CE203
ساختمان داده ها و الگوریتم ها

سجاد شیرعلی شهرضا

پاییز 1400

Based on Stanford CS161 Slides from Summer 2020 by Karey Shi

جلسه ششم و هفتم:
حل با روش جایگذاری و انتخاب kامین عضو

شنبه، 17 مهر 1400 و شنبه، 24 مهر 1400

اطلاع رسانی

- بخش مرتبط کتاب برای این جلسه: 4.3
- ارائه تمرین اول
  - مهلت ارسال تمرین اول : صبح شنبه، 24 مهر 1400 (ساعت 8 صبح)
- امتحانک اول:
  - دوشنبه همین هفته، 16 مهر 1400
  - به صورت آنلاین
  - از طریق سامانه کورسس
  - در طی ساعت کلاس

SUBSTITUTION METHOD

Algorithm, Proof of Correctness, Runtime

حل با روش جایگذاری

الگوریتم، اثبات درستی، زمان اجرا

SO FAR:

- Proving correctness:
  - Iterative algorithms: proof by induction on the iteration (e.g. Insertion Sort)
  - Recursive algorithms: proof by induction on the input size (e.g. MergeSort)
- Proving runtime:
  - Iterative algorithms: ~intuition~ (basically directly analyze the work being done)
  - Recursive algorithms: **Defining & Solving recurrence relations**

SO FAR:

- Proving correctness:
    - Iterative algorithms: proof by induction on the iteration (e.g. Insertion Sort)
    - Recursive algorithms: proof by induction on the input size (e.g. MergeSort)
- Proving runtime:
    - Iterative algorithms: ~intuition~ (basically directly analyze the work being done)
    - Recursive algorithms: **Defining & Solving recurrence relations**

We saw how to solve some recurrence relations with RECURSION TREES & the MASTER THEOREM. Here's another way:

THE SUBSTITUTION METHOD

SUBSTITUTION METHOD

1. Guess what the answer is (expand for a few iterations)
2. Prove your guess is correct (using induction)

SUBSTITUTION METHOD

1. Guess what the answer is (expand for a few iterations)
2. Prove your guess is correct (using induction)

This is a good technique to turn to if you find that the Master Theorem doesn't work. It's also especially helpful with recurrences that have differently sized subproblems (i.e. when the recursion tree & table aren't helpful either).

Let's try it on some example recurrences...

SUBSTITUTION METHOD: EXAMPLE

$$T(n) = 2 \cdot T(n/2) + n$$

$$T(1) = 1$$

STEP 1: guess what the answer is!

You can "unravel" the recursion a few steps & follow the pattern to get a closed form expression!

SUBSTITUTION METHOD: EXAMPLE

$$T(n) = 2 \cdot T(n/2) + n$$

$$T(1) = 1$$

STEP 1: guess what the answer is!

You can "unravel" the recursion a few steps & follow the pattern to get a closed form expression!

$$\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4) + n/2) + n \\
&= 4T(n/4) + 2n \\
&= 4(2(T(n/8) + n/4)) + 2n \\
&= 8(T(n/8)) + 3n \\
&= \cdots
\end{aligned}$$

SUBSTITUTION METHOD: EXAMPLE

$$T(n) = 2 \cdot T(n/2) + n$$

$$T(1) = 1$$

STEP 1: guess what the answer is!

You can "unravel" the recursion a few steps & follow the pattern to get a closed form expression!

$$\begin{aligned}
T(n) &= 2T(n/2) + n \\
&= 2(2T(n/4) + n/2) + n \\
&= 4T(n/4) + 2n \\
&= 4(2(T(n/8) + n/4)) + 2n \\
&= 8(T(n/8)) + 3n \\
&= \cdots
\end{aligned}$$

$$\begin{aligned}
\mathbf{T(n)} &= \cdots \\
&= nT(n/n) + (\log n)n \\
&= nT(1) + n \log n \\
&= n \log n + n
\end{aligned}$$

SUBSTITUTION METHOD: EXAMPLE

$$T(n) = 2 \cdot T(n/2) + n$$

$$T(1) = 1$$

STEP 1: guess what the answer is!

You can "unravel" the recursion a few steps & follow the pattern to get a closed form expression!

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(2T(n/4) + n/2) + n \\ &= 4T(n/4) + 2n \\ &= 4(2(T(n/8) + n/4)) + 2n \\ &= 8(T(n/8)) + 3n \\ &= \cdots \end{aligned}$$

$$\begin{aligned} \mathbf{T(n)} &= \cdots \\ &= nT(n/n) + (\log n)n \\ &= nT(1) + n \log n \\ &= n \log n + n \end{aligned}$$

let's guess that $T(n) = n \log n + n$ and try to prove it!

SUBSTITUTION METHOD: EXAMPLE

$T(n) = 2 \cdot T(n/2) + n$

$T(1) = 1$

Our guess from Step 1:

$\mathbf{T(n) = n \log n + n}$

STEP 2: Try to prove your guess!

SUBSTITUTION METHOD: EXAMPLE

$T(n) = 2 \cdot T(n/2) + n$
$T(1) = 1$

Our guess from Step 1:
$T(n) = n \log n + n$

STEP 2: Try to prove your guess!

- **Inductive Hypothesis**: $T(n) = n \log n + n$

SUBSTITUTION METHOD: EXAMPLE

$T(n) = 2 \cdot T(n/2) + n$

$T(1) = 1$

Our guess from Step 1:

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

STEP 2: Try to prove your guess!

- **Inductive Hypothesis**: $T(n) = n \log n + n$
- **Base case**: Prove IH holds for $n = 1$. $T(1) = 1 = 1 \log 1 + 1$.

SUBSTITUTION METHOD: EXAMPLE

$T(n) = 2 \cdot T(n/2) + n$

$T(1) = 1$

Our guess from Step 1:

T(n) = n log n + n

STEP 2: Try to prove your guess!

- **Inductive Hypothesis**: $T(n) = n \log n + n$
- **Base case**: Prove IH holds for $n = 1$. $T(1) = 1 = 1 \log 1 + 1$.
- **Inductive step**:
    - Let $k > 1$. Assume that the IH holds for all n such that $1 \le n < k$.
    - $$\begin{aligned} T(k) &= 2 \cdot T(k/2) + k \\ &= 2 \cdot ((k/2)(\log (k/2)) + (k/2)) + k \\ &= 2 \cdot ((k/2)(\log k - 1 + 1)) + k \\ &= 2 \cdot (k/2)(\log k) + k \\ &= k \log k + k \end{aligned}$$

SUBSTITUTION METHOD: EXAMPLE

$T(n) = 2 \cdot T(n/2) + n$
$T(1) = 1$

Our guess from Step 1:
$T(n) = n \log n + n$

STEP 2: Try to prove your guess!

- **Inductive Hypothesis**: $T(n) = n \log n + n$
- **Base case**: Prove IH holds for $n = 1$. $T(1) = 1 = 1 \log 1 + 1$.
- **Inductive step**:
  - Let $k > 1$. Assume that the IH holds for all n such that $1 \leq n < k$.
  - $$\begin{aligned} T(k) &= 2 \cdot T(k/2) + k \\ &= 2 \cdot ((k/2)(\log (k/2)) + (k/2)) + k \\ &= 2 \cdot ((k/2)(\log k - 1 + 1)) + k \\ &= 2 \cdot (k/2)(\log k) + k \\ &= k \log k + k \end{aligned}$$
- **Conclusion:** By induction, $T(n) \leq n \log n + n$ for all $n > 0$.

This satisfies the Big-O definition for $O(n \log n)$ (imagine choosing $c = 2$, $n_0 = 1$)

SUBSTITUTION METHOD: EXAMPLE 1

$T(n) \le n \log n + n$
for all $n > 0$ ⇒ **$T(n) = O(n \log n)$**

1. Guess what the answer is (expand for a few iterations)
2. Prove your guess is correct (using induction)

SUBSTITUTION METHOD: EXAMPLE 1

$T(n) \leq n \log n + n$ for all $n > 0$ $\Rightarrow$ $T(n) = O(n \log n)$

1. Guess what the answer is (expand for a few iterations)
2. Prove your guess is correct (using induction)

But sometimes expanding gets complicated...

