Design & Analysis of Algorithms

Selection & Substitution Method

Prof. Wilson Rivera

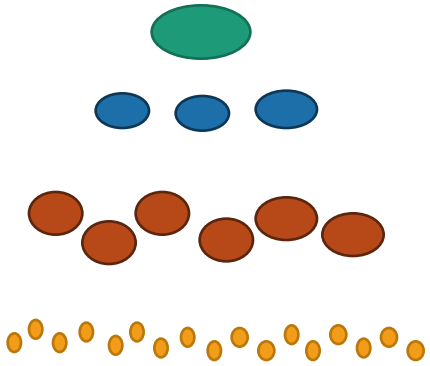
Spring 2018

Outline

- Master Theorem problem
- Selection Problem
- Linear time selection
- Substitution Method

Master Theorem

Work at level t: $O(n^d (a/b^d)^t)$



- Case 1: $a = b^d$
  - The recursion tree has the same amount of work at every level. (Like MergeSort).
- Case 3: $a > b^d$
  - The tree branches really quickly compared to work per problem! The bulk of the work is done at the bottom of the tree. (Like Karatsuba).
- Case 2: $a < b^d$
  - The work done shrinks way faster than we branch new problems. The bulk of the work is done at the root of the tree. (We haven't seen this yet but we will today).

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

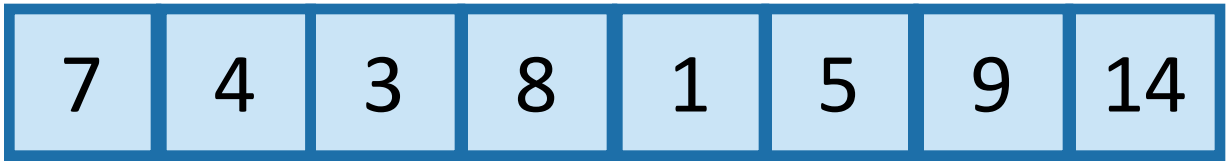
Master Theorem

- The Master Theorem only works when all sub-problems are the same size.
- That's not always the case.
- Today we'll see an example where the Master Theorem won't work.
- We'll use something called the substitution method instead.

Selection

A is an array of size n, k is in {1,…,n}

- SELECT(A, k):
  - Return the k'th smallest element of A.



- SELECT(A, 1) = 1
- SELECT(A, 2) = 3
- SELECT(A, 3) = 4
- SELECT(A, 8) = 14

SELECT(A, k); k=1

- Let's start with MIN(A) aka SELECT(A, 1).
- MIN(A):
  - ret = ∞
  - **For** i=1, .., n:
    - If A[i] < ret:
      - ret = A[i]
  - **Return** ret

This stuff is O(1)

This loop runs O(n) times

- Time O(n).

Lecture 4:

[8, 7, 3, 4]

$i = 1$

$A[1] < ret \;\&\; A[1] < min$

$ret = \infty$

$min = 8$

$i = 2$

$A[2] < ret \;\&\; A[2] < min$ $O(n)$

$ret = 8$

$min = 7$

$i = 3$

$A[3] < ret \;\&\; A[3] < min$

$ret = 7$

$min = 3$

$i = 4$

$A[4] < ret \;\&\; A[4] < min$ False

$A[4] < ret \;\&\; A[4] \geq min$

$ret = A[4]$

How about SELECT(A,2)?

- SELECT2(A):
  - ret = ∞
  - minSoFar = ∞
  - **For** i=1, .., n:
    - If A[i] < ret and A[i] < minSoFar:
      - ret = minSoFar
      - minSoFar = A[i]
    - Else if A[i] < ret and A[i] >= minSoFar:
      - ret = A[i]
  - **Return** ret

Still O(n)

SO FAR SO GOOD.

SELECT(A, n/2) aka MEDIAN(A)?

- MEDIAN(A):
  - ret = ∞
  - minSoFar = ∞
  - secondMinSoFar = ∞
  - thirdMinSoFar = ∞
  - fourthMinSoFar = ∞
  - ….
- This is not a good idea for large k (like n/2 or n).
- Basically this is just going to turn into something like INSERTIONSORT…and that was $O(n^2)$.

A much better idea for large k

- SELECT(A, k):
  - A = MergeSort(A)
  - **return** A[k]

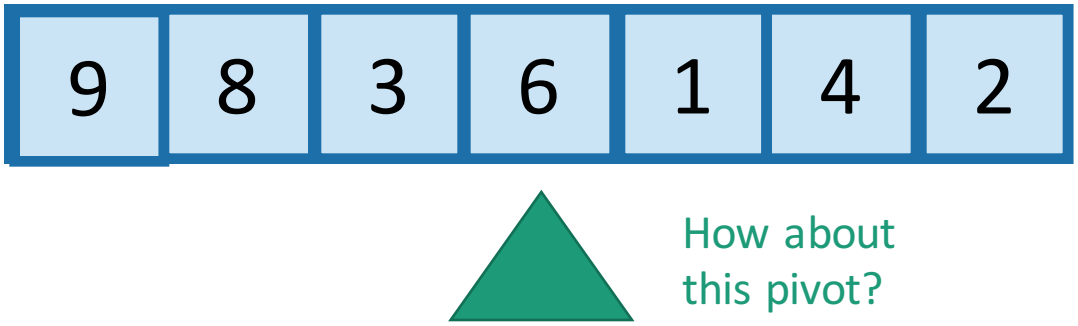
- Running time is O(n log(n)).
- So that's the benchmark….

Can we do better?

We're hoping to get O(n)

Idea: recursion

Say we want to find SELECT(A, k)



First, pick a "pivot." We'll see how to do this later.

