Lecture 2:

From last time, running time of insertion sort is

$$2n - 1 + 3\sum_{j=2}^{n} t_j$$

where t_j = # times we execute the loop

while $i > 0$ and $A[i] > A[j]$

starting with $i = j-1$.

- Best case: exit loop because $\overbrace{A[i] \leq A[j]}^{\text{when } i=j-1} \Rightarrow t_j = 1$
- Worst case: exit loop because $i = 0 \Rightarrow t_j = j$
- Avg case: $t_j \approx j/2$

Best Case: $2n-1+3\sum_{j=2}^{n} 1 = 2n-1+3(n-1) = 5n-4 = \Theta(n)$

Worst case: $2n-1+3\sum_{j=2}^{n} j = 2n-1+3\left(\sum_{j=1}^{n} j - 1\right)$

$= 2n-1+3\sum_{j=1}^{n} j - 3$

$= 2n-4+3\frac{n(n+1)}{2}$

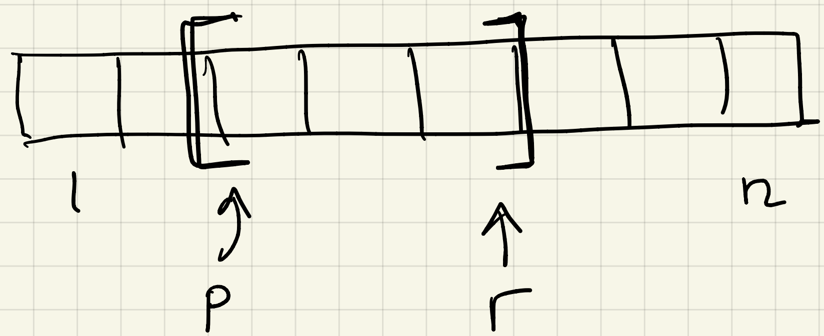
Note: $\sum_{j=1}^{n} j = 1+2+3+\cdots+n = n(n+1)/2$

$2n-4+\frac{3n(n+1)}{2} = \Theta(n^2)$

Merge Sort based on divide and conquer.

sort subproblems, and combine them in some way.

$A[p \dots r]$ defines a subproblem



p & r change over time

of course we start with $p=1$, $r=n$

<u>Divide</u> : Split array $A[p \dots r]$ into two sub arrays $A[p \dots q]$ $A[q+1 \dots r]$, choose q to be half-way between p and r

<u>Conquer step:</u> Recursively sort $A[p \dots q]$ & $A[q+1 \dots r]$

<u>Combination:</u> Merge the two sorted arrays $A[p \dots q]$ and $A[q+1 \dots r]$ into one sorted array.

Merge Sort (A, p, r)

if $p < r$

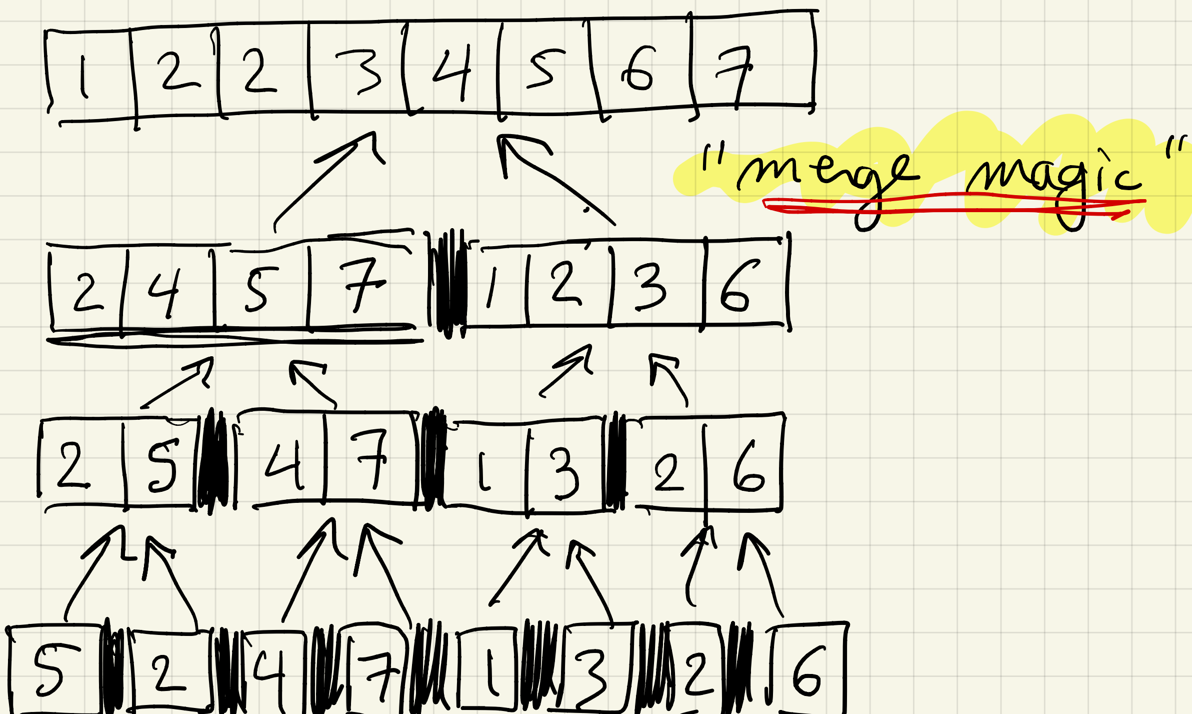
then $q \leftarrow \left\lfloor \frac{p+r}{2} \right\rfloor$

