



THE UNIVERSITY OF
MELBOURNE

COMP90038

Algorithms and Complexity

Lecture 9: Decrease-and-Conquer-by-a-Constant
(with thanks to Harald Søndergaard)

Toby Murray



toby.murray@unimelb.edu.au



DMD 8.17 (Level 8, Doug McDonnell Bldg)



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



@tobycmurray

Decrease-and-Conquer-by-a-Constant

- In this approach, the size of the problem is reduced by some **constant** in each iteration of the algorithm.
- A simple example is the following approach to sorting: To sort an array of length n , just
 1. sort the first $n - 1$ items, then
 2. locate the cell that should hold the last item, shift all elements to its right to the right, and place the last element.

Sorting n items



THE UNIVERSITY OF
MELBOURNE

A:

23	9	52	12	41	83	46
0	1	2	3	4	5	6

Sorting n items

A:

23	9	52	12	41	83	46
0	1	2	3	4	5	6

Sort first $n-1$ items

Sorting n items

A:

23	9	52	12	41	83	46
0	1	2	3	4	5	6

Sort first $n-1$ items

Sorting n items

A:

9	12	23	41	52	83	46
0	1	2	3	4	5	6

Sorting n items



THE UNIVERSITY OF
MELBOURNE

A:

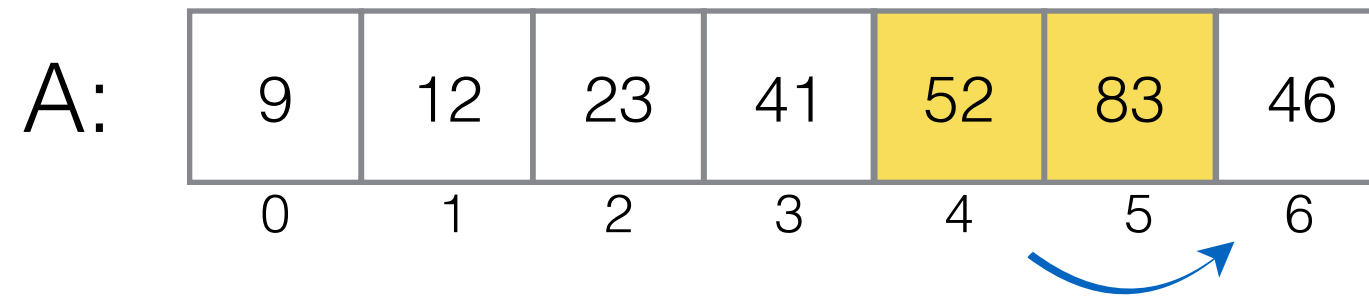
9	12	23	41	52	83	46
0	1	2	3	4	5	6

Sorting n items

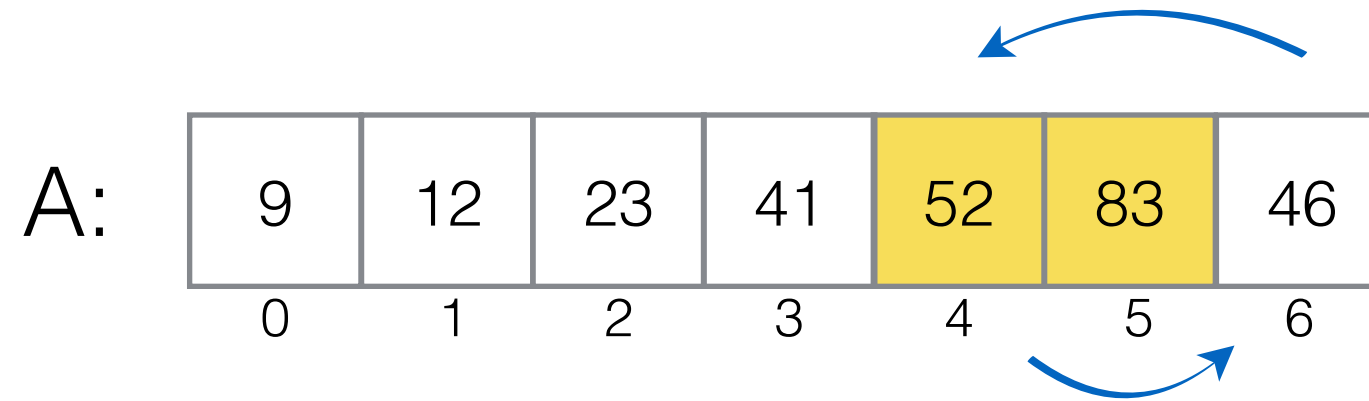
A:

9	12	23	41	52	83	46
0	1	2	3	4	5	6

Sorting n items



Sorting n items



Sorting n items

A:

9	12	23	41	52	83	46
0	1	2	3	4	5	6

Sorting n items



THE UNIVERSITY OF
MELBOURNE

A:

9	12	23	41	46	52	83
0	1	2	3	4	5	6

Sorting n items



THE UNIVERSITY OF
MELBOURNE

A:

9	12	23	41	52	83	46
0	1	2	3	4	5	6

Sorting n items

A:

9	12	23	41	52	83	46
0	1	2	3	4	5	6

v: 46

Sorting n items

A:

9	12	23	41	52	83	46
0	1	2	3	4	5	6

↑
A[j]

v: 46

Sorting n items

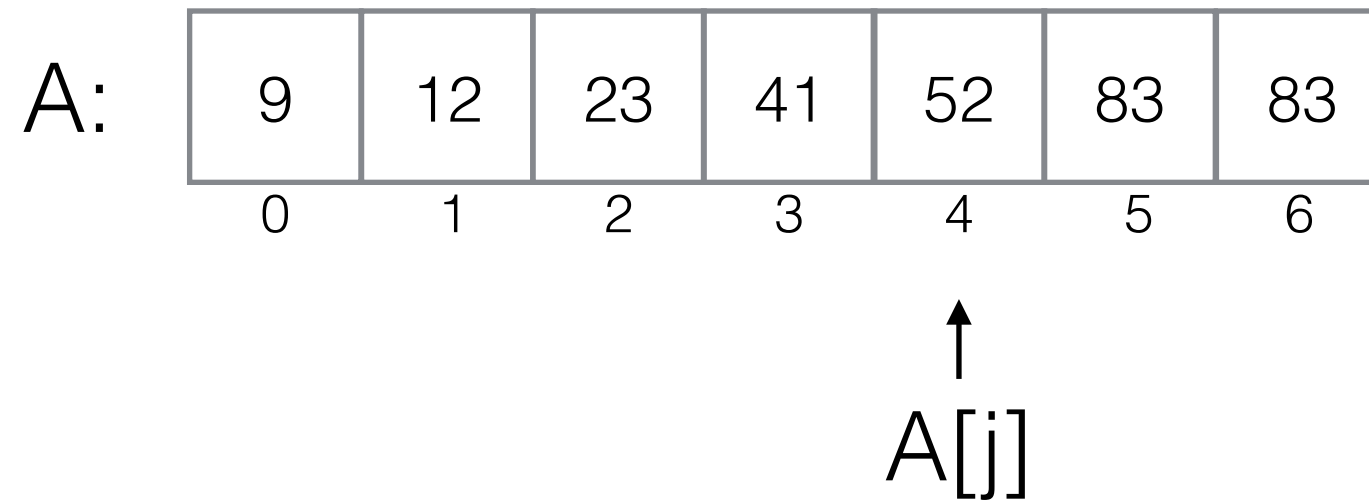
A:

9	12	23	41	52	83	83
0	1	2	3	4	5	6

↑
A[j]

