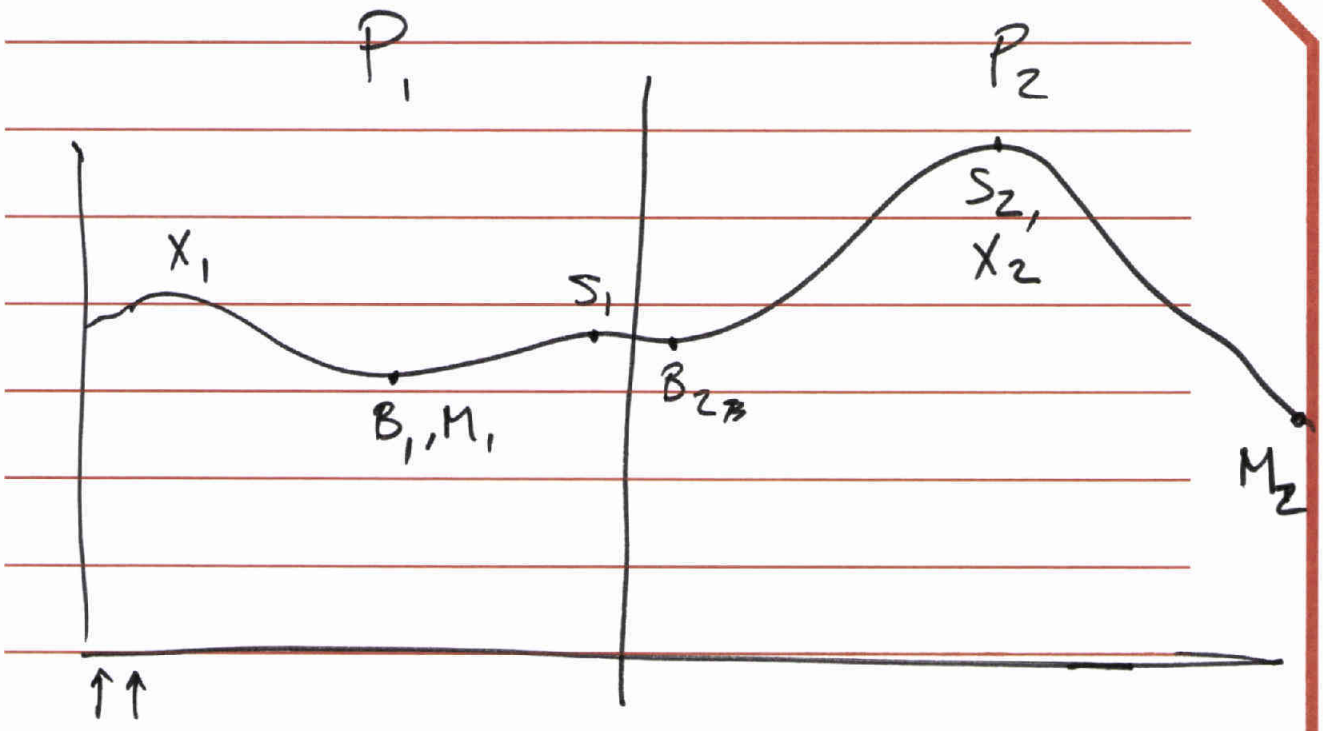
Then

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

$$n^{\log_b a} = n^2 \quad , \quad f(n) = n^2 \lg^3 n$$

$$T(n) = n^2 \lg^4 n$$

Stock Market Problem



$(n-1) + (n-2) + (n-3) + \dots + 1$

$O(n^2)$

Case 1 – buy & sell in P_1

$B = B_1$ $\quad M = Min(M_1, M_2)$

$S = S_1$ $\quad X = Max(X_1, X_2)$

Case 2 – buy & sell in P_2

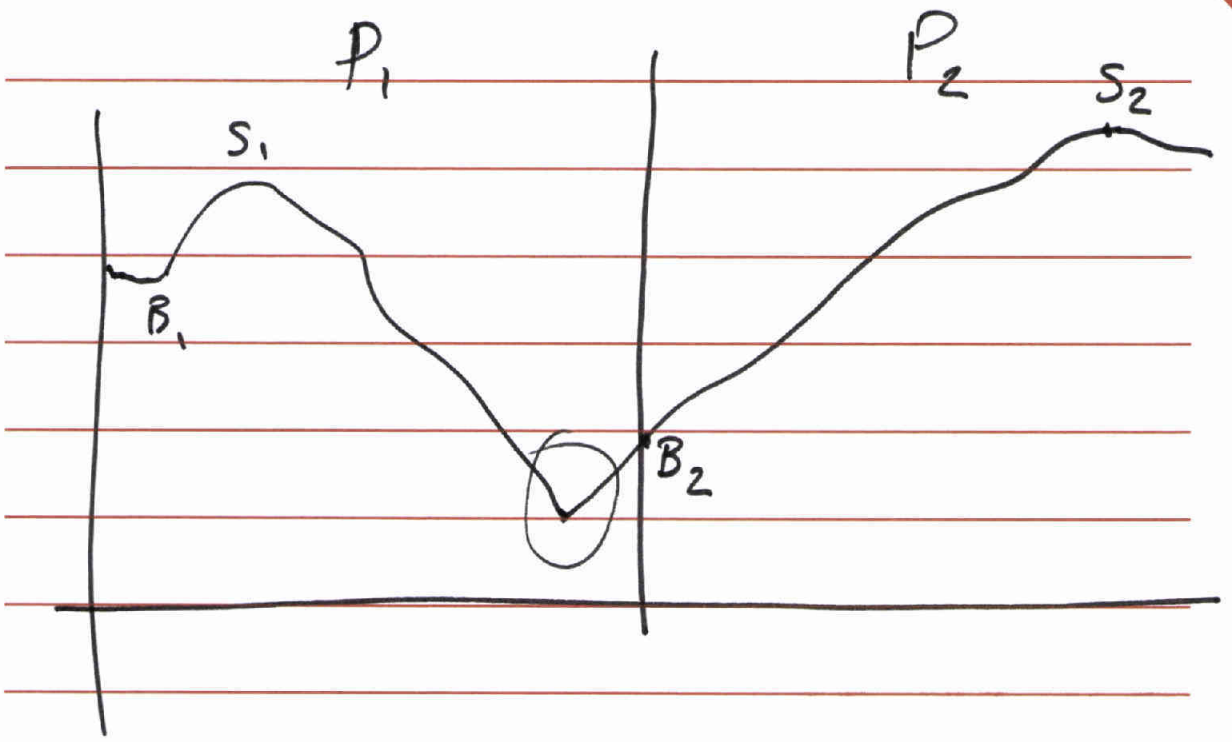
$B = B_2$ $\quad M = Min(M_1, M_2)$

$S = S_2$ $\quad X = Max(X_1, X_2)$

Case 3 – buy in P_1 & Sell in P_2

$B = B_1, M_1$ $\quad M = Min(M_1, M_2)$

$S = S_2 X_2$ $\quad X = Max(X_1, X_2)$



