Introduction to Algorithms

Meng-Tsung Tsai

09/26/2019

Course Materials

Textbook

Introduction to Algorithms (I2A) 3rd ed. by Cormen, Leiserson, Rivest, and Stein.

Reference Book

Algorithms (JfA) 1st ed. by Erickson. An e-copy can be downloaded from author's website: http://jeffe.cs.illinois.edu/teaching/algorithms/

Websites

http://e3new.nctu.edu.tw for slides, written assignments, and solutions.

http://oj.nctu.me for programming assignments.

Office Hours

Lecturer's

On Wednesdays 16:30 - 17:20 at EC 336 (工程三館).

TA. Erh-Hsuan Lu (呂爾軒) and Tsung-Ta Wu (吳宗達)

On Mondays 10:10 - 11:00 at ES 724 (電資大樓).

TA. Yung-Ping Wang (王詠平) and Chien-An Yu (俞建安)

On Thursdays 11:10 - 12:00 at ES 724 (電資大樓).

Announcements

Programming Assignment 1 is due by Oct 9, 23:59. at https://oj.nctu.me

Written Assignment 1 will be announced tomorrow evening. at https://e3new.nctu.edu.tw

We will not normalize the points that you receive from assignments. 100 points is a perfect score, and extra points are considered as a bonus.

Caution: it is very difficult to solve all problems in an assignment.

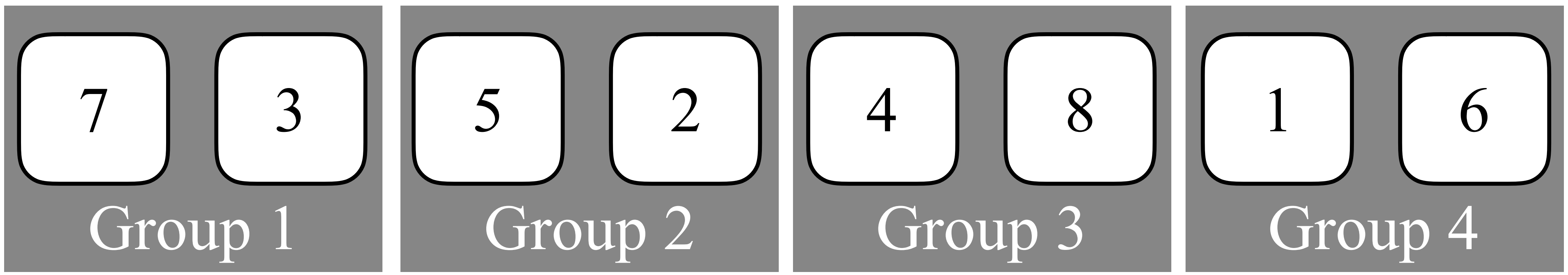
Selection in Linear Time

Warm-up

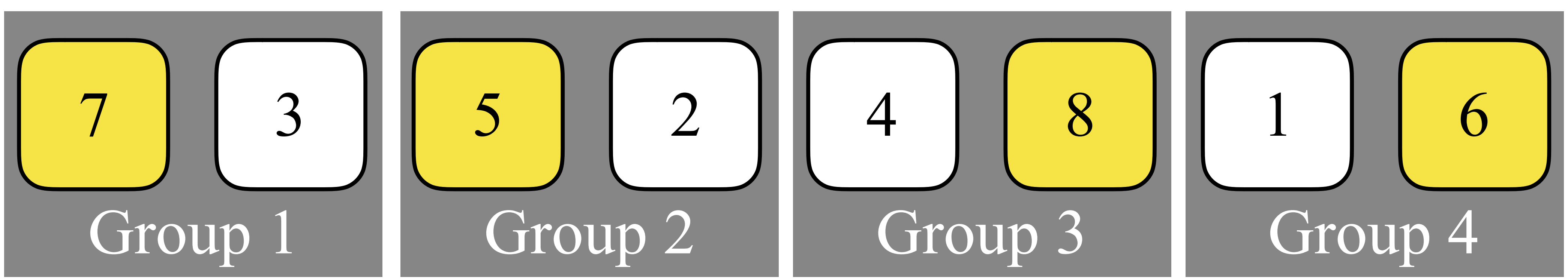
The i-th order statistic of n elements is the i-th smallest one, breaking ties arbitrarily.