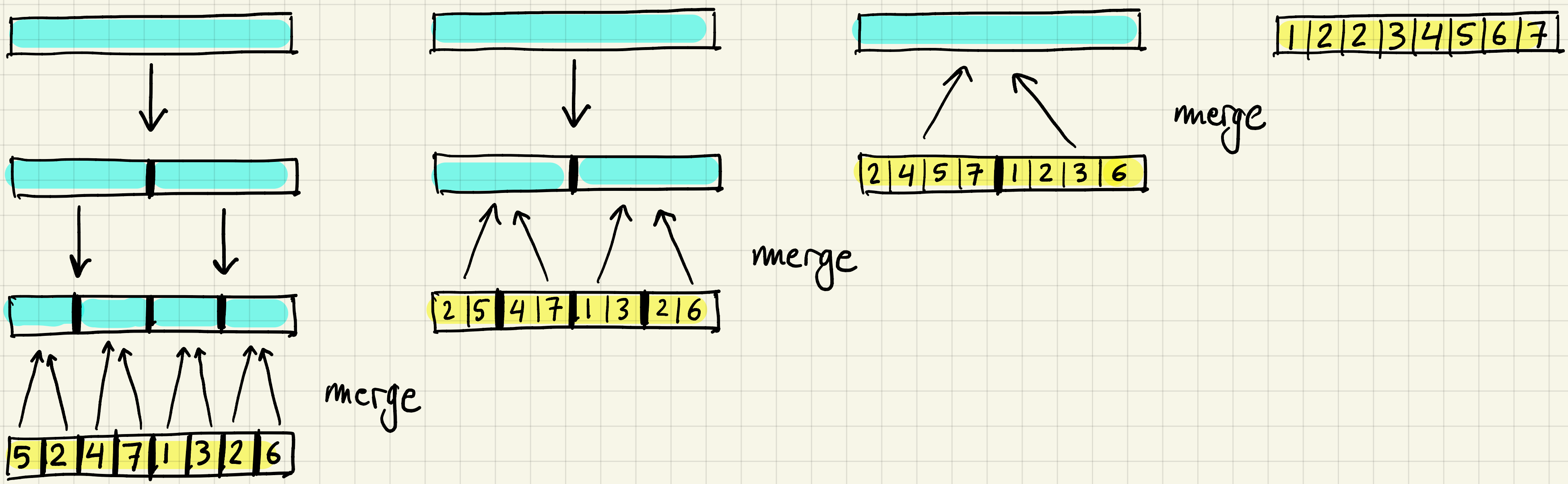
Merge Sort (A, p, q)