سوال؟

SUBSTITUTION METHOD: EXAMPLE 2

$$T(n) = T(n/5) + T(7n/10) + n$$

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

Note:

While Example 1 could have also been solved with the Master Theorem, this one has differently sized subproblems, so the Master Theorem won't apply.

So… Time to use the Substitution Method!

SUBSTITUTION METHOD: EXAMPLE 2

$$T(n) = T(n/5) + T(7n/10) + n$$

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

STEP 1: guess what the answer is!

Unraveling this expression gets ugly… (feel free to try it!).

You can also make a semi-educated guess and just hope for the best.

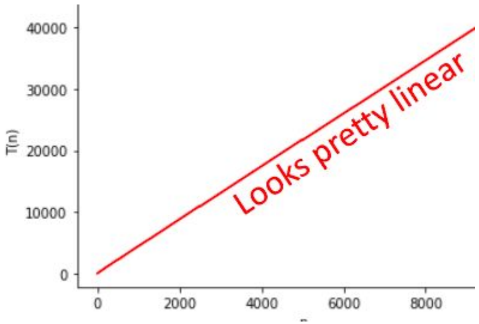
SUBSTITUTION METHOD: EXAMPLE 2

$$T(n) = T(n/5) + T(7n/10) + n$$

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

STEP 1: guess what the answer is!

Unraveling this expression gets ugly… (feel free to try it!).
You can also make a semi-educated guess and just hope for the best.



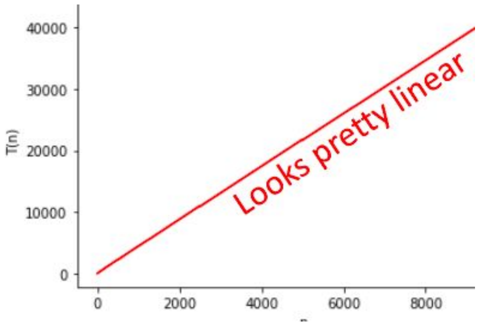
SUBSTITUTION METHOD: EXAMPLE 2

$$T(n) = T(n/5) + T(7n/10) + n$$

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

STEP 1: guess what the answer is!

Unraveling this expression gets ugly… (feel free to try it!).
You can also make a semi-educated guess and just hope for the best.



It also feels like it could be better than 2T(n/2) + n, which we know to be O(n log n)...

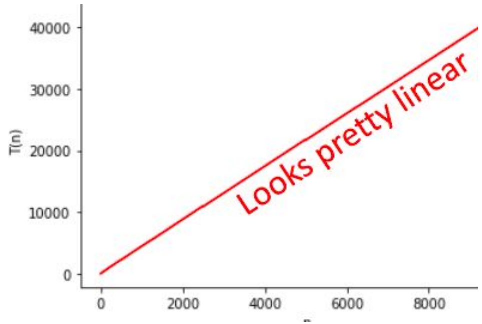
SUBSTITUTION METHOD: EXAMPLE 2

$$T(n) = T(n/5) + T(7n/10) + n$$

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

STEP 1: guess what the answer is!

Unraveling this expression gets ugly… (feel free to try it!).
You can also make a semi-educated guess and just hope for the best.



It also feels like it could be better than 2T(n/2) + n, which we know to be O(n log n)...

Let's guess O(n)

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \leq n \leq 10$

$\Rightarrow$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \leq n \leq 10$

Our guess from Step 1:

$T(n)$ is $O(n)$

STEP 2: Prove it!

WARNING:

You might be tempted to prove this with the inductive hypothesis "$T(n) = O(n)$"

But that doesn't make sense! Formally, this is what your IH would be saying:

"There is some $n_0 > 0$ and some $C > 0$ such that for all $n \geq n_0$, $T(n) \leq C \cdot n$"

Your IH is supposed to hold for a *specific* n, not an unbounded *range* of n!

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \leq n \leq 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

You ... esis

But t... ing:

"There is some n_0 ... 0 and some C ... 0 such that for all $n \geq n_0$, $T(n) \leq C \cdot n$"

Instead, we need to pick a C first,
and have our inductive hypothesis be

$T(n) \leq C \cdot n$

Your IH is supposed to hold for a *specific* n, not an unbounded *range* of n!

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \leq n \leq 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

Use a placeholder **C** constant in the big-O proof. We don't know what C should be yet, but let's go through the proof leaving it as C and then figure out what works.

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \leq n \leq 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

Use a placeholder **C** constant in the big-O proof. We don't know what C should be yet, but let's go through the proof leaving it as C and then figure out what works.

- **Inductive Hypothesis**: $T(n) \leq Cn$
- **Base case**: Prove IH holds for $1 \leq n \leq 10$. $T(n) = 1 \leq Cn$

Whatever we choose C to be, we know C needs to be at least 1

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \le n \le 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

Use a placeholder **C** constant in the big-O proof. We don't know what C should be yet, but let's go through the proof leaving it as C and then figure out what works.

- **Inductive Hypothesis**: $T(n) \le Cn$
- **Base case**: Prove IH holds for $1 \le n \le 10$. $T(n) = 1 \le Cn$
- **Inductive step**:
  - Let $k > 10$. Assume that the IH holds for all n such that $1 \le n < k$.

Whatever we choose C to be, we know C needs to be at least 1

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \le n \le 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

Use a placeholder **C** constant in the big-O proof. We don't know what C should be yet, but let's go through the proof leaving it as C and then figure out what works.

- **Inductive Hypothesis**: $T(n) \le \mathbf{C}n$
- **Base case**: Prove IH holds for $1 \le n \le 10$. $T(n) = 1 \le \mathbf{C}n$
- **Inductive step**:
  - Let $k > 10$. Assume that the IH holds for all n such that $1 \le n < k$.
  - $$\begin{aligned} T(k) &= k + T(k/5) + T(7k/10) \\ &\le k + \mathbf{C}\cdot(k/5) + \mathbf{C}\cdot(7k/10) \\ &= k\cdot(1 + \mathbf{C}/5 + 7\mathbf{C}/10) \\ &\le \mathbf{C}k \quad ??? \end{aligned}$$
  - (If we find the right C, then we've shown IH holds for n = k)

Whatever we choose C to be, we know C needs to be at least 1

We can just solve for C:

$$1 + \mathbf{C}/5 + 7\mathbf{C}/10 \le \mathbf{C}$$
$$1 + 9\mathbf{C}/10 \le \mathbf{C}$$
$$1 \le \mathbf{C}/10$$

So let's choose C = 10!

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \le n \le 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

We can choose C = 10!

- **Inductive Hypothesis**: $T(n) \le \mathbf{10}n$
- **Base case**: Prove IH holds for $1 \le n \le 10$. $T(n) = 1 \le \mathbf{10}n$
- **Inductive step**:
  - Let $k > 10$. Assume that the IH holds for all n such that $1 \le n < k$.
  - $$\begin{aligned} T(k) &= k + T(k/5) + T(7k/10) \\ &\le k + \mathbf{10} \cdot (k/5) + \mathbf{10} \cdot (7k/10) \\ &= k + 2k + 7k \\ &= \mathbf{10}k \end{aligned}$$
  - Thus, the IH holds for $n = k$!
- **Conclusion:** With $C = 10$ and $n_0 = 1$, $T(n) \le Cn$ for all $n \ge n_0$. By the Big-O definition, $T(n) = O(n)$.

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \le n \le 10$

Our guess from Step 1:

$T(n)$ is $O(n)$

STEP 2: Prove it!

We can choose $C = 10$!

- **Induct**
- **Base**
- **Induc**
  -
  -

    $= \mathbf{10}k$
  - Thus, the IH holds for $n = k$!
- **Conclusion:** With $C = 10$ and $n_0 = 1$, $T(n) \le Cn$ for all $n \ge n_0$. By the Big-O definition, $T(n) = O(n)$.

Yay! Our guess worked!

But what if you make a bad guess?

سوال؟

WHAT IF YOU MAKE A BAD GUESS?

$$T(n) = 2 \cdot T(n/2) + n$$
$$T(1) = 1$$

Bad guess:

$T(n) = O(n)$

WHAT IF YOU MAKE A BAD GUESS?

$T(n) = 2 \cdot T(n/2) + n$

$T(1) = 1$

Bad guess:

$T(n) = O(n)$

STEP 2: Prove it!

Use a placeholder C constant in the big-O proof. We don't know what C should be yet, but let's go through the proof leaving it as C and see if we run into any trouble.

