Some ways to get $O(n)$-time algorithms

- iterate through input, $O(1)$ times.
  - ↳ e.g., verify sorted order, find 357-th smallest, partition
- classic divide and conquer with $O(n^{1-\varepsilon})$ non-recursive work. $\varepsilon = \Theta(1)$
  - ↳ $T(n) = b \cdot T\left(\frac{n}{b}\right) + O(n^{1-\varepsilon})$ leaf level dominates polynomially
- divide and conquer with $\Theta(n)$ non-recursive work.
  - ↳ $T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n)$ make sure $a < b$

e.g., suppose you must access all data → $\Omega(n)$ lower bound

(and iterating doesn't work)

"prune and search"

$$T(n) = T\left(\frac{n}{b}\right) + \Theta(n) \quad = \Theta(n)$$

Do $\Theta(n)$ non-recursive work,

eliminate a constant fraction of the data

Works for any polynomial $f(n)$

$$T(n) = T\left(\frac{n}{b}\right) + f(n) \quad = \Theta(f(n))$$

if all of that fails... here's one more idea:

Do $\Theta(n)$ non-recursive work, and some recursive work

in order to eliminate a constant fraction of the data

$$T(n) = \Theta(n) + T\left(\frac{n}{x}\right) + T\left(\frac{n}{b}\right)$$

this will work if $\frac{1}{x} + \frac{1}{b} = \frac{1}{c}$ for $c > 1$

e.g. $T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right)$ ✓ $T\left(\frac{3n}{4}\right) + T\left(\frac{n}{3}\right)$ ✗ $T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$ ✗

$$T(n) = T\left(\frac{n}{x}\right) + T\left(\frac{n}{b}\right) + pn \qquad \text{if } \frac{1}{x} + \frac{1}{b} = \frac{1}{c} \quad \text{for } c > 1$$

Hypothesis: for all $k < n$, $T(k) \leq d \cdot k$

$$T(n) \leq d \cdot \frac{n}{x} + d \cdot \frac{n}{b} + p \cdot n$$

$$= dn \cdot \left(\frac{1}{x} + \frac{1}{b}\right) + p \cdot n$$

$$= dn \cdot \frac{1}{c} + p \cdot n$$

$$= dn - \left(\frac{c-1}{c} \cdot dn - p \cdot n\right)$$

$$\leq dn \qquad \text{if } d \geq p \cdot \frac{c}{c-1}$$