# comparisons needed to find x	naively	a clever way
the smallest one	n	-
both the smallest and the largest one	2n	1.5 n

Finding both the first and last order statistics

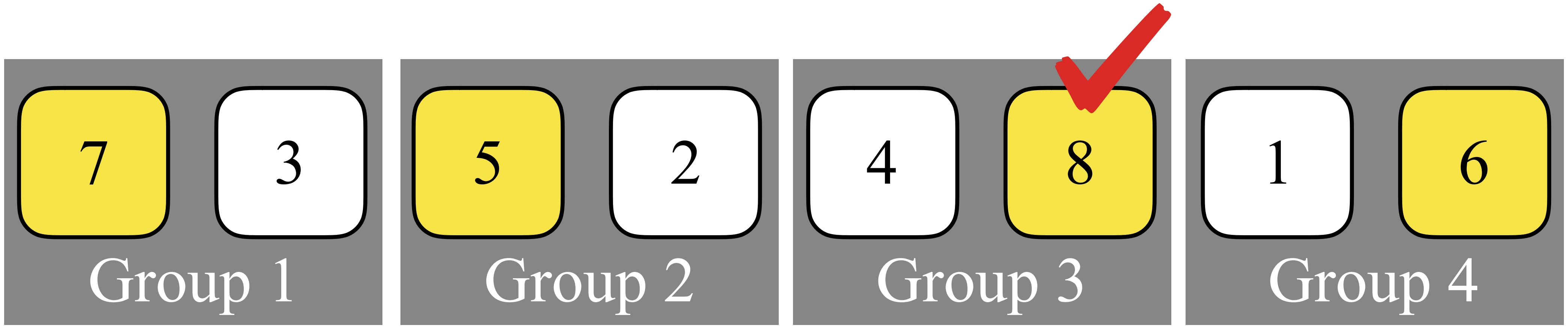


Step 1. Partition the n elements into groups of 2.

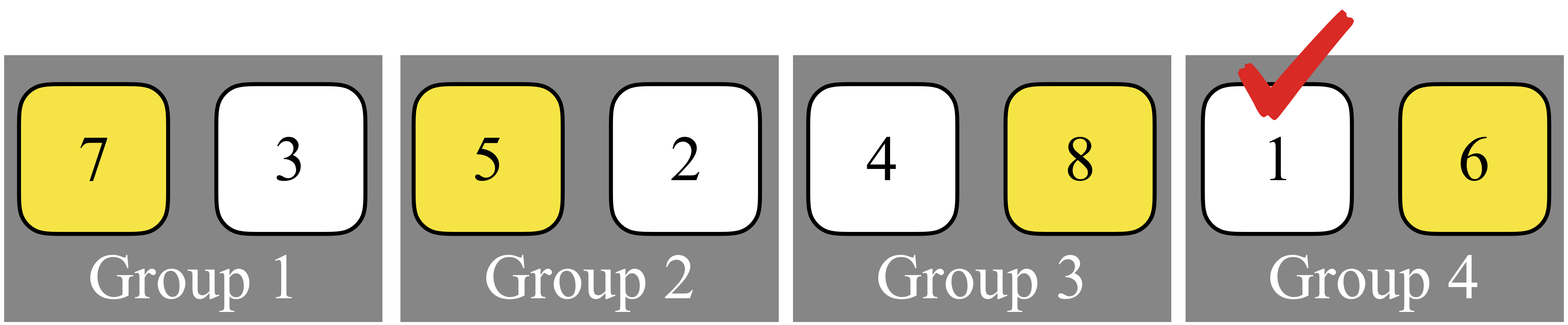


Step 2. Using n/2 comprisons, one can find the local maximum in each group.

