Example: Merge-Sort

L3

Problem: sort an array of n numbers.

- Divide? $A[1..n/2]$, $A[n/2..n]$
- Merge?

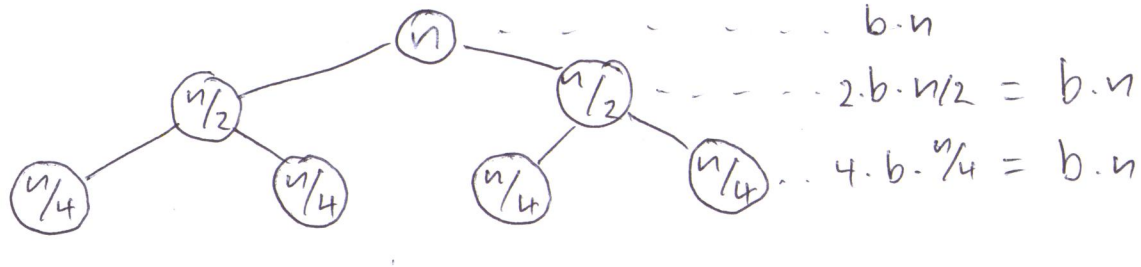
  Iteratively compare the first element in both lists and move the smallest to the next spot in the output.

What is the running time?

$$T(n) = \cancel{2T(n/2) + O(n)} \begin{cases} O(1) & \text{if } n=1 \\ 2T(n/2) + O(n) & \text{o.w.} \end{cases}$$

$$T(n) \leq \begin{cases} a & \text{if } n=1 \\ 2T(n/2) + b \cdot n & \text{o.w.} \end{cases}, \text{ for some constants } a, b > 0.$$

Draw the recursion tree:



$\vdots$

$1 \quad 2 \quad 4 \quad 8 \quad \cdots \quad 2^d \cdot a$

what is the depth? $n/2^d = 1 \Rightarrow n = 2^d \Rightarrow d = \log_2 n$

So $T(n) = \sum_{i=0}^{d-1} b \cdot n + 2^d \cdot a$

$= \log_2 n \cdot b \cdot n + n \cdot a$

$= O(n \log n)$

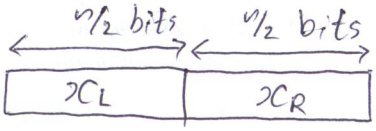
Example: Multiplying Integers

Problem: multiply n-bit integers x and y

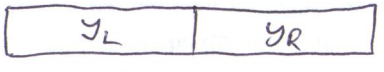
Long multiplication uses $O(n^2)$ bit operations.

Can we do better? Yes, with divide-and-conquer.

- Divide?



$x = 2^{n/2} x_L + x_R$



$y = 2^{n/2} y_L + y_R$

$x \cdot y = (2^{n/2} x_L + x_R) \cdot (2^{n/2} y_L + y_R)$

$= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$

- Merge? Recursively compute these

What is the running time?

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ 4T(n/2) + O(n) & \text{o.w.} \end{cases} \leq \begin{cases} a & \text{if } n=1 \\ 4T(n/2) + b \cdot n & \text{o.w.} \end{cases}$$

3 additions
2 bit-shifts

Assume that $n = 2^k$, $a = b = 1$.

We can also solve recurrences by *unfolding*:

$$
\begin{aligned}
T(n) &\le 4T(n/2) + n \\
&\le 4(4T(n/4) + n/2) + n \\
&= 4^2 T(n/2^2) + 2n + n \\
&\le 4^2(4T(n/2^3) + n/2^2) + 2n + n \\
&= 4^3 T(n/2^3) + 4n + 2n + n \\
&\le 4^4 T(n/2^4) + 8n + 4n + 2n + n \\
&\le \dots \\
&\le 4^k \cdot T(n/2^k) + (2^{k-1} + 2^{k-2} + \dots + 2 + 1) \cdot n \\
&= (2^2)^k \cdot T(1) + (2^{k-1+1} - 1) \cdot n \\
&= (2^k)^2 \cdot 1 + (2^k - 1) \cdot n \\
&= n^2 + (n-1) \cdot n \\
&\le 2n^2 \\
&= O(n^2) \qquad \Rightarrow \text{No improvement over long multiplication!}
\end{aligned}
$$

Why? Because recursive calls are expensive.

Can we make fewer recursive calls? Yes!

Karatsuba (1960):

$$xy = 2^n x_L y_L + 2^{n/2}\left[(x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R\right] + x_R y_R$$

This reduces the number of recursive calls to 3: $x_L y_L$, $x_R y_R$, and $(x_L + x_R)(y_L + y_R)$.

What is the running time?

$$T(n) \leq \begin{cases} a & \text{if } n = 1 \\ 3 \cdot T(n/2) + b \cdot n & \text{otherwise} \end{cases}$$

Unfolding: (Assume $n = 2^k$ and $a = b = 1$)

$$\begin{aligned}
T(n) &\leq 3T(n/2) + n \\
&\leq 3(3T(n/2^2) + n/2) + n \\
&= 3^2 T(n/2^2) + \tfrac{3}{2} \cdot n + n \\
&\leq 3^2(3T(n/2^3) + n/2^2) + \tfrac{3}{2} \cdot n + n \\
&= 3^3 T(n/2^3) + \left(\tfrac{3^2}{2^2} + \tfrac{3}{2} + 1\right) \cdot n \\
&\vdots \\
&\leq 3^k \cdot T(1) + n \cdot \sum_{i=0}^{k-1} \left(\tfrac{3}{2}\right)^i \\
&= 3^k + n \cdot \frac{\left(\frac{3}{2}\right)^k - 1}{\frac{3}{2} - 1} \\
&= 3^k + n \cdot 2 \cdot \left(\frac{3^k}{2^k} - 1\right) \\
&= 3^k + 2 \cdot n \cdot \frac{3^k}{2^k} - 2n \\
&= 3^k + 2 \cdot 3^k - 2n \qquad (n = 2^k) \\
&\leq 3 \cdot 3^k \\
&= 3 \cdot \left(2^{\log_2 3}\right)^k = 3 \cdot \left(2^k\right)^{\log_2 3} = 3 \cdot n^{\log_2 3} = O(n^{\log_2 3}) \\
&\approx O(n^{1.58\ldots})
\end{aligned}$$