Lecture 5: More Recursion

Tim LaRock

larock.t@northeastern.edu

bit.ly/cs3000syllabus

Business

- Homework 1 due tonight!
  - Only turn in compiled PDF, no need for .tex file
  - Make sure you turn something in!
    - It is okay, even expected, if you aren’t totally sure about some solutions. Do your best.
    - Remember that 1 homework grade is dropped
- Homework 2 will be released tomorrow and due next Monday at midnight Boston time
- Office hours: Please email ahead of time with topic!
- Reminder: Use Piazza for questions as much as possible
  - You can ask private questions to the instructors. This is preferable to email.

Business 2: Exams

Tentative Schedule for Exams:

Midterm 1: Release next **Weds 5/20 8pm** and due **Friday 5/22 8pm**

Midterm 2 (tentative): Same deal starting Wed June 4

Final Exam TBD (probably either June 17-19 or during finals week)

Today

More recursion examples

- Selection without sorting
- Binary Search
- Master Theorem for solving recurrence relations

Finding the median without sorting

We motivated sorting with the median problem

```
Input: L, an array of N numbers
Output: The median of L
Procedure:
    1. Sort L
    2. If N is odd, return the number at L[⌈N/2⌉]
    3. If N is even, return the mean of the
       numbers at L[⌈N/2⌉] and L[⌈N/2⌉+1]
```

Can we compute the median without sorting the whole list first?

Selection without sorting

More general goal: Given unsorted array of integers `A`, how long to find the:

- Smallest number?
- Second smallest number?
- k^{th} smallest number?
- Median?

`A`	11	3	5	6	8	2

Selection without sorting

More general goal: Given unsorted array of integers `A`, how long to find the:

- Smallest number?
- Second smallest number?
- k^{th} smallest number?
- Median?

`A`					
11	3	5	6	8	2

Idea: What if we break the input array into subarrays in a "smart" way so that only 1 subarray needs to be searched recursively?

Today: Smart recursion for an $O(n)$ selection algorithm.

QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a "smart" way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

→ Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]

A	11	3	5	4	8	2

QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]

A	11	3	**5**	4	8	2

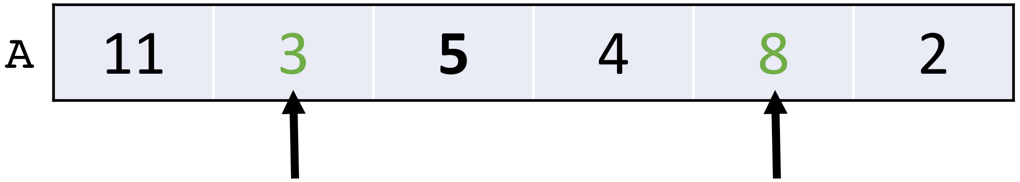
Pivot Element

QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]



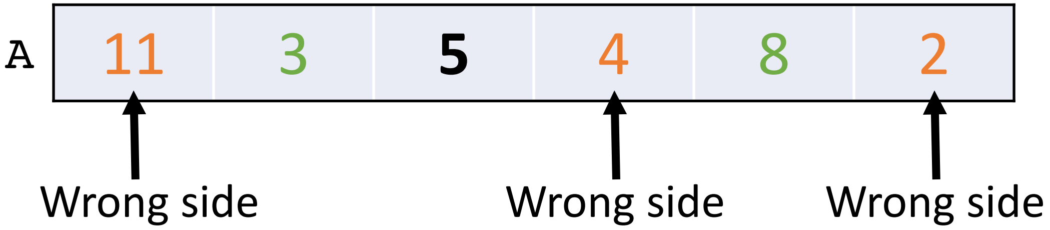
Already on the correct side of A[p]

QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]

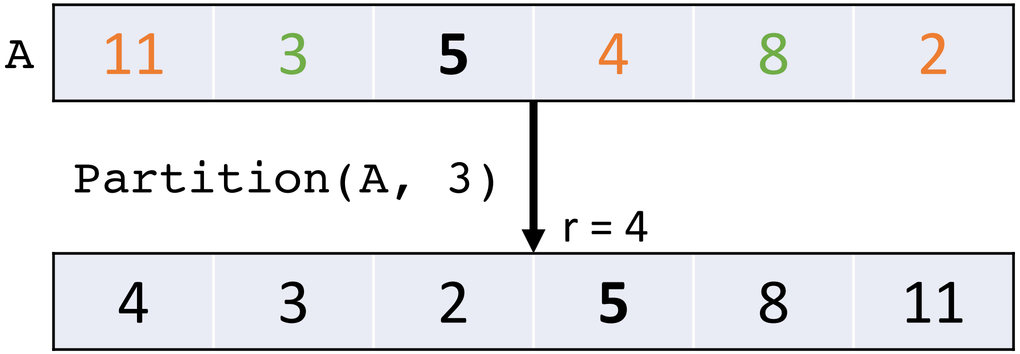


QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]

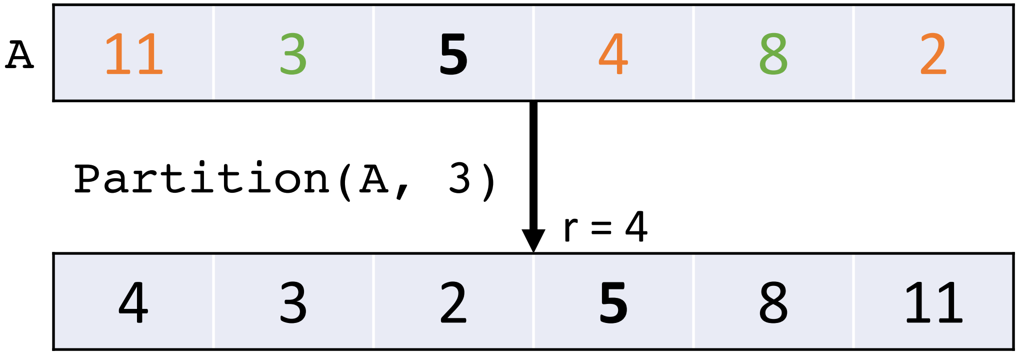


QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]



Note: Partitioning does **not** sort the array!