Merge Sort $(A, q+1, r)$

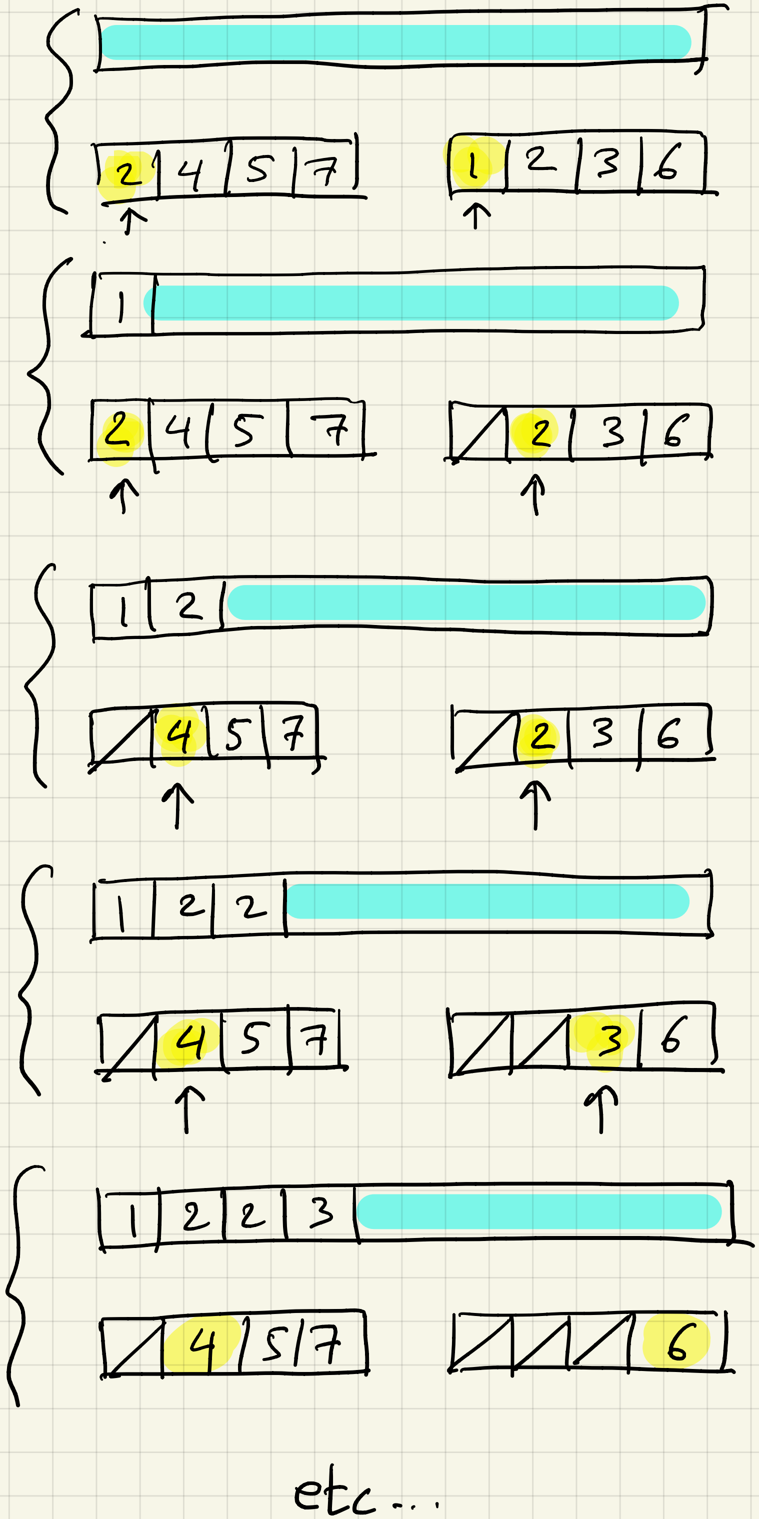
Merge (A, p, q, r)

<u>Example:</u>





merge
merge
merge
5 2 4 7 1 3 2 6
2 5 4 7 1 3 2 6
2 4 5 7 1 2 3 6
1 2 2 3 4 5 6 7



Merge is NOT in place

Merging takes linear time, that's the benefit we are getting.

