Recitation 1

Divide-and-Conquer

Rebecca Lin

rebeccayelin.github.io/6.1220-SP25

Agenda

Divide-and Conquer

- **Merge Sort and Recurrence Trees**

Maximum Subarray Sum

Techniques for Solving Recurrences

- **Master Theorem**
- **Substitution**

Median Finding

Divide-and-Conquer

1. **Divide** problem into subproblems of the same type
2. **Conquer** (solve) each subproblem recursively
3. **Combine** the solutions to a solution of the original problem

Example **Recurrence**:

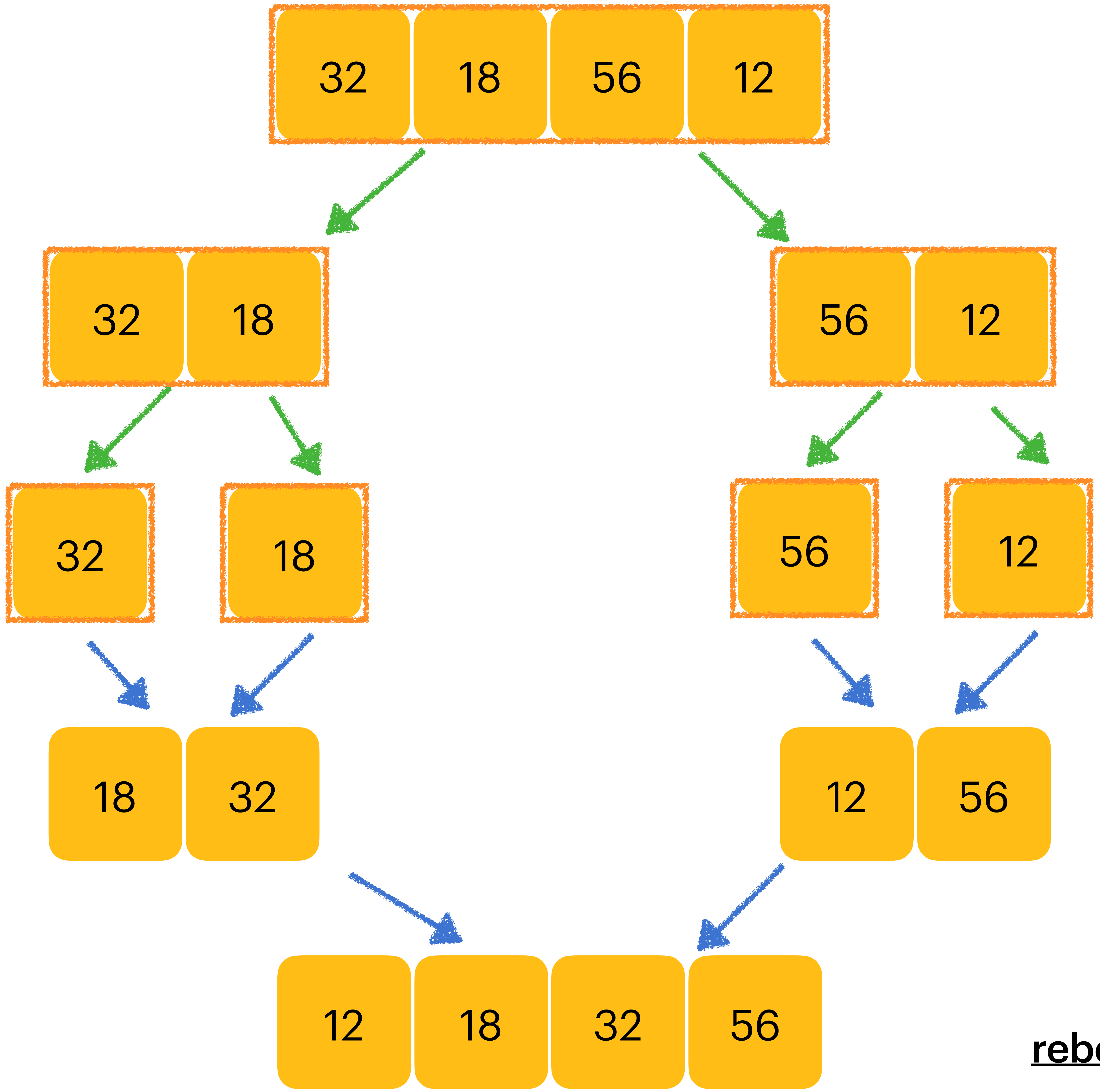
"work to conquer subproblems"

"work to reduce to and merge subproblems"

$$T(n) = \boxed{aT(\frac{n}{b})} + \boxed{f(n)}$$

"split into $a \geq 1$ number of subproblems, each of size $\frac{n}{b}$ where $b > 1$"