- Inductive Hypothesis: $T(n) \le Cn$
- Base case: $1 = T(n) \le Cn$ for $n = 1$.
- Inductive step:
  - Let $k > 1$. Assume that the IH holds for all n such that $1 \le n < k$.
  - $$\begin{aligned} T(k) &= 2 \cdot T(k/2) + k \\ &\le 2 \cdot C(k/2) + k \\ &= Ck + k \\ &\le Ck??? \end{aligned}$$

WHAT IF YOU MAKE A BAD GUESS?

$T(n) = 2 \cdot T(n/2) + n$

$T(1) = 1$

Bad guess:

$T(n) = O(n)$

STEP 2: Prove it!

Use a placeholder C constant in the big-O proof. We don't know what C should be yet, but let's go through the proof leaving it as C and see if we run into any trouble.

- Inductive Hypothesis: $T(n) \le Cn$
- Base case: $1 = T(n) \le Cn$ for $n = 1$.
- Inductive step:
  - Let $k > 1$. Assume that the IH holds for all n such that $1 \le n < k$.
  - $$\begin{aligned} T(k) &= 2 \cdot T(k/2) + k \\ &\le 2 \cdot C(k/2) + k \\ &= Ck + k \\ &\le Ck??? \end{aligned}$$

We need this inequality to hold for the Inductive Step to be complete. However, no choice of C could ever make $Ck + k \le Ck$!

WHAT IF YOU MAKE A BAD GUESS?

$T(n) = 2 \cdot T(n/2) + n$

$T(1) = 1$

Bad guess:

$T(n) = O(n)$

STEP 2: Prove it!

Use a placeholder C constant in the big-O proof. We don't know what C should be yet, but let's go

A few tips:

If you stumble across impossible inequalities, then your guess was too small! If you end up with an inequality that seems too loose (e.g. $k \le k^2$, $\log k \le k$), maybe try a smaller guess.

hold

for the inductive step to be complete. However, no choice of C could ever make Ck+ k ≤ Ck!

$= \quad Ck + k$

$\le \quad Ck???$

SO WHAT HAVE WE LEARNED?

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

1. **Guess what the answer is (expand for a few iterations)**
2. **Prove your guess is correct (using induction)**

In your final proof, pretend like you didn't do Steps 1 & 2 - no need to say how you unraveled the expression or why you made your guess. Just make sure your proof checks out!

سوال؟

انتخاب kامین عضو

الگوریتم، اثبات درستی، زمان اجرا

THE SELECT PROBLEM

INPUT:

an unsorted array **A** of n elements (assume all elements are distinct),
& an integer **k** in {1, …, n}

7	2	6	9	1	5	4	11

OUTPUT of SELECT(A, k): the k^{th} smallest element of A

THE SELECT PROBLEM

INPUT:
an unsorted array **A** of n elements (assume all elements are distinct),
& an integer **k** in {1, …, n}

7	2	6	9	1	5	4	11

OUTPUT of SELECT(A, k): the k^{th} smallest element of A

SELECT(A, 1) = 1
SELECT(A, 2) = 2
SELECT(A, 3) = 4
SELECT(A, 8) = 11

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

Note: k is a 1-indexed number!

THE SELECT PROBLEM

INPUT:

an unsorted array **A** of n elements (assume all elements are distinct),
& an integer **k** in {1, …, n}

7	2	6	9	1	5	4	11

OUTPUT of SELECT(A, k): the k^{th} smallest element of A

Can you come up with an O(n log n) algorithm for SELECT?

AN O(n log n) ALGORITHM

```
SELECT(A,k):
    A = MERGESORT(A)
    return A[k-1]
```

It's k-1 (rather than k) since my pseudocode is 0-indexed and k is a 1-indexed number

Okay, great! We're done!

سوال؟

AN O(n log n) ALGORITHM

SE

THE QUESTION IS...

CAN WE DO BETTER?

It's k-1 (rather than k) since my pseudocode is 0-indexed and k is a 1-indexed number

~~Okay, great! We're done!~~

GOAL: AN O(n) ALGORITHM

If k = 1, then we want the minimum of A. There's an easy O(n) algorithm for that:

Pretty much the same if k = n (we're just finding MAX(A) instead)

GOAL: AN O(n) ALGORITHM

If k = 1, then we want the minimum of A. There's an easy O(n) algorithm for that:

Pretty much the same if k = n (we're just finding MAX(A) instead)

```
SELECT-1(A):
    result = infinity
    for i in [0,...,n-1]:
        if A[i] < result:
            result = A[i]
    return result
```

This loop runs O(n) times

The body of each iteration is O(1) work.

Runtime of SELECT-1: O(n)

GOAL: AN O(n) ALGORITHM

If k = 2, then we want the second-smallest element in A.

There's an easy-ish O(n) algorithm for that:

(Not a very important algorithm, because this will end up being a bad idea…)

GOAL: AN O(n) ALGORITHM

If k = 2, then we want the second-smallest element in A.
There's an easy-ish O(n) algorithm for that:

(Not a very important algorithm, because this will end up being a bad idea...)

```
SELECT-2(A):
    result = infinity
    minSoFar = infinity
    for i in [0,...,n-1]:
        if A[i] < result & A[i] < minSoFar:
            result = minSoFar
            minSoFar = A[i]
        else if A[i] < result & A[i] >= minSoFar
            result = A[i]
    return result
```

This loop runs O(n) times

The body of each iteration is still O(1) work.

Runtime of SELECT-2: O(n)

GOAL: AN O(n) ALGORITHM

If k = n/2, then we want the median element in A.

```
SELECT-n/2(A):
    result = infinity
    minSoFar = infinity
    secondMinSoFar = infinity
    thirdMinSoFar = infinity
    fourthMinSoFar = infinity
    fifthMinSoFar = infinity
    ...
```

GOAL: AN O(n) ALGORITHM

If k = n/2, then we want the median element in A.

```
SELECT-n/2(A):
    result = infinity
    minSoFar = infinity
    secondMinSoFar = infinity
    thirdMinSoFar = infinity
    fourthMinSoFar = infinity
    fifthMinSoFar = infinity
    ...
```

Runtime of SELECT-n/2: $O(n^2)$

Clearly, this algorithm style isn't a good idea for large k (e.g. n/2).
This basically ends up looking like InsertionSort.

LINEAR SELECTION: THE IDEA

Let's use DIVIDE-and-CONQUER!

LINEAR SELECTION: THE IDEA

Let's use DIVIDE-and-CONQUER!

Select a pivot

Partition around it

Recurse!

LINEAR SELECTION: THE IDEA

Let's use DIVIDE-and-CONQUER!

Select a pivot

Partition around it

Recurse!

kind of like a "binary search" for the k^{th} smallest element (except that the array isn't sorted!)

LINEAR SELECTION: THE IDEA

3	2	9	8	1	6	4	11

LINEAR SELECTION: THE IDEA

Select a pivot

3	2	9	8	1	6	4	11

How do we pick a pivot?? We'll see this later.
For now, imagine we pick it randomly.

LINEAR SELECTION: THE IDEA

Select a pivot

3	2	9	8	1	6	4	11

How do we pick a pivot?? We'll see this later.
For now, imagine we pick it randomly.

Partition around it

L

3	2	1	4

6

9	8	11

R

Partition around pivot: **L** has elements less than pivot, and **R** has elements greater than pivot.
(Note that **L** and **R** remain unsorted).

LINEAR SELECTION: THE IDEA

Select a pivot

3	2	9	8	1	6	4	11

How do we pick a pivot?? We'll see this later.
For now, imagine we pick it randomly.

Partition around it

L

3	2	1	4

6

9	8	11

R

Partition around pivot: **L** has elements less than pivot, and **R** has elements greater than pivot.
(Note that **L** and **R** remain unsorted).

Recurse!

The pivot is in position **5**. We have three cases:

1. **if k = 5: return pivot** — the k^{th} smallest element is the pivot!
2. **if k < 5: return SELECT(L, k)** — the k^{th} smallest element lives in L
3. **if k > 5: return SELECT(R, k-5)** — the k^{th} smallest element is the $(k\text{-}5)^{th}$ smallest element in R

LINEAR SELECTION: EXAMPLE

SELECT`(A, 7):`

1	12	4	20	31	6	18	9

LINEAR SELECTION: EXAMPLE

PICK A PIVOT
How do we pick a pivot???
We'll see later...

