Lecture 4

Median and Selection

Announcements!

- HW1 due tomorrow! 😃

Last Time: Solving Recurrence Relations

- A **recurrence relation** expresses $T(n)$ in terms of $T(\text{less than } n)$
- For example, $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$
- Two methods of solution:
    1. Master Theorem (aka, generalized "tree method")
    2. Substitution method (aka, guess and check)

The Master Theorem

- Suppose $a \geq 1, b > 1,$ and d are constants (that don't depend on n).
- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

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

Three parameters:
a : number of subproblems
b : factor by which input size shrinks
d : need to do n^d work to create all the subproblems and combine their solutions.

A powerful theorem it is…

Jedi master Yoda

The Substitution Method

- Step 1: Guess what the answer is.
- Step 2: Prove by induction that your guess is correct.
- Step 3: Profit.

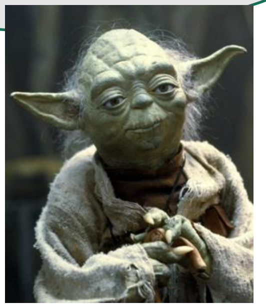
The plan for today

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
4. Return of the Substitution Method.

A fun recurrence relation

- $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + n$ for $n > 10.$
- Base case: $T(n) = 1 \text{ when } 1 \leq n \leq 10$

Apply here, the Master Theorem does NOT.



Jedi master Yoda

The Substitution Method

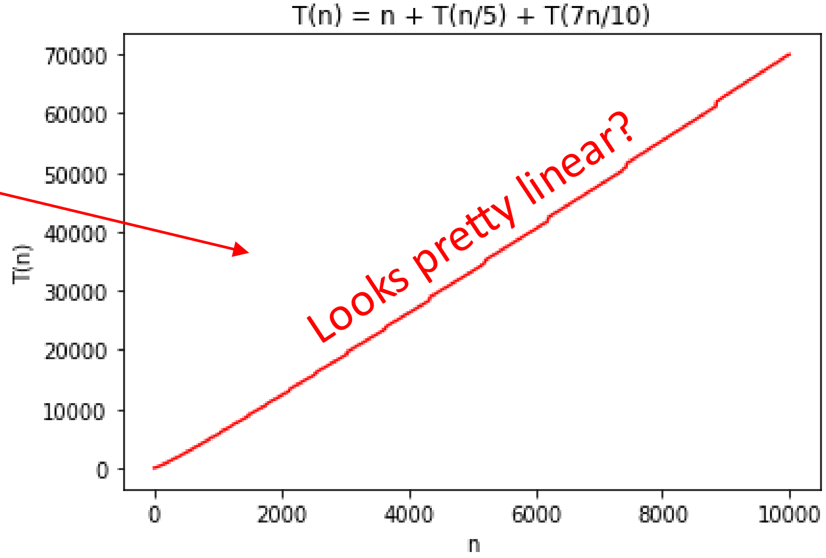
- Step 1: Guess what the answer is.
- Step 2: Prove by induction that your guess is correct.
- Step 3: Profit.

Step 1: guess the answer

$$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + n \text{ for } n > 10.$$

Base case: $T(n) = 1$ when $1 \leq n \leq 10$

- Trying to work backwards gets gross fast…
- We can also just try it out.
  - (see IPython Notebook)
- Let's guess O(n) and try to prove it.



Aside: Warning!

- It may be tempting to try to prove this with the inductive hypothesis "T(n) = O(n)"
- But that doesn't make sense!
- Formally, that's the same as saying:
  - Inductive Hypothesis for n:
  - There is some $n_0 > 0$ and some $C > 0$ so that, for all $n \geq n_0$, $T(n) \leq C \cdot n$.

The IH is supposed to hold for a *specific* n.

But now we are letting n be anything big enough!

- Instead, we should pick C, n_0 first…

Step 2: prove our guess is right

$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + n$ for $n > 10$.

Base case: $T(n) = 1$ when $1 \leq n \leq 10$

- Inductive Hypothesis: $T(n) \leq \boldsymbol{C}n$
- Base case: $1 = T(n) \leq \boldsymbol{C}n$ for all $1 \leq \mathrm{n} \leq 10$
- Inductive step:
  - Let k > 10. Assume that the IH holds for all n so that $1 \leq n < k$.
  - $T(k) \leq k + T\left(\frac{k}{5}\right) + T\left(\frac{7k}{10}\right)$

    $\leq k + \boldsymbol{C} \cdot \left(\frac{k}{5}\right) + \boldsymbol{C} \cdot \left(\frac{7k}{10}\right)$

    $= k + \frac{\boldsymbol{C}}{5} k + \frac{7\boldsymbol{C}}{10} k$

    $\leq \boldsymbol{C}k$??
  - (aka, want to show that IH holds for n=k).
- Conclusion:
  - There is some $\boldsymbol{C}$ (=10) so that for all $n \geq 1, T(n) \leq \boldsymbol{C}n$
  - By the definition of big-Oh, T(n) = O(n).

We don't know what C should be yet! Let's go through the proof leaving it as "C" and then figure out what works...

Whatever we choose C to be, it should have C≥1

Let's solve for C and make this true!
C = 10 works.
(on board)

$$T(n) \leq n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \text{ for } n > 10.$$

Base case: $T(n) = 1$ when $1 \leq n \leq 10$

Step 3: Profit

(Aka, pretend we knew this all along).

Theorem: $T(n) = O(n)$

Proof:

- Inductive Hypothesis: $T(n) \leq \mathbf{10}n$.
- Base case: $1 = T(n) \leq \mathbf{10}n$ for all $1 \leq \mathrm{n} \leq 10$
- Inductive step:
    - Let k > 10. Assume that the IH holds for all n so that $1 \leq n < k$.
    - $$\begin{aligned} T(k) &\leq k + T\left(\frac{k}{5}\right) + T\left(\frac{7k}{10}\right) \\ &\leq k + \mathbf{10} \cdot \left(\frac{k}{5}\right) + \mathbf{10} \cdot \left(\frac{7k}{10}\right) \\ &= k + 2k + 7k = \mathbf{10}k \end{aligned}$$
    - Thus IH holds for n=k.
- Conclusion:
    - For all $n \geq 1, T(n) \leq \mathbf{10}n$
    - Then, T(n) = O(n), using the definition of big-Oh with $n_0 = 1, c = 10$.

What have we learned?

- The substitution method can work when the master theorem doesn't.
  - For example with different-sized sub-problems.

- Step 1: generate a guess
  - Throw the kitchen sink at it.
- Step 2: try to prove that your guess is correct
  - You may have to leave some constants unspecified till the end – then see what they need to be for the proof to work!!
- Step 3: profit
  - Pretend you didn't do Steps 1 and 2 and write down a nice proof.

The Plan

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
4. Return of the Substitution Method.

The k-SELECT problem

from your pre-lecture exercise

For today, assume all arrays have distinct elements.

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

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

7	4	3	8	1	5	9	14

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

- SELECT(A, 1) = MIN(A)
- SELECT(A, n/2) = MEDIAN(A)
- SELECT(A, n) = MAX(A)

Being sloppy about floors and ceilings!

Note that the definition of Select is 1-indexed…

On your pre-lecture exercise…

An O(nlog(n))-time algorithm

- SELECT(A, k):
  - A = MergeSort(A)
  - **return** A[k-1]

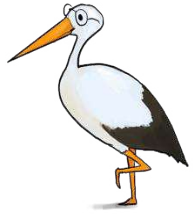
It's k-1 and not k since my pseudocode is 0-indexed and the problem is 1-indexed…

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

Can we do better?

We're hoping to get O(n)

Show that you can't do better than O(n).



Goal: An O(n)-time algorithm

- On your pre-lecture exercise: SELECT(A, 1).
  - (aka, MIN(A))
- MIN(A):
  - ret = ∞
  - **For** i=0, ..., n-1:
    - If A[i] < ret:
      - ret = A[i]
  - **Return** ret

This stuff is O(1)

This loop runs O(n) times

- Time O(n). Yay!

Also on your pre-lecture exercise

How about SELECT(A,2)?

- SELECT2(A):
  - ret = ∞
  - minSoFar = ∞
  - **For** i=0, .., n-1:
    - **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

(The actual algorithm here is not very important because this won't end up being a very good idea…)

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 has running time $\Theta(n^2)$

The Plan

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
4. Return of the Substitution Method.

Idea: divide and conquer!

Say we want to find SELECT(A, k)

9	8	3	6	1	4	2

How about this pivot?

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).

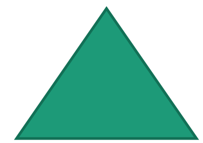
L = array with things smaller than A[pivot]

R = array with things larger than A[pivot]

Idea: divide and conquer!

Say we want to find SELECT(A, k)

6



How about this pivot?

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).