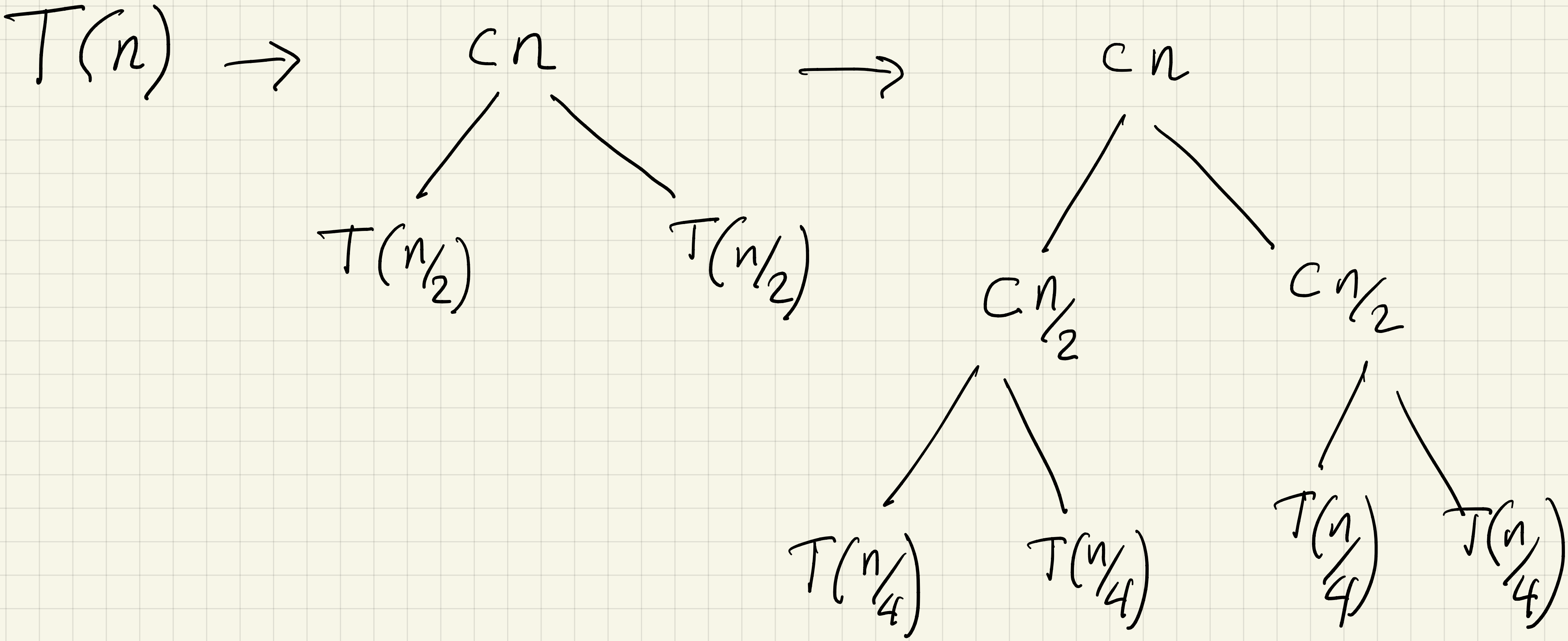
Recurrence: Let $T(n)$ be time needed to sort an array of size n.

size of subproblem

$$T(n) = \begin{cases} \underline{\underline{2}}\, T(n/2) + cn & n > 1 \\ c & n = 1 \end{cases}$$