Next, partition the array into "bigger than 6" or "less than 6"

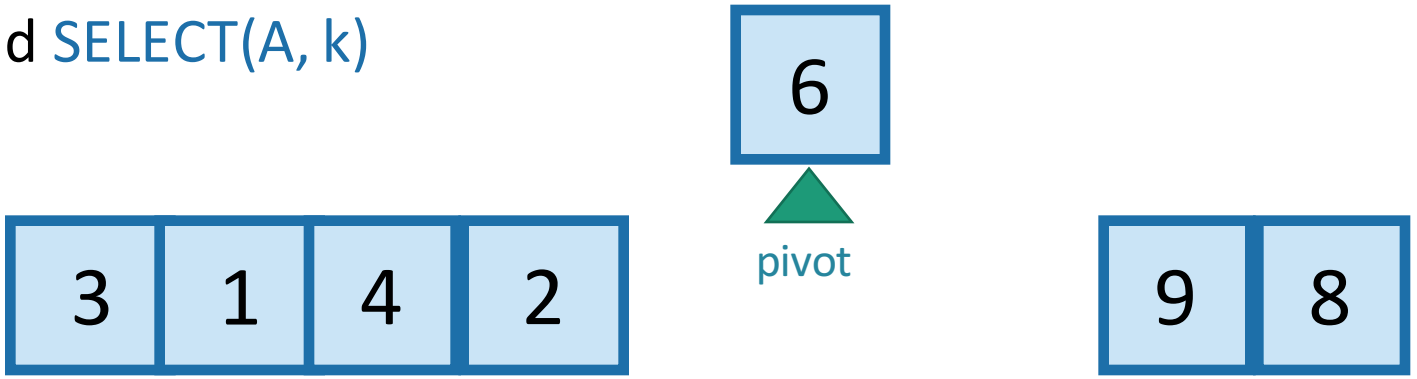
This PARTITION step takes time O(n). (Notice that we don't sort each half).

L = array with things smaller than A[pivot]

R = array with things larger than A[pivot]

Idea continued…

Say we want to find SELECT(A, k)



L = array with things smaller than A[pivot]

R = array with things larger than A[pivot]

- If k = 5 = len(L) + 1:
  - We should return A[pivot]
- If k < 5:
  - We should return SELECT(L, k)
- If k > 5:
  - We should return SELECT(R, k – 5)

This suggests a recursive algorithm

(still need to figure out how to pick the pivot…)

Let's make that a bit more formal

- PARTITION(A, p):
  - L = new array
  - R = new array
  - **For** i=1,…,n:
    - **If** i==p:
      - continue
    - **Else if** A[i] <= A[p]:
      - L.append(A[i])
    - **Else if** A[i] > A[p]:
      - R.append(A[i])
  - **Return** L, A[p], R

- This is the O(n) PARTITION algorithm that we saw before.

Selection(A, k)

- SELECT(A, k):
  - Choose p in {1,…,n}
  - L, A[p], R = PARTITION(A,p)
  - **If** len(L) = k - 1:
    - **Return** A[p]
  - **Else If** len(L) > k - 1:
    - **Return** SELECT(L, k)
  - **Else if** len(L) < k - 1:
    - return SELECT(R, k – len(L) – 1)

- PARTITION(A, p):
  - L = new array
  - R = new array
  - **For** i=1,…,n:
    - **If** i==p:
      - continue
    - **else If** A[i] <= A[p]:
      - L.append(A[i])
    - **Else if** A[i] > A[p]:
      - R.append(A[i])
  - **Return** L, A[p], R

Correctness

- SELECT(A, p=k):
  - Choose p in {1,…,n}
  - L, A[p], R = PARTITION(A,p)
  - **If** len(L) = k - 1:
    - **Return** A[p]
  - **Else If** len(L) > k - 1:
    - **Return** SELECT(L, k)
  - **Else if** len(L) < k - 1:
    - return SELECT(R, k – len(L) – 1)

- Recursion invariant:

At the return of each recursive call of size < n, SELECT(A,k) returns the k'th smallest element of A.

- Base case ("Initialization"):
  - Select (A,1) is correct.
- Inductive step: ("Maintenance")
  - Suppose that the recursion invariant holds for n.
  - Want to show that it holds for n + 1.
  - Three cases:
    - **if len(L) = k-1**, then A[p] is the correct thing to return.
    - **If len(L) > k-1**, then the k'th smallest element of L is the correct thing to return
      - And by induction, this is indeed what we return.
    - **If len(L) < k-1**, then the (k – (len(L) - 1)'st smallest elt of R is the correct thing to return.
      - And by induction, this is indeed what we return.

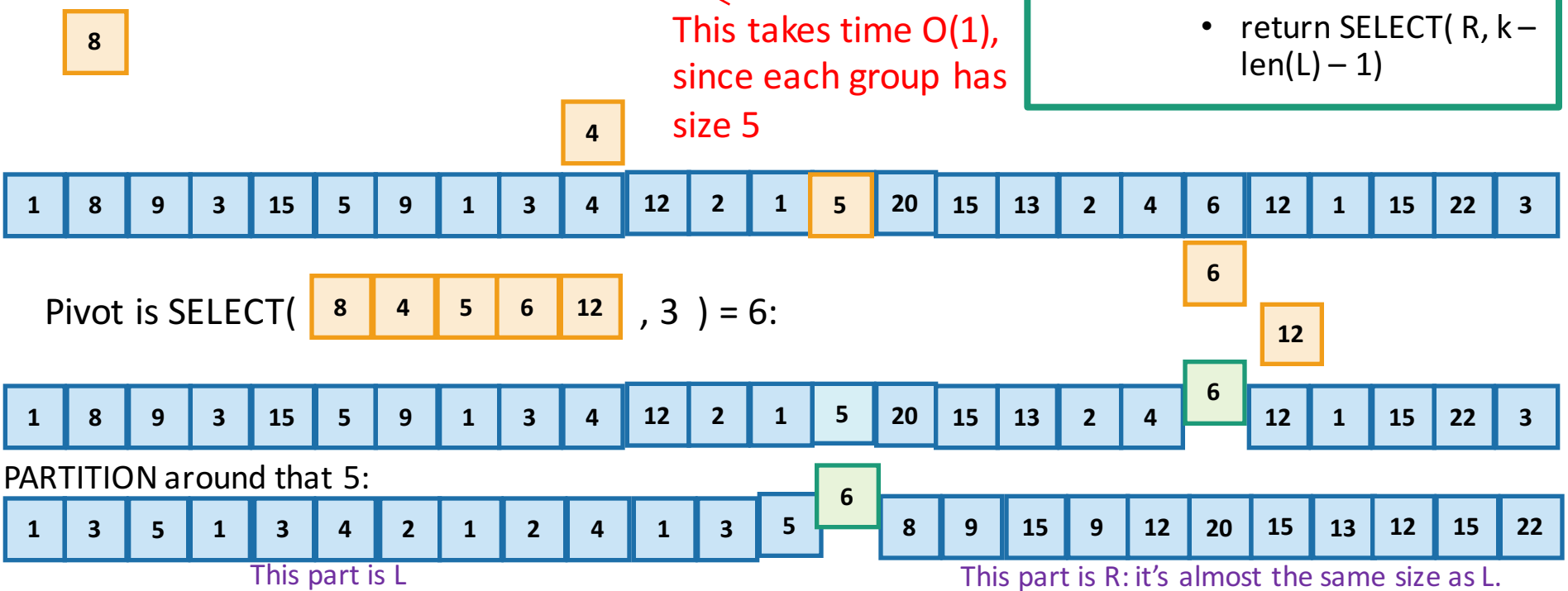
- Conclusion ("Termination")
- By induction, the recursion invariant holds for n + 1, which means that SELECT(A,k) is correct.

