



THE UNIVERSITY OF  
**MELBOURNE**

# COMP90038

# Algorithms and Complexity

Lecture 10: Decrease-and-Conquer-by-a-Factor  
(with thanks to Harald Søndergaard)

Toby Murray



[toby.murray@unimelb.edu.au](mailto:toby.murray@unimelb.edu.au)



DMD 8.17 (Level 8, Doug McDonnell Bldg)



<http://people.eng.unimelb.edu.au/tobym>



@tobycmurray

# Decrease-and-Conquer

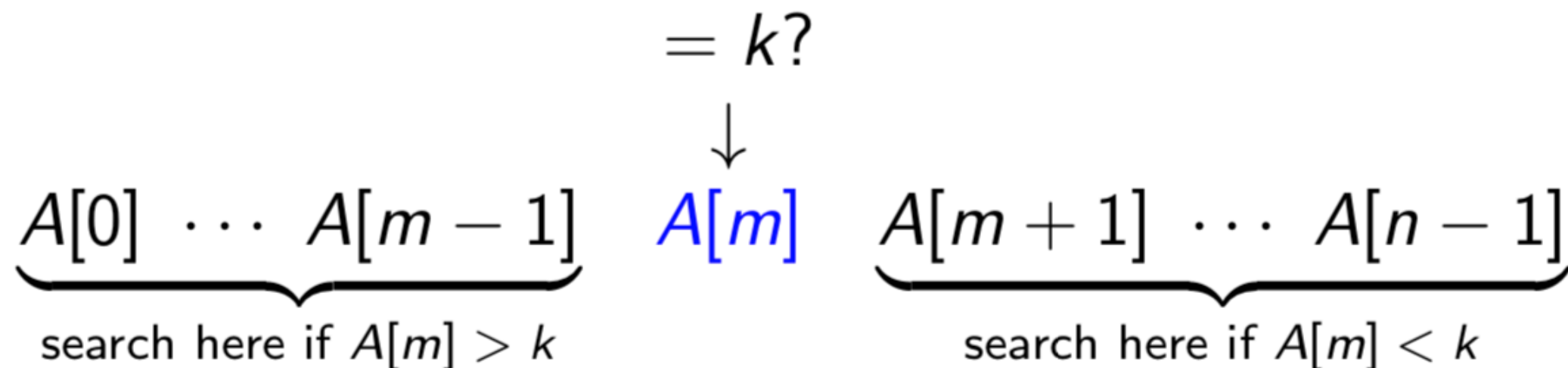
- **Last lecture:** to solve a problem of size  $n$ , try to express the solution in terms of a solution to the same problem of size  $n-1$ .
- A simple example was sorting: To sort an array of length  $n$ , just:
  1. sort the first  $n - 1$  items, then
  2. locate the cell  $A[j]$  that should hold the last item, right-shift all elements to its right, then place the last element in  $A[j]$ .
- This led to an  $O(n^2)$  algorithm called **insertion sort**. We can implement the idea either with recursion or iteration (we chose iteration).

# Decrease-and-Conquer by-a-Factor

- We now look at better utilization of the approach, often leading to methods with logarithmic time behaviour or better!
- **Decrease-by-a-constant-factor** is exemplified by binary search.
- **Decrease-by-a-variable-factor** is exemplified by interpolation search.
- Let us look at these and other instances.

# Binary Search

- This is a well-known approach for searching for an element  $k$  in a sorted array.
- Start by comparing against the array's middle element  $A[m]$ . If  $A[m] = k$  we are done.
- If  $A[m] > k$ , search the sub-array up to  $A[m - 1]$  recursively.
- If  $A[m] < k$ , search the sub-array from  $A[m + 1]$  recursively.



# Binary Search

- We have already seen a recursive formulation in Lecture 4. Here is an iterative one.

**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )



# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

$lo: 0$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

$lo: 0$

$hi: 6$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

$lo: 0$

$hi: 6$

$m: 3$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

**lo: 4**

$hi: 6$

$m: 3$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



```
function BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )
```

```
     $lo \leftarrow 0$ 
```

```
     $hi \leftarrow n - 1$ 
```

```
    while  $lo \leq hi$  do
```

```
         $m \leftarrow \lfloor (lo + hi) / 2 \rfloor$ 
```

```
        if  $A[m] = k$  then
```

```
            return  $m$ 
```

```
        if  $A[m] > k$  then
```

```
             $hi \leftarrow m - 1$ 
```

```
        else
```

```
             $lo \leftarrow m + 1$ 
```

```
    return  $-1$ 
```

$k$ : 41

**lo**: 4

$hi$ : 6

$m$ : 3

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch(A, 7, 41)

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

$lo: 4$

$hi: 6$

**$m: 5$**

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch(A,7,41)

# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

$lo: 4$

**$hi: 4$**

$m: 5$

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



```
function BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )
```

```
   $lo \leftarrow 0$ 
```

```
   $hi \leftarrow n - 1$ 
```

```
  while  $lo \leq hi$  do
```

```
     $m \leftarrow \lfloor (lo + hi) / 2 \rfloor$ 
```

```
    if  $A[m] = k$  then
```

```
      return  $m$ 
```

```
    if  $A[m] > k$  then
```

```
       $hi \leftarrow m - 1$ 
```

```
    else
```

```
       $lo \leftarrow m + 1$ 
```

```
  return  $-1$ 
```

$k$ : 41

$lo$ : 4

**$hi$ : 4**

$m$ : 5

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )



# Example:

## Binary Search in Sorted Array



**function** BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )

$lo \leftarrow 0$

$hi \leftarrow n - 1$

**while**  $lo \leq hi$  **do**

$m \leftarrow \lfloor (lo + hi) / 2 \rfloor$

**if**  $A[m] = k$  **then**

**return**  $m$

**if**  $A[m] > k$  **then**

$hi \leftarrow m - 1$

**else**

$lo \leftarrow m + 1$

**return**  $-1$

$k: 41$

$lo: 4$

$hi: 4$

**$m: 4$**

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Example:

## Binary Search in Sorted Array



```
function BINSEARCH( $A[\cdot]$ ,  $n$ ,  $k$ )
```

```
   $lo \leftarrow 0$ 
```

```
   $hi \leftarrow n - 1$ 
```

```
  while  $lo \leq hi$  do
```

```
     $m \leftarrow \lfloor (lo + hi) / 2 \rfloor$ 
```

```
    if  $A[m] = k$  then
```

```
      return  $m$ 
```

```
    if  $A[m] > k$  then
```

```
       $hi \leftarrow m - 1$ 
```

```
    else
```

```
       $lo \leftarrow m + 1$ 
```

```
  return  $-1$ 
```

$k: 41$

$lo: 4$

$hi: 4$

**$m: 4$**

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

BinSearch( $A, 7, 41$ )

# Complexity of Binary Search



THE UNIVERSITY OF  
MELBOURNE

- Worst-case input to binary search:
  - When  $k$  is not in the array
- In that case, its complexity is given by the following recursive equation:

$$C(n) = \begin{cases} 1 & \text{if } n = 1 \\ C(\lfloor n/2 \rfloor) + 1 & \text{if } n > 1 \end{cases}$$

- A closed form is:  $C(n) = \lfloor \log_2 n \rfloor + 1$
- In the worst case, searching for  $k$  in an array of size 1,000,000 requires 20 comparisons.
- The average-case time complexity is also  $\Theta(\log n)$

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

**n**

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

**n**

**m**

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

**n**

**m**



# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m
81	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m
81	92

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40		

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20		

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	



# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	
10		

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	
10	736	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	
10	736	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	
10	736	
5		

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	
10	736	
5	1472	1472

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	1472
2		



# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
81	92	92
40	184	
20	368	
10	736	
5	1472	1472
2	2944	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	1472
2	2944	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	1472
2	2944	
<b>1</b>		

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	1472
2	2944	
<b>1</b>	5888	

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	1472
2	2944	
<b>1</b>	5888	5888

# Russian Peasant Multiplication



- A way of doing multiplication.

- For even  $n$ :

$$n \cdot m = \frac{n}{2} \cdot 2m$$

- For odd  $n$ :

$$n \cdot m = \frac{n-1}{2} \cdot 2m + m$$

- Thus, ~halve  $n$  repeatedly, until  $n = 1$ . Add up all odd values of  $m$

n	m	
<b>81</b>	92	92
40	184	
20	368	
10	736	
<b>5</b>	1472	1472
2	2944	
<b>1</b>	5888	5888
		<hr/>
		= 7452

# Finding the Median

- Given an array, an important problem is how to find the **median**, that is, an array value which is no larger than half the elements and no smaller than half.

A:

9	23	3	41	22	8	46
0	1	2	3	4	5	6

- More generally, we would like to solve the problem of finding the kth smallest element. (e.g. when  $k=3$ )

A:

9	23	3	41	22	8	46
0	1	2	3	4	5	6

- If the array is sorted, the solution is straight-forward, so one approach is to start by sorting (as we'll soon see, this can be done in time  $O(n \log n)$ ).

A:

3	8	9	22	23	41	46
0	1	2	3	4	5	6

- However, sorting the array seems like overkill.

# A Detour via Partitioning

- Partitioning an array around some pivot element  $p$  means reorganizing the array so that all elements to the left of  $p$  are no greater than  $p$ , while those to the right are no smaller.

A:

9	23	3	41	22	8	46
0	1	2	3	4	5	6



# A Detour via Partitioning

- Partitioning an array around some pivot element  $p$  means reorganizing the array so that all elements to the left of  $p$  are no greater than  $p$ , while those to the right are no smaller.

A:

9	23	3	41	22	8	46
0	1	2	3	4	5	6

Partitioning around the pivot 9

# A Detour via Partitioning

- Partitioning an array around some pivot element  $p$  means reorganizing the array so that all elements to the left of  $p$  are no greater than  $p$ , while those to the right are no smaller.