3	1	4	2

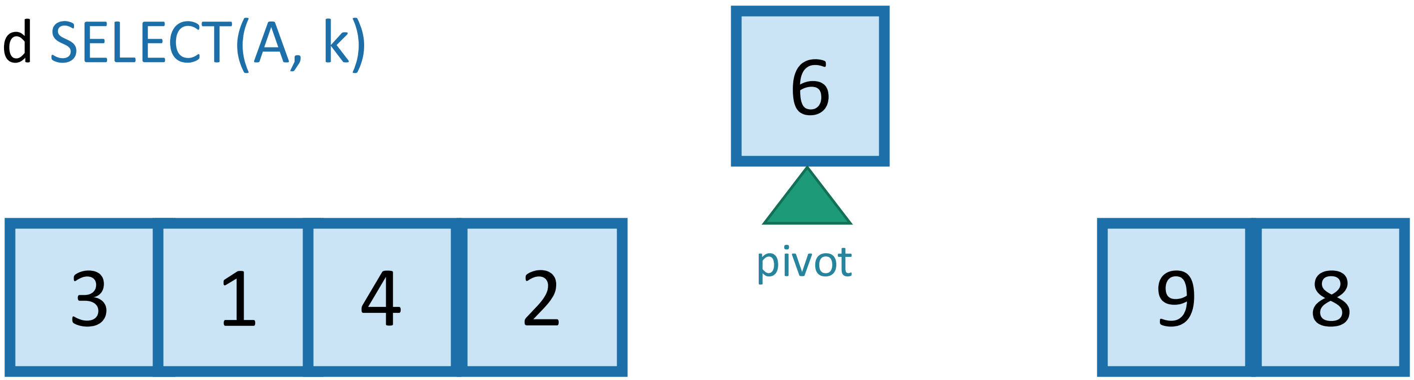
L = array with things smaller than A[pivot]

9	8

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…)

Pseudocode

- **getPivot**(A) returns some pivot for us.
  - How?? We'll see later…
- **Partition**(A,p) splits up A into L, A[p], R.
  - See Lecture 4 IPython notebook for code

- **Select**(A,k):
  - **If** len(A) <= 50:
    - A = **MergeSort**(A)
    - **Return** A[k-1]
  - p = **getPivot**(A)
  - L, pivotVal, R = **Partition**(A,p)
  - **if** len(L) == k-1:
    - return pivotVal
  - **Else if** len(L) > k-1:
    - return **Select**(L, k)
  - **Else if** len(L) < k-1:
    - return **Select**(R, k – len(L) – 1)

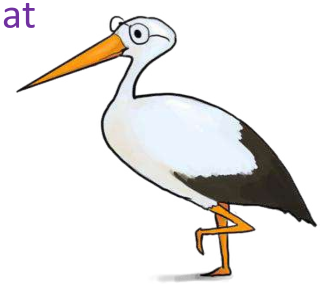
Base Case: If the len(A) = O(1), then any sorting algorithm runs in time O(1).

Case 1: We got lucky and found exactly the k'th smallest value!

Case 2: The k'th smallest value is in the first part of the list

Case 3: The k'th smallest value is in the second part of the list

Convince yourself that `Select` is correct!



Does it work?

- Check out the IPython notebook for Lecture 4, which implements this with a bunch of different pivot-selection methods.
  - Seems to work!
- Check out the handout posted on the website for a rigorous proof that this works, with any pivot-choosing mechanism.
  - It provably works!
  - Also, this is a good example of proving that a recursive algorithm is correct.

What is the running time?

Assuming we pick the pivot in time O(n)…

- (go to board and think about it…)

- **`Select`**(A,k):
  - **If** len(A) <= 50:
    - A = **`MergeSort`**(A)
    - **Return** A[k-1]
  - p = **`getPivot`**(A)
  - L, pivotVal, R = **`Partition`**(A,p)
  - **if** len(L) == k-1:
    - return pivotVal
  - **Else if** len(L) > k-1:
    - return **`Select`**(L, k)
  - **Else if** len(L) < k-1:
    - return **`Select`**(R, k – len(L) – 1)

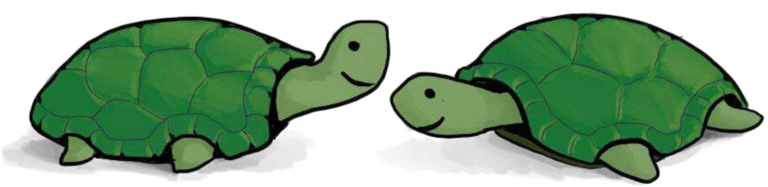
What is the running time?

Assuming we pick the pivot in time O(n)...

- $T(n) = \begin{cases} T(\mathbf{len(L)}) + O(n) & \mathbf{len(L)} > k - 1 \\ T(\mathbf{len(R)}) + O(n) & \mathbf{len(L)} < k - 1 \\ O(n) & \mathbf{len(L)} = k - 1 \end{cases}$

- What are **len(L)** and **len(R)**?
- That depends on how we pick the pivot...

What would be a "good" pivot?
What would be a "bad" pivot?



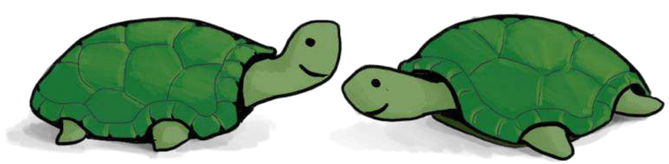
Think-Pair-Share Terrapins

The best way would be to pick the pivot so that len(L) = k-1. But say we want to pick a pivot in a way that's good no matter what k is.

The ideal pivot

- We split the input exactly in half:
  - len(L) = len(R) = (n-1)/2

What would be the running time in that case?
(If we could always choose that ideal pivot)



In case it's helpful…

- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

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

$$T(n) = \begin{cases} T\left(\mathbf{len}(\mathbf{L})\right) + O(n) & \mathbf{len}(\mathbf{L}) > k - 1 \\ T\left(\mathbf{len}(\mathbf{R})\right) + O(n) & \mathbf{len}(\mathbf{L}) < k - 1 \\ O(n) & \mathbf{len}(\mathbf{L}) = k - 1 \end{cases}$$

The ideal pivot

- We split the input exactly in half:
  - len(L) = len(R) = (n-1)/2

Apply here, the Master Theorem does NOT. Making unsubstantiated assumptions about problem sizes, we are.

Jedi master Yoda

- Let's pretend that's the case and use the **Master Theorem**!
  - $T(n) \leq T\left(\frac{n}{2}\right) + O(n)$
  - So a = 1, b = 2, d = 1
  - $T(n) \leq O\left(n^d\right) = O(n)$

- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

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

That would be great!

The worst pivot

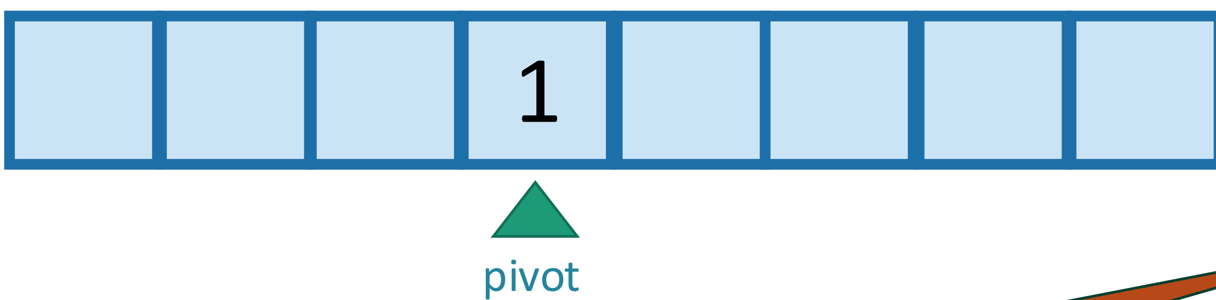
- The worst case is when we always recurse on almost the whole list…
- …how might that happen?

What is our recurrence relation in this worst case? What running time comes out of that?

Siggi the studious stork

In a worst-case analysis setting…

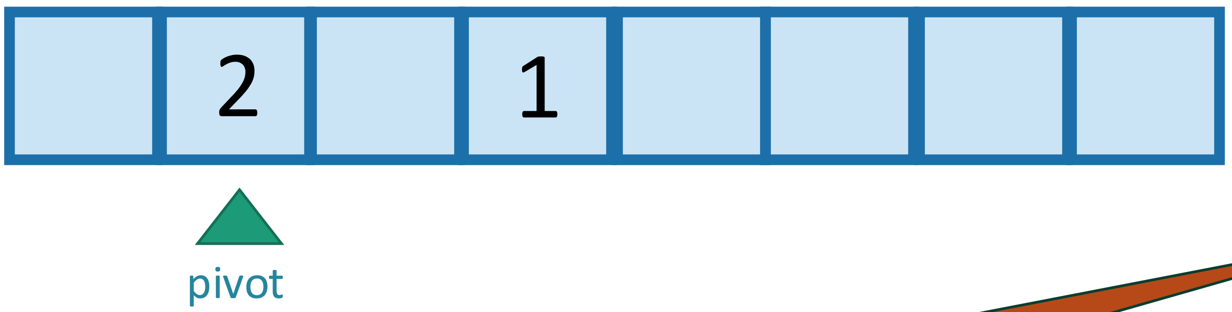
- **Any** version of **`getPivot`** that doesn't look at A is opening us up for a worst-case pivot.
- Suppose bad guy who knows what **`getPivot`** will do gets to come up with A.