How to pick the pivot

- CHOOSEPIVOT(A):
  - Split A into m = $\lceil \frac{n}{5} \rceil$ groups, of size <=5 each.
  - **For** i=1, .., m:
    - Find the median within the i'th group, call it p_i
  - p = SELECT([$p_1, p_2, p_3, \ldots, p_m$] , m/2)
  - **return** p

- SELECT(A, p=k):
  - **If** len(A) <= 50:
    - A = MergeSort(A)
    - Return A[k]
  - p = CHOOSEPIVOT(A)
  - L, A[p], R = PARTITION(A,p)
  - **If** len(L) = k - 1:
    - **Return** A[p]
  - **Else If** len(L) < k - 1:
    - **Return** SELECT(L, k)
  - **Else if** len(L) > k - 1:
    - return SELECT(R, k – len(L) – 1)

This takes time O(1), since each group has size 5



So this gives the whole algorithm

- SELECT(A, p=k):
  - **If** len(A) <= 50:
    - A = MergeSort(A)
    - Return A[k]
  - p = CHOOSEPIVOT(A)
  - L, A[p], R = PARTITION(A,p)
  - **If** len(L) = k - 1:
    - **Return** A[p]
  - **Else If** len(L) > k - 1:
    - **Return** SELECT(L, k)
  - **Else if** len(L) < k - 1:
    - return SELECT(R, k – len(L) – 1)

- PARTITION(A, p):
  - L = new array
  - R = new array
  - **For** i=1,…,n:
    - **If** i==p, continue
    - **Else If** A[i] <= A[p]:
      - L.append(A[i])
    - **Else if** A[i] > A[p]:
      - R.append(A[i])
  - **Return** L, A[p], R

- CHOOSEPIVOT(A):
  - Split A into m = $\left\lceil \frac{n}{5} \right\rceil$ groups, of size <=5 each.
  - **For** i=1, .., m:
    - Find the median within the i'th group, call it p_i
  - p = SELECT([$p_1, p_2, p_3, \ldots, p_m$] , m/2)
  - **return** p

Lemma: that median-of-medians thing is a good idea.

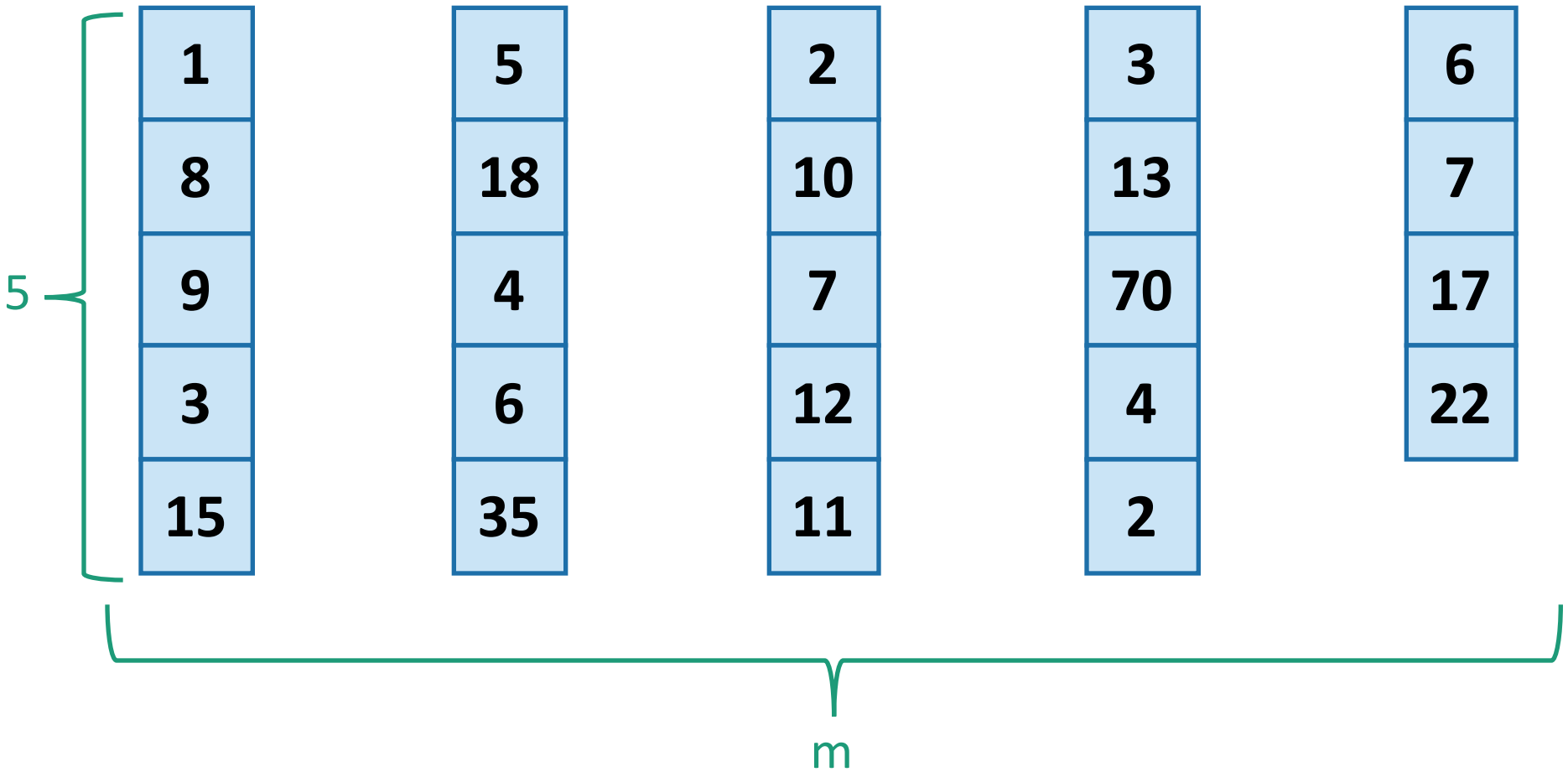
- **Lemma:** If L and R are as in the algorithm SELECT given above, then