QuickSelect: Selection without sorting

Idea: Break the input array into subarrays in a “smart” way so that only 1 subarray needs to be searched recursively

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Key Observation: If I want the 3rd smallest value in this example (4) this partitioning scheme guarantees it is in the left subarray!

Given A and p, return the array transformed so that all elements in the left half are less than A[p], the middle value is A[p] , and all the elements in the right half are greater than A[p]

A					
11	3	**5**	4	8	2

`Partition(A, 3)`

r = 4

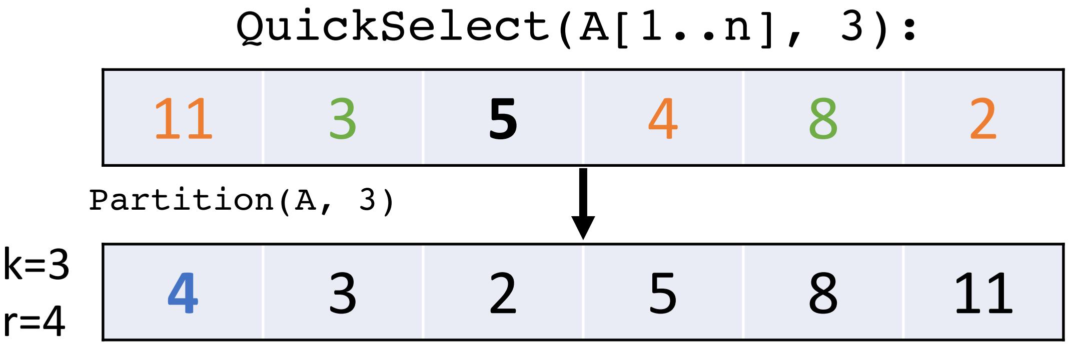
4	3	2	**5**	8	11

Note: Partitioning does **not** sort the array!

QuickSelect Example

Assume `Partition(A, p)` is correct and I will "randomly" choose pivots

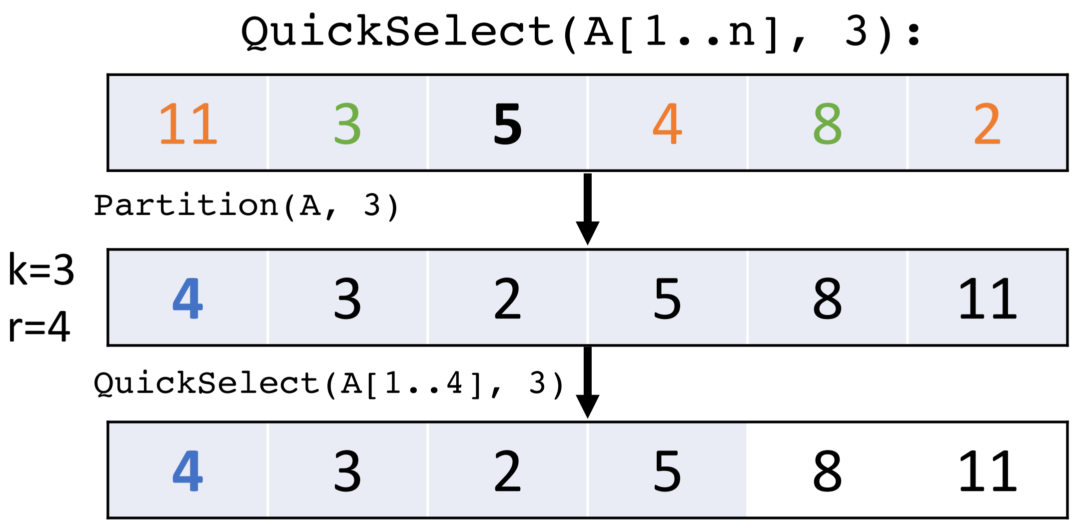
```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```



QuickSelect Example

Assume `Partition(A, p)` is correct and I will "randomly" choose pivots

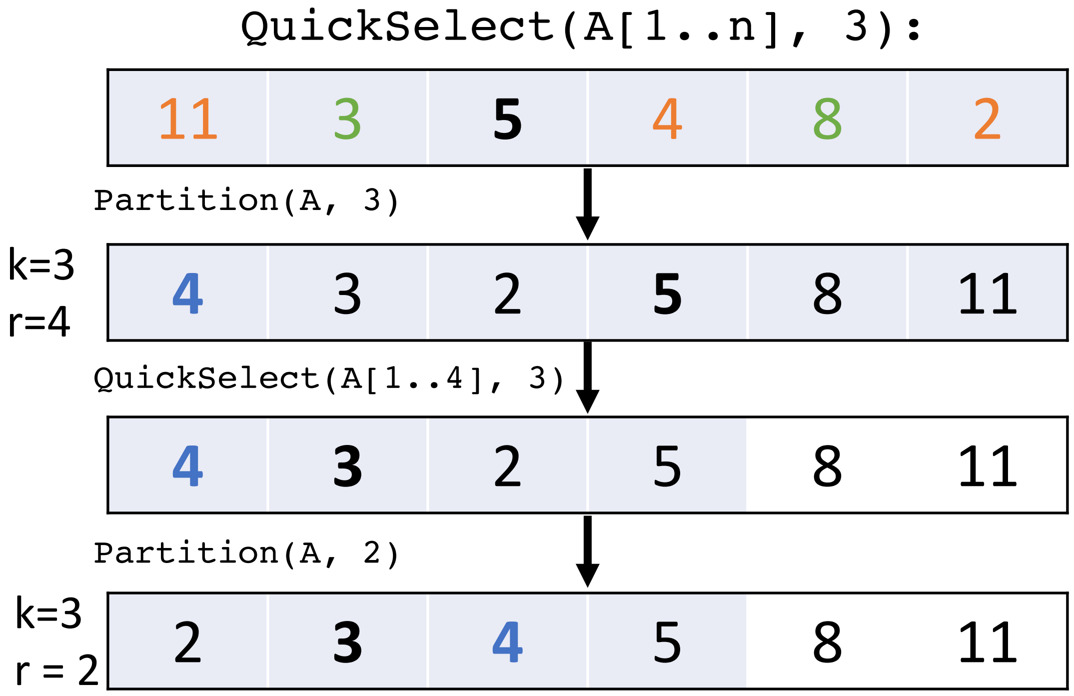
```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```