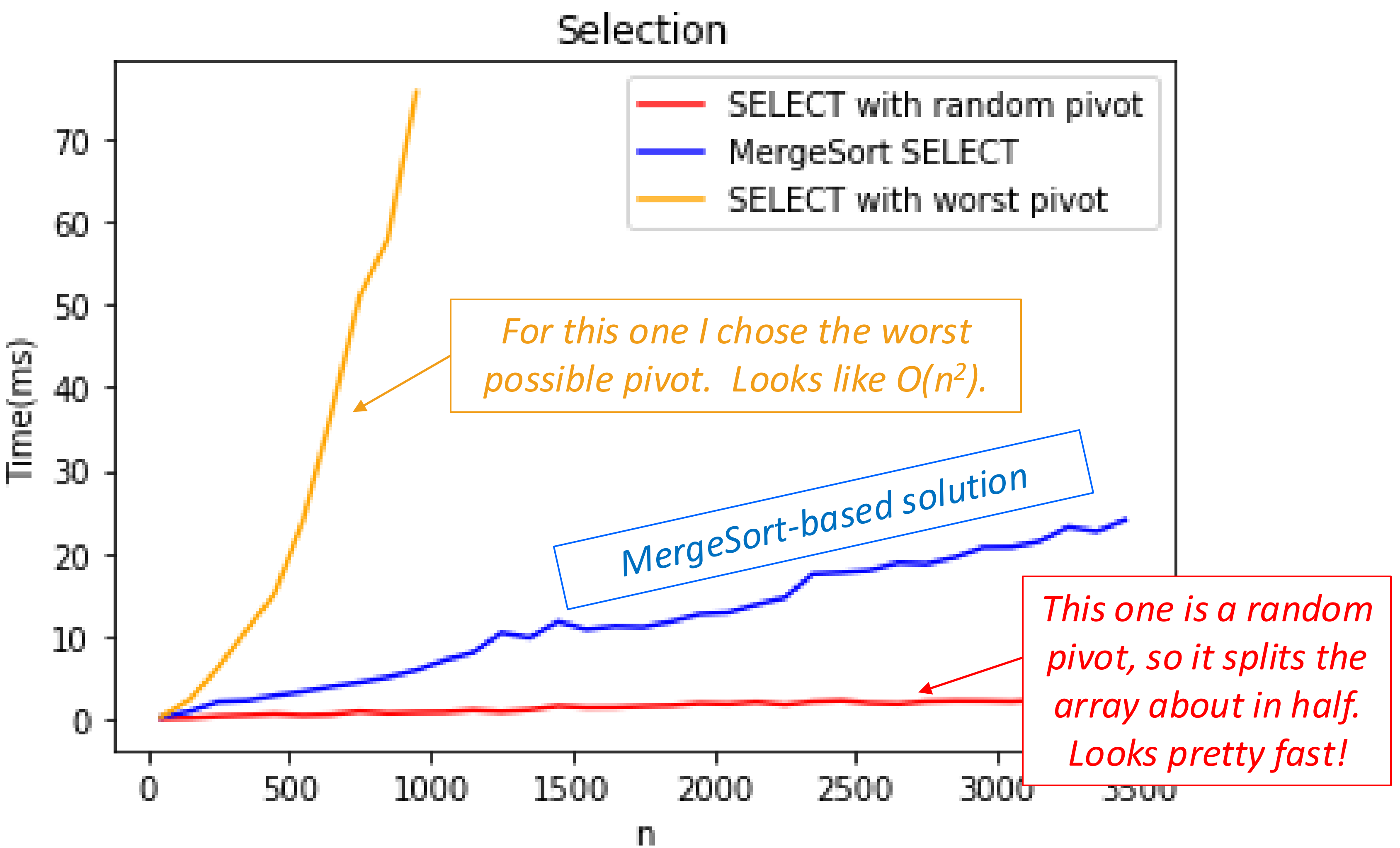
HaHA! I shall put the smallest element wherever you first put your pivot.

In a worst-case analysis setting…

- **Any** version of **`getPivot`** that doesn't look at A is opening us up for a worst-case pivot.
- Suppose bad guy who knows what **`getPivot`** will do gets to come up with A.



The distinction matters!



See Lecture 4 IPython notebook for code that generated this picture.

How do we pick a good pivot?

- Randomly?
  - That works well if there's no bad guy.
  - But if there is a bad guy who gets to see our pivot choices, a random pivot is just as bad as the worst-case pivot!
  - In this class, we're doing **worst-case analysis**, so we won't be happy with a random pivot.

Aside:

- In practice, there is often no bad guy. In that case, just pick a random pivot and it works really well!
- (More on this next week)

How do we pick a good pivot?

- For today, let’s assume there’s this bad guy.
- Reasons:
  - This gives us a very strong guarantee
  - We’ll get to see a really clever algorithm.
    - It will have to look at A in order to choose the pivot!
  - We’ll get to use the substitution method.

The Plan

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
   a) The outline of the algorithm.
   b) How to pick the pivot.
4. Return of the Substitution Method.

Approach

- First, we'll figure out what the ideal pivot would be.
  - But we won't be able to get it.
- Then, we'll figure out what a **pretty good** pivot would be.
  - But we still won't know how to get it.
- Finally, we will see how to get our pretty good pivot!
  - And then we will celebrate. 🎉

How do we pick our ideal pivot?

- We'd like to live in the ideal world.

- Pick the pivot to divide the input in half.
- Aka, pick the median!
- Aka, pick SELECT(A, n/2)!

How about a good enough pivot?

- We'd like to approximate the ideal world.

- Pick the pivot to divide the input about in half!
- Maybe this is easier!

A good enough pivot

We still don't know that we can get such a pivot, but at least it gives us a goal and a direction to pursue!

Lucky the lackadaisical lemur

- We split the input not quite in half:
  - 3n/10 < len(L) < 7n/10
  - 3n/10 < len(R) < 7n/10
- If we could do that (let's say, in time O(n)), the **Master Theorem** would say:

- $T(n) \leq T\left(\frac{7n}{10}\right) + O(n)$
- So a = 1, b = 10/7, d = 1
- $T(n) \leq O\left(n^d\right) = O(n)$

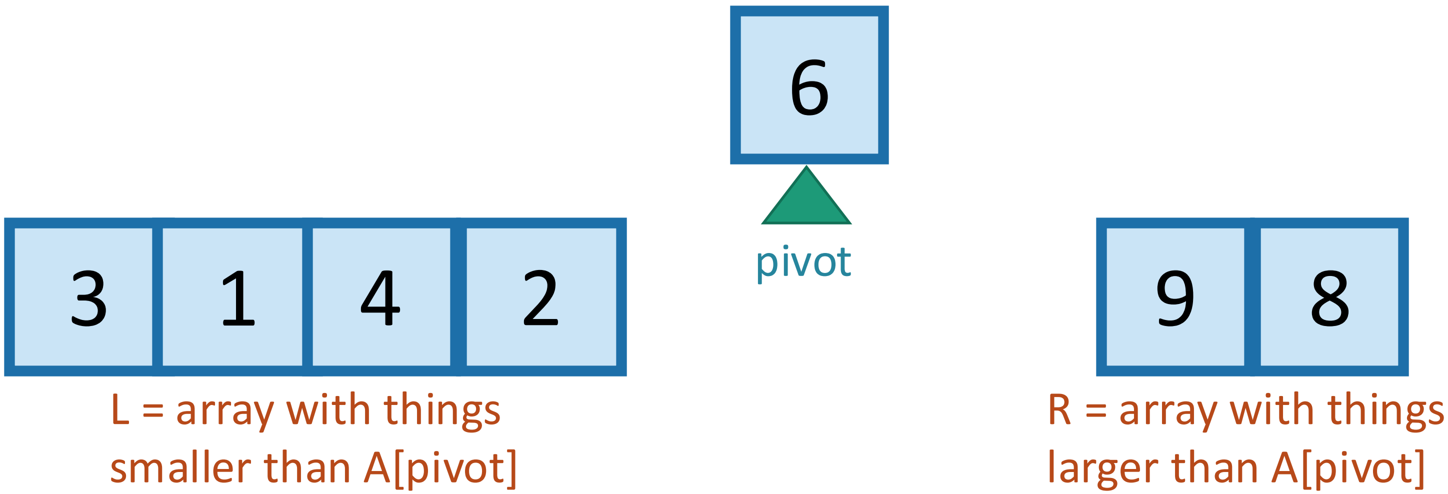
STILL GOOD!

- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

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

Goal

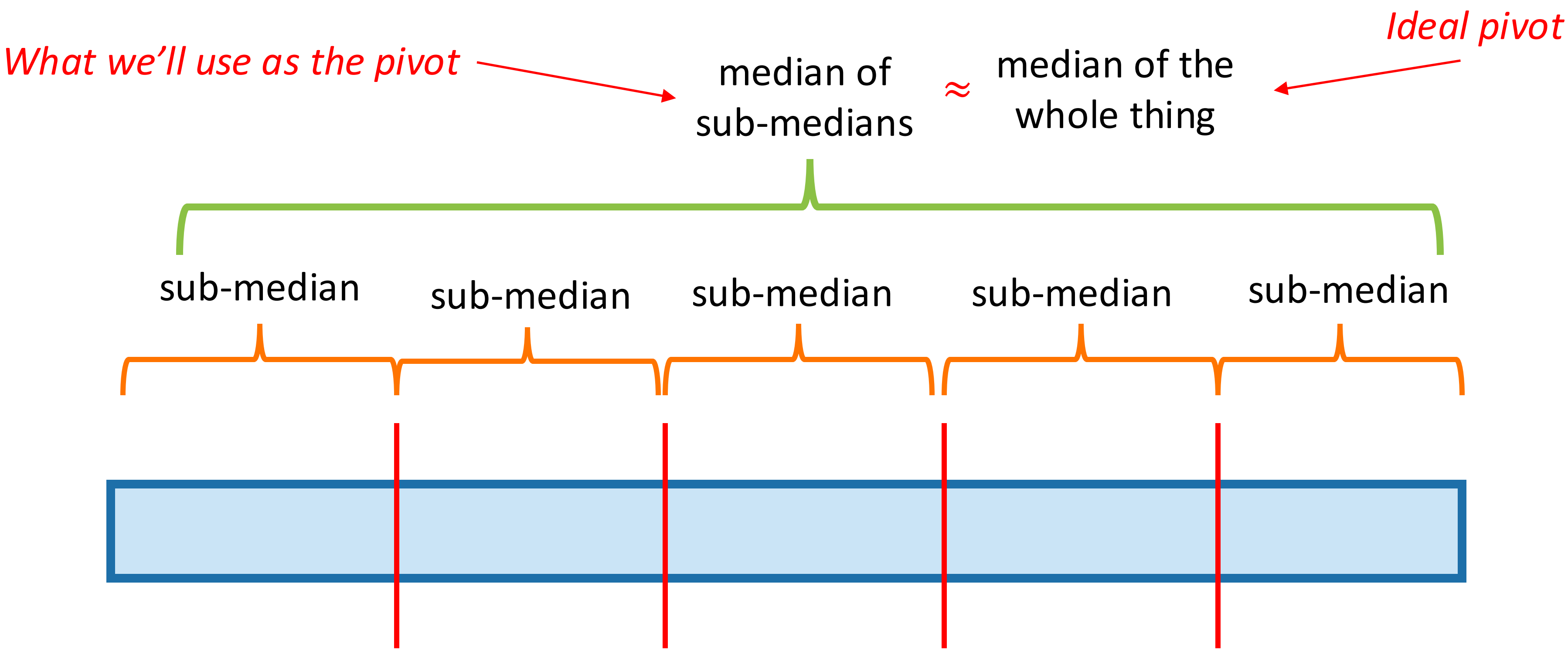
- Efficiently pick the pivot so that



$$\frac{\mathbf{3}\boldsymbol{n}}{\mathbf{10}} < \mathbf{len}(\boldsymbol{L}) < \frac{\mathbf{7}\boldsymbol{n}}{\mathbf{10}} \qquad \frac{\mathbf{3}\boldsymbol{n}}{\mathbf{10}} < \mathbf{len}(\boldsymbol{R}) < \frac{\mathbf{7}\boldsymbol{n}}{\mathbf{10}}$$

Another divide-and-conquer alg!