Example: Merge Sort



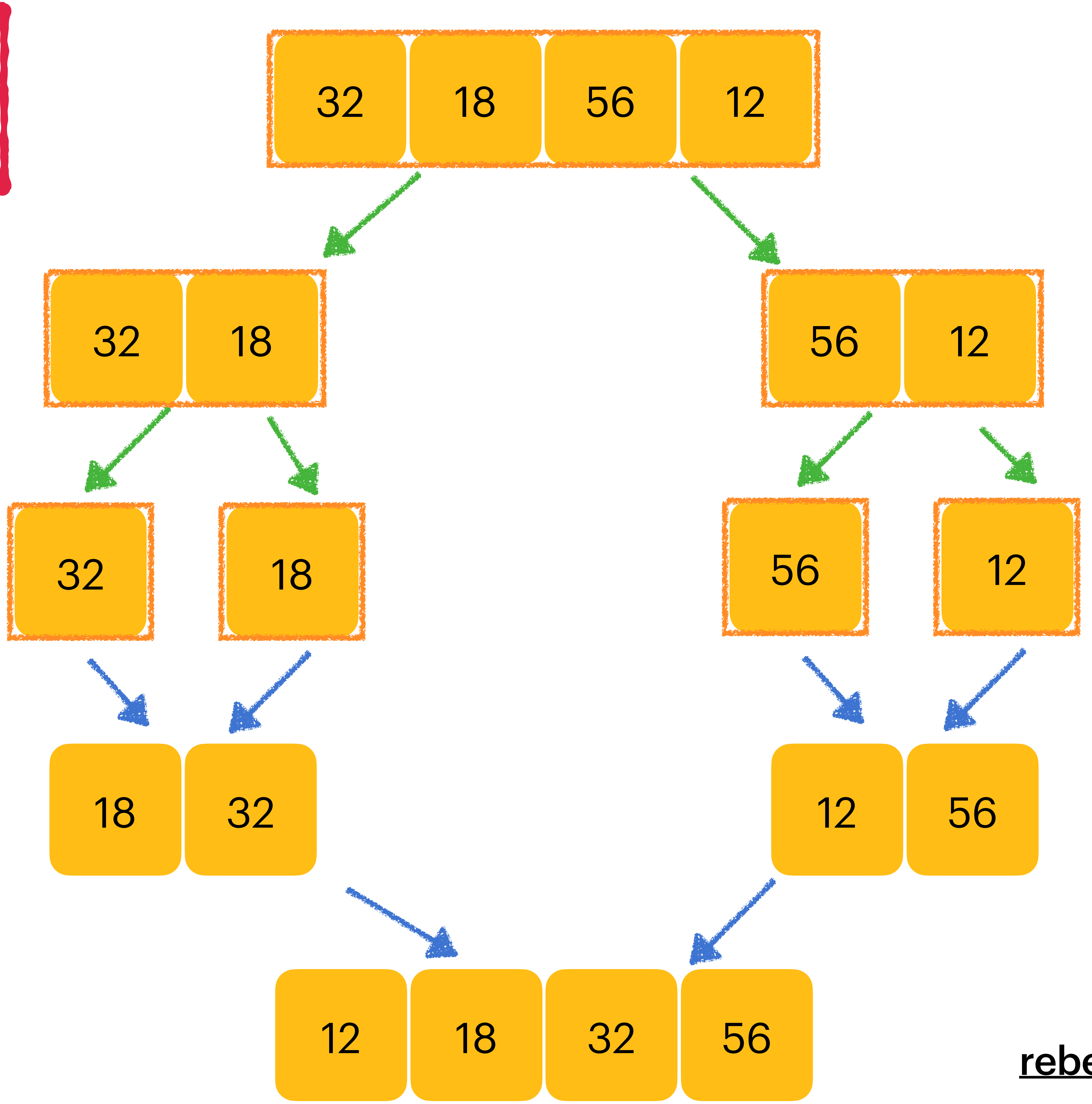
Rebecca Lin

rebeccayelin.github.io/6.1220-SP25

Example: Merge Sort

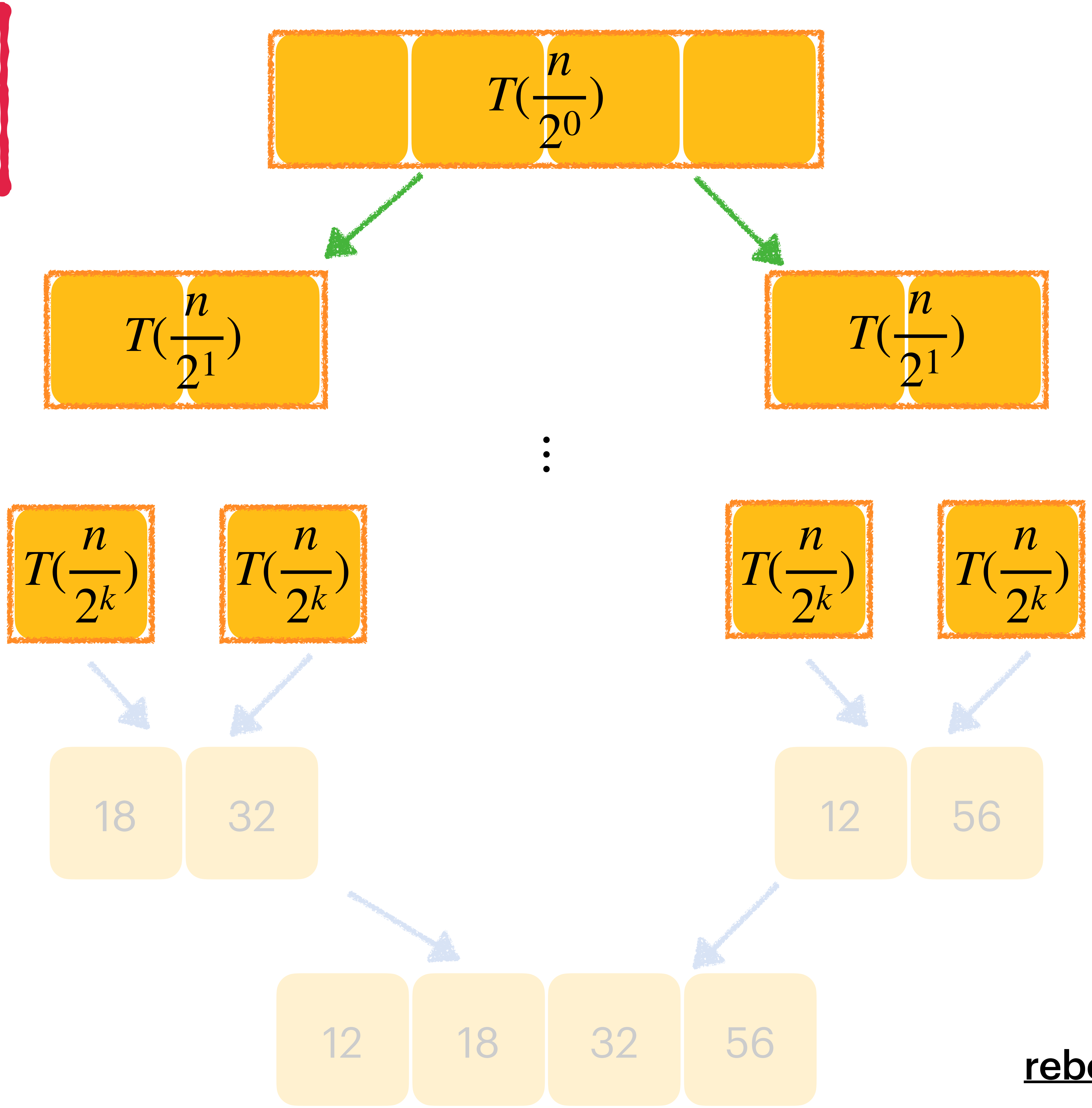
$$T(n) = 2T(\frac{n}{2}) + O(n)$$

Base Case
$T(1) = O(1)$



Example: Merge Sort

$$\boxed{T(n) = 2T(\frac{n}{2}) + O(n)}$$



$O_0(n)$

$+$

$O_1(n)$

$+$

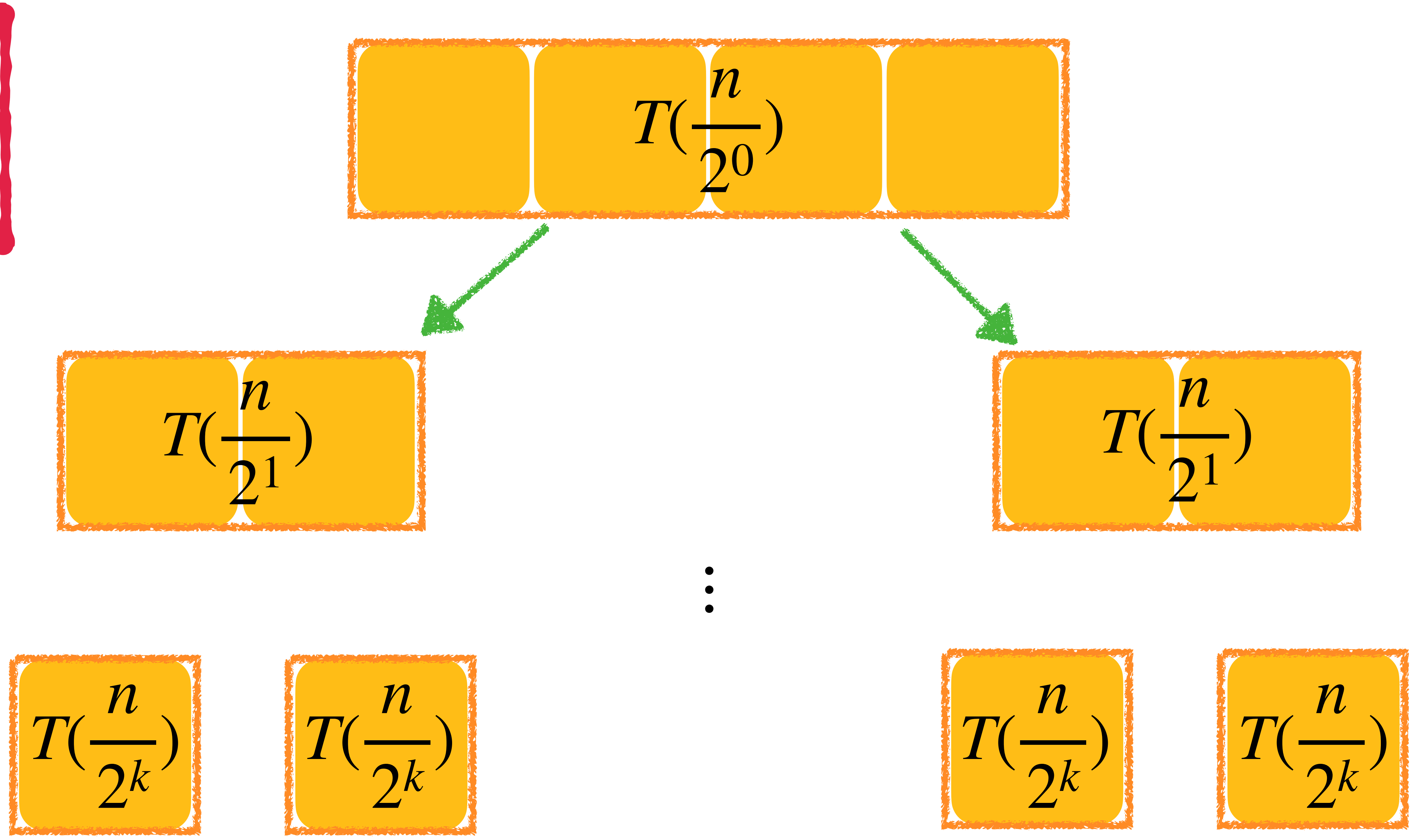
$O_{k=\lg n}(n)$

Base Case

$T(1) = O(1)$

Example: Merge Sort

$$\boxed{T(n) = 2T(\frac{n}{2}) + O(n)}$$



$T(\frac{n}{2^0})$

$T(\frac{n}{2^1})$ $T(\frac{n}{2^1})$