A:

9	23	3	41	22	8	46
0	1	2	3	4	5	6

Partitioning around the pivot 9

A:

3	8	9	23	22	41	46
0	1	2	3	4	5	6

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

lo							hi
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	
s							

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

lo							hi
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	
s	i						

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	
$s$	$i$						

$lo$	$s$	$i$	$hi$
$p$	$< p$	$\geq p$	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	
$s$		$i$					

$lo$	$s$	$i$	$hi$
$p$	$< p$	$\geq p$	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	
	$s$		$i$				

$lo$	$s$		$i$	$hi$
$p$	$< p$	$\geq p$		



# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	3	23	41	22	8	46	
0	1	2	3	4	5	6	
	$s$	$i$					

$lo$	$s$		$i$	$hi$
$p$	$< p$	$\geq p$		

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	3	23	41	22	8	46	
0	1	2	3	4	5	6	
	$s$		$i$				

$lo$	$s$		$i$	$hi$
$p$	$< p$		$\geq p$	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
	9	3	23	41	22	8	46
0	1	2	3	4	5	6	
	$s$			$i$			

$lo$	$s$		$i$	$hi$
$p$	$< p$		$\geq p$	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	3	23	41	22	8	46	
0	1	2	3	4	5	6	
	$s$				$i$		

$lo$	$s$		$i$	$hi$
$p$	$< p$	$\geq p$		

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	3	23	41	22	8	46	
0	1	2	3	4	5	6	
		$s$			$i$		

$lo$	$s$		$i$	$hi$
$p$	$< p$	$\geq p$		

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	3	8	41	22	23	46	
0	1	2	3	4	5	6	
		$s$			$i$		

$lo$	$s$		$i$	$hi$
$p$	$< p$		$\geq p$	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$							$hi$
9	3	8	41	22	23	46	
0	1	2	3	4	5	6	
		$s$					$i$

$lo$	$s$		$i$	$hi$
$p$	$< p$		$\geq p$	

# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

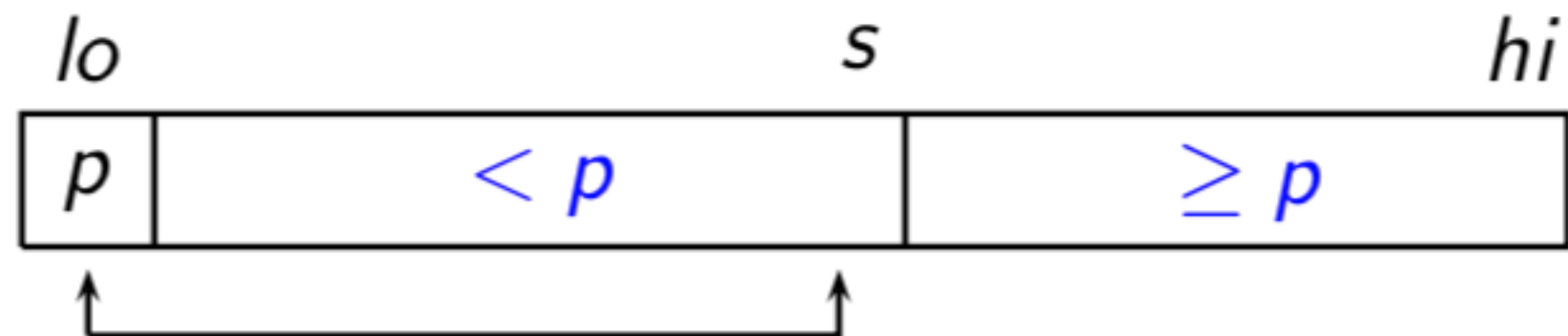
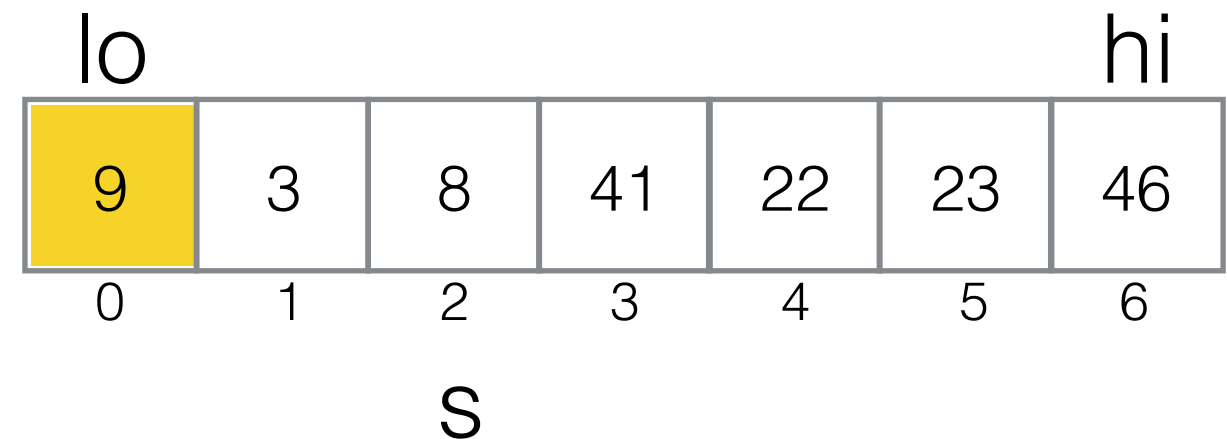
**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$