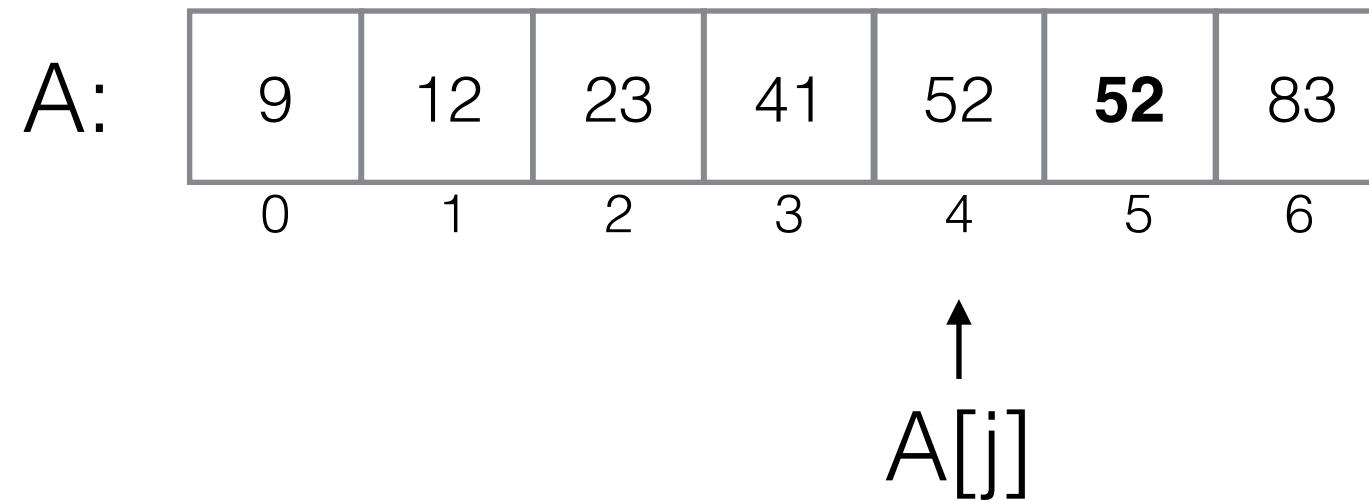
v: 46

Sorting n items



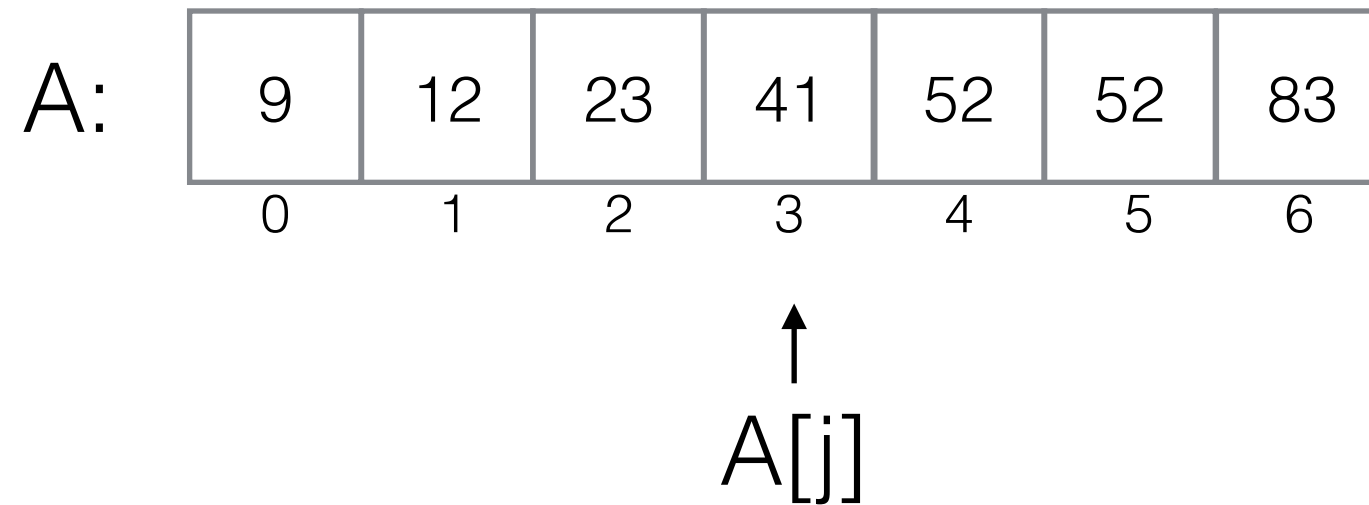
v: 46

Sorting n items



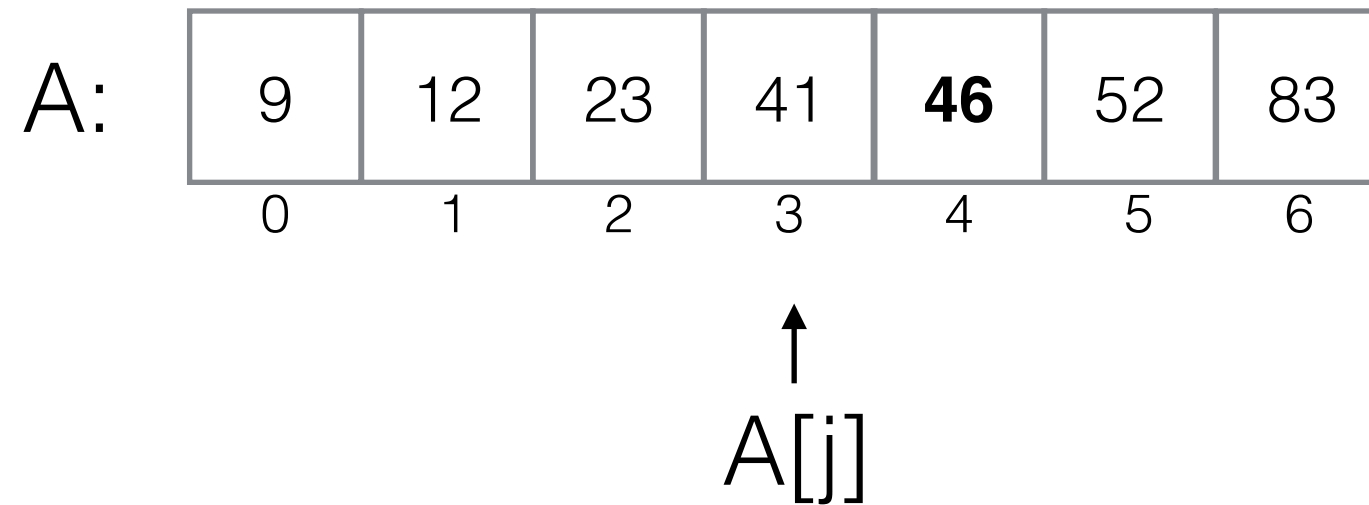
v: 46

Sorting n items



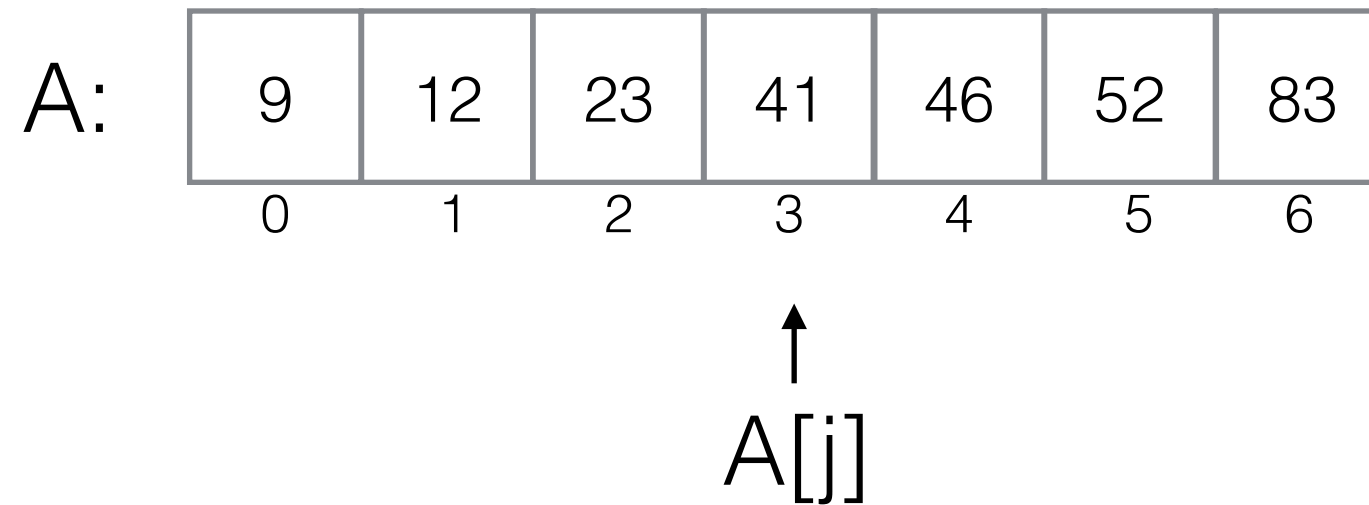
v: 46

Sorting n items



v: 46

Sorting n items



v: 46

Insertion Sort

- Sorting an array $A[0]..A[n - 1]$:
- To sort $A[0] .. A[i]$ first sort $A[0] .. A[i-1]$, then insert $A[i]$ in its proper place

```
function INSERTIONSORT( $A[\cdot]$ ,  $n$ )  
  for  $i \leftarrow 1$  to  $n - 1$  do  
     $v \leftarrow A[i]$   
     $j \leftarrow i - 1$   
    while  $j \geq 0$  and  $v < A[j]$  do  
       $A[j + 1] \leftarrow A[j]$   
       $j \leftarrow j - 1$   
     $A[j + 1] \leftarrow v$ 
```

Complexity of Insertion Sort



THE UNIVERSITY OF
MELBOURNE

- The for loop is traversed $n - 1$ times. In the i th round, the test $v < A[j]$ is performed i times, in the worst case.
- Hence the worst-case running time is

$$\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

- What does input look like in the worst case?