SELECT`(A, 7):`

1	12	4	20	31	6	18	9

LINEAR SELECTION: EXAMPLE

SELECT`(A, 7):`

1	12	4	20	31	6	18	9

PARTITION

L

1	12	4	6	9

18

20	31

R

LINEAR SELECTION: EXAMPLE

SELECT`(A, 7):`

1	12	4	20	31	6	18	9

L

1	12	4	6	9

18

20	31

R

Recurse here (since 18 occupies index 6 and k = 7 > 6)

RECURSE

SELECT`(R, 1):`

20	31

1 = 7 - 6
(aka k minus pivot position)

LINEAR SELECTION: EXAMPLE

SELECT`(A, 7)`:

1	12	4	20	31	6	18	9

L

1	12	4	6	9

18

20	31

R

Recurse here (since 18 occupies index 6 and k = 7 > 6)

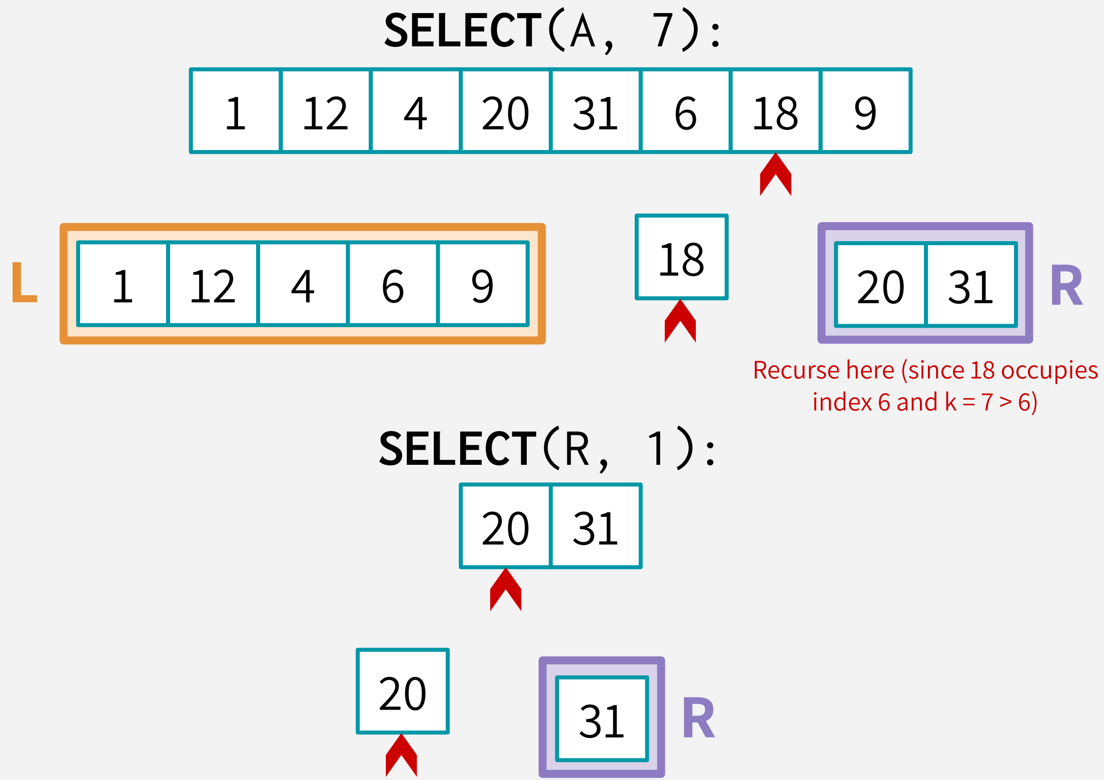
SELECT`(R, 1)`:

20	31

PICK A PIVOT

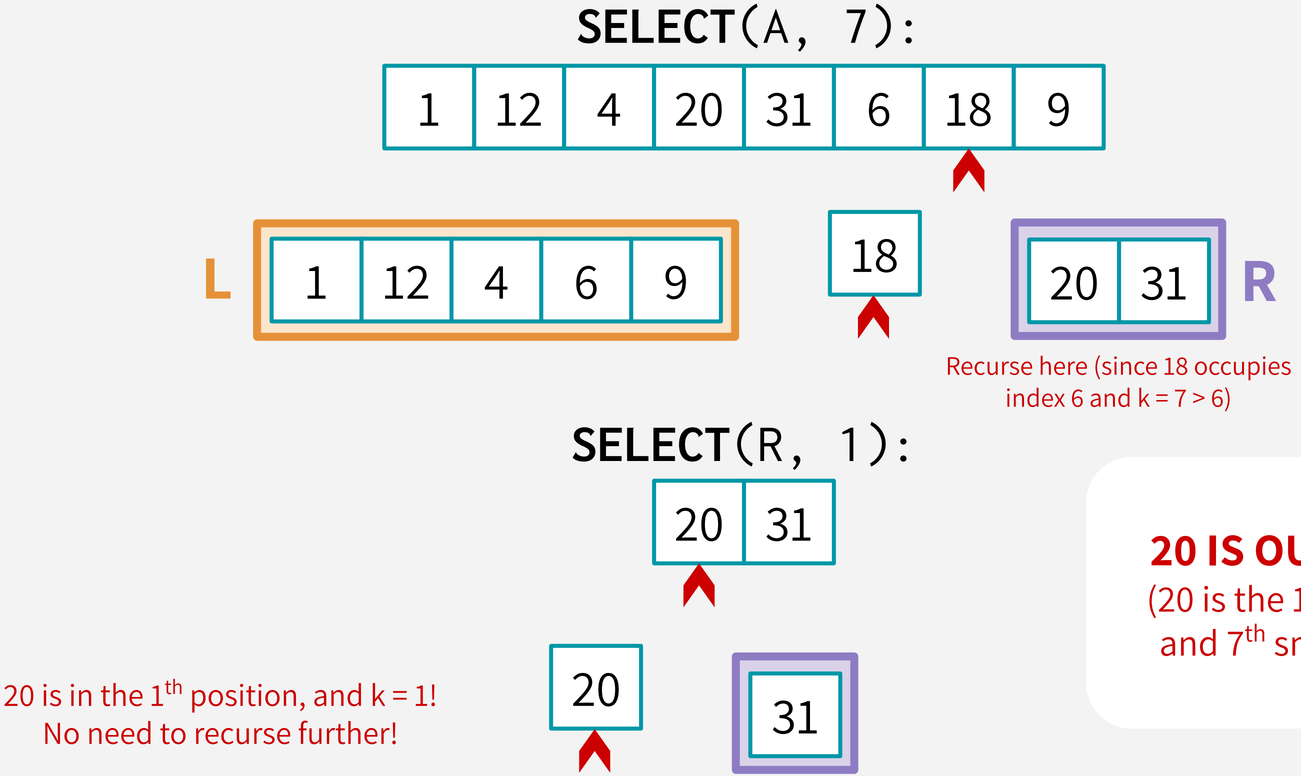
How do we pick a pivot???
We'll see later...

LINEAR SELECTION: EXAMPLE



PARTITION

LINEAR SELECTION: EXAMPLE



20 IS OUR ANSWER!
(20 is the 1th smallest in R, and 7th smallest overall)

LINEAR SELECTION: PSEUDOCODE

Base Case:
if len(A) = 1, then just go ahead and return the element itself

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else:
        return SELECT(R, k-len(L)-1)
```

Case 1:
We got lucky and found exactly the k^{th} smallest!

Case 2:
The k^{th} smallest is in the first part of the array (L)

Case 3:
The k^{th} smallest is in the second part of the array (R)

LINEAR SELECTION: PSEUDOCODE

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else:
        return SELECT(R, k-len(L)-1)
```

```
PARTITION(A, pivot):
    L, R = [], []
    for i in [1,...,len(A)]:
        if A[i] == pivot:
            continue
        else if A[i] < pivot:
            add A[i] to L
        else:
            add A[i] to R
```

سوال؟

LINEAR SELECTION: SO FAR

- Intuition:
  - Partition the array around a pivot (how do we select?? still TBD)
  - Either return the pivot itself or recurse on the left or right subarrays (but not both!)

