Recap:

Merge Sort ("Divide and Conquer")

```
MergeSort(A[1,...,n]):
    MergeSort(A[1,...,n/2])
    MergeSort(A[n/2+1,...,n])
    Merge A[1,...,n/2] & A[n/2+1,...,n]
```

```
Merge(A, m, B, n, C) // A[1,...,m] & B[1,...,n] sorted
    i, j = 1
    For k = 1 to (m+n)
        If A[i] ≤ B[j]
            C[k] = A[i]
            i = i+1
        Else
            C[k] = B[j]
            j = j+1
        EndIf
    EndFor
```

1. Analysis of Merge Sort

Correctness of Merge: ✓

Runtime of Merge: $\Theta(m+n)$

Correctness of MergeSort: follows from the correctness of Merge and strong induction

Inductive Hypothesis: MergeSort is correct for arrays of length k

Base Case: $k=1$, trivial

Inductive Step: Assume inductive hypothesis holds for all $1, 2, \dots, k$, then it also holds for $k+1$.

Runtime of MergeSort: $\Theta(n \log n)$

$$T(n) = \overbrace{2T\left(\frac{n}{2}\right)}^{\text{Recursive Calls}} + \overbrace{n}^{\text{Merge}}$$

(Recurrence Relation)

~~We show by induction that $T(n) = \Theta(n)$~~

~~Base case: $n=1$ ✓~~

~~Induction: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$~~

~~$= 2 \cdot \Theta\left(\frac{n}{2}\right) + n = \Theta(n)$~~

Sorting requires $\Omega(n \log n)$ (in comparison model) e.g. Consider this recursive "Contains" function that checks if an array contains a specific value:

```
Contains(A, x):
    If A.length=0:
        Return false
    If A[1]== x:
        Return true
    Return Contains(A[2:], x)
```

~~We show its runtime is $O(1)$ using induction.~~

~~Base Case: A empty, ✓~~

~~Inductive Step: $T(n) = T(n-1)+1 = O(1)+1 = O(1)$~~

What's wrong with this?

Let's plug in the def. of big O:

$\exists c, n_0 > 0$, s.t. $\forall n > n_0$, $0 \leq f(n) \leq c \cdot g(n)$

So when we assume $T(n-1) = O(1)$, we are assuming for some c, $T(n-1) \leq c \cdot 1 = \underline{c}$,

but then $T(n) = T(n-1) + 1 \leq \underline{c+1}$

This is a different statement!

Induction ☺ Asymptotics ☺ Induction+Asymptotics ☹

Merge Sort:

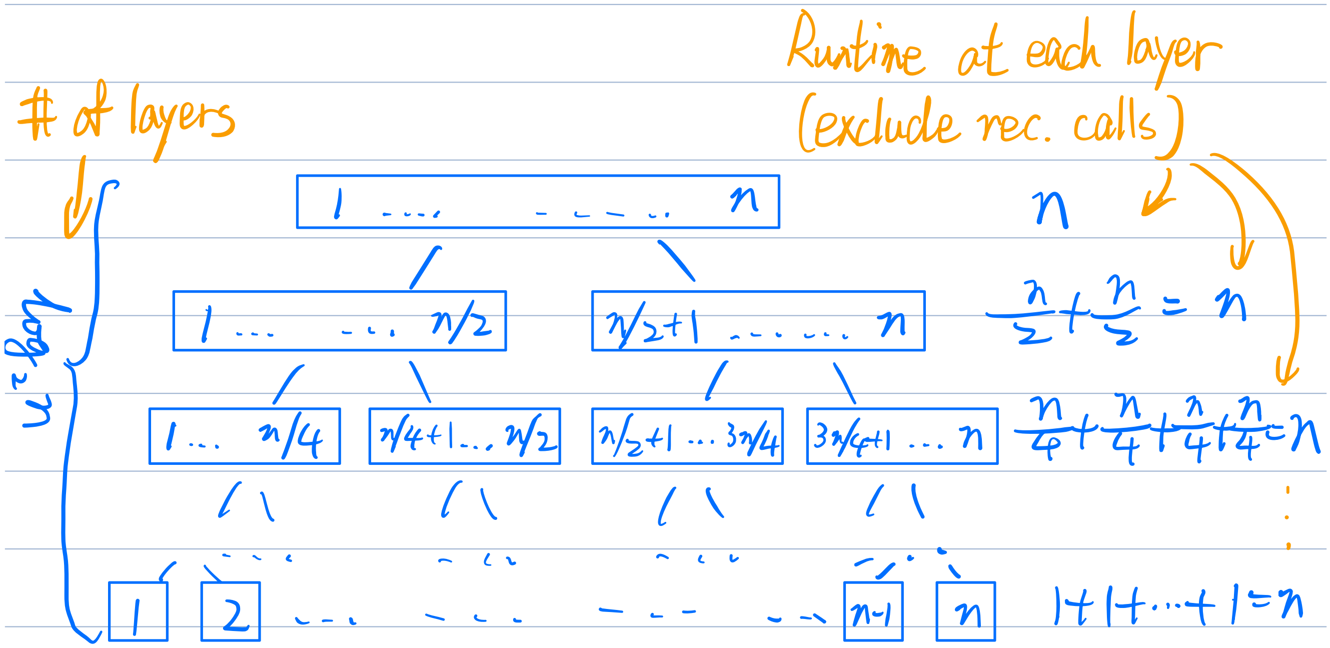
- $\Theta(n \log n)$ worst/average/best case
- "Stable": if two elements have the same key, their relative order is unchanged after sorting.
- Not "in place"
- In practice, use index into arrays instead of copying the subarrays (kinda "in place")
- Based on "Divide and Conquer"
  Divide → Conquer → Combine