$\vdots$

$T(\frac{n}{2^k})$ $T(\frac{n}{2^k})$ $T(\frac{n}{2^k})$ $T(\frac{n}{2^k})$

$O_0(n)$

$+$

$O_1(n)$

$+$

$O_{k=\lg n}(n)$

Base Case
$T(1) = O(1)$

$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(kn) = O(n \log n)$$

Example: Maximum Subarray Sum

Given an array $A[1,\ldots,n]$, find the maximum sum of consecutive entries of A.

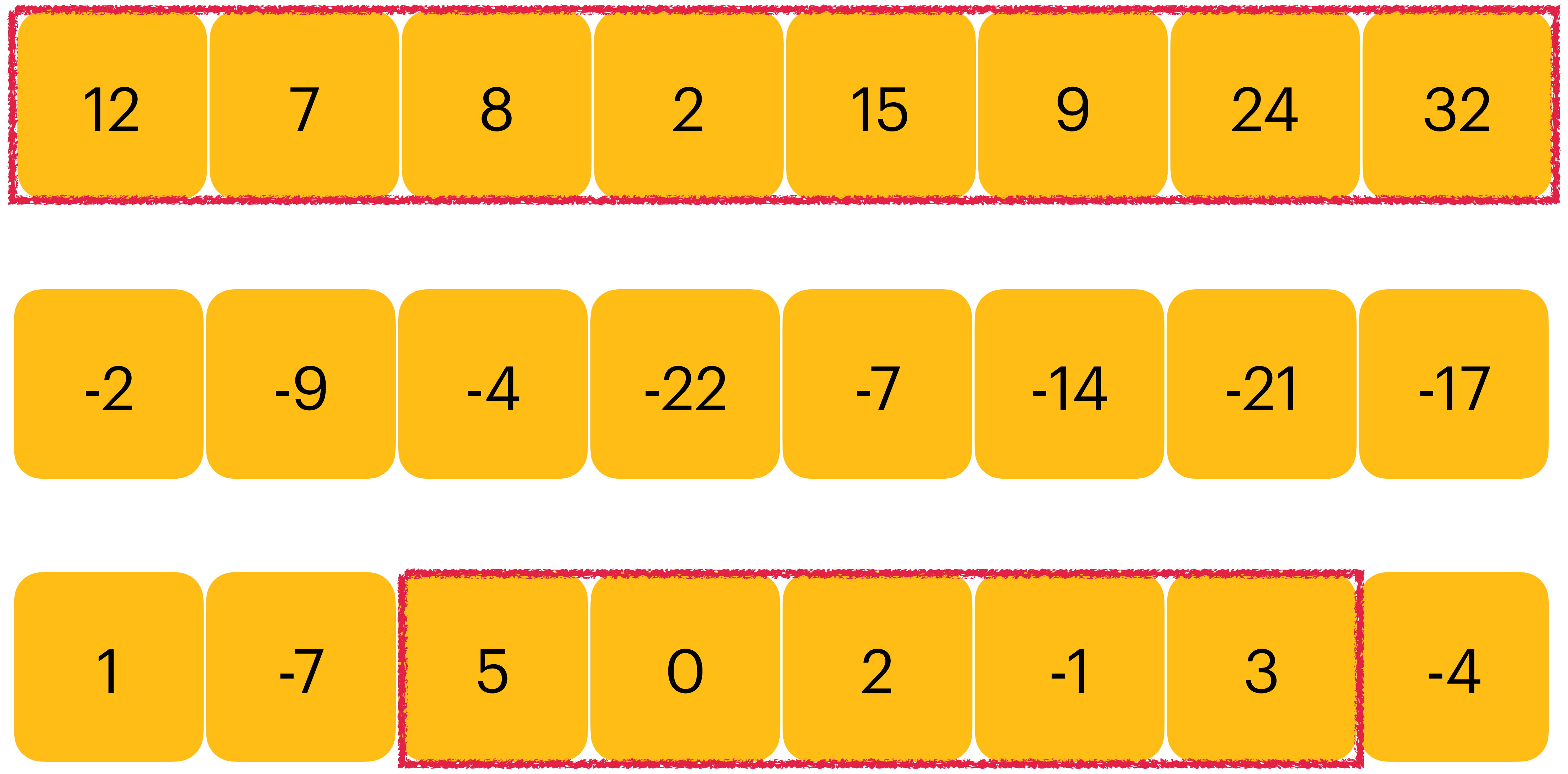
Example: Maximum Subarray Sum

Given an array $A[1,\ldots,n]$, find consecutive entries of A that yield the maximum sum.