Complexity of Insertion Sort



THE UNIVERSITY OF
MELBOURNE

- The for loop is traversed $n - 1$ times. In the i th round, the test $v < A[j]$ is performed i times, in the worst case.
- Hence the worst-case running time is

$$\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i :$$

- What does input look like in the worst case?

Complexity of Insertion Sort



THE UNIVERSITY OF
MELBOURNE

- The for loop is traversed $n - 1$ times. In the i th round, the test $v < A[j]$ is performed i times, in the worst case.
- Hence the worst-case running time is

$$\sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

- What does input look like in the worst case?

The Trick of Posting a Sentinel

- If we are sorting elements from a domain that is bounded from below, that is, there is a minimal element \min , and the array A was known to have a free cell to the left of $A[0]$, then we could simplify the test. Namely, we would place \min (a sentinel) in that cell ($A[-1]$) and change the test from

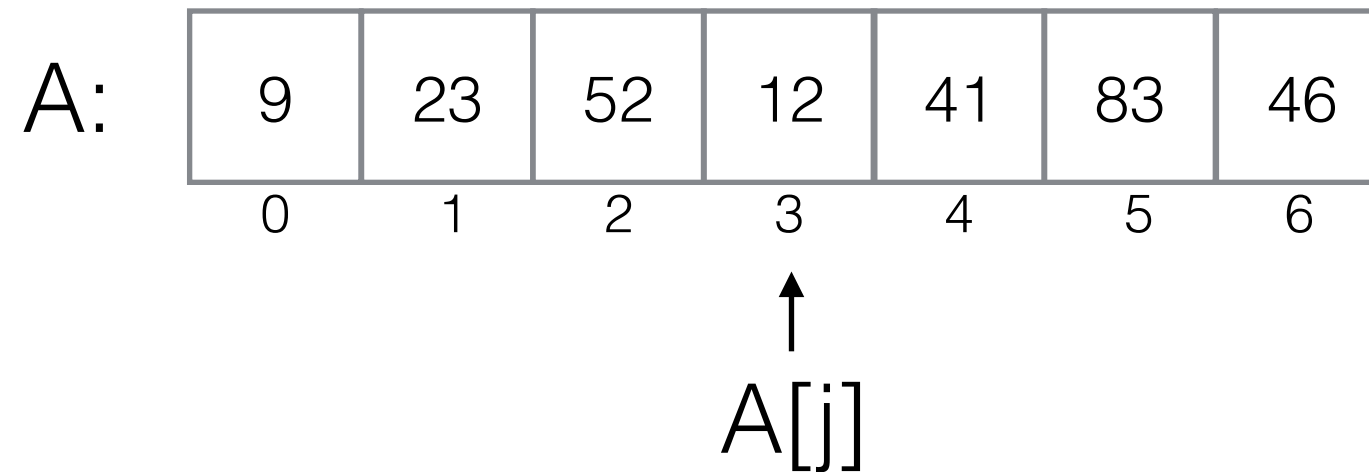
$$j \geq 0 \text{ and } v < A[j]$$

to just

$$v < A[j]$$

- That will speed up insertion sort by a constant factor.
- For this reason, extreme array cells (such as $A[0]$ in C, and/or $A[n + 1]$) are sometimes left free deliberately, so that they can be used to hold sentinels; only $A[1]$ to $A[n]$ hold proper data.

Posting a Sentinel



Test required: $j \geq 0$ and $v < A[j]$

Posting a Sentinel

A:

-1	9	23	52	12	41	83	46
0	1	2	3	4	5	6	7

↑
A[j]

Test required: $v < A[j]$

Properties of Insertion Sort



- Easy to understand and implement.
- Average-case and worst-case complexity both quadratic.
- However, linear for almost-sorted input.
- Some cleverer sorting algorithms perform almost-sorting and then let insertion sort take over.
- Very good for small arrays (say, a couple of hundred elements).
- In-place?
- Stable?

Properties of Insertion Sort



- Easy to understand and implement.
- Average-case and worst-case complexity both quadratic.
- However, linear for almost-sorted input.
- Some cleverer sorting algorithms perform almost-sorting and then let insertion sort take over.
- Very good for small arrays (say, a couple of hundred elements).
- In-place? *yes*
- Stable?

Properties of Insertion Sort

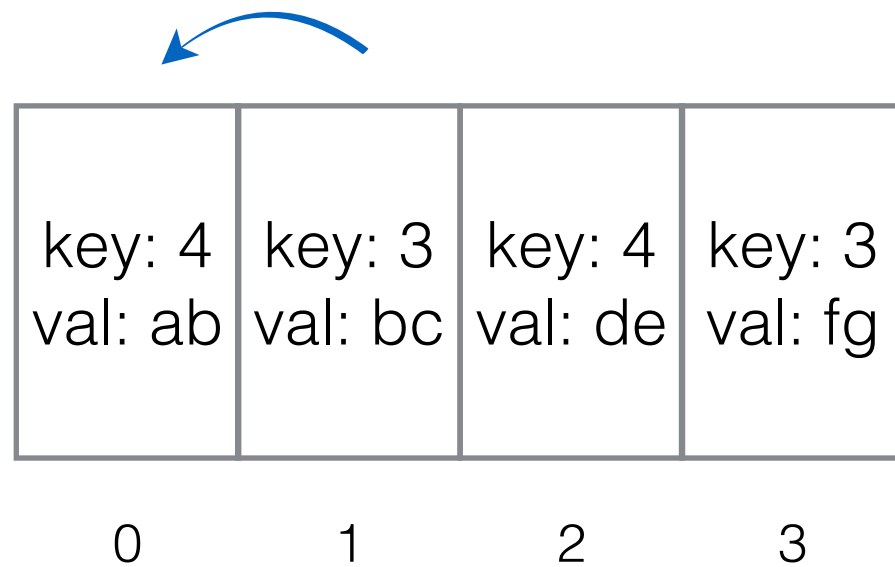


- Easy to understand and implement.
- Average-case and worst-case complexity both quadratic.
- However, linear for almost-sorted input.
- Some cleverer sorting algorithms perform almost-sorting and then let insertion sort take over.
- Very good for small arrays (say, a couple of hundred elements).
- In-place? *yes*
- Stable? *?*

