THE UNIVERSITY OF MELBOURNE

COMP90038 Algorithms and Complexity

Lecture 10: Deuer-by-a-Factor

(with thanks to Haraergaard)

Toby Murray

toby.murray@unimelb.edu.au

DMD 8.17 (Level 8, Doug McDonell Bldg)

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

@tobycmurray

Decrease-and-Conquer

THE UNIVERSITY OF MELBOURNE

- **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
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

THE UNIVERSITY OF MELBOURNE

- We now look at better utilization of the approach, often leading to methods with logarithmic time behaviour or better!

- **Decrease-by-a-c** 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 th [m – 1] recursively.

- If A[m] < k, search the sub-array + 1] recursively.

$$\underbrace{A[0] \;\cdots\; A[m-1]}_{\text{search here if } A[m] > k} \quad \overset{\displaystyle = k?}{\overset{\downarrow}{\color{blue}{A[m]}}} \quad \underbrace{A[m+1] \;\cdots\; A[n-1]}_{\text{search here if } A[m] < k}$$

Binary Search

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

Example:
Binary Search in Sorted Array

THE UNIVERSITY OF MELBOURNE

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

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

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

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

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 sarch:
  - When *k* is not in the array
- In that case, its complexity is given by the following recursive equation:

- A closed form is:
- 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 Θ(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*

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

= 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. w

A:

9	23	3	41	[illegible]	[illegible]	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					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 $\text{LomutoPartition}(A[\cdot], lo, hi)$

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

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

Lomuto Partitioning

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

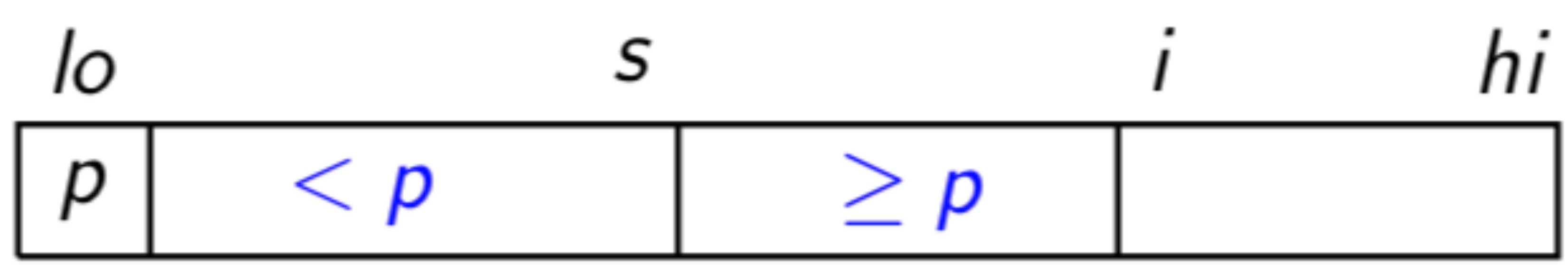
lo						hi
9	23	3	41	22	8	46
	1	2	3	4	5	6
		i				