QuickSelect Example

Assume `Partition(A, p)` is correct and I will "randomly" choose pivots

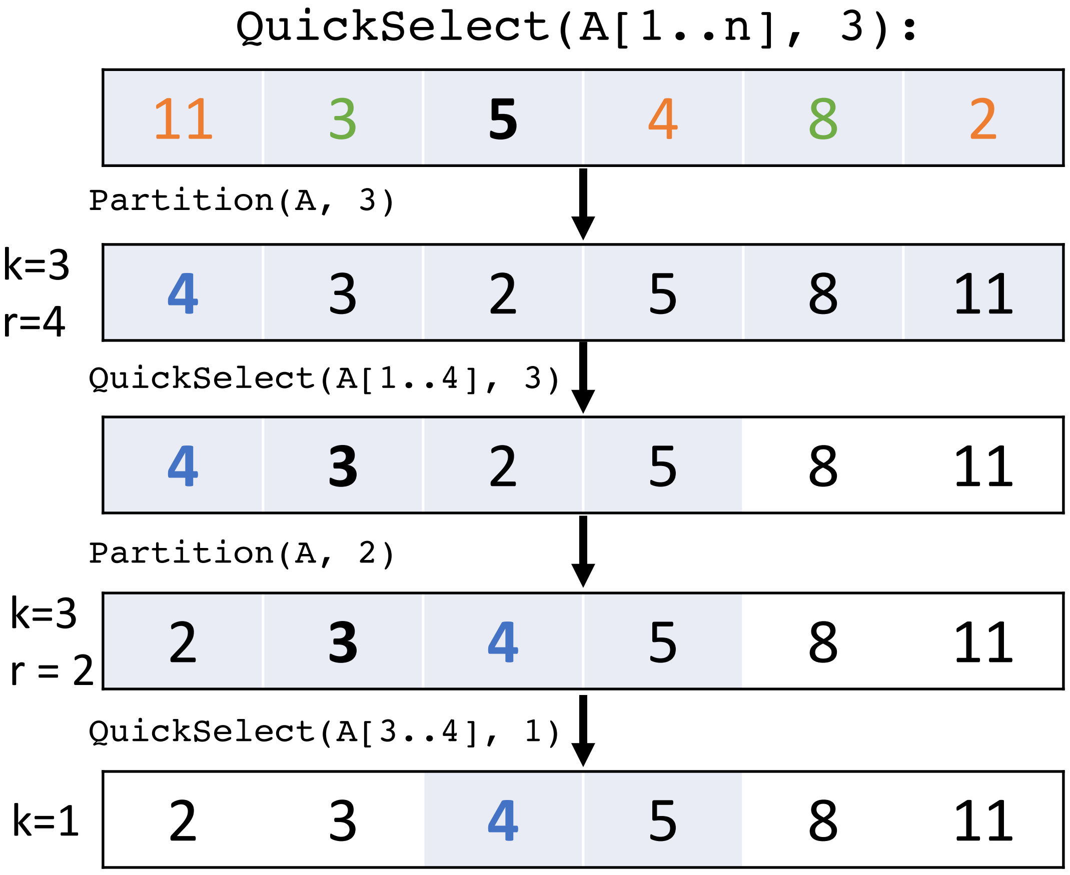
```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```



QuickSelect Example

Assume `Partition(A, p)` is correct and I will "randomly" choose pivots

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

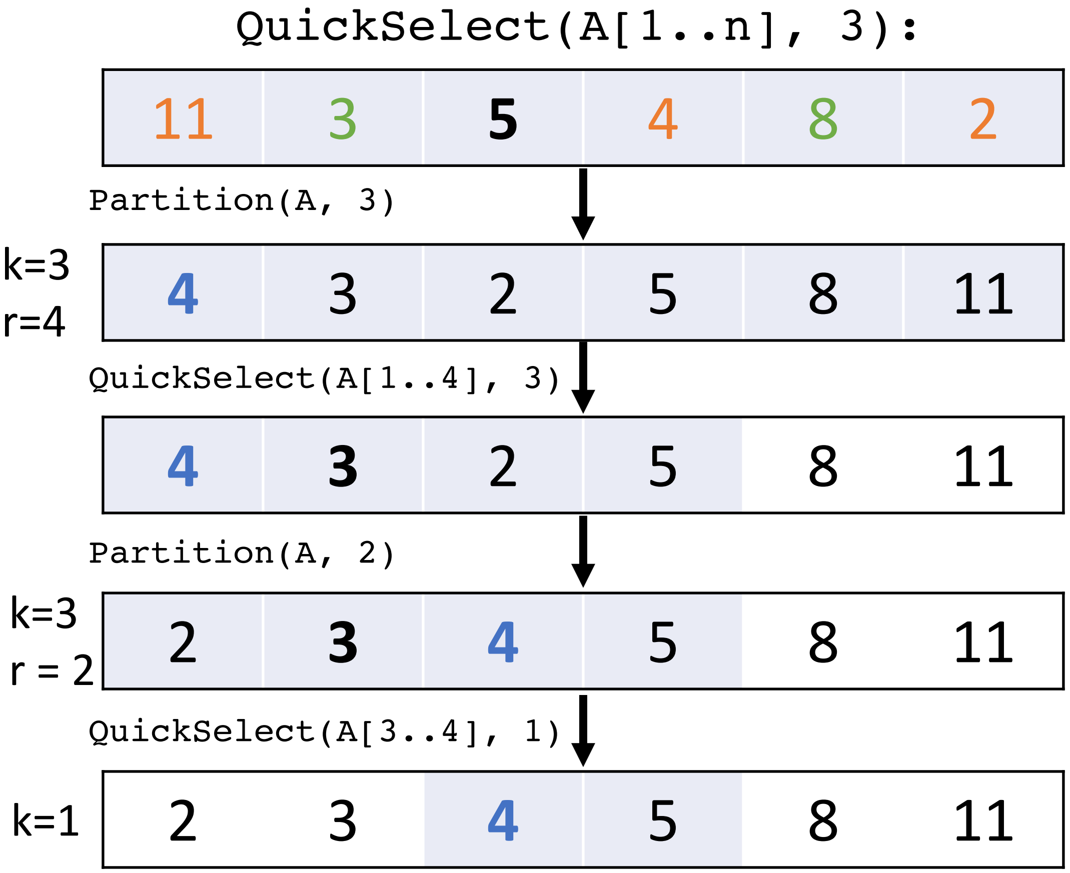


Two possibilities:

QuickSelect Example

Assume `Partition(A, p)` is correct and I will "randomly" choose pivots

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```



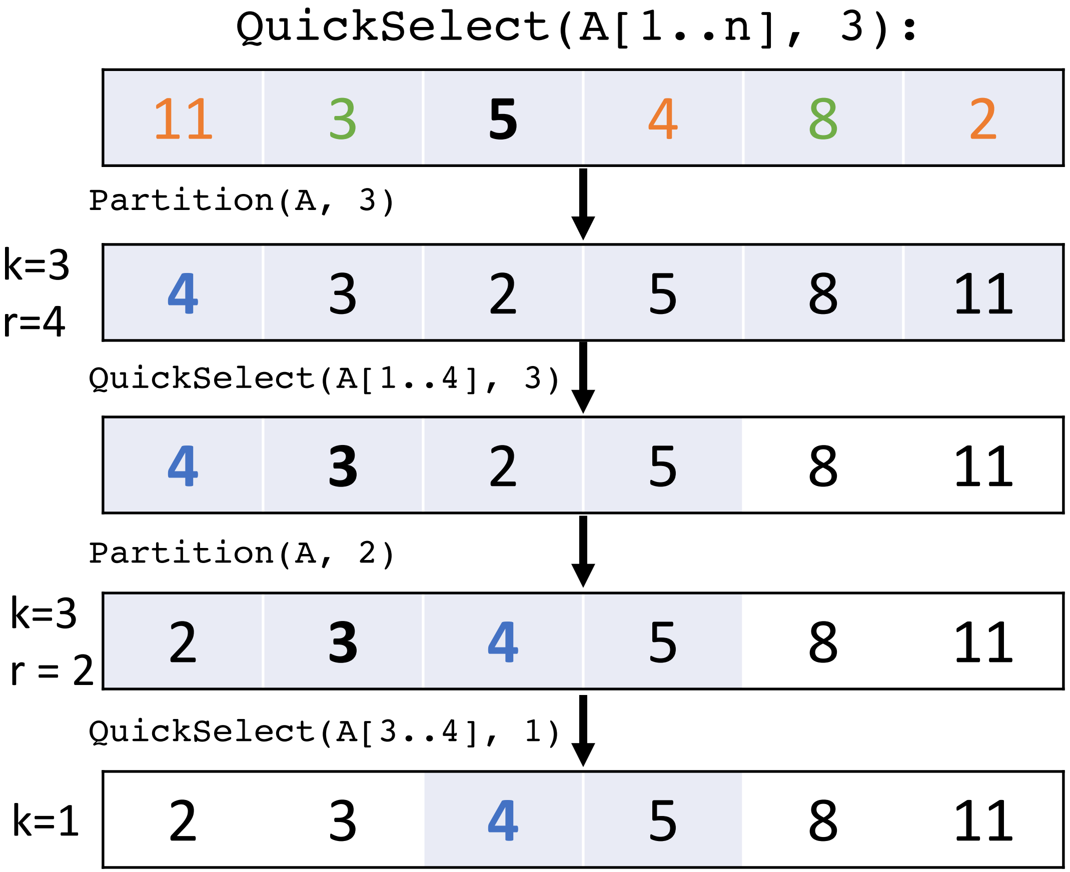
Two possibilities:

1. We pivot on 4 (r=1), in which case r=k and we return A[1] = 4

QuickSelect Example

Assume `Partition(A, p)` is correct and I will "randomly" choose pivots

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```



Two possibilities:

1. We pivot on 4 (r=1), in which case r=k and we return A[1] = 4
2. We pivot on 5 (r=2), in which case we recurse on just 4, meaning n=1 and we return 4

QuickSelect: Choosing pivot elements

Problem: How do we choose a “good” pivot element?

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

- What happens if you choose the minimum value as the pivot? Or maximum value?
- Without assuming anything about the input array, it is difficult to pick a good pivot *a priori*!
- What is our goal for a “good pivot”?

QuickSelect: Choosing pivot elements

Problem: How do we choose a “good” pivot element?