Insertion Sort Stability

key: 4 val: ab	key: 3 val: bc	key: 4 val: de	key: 3 val: fg
0	1	2	3

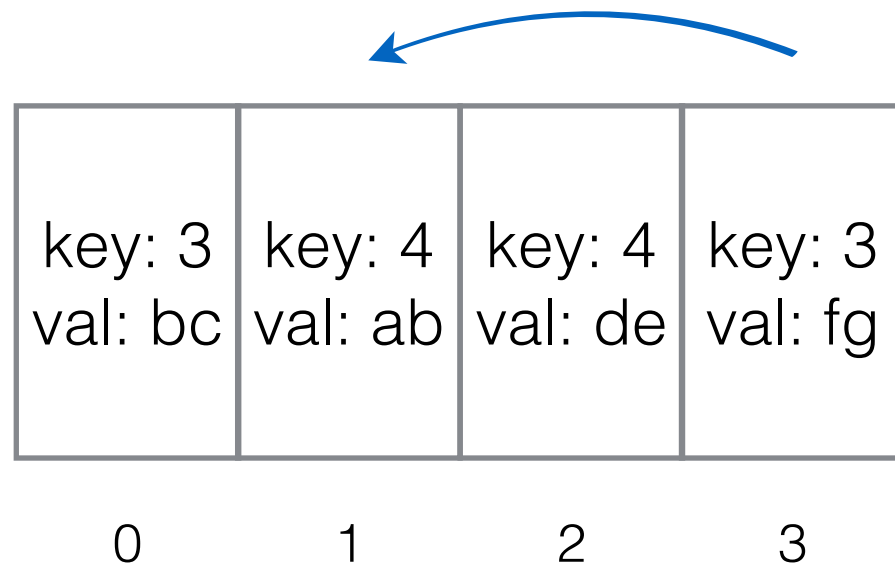
Insertion Sort Stability



Insertion Sort Stability

key: 3 val: bc	key: 4 val: ab	key: 4 val: de	key: 3 val: fg
0	1	2	3

Insertion Sort Stability



Insertion Sort Stability

key: 3 val: bc	key: 3 val: fg	key: 4 val: ab	key: 4 val: de
0	1	2	3

Insertion Sort Stability

key: 3 val: bc	key: 3 val: fg	key: 4 val: ab	key: 4 val: de
0	1	2	3

Stable

Properties of Insertion Sort



- Easy to understand and implement.
- Average-case and worst-case complexity both quadratic.
- However, linear for almost-sorted input.
- Some cleverer sorting algorithms perform almost-sorting and then let insertion sort take over.
- Very good for small arrays (say, a couple of hundred elements).
- In-place?
- Stable?

Properties of Insertion Sort



- Easy to understand and implement.
- Average-case and worst-case complexity both quadratic.
- However, linear for almost-sorted input.
- Some cleverer sorting algorithms perform almost-sorting and then let insertion sort take over.
- Very good for small arrays (say, a couple of hundred elements).
- In-place? *yes*
- Stable?

Properties of Insertion Sort



- Easy to understand and implement.
- Average-case and worst-case complexity both quadratic.
- However, linear for almost-sorted input.
- Some cleverer sorting algorithms perform almost-sorting and then let insertion sort take over.
- Very good for small arrays (say, a couple of hundred elements).
- In-place? *yes*
- Stable? *yes*

Shellsort: Motivation

A:

9	8	7	6	5	4	3
0	1	2	3	4	5	6

Shellsort: Motivation

A:

9	8	7	6	5	4	3
0	1	2	3	4	5	6

A:

8	9	7	6	5	4	3
0	1	2	3	4	5	6

Shellsort: Motivation

A:

9	8	7	6	5	4	3
0	1	2	3	4	5	6

A:

8	9	7	6	5	4	3
0	1	2	3	4	5	6

A:

7	8	9	6	5	4	3
0	1	2	3	4	5	6

Shellsort: Motivation

A:

9	8	7	6	5	4	3
0	1	2	3	4	5	6

A:

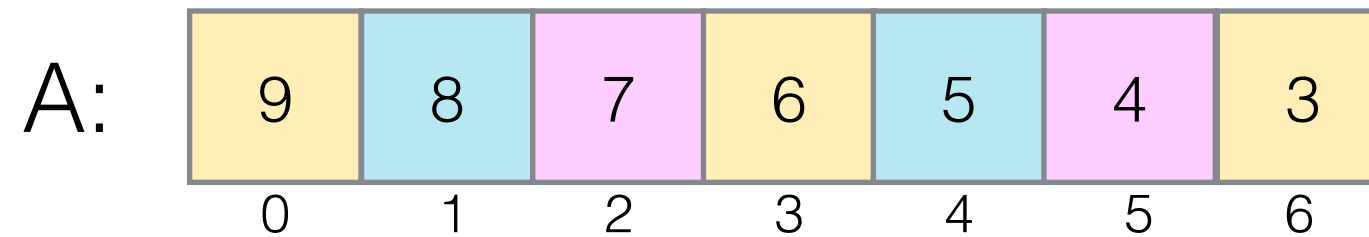
8	9	7	6	5	4	3
0	1	2	3	4	5	6

A:

7	8	9	6	5	4	3
0	1	2	3	4	5	6

It would be better if we could move the 9, 8, etc.
to the right faster

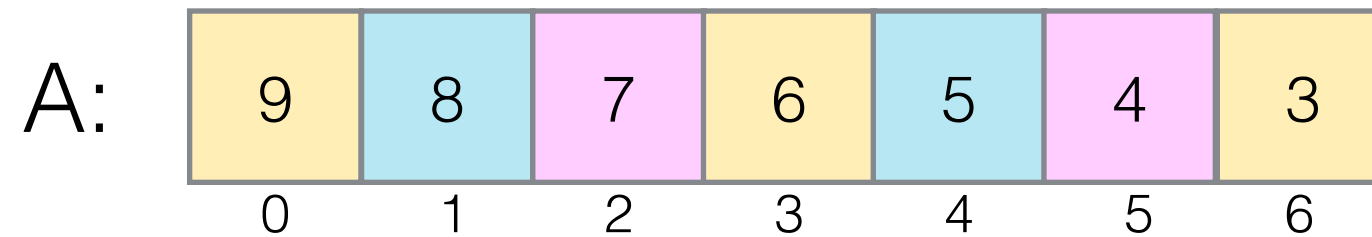
Shellsort: Motivation



Shellsort: Motivation



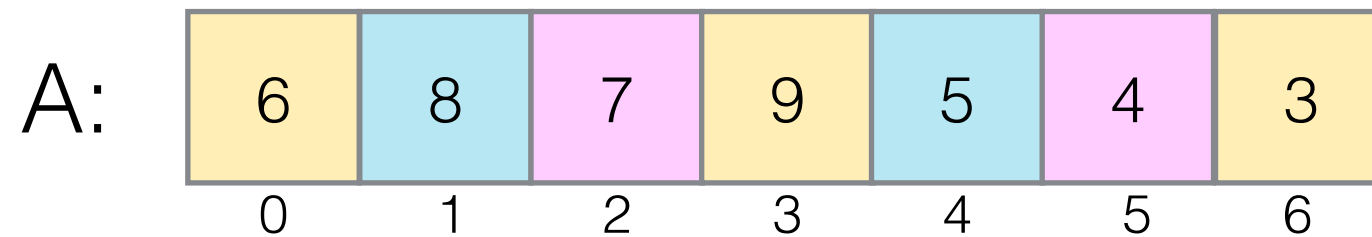
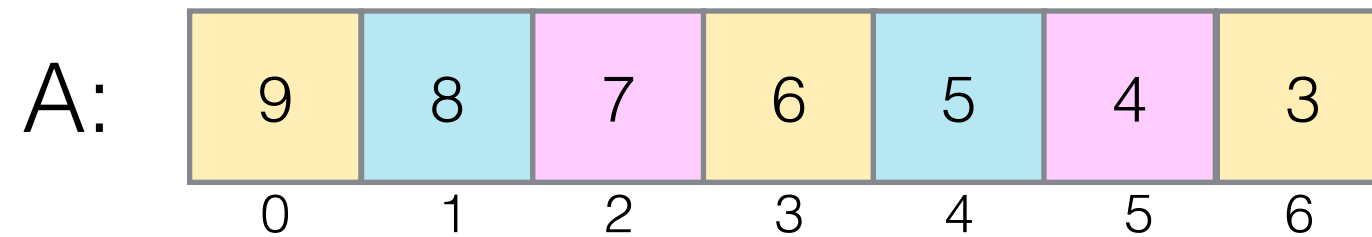
Sort the yellow entries