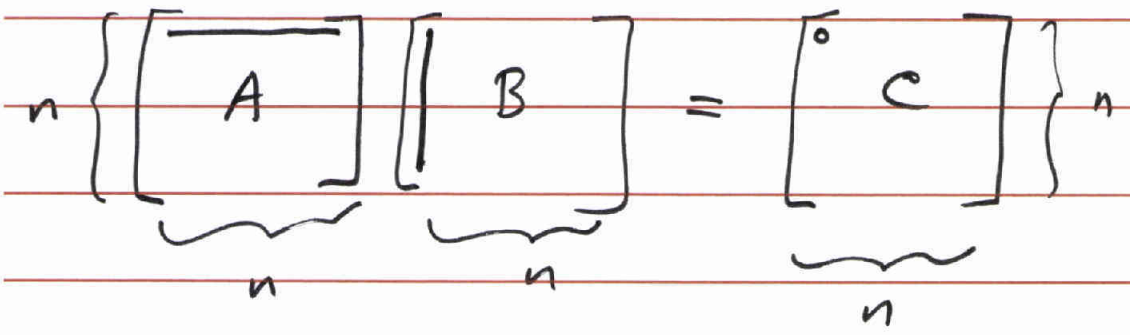
Divide : $O(1)$

Combine : $O(1)$

$f(n) = O(1)$

$n^{\log_b a} = n^{\log_2 2} = n^1$

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



Brute force $\longrightarrow O(n^3)$

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

$f(n) = D(n) + C(n) = O(n^2)$

$D(n) \rightarrow O(1)$ $\quad$ $C(n) \rightarrow O(n^2)$

$n^{\log_b a} = n^{\log_2 8} = n^3 \Longleftrightarrow O(n^3)$

Compute 7 $n/2 \times n/2$ intermediate matrices

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

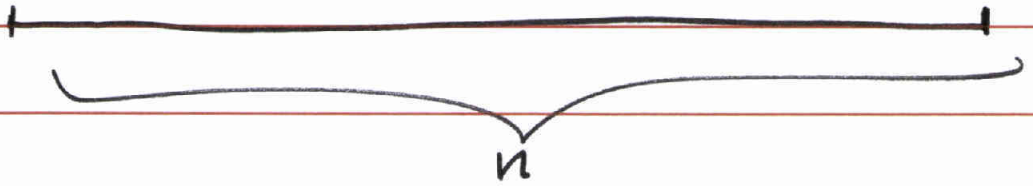
$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

