Lecture 4

The Substitution Method and Median and Selection

Announcements!

- HW1 due Friday.
  - (And HW2 also posted Friday).

Last Time: The Master Theorem

- 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.

Jedi master Yoda

Today

more recursion, beyond the 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.

I can handle all the recurrence relations that look like $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Before this theorem I was but the learner. Now I am the master.

Only a master of evil*, Darth.

*More precisely, only a master of same-size sub-problems…still pretty handy, actually.

The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
4. Return of the **Substitution Method.**

A non-tree method

For most of this lecture, division is integer division: $\frac{n}{2}$ means $\left\lfloor \frac{n}{2} \right\rfloor$.
As we noted last time we'll be pretty sloppy about the difference.

- Here's another way to solve:
  - $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$
  - $T(0) = 0, \; T(1) = 1$

1. Guess what the answer is.
2. Formally prove that that's what the answer is.

You did this for your pre-lecture exercise! Let's go through it now quickly to make sure we are all on the same page.

1. Guess what the answer is.

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$

- $T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$
- $T(n) = 4 \cdot T\left(\frac{n}{4}\right) + 2 \cdot n$

- $T(n) = 4 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2 \cdot n$
- $T(n) = 8 \cdot T\left(\frac{n}{8}\right) + 3 \cdot n$

- Following the pattern…
- $\boldsymbol{T(n) = n \cdot T(1) + log(n) \cdot n = n(log(n) + 1\ )}$

So that is our guess!

We'll go fast through these computations because you all did it on your pre-lecture exercise!

- Inductive hypothesis:
  - $\boldsymbol{T(k) \leq k(log(k) + 1)}$ for all $1 \leq \mathrm{k} \leq \mathrm{n}$
- Base case:
  - $\boldsymbol{T(1) = 1 = 1(log(1) + 1)}$
- Inductive step:
  - $\boldsymbol{T(n)} = 2 \cdot T\left(\frac{n}{2}\right) + n$

$$\leq 2\left(\frac{n}{2}\left(\log\left(\frac{n}{2}\right) + 1\right)\right) + n$$

$$= 2\left(\frac{n}{2}(\log(n) - 1 + 1)\right) + n$$

$$= 2\left(\frac{n}{2}\log(n)\right) + n$$

$$= \boldsymbol{n(\log(n) + 1)}$$

What happened between these two lines?

- Conclusion:
  - By induction, $\boldsymbol{T(n) = n(log(n) + 1)}$ for all n > 0.

That’s called the substitution method

- So far, just seems like a different way of doing the same thing.

- But consider this!

$$T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$$

$$T(n) = 10n \text{ when } 1 \leq n \leq 10$$

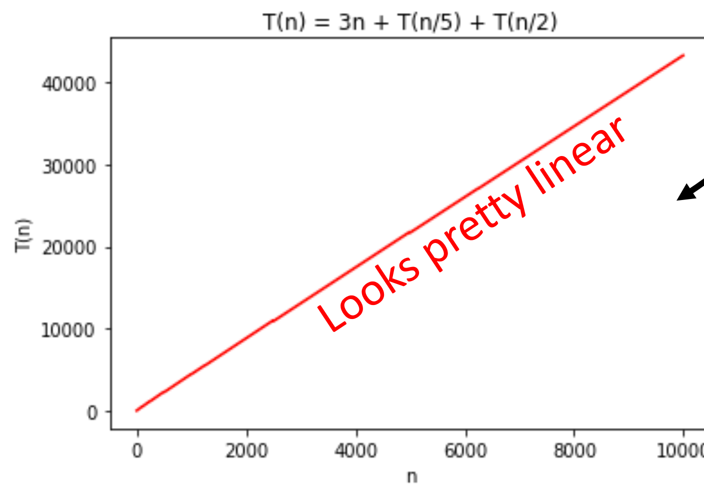
Gross!

Step 1: guess what the answer is

$$T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$$

$$T(n) = 10n \text{ when } 1 \le n \le 10$$

- Let's try the same unwinding thing to get a feel for it.
  - *[On board]*
- Okay, that gets gross fast. We can also just try it out.
  - *[IPython Notebook]*



- What else do we know?:
  - $T(n) \le 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$

    $\le 3n + 2 \cdot T\left(\frac{n}{2}\right)$

    $= O(n\log(n))$
  - $T(n) \ge 3n$
  - So the right answer is somewhere between O(n) and O(nlog(n))…

$$T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$$

$$T(n) = 10n \text{ when } 1 \leq n \leq 10$$

Let's guess O(n)

Step 2: prove our guess is right

- Inductive Hypothesis: $\boldsymbol{T(k) \leq Ck}$ for all $1 \leq k < n$.
- Base case: $\boldsymbol{T(k) \leq Ck}$ for all $\mathrm{k} \leq 10$
- Inductive step:
  - $T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$
    $\leq 3n + \boldsymbol{C}\left(\frac{n}{5}\right) + \boldsymbol{C}\left(\frac{n}{2}\right)$
    $= 3n + \frac{\boldsymbol{C}}{5}n + \frac{\boldsymbol{C}}{2}n$
    $\leq \boldsymbol{C}n$??
- Conclusion:
  - There is some $\boldsymbol{C}$ so that for all $n \geq 1$, $\boldsymbol{T(n) \leq Cn}$
  - Aka, T(n) = O(n).

C is some constant we'll have to fill in later!

Whatever we choose C to be, it should have C≥10

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

Now pretend like we knew it all along.

$$T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$$

$$T(n) = 10n \text{ when } 1 \leq n \leq 10$$

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

Proof:

- Inductive Hypothesis: $\boldsymbol{T(k) \leq 10k}$ for all k < n.
- Base case: $\boldsymbol{T(k) \leq 10k}$ for all k $\leq$ 10
- Inductive step:
  - $T(n) = 3n + T\left(\frac{n}{5}\right) + T\left(\frac{n}{2}\right)$
  - $T(n) \leq 3n + \mathbf{10}\left(\frac{n}{5}\right) + \mathbf{10}\left(\frac{n}{2}\right)$
  - $T(n) \leq 3n + 2n + 5n = 10n.$
- Conclusion:
  - For all $n \geq 1$, $\boldsymbol{T(n) \leq 10n}$, aka T(n) = O(n).

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. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
4. Return of the **Substitution Method.**

The problem we will solve

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

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

For today, assume all arrays have distinct elements.

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…

We're gonna do it in time O(n)

- Let's start with MIN(A) aka SELECT(A, 1).
- 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!

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

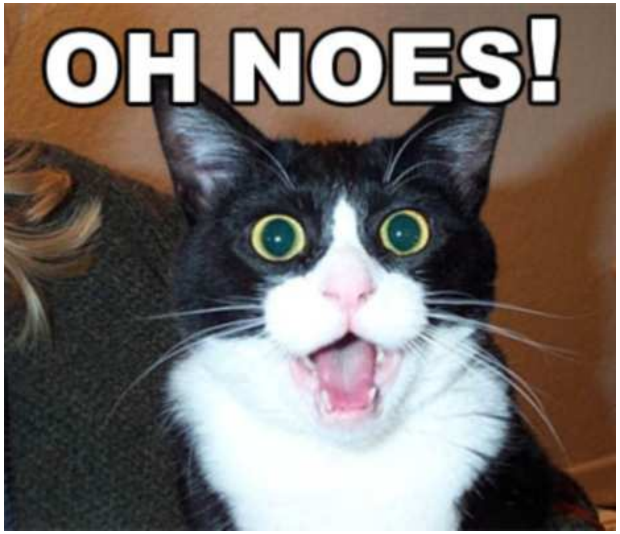
(The actual algorithm here is not very important because this won't end up being a very good idea…)

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 was $O(n^2)$.

A much better idea for large k

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

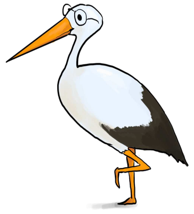
It's k-1 and not k since my pseudocode is 0-indexed and the problem is 1-indexed…

- 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)! (Or see lecture notes).



The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **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.