# Lomuto Partitioning



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

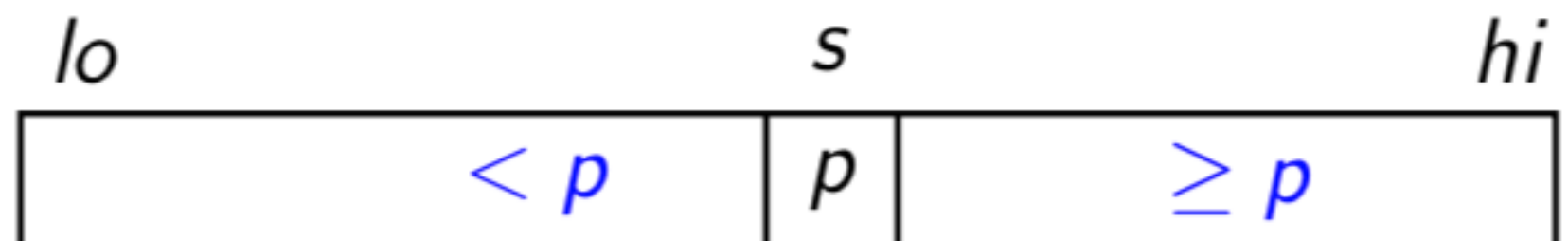
$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

$lo$						$hi$
8	3	9	41	22	23	46
0	1	2	3	4	5	6
		$s$				



# Finding the $k$ th-smallest Element



```
function QUICKSELECT( $A[\cdot]$ ,  $lo$ ,  $hi$ ,  $k$ )  
     $s \leftarrow$  LOMUTOPARTITION( $A$ ,  $lo$ ,  $hi$ )  
    if  $s - lo = k - 1$  then  
        return  $A[s]$   
    else  
        if  $s - lo > k - 1$  then  
            QUICKSELECT( $A$ ,  $lo$ ,  $s - 1$ ,  $k$ )  
        else  
            QUICKSELECT( $A$ ,  $s + 1$ ,  $hi$ ,  $(k - 1) - (s - lo)$ )
```

$lo$						$hi$
8	3	9	41	22	23	46
0	1	2	3	4	5	6
$S$						

# Example: Find the Sixth Smallest Element



```
function QUICKSELECT( $A[\cdot]$ ,  $lo$ ,  $hi$ ,  $k$ )  
     $s \leftarrow$  LOMUTOPARTITION( $A$ ,  $lo$ ,  $hi$ )  
    if  $s - lo = k - 1$  then  
        return  $A[s]$   
    else  
        if  $s - lo > k - 1$  then  
            QUICKSELECT( $A$ ,  $lo$ ,  $s - 1$ ,  $k$ )  
        else  
            QUICKSELECT( $A$ ,  $s + 1$ ,  $hi$ ,  $(k - 1) - (s - lo)$ )
```

$k$ : 6

$lo$							$hi$
9	23	3	41	22	8	46	
0	1	2	3	4	5	6	

# Example: Find the Fifth Smallest Element



```
function QUICKSELECT( $A[\cdot]$ ,  $lo$ ,  $hi$ ,  $k$ )  
     $s \leftarrow$  LOMUTOPARTITION( $A$ ,  $lo$ ,  $hi$ )  
    if  $s - lo = k - 1$  then  
        return  $A[s]$   
    else  
        if  $s - lo > k - 1$  then  
            QUICKSELECT( $A$ ,  $lo$ ,  $s - 1$ ,  $k$ )  
        else  
            QUICKSELECT( $A$ ,  $s + 1$ ,  $hi$ ,  $(k - 1) - (s - lo)$ )
```

$k$ : 6

$lo$						$hi$
8	3	9	41	22	23	46
0	1	2	3	4	5	6
$s$						

# Example: Find the Fifth Smallest Element



```
function QUICKSELECT( $A[\cdot]$ ,  $lo$ ,  $hi$ ,  $k$ )  
     $s \leftarrow$  LOMUTOPARTITION( $A$ ,  $lo$ ,  $hi$ )  
    if  $s - lo = k - 1$  then  
        return  $A[s]$   
    else  
        if  $s - lo > k - 1$  then  
            QUICKSELECT( $A$ ,  $lo$ ,  $s - 1$ ,  $k$ )  
        else  
            QUICKSELECT( $A$ ,  $s + 1$ ,  $hi$ ,  $(k - 1) - (s - lo)$ )
```