Strassen's Method

$a = 7$
$b = 2$
$\Rightarrow n^{\log_b a} = n^{\log_2 7} = n^{2.81}$

$f(n) = \Theta(n^2) \Rightarrow \Theta(n^{2.81})$

Finding Min & Max in an unsorted array



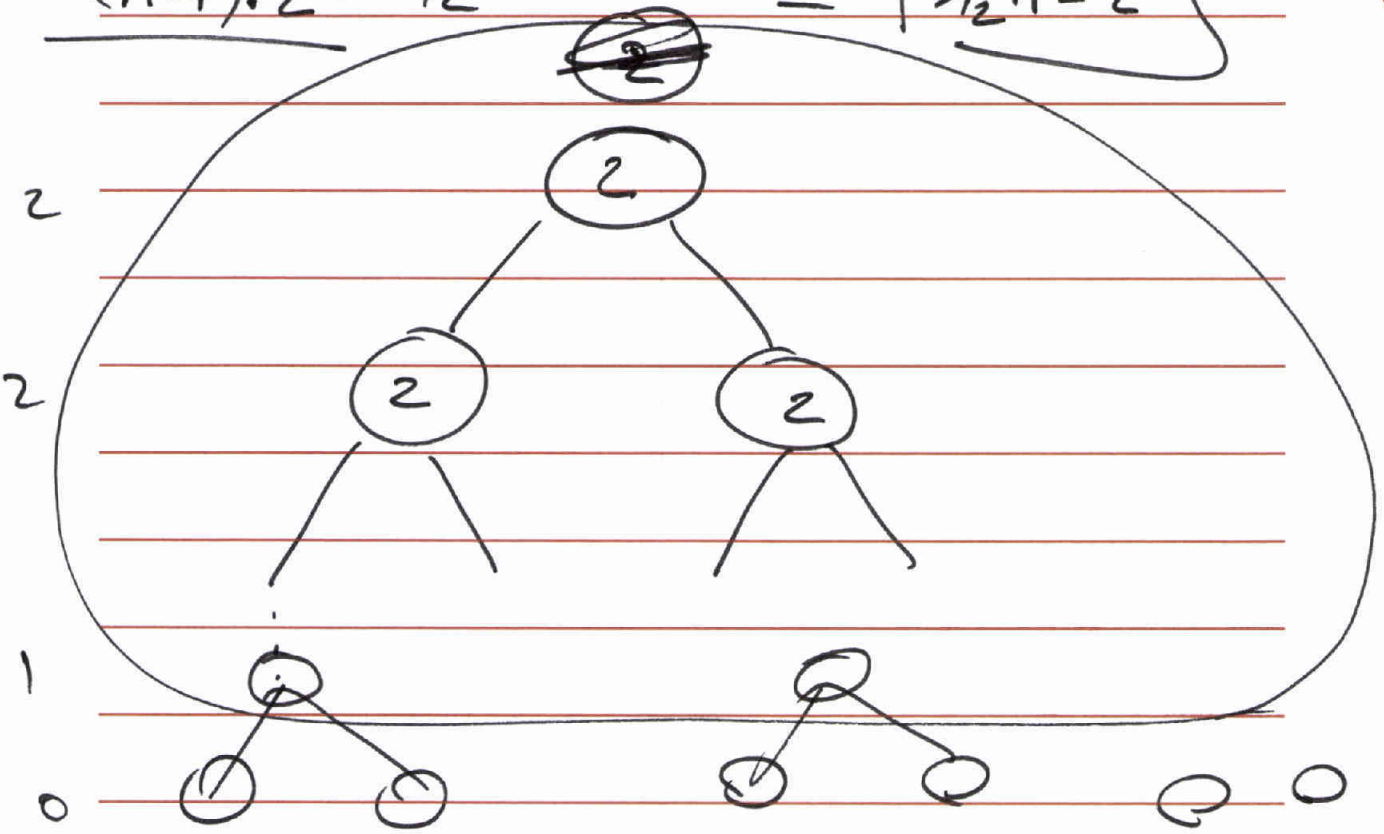
O(n) Brute force

takes ~~O(n-1)~~ (n-1) comp's to find Min

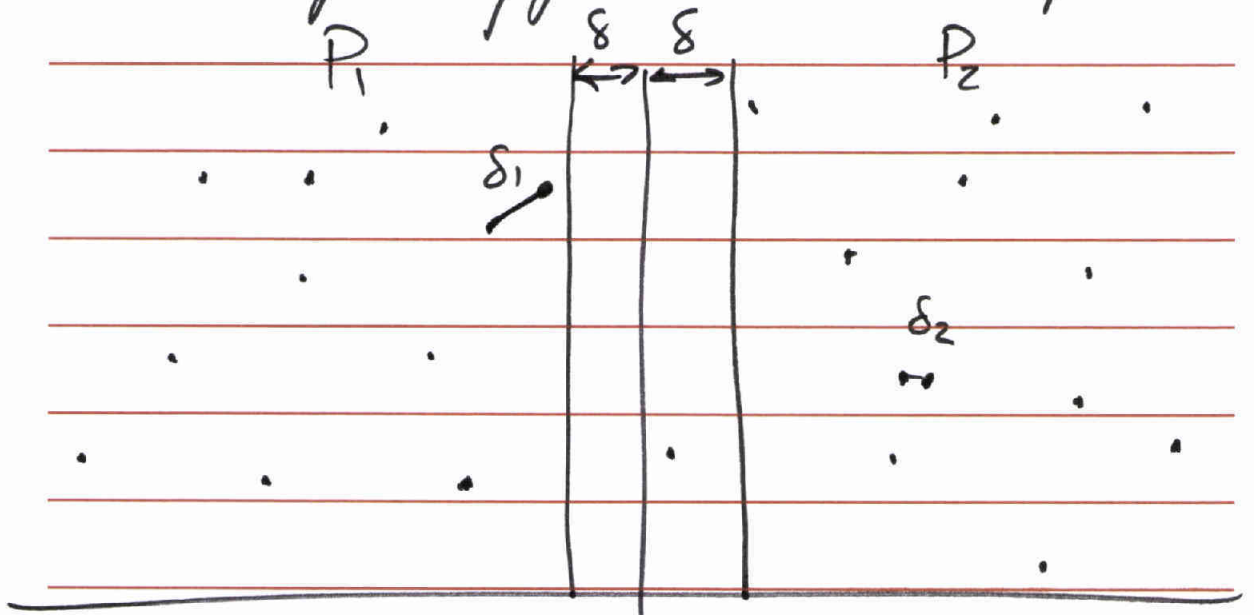
" (n-1) " " " Max.