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

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

3	1	4	2

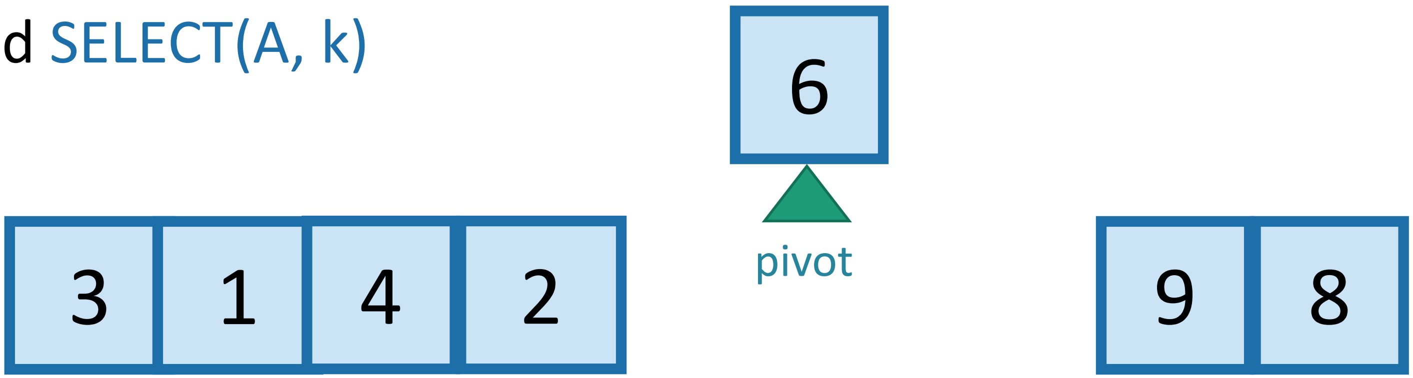
L = array with things smaller than A[pivot]

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 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

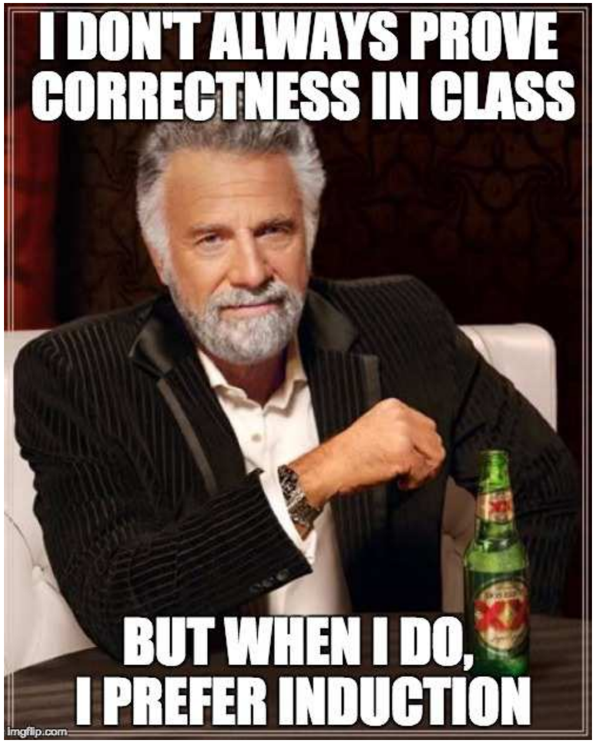
Case 3: The k'th smallest value is in the second part of the list

Let's make sure it works

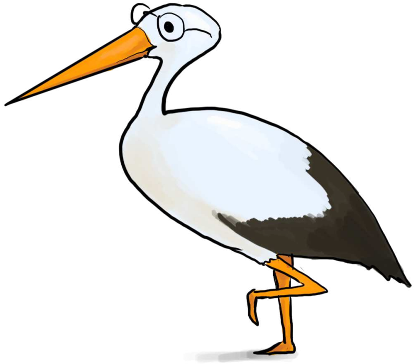
- [IPython Notebook for Lecture 4]

Now we should be convinced

- No matter what procedure we use for **`getPivot`**(A), **`Select`**(A,k) returns a correct answer.



Formally prove the correctness of **`Select`**!



Siggi the Studious Stork

What is the running time?

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

- What are **len(L)** and **len(R)**?
  - That depends on how we pick the pivot…
  - What do we hope happens?
  - What do we hope doesn't happen?

In an ideal world*…

- We split the input in half:
  - **len(L) = len(R) = (n-1)/2**
- Let's use the **Master Theorem**!
  - $T(n) \leq T\left(\frac{n}{2}\right) + O(n)$
  - So a = 1, b = 2, d = 1
  - $T(n) \leq O\left(n^d\right) = O(n)$

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

Jedi master Yoda

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

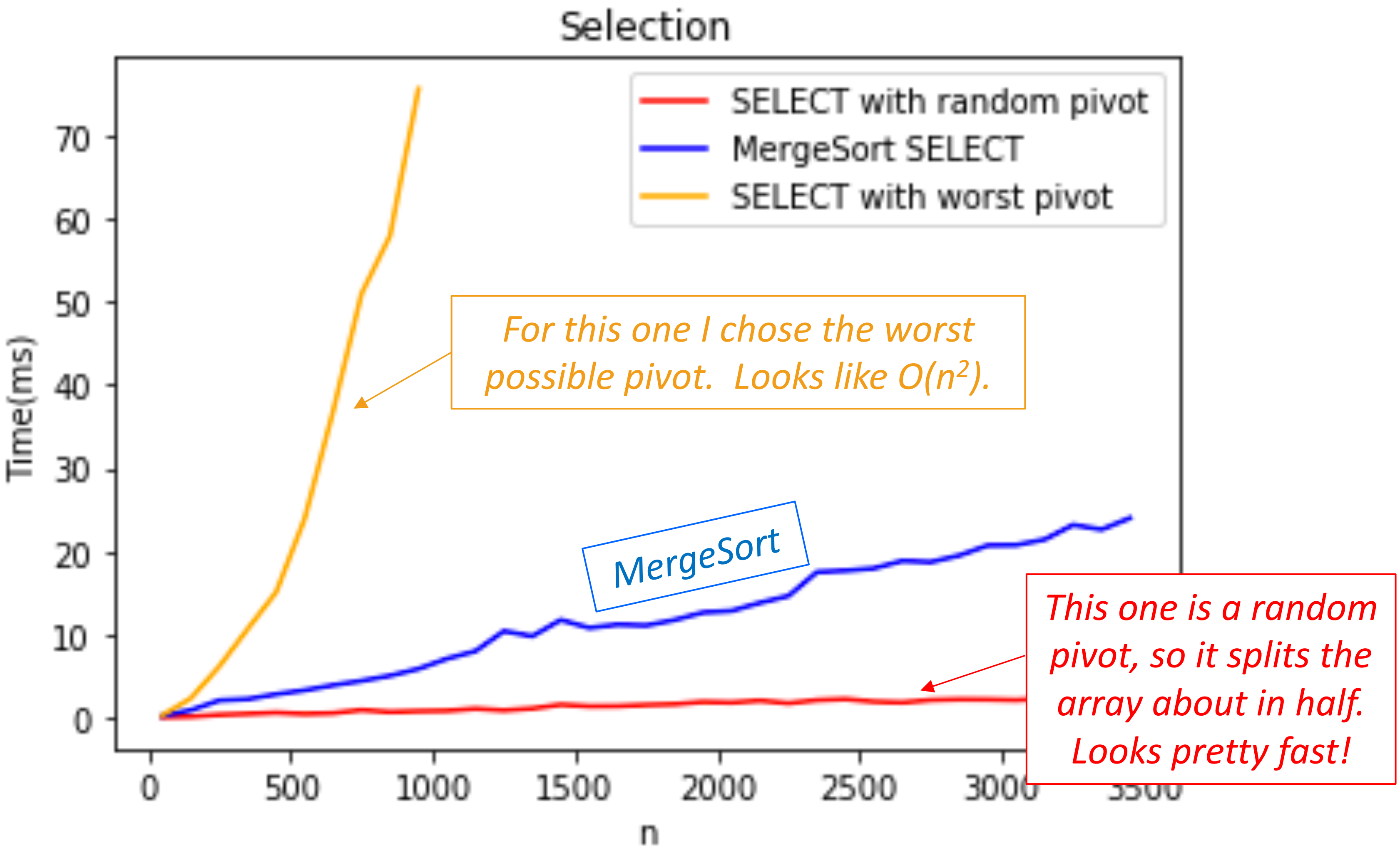
$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

*Okay, really ideal would be that we always pick the pivot so that len(L) = k-1. But say we don't have control over k, just over how we pick the pivot.