```
QuickSelect(A[1..n], k):
  If n = 1:
    return A[1]
  Else:
    Choose a pivot element A[p]
    r ← Partition(A, p)

    If k < r:
      Return QuickSelect(A[1..r], k)
    ElseIf k > r:
      Return QuickSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

- What happens if you choose the minimum value as the pivot? Or maximum value?
- Without assuming anything about the input array, it is difficult to pick a good pivot *a priori*!
- What is our goal for a “good pivot”?
  - Close to the median!

Median of Medians

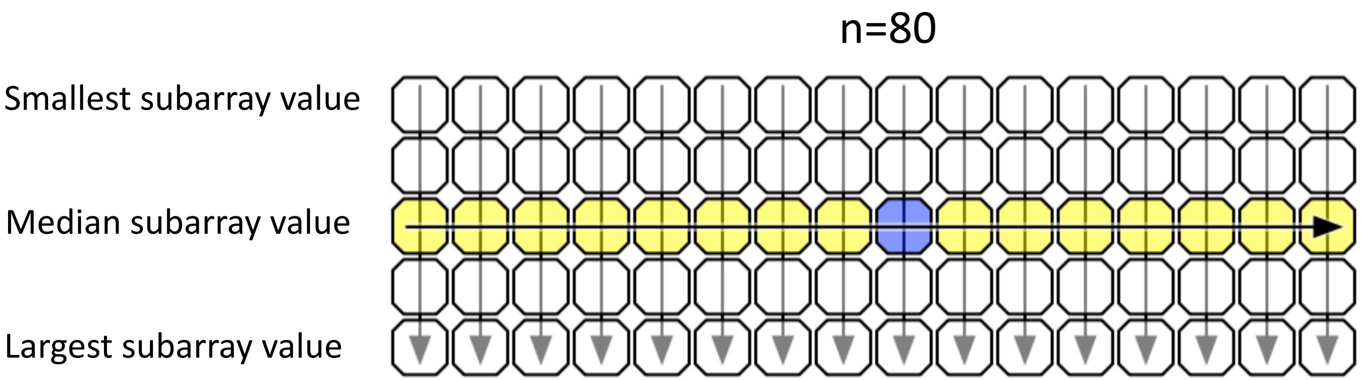
Idea: Choose a pivot element by approximating the median.

```
MOM(A[1..n]):
  Let m ← ⌈n/5⌉
  For i in 1,..,m:
    Medians[i] = Median(A[5i-4..5i])
  med ← MOMSelect(Medians[i..m], ⌊m/2⌋)
  return index of med in A
```

Break the input up into $\left\lceil \frac{n}{5} \right\rceil$ subarrays, take the median of each, then find the median of those medians (MoM).

Median of Medians

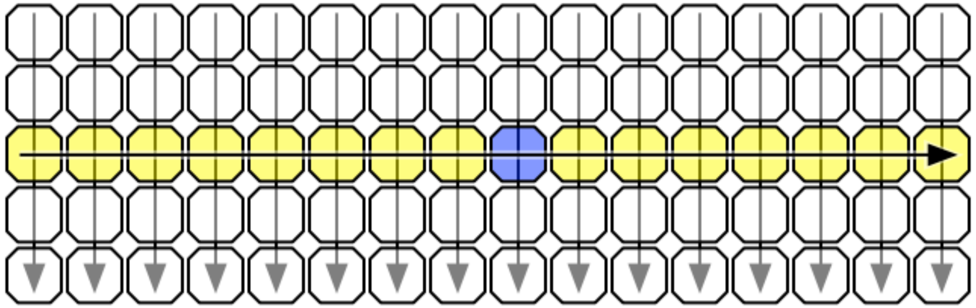
- **Claim:** For every $\boldsymbol{A}$ there are at least $3n/10$ items that are smaller than $\mathbf{MOM}(\boldsymbol{A})$ and at least $3n/10$ items that are larger.



Visualizing the median of medians

Median of Medians

- **Claim:** For every $\boldsymbol{A}$ there are at least $3n/10$ items that are smaller than $\mathbf{MOM}(\boldsymbol{A})$ and at least $3n/10$ items that are larger.

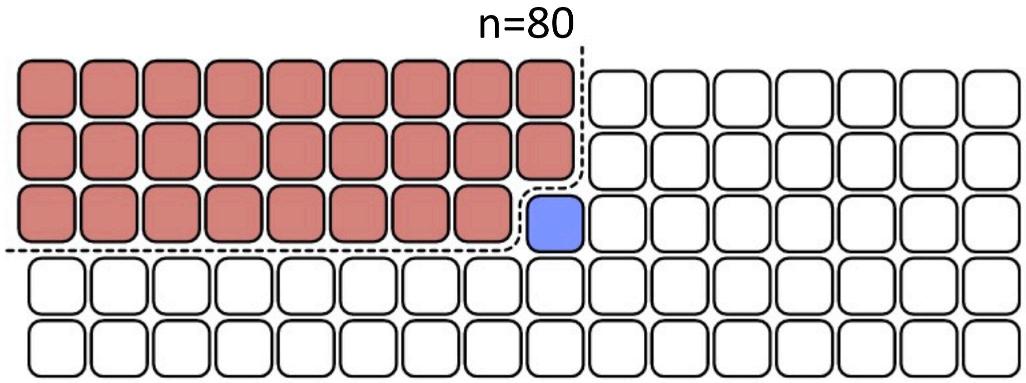


Visualizing the median of medians

Median of Medians

- **Claim:** For every $\boldsymbol{A}$ there are at least $3n/10$ items that are smaller than $\mathbf{MOM}(\boldsymbol{A})$ and at least $3n/10$ items that are larger.

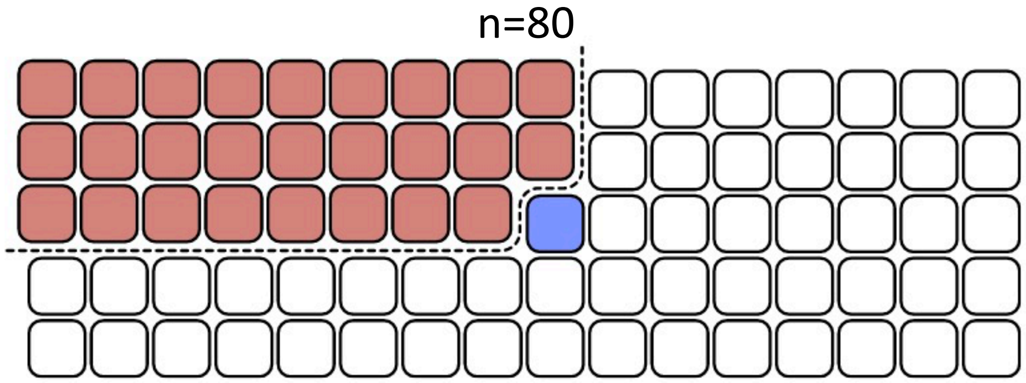
$\frac{3n}{10} = 26$



Median of Medians

- **Claim:** For every $\boldsymbol{A}$ there are at least $3n/10$ items that are smaller than $\mathbf{MOM}(\boldsymbol{A})$ and at least $3n/10$ items that are larger.