2. Solving Recurrences

So... how do we handle $T(n) = 2 \cdot T(\frac{n}{2}) + n$?

Method ①: Recursion Tree

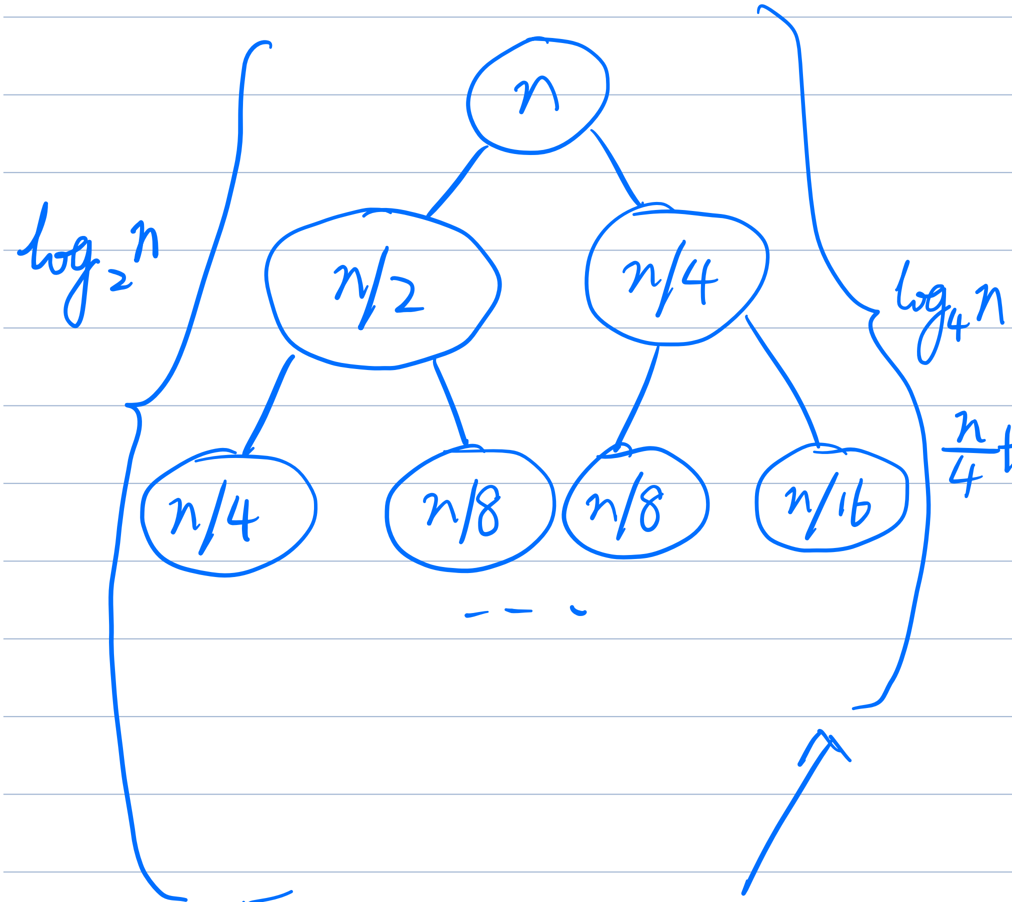
Informal but helpful



Overall runtime is the sum of runtime at each layer.