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:      return p
    else if len(L) > k-1:  return SELECT(L, k)
    else:                  return SELECT(R, k-len(L)-1)
```

LINEAR SELECTION: SO FAR

- Intuition:
  - Partition the array around a pivot (how do we select?? still TBD)
  - Either return the pivot itself or recurse on the left or right subarrays (but not both!)
- Our two favorite questions:
  - Does this work?
  - What’s the runtime?

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:     return p
    else if len(L) > k-1: return SELECT(L, k)
    else:                 return SELECT(R, k-len(L)-1)
```

LINEAR SELECTION: DOES IT WORK?

FROM LAST WEEK!

RECURSIVE ALGORITHMS

1. **Inductive hypothesis**: your algorithm is correct for sizes *up to* **i**
2. **Base case**: IH holds for i < small constant
3. **Inductive step**:
   - assume IH holds for k ⇒ prove k+1, *OR*
   - assume IH holds for {1,2,...,k-1} ⇒ prove k.
4. **Conclusion**: IH holds for i = n ⇒ yay!

INDUCTION PROOF

INDUCTIVE HYPOTHESIS (IH)

When run on an array A of size **i** and an integer $1 \le k \le$ **i**, SELECT(A,k) correctly returns the k^{th} smallest element of A.

INDUCTION PROOF

INDUCTIVE HYPOTHESIS (IH)

When run on an array A of size **i** and an integer $1 \le k \le$ **i**, SELECT(A,k) correctly returns the k^{th} smallest element of A.

BASE CASE

The IH holds for i = 1: We know k must be 1, so SELECT does indeed return the smallest (and only) element of A.

INDUCTION PROOF

INDUCTIVE HYPOTHESIS (IH)

When run on an array A of size **i** and an integer $1 \le k \le$ **i**, SELECT(A,k) correctly returns the k^{th} smallest element of A.

BASE CASE

The IH holds for i = 1: We know k must be 1, so SELECT does indeed return the smallest (and only) element of A.

(OUTLINE OF) INDUCTIVE STEP *(strong/complete induction)*

Let j be an integer, where $j > 1$. Assume that the IH holds for all i where $1 \le i < j$. We want to show that the IH holds for $i = j$, i.e. that for an array A of size j and an integer $k \le j$, SELECT returns the k^{th} smallest element of A.

We consider three cases, depending on the pivot chosen by GET_PIVOT. PARTITION gives us L, and R.

- **CASE 1**: $|L| = k-1$.
- **CASE 2**: $|L| > k-1$.
- **CASE 3**: $|L| < k-1$.

We use STRONG induction because cases 2 and 3 rely on the correctness of the smaller recursive calls.

Thus, in each of the three cases, SELECT(A,k) returns the k^{th} smallest element of A. This establishes the IH for $i = j$.

INDUCTION PROOF

INDUCTIVE HYPOTHESIS (IH)

When run on an array A of size **i** and an integer $1 \leq k \leq$ **i**, SELECT(A,k) correctly returns the k^{th} smallest element of A.

BASE CASE

The IH holds for i = 1: We know k must be 1, so SELECT does indeed return the smallest (and only) element of A.

(OUTLINE OF) INDUCTIVE STEP *(strong/complete induction)*

Let j be an integer, where j > 1. Assume that the IH holds for all i where $1 \leq i < j$. We want to show that the IH holds for i = j, i.e. that for an array A of size j and an integer $k \leq j$, SELECT returns the k^{th} smallest element of A.

We consider three cases, depending on the pivot chosen by GET_PIVOT. PARTITION gives us L, and R.

- **CASE 1**: |L| = k-1.
- **CASE 2**: |L| > k-1.
- **CASE 3**: |L| < k-1.

We use STRONG induction because cases 2 and 3 rely on the correctness of the smaller recursive calls.

Thus, in each of the three cases, SELECT(A,k) returns the k^{th} smallest element of A. This establishes the IH for i = j.

CONCLUSION

By induction, we conclude that the IH holds for all $1 \leq i \leq n$. Thus, we conclude that SELECT(A, k) returns the k^{th} smallest element of A on any array A, provided that $1 \leq k \leq |A|$. That is, SELECT is correct!

سوال؟

RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else if len(L) < k-1:
        return SELECT(R, k-len(L)-1)
```

Recurrence Relation for SELECT

For now, assume we'll pick the pivot in time O(n)

RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else if len(L) < k-1:
        return SELECT(R, k-len(L)-1)
```

Recurrence Relation for SELECT

For now, assume we'll pick the pivot in time O(n)

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

RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else if len(L) < k-1:
        return SELECT(R, k-len(L)-1)
```

Recurrence Relation for SELECT

For now, assume we'll pick the pivot in time O(n)

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

But what are `len(L)` and `len(R)`?
That depends on how we pick the pivot...

RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = GET_PIVOT(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else if len(L) < k-1:
        return SELECT(R, k-len(L)-1)
```

What's a "good" pivot?
What's a "bad" pivot?

e Relation for SELECT

we'll pick the pivot in time O(n)

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

But what are `len(L)` and `len(R)`?
That depends on how we pick the pivot...

THE WORST PIVOT

The WORST pivot: picking the max or the min each time!

Then, in the worst case, the recurrence relation looks like T(n) = T(n-1) + O(n).

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

THE WORST PIVOT

The WORST pivot: picking the max or the min each time!

Then, in the worst case, the recurrence relation looks like $T(n) = T(n-1) + O(n)$.

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

This ends up being $\Omega(n^2)$!

A call to SELECT(A, n/2) would already consist of ~n/2 recursive calls
(each with a subarray of length at least n/2)!

THE IDEAL PIVOT

The IDEAL pivot: splits the input array exactly in half!

`len(L)` = `len(R)` = (n-1)/2

$$T(n) = \begin{cases} O(n) & \texttt{len(L) == k-1} \\ T(\texttt{len(L)}) + O(n) & \texttt{len(L) > k-1} \\ T(\texttt{len(R)}) + O(n) & \texttt{len(L) < k-1} \end{cases} \Rightarrow \mathbf{T(n) \leq T(n/2) + O(n)}$$

THE IDEAL PIVOT

The IDEAL pivot: splits the input array exactly in half!

`len(L)` = `len(R)` = (n-1)/2

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

$\Rightarrow$

$$\mathbf{T(n) \leq T(n/2) + O(n)}$$

a = 1
b = 2
d = 1

$\mathbf{a < b^d}$

Suppose $\mathbf{T(n) = a \cdot T(n/b) + O(n^d)}$. The Master Theorem states:

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

THE IDEAL PIVOT

The IDEAL pivot: splits the input array exactly in half!

len(L ... (n-1)/2

$$T(n) = \begin{cases} O(n) \\ T(\texttt{len(L)}) + O(n) \\ T(\texttt{len(R)}) + O(n) \end{cases}$$

With the ideal pivot, the runtime would be:

O(n)

$$\mathbf{T(n) \le T(n/2) + O(n)}$$

a = 1
b = 2
d = 1

$\mathbf{a < b^d}$

Suppose **T(n) = a · T(n/b)** ... Master Theorem states:

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

THE IDEAL PIVOT

The IDEAL pivot: splits the input array exactly in half!

$T(n) = \begin{cases} O(\ldots \\ T(\ldots \\ T(\ldots \end{cases}$ **+ O(n)**

b^d

Sadly, the pivot to divide the input in half is the

MEDIAN

aka ***SELECT(A, n/2)***

aka exactly the problem we're trying to solve...

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

سوال؟

THE GOOD-ENOUGH PIVOT

The GOOD-ENOUGH pivot: splits the input array kind of in half!

3n/10 < `len(L)` < 7n/10

3n/10 < `len(R)` < 7n/10

THE GOOD-ENOUGH PIVOT

The GOOD-ENOUGH pivot: splits the input array kind of in half!

3n/10 < `len(L)` < 7n/10

3n/10 < `len(R)` < 7n/10

If we could fetch this good-enough pivot in time O(n), let's say, the recurrence looks like:

$$T(n) = \begin{cases} O(n) & \texttt{len(L) == k-1} \\ T(\texttt{len(L)}) + O(n) & \texttt{len(L) > k-1} \\ T(\texttt{len(R)}) + O(n) & \texttt{len(L) < k-1} \end{cases} \Rightarrow \quad \mathbf{T(n) \le T(7n/10) + O(n)}$$

THE GOOD-ENOUGH PIVOT

The GOOD-ENOUGH pivot: splits the input array kind of in half!