$$|L| \leq \frac{7n}{10} + 5$$

and

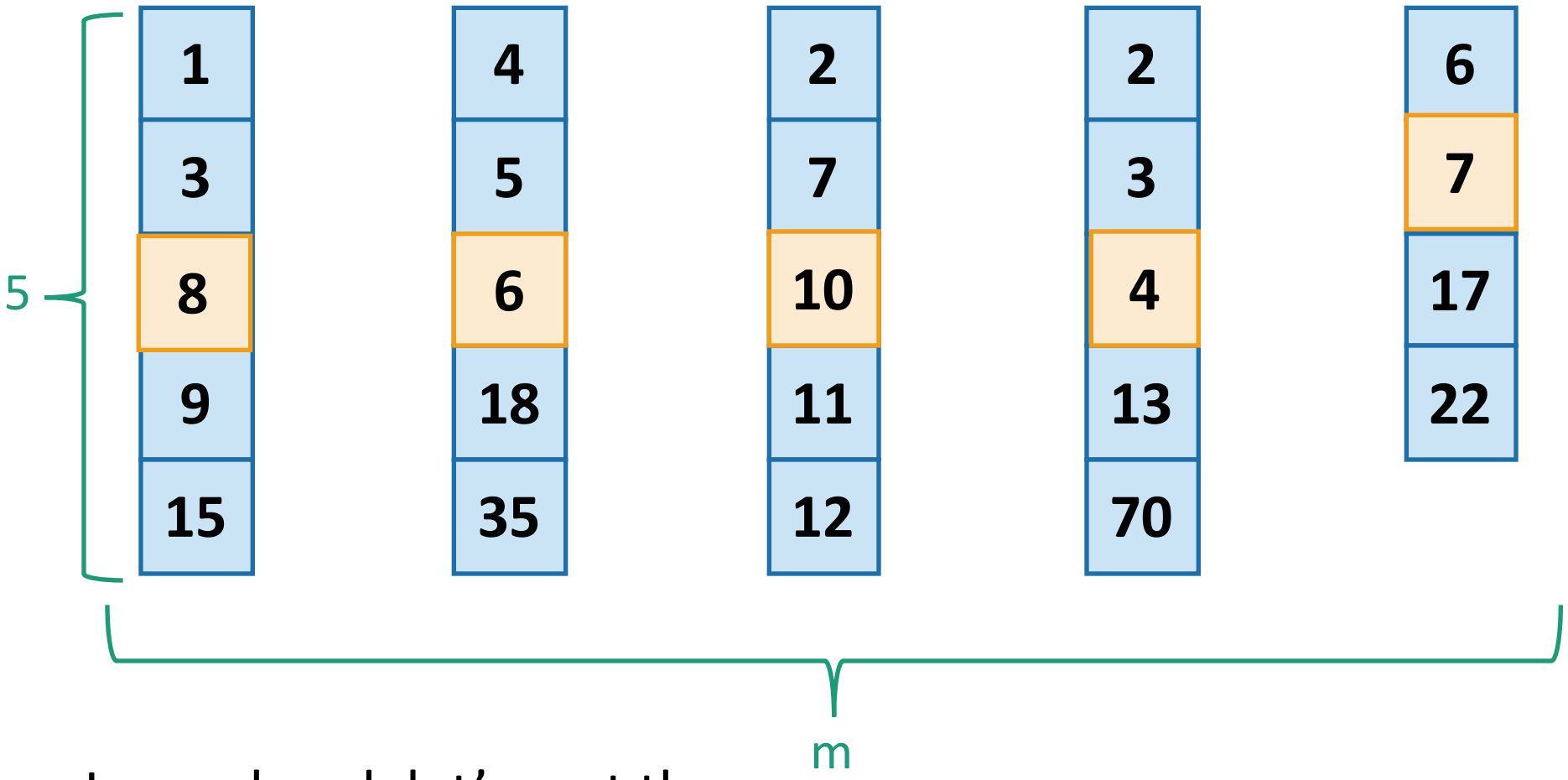
$$|R| \leq \frac{7n}{10} + 5$$

Proof by picture



Say these are our m = [n/5] sub-arrays of size at most 5.