# of subproblem in the divide step

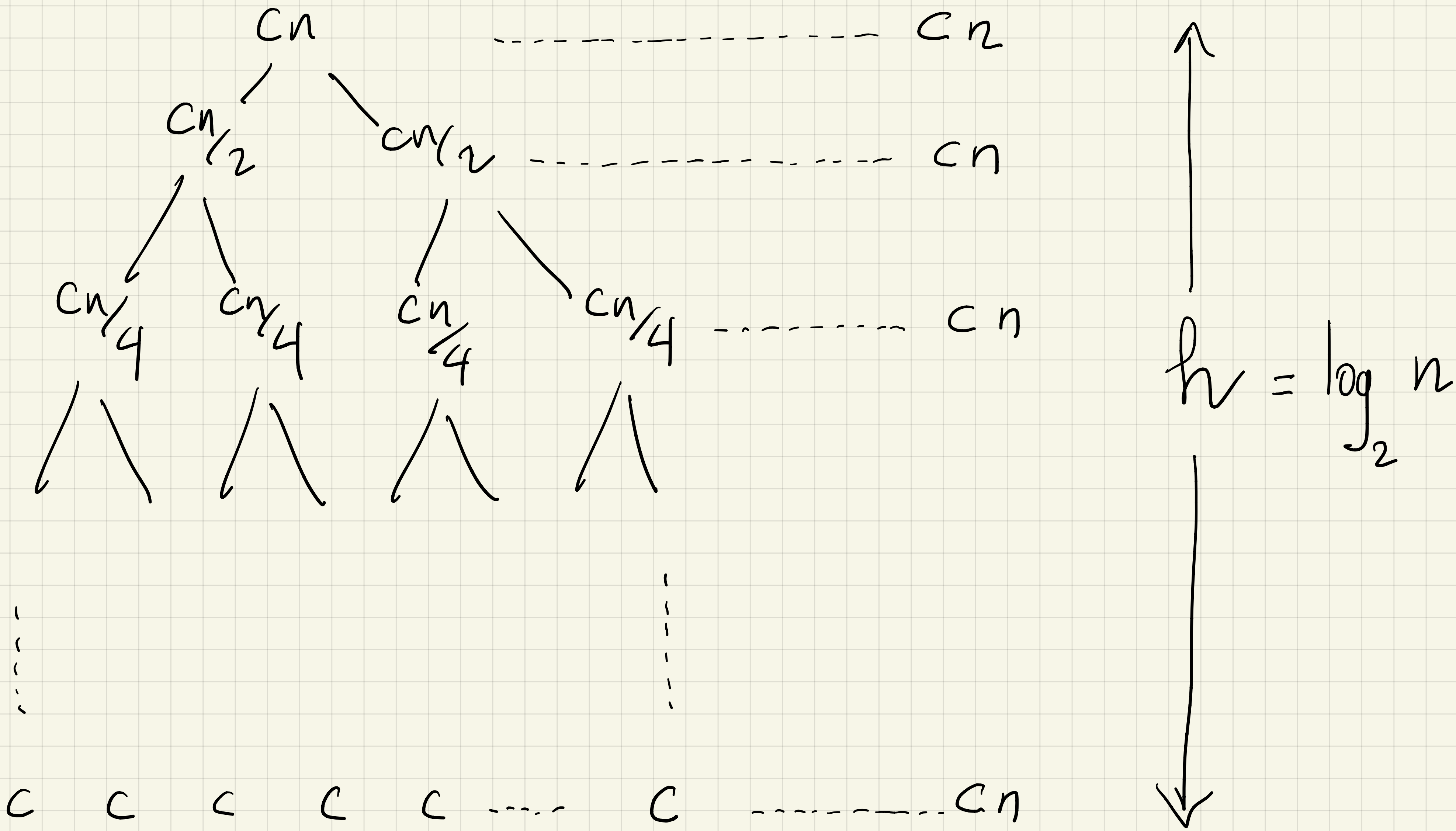
I am using the same constant c for simplicity.



$\cdots$

Finally $T(1) \longrightarrow c$

Assume n is power of 2 for simplicity



$$\# \text{ total time} = cn \times (\# \text{ levels})$$

$$cn \times (\text{height tree} + 1)$$

$$= cn(\log_2 n + 1) = cn\log_2 n + cn$$

$$cn \log_2 n + cn = \Theta(n \log_2 n) = \Theta(n \lg n)$$

But base does not matter as we will see later

Strassen's Alg.

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$r = ae + bg$$

$$u = cf + dh$$

Straight forward algorithm: For each entry multiply a row by a col.

$n \times n$ matrix $\Rightarrow n^2$ entries to compute.

each requires a row of size n to be mult. by a col of size n

row / col combination $\Rightarrow n$ multiplication, $n-1$ addition

$\Rightarrow \Theta(n)$ operations / entry

This alg. require $n^2 \Theta(n) \approx \Theta(n^3)$ time

Divide & Conquer for $n \times n$ matrix

$$\begin{array}{cc} & \xleftrightarrow{n/2} \\ \updownarrow n & \left[\begin{array}{c|c} R & S \\ \hline T & U \end{array}\right] \end{array} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \times \left[\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right]$$

$$R = AE + BG$$

$$T(n) = \begin{cases} 8T(n/2) + cn^2 & n > 1 \\ c & n = 1 \end{cases}$$