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

Lomuto Partitioning

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

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



Lomuto Partitioning

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

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

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

Lomuto Partitioning

THE UNIVERSITY OF MELBOURNE

function $\text{LomutoPartition}(A[\cdot], lo, hi)$

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

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

Lomuto Partitioning

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

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

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

Lomuto Partitioning

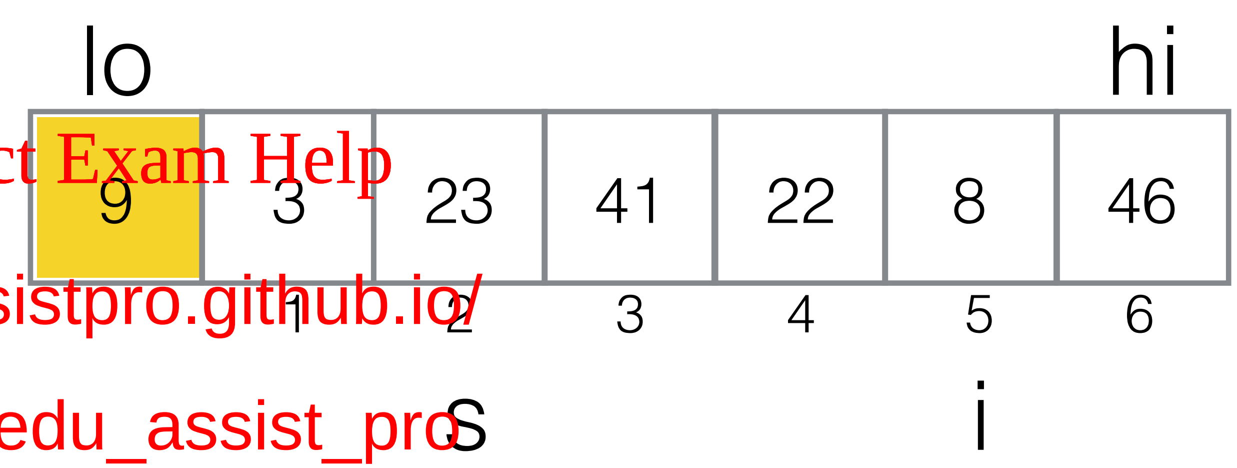
function LomutoPartition($A[\cdot]$, lo, hi)

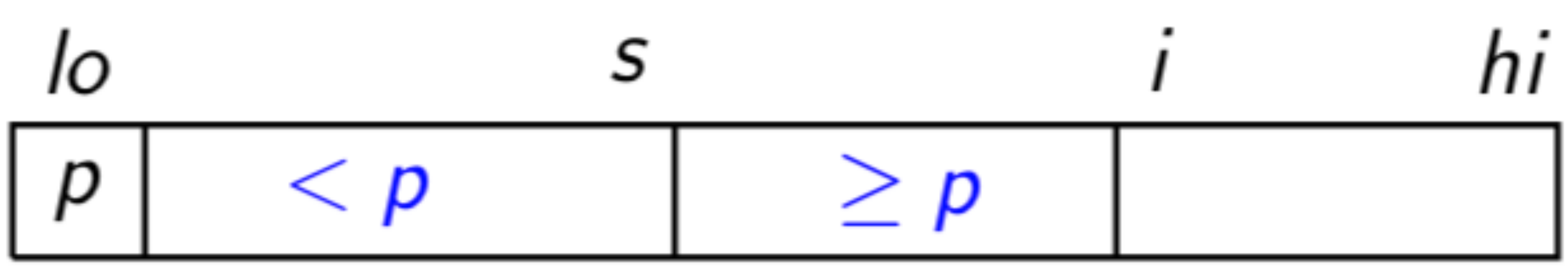
lo						hi
9	3	23	41	22	8	46
	1	2	3	4	5	6
					i	