Finding both the first and last order statistics



Step 3. One can find the global maximum
among the n/2 local maximum using n/2 comparisons.



Step 4. Analogously, one can find the global minimum
among the n/2 local minimum using n/2 comparisons.

Exercise

What happens if you partition the n elements into groups of 3, rather than groups of 2? Does it use less than 1.5 n comparisons?

Exercise

Devise an algorithm to find the second order statistic using n+O(log n) comparisons.

algorithms to find *x* among n elements	naively	a clever way
the smallest one	n	-
both the smallest and the largest one	2n	1.5n
the second smallest one	2n	n+O(log n)

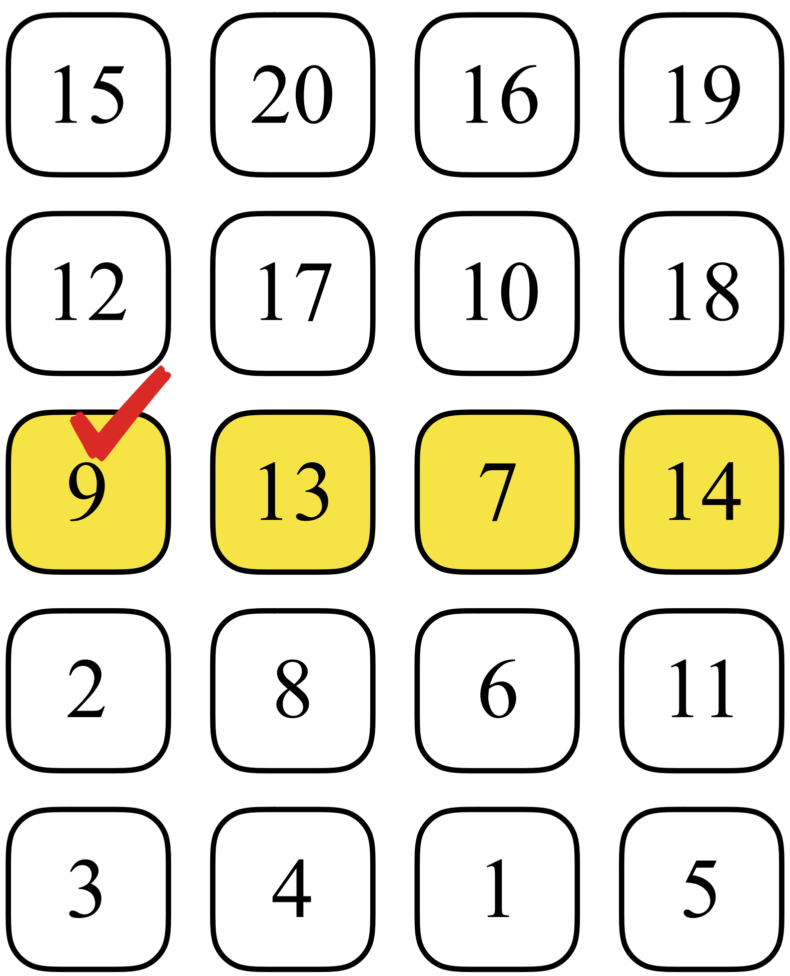
Selection (Deterministic)

Input: an array of n integers, and an index k in [1, n].

Output: the k-th order statistic of the given integers.

Our goal is devising an algorithm to solve this problem in linear time deterministically (i.e. without randomness).

Selection - Finding a Good Pivot



Step 1. Find a pivot p with rank in $(\alpha n, \beta n)$ for some constants $\alpha, \beta \in (0, 1)$.

There are many ways to find such a pivot.