$\frac{3n}{10} = 26$



- If *k* is smaller than $\frac{3n}{10}$, recurse on those items
- If *k* is larger than $\frac{3n}{10}$, recurse on the remaining

$$n - \frac{3n}{10} = \frac{7n}{10} \text{ items}$$

MOMSelect

```
MOMSelect(A[1..n], k):
  If n <= 25:
    return median(A)
  Else:
    mom ← MOM(A[1..n])
    r ← Partition(A, mom)

    If k < r:
      Return MOMSelect(A[1..r], k)
    ElseIf k > r:
      Return MOMSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

MOMSelect Running Time

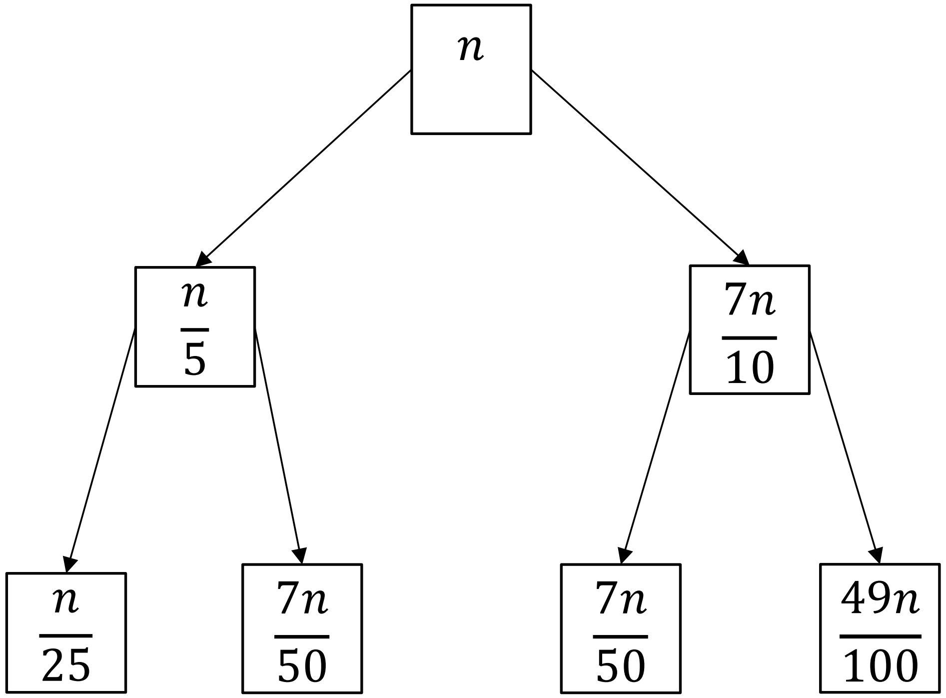
```
MOMSelect(A[1..n], k):
  If n <= 25:
    return median(A)
  Else:
    mom ← MOM(A[1..n])
    r ← Partition(A, mom)

    If k < r:
      Return MOMSelect(A[1..r], k)
    ElseIf k > r:
      Return MOMSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

What is a recurrence relation for MOMSelect?

T(n) = T(Selection) + T(MOM) + f(ops per step)

Recursion Tree

$$T(n) = T(\frac{7n}{10}) + T(\frac{n}{5}) + O(n)$$



Since the work at each level is decreasing exponentially, the O(n) term dominates!

Proof by induction

T(n) = T($\frac{7n}{10}$) + T($\frac{n}{5}$) + O(n)
T(1) = 1

We want to show that

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

Proof by induction

$T(n) = T(\frac{7n}{10}) + T(\frac{n}{5}) + O(n)$
$T(1) = 1$

We want to show that

$$T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + O(n) \le O(n), \text{ meaning}$$

$$T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + n \le Cn \text{ (for some } C\text{)}$$

Proof by induction

T(n) = T($\frac{7n}{10}$) + T($\frac{n}{5}$) + O(n)
T(1) = 1

We want to show that

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

$$T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + n \leq Cn \text{ (for some } C\text{)}$$

By induction, since $\frac{1}{5}n < \frac{7}{10}n < n$, we have

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

Proof by induction

T(n) = T($\frac{7n}{10}$) + T($\frac{n}{5}$) + O(n)
T(1) = 1

We want to show that

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

$$T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + n \leq Cn \text{ (for some } C\text{)}$$

By induction, since $\frac{1}{5}n < \frac{7}{10}n < n$, we have

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

Pulling out n, we get

$$n\left(C\frac{7}{10} + C\frac{1}{5} + 1\right)$$

$$n\left(C\frac{9}{10} + 1\right)$$

$$\leq Cn$$

Proof by induction

T(n) = T($\frac{7n}{10}$) + T($\frac{n}{5}$) + O(n)
T(1) = 1

We want to show that

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

$$T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + n \leq Cn \text{ (for some } C\text{)}$$

By induction, since $\frac{1}{5}n < \frac{7}{10}n < n$, we have

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

Pulling out n, we get

$$n\left(C\frac{7}{10} + C\frac{1}{5} + 1\right)$$

$$n\left(C\frac{9}{10} + 1\right)$$

$$\leq Cn$$

For which values of C?

$$C\frac{9}{10} + 1 \leq C$$

$$9C + 10 \leq 10C$$

$$C \geq 10$$

Proof by induction

T(n) = T($\frac{7n}{10}$) + T($\frac{n}{5}$) + O(n)
T(1) = 1

We want to show that

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

$$T\left(\frac{7n}{10}\right) + T\left(\frac{n}{5}\right) + n \leq Cn \text{ (for some } C\text{)}$$

By induction, since $\frac{1}{5}n < \frac{7}{10}n < n$, we have

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

Pulling out n, we get

$$n\left(C\frac{7}{10} + C\frac{1}{5} + 1\right)$$

$$n\left(C\frac{9}{10} + 1\right)$$

$$\leq Cn \quad \text{(as long as } C \geq 10\text{)}$$

For which values of C?