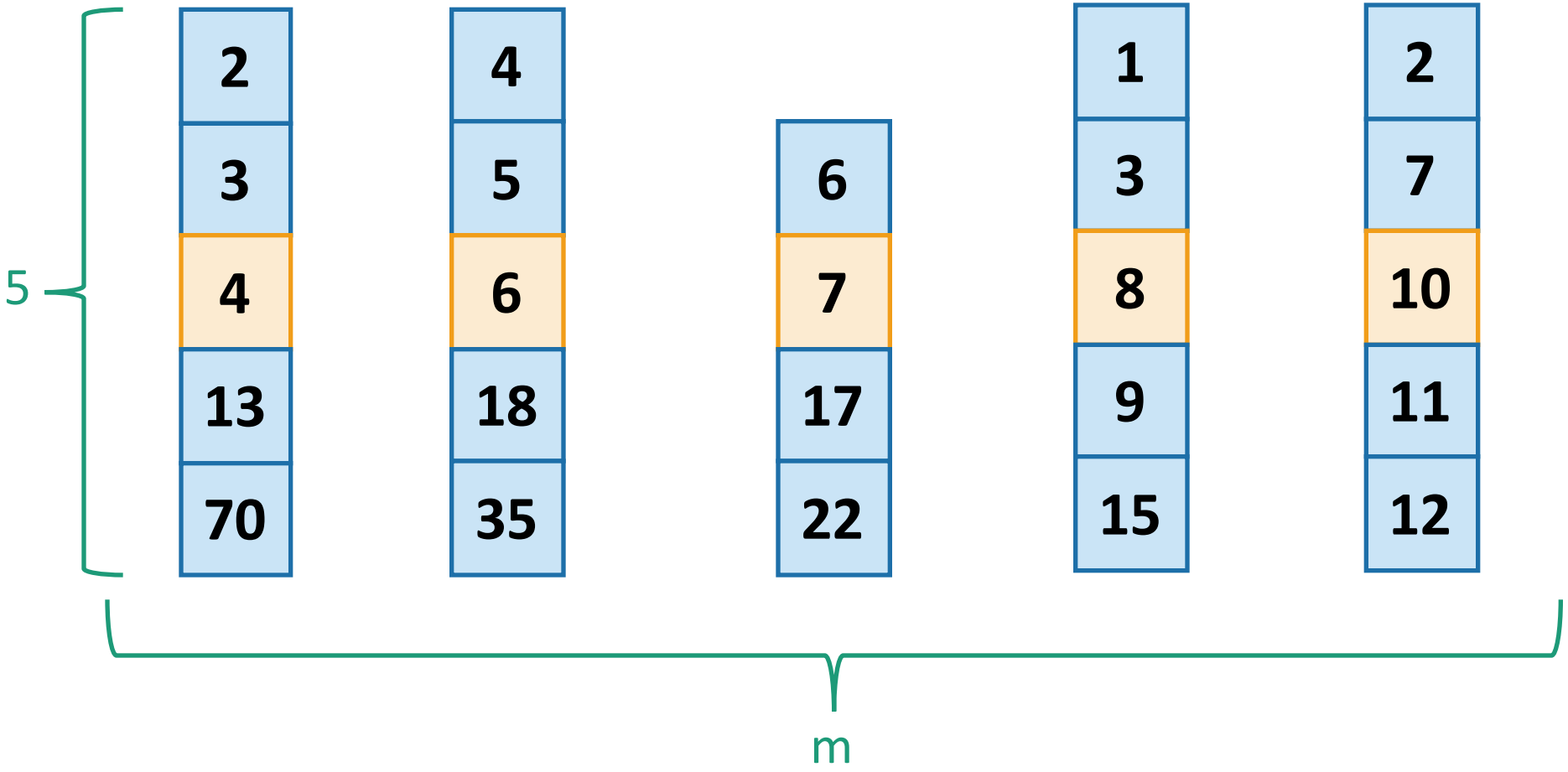
Proof by picture



In our head, let's sort them.

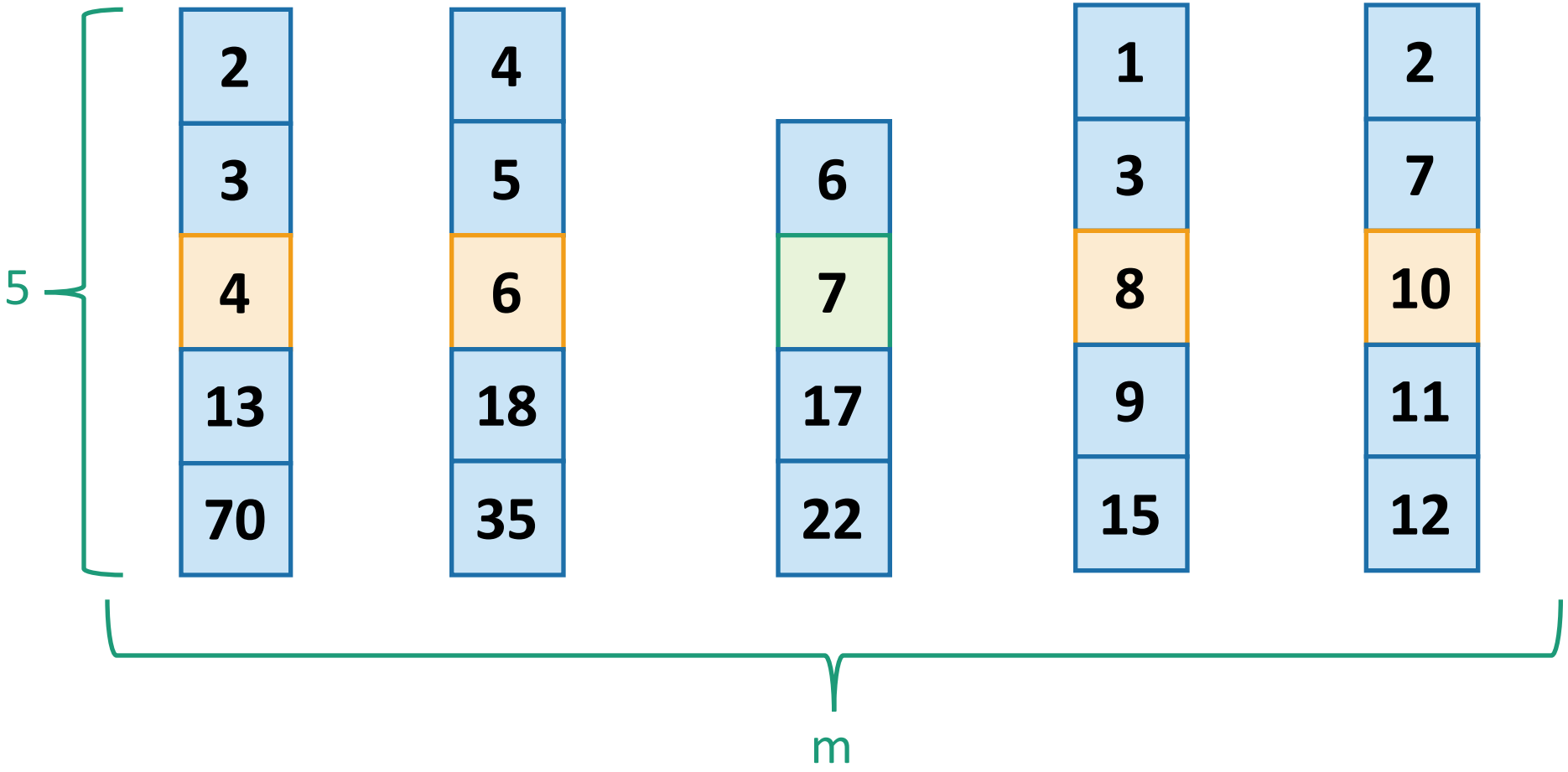
Then find medians.