But the world is not ideal.

- Suppose we choose a pivot first, but then a bad guy who **knows what pivots we will choose** gets to come up with A.
- *[Discussion on board]*

The distinction matters!



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

Question

- ***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, that's just as bad as the worst-case 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)

But for today

- Let's assume there's this bad guy.
- We'll get a **stronger guarantee**
- We'll get to see a **really clever algorithm**
- And we'll get more practice with the **substitution method.**

The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
   a) The outline of the algorithm.
   b) How to pick the pivot.
4. Return of the **Substitution Method.**

How should we pick the 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 should we pick the pivot?

- We'd like to **approximate** the ideal world.

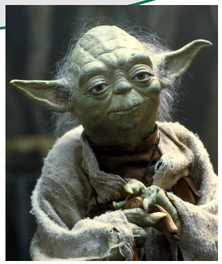
- Pick the pivot to divide the input **about** in half!
- Maybe this is easier!

In an ~~ideal~~ okay world…

Apply here, the Master Theorem STILL does NOT. (Since we don't know that we can do this – and if we could how long would it take?).

But at least it gives us a goal!

Lucky the lackadaisical lemur



Jedi master Yoda

- 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, the **Master Theorem** would say:
  - $T(n) \leq T\left(\frac{7n}{10}\right) + O(n)$
  - So a = 1, b = 10/7, d = 1
  - $T(n) \leq O\left(n^d\right) = O(n)$

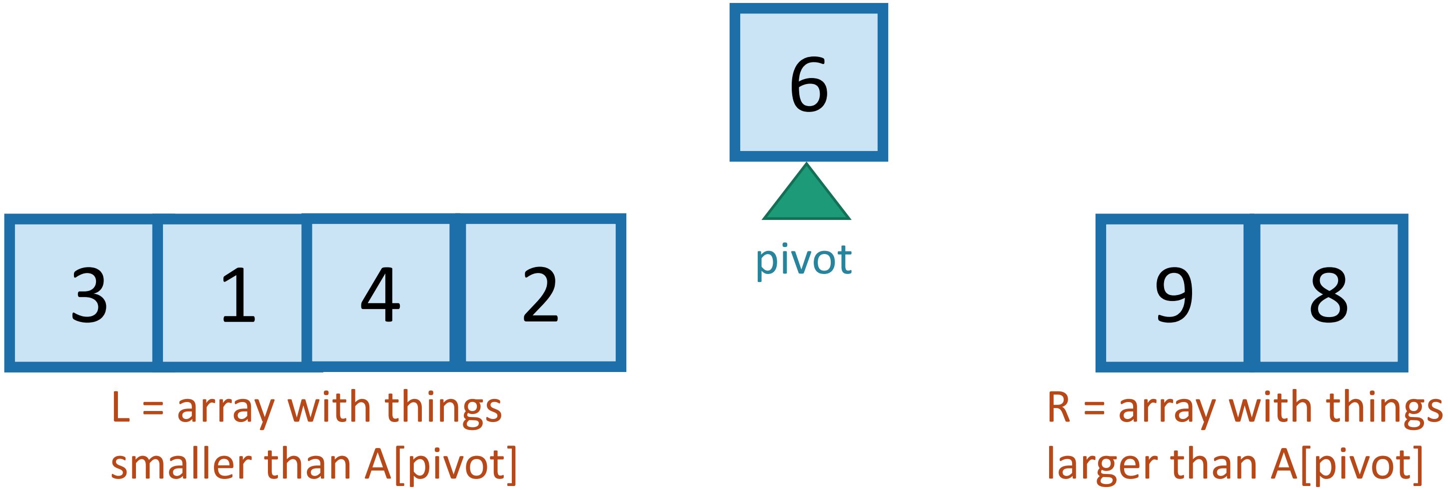
STILL GOOD!

- 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

- Pick the pivot so that

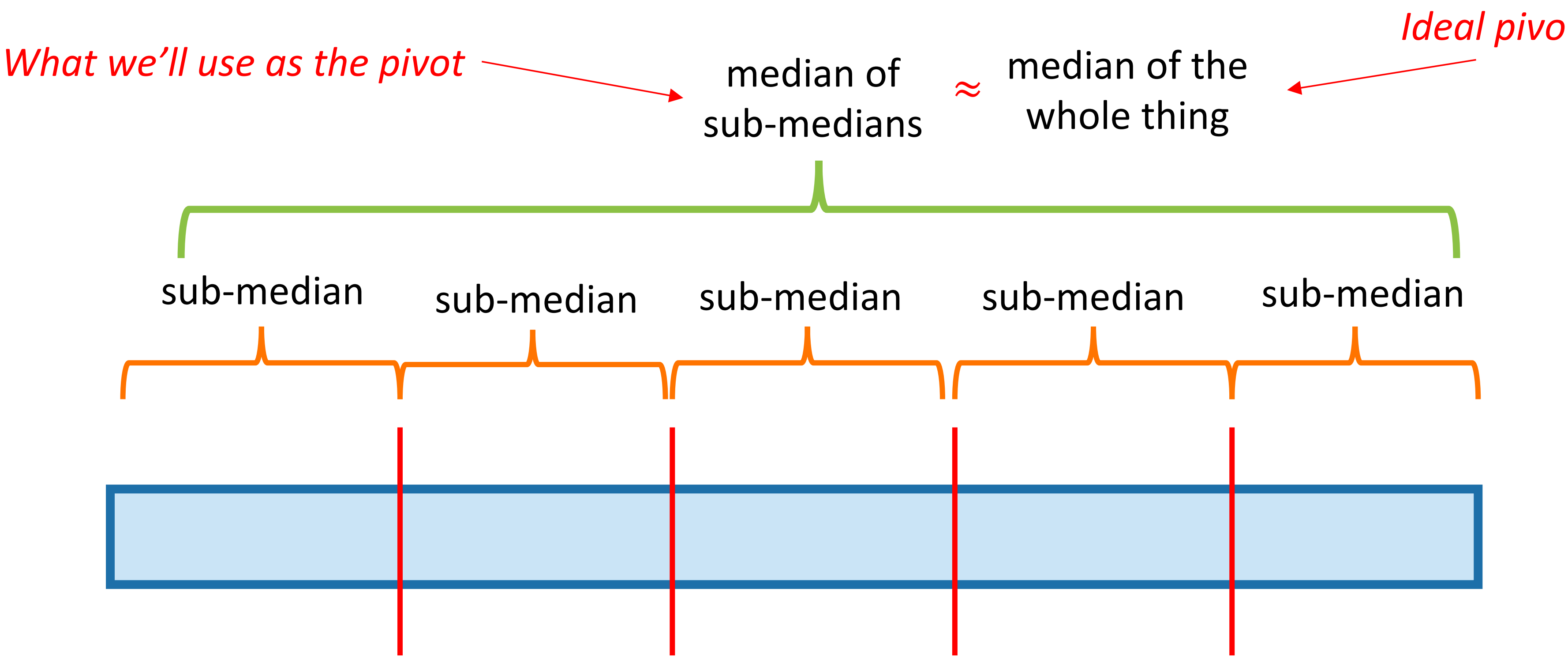


$$\frac{\mathbf{3}n}{\mathbf{10}} < \mathbf{len}(L) < \frac{\mathbf{7}n}{\mathbf{10}}$$

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

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.