3n/10 < `len(L)` < 7n/10

3n/10 < `len(R)` < 7n/10

If we could fetch this good-enough pivot in time O(n), let's say, the recurrence looks like:

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

$$\Rightarrow$$

$$\mathbf{T(n) \le T(7n/10) + O(n)}$$

a = 1
b = 10/7
d = 1

$\mathbf{a < b^d}$

Suppose $\mathbf{T(n) = a \cdot T(n/b) + O(n^d)}$. The Master Theorem states:

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

THE GOOD-ENOUGH PIVOT

The GOOD-ENOUGH pivot: splits the input array kind of in half!

3n/10 < `len(L)` < 7n/10
3n/ ... /10

If we could fetch this good-enou ... t's say, the recurrence looks like:

$T(n) = \begin{cases} O(n) & \texttt{l} \\ T(\texttt{len(L)}) + O(n) & \texttt{l} \\ T(\texttt{len(R)}) + O(n) & \texttt{le} \end{cases}$

This good-enough pivot would still give us:

O(n)

$$\mathbf{T(n) \leq T(7n/10) + O(n)}$$

a = 1
b = 10/7
d = 1

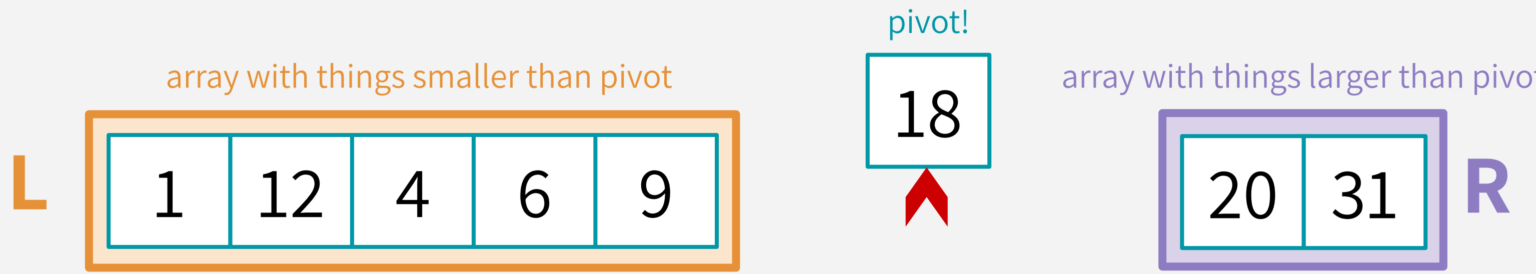
$\mathbf{a < b^d}$

Suppose **T(n) = a · T(n/b)** ... Master Theorem states:

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

OUR GOAL

Efficiently pick the pivot in time O(n) so that



3n/10 < `len(L)` < 7n/10

3n/10 < `len(R)` < 7n/10

Then, our recurrence $T(n) \leq T(7n/10) + O(n)$ comes out to **O(n)**!

سوال؟

میانه ی میانه ها!

ایده اصلی الگوریتم خطی برای انتخاب kامین عضو

MEDIAN-OF-MEDIANS

The ideal world wasn't feasible because we can't just compute SELECT(A, n/2) ⇒ that would throw us into infinite recursion since problem sizes aren't shrinking between recursive calls…

But we can instead generate a ***smaller*** list and call SELECT on that smaller list!

MEDIAN-OF-MEDIANS

The ideal world wasn't feasible because we can't just compute SELECT(A, n/2) ⇒ that would throw us into infinite recursion since problem sizes aren't shrinking between recursive calls…

But we can instead generate a ***smaller*** list and call SELECT on that smaller list!

OUR GAME PLAN:

We'll make a smaller list out of SUB-MEDIANS.

Then, we'll use SELECT to find the median of the sub-medians.

This "median of medians" will be our proxy for the true median!

MEDIAN-OF-MEDIANS

GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

1	14	4	18	25	6	17	9	3	5	10	16	12	23	19	13	20	8	15	24	7	21	22	2	11

MEDIAN-OF-MEDIANS

GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

Divide the original list into ⌈n/5⌉ groups (each group has ≤ 5 elements)

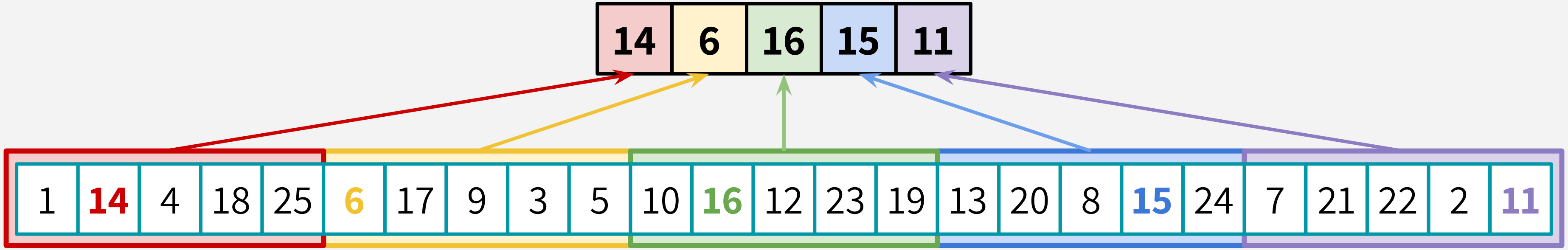
1	14	4	18	25	6	17	9	3	5	10	16	12	23	19	13	20	8	15	24	7	21	22	2	11

MEDIAN-OF-MEDIANS

GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

Divide the original list into ⌈n/5⌉ groups (each group has ≤ 5 elements)

Find the sub-median of each small group (3rd smallest out of the 5)



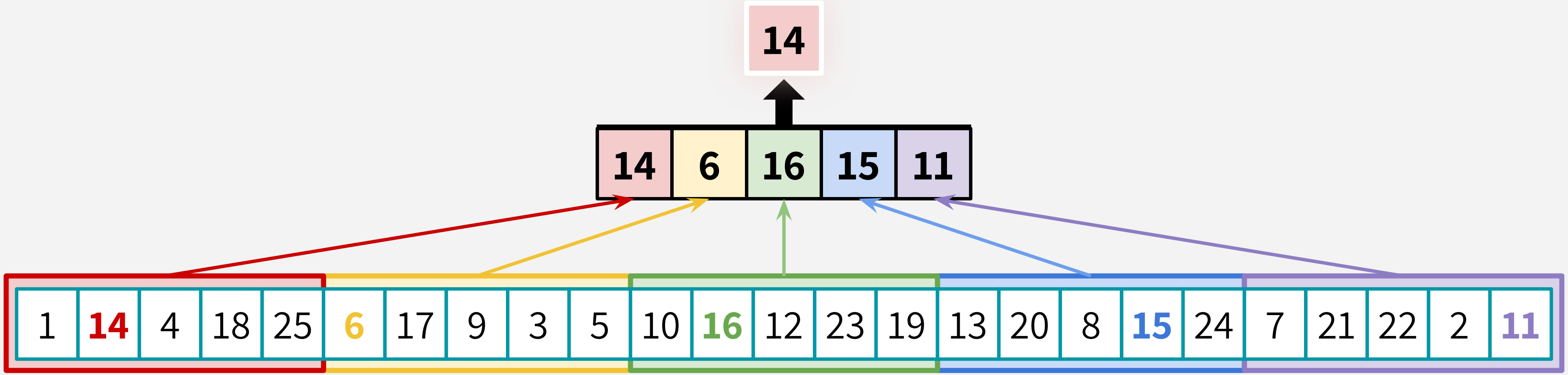
MEDIAN-OF-MEDIANS

GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

Divide the original list into ⌈n/5⌉ groups (each group has ≤ 5 elements)

Find the sub-median of each small group (3rd smallest out of the 5)

Find the median of all the sub-medians (call SELECT)