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

Lomuto Partitioning

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





Lomuto Partitioning

function $\text{LomutoPartition}(A[\cdot], lo, hi)$

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

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

Lomuto Partitioning

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

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

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

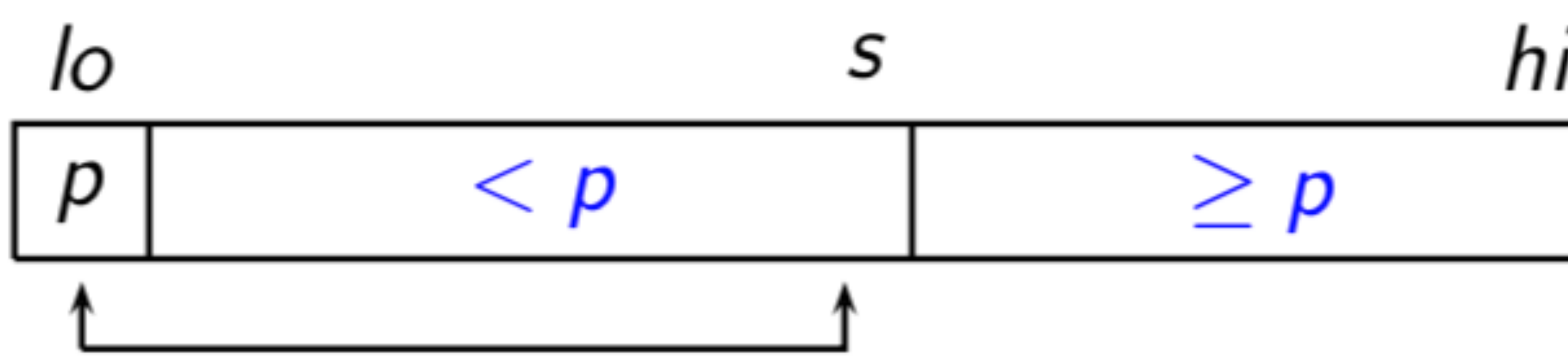
Lomuto Partitioning

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

lo hi

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

S



Lomuto Partitioning

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

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

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

Finding the *k*th-smallest Element

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

Example: Find the Sixth Smallest Element

k: 6

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

Example: Find the Fifth Smallest Element

k: 6

lo hi

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

s

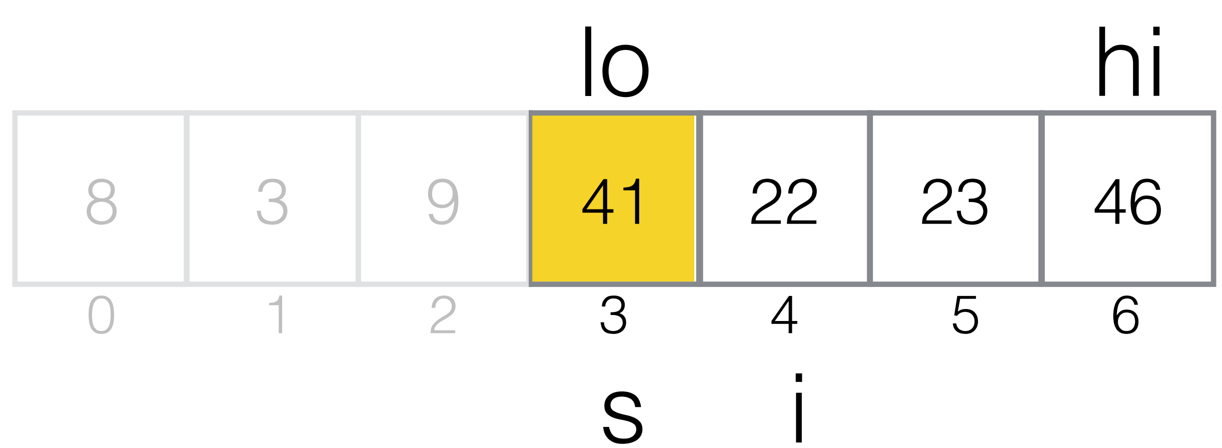
Example: Find the Fifth Smallest Element

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($A[\cdot], lo, hi$)