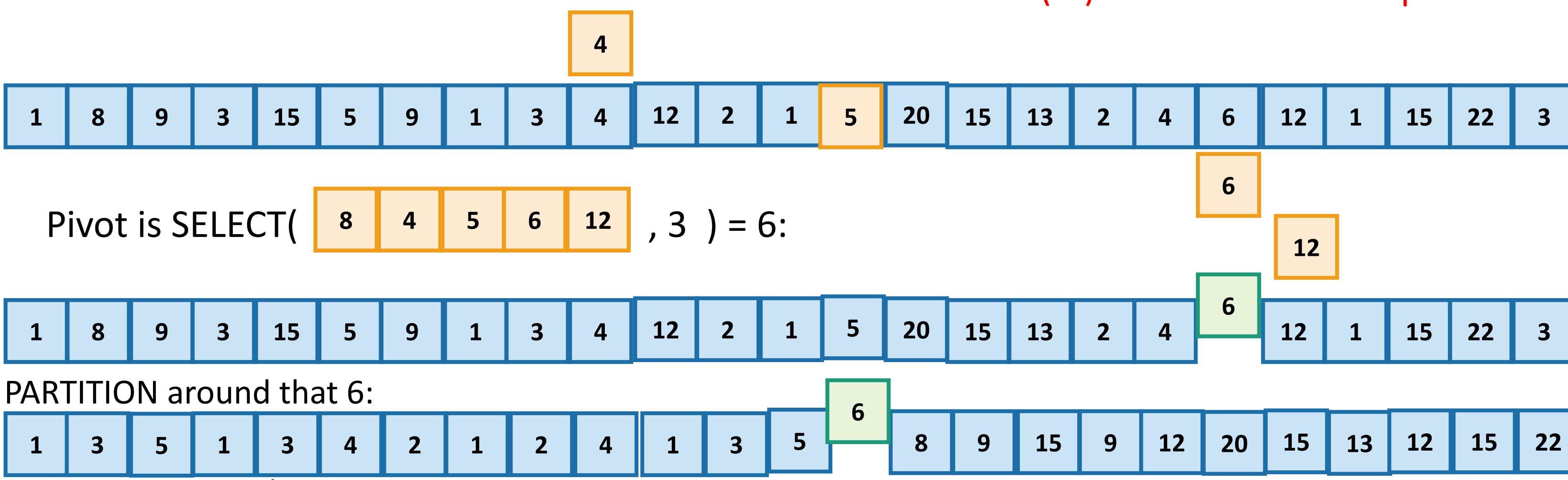


*we will make this a bit more precise.

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, …, p_m] , m/2)
  - **return** p

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

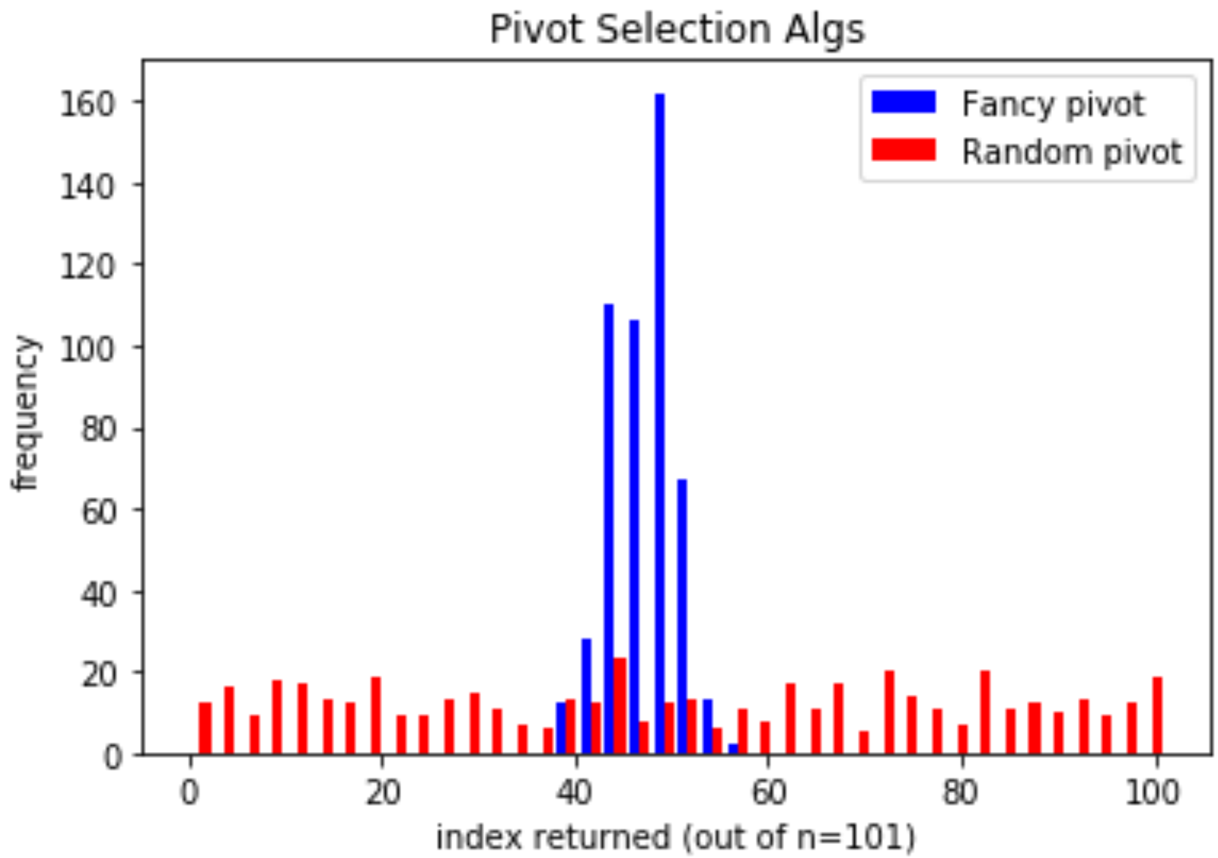


This part is L

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

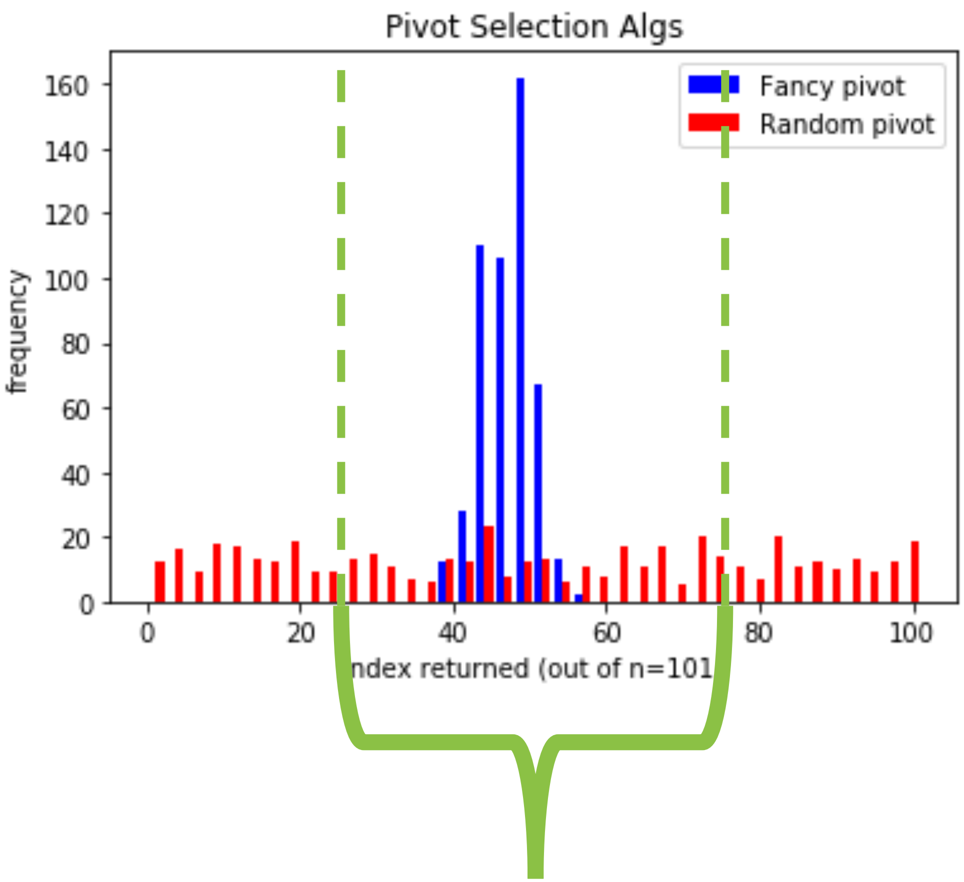
$$|L| \leq \frac{7n}{10} + 5$$

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-cast 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

- **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 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

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 Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **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

[On board]

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

Like we did last time, treat this O(n) as cn for our analysis.
(For simplicity in class – to be rigorous we should use the formal definition!)