2(n-1)

$(n-1) \cdot 2 - n/2 = \boxed{3/2\, n - 2}$



Closest pair of points on a 2D plane



Brute force: $(n-1)+(n-2)+\cdots+1$

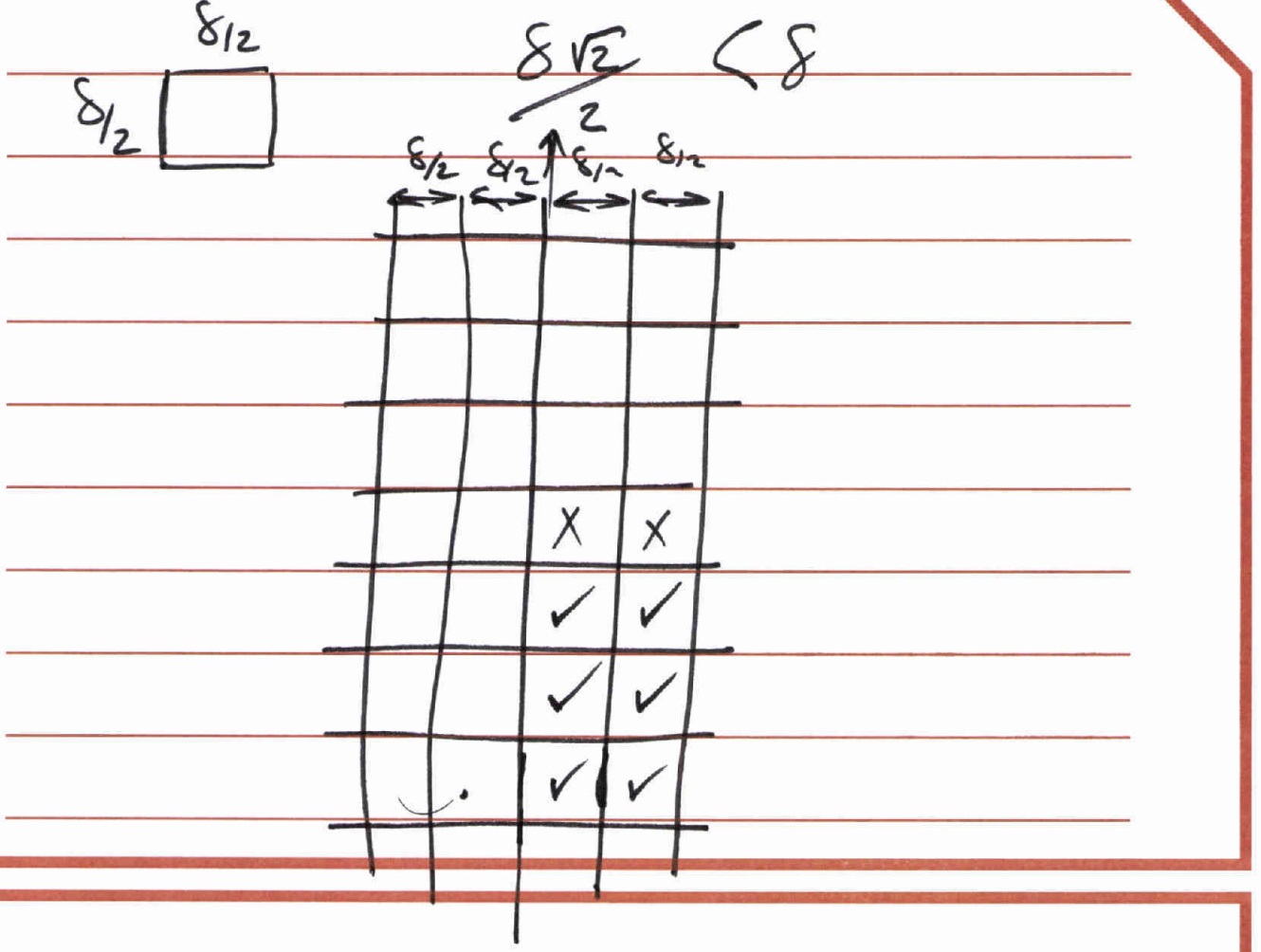
$O(n^2)$

Case 1: Both pts are in P_1

Case 2: " " " " P_2

Case 3: one pt. in P_1 & the other in P_2

$\delta = \min(\delta_1, \delta_2)$


δ/2
δ/2
δ√2/2 < δ
δ/2
δ/2
δ/2
δ/2
X
X
✓
✓
✓
✓
✓
✓

Implementation

closest-pair (P)

$O(n \log n)$ Construct P_x : list of points sorted by x-coord.

$O(n \log n)$ " P_y : list of points sorted by y-coord.

$(p_0, p_1) =$ Closest_pair_Rec(P_x, P_y)

Closest_pair_Rec (P_x, P_y)

if $|P| \leq 3$ then

solve it directly

else

$O(1)$ Construct Q_x ... left half of P_x

$O(n)$ " Q_y ... list of points in Q_x sorted by y-coord.

$O(1)$ Construct R_x ... right half of P_x

$O(n)$ " R_y ... list of points in R_x sorted by y-coord.