$k: 3$

			lo		hi	
8	3	9	41	22	23	46
0	1	2	3	4	5	6

# Example: Find the Fifth Smallest Element



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

			lo				hi
8	3	9	41	22	23	46	
0	1	2	3	4	5	6	
			s	i			

# Example: Find the Fifth Smallest Element



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

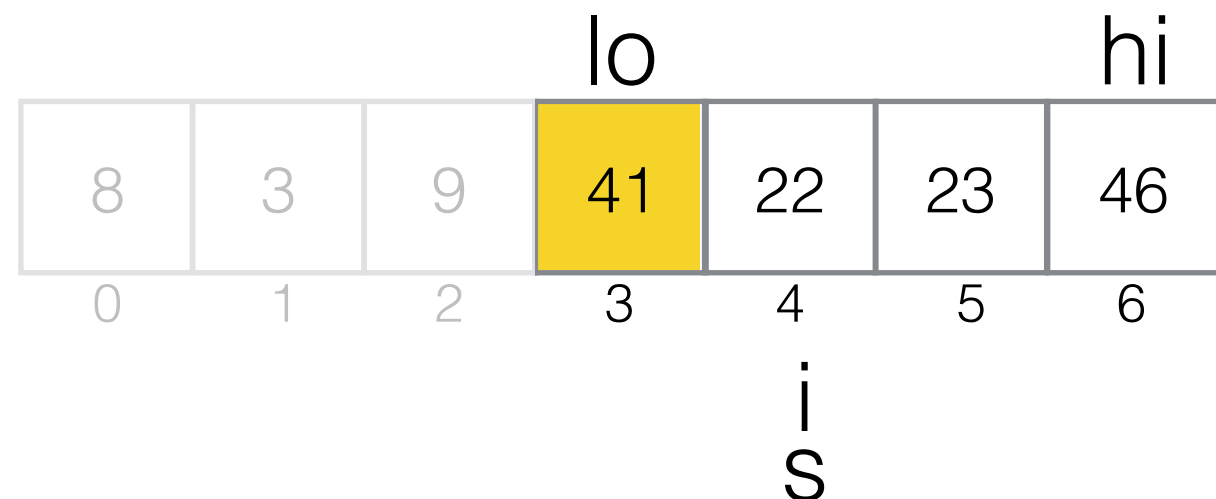
**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$





# Example: Find the Fifth Smallest Element



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

			lo				hi
8	3	9	41	22	23	46	
0	1	2	3	4	5	6	
				s	i		



# Example: Find the Fifth Smallest Element



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

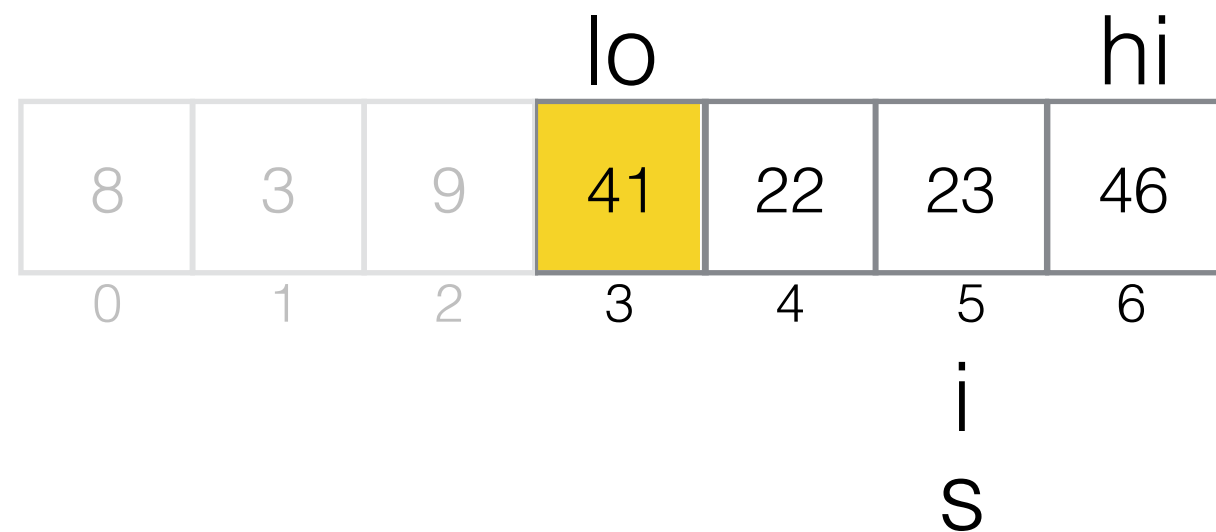
**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$