Proof by picture



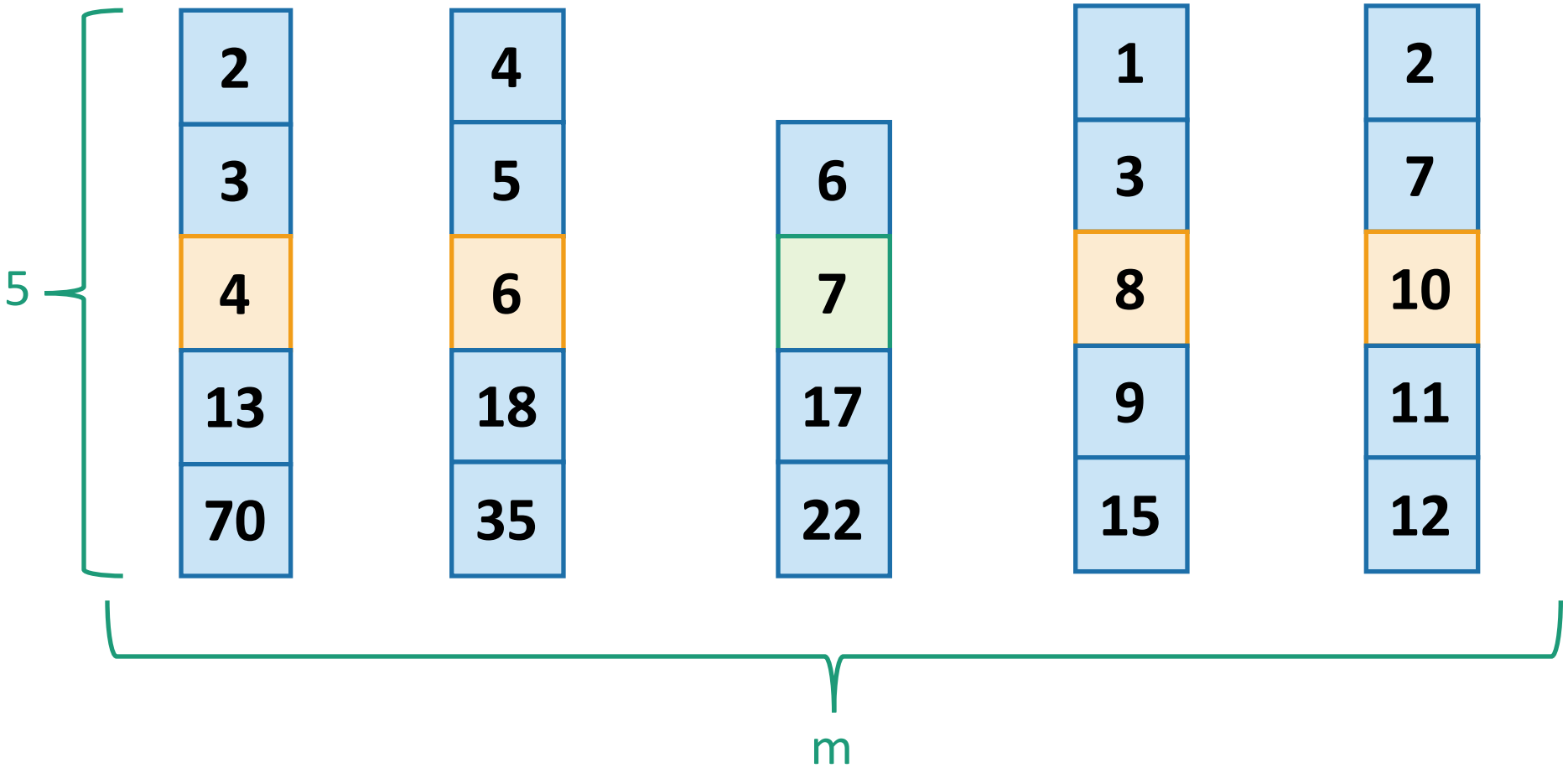
Then let’s sort them by the median

Proof by picture



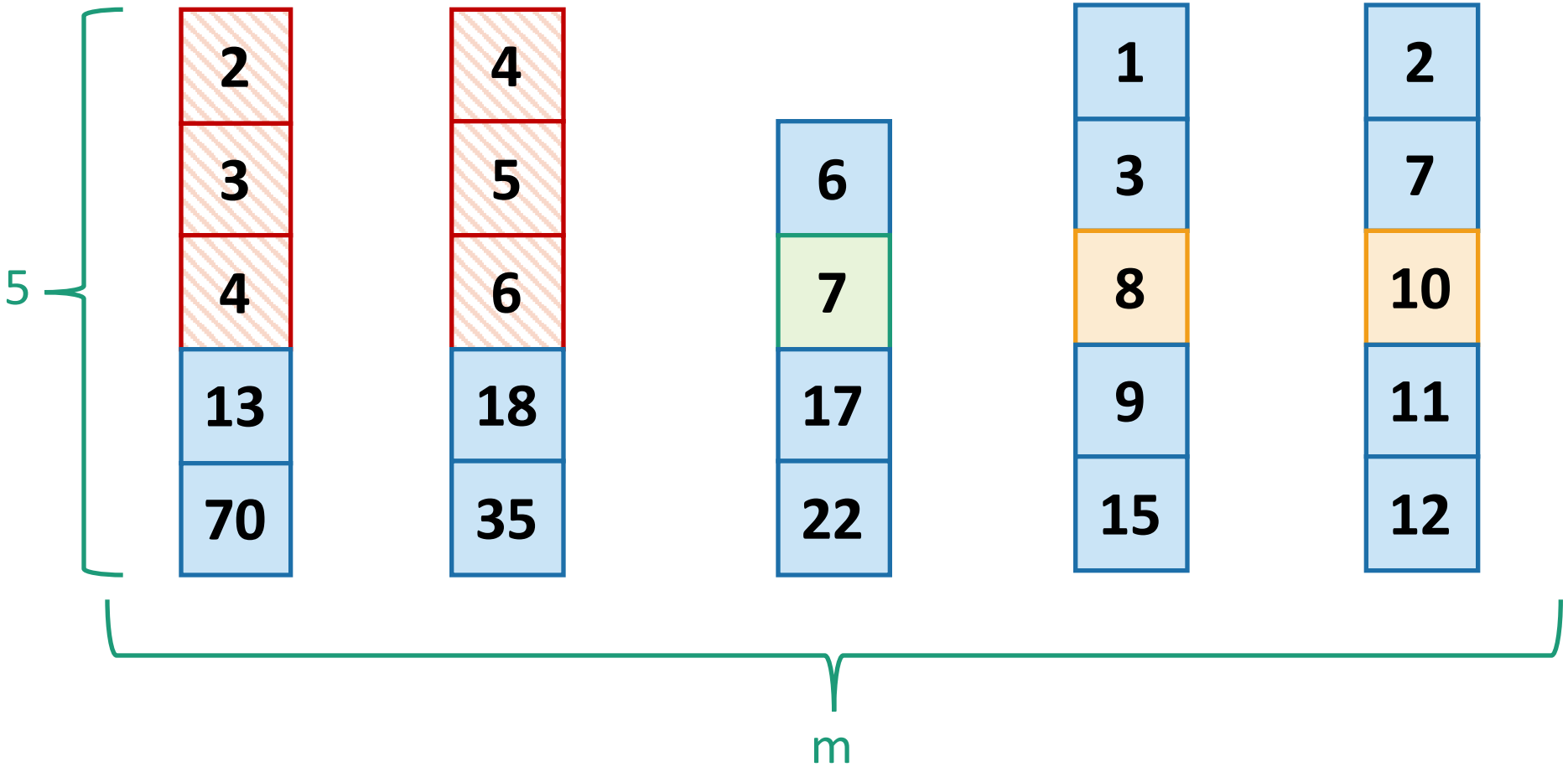
The median of the medians is 7. That's our pivot!