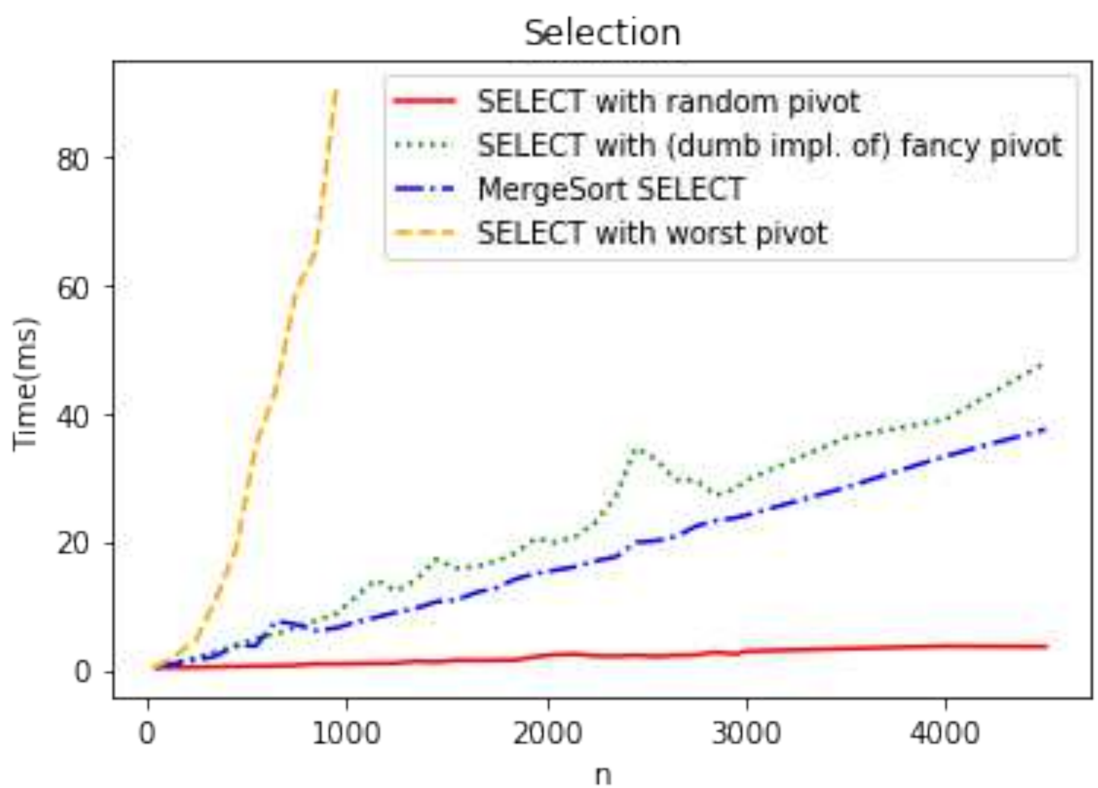
Conclusion: $T(n) = O(n)$

In practice?

- With my dumb implementation, our fancy version of **`Select`** is worse than **`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?

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 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**.

*A bad guy who knows your pivot choices ahead of time.

The Plan

1. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **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

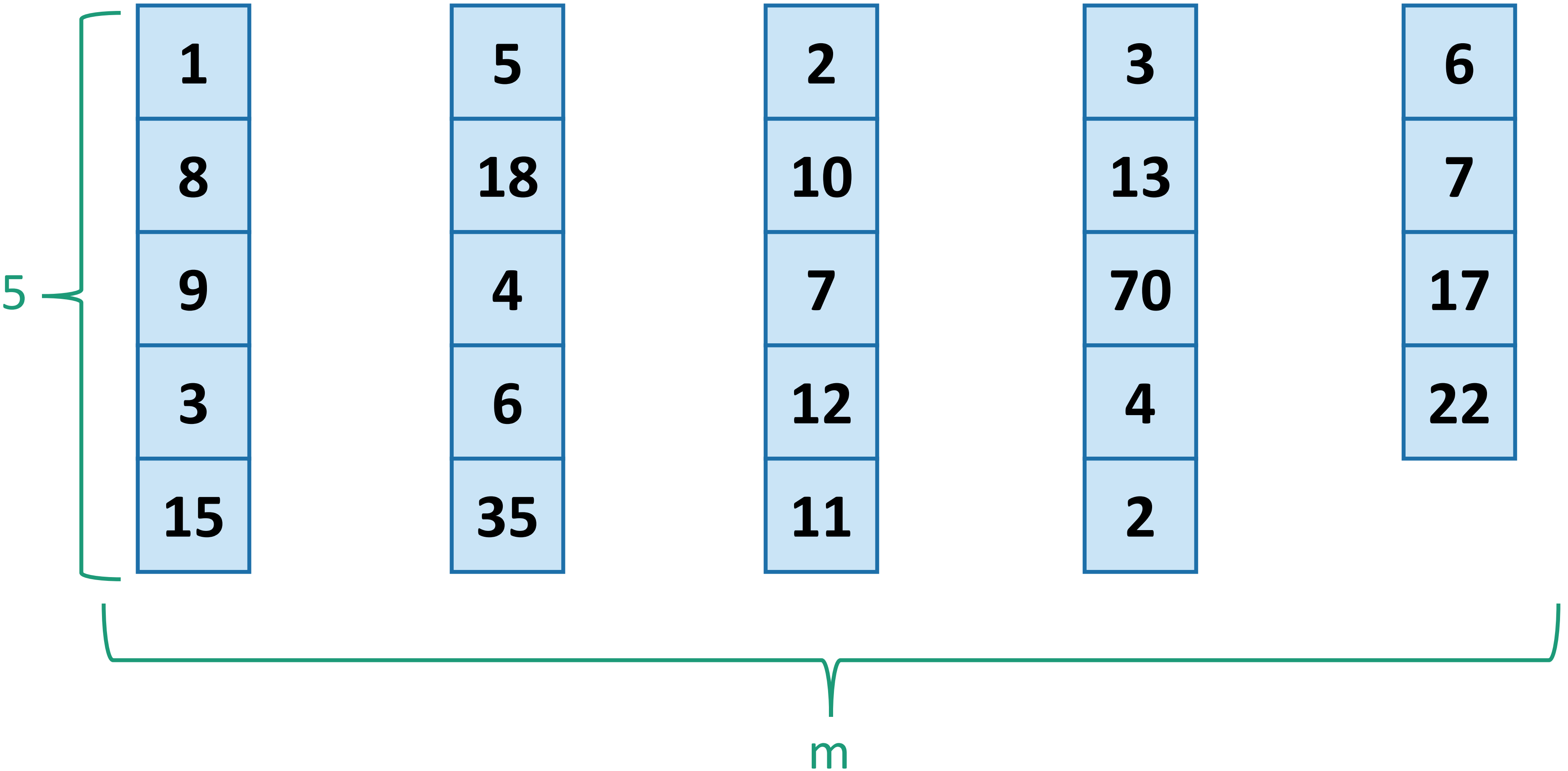
$$|L| \leq \frac{7n}{10} + 5$$

and

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

- We will see a proof by picture.
- See CLRS for proof by proof.