$(q_0, q_1) = \text{Closest\_pair\_Rec}(Q_x, Q_y)$

$(r_0, r_1) = \text{Closest\_pair\_Rec}(R_x, R_y)$

$\delta = \min(d(q_0, q_1), d(r_0, r_1))$

$O(n)$ {
- S = set of points in P within distance of δ from L.
- Construct S_y -- set of points in S sorted by y-coord.

$O(n)$ {
- for each point $s \in S_y$, compute distance from s to each of next 11 points in S_y.
- let (s, s') be pair with min. distance

$O(1)$ {
- if $d(s, s') < \delta$ then
  - Return (s, s')
- else if $d(q_0, q_1) < d(r_0, r_1)$ then
  - Return (q_0, q_1)
- else
  - Return (r_0, r_1)
- endif

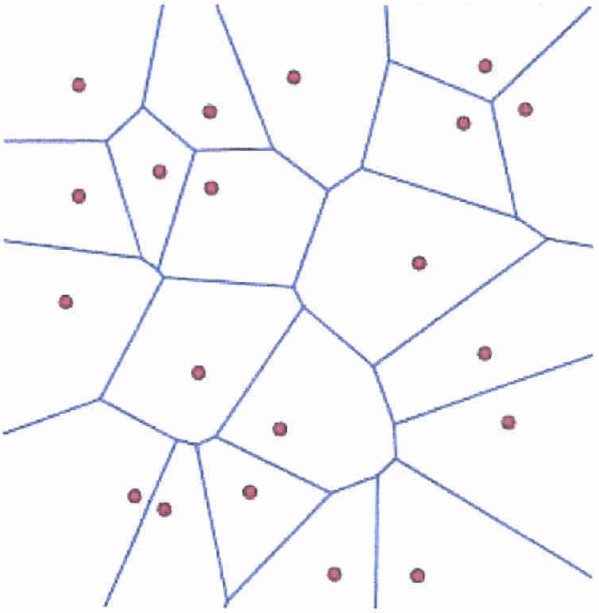
$$f(n) = O(n) + O(n) = O(n)$$

$$n^{\log_b a} = n^{\log_2 2} = n^1$$

$$\Rightarrow \Theta(n \lg n)$$

initial sort takes $\Theta(n \lg n) + \underbrace{\Theta(n \lg n)}_{\text{D\&C sol.}} = \Theta(n \lg n)$

All-pairs Closest Points



Voronoi Diagram

Find the median of an array.