Shellsort: Motivation



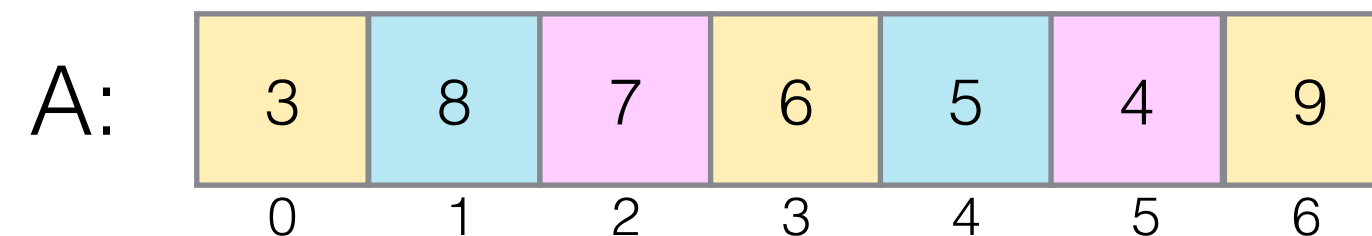
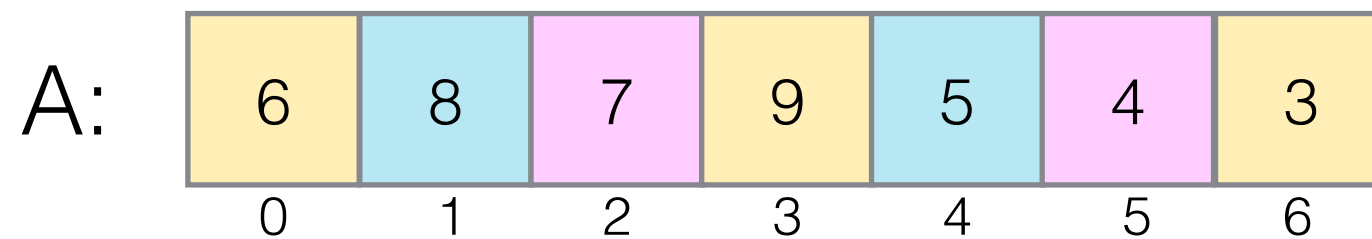
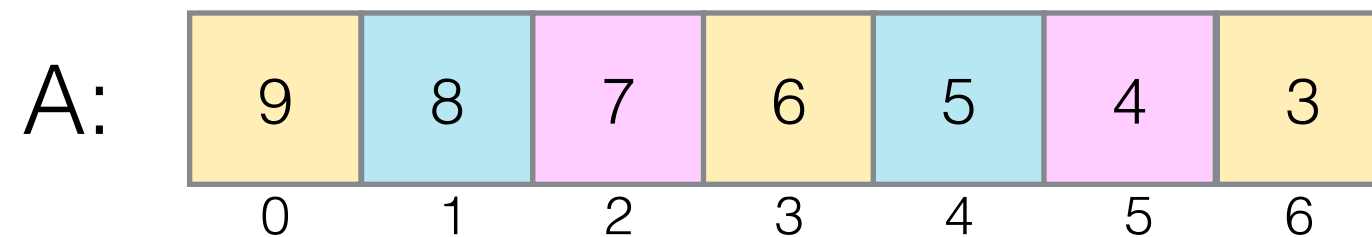
Sort the yellow entries



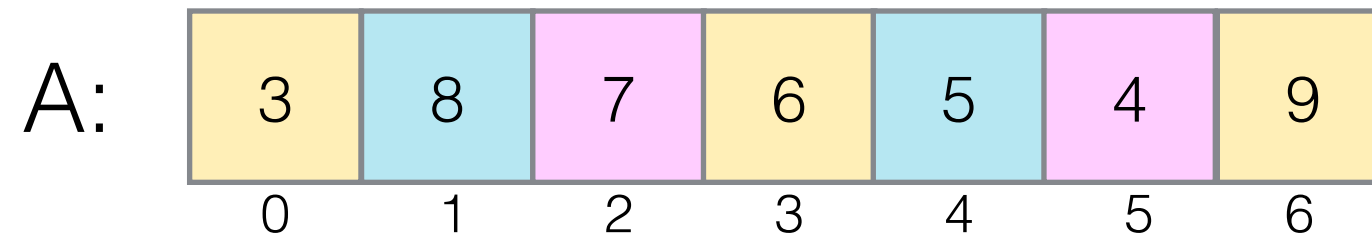
Shellsort: Motivation



Sort the yellow entries

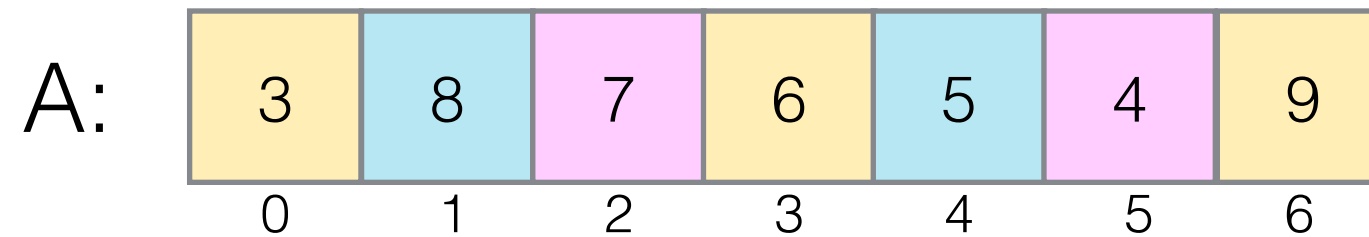


Shellsort: Motivation



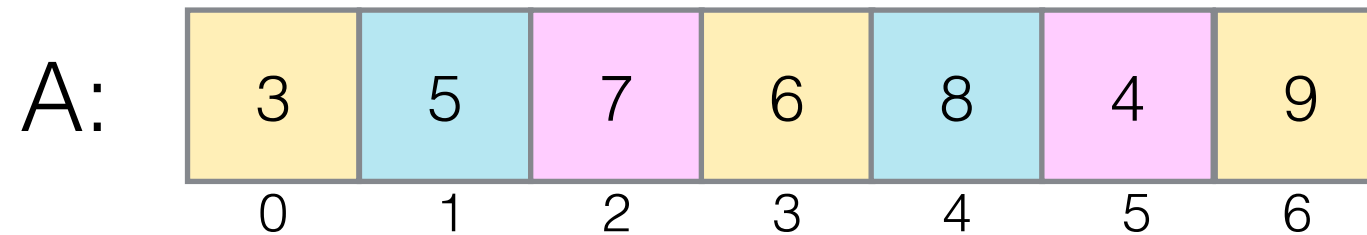
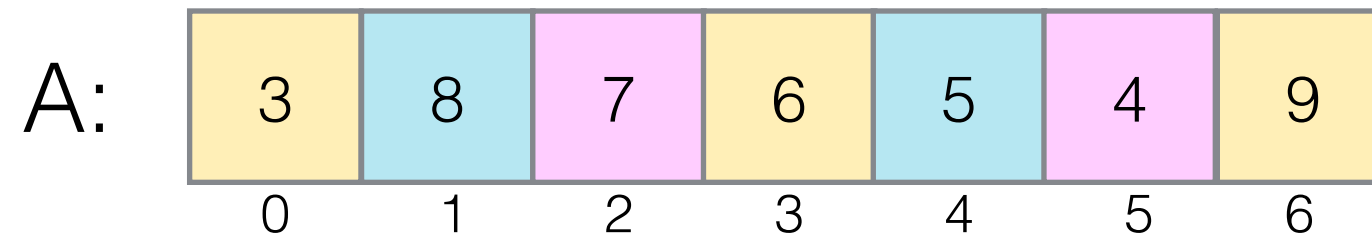
Shellsort: Motivation

Sort the blue entries



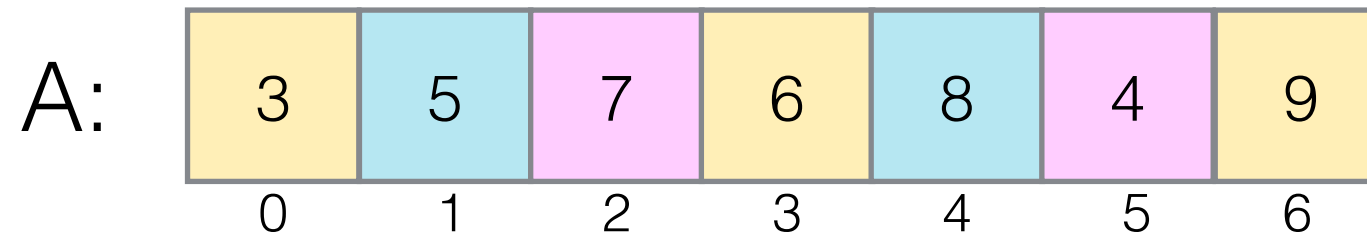
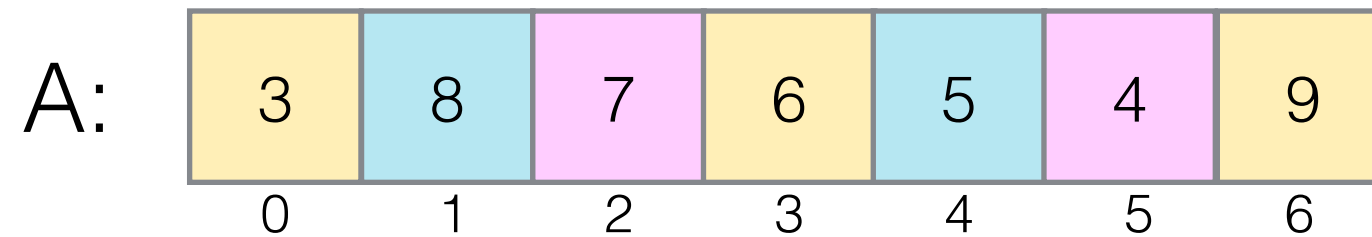
Shellsort: Motivation