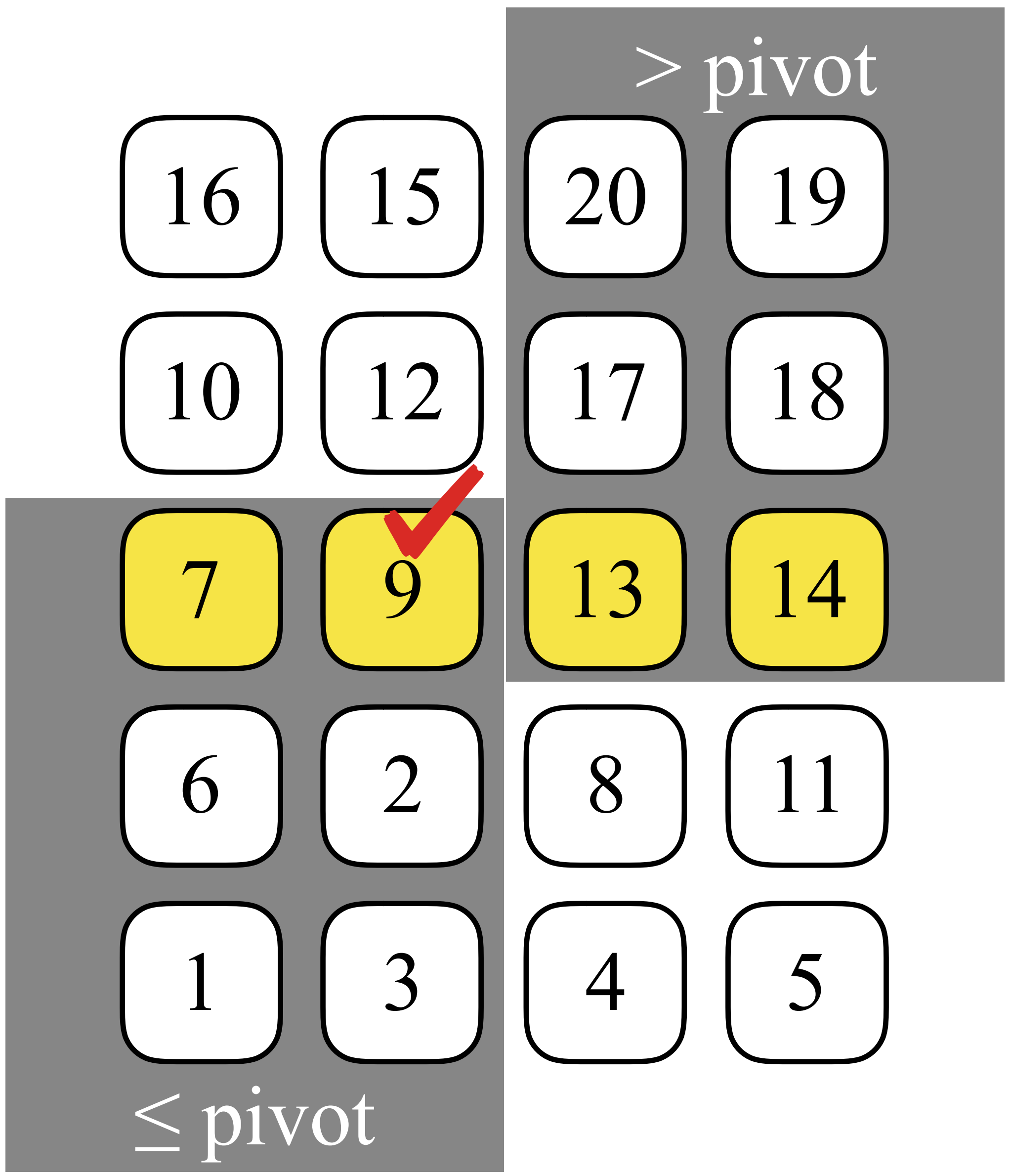
One choice is the median of medians.

Step 1.1: Partition n elements into groups of 5.