Example 1	12	7	8	2	15	9	24	32
Example 2	-2	-9	-4	-22	-7	-14	-21	-17
Example 3	1	-7	5	0	2	-1	3	-4

Example: Maximum Subarray Sum

Given an array $A[1,\ldots,n]$, find consecutive entries of A that yield the maximum sum.



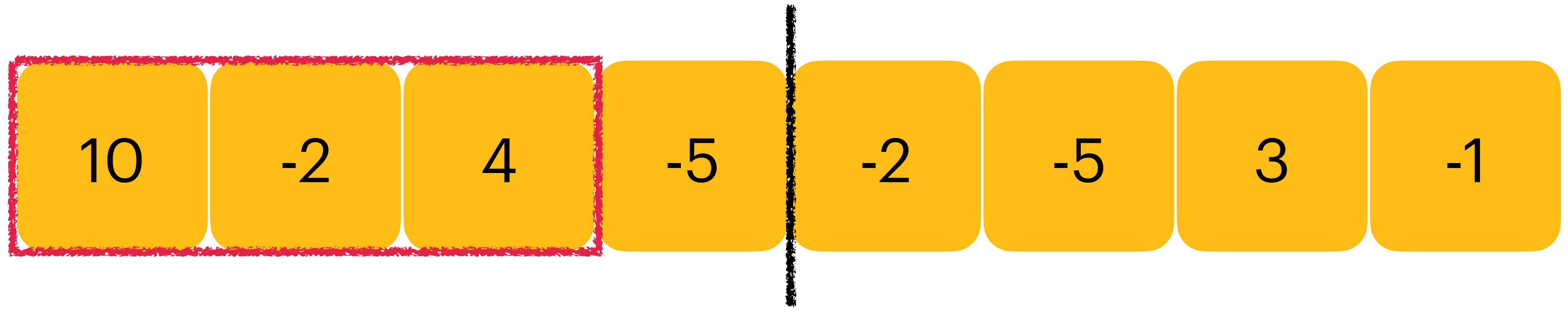
Example 1	12	7	8	2	15	9	24	32
Example 2	-2	-9	-4	-22	-7	-14	-21	-17
Example 3	1	-7	5	0	2	-1	3	-4

What are some naive approaches?

Example: Maximum Subarray Sum

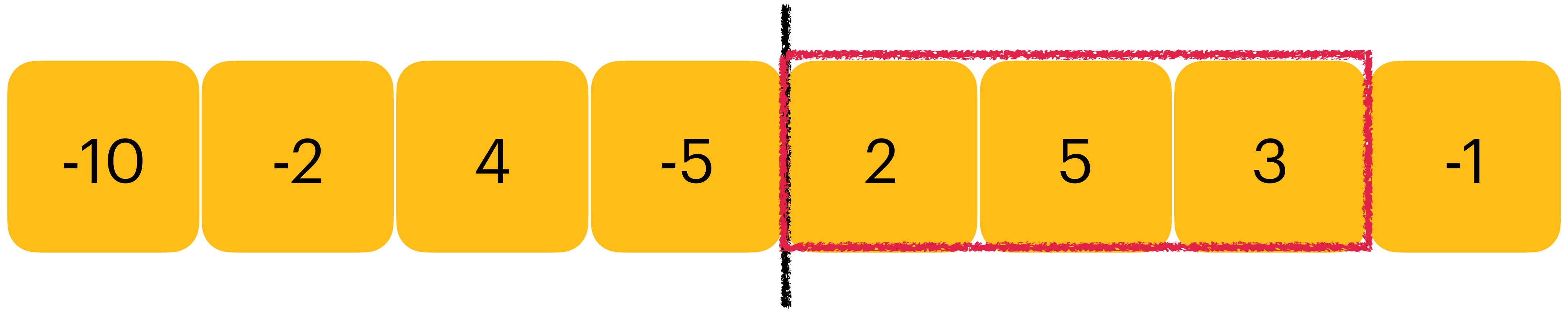
Divide-and-Conquer

Case 1: Best solution is entirely in the left subarray



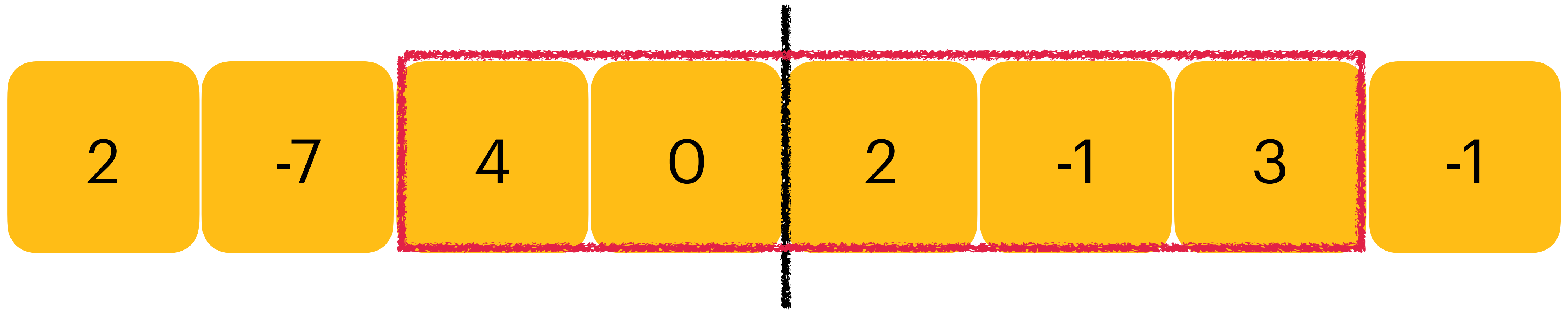
10 -2 4 -5 -2 -5 3 -1

Case 2: Best solution is entirely in the right subarray



-10 -2 4 -5 2 5 3 -1

Case 3: Best solution *crosses* the partition



2 -7 4 0 2 -1 3 -1

Example: Maximum Subarray Sum

Divide-and-Conquer

1. Recurse:
   - $maxL \leftarrow$ best solution left of partition
   - $maxR \leftarrow$ best solution right of partition
2. Compute best solution $maxM$ crossing partition
3. Return the best of $maxL, maxR, maxM$