Sort the blue entries



Shellsort: Motivation

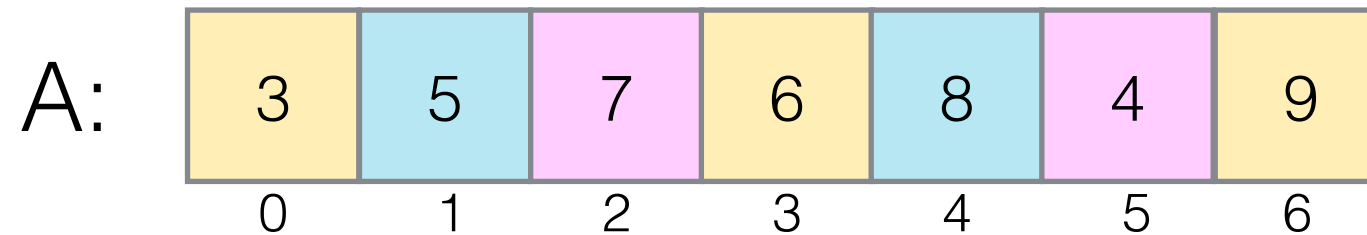
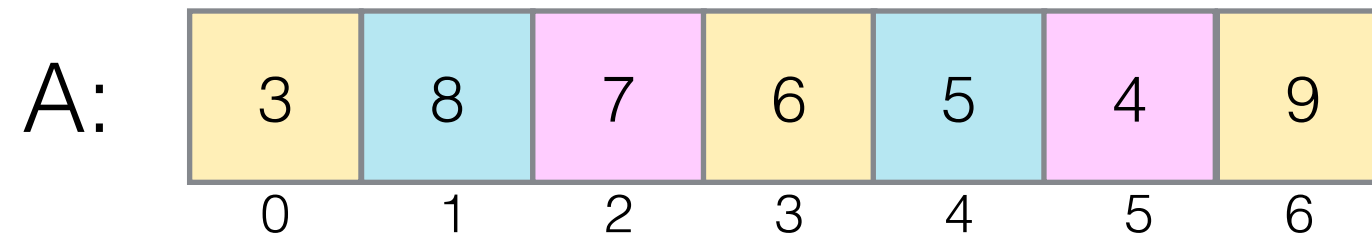
Sort the blue entries



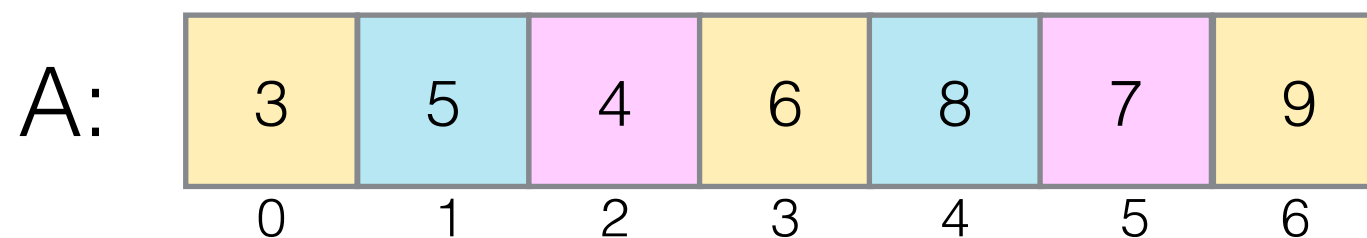
Sort the pink entries

Shellsort: Motivation

Sort the blue entries

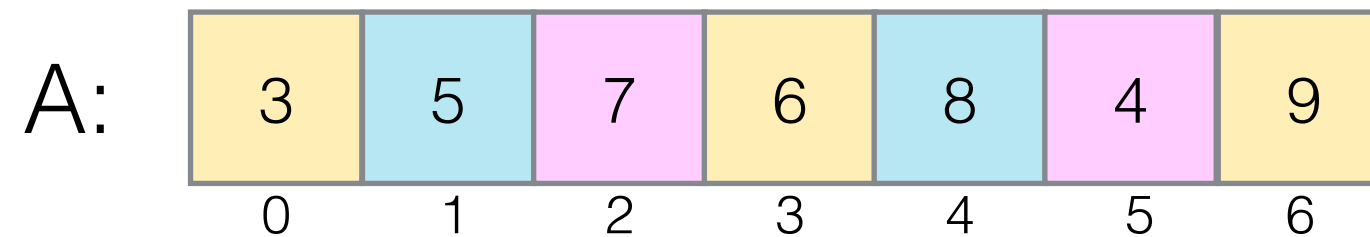
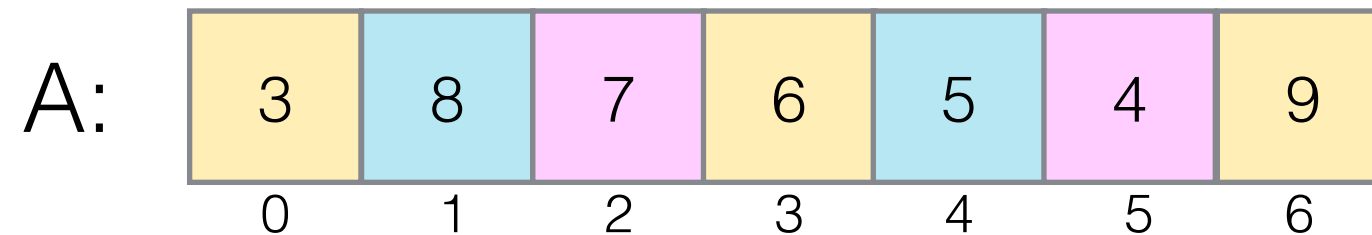


Sort the pink entries

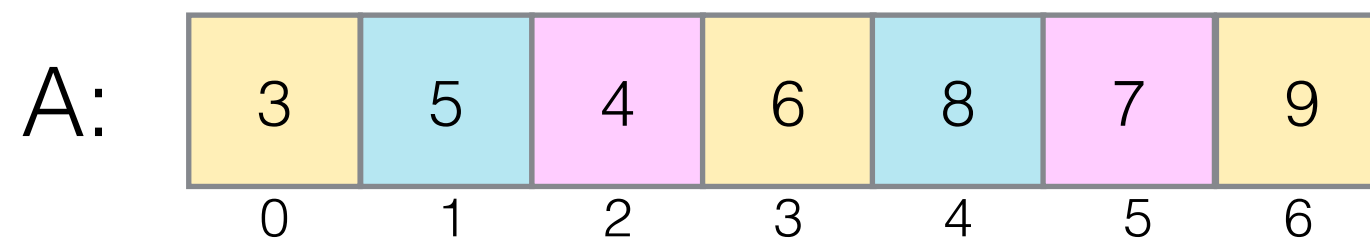


Shellsort: Motivation

Sort the blue entries



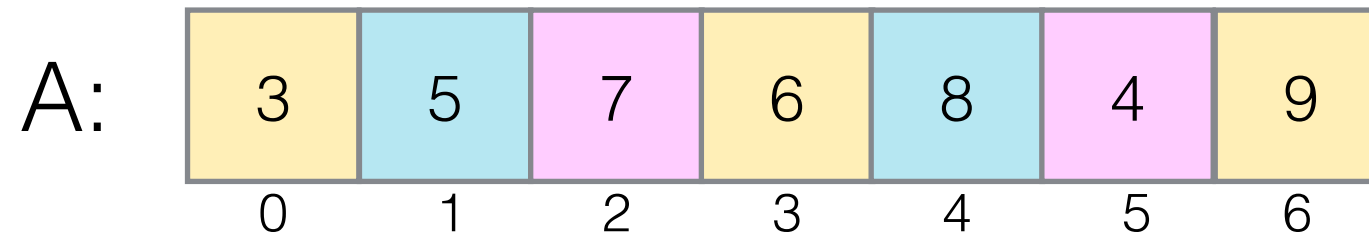
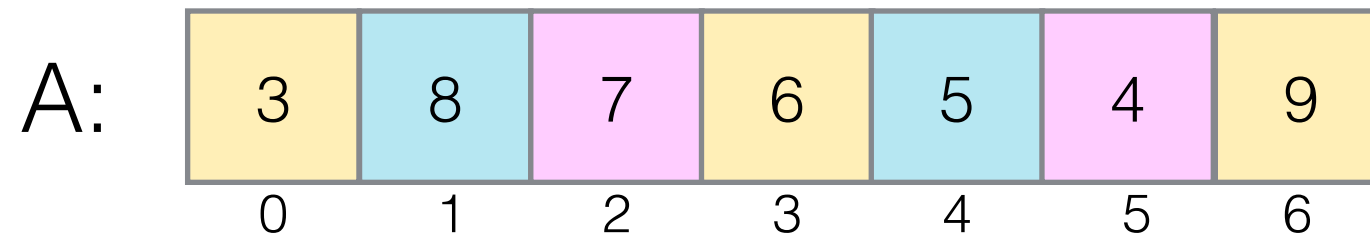
Sort the pink entries