MEDIAN-OF-MEDIANS

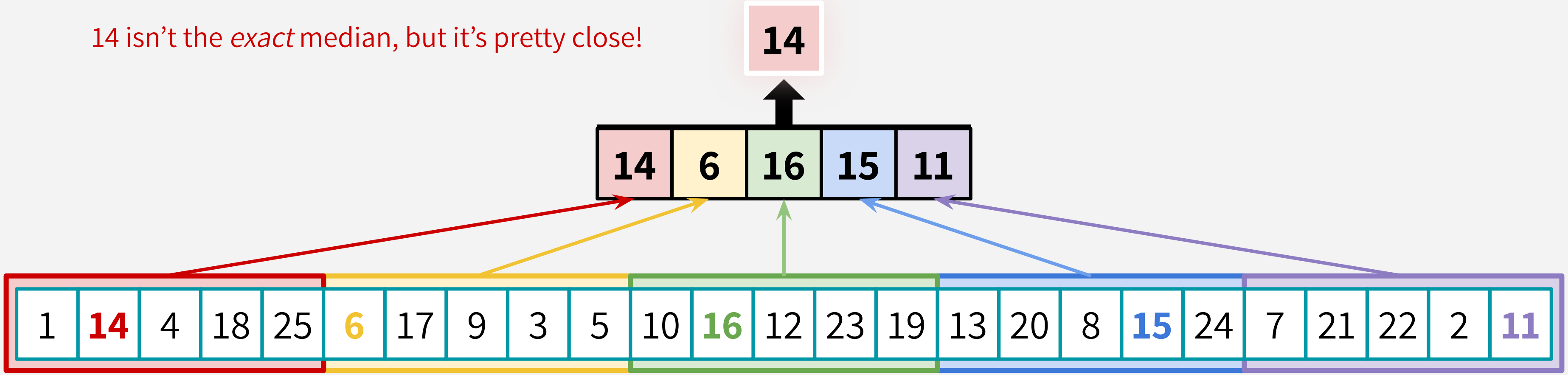
GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

Divide the original list into ⌈n/5⌉ groups (each group has ≤ 5 elements)

Find the sub-median of each small group (3rd smallest out of the 5)

Find the median of all the sub-medians (call SELECT)

14 isn't the *exact* median, but it's pretty close!



MEDIAN-OF-MEDIANS

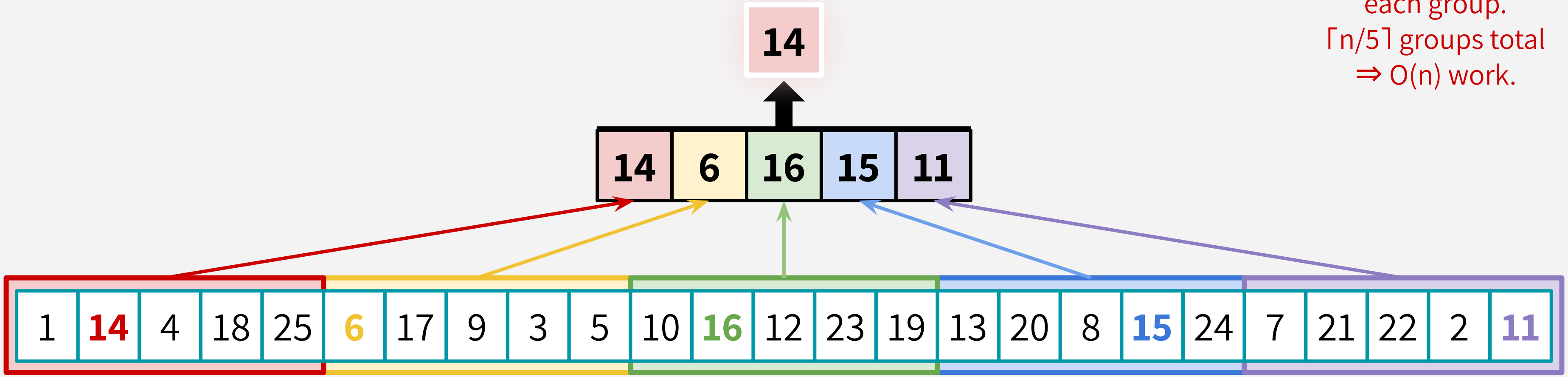
GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

Divide the original list into ⌈n/5⌉ groups (each group has ≤ 5 elements)

Find the sub-median of each small group (3rd smallest out of the 5)

Find the median of all the sub-medians (call SELECT)

constant work for each group. ⌈n/5⌉ groups total ⇒ O(n) work.



MEDIAN-OF-MEDIANS

GOAL: get a proxy for the true median by finding the exact median of all the sub-medians!

Divide the original list into ⌈n/5⌉ groups (each group has ≤ 5 elements)

Find the sub-median of each small group (3rd smallest out of the 5)

Find the median of all the sub-medians (call SELECT)

constant work for each group. ⌈n/5⌉ groups total ork.

14

To compute our pivot:

Do O(n) work to set up (divide into groups & get a list of submedians), then make a call to **SELECT**(Submedians, |Submedians|/2)

1	**14**	4	18	25	**6**	17	9	3	5	10	**16**	12	23	19	13	20	8	**15**	24	7	21	22	2	**11**

سوال؟

ANALYZING RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = MEDIAN_OF_MEDIANS(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else:
        return SELECT(R, k-len(L)-1)
```

What does the recurrence relation for T(n) look like?

ANALYZING RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = MEDIAN_OF_MEDIANS(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else:
        return SELECT(R, k-len(L)-1)
```

O(n) work outside of recursive calls

(base case, set-up within MEDIAN_OF_MEDIANS, partitioning)

T(n/5) work hidden in this recursive call

(remember, MEDIAN_OF_MEDIANS calls SELECT on ⌈n/5⌉-size array)

T(???) work hidden in this recursive call

What is the maximum size of either L or R?

ANALYZING RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
```

What is the smallest number of elements that could be smaller than our MEDIAN OF MEDIANS?