Proof by picture



How many elements are SMALLER than the pivot?

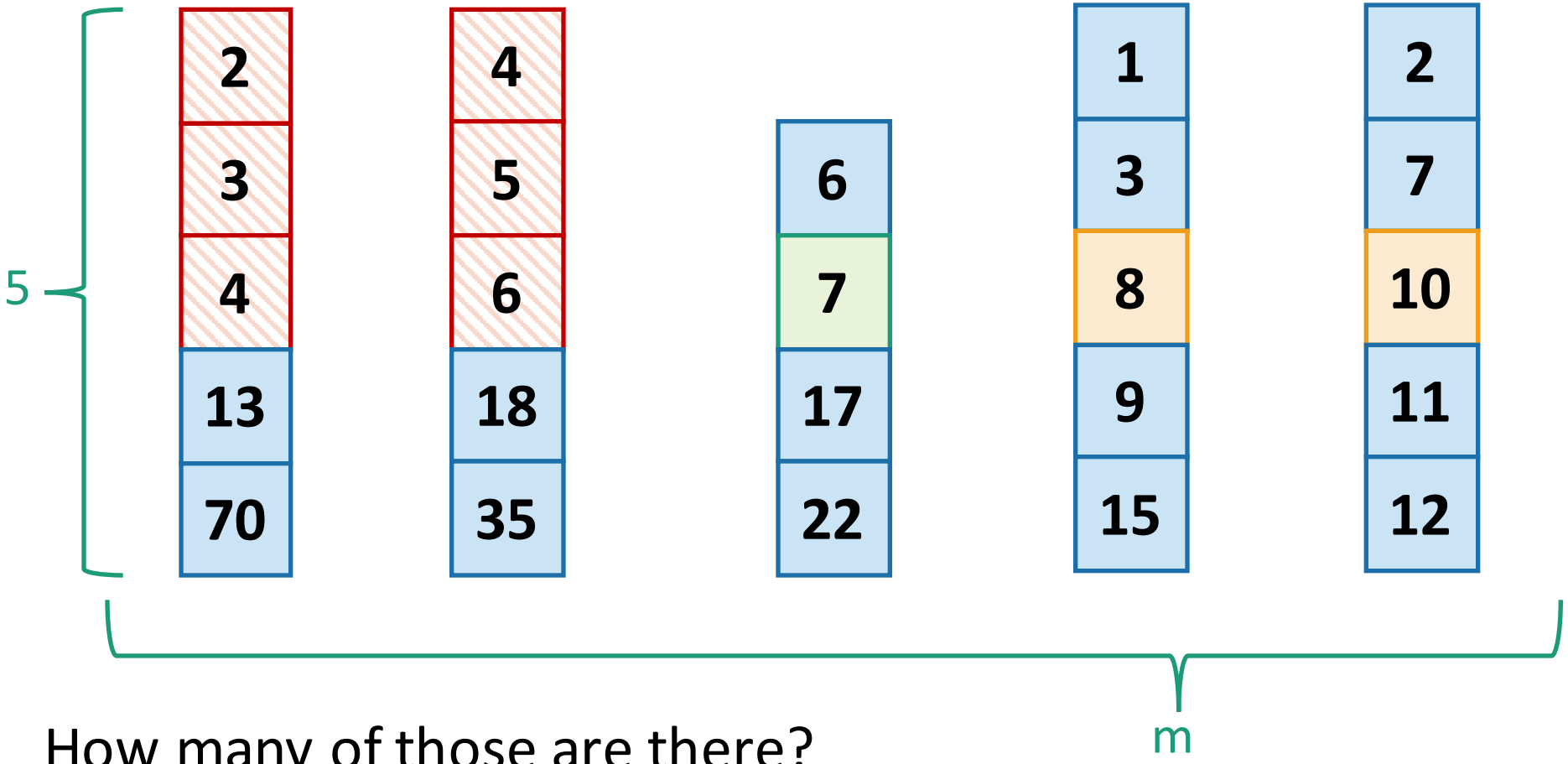
Proof by picture



At least these ones: everything above and to the left.