Step 1.2: Sort each group to get the median of each group. - O(n) time

Step 1.3: Find the median of the medians. - T(n/5) time

Selection - Finding a Good Pivot

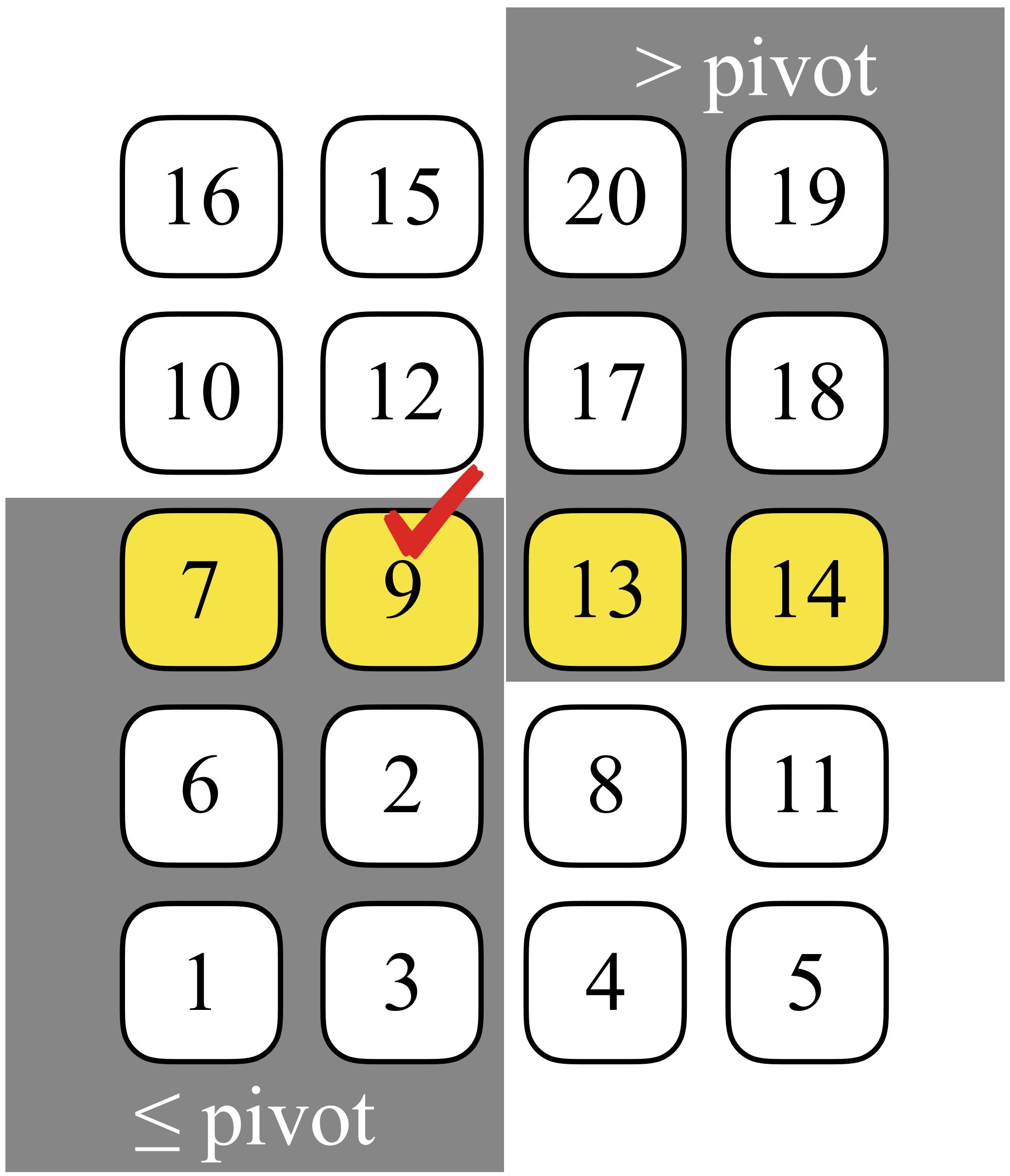


Clearly, the rank of the pivot p is in (0.3n, 0.7n).