- We can't solve SELECT(A,n/2) (yet)
- But we can divide and conquer and solve SELECT(B,m/2) for smaller values of m (where len(B) = m).
- **Lemma*:** The median of sub-medians is close to the median.

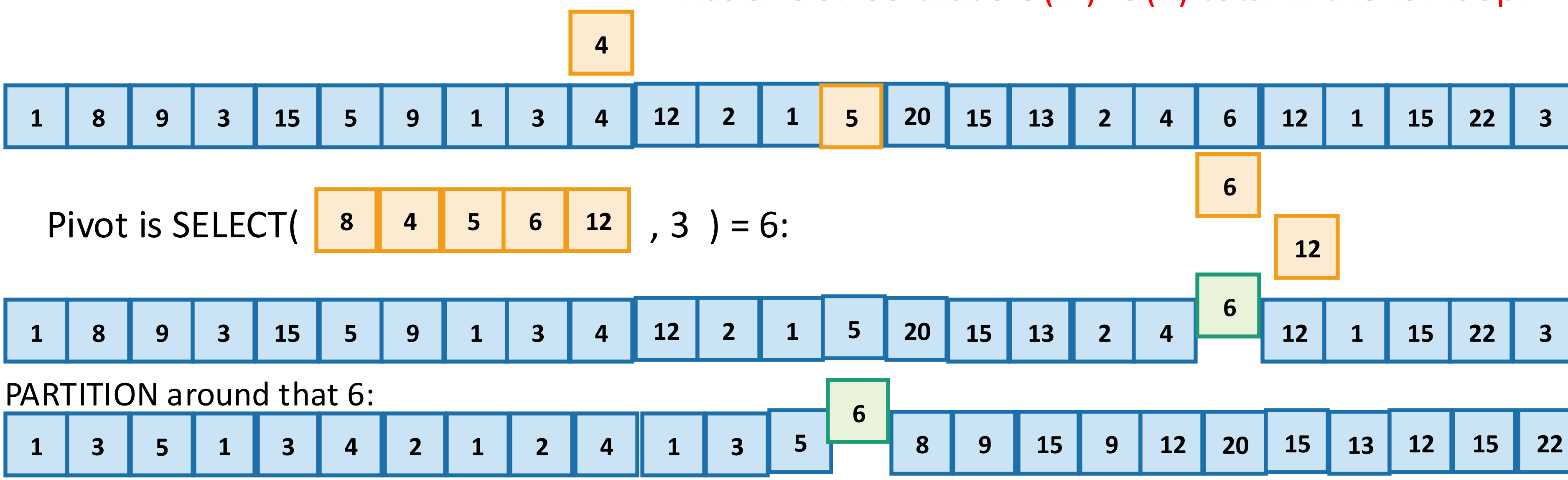


*we will make this a bit more precise.

How to pick the pivot

- 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** the index of p in A

This takes time O(1) for each group, since each group has size 5. So that’s O(m)=O(n) total in the for loop.



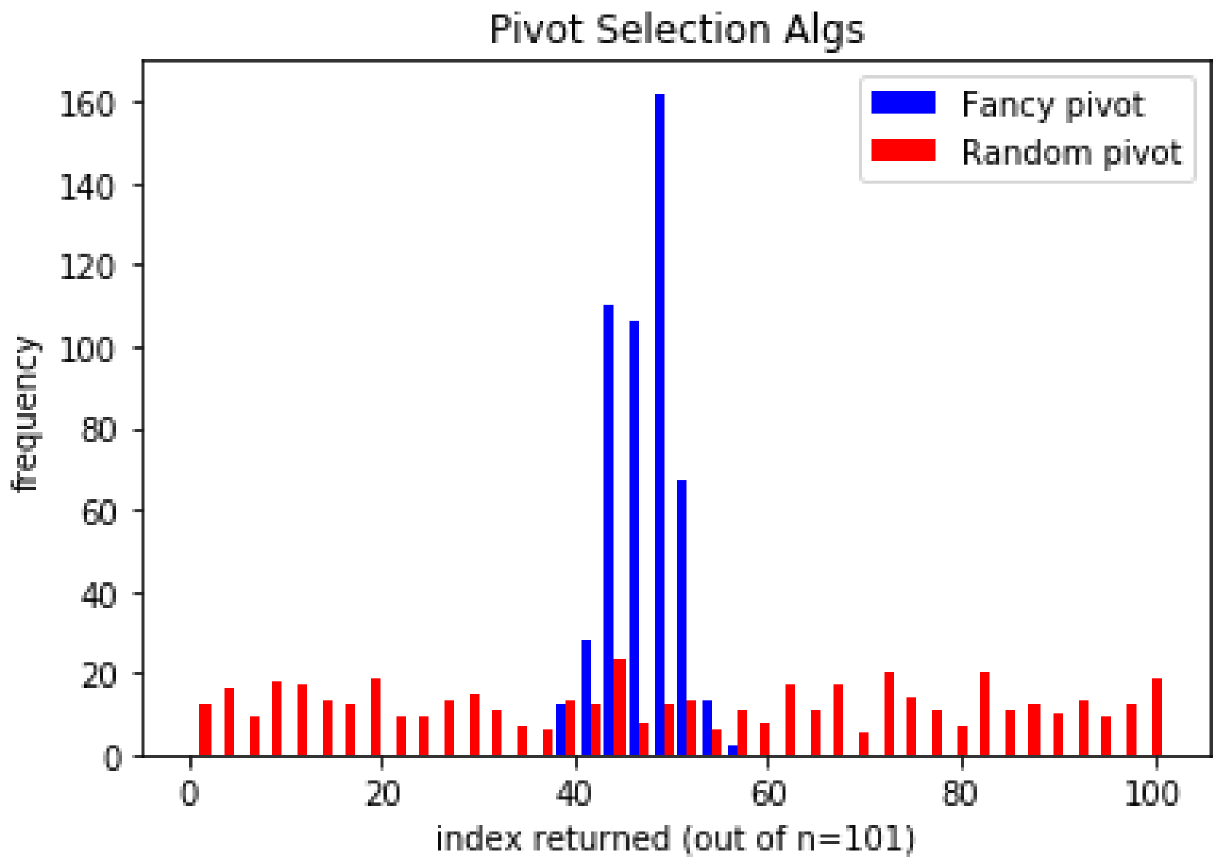
This part is L

This part is R: it’s almost the same size as L.

CLAIM: this works

divides the array *approximately* in half

- Empirically (see Lecture 4 IPython Notebook):



CLAIM: this works

divides the array *approximately* in half

- Formally, we will prove (later):

Lemma: If we choose the pivots like this, then

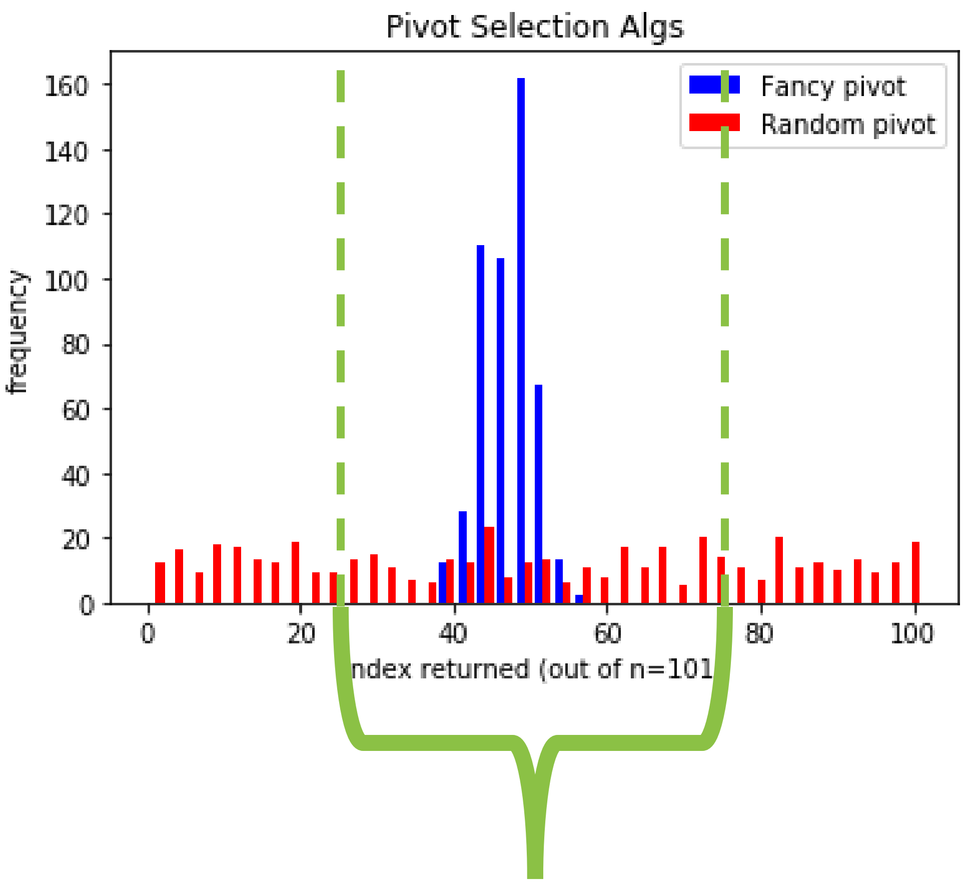
$$|L| \leq \frac{7n}{10} + 5$$

and

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

Sanity Check

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



That's this window

Actually in practice (on randomly chosen arrays) it looks **even better**!