Proof by picture



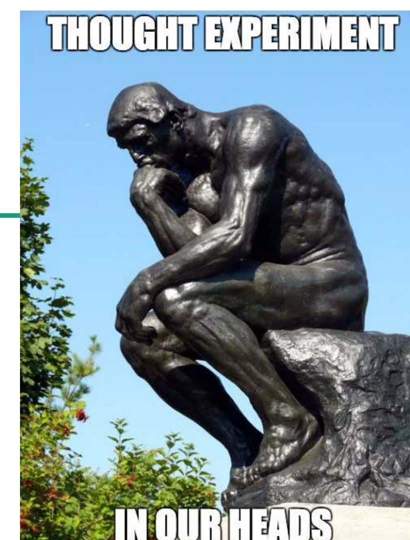
Say these are our m = [n/5] sub-arrays of size at most 5.

Proof by picture

1	4	2	2	6
3	5	7	3	7
8	6	10	4	17
9	18	11	13	22
15	35	12	70	

5

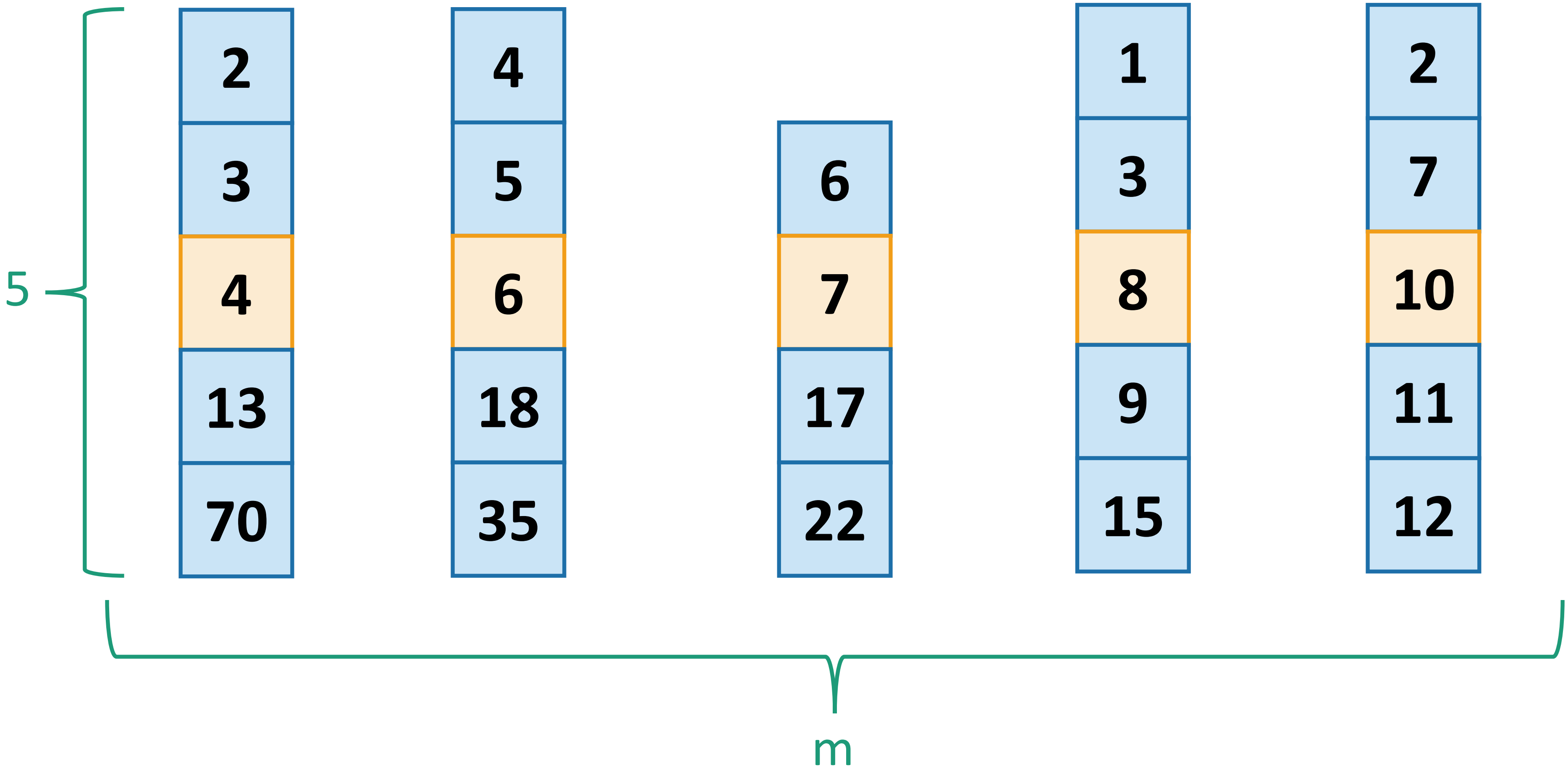
m



In our head, let's sort them.

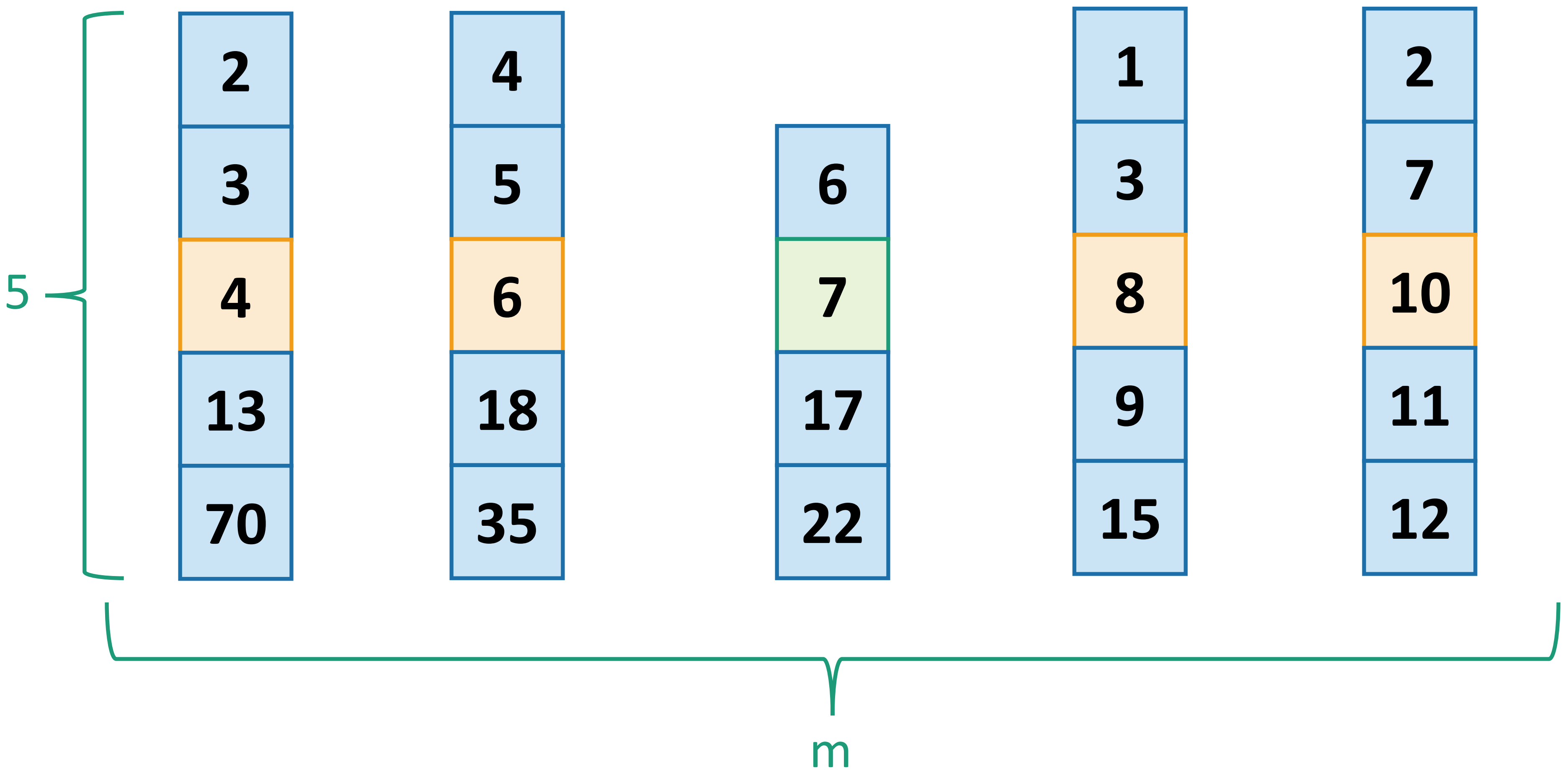
Then find medians.