We partition the input into S and L where S contains all elements less than or equal to p and L contains the rest. - O(n) time

Note that |S| and |L| both have size at most 7n/10.

Selection - Reduce to a Subproblem




```
if rank(p) == k
    we are done;
else
    if rank(p) < k
        select(L, k-rank(p));
    else // rank(p) > k
        select(S, k);
```

Selection - Running Time

$$T(n) = \begin{cases} T(\lceil n/5 \rceil) + T(\lceil 7n/10 \rceil) + O(n) & \text{if } n \geq 10 \\ O(1) & \text{if } n < 10 \end{cases}$$

Guess T(n) = O(n) and verify the correctness of the substitution method.

--- insight of the guess ---

Problem size decreases geometrically and each takes time linear to the size.

$n + rn + r^2n + \ldots = n/(1-r) = O(n)$ for some r in (0, 1).

Sorting in Linear Time
(Restricted Inputs)

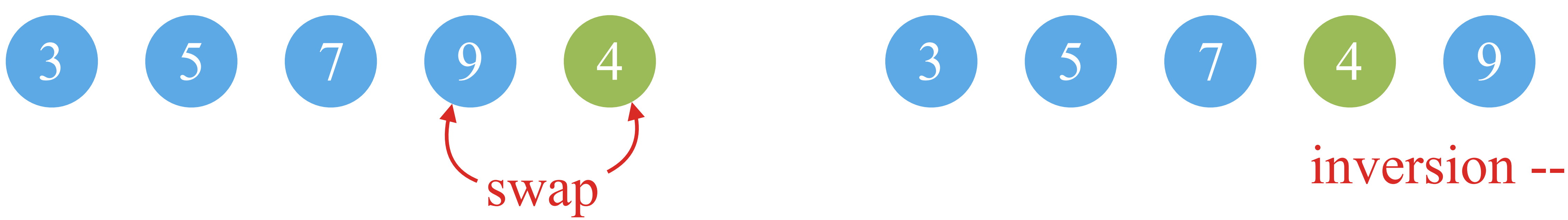
Cases of bounded number of inversions

We say a pair of elements A[i] and A[j] in an array A is an *inversion* if

$$i < j \text{ and } A[i] > A[j].$$

Insertion Sort runs in O(n) time if the number of inversions is O(n).

To see why, every swap performed in Insertion Sort will decrease the number of inversions by 1, and there are O(n) inversions initially.



Exercise

The number of inversions in an array of length n can be calculated in O(n log n) time using a generalized Merge Sort. How?

* Devise an O(n)-time randomized algorithm to approximate the number of inversions. Formally, suppose the number of inversion is *Inv*, then your approximate shall fall within [0.9Inv, 1.1Inv] with probability at least 99%.

Cases of bounded domains (Counting Sort)

Assume that the domain of input is [1, n].

To sort the input, one may simply counting the frequencies of each value as follows:

--- Pseudo Code ---

```
Initialize freq as 0's;

foreach a_i
    freq[a_i] ++;

foreach i in [1, n]
    foreach j in [1, freq[i]]
        print i;
```

Cases of bounded domains (Counting Sort)

Assume that the domain of input is [1, n].

To *stably sort* the input, i.e. breaking ties by indices (if two elements have the same values, place the one that comes earlier earlier).

--- Example ---

Stably sort the students by their age.

23 Alice	25 Bob	23 Carol	24 David

23 Alice	23 Carol	24 David	25 Bob

23 Carol	23 Alice	24 David	25 Bob

Cases of bounded domains (Counting Sort)

23 Alice	25 Bob	23 Carol	24 David

freq	...	23	24	25	...
count	0	2	1	1	0
acc. account	0	2	3	4	4

rank 1	rank 2	rank 3	rank 4

Cases of bounded domains (Counting Sort)

23 Alice	25 Bob	23 Carol

freq	...	23	24	25	...
count	0	2	1	1	0
acc. account	0	2	~~3~~ 2	4	4

rank 1	rank 2	24 David	rank 4

Cases of bounded domains (Counting Sort)