$$C\frac{9}{10} + 1 \leq C$$

$$9C + 10 \leq 10C$$

$$C \geq 10$$

MOMSelect Wrap

- We can find the median of a list of numbers in $O(n)$ time (faster than sorting) using divide and conquer approach
- Key: Selecting a good pivot with median-of-medians-of-five
- This technique also works for sorting (QuickSort) in $O(nlogn)$

```
MOMSelect(A[1..n], k):
  If n <= 25:
    return median(A)
  Else:
    mom ← MOM(A[1..n])
    r ← Partition(A, mom)

    If k < r:
      Return MOMSelect(A[1..r], k)
    ElseIf k > r:
      Return MOMSelect(A[r+1..n], k-r)
    Else:
      Return A[r]
```

Switching gears: Searching

Searching

Given a sorted array, what is the run time to find an element?

2	3	4	**5**	8	11

Searching

Given a sorted array, what is the run time to find an element?

2	3	4	**5**	8	11

Can we do it faster?

Binary Search

Idea: We can use the fact that the array is sorted to be smart about choosing the next subarray to search!

Binary Search

Idea: We can use the fact that the array is sorted to be smart about choosing the next subarray to search!

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m-1,t)
  Else: return Search(A,m+1,r,t)
```

Binary Search

Idea: We can use the fact that the array is sorted to be smart about choosing the next subarray to search!

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m-1,t)
  Else: return Search(A,m+1,r,t)
```

`StartSearch(A,5):`

ℓ=1
r=6
m=3

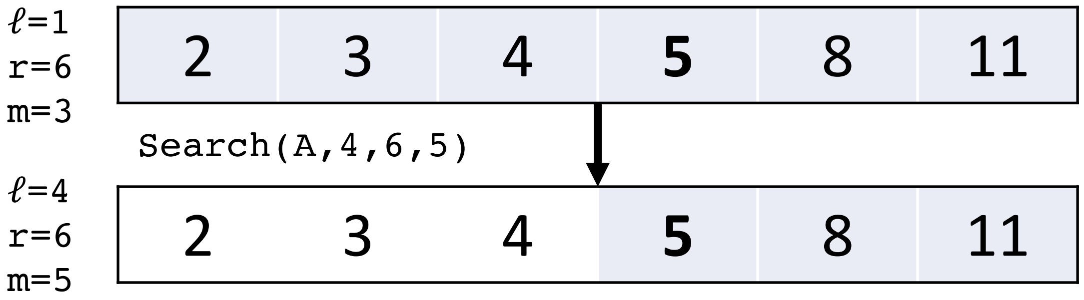
2	3	4	**5**	8	11

Binary Search

Idea: We can use the fact that the array is sorted to be smart about choosing the next subarray to search!

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m−1,t)
  Else: return Search(A,m+1,r,t)
```

StartSearch(A,5):

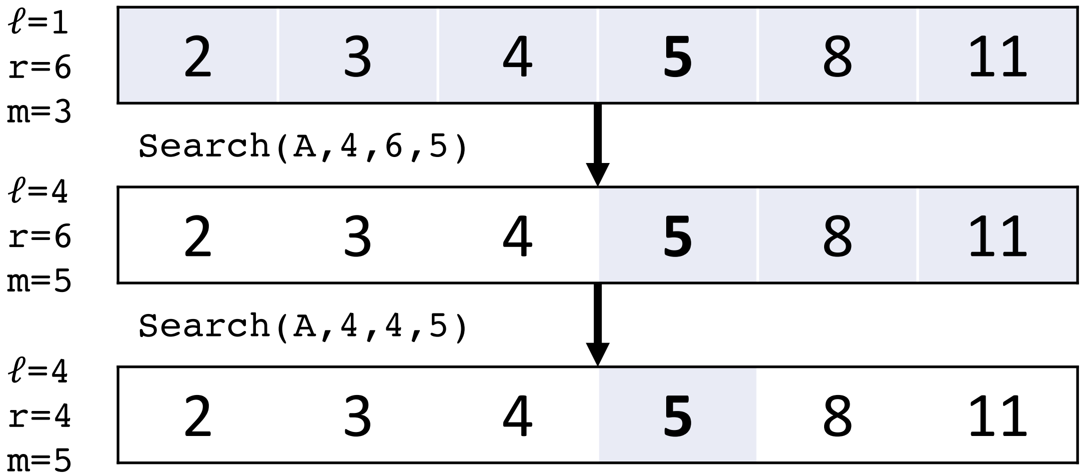


Binary Search

Idea: We can use the fact that the array is sorted to be smart about choosing the next subarray to search!

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m-1,t)
  Else: return Search(A,m+1,r,t)
```