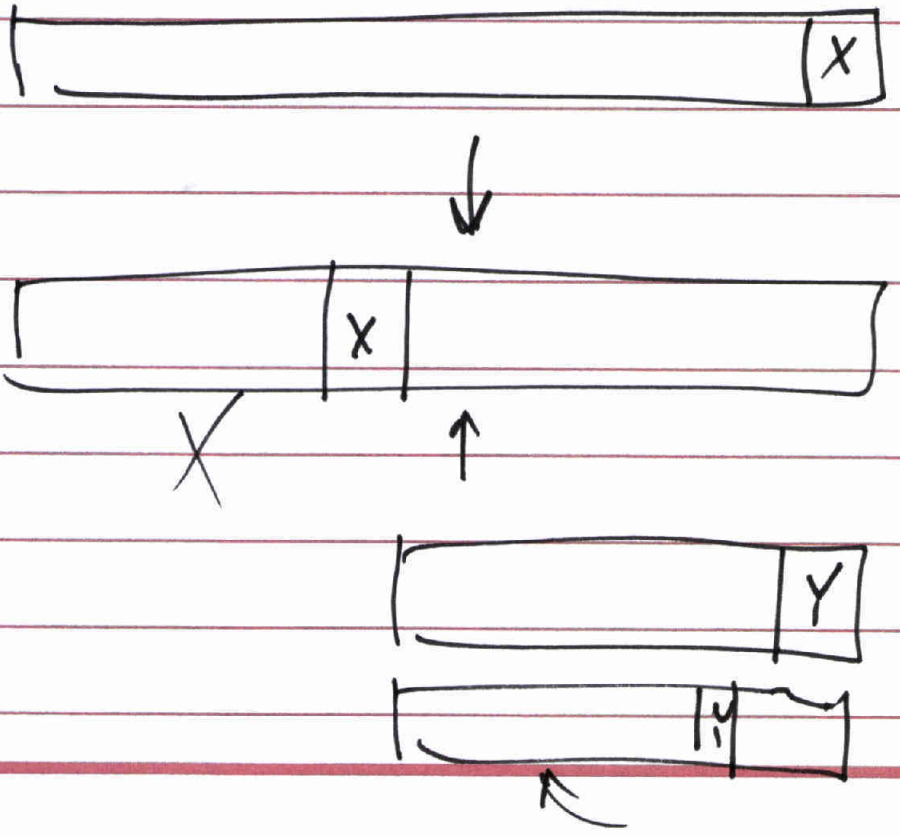
Sort takes $O(n \lg n)$

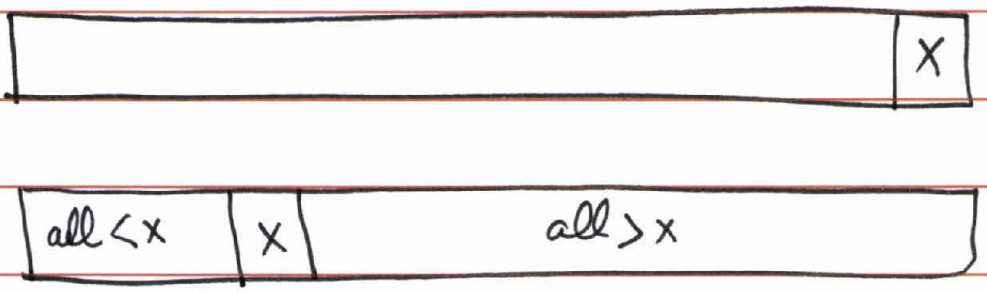
Randomized Algorithms

Two general types:

1- Algorithm relies on random nature of input

2- Algorithm behaves randomly





if looking for median we only need to look in the right half now.

Size of your subproblem is $\frac{n}{1+\alpha}$

$$\text{Complexity} = T(n) = T\left(\frac{n}{1+\alpha}\right) + O(n)$$

$f(n) = O(n)$

$$n^{\log_b q} = n^{\log_{1+\alpha} 1} = n^{0} = O(n)$$