23 Alice | 25 Bob | 23 Carol

freq	...	23	24	25	...
count	0	2	1	1	0
acc. account	0	2	~~3~~ 2	4	4

rank 1 | rank 2 | 24 David | rank 4

Cases of bounded domains (Counting Sort)

23 Alice	25 Bob

freq	...	23	24	25	...
count	0	2	1	1	0
acc. account	0	~~2~~ 1	~~3~~ 2	4	4

rank 1	23 Carol	24 David	rank 4

Cases of bounded domains (Counting Sort)

Assume that the domain of input is [1, n].

To stably sort students by their age,

--- Pseudo Code ---

```
Initialize freq as 0's;

for i = 1 to n
    freq[A[i].age] ++; // freq[k] denotes # students with age k
for i = 2 to n
    freq[i] += freq[i-1]; // freq[k] now denotes # students with age ≤ k

for i = n to 1
    B[freq[A[i].age]--] = A[i]; // the last index i whose A[i].age is k
                                   shall be have rank freq[k]
```

Cases of moderate domains (Radix Sort)

Assume that the domain of input is $[1, n^C]$ for some constant C.

Step 1. Represent each given integer in n-ary.

Step 2. Sort the input by the least significant digit.

Step 3. Stably sort the input by the second least significant digit.

...

Last step. Stably sort the input by the most significant digit.

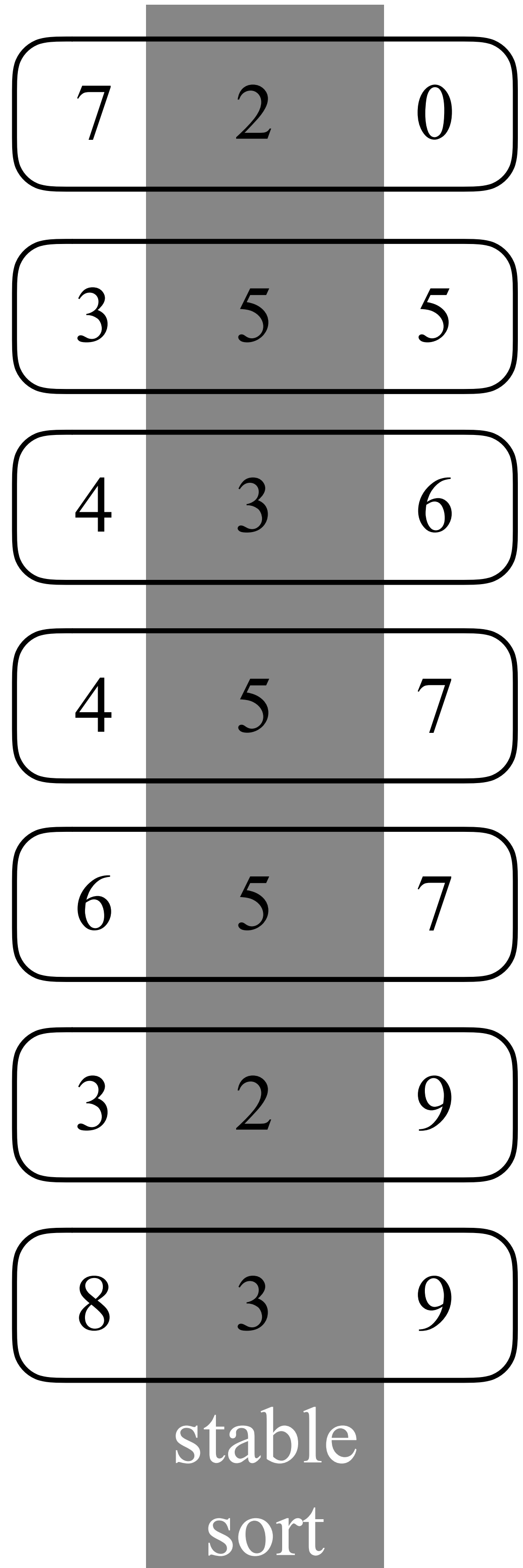
Cases of moderate domains (Radix Sort)