Proof by picture

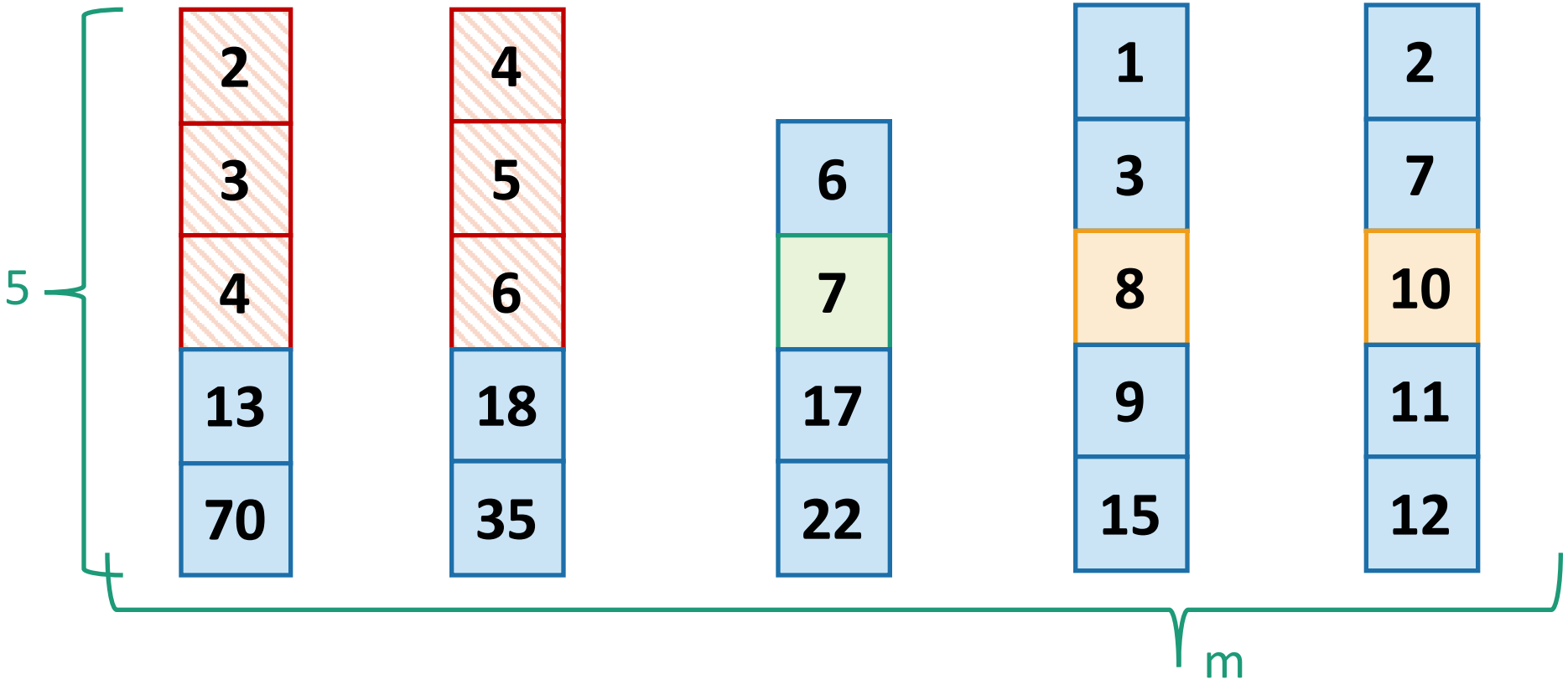
$3 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 1\right)$ of these, but then one of them could have been the "leftovers" group.



How many of those are there?

at least $3 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 2\right)$

Proof by picture



So how many are LARGER than the pivot? At most

(derivation on board)

$$n - 1 - 3\left(\left\lceil\frac{m}{2}\right\rceil - 2\right) \leq \frac{7n}{10} + 5$$

Remember $m = \left\lceil\frac{n}{5}\right\rceil$

That was one part of the lemma

- **Lemma:** If L and R are as in the algorithm SELECT given above, then

$$|L| \leq \frac{7n}{10} + 5$$

and

$$|R| \leq \frac{7n}{10} + 5$$

The other part is exactly the same.

*

Induction

$\forall_x P(x)$

(1) Prove $P(1)$

(2) Assume $P(n)$ $\forall$ $n < k$ (induction hypothesis)

(3) Prove that $P(k)$

$m = \frac{n}{5}$

$(n-1) - 3([\frac{m}{2}] - 2)$

$(n-1) - 3([\frac{n}{10}] - 2)$

$n - 1 - 3\frac{n}{10} + 6$

$\frac{10n}{10} - 3\frac{n}{10} + 5$

$\frac{7n}{10} + 5$

Substitution Method

- Recursion trees can get pretty messy here, since we have a recurrence relation that doesn't nicely break up our big problem into sub-problems of the same size.
- Instead, we will try to:
  - Make a guess
  - Check using an inductive argument
- This is called the substitution method.

Substitution method

work to call recursive sub-problems and merge them back

- Suppose that $T(n) \leq c \cdot f(n) + \sum_{i=1}^{r} T(n_i)$

Work in r different sub-problems, which might have different sizes.

- Let's guess the solution is

$$T(n) \leq \begin{cases} d \cdot g(n_0) & if \quad n \leq n_0 \\ d \cdot g(n) & if \quad n > n_0 \end{cases} \qquad (*)$$

- (aka, guessing T(n) = O(g(n)))
- We'll prove this by induction, with the inductive hypothesis (*) for all smaller n's.

In our case

The cn is the O(n) work done at each level for PARTITION

- $T(n) \leq c \cdot n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10} + 5\right)$

The T(7n/10 + 5) is for the recursive call to SELECT for either L or R.

The T(n/5) is for the recursive call to get the median in FINDPIVOT

- Inductive hypothesis:
- $T(n) \leq \begin{cases} d \cdot 50 & if \ n \leq 50 \\ d \cdot n & if \ n > 50 \end{cases}$

(aka, T(n) = O(n)).

for d = 10c.

(1) $T(1) = 1 \leq dn$ $\quad d=1$

(2) Assume that $n \leq k$ (hypothesis)

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10} + 5\right) \leq nd$$

(3) Prove that it is true for k

$$T(k) \leq ck + \frac{k}{5}d + \frac{7k}{10}d \overset{?}{\leq} kd$$

$$ck + \frac{9}{10}kd \leq kd$$

$$ck \leq \frac{kd}{10}$$

$$10c \leq d$$

$\therefore d = \max\{1, 10c\}$ then $T(n) < nd \ \forall n,$ ~~d = max{1, 10c}~~

$d = \max\{1, 10c\}$

$T(n) \leq dn \quad n \geq 50$

Summary

- We saw a (pretty clever) algorithm to do SELECT in time O(n).
- We proved that it worked using the Substitution Method.
- The Master Theorem wouldn't have worked for this.