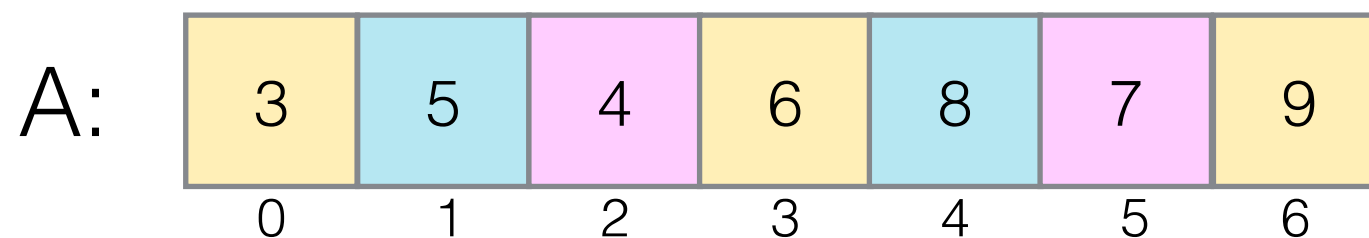
Notice how it
is now **almost
sorted**

Shellsort: Motivation

Sort the blue entries



Sort the pink entries

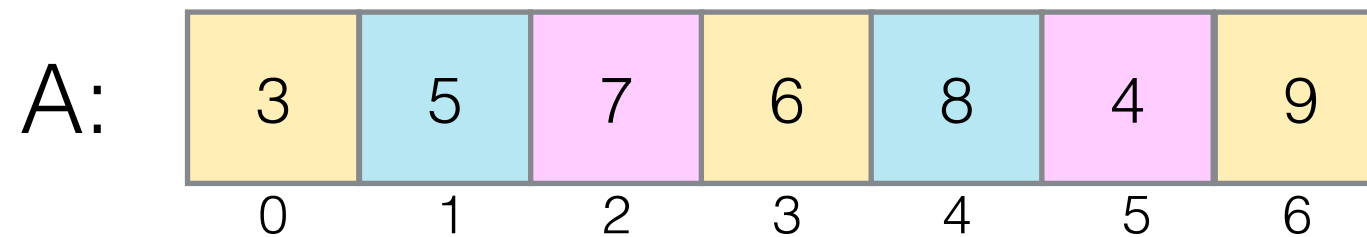
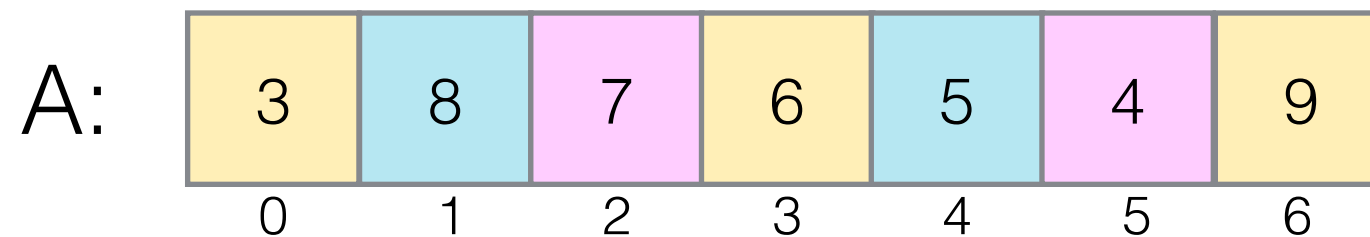


Notice how it
is now **almost
sorted**

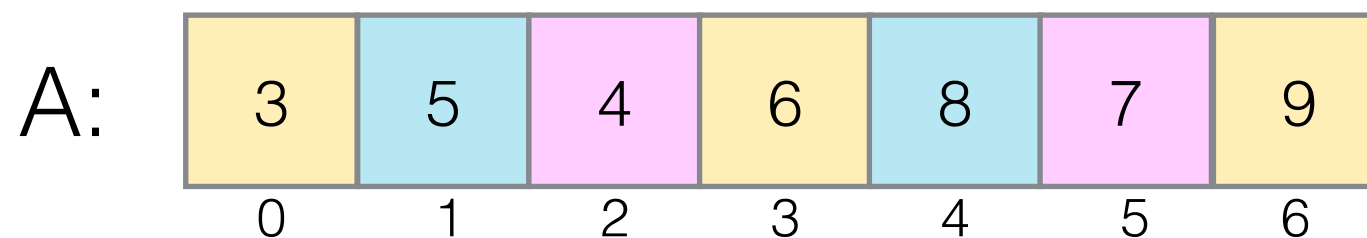
Now do a final round of insertion sort over the entire array

Shellsort: Motivation

Sort the blue entries

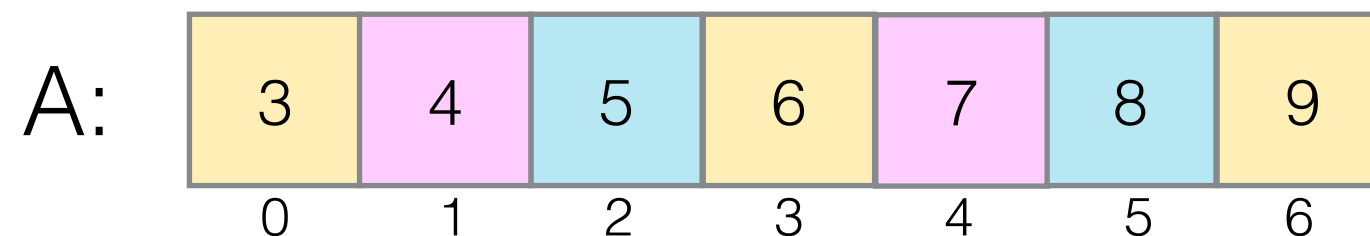


Sort the pink entries



Notice how it
is now **almost
sorted**

Now do a final round of insertion sort over the entire array



Shellsort

- We just did a shellsort for $k=3$
- In general:
 - Think of the array as an interleaving of k lists
 - Sort each list separately using insertion sort
 - Then sort the resulting entire array using a final pass of insertion sort

Shellsort Passes and Gap Sequences

- For large files, start with larger k and then repeat with smaller k s
- It is common to start from somewhere in the sequence 1, 4, 13, 40, 121, 364, 1093, ... and work backwards.
 - what is the sequence?
- For example, for an array of size 20,000, start by 364-sorting, then 121-sort, then 40-sort, and so on.
- Sequences with smaller gaps (a factor of about 2.3) appear to work better, but nobody really understands why.

Properties of Shellsort

- Fewer comparisons than insertion sort. Known to be worst-case $O(n\sqrt{n})$ for good gap sequences.
- Conjectured to be $O(n^{1.25})$ but the algorithm is very hard to analyse.
- Very good on medium-sized arrays (up to size 10,000 or so).
- In-place?
- Stable?