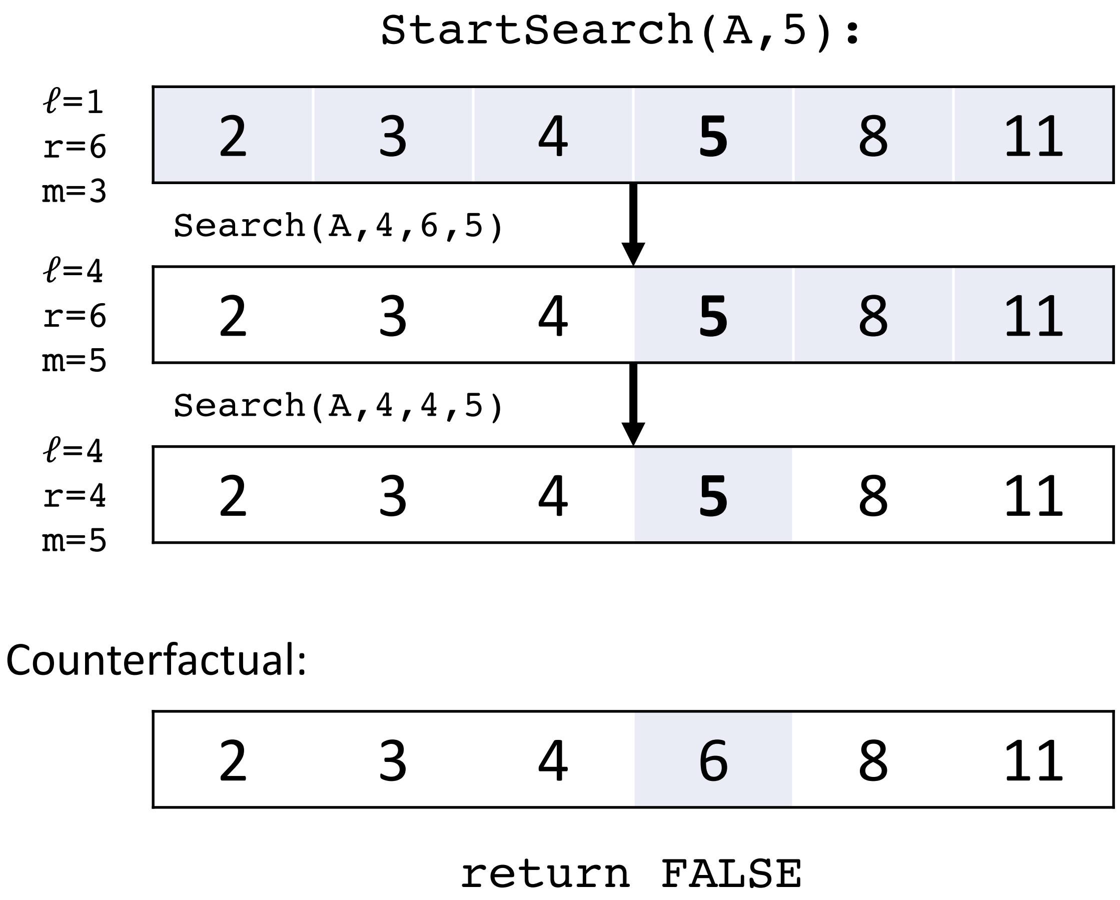
StartSearch(A,5):



Binary Search

Idea: We can use the fact that the array is sorted to be smart about choosing the next subarray to search!

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m-1,t)
  Else: return Search(A,m+1,r,t)
```



Binary Search Recurrence Relation

What does the recurrence relation look like for binary search?

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m−1,t)
  Else: return Search(A,m+1,r,t)
```

Binary Search Recurrence Relation

What does the recurrence relation look like for binary search?

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m−1,t)
  Else: return Search(A,m+1,r,t)
```

$T(n) = T(\frac{n}{2}) + O(1)$

Binary Search Recurrence Relation

What does the recurrence relation look like for binary search?

```
StartSearch(A,t):
  // A[1:n] sorted in ascending order
  Return Search(A,1,n,t)

Search(A,ℓ,r,t):
  If(ℓ > r): return FALSE

  m ← ℓ + ⌊(r−ℓ)/2⌋

  If(A[m] = t): return m
  ElseIf(A[m] > t): return Search(A,ℓ,m−1,t)
  Else: return Search(A,m+1,r,t)
```

$T(n) = T(\frac{n}{2}) + O(1)$

We could use a recursion tree to get the running time, but there is also a more general result we can use…

Master Theorem

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) > 1: \; T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) = 1: \; T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) < 1: \; T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Master Theorem

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) > 1: \; T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1: \; T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) < 1: \; T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Binary Search:

T(n) = T($\frac{n}{2}$) +O(1)

Master Theorem

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^d}\right) > 1: \ T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^d}\right) = 1: \ T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^d}\right) < 1: \ T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Binary Search:

$T(n) = T(\frac{n}{2}) + O(1)$

$T(n) = 1T(\frac{n}{2}) + n^0$

Master Theorem

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) > 1$: $T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1$: $T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) < 1$: $T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Binary Search:

$T(n) = T(\frac{n}{2}) + O(1)$

$T(n) = 1T(\frac{n}{2}) + n^0$

$$\left(\frac{1}{2^0}\right) = 1$$

Master Theorem

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) > 1: \ T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) = 1: \ T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{d}}\right) < 1: \ T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Binary Search:

$T(n) = T(\frac{n}{2}) + O(1)$

$T(n) = 1T(\frac{n}{2}) + n^0$

$$\left(\frac{1}{2^0}\right) = 1$$

So

$$T(n) = \Theta\left(n^{\mathbf{0}} \log n\right)$$

and we get

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

Master Theorem

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$
- Three cases:
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) > 1 :\ T(n) = \Theta\left(n^{\log_{\boldsymbol{b}} \boldsymbol{a}}\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) = 1 :\ T(n) = \Theta\left(n^{\boldsymbol{d}} \log n\right)$
  - $\left(\frac{\boldsymbol{a}}{\boldsymbol{b}^{\boldsymbol{d}}}\right) < 1 :\ T(n) = \Theta\left(n^{\boldsymbol{d}}\right)$

Note that the theorem does not apply to our MOMSelect recurrence:

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

Binary Search:

$\mathrm{T(n)} = \mathrm{T}(\frac{n}{2}) + \mathrm{O}(1)$

$\mathrm{T(n)} = 1\mathrm{T}(\frac{n}{2}) + \mathrm{n}^0$

$$\left(\frac{1}{2^0}\right) = 1$$

So

$$T(n) = \Theta\left(n^{\mathbf{0}} \log n\right)$$

and we get

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

Wrap up

Homework 1 due tonight

Homework 2 will be released at 8AM

Next time:

- Backtracking
- Fibonacci numbers
- Dynamic Programming

Ask the Audience!

- Use the Master Theorem to Solve:
  - $T(n) = 16 \cdot T\left(\frac{n}{4}\right) + n^2$
  - $T(n) = 21 \cdot T\left(\frac{n}{5}\right) + n^2$
  - $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 1$
  - $T(n) = 1 \cdot T\left(\frac{n}{2}\right) + 1$