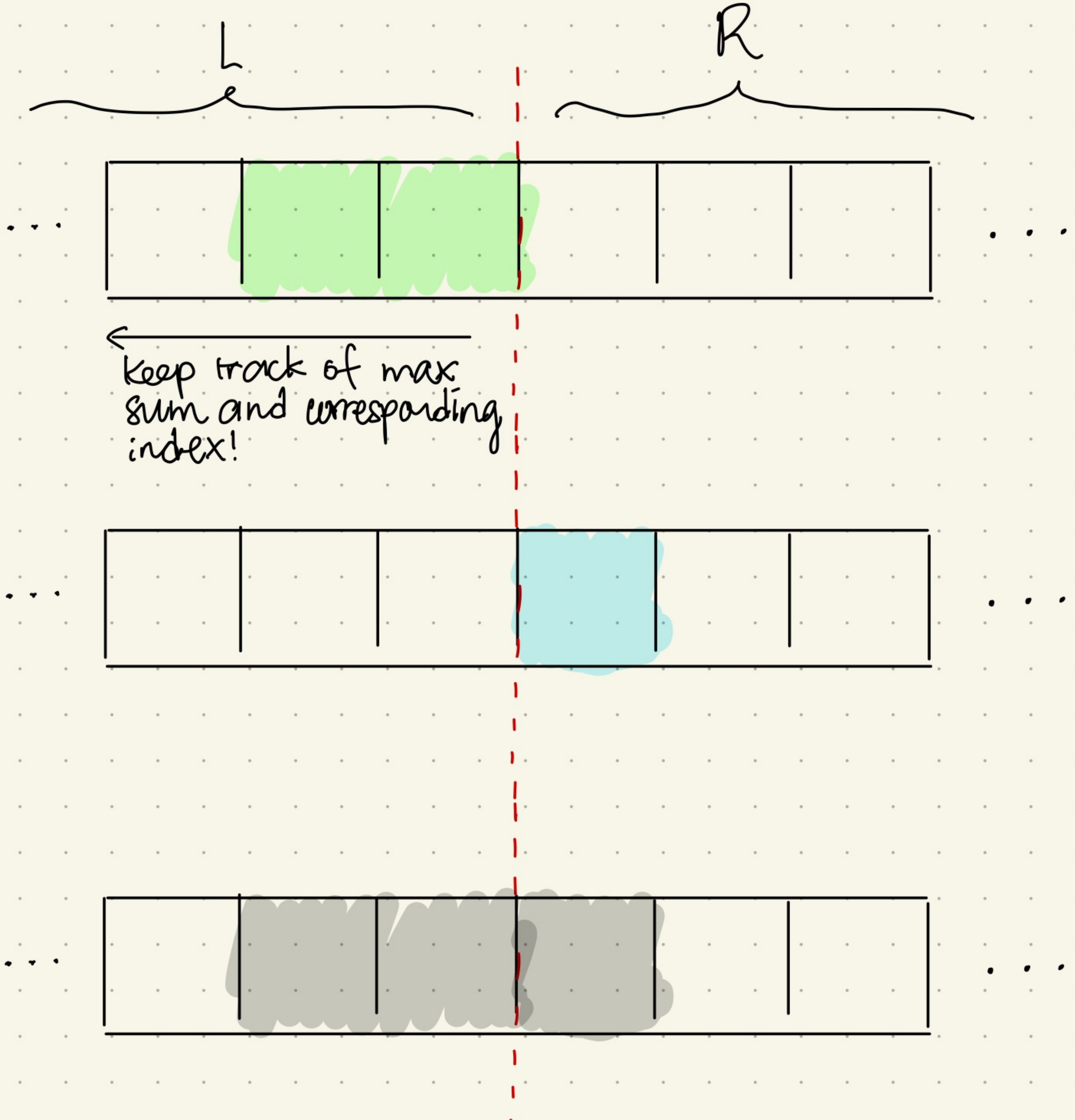
Claim: Step 2 (computing *maxM*) can be performed in $\Theta(n)$ time. How?

Compute maxM in $\Theta(n)$ Time

1. Find max-sum subsequence in L right-aligned at the partition.

2. Find max-sum subsequence in R left-aligned at the partition.

3. Return the union of the two max-sum sequences.



Keep track of max sum and corresponding index!

Techniques for Solving Recurrences

Recurrence tree: Section 3.5: Practice Problems Q4b; Appendix A

Unrolling: Section 3.1 $T(n) = 2T(\frac{n}{2}) + \Theta(\frac{n}{\log n})$; Appendix B

Substitution: Section 3.2; Section 3.5 Q3

Master Theorem: Section 3.1 (note when it *cannot* be used); Section 3.5 Q1, Q2

Guess-and-check: Section 3.3 (note common mistakes); Section 3.5 Q4a, Q5, Q6, Q7

Master Theorem

Recurrence: $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1, b > 1$

Idea: Compare weight at the root of recurrence tree, $f(n)$, to the # of leaves, $n^{\log_b a}$

Three cases depending on the relationship between our "benchmark" value $n^{\log_b a}$ and $f(n)$:

1. If $f(n)$ is polynomially less than $n^{\log_b a}$ (i.e., $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$), then $T(n) = \Theta(n^{\log_b a})$

2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$, then $T(n) = \Theta(f(n) \log n)$

3. If $f(n)$ is polynomially greater than $n^{\log_b a}$ (i.e., $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$), then $T(n) = \Theta(f(n))$

Substitution: Worked Example (Section 3.2)

uh oh!

$$T(n) = 2T(\sqrt{n}) + 1.$$

- Define m s.t. $n = 2^m$
- Substitute: $T(2^m) = 2T(2^{\frac{m}{2}}) + 1$
- Create new function $S(m) = T(2^m)$

$$\Rightarrow \quad S(m) = 2S(\tfrac{m}{2}) + 1 = O(m)$$

Master Theorem

$a = 2$, $b = 2$, so $n^{\log_b a} = n$

$f(n) = 1.$

$\Rightarrow$ $\boxed{\text{Case } 1.}$

- Resubstitute: $T(2^m) = O(m)$

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

$m = \lg n$

Median Finding

Given a set S of n distinct elements and a number $i \in \{1,2,\ldots,n\}$, find the element $x \in S$ such that $\mathrm{RANK}(x) = i$, that is, the ith smallest element.

Median Finding

Given a set S of n distinct elements and a number $i \in \{1,2,\ldots,n\}$, find the element $x \in S$ such that $\text{RANK}(x) = i$, that is, the ith smallest element.

8	-1	19	11	12	0	-7	3	-4

Median Finding

Given a set S of n distinct elements and a number $i \in \{1,2,\ldots,n\}$, find the element $x \in S$ such that $\text{RANK}(x) = i$, that is, the ith smallest element.

8	-1	19	11	12	0	-7	3	-4

-7	-4	-1	0	3	8	11	12	19

Median Finding

Given a set S of n distinct elements and a number $i \in \{1,2,\ldots,n\}$, find the element $x \in S$ such that $\text{RANK}(x) = i$, that is, the ith smallest element.

8	-1	19	11	12	0	-7	3	-4

-7	-4	-1	0	3	8	11	12	19

Median Finding

Given a set S of n distinct elements and a number $i \in \{1,2,\ldots,n\}$, find the element $x \in S$ such that $\text{RANK}(x) = i$, that is, the ith smallest element.