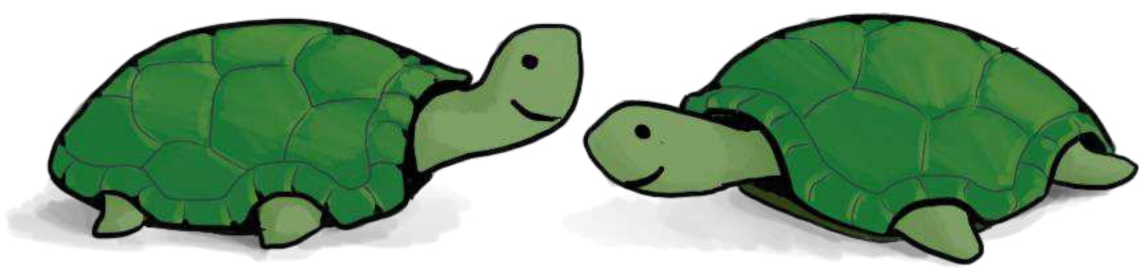
But this is a worst-case bound.

How about the running time?

- Suppose the Lemma is true. (It is).
  - $|L| \leq \frac{7n}{10} + 5$ and $|R| \leq \frac{7n}{10} + 5$

- Recurrence relation:

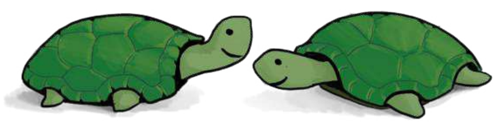
$$T(n) \leq ?$$



Pseudocode for **`choosePivot`**

while you think…

- Lemma: $|L| \leq \frac{7n}{10} + 5$ and $|R| \leq \frac{7n}{10} + 5$
- Suppose **`Partition`** runs in time O(n)
- Come up with a recurrence relation for T(n), the running time of **`Select`**, using the **`choosePivot`** algorithm we just described.



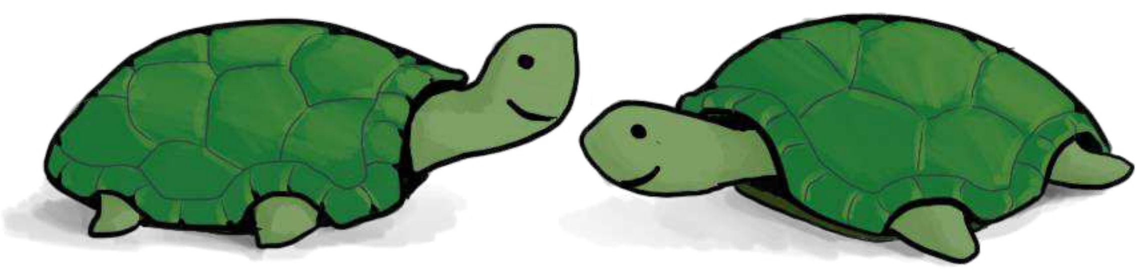
- **`choosePivot`**(A):
  - Split A into m = $\left\lceil \frac{n}{5} \right\rceil$ groups, of size <=5 each.
  - **For** i=1, .., m:
    - Find the median within the i'th group, call it p_i
  - p = SELECT([$p_1, p_2, p_3, \ldots, p_m$] , m/2)
  - **return** the index of p in A

How about the running time?

- Suppose the Lemma is true. (It is).
  - $|L| \leq \frac{7n}{10} + 5$ and $|R| \leq \frac{7n}{10} + 5$

- Recurrence relation:

$$T(n) \leq ?$$



How about the running time?

- Suppose the Lemma is true. (It is).
  - $|L| \leq \frac{7n}{10} + 5$ and $|R| \leq \frac{7n}{10} + 5$
- Recurrence relation:

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

The call to CHOOSEPIVOT makes one further recursive call to SELECT on an array of size n/5.

Outside of CHOOSEPIVOT, there's at most one recursive call to SELECT on array of size 7n/10 + 5.

We're going to drop the "+5" for convenience, but you can see CLRS for a more careful treatment if you're curious.

The Plan

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
   a) The outline of the algorithm.
   b) How to pick the pivot.
4. Return of the Substitution Method.

This sounds like a job for…

The Substitution Method!

Step 1: generate a guess
Step 2: try to prove that your guess is correct
Step 3: profit

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

That's convenient! We did this at the beginning of lecture!

Conclusion: $T(n) = O(n)$

Technically we only did it for $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + n$, not when the last term has a big-Oh…

Plucky the Pedantic Penguin

Recap of approach

- First, we figured out what the ideal pivot would be.
  - Find the median
- Then, we figured out what a **pretty good** pivot would be.
  - An approximate median
- Finally, we saw how to get our pretty good pivot!
  - Median of medians and divide and conquer!
  - Hooray!

In practice?

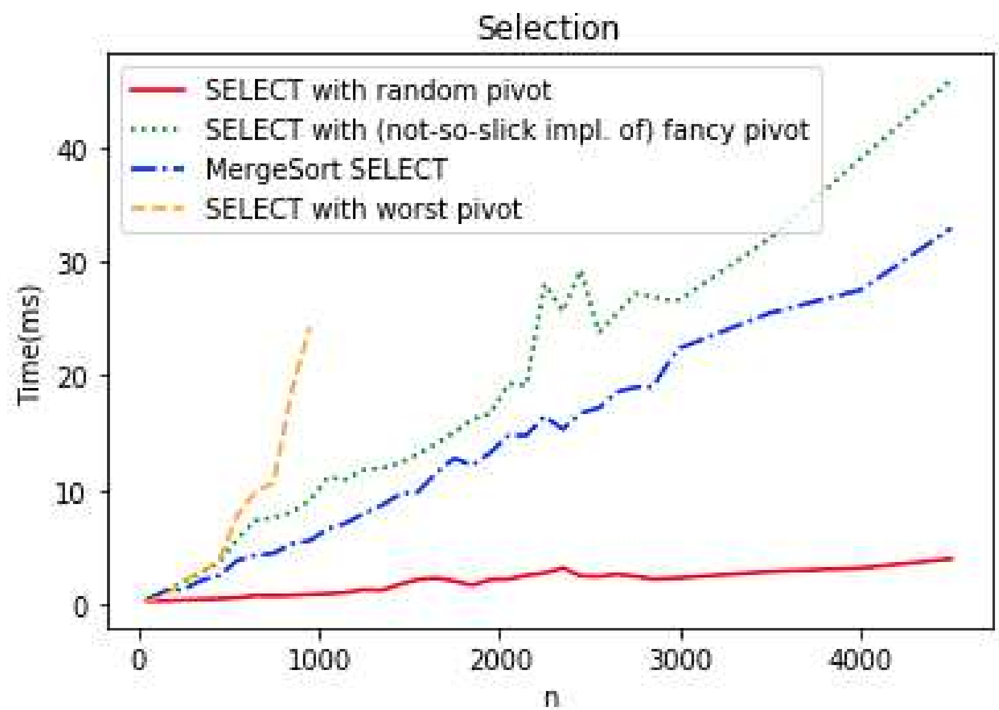
- With my not-very-slick implementation, our fancy version of SELECT is worse than the MergeSort-based SELECT ☹
  - But O(n) is better than O(nlog(n))! How can that be?
  - *What's the constant in front of the n in our proof? 20? 30?*

- On **non-adversarial** inputs, random pivot choice is much better.

Moral:
Just pick a random pivot if you don't expect nefarious arrays.

Optimize the implementation of SELECT (with the fancy pivot). Can you beat MergeSort for reasonable-sized inputs?

Siggi the Studious Stork



What have we learned?

Pending the Lemma

- It is possible to solve SELECT in time O(n).
  - Divide and conquer!
- If you want a deterministic algorithm expect that a bad guy will be picking the list, **choose a pivot cleverly.**
  - More divide and conquer!
- If you don't expect that a bad guy will be picking the list, in practice it's better just to **pick a random pivot.**

The Plan

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
   a) The outline of the algorithm.
   b) How to pick the pivot.
4. Return of the Substitution Method.
5. (If time) Proof of that Lemma.