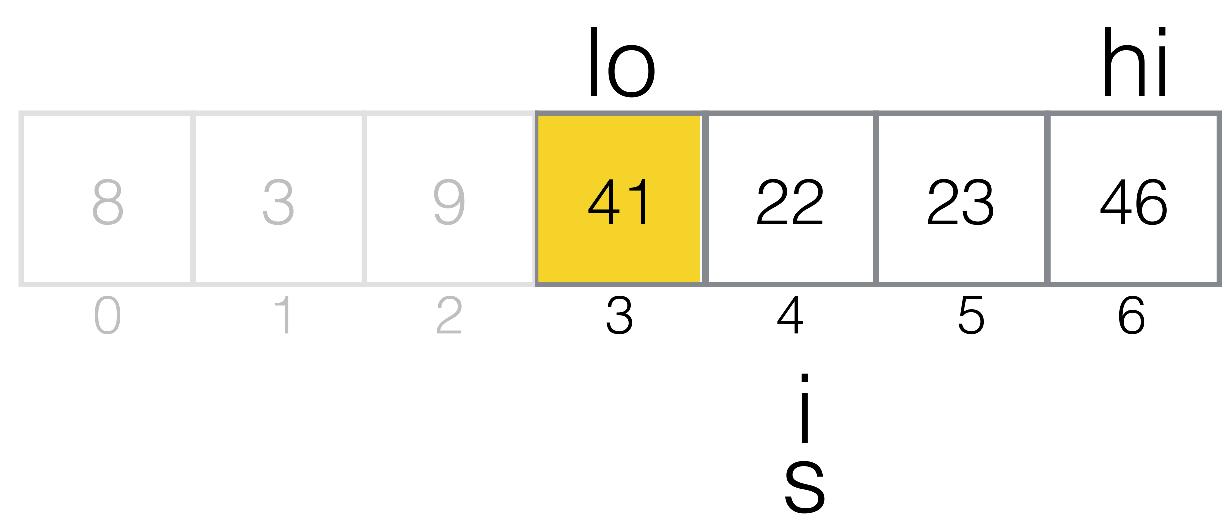
Example: Find the Fifth Smallest Element

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

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro



Example: Find the Fifth Smallest Element

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

			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)

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

Example: Find the Fifth Smallest Element

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

			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$(A[\cdot], lo, hi)$

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

Example: Find the Fifth Smallest Element

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

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

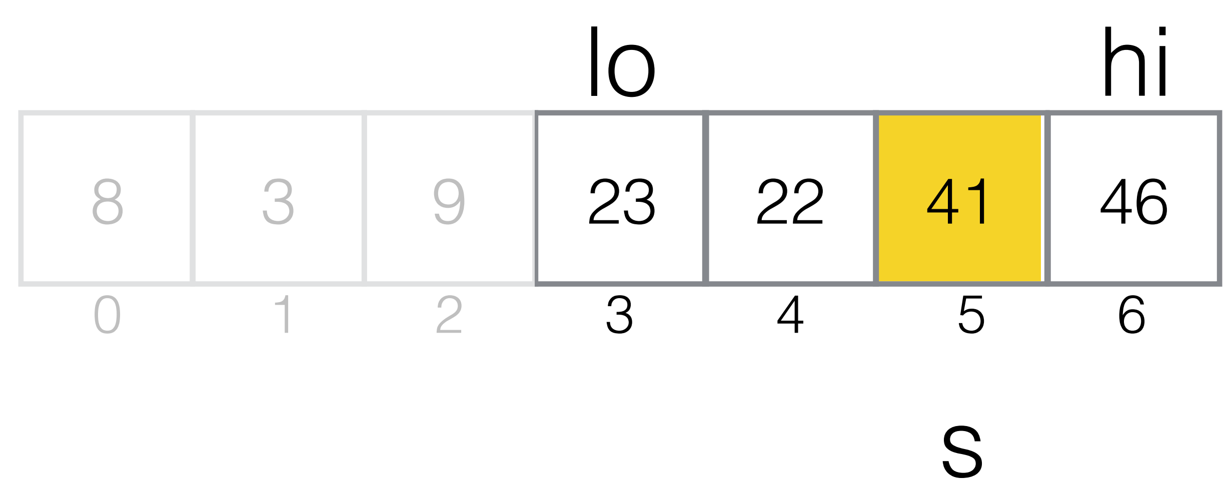
Example: Find the Fifth Smallest Element

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

k: 3



returns 41!

QuickSelect Complexity

- **Worst case** com
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 director obel', you make a rough estimate of st probe—very close to the end of the direct

- 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].