Proof by picture



Then let's sort them by the median

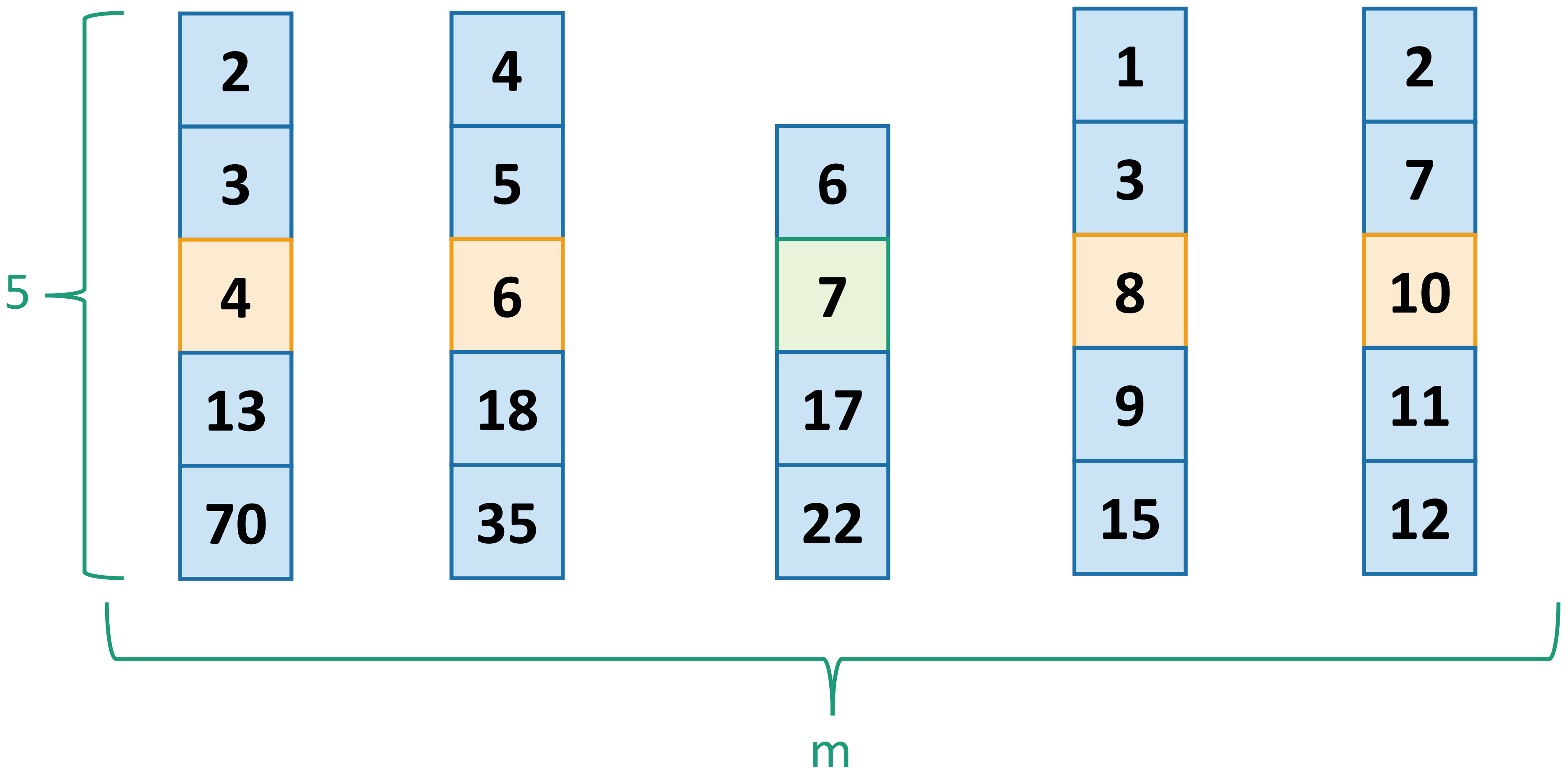
Proof by picture



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

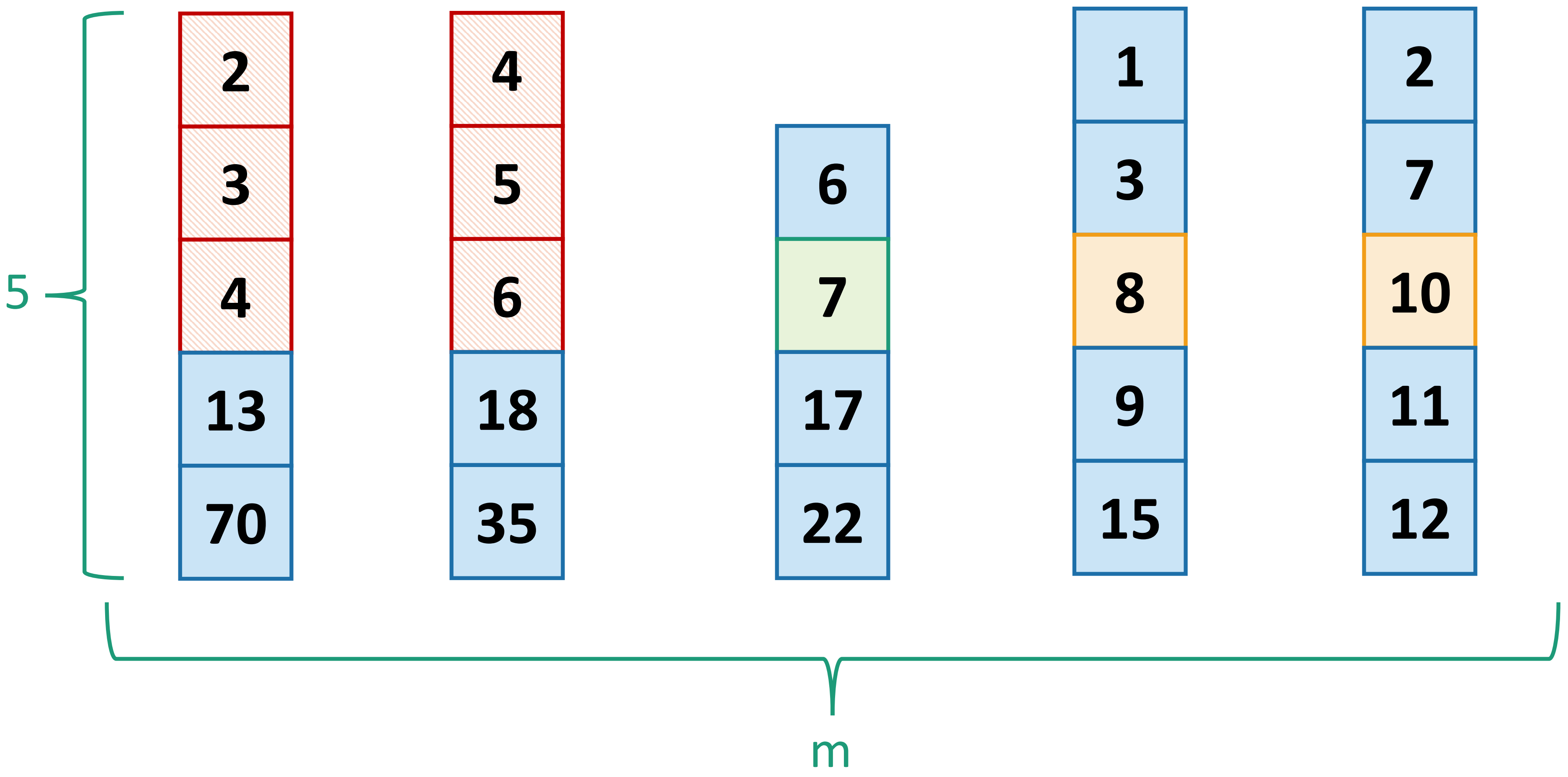
Proof by picture

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



How many elements are SMALLER than the pivot?