8	-1	19	11	12	0	-7	3	-4
-7	-4	-1	0	3	8	11	12	19

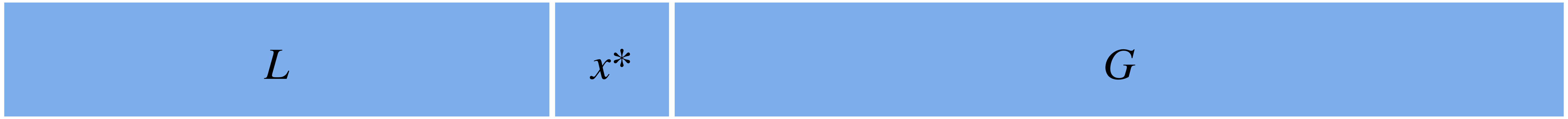
finding the median via sorting requires $O(n \log n)$ time

can we do better?

Algorithm Sketch

1. Pick a pivot $x^* \in S$ cleverly

2. Compute $L = \{y \in S : y < x^*\}$ and $G = \{y \in S : y > x^*\}$



L | x^* | G

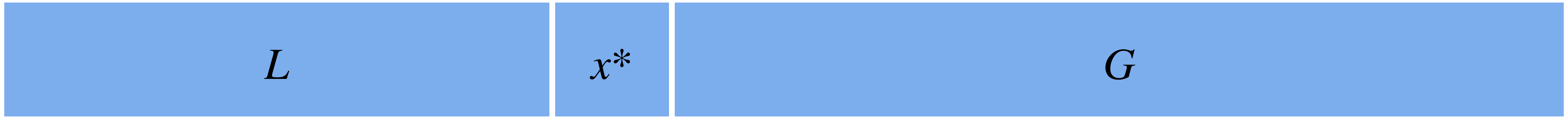
3. Since so $\text{RANK}(x^*) = |L| + 1$:

- If $\text{RANK}(x^*) = i$, then $x = x^*$

- If $\text{RANK}(x^*) < i$, then recurse on L

- If $\text{RANK}(x^*) > i$, then recurse on G

Analysis



L x^* G

Bad: If $|L| = 0$ at each level, then

$$T(n) = T(n-1) + O(n) = O(n^2)$$

Good: If $|L|, |G| \leq cn$ for some constant $c < 1$ at each level, then:

$$T(n) = T(cn) + O(n) = O(n)$$

Goal: In $O(n)$ time, pick an x^* that is "c-balanced":

$$\max\{\text{RANK}(x^*), n - \text{RANK}(x^*)\} \leq cn$$

Algorithm

SELECT(i, S)

```
1. Divide $S$ into $\frac{n}{5}$ groups of 5 elements each, padded by large numbers, if necessary
2. Find the median of each 5-element group by sorting
3. **Recursively** SELECT the median $x^*$ of the $\frac{n}{5}$ group medium as the pivot
4. Compute $L = \{y \in S : y < x^*\}$ and $G = \{y \in S : y > x^*\}$
5. Since RANK$(x^*) = |L| + 1$:
    • If RANK$(x^*) = i$, then $x = x^*$
    • If RANK$(x^*) > i$, then SELECT$(i, L)$
    • If RANK$(x^*) < i$, then SELECT$(i - |L| - 1, G)$
```

"clever" selection

as before (pg.2)

Algorithm

SELECT(i, S)

1. Divide S into $\frac{n}{5}$ groups of 5 elements each, padded by large numbers, if necessary
2. Find the median of each 5-element group by sorting

$O(n)$

3. **Recursively** SELECT the median x^* of the $\frac{n}{5}$ group medium as the pivot