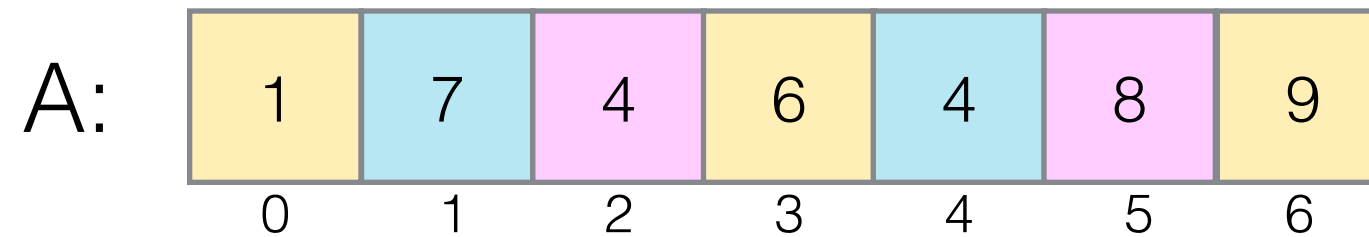
Properties of Shellsort

- Fewer comparisons than insertion sort. Known to be worst-case $O(n\sqrt{n})$ for good gap sequences.
- Conjectured to be $O(n^{1.25})$ but the algorithm is very hard to analyse.
- Very good on medium-sized arrays (up to size 10,000 or so).
- In-place? *yes*
- Stable?

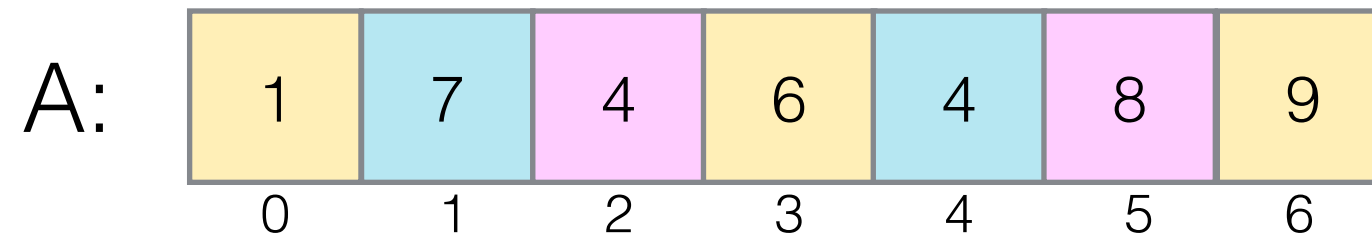
Properties of Shellsort

- Fewer comparisons than insertion sort. Known to be worst-case $O(n\sqrt{n})$ for good gap sequences.
- Conjectured to be $O(n^{1.25})$ but the algorithm is very hard to analyse.
- Very good on medium-sized arrays (up to size 10,000 or so).
- In-place? *yes*
- Stable? *?*

Shellsort: Stability

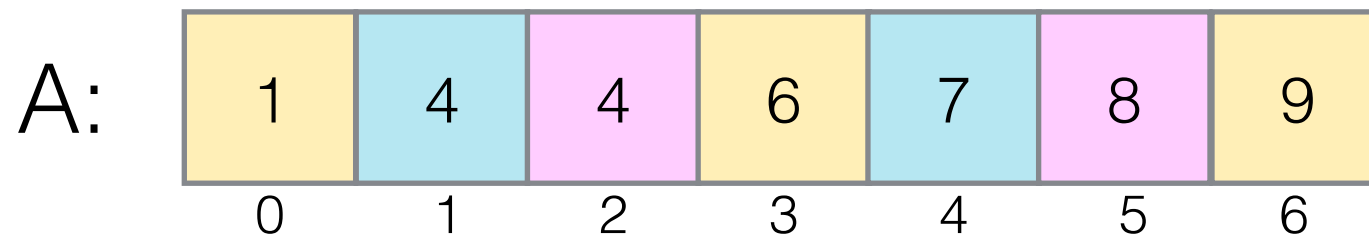
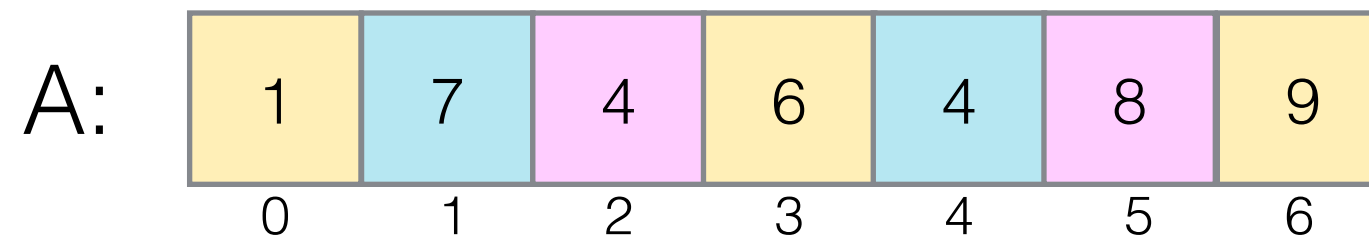


Shellsort: Stability



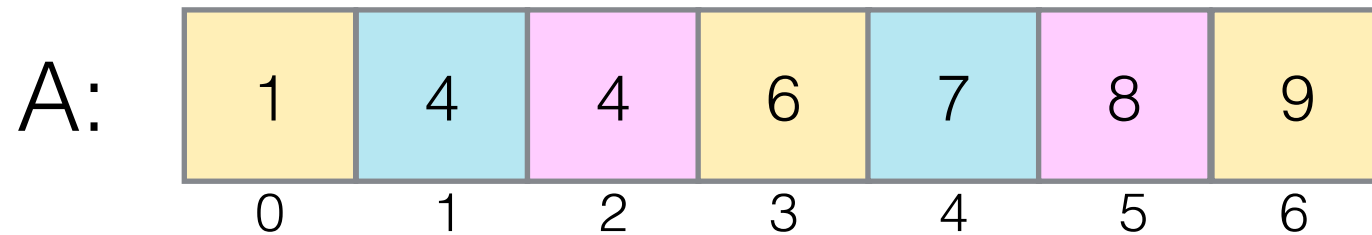
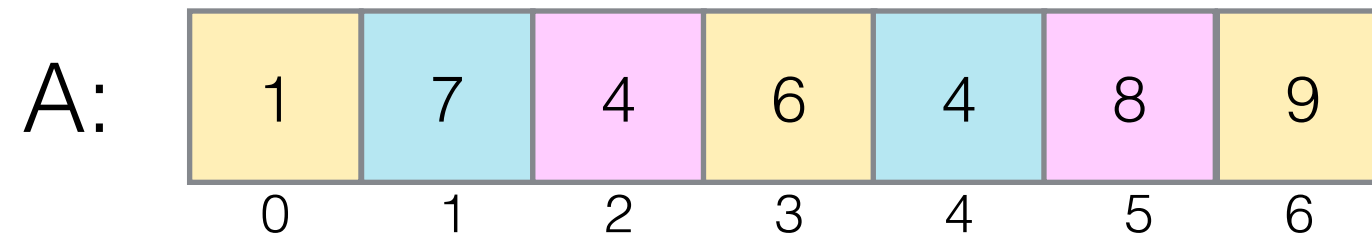
after sorting
the blues

Shellsort: Stability



after sorting
the blues

Shellsort: Stability



after sorting
the blues

relative order of the two 4s has changed!

Properties of Shellsort

- Fewer comparisons than insertion sort. Known to be worst-case $O(n\sqrt{n})$ for good gap sequences.
- Conjectured to be $O(n^{1.25})$ but the algorithm is very hard to analyse.
- Very good on medium-sized arrays (up to size 10,000 or so).
- In-place?
- Stable?

Properties of Shellsort

- Fewer comparisons than insertion sort. Known to be worst-case $O(n\sqrt{n})$ for good gap sequences.
- Conjectured to be $O(n^{1.25})$ but the algorithm is very hard to analyse.
- Very good on medium-sized arrays (up to size 10,000 or so).
- In-place? *yes*
- Stable?

Properties of Shellsort

- Fewer comparisons than insertion sort. Known to be worst-case $O(n\sqrt{n})$ for good gap sequences.
- Conjectured to be $O(n^{1.25})$ but the algorithm is very hard to analyse.
- Very good on medium-sized arrays (up to size 10,000 or so).
- In-place? *yes*
- Stable? *no*

Other Instances of Decrease-and-Conquer by a Constant



THE UNIVERSITY OF
MELBOURNE

- Insertion sort is a simple instance of the “decrease-and-conquer by a constant” approach.
- Another is the approach to topological sorting that repeatedly removes a source.
- In the next lecture we look at examples of “decrease by some factor”, leading to methods with logarithmic time behaviour or better!