$$T(n) = n \cdot \log_2 n = \Theta(n \log n)$$

e.g. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + n$



n

$\frac{n}{2} + \frac{n}{4} = \frac{3}{4}n$

$\frac{n}{4} + \frac{n}{8} + \frac{n}{8} + \frac{n}{16} = \frac{9}{16}n$

$\left(\frac{3}{4}\right)^3 n$

$\vdots$

More layers on the left-hand side!

$$T(n) = n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \left(\frac{3}{4}\right)^3 n + \cdots$$
$$= n \cdot \left(1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \cdots\right)$$
$$= 4 \cdot n = \Theta(n)$$

Method ②: Substitution Method

To prove formally

e.g. To show for $T(n) = 2\cdot T(\frac{n}{2}) + n$,

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

Claim: For some large enough constant c, $\forall n \geq 2$

$$T(n) \leq c \cdot n \log n$$

Proof: Assume by induction the above claim is true for $n' < n$, then

$$\begin{aligned} T(n) &= 2\cdot T(\tfrac{n}{2}) + n \\ &\leq 2\cdot(c\cdot\tfrac{n}{2}\cdot\log\tfrac{n}{2}) + n \\ &= c\cdot n\cdot(\log n - 1) + n \\ &= cn\log n - cn + n \\ &\leq cn\log n \quad (\text{as long as } c \geq 1) \end{aligned}$$

$\divideontimes$

Method ③: Master Theorem

For $T(n)$ in the form of

$$T(n) = a\cdot T(\tfrac{n}{b}) + f(n);\ a \geq 1,\ b > 1$$

e.g. $f(n)=0$

$$T(n) = a\cdot T\left(\frac{n}{b}\right) = a\left(a\cdot T\left(\frac{n}{b^2}\right)\right)$$
$$= a^2 T\left(\frac{n}{b^2}\right) = a^3\cdot T\left(\frac{n}{b^3}\right)$$
$$\cdots$$
$$= a^k T\left(\frac{n}{b^k}\right)$$

Say we want $b^k = n$ so that we have $T(1)$

$k = \log_b n$

$$\Rightarrow T(n) = a^{\log_b n} T(1) = \Theta\left(a^{\log_b n}\right)$$

$$a^{\log_b n} = a^{\frac{\log_2 n}{\log_2 b}} = \left(2^{\log_2 a}\right)^{\frac{\log_2 n}{\log_2 b}}$$
$$= \left(2^{\log_2 n}\right)^{\frac{\log_2 a}{\log_2 b}} = n^{\log_b a}$$

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

e.g. $f(n) = n^{100}$

$$T(n) = aT\left(\frac{n}{b}\right) + n^{100}$$
$$= a\cdot\left(a\cdot T\left(\frac{n}{b^2}\right) + \left(\frac{n}{b}\right)^{100}\right) + n^{100}$$
$$= a^2 T\left(\frac{n}{b^2}\right) + \left(\frac{a}{b^{100}}\right)\cdot n^{100} + n^{100}$$
$$\vdots$$
$$= \Theta\left(n^{100}\right)$$

$n^{\log_b a}$ $\qquad \frac{a}{b^{100}}\cdot n^{100}$

$a\cdot f\left(\frac{n}{b}\right)$

Master Theorem:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n); \quad a \geq 1 \quad b > 1$$

Case 1: $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$

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

Case 2: $f(n) = \Theta(n^{\log_b a} \cdot \log^k n)$ $\quad k \geq 0$

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

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$

and $af(n/b) \leq c \cdot f(n)$ for some $c < 1$ (regularity condition)

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

Intuition: let $Q = n^{\log_b a}$, then compare it with f:

Case 1: f polynomially smaller than Q

Case 2: f is larger than Q by a polylog factor

Case 3: f is polynomially larger than Q + regularity

Examples:

① $T(n) = 2T(\frac{n}{2}) + \Theta(n)$

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

$\Theta(n)/n = \Theta(1) = \Theta(\log^0 n)$

Case 2!

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

② $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$

$Q = n^{\log_b a} = n^{\log_2 7} \approx n^{2.8}$

$\frac{\Theta(n^2)}{n^{2.8}} = \Theta(n^{-0.8}) = O(n^{-0.8})$

Case 1!

$\Rightarrow T(n) = \Theta(n^{\log_2 7})$

③ $T(n) = 4T(n/2) + n^3$

$Q = n^{\log_b a} = n^{\log_2 4} = n^2$

$\frac{n^3}{n^2} = n = \Omega(n)$

Case 3?

$af(\frac{n}{b}) = 4 \cdot (\frac{n}{2})^3 = \frac{n^3}{2} \leq (\frac{3}{4}) \cdot n^3$

$\uparrow$
$c < 1$

Case 3!

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