$T\left(\frac{n}{5}\right)$

4. Compute $L = \{y \in S : y < x^*\}$ and $G = \{y \in S : y > x^*\}$

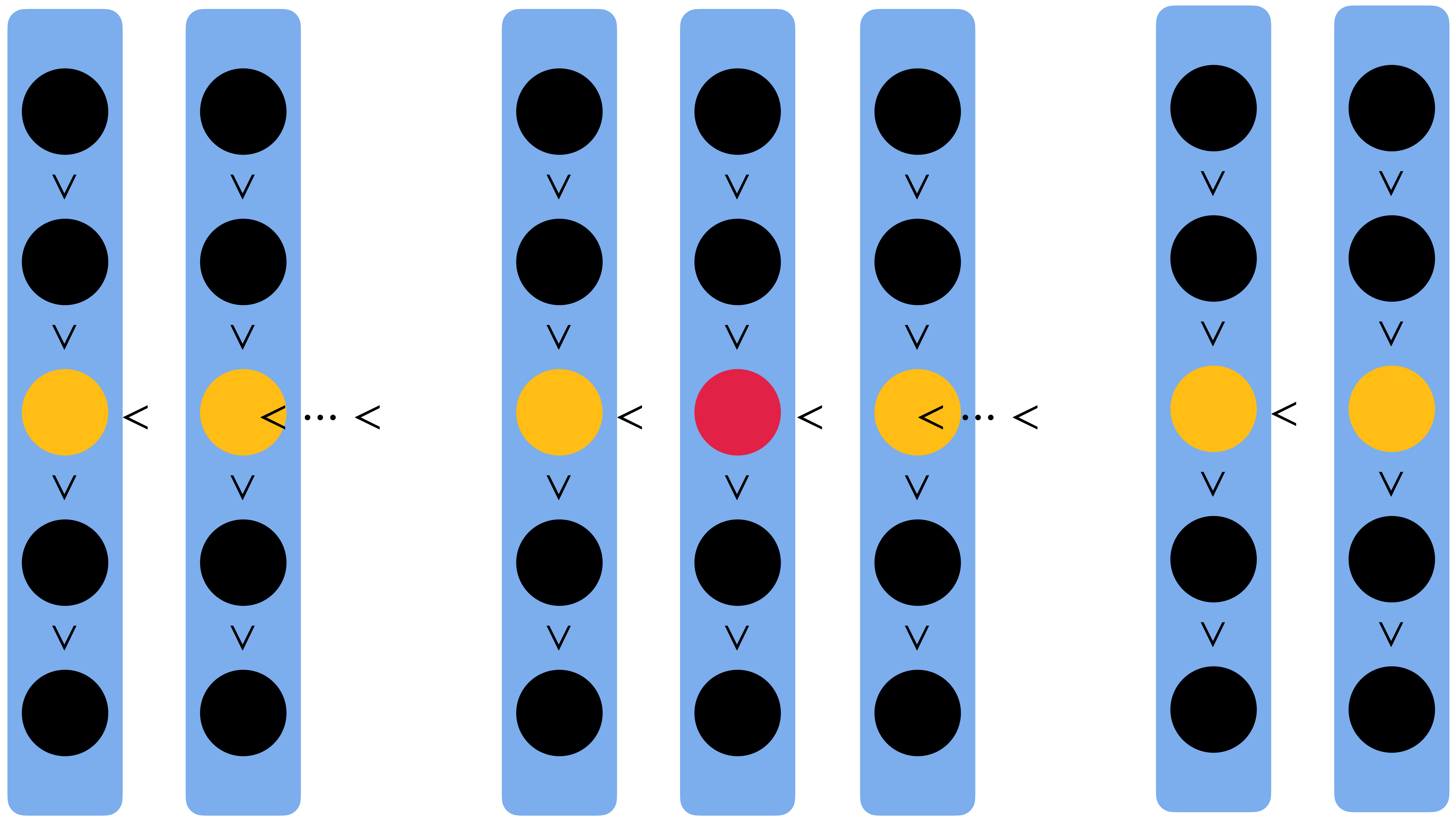
$O(n)$

5. Since $\text{RANK}(x^*) = |L| + 1$:
   - If $\text{RANK}(x^*) = i$, then $x = x^*$
   - If $\text{RANK}(x^*) > i$, then $\text{SELECT}(i, L)$ — $T(|L|)$
   - If $\text{RANK}(x^*) < i$, then $\text{SELECT}(i - |L| - 1, G)$ — $T(|G|)$

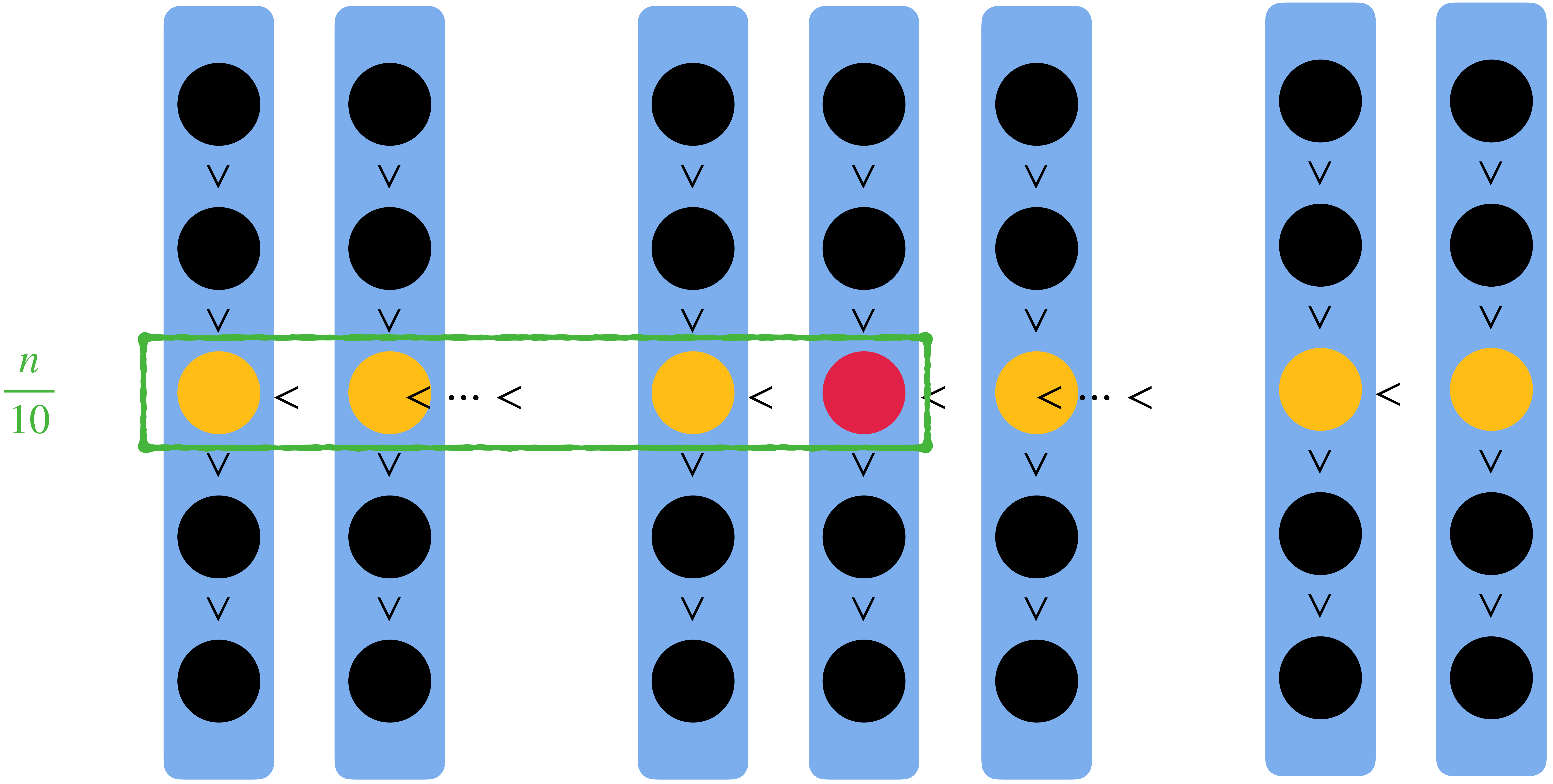
Analysis

Claim: Pivot x^* is $\frac{7}{10}$-balanced, that is, $\max\{\,|L|, |G|\,\} \leq \frac{7n}{10}$.