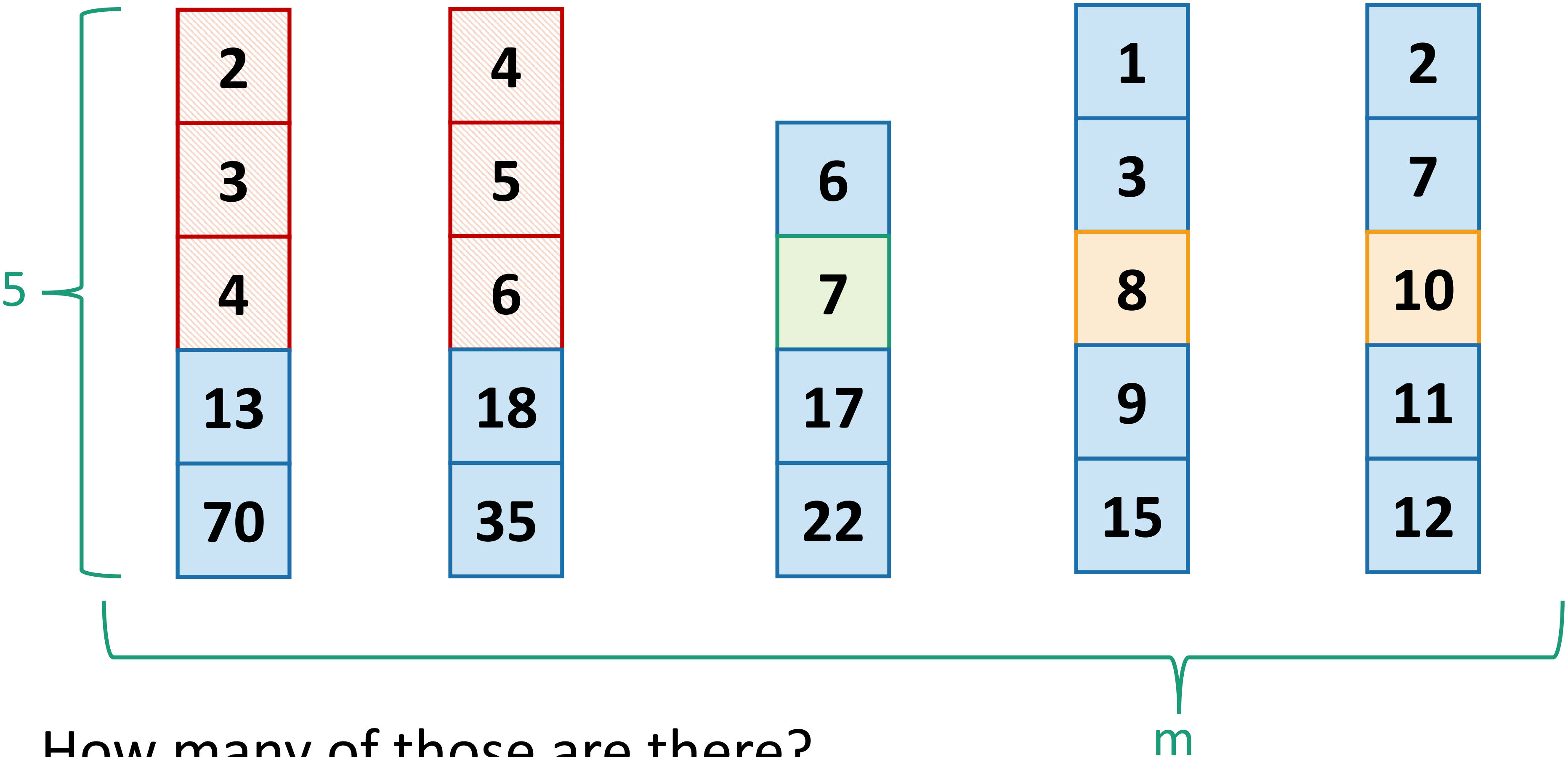
Proof by picture



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

Proof by picture

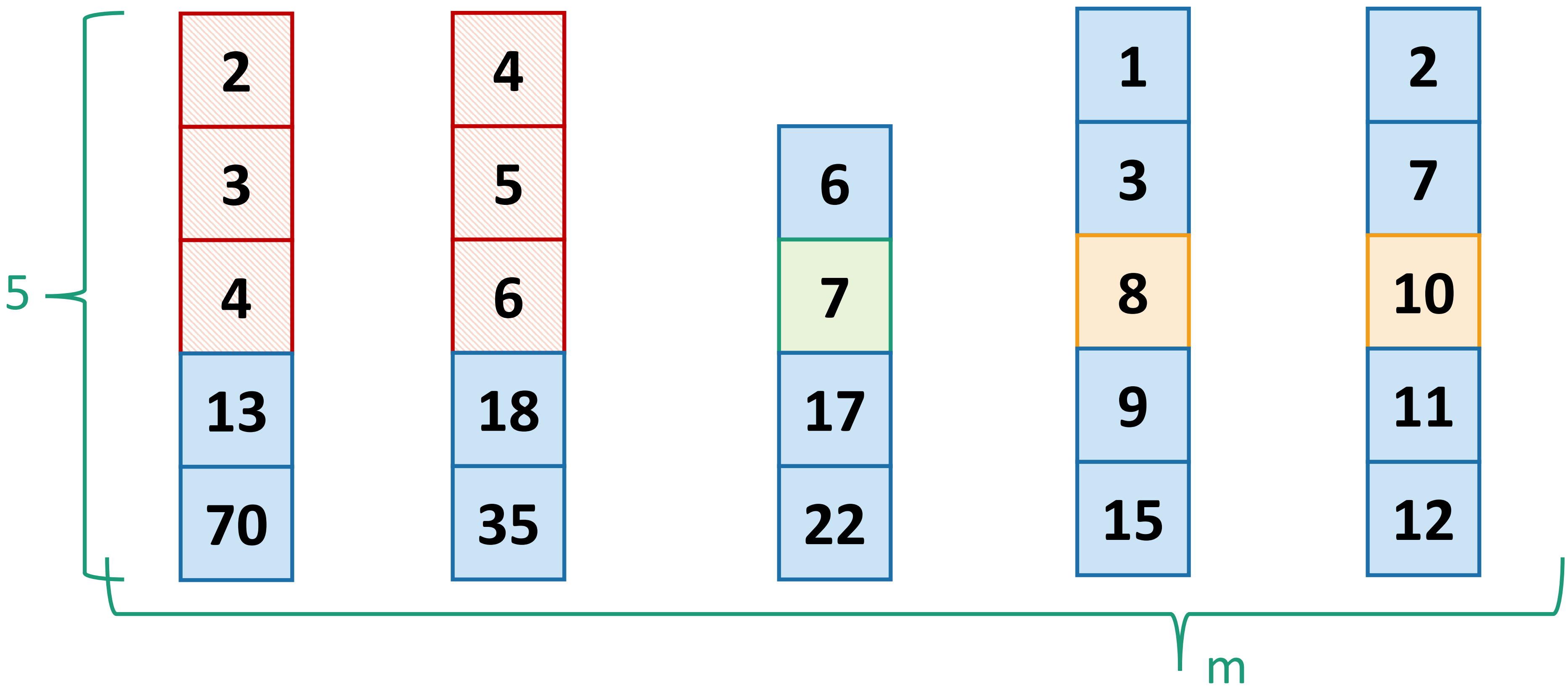
$3 \cdot \left(\left\lceil \frac{m}{2} \right\rceil - 1 \right)$ of these, but then one of them could have been the "leftovers" group.



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

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

(derivation on board)

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. The **Substitution Method**
   - You got a sneak peak on your pre-lecture exercise
2. The **SELECT** problem.
3. The **SELECT** solution.
   a) The outline of the algorithm.
   b) How to pick the pivot.
4. Return of the **Substitution Method.**

Recap

Recap

- The substitution method is another way to solve recurrence relations.
  - 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

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