# Example: Find the Fifth Smallest Element



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

			lo				hi
8	3	9	41	22	23	46	
0	1	2	3	4	5	6	
					s	i	

# Example: Find the Fifth Smallest Element



**function** LOMUTOPARTITION( $A[\cdot]$ ,  $lo$ ,  $hi$ )

$p \leftarrow A[lo]$

$s \leftarrow lo$

**for**  $i \leftarrow lo + 1$  **to**  $hi$  **do**

**if**  $A[i] < p$  **then**

$s \leftarrow s + 1$

$swap(A[s], A[i])$

$swap(A[lo], A[s])$

**return**  $s$

			lo				hi
8	3	9	41	22	23	46	
0	1	2	3	4	5	6	

S

```
return s
```

Diagram illustrating an array structure with 7 elements. The elements are 8, 3, 9, 23, 22, 41, and 46. The first three elements (8, 3, 9) are in light gray boxes. The last three elements (23, 22, 41, 46) are in white boxes. The element 41 is highlighted in yellow. The range from index 3 to 6 is labeled 'lo' and 'hi'.

Index	Value
0	8
1	3
2	9
3	23
4	22
5	41
6	46

# Example: Find the Fifth Smallest Element



```
function QUICKSELECT( $A[\cdot]$ ,  $lo$ ,  $hi$ ,  $k$ )  
     $s \leftarrow$  LOMUTOPARTITION( $A$ ,  $lo$ ,  $hi$ )  
    if  $s - lo = k - 1$  then  
        return  $A[s]$   
    else  
        if  $s - lo > k - 1$  then  
            QUICKSELECT( $A$ ,  $lo$ ,  $s - 1$ ,  $k$ )  
        else  
            QUICKSELECT( $A$ ,  $s + 1$ ,  $hi$ ,  $(k - 1) - (s - lo)$ )
```

$k: 3$

			$lo$			$hi$
8	3	9	23	22	41	46
0	1	2	3	4	5	6
						$s$

# Example: Find the Fifth Smallest Element



```
function QUICKSELECT( $A[\cdot]$ ,  $lo$ ,  $hi$ ,  $k$ )  
   $s \leftarrow$  LOMUTOPARTITION( $A$ ,  $lo$ ,  $hi$ )  
  if  $s - lo = k - 1$  then  
    return  $A[s]$   
  else  
    if  $s - lo > k - 1$  then  
      QUICKSELECT( $A$ ,  $lo$ ,  $s - 1$ ,  $k$ )  
    else  
      QUICKSELECT( $A$ ,  $s + 1$ ,  $hi$ ,  $(k - 1) - (s - lo)$ )
```

$k: 3$

returns 41!

			$lo$			$hi$
8	3	9	23	22	41	46
0	1	2	3	4	5	6
						$s$

# QuickSelect Complexity



- **Worst case** complexity for QuickSelect is quadratic,
- **Average-case** complexity is linear.

# Interpolation Search

- If the elements of a sorted array are distributed reasonably evenly, we can do better than binary search!
- Think about how you search for an entry in the telephone directory: If you look for ‘Zobel’, you make a rough estimate of where to do the first probe—very close to the end of the directory.
- This is the idea in interpolation search.
- When searching for  $k$  in the array segment  $A[lo]$  to  $A[hi]$ , take into account where  $k$  is, relative to  $A[lo]$  and  $A[hi]$ .