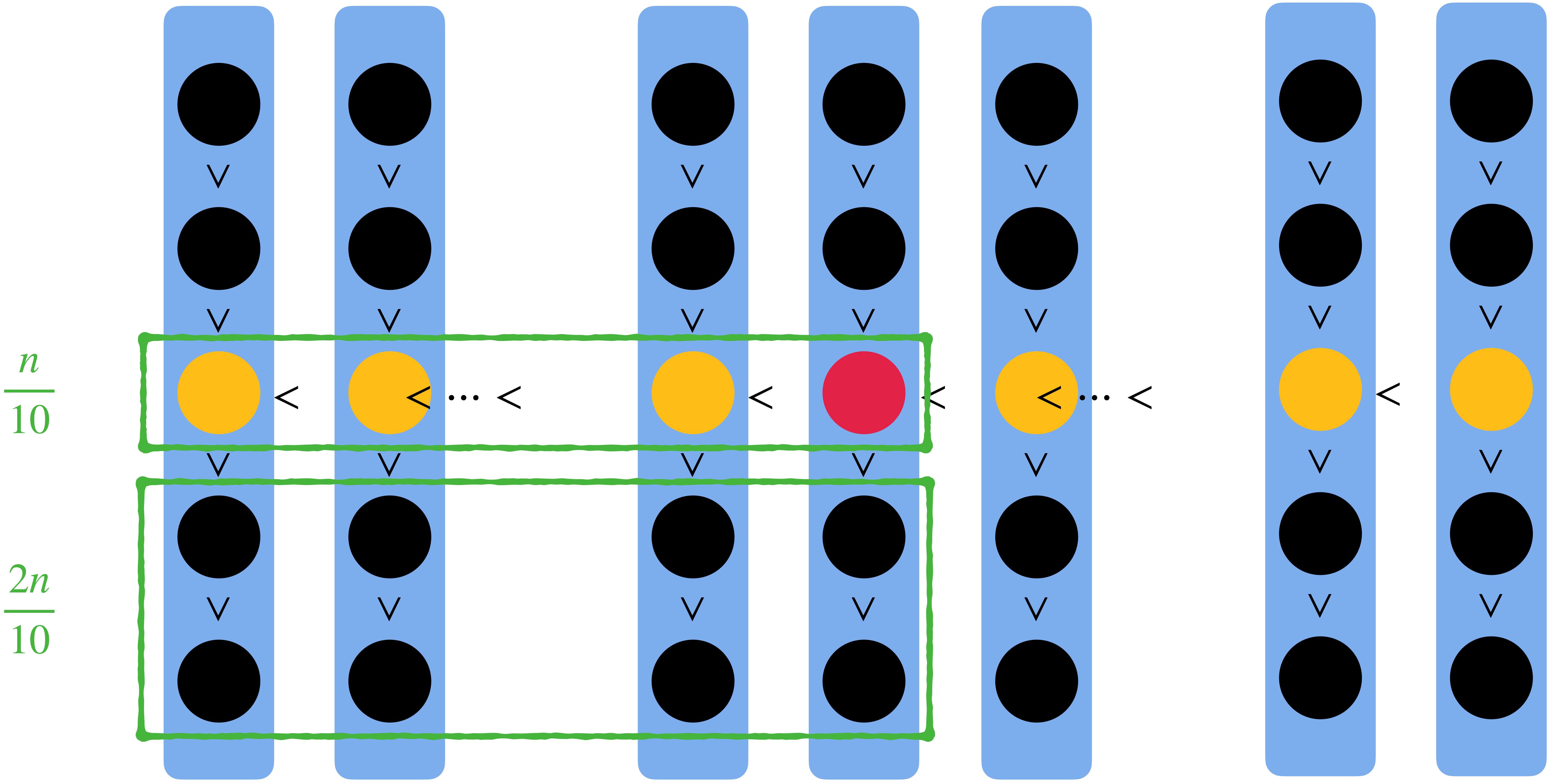
$\frac{n}{5}$ groups



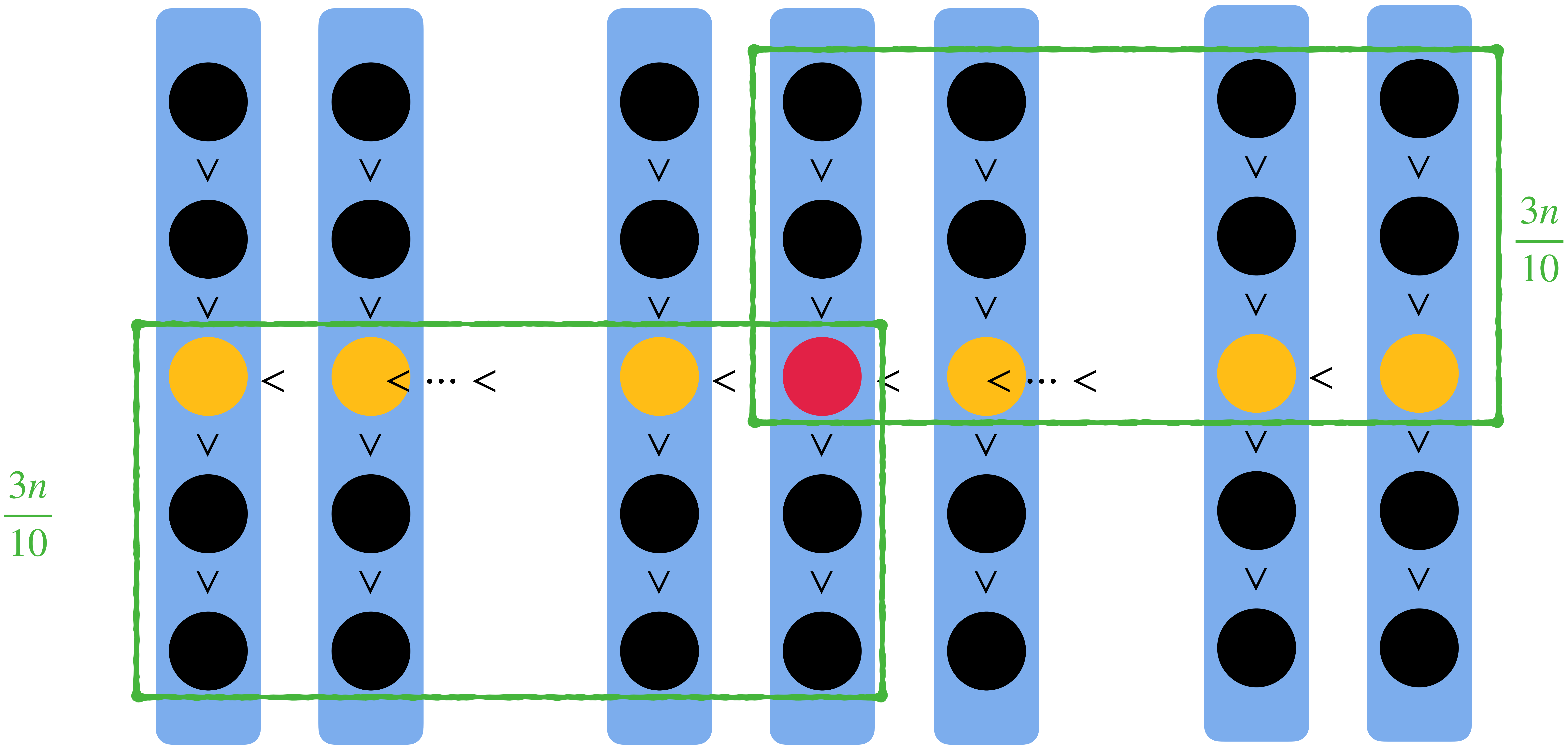
$\frac{n}{5}$ groups



$\frac{n}{5}$ groups



$\frac{n}{5}$ groups



Analysis

Claim: x^* **is** $\frac{7}{10}$**-balanced, that is** $\max\{\,|L|, |G|\,\} \le \frac{7n}{10}$**.**

$$|L| + 1 \ge \frac{3n}{10} \implies |G| \le \frac{7n}{10}$$
$$|G| + 1 \ge \frac{3n}{10} \implies |L| \le \frac{7n}{10}$$

Recurrence: $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$

What if we had chosen groups of $\frac{n}{3}$ elements? How about groups of $\frac{n}{7}$ elements?

Closing Thoughts

- Review recitation notes:
  - *Lots* of practice problems, e.g., Section 3.5, Appendix AB, etc.
  - Recap of lecture (Section 4)
  - Asymptotic Notation Reference (Appendix C)
- Watch out for my Canvas note:
  - Link to recitation slides
  - Form for anonymous feedback
- Probability review this Sunday 2/9
- Email: ryelin@mit.edu