```
    else:
        return SELECT(R, k-len(L)-1)
```

O(n) work outside of recursive calls

(base case, set-up within MEDIAN_OF_MEDIANS, partitioning)

T(n/5) work hidden in this recursive call

(remember, MEDIAN_OF_MEDIANS calls SELECT on ⌈n/5⌉-size array)

T(???) work hidden in this recursive call

What is the maximum size of either L or R?

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)

$m = \lceil n/5 \rceil$ groups

1	14	4	18	25
6	17	9	3	5
10	16	12	23	19
13	20	8	15	24
7	21	22	2	11

at most 5 elements

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)

$m = \lceil n/5 \rceil$ groups

1	**14**	4	18	25
6	17	9	3	5
10	16	12	23	19
13	20	8	15	24
7	21	22	2	11

at most 5 elements

At least how many elements are guaranteed to be **smaller** than the median of medians?

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)

$m = \lceil n/5 \rceil$ groups

3	5	**6**	9	17
2	7	**11**	21	22
1	4	**14**	18	25
8	13	**15**	20	24
10	12	**16**	19	23

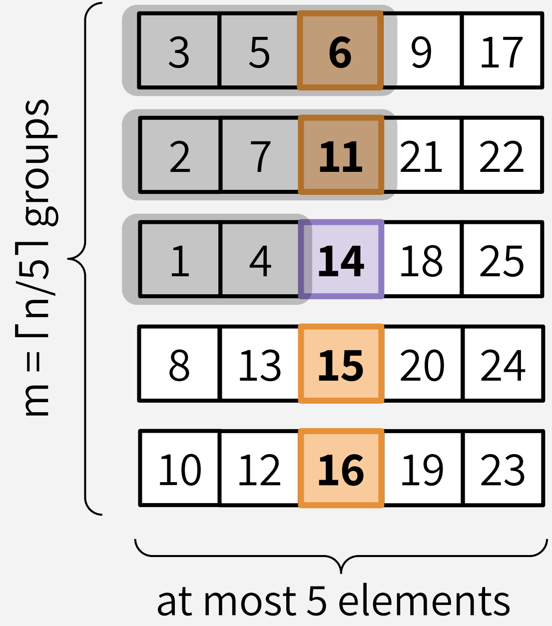
at most 5 elements

At least how many elements are guaranteed to be **smaller** than the median of medians?

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)



At least how many elements are guaranteed to be **smaller** than the median of medians?

3 elements from each group that has a **median** smaller than the **median of medians** + 2 elements from the group containing the **median of medians**

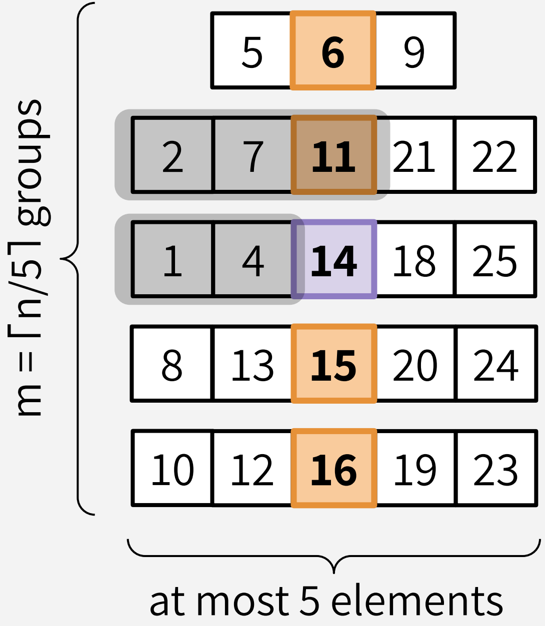
$$3 \cdot (\lceil m/2 \rceil - 1) + 2$$

To exclude the group with the **median of medians**

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)



At least how many elements are guaranteed to be **smaller** than the median of medians?

3 elements from each (non-leftover) group that has a **median** smaller than the **median of medians** + 2 elements from the group containing the **median of medians**

$$3 \cdot (\lceil m/2 \rceil - 1 - 1) + 2$$

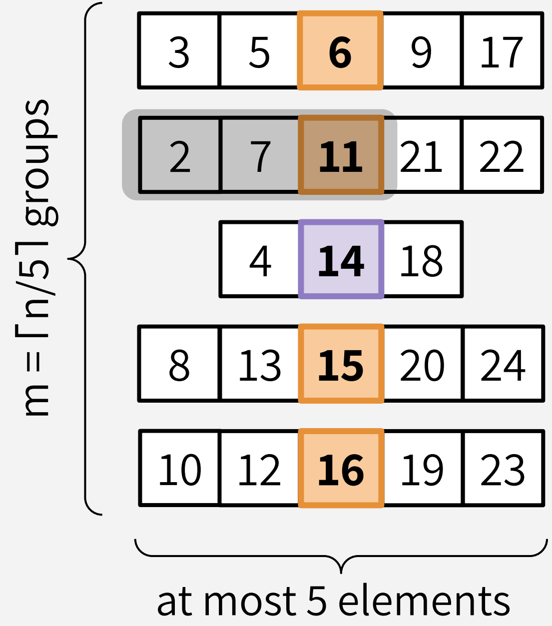
To exclude the group with the **median of medians**

To exclude any of those groups that might be a "leftover" group!

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)



At least how many elements are guaranteed to be **smaller** than the median of medians?

3 elements from each (non-leftover) group that has a **median** smaller than the **median of medians** + ~~2 elements from the group containing the **median of medians**~~

$$3 \cdot (\lceil m/2 \rceil - 1 - 1) \cancel{+ 2}$$

To exclude the group with the **median of medians**

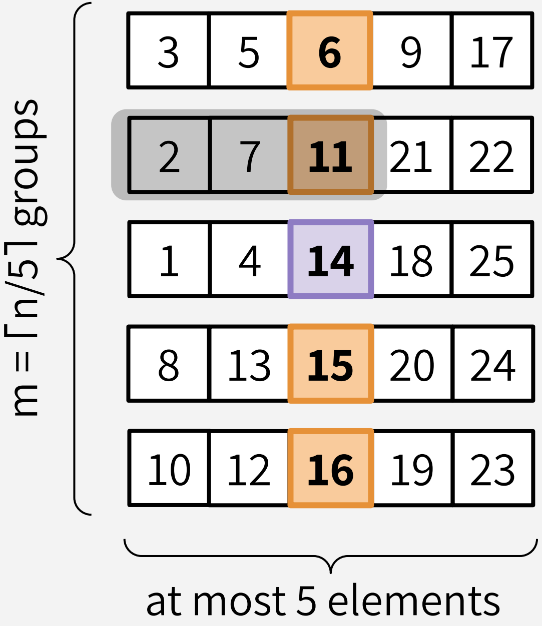
To exclude any of those groups that might be a "leftover" group!

The group with the **median of medians** might be a "leftover" group! Might as well just get rid of the +2 to be safe

ANALYZING RUNTIME

MEDIAN_OF_MEDIANS will choose a pivot greater than at least 3n/10 - 6 elements

(The same reasoning we're about to do also shows that the pivot will be less than at least 3n/10 - 6 elements)



At least how many elements are guaranteed to be **smaller** than the median of medians?

3 elements from each (non-leftover) group that has a **median** smaller than the **median of medians**

$$
\begin{aligned}
& 3 \cdot (\lceil m/2 \rceil - 2) \\
= \; & 3 \cdot (\lceil \lceil n/5 \rceil / 2 \rceil - 2) \\
\geq \; & 3 \cdot (n/10 - 2) \\
= \; & 3n/10 - 6
\end{aligned}
$$

ANALYZING RUNTIME

We just showed:

$$3n/10 - 6 \leq \texttt{len(L)}$$

$$\texttt{len(R)} \leq 7n/10 + 5$$

ANALYZING RUNTIME

We can similarly show the inverse:

3n/10 - 6 ≤ `len(L)` ≤ 7n/10 + 5

3n/10 - 6 ≤ `len(R)` ≤ 7n/10 + 5

ANALYZING RUNTIME

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = MEDIAN_OF_MEDIANS(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else:
        return SELECT(R, k-len(L)-1)
```

O(n) work outside of recursive calls

(base case, set-up within MEDIAN_OF_MEDIANS, partitioning)

T(n/5) work hidden in this recursive call

(remember, MEDIAN_OF_MEDIANS calls SELECT on ⌈n/5⌉-size array)

T(???) work hidden in this recursive call

What is the maximum size of either L or R?

ANALYZING RUNTIME

We can similarly show the inverse:

$$3n/10 - 6 \leq \texttt{len(L)} \leq 7n/10 + 5$$

$$3n/10 - 6 \leq \texttt{len(R)} \leq 7n/10 + 5$$

What does the recurrence relation for T(n) look like?

$$T(n) \leq T(n/5) + T(???) + O(n)$$

ANALYZING RUNTIME

We can similarly show the inverse:

$$3n/10 - 6 \le \texttt{len(L)} \le 7n/10 + 5$$

$$3n/10 - 6 \le \texttt{len(R)} \le 7n/10 + 5$$

What does the recurrence relation for T(n) look like?

$$T(n) \le T(n/5) + T(7n/10) + O(n)$$

ANALYZING RUNTIME

$$T(n) \le T(n/5) + T(7n/10) + O(n)$$

We solved this recurrence using the Substitution Method at the start of class!

SUBSTITUTION METHOD: EXAMPLE 2

$T(n) = T(n/5) + T(7n/10) + n$

$T(n) = 1$ when $1 \le n \le 10$

Our guess from Step 1:

T(n) is O(n)

STEP 2: Prove it!

We can choose C = 10!

- **Inductive Hypothesis**: $T(n) \le \mathbf{10}n$
- **Base case**: Prove IH holds for $1 \le n \le 10$. $T(n) = 1 \le \mathbf{10}n$
- **Inductive step**:
  - Let $k > 10$. Assume that the IH holds for all n such that $1 \le n < k$.
  - $$\begin{aligned} T(k) &= k + T(k/5) + T(7k/10) \\ &\le k + \mathbf{10} \cdot (k/5) + \mathbf{10} \cdot (7k/10) \\ &= k + 2k + 7k \\ &= \mathbf{10}k \end{aligned}$$
  - Thus, the IH holds for $n = k$!
- **Conclusion:** With $C = 10$ and $n_0 = 1$, $T(n) \le Cn$ for all $n \ge n_0$. By the Big-O definition, $T(n) = O(n)$.

ANALYZING RUNTIME

$$T(n) \leq T(n/5) + T(7n/10) + O(n)$$

We solved this recurrence using the Substitution Method at the start of class!

⬇

O(n)

Worst-case Runtime!

LINEAR-TIME SELECTION

```
SELECT(A,k):
    if len(A) == 1:
        return A[0]
    p = MEDIAN_OF_MEDIANS(A)
    L, R = PARTITION(A,p)
    if len(L) == k-1:
        return p
    else if len(L) > k-1:
        return SELECT(L, k)
    else if len(L) < k-1:
        return SELECT(R, k-len(L)-1)
```

O(n)

Worst-case Runtime!

سوال؟