# Interpolation Search



THE UNIVERSITY OF  
**MELBOURNE**

A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

Suppose we are searching for  $k = 83$

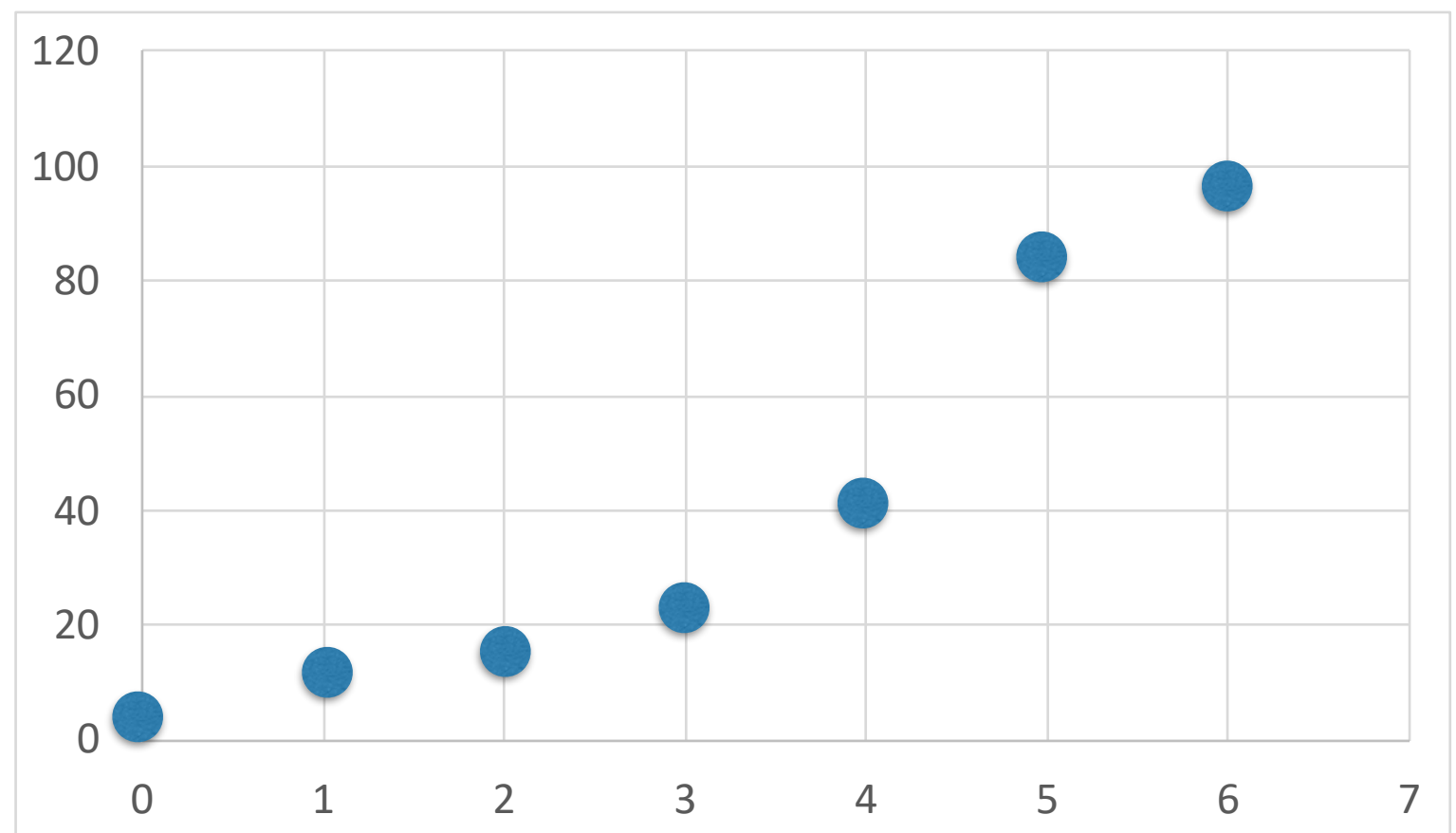
# Interpolation Search



A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

Suppose we are searching for  $k = 83$



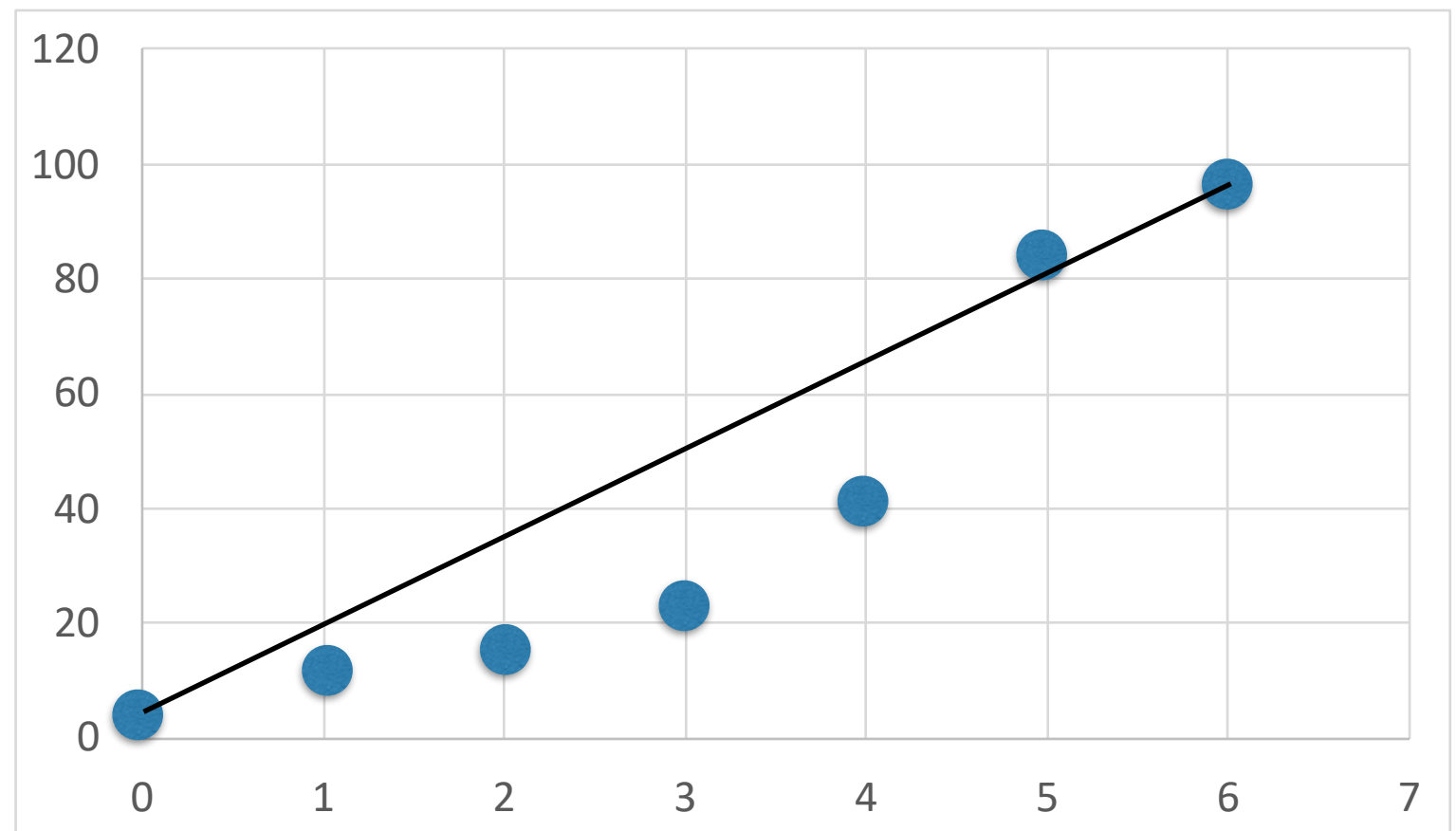
# Interpolation Search



A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

Suppose we are searching for  $k = 83$



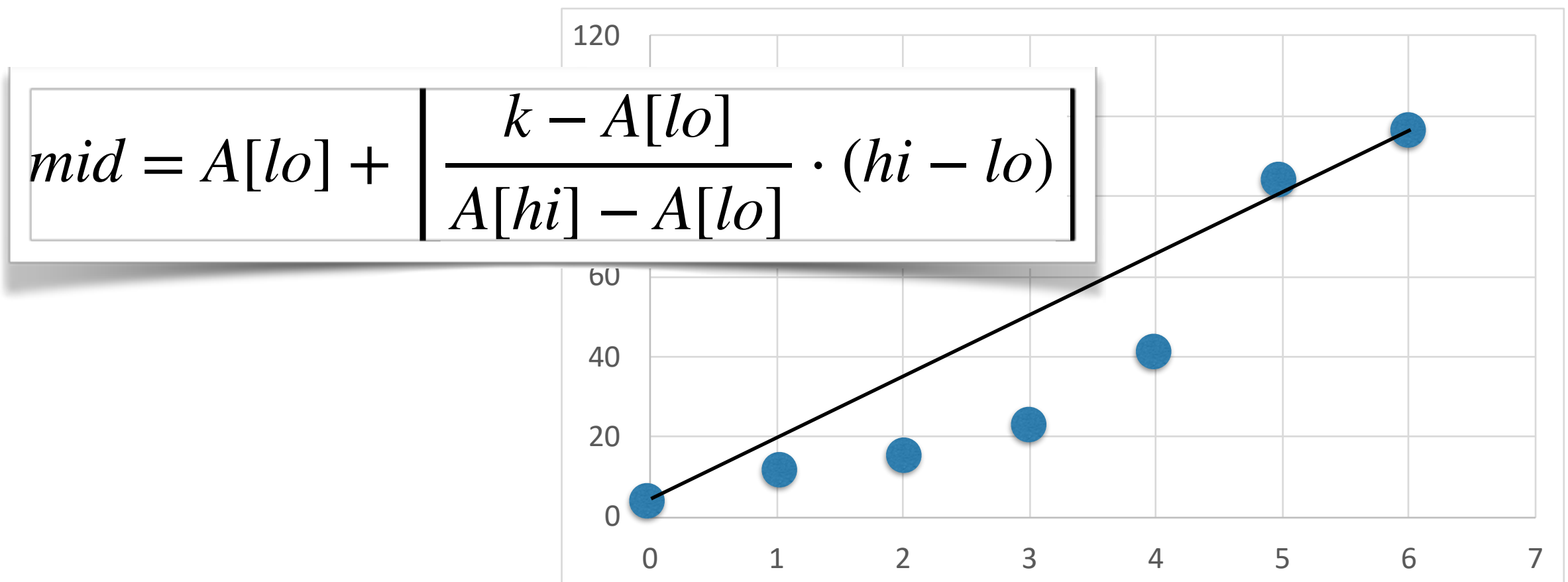
# Interpolation Search



A:

4	9	13	22	41	83	96
0	1	2	3	4	5	6

Suppose we are searching for  $k = 83$



# Interpolation Search

- Instead of computing the mid-point  $m$  as in binary search:

$$m \leftarrow \lfloor (lo + hi)/2 \rfloor$$

we instead perform linear interpolation between the points  $(lo, A[lo])$  and  $(hi, A[hi])$ . That is, we use:

$$m \leftarrow lo + \left\lfloor \frac{k - A[lo]}{A[hi] - A[lo]} (hi - lo) \right\rfloor$$

- Interpolation search has average complexity  $O(\log \log n)$
- It is the right choice for large arrays when elements are **uniformly distributed**

# Next Week

- Learn to divide and conquer!
- Read Levitin Chapter 5, but skip 5.4.