3	2	9
4	5	7
6	5	7
8	3	9
4	3	6
7	2	0
3	5	5
		stable sort

Cases of moderate domains (Radix Sort)

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

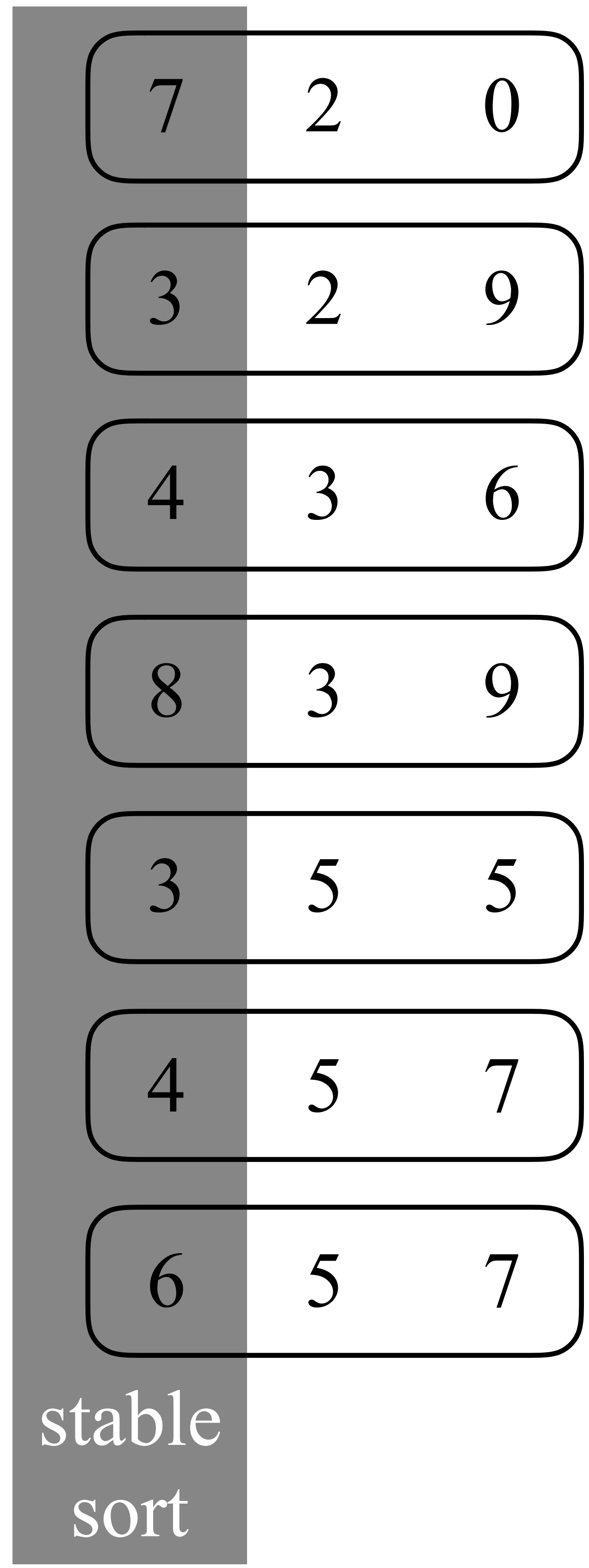
Cases of moderate domains (Radix Sort)



Cases of moderate domains (Radix Sort)

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

Cases of moderate domains (Radix Sort)



Cases of moderate domains (Radix Sort)

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

Exercise

Why happens if we do not use a stable sort for the digits other than the least significant one?

Summary

To sort n integers, each has d digits with value in {0, ..., k-1}.

	Running Time
Counting Sort	$O(n+d^k)$
Radix Sort	$O(d(n+k))$