If time, back to the Lemma

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

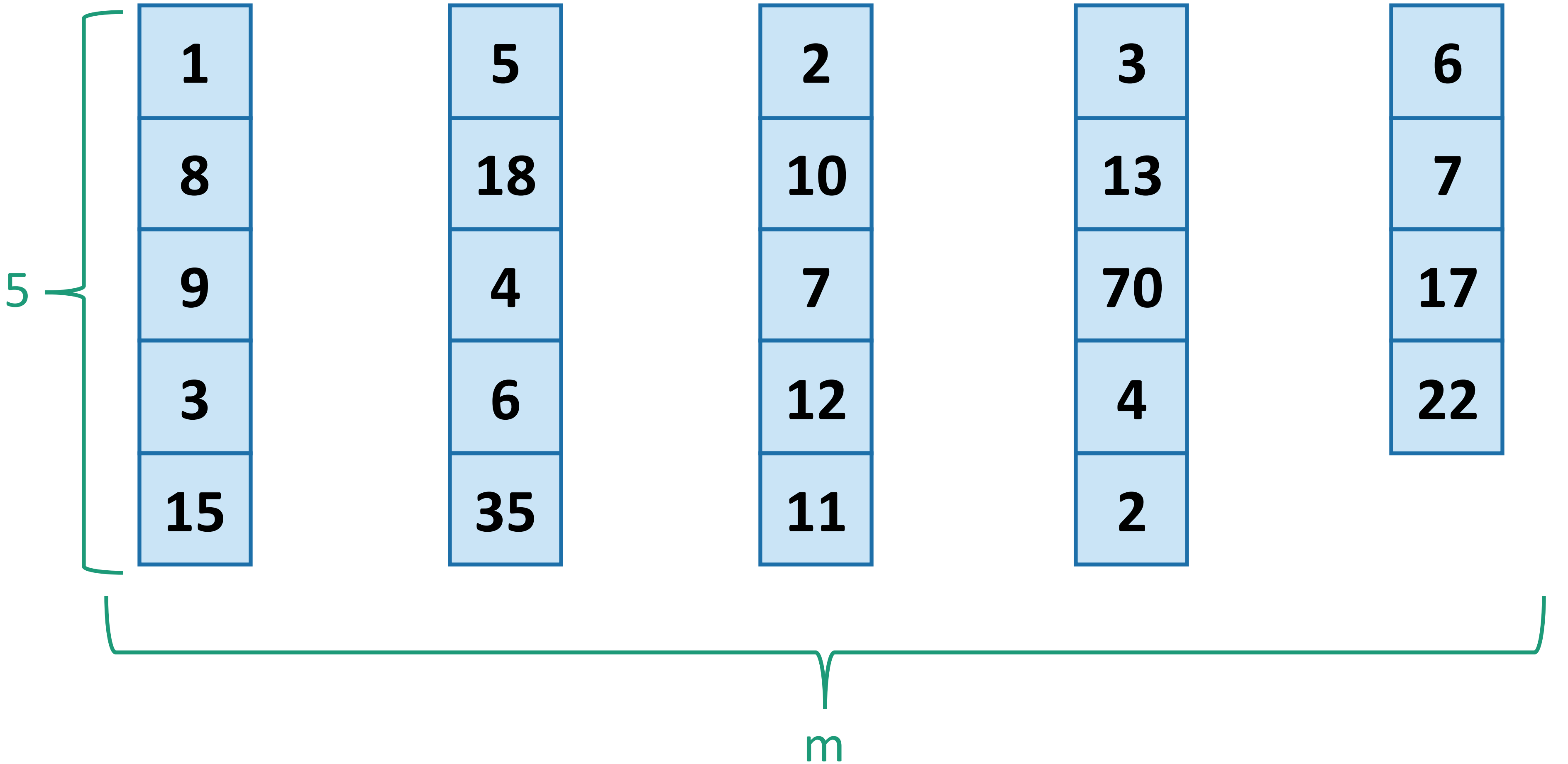
$$|L| \leq \frac{7n}{10} + 5$$

and

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

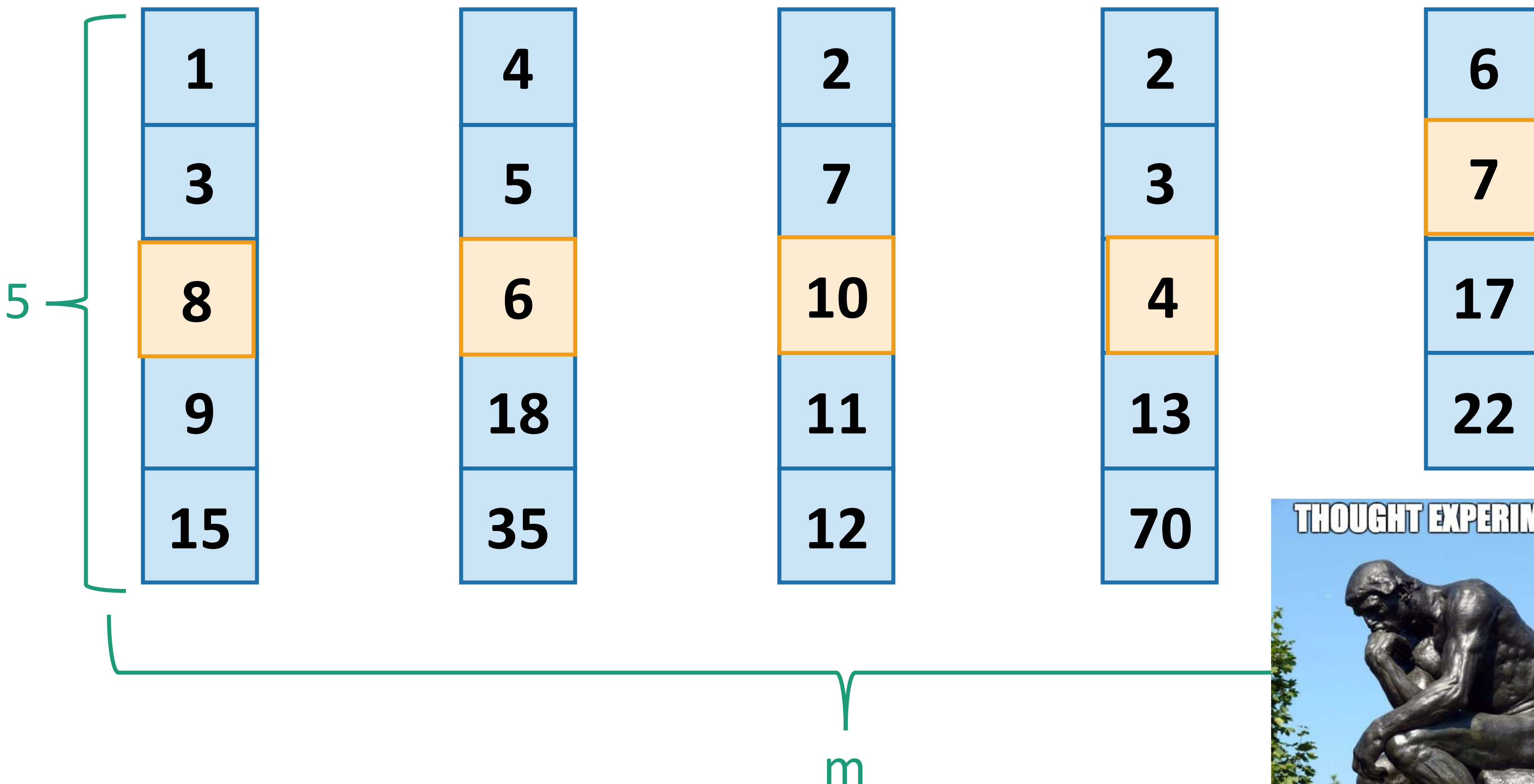
- We will see a proof by picture.
- See Algs Illuminated textbook (Lemma 6.7) for proof by proof.

Proof by picture



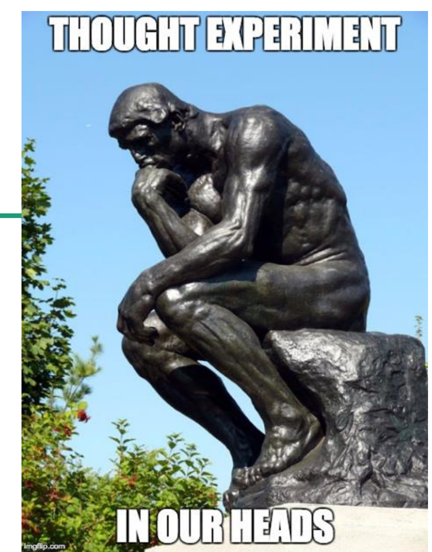
Say these are our m = [n/5] sub-arrays of size at most 5.

Proof by picture

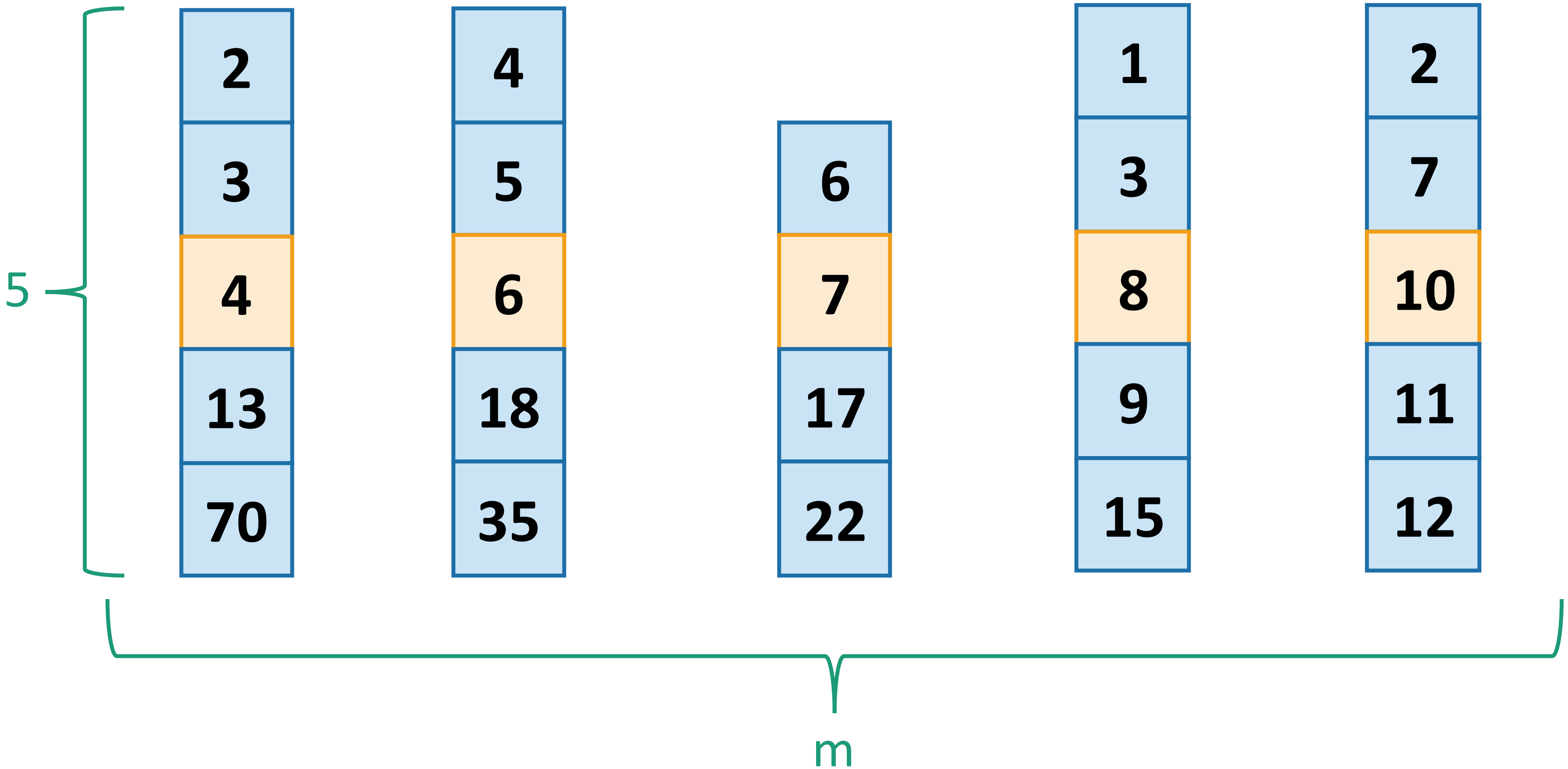


In our head, let's sort them.