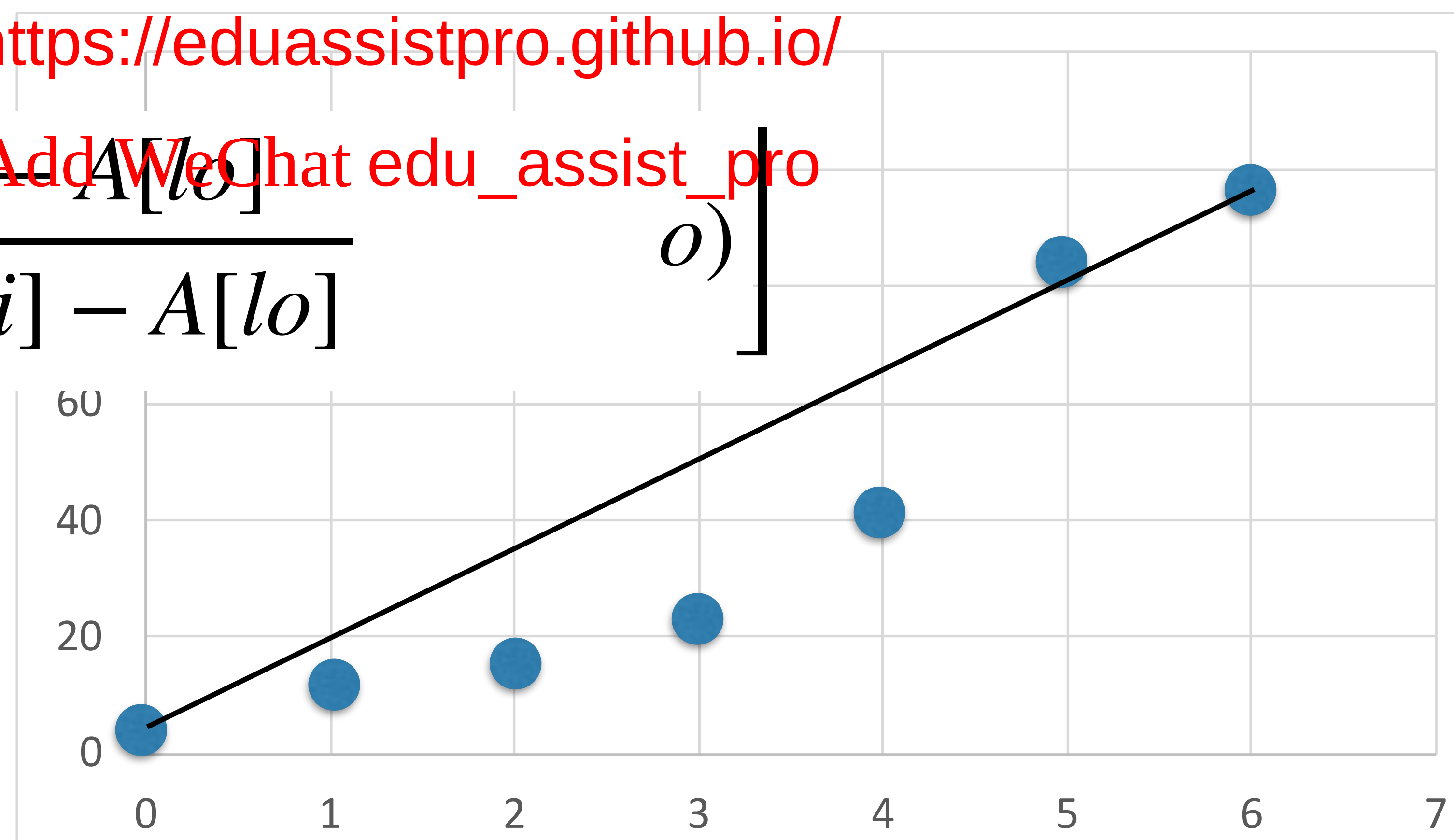
Interpolation Search

A:

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

Suppose we are searching for k = 83

$$mid = A[lo] + \left\lfloor \frac{k - A[lo]}{A[hi] - A[lo]} \quad o) \right\rfloor$$



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 , 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 a

- Read Levitin Chapter 5, bu .4.