Then find medians.

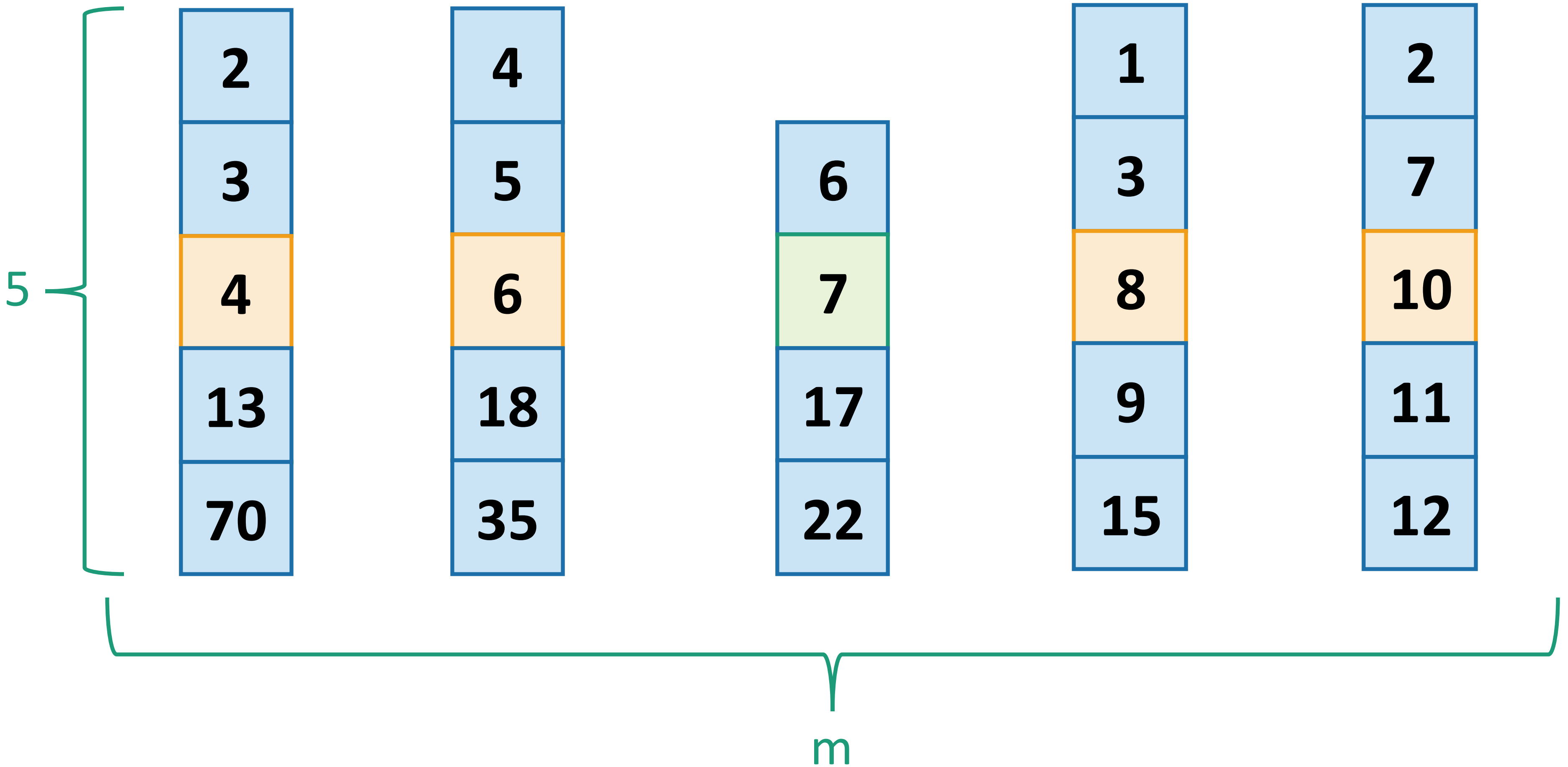


Proof by picture



Then let’s sort them by the median

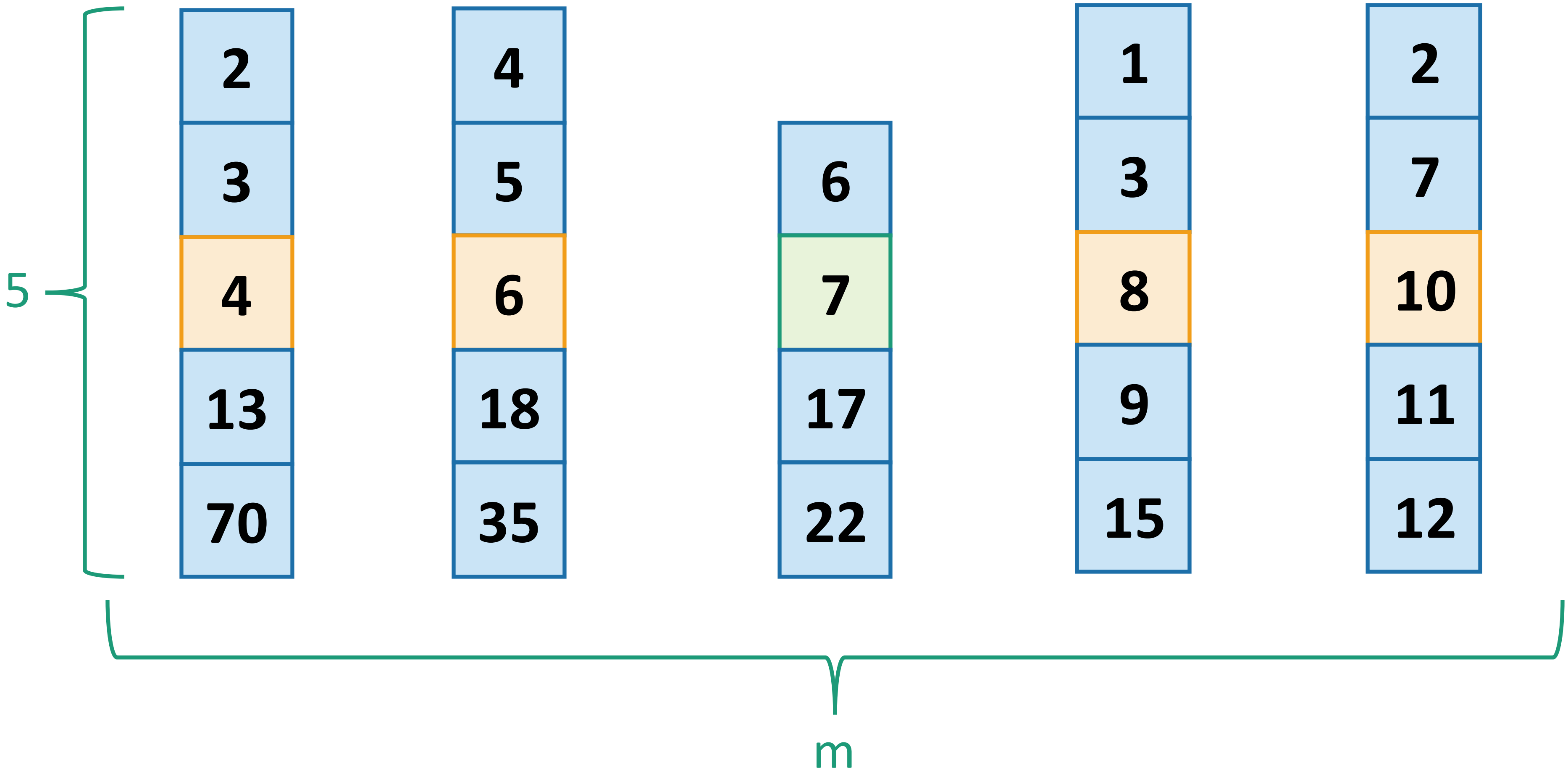
Proof by picture



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

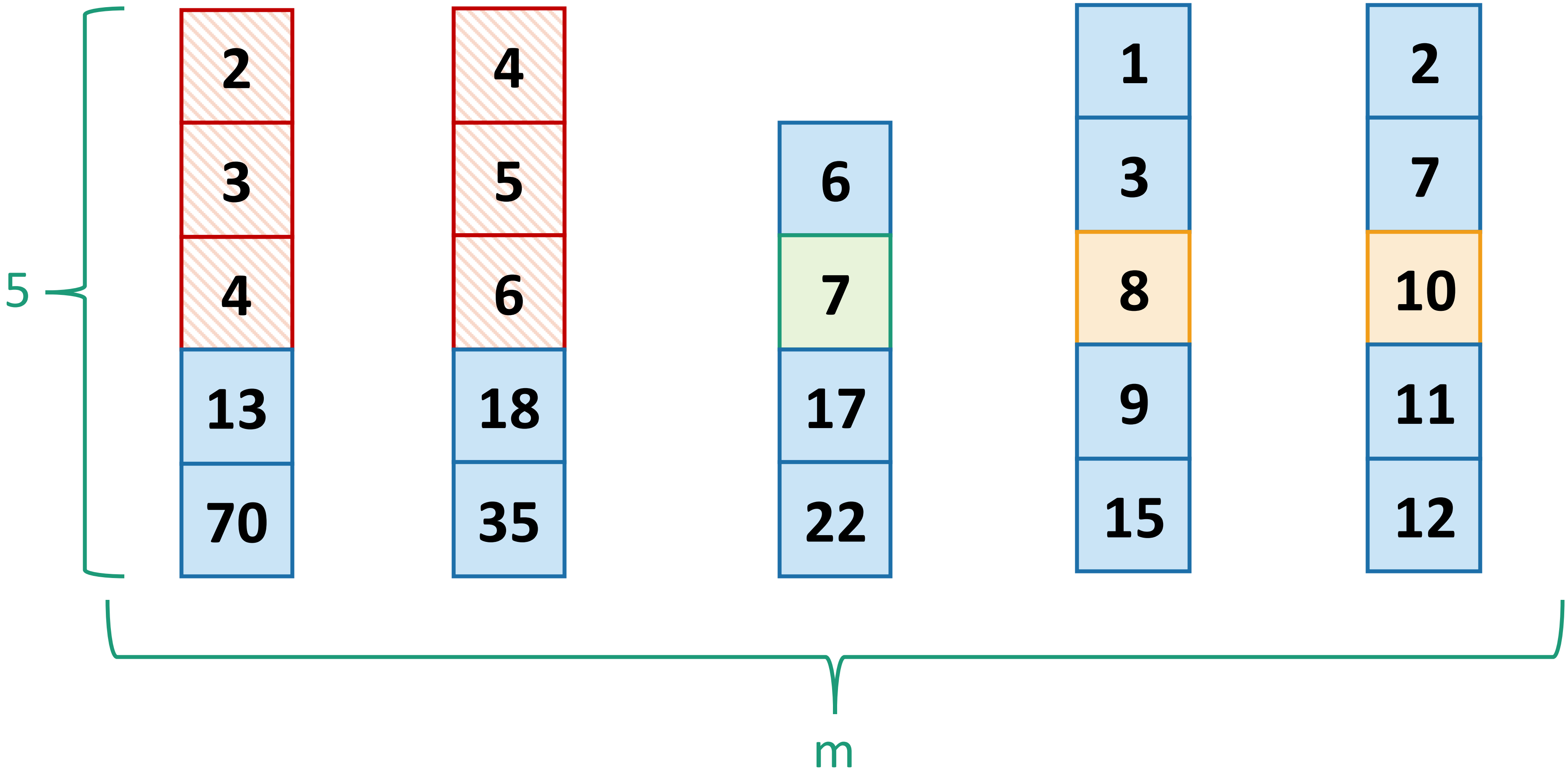
We will show that lots of elements are smaller than the pivot, hence not too many are larger than the pivot.

Proof by picture



How many elements are SMALLER than the pivot?

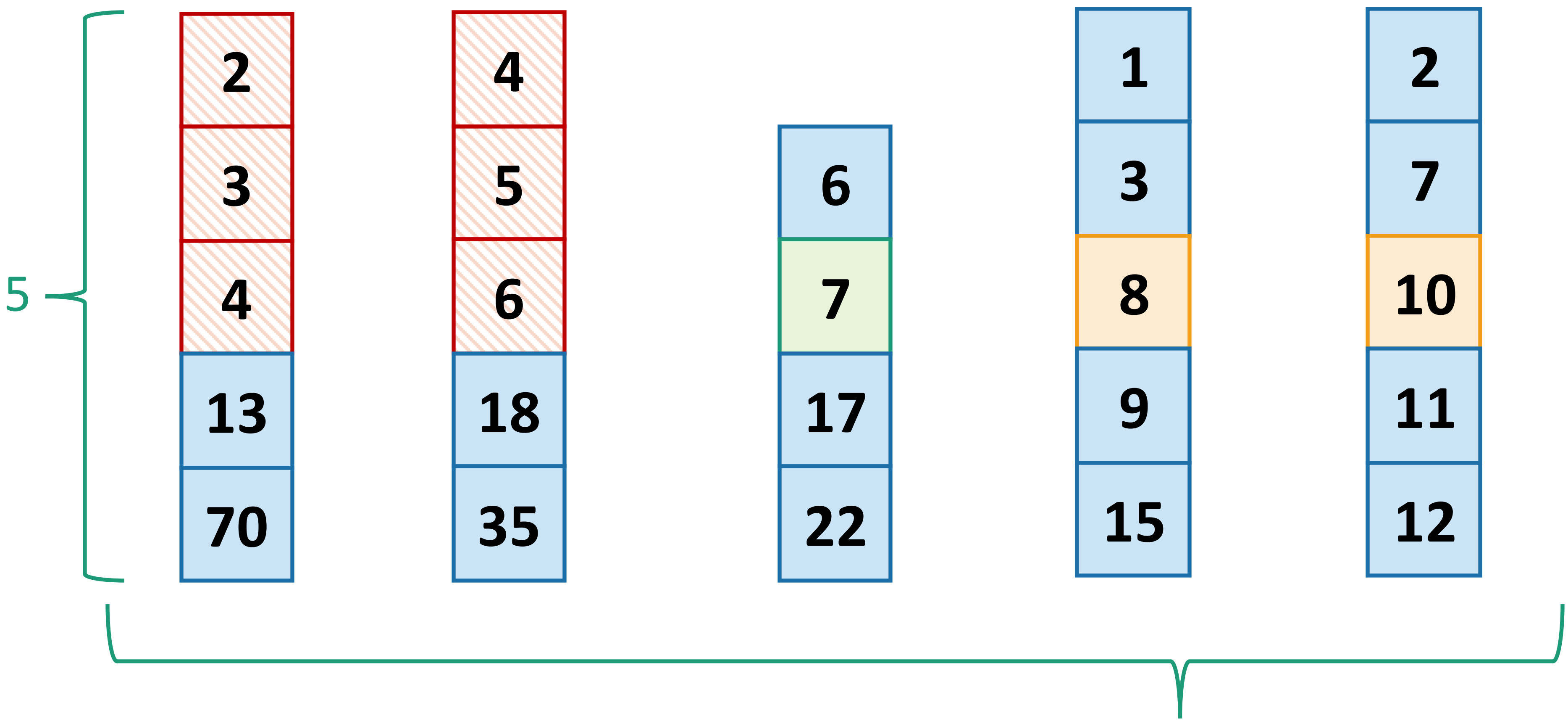
Proof by picture



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

Proof by picture

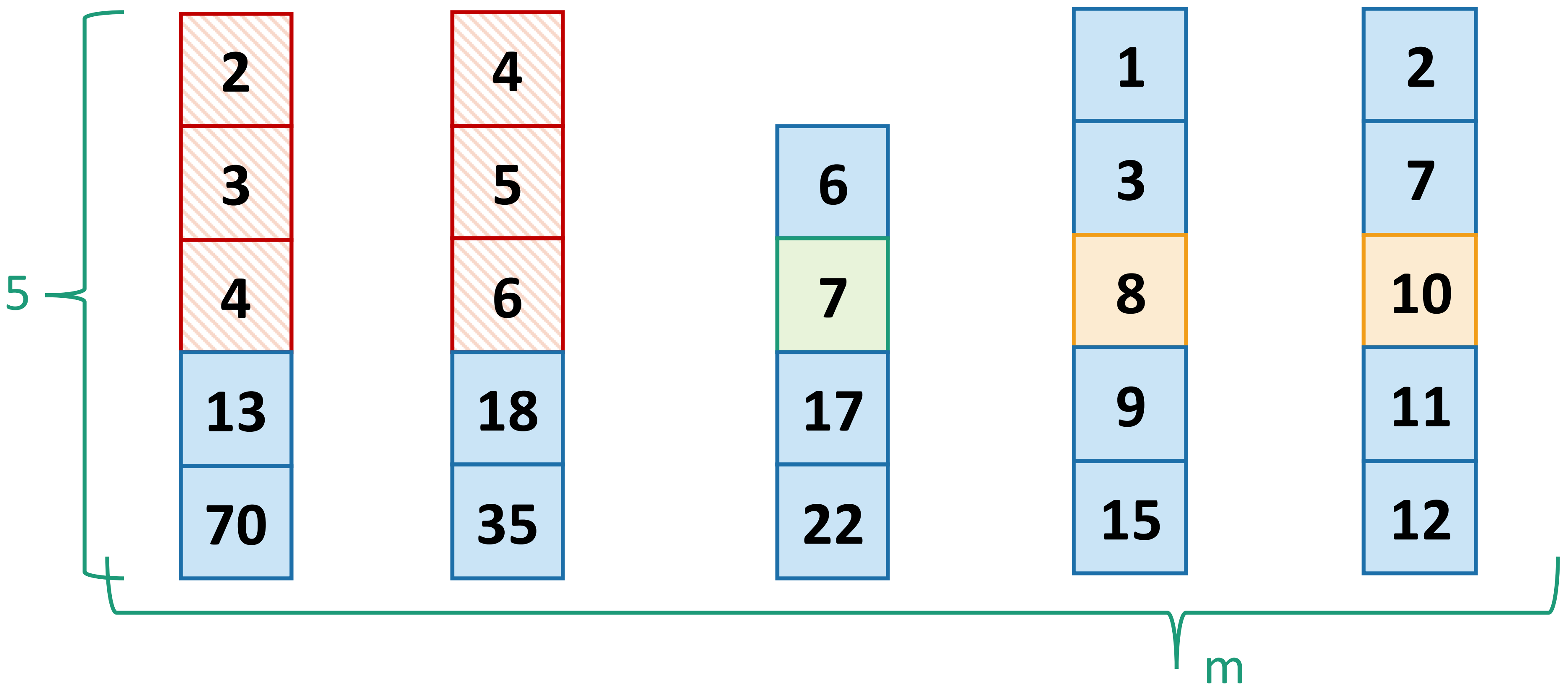
$3 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 1 \right)$ of these, but then one of them could have been the "leftovers" group.



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)

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

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.

The Plan

1. More practice with the Substitution Method.
2. k-SELECT problem
3. k-SELECT solution
   a) The outline of the algorithm.
   b) How to pick the pivot.
4. Return of the Substitution Method.
5. (If time) Proof of that Lemma.

Recap

Recap

- Substitution method can work when the master theorem doesn't.
- One place we needed it was for SELECT.
  - Which we can do in time O(n)!

Next time

- Randomized algorithms and QuickSort!

BEFORE next time

- Hand in HW1!
- Get started on HW2!
  - Homework party Monday!

- Pre-Lecture Exercise 5
  - Remember probability theory?
  - The pre-lecture exercise will jog your memory.