COMP 6651 / Winter 2022
Dr. B. Jaumard

Lecture 2: Order Statistics

January 14, 2022

Outline

Median and Order Statistics

The ith order statistics of a set of n elements is the ith smallest element.
A median is the "halfway point" of a set.
n odd: median is $(n+1)/2$ th element (and therefore the $(n+1)/2$th order statistics)

n even: lower median is $n/2$ th element,
upper median is $n/2+1$ th element.

Regardless of the parity:
Lower median $\lfloor (n+1)/2 \rfloor$; Upper median $\lceil (n+1)/2 \rceil$

Selection Problem

Input: A set A of n (distinct) elements and a number i, with $1 \leq i \leq n$
Output: The element $x \in A$ that is larger than exactly $i-1$ other elements of A
(= find ith **order statistics** of A)

Complexity of the Selection Problem

Selection Problem

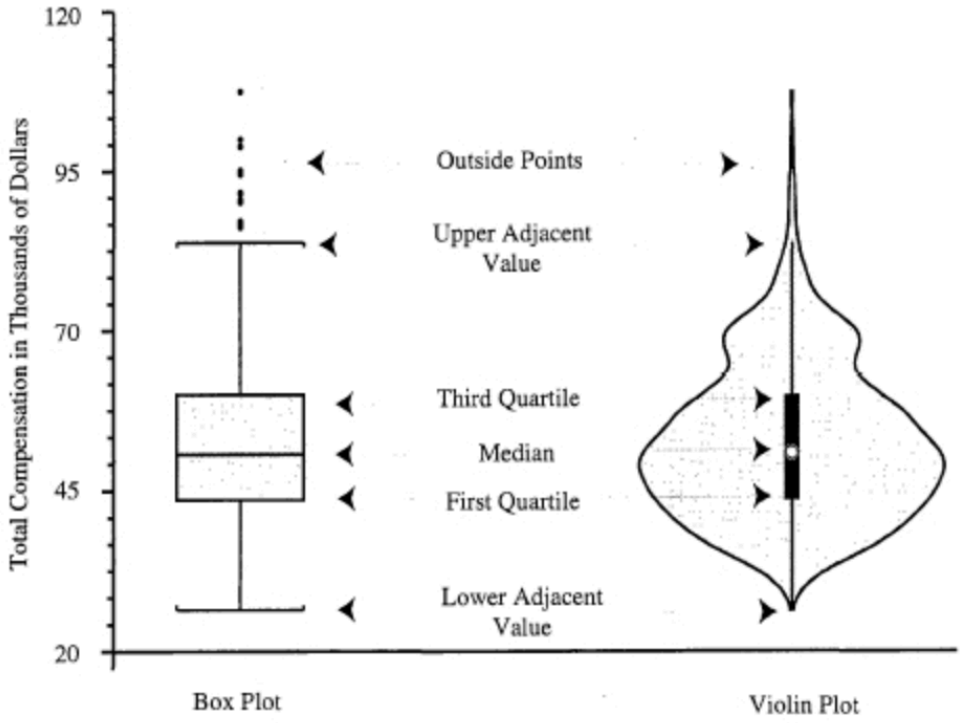
Input: A set A of n (distinct) elements and a number i, with $1 \leq i \leq n$
Output: The element $x \in A$ that is larger than exactly $i-1$ other elements of A
(= find ith **order statistics** of A)

We will go through three complexity analysis:

- First analysis. Worst case (see the complexity of sorting algorithms): $O(n \log n)$
- Second analysis. Average case: $O(n)$
- Third analysis. Worst case: $O(n)$

Common Metrics in Machine Learning

A Violin plot is an abstract representation of the probability distribution of the sample. Violin plots use kernel density estimation (KDE) to compute an empirical distribution of the sample.



Source: Jerry L. Hintze and Ray D. Nelson, Violin Plots: A Box Plot-Density Trace Synergism, *The American*

Second analysis: Selection in expected (or average) linear time

- Finding the minimum or the maximum of a set of n elements: $n - 1$ comparisons
- Simultaneous minimum and maximum: $\Theta(n)$
- Selection problem is more difficult ...

Divide-and-conquer algorithm RANDOMIZED-SELECT modeled after the QUICKSORT algorithm.

- Pick one element x of the array A and partition A into two sub-arrays:
  - $A[0], A[1], ..., A[q-1]$ contains elements $\leq A[q] = x$,
  - $A[q+1], A[q+2], ..., A[n]$ contains elements $> A[q] = x$.
- If $q = i$ then we are done, else select an appropriate element in **one of the two** sub-arrays.

Partitioning an array

For a given subarray $A = [p..r]$ with n elements

PARTITION(A, p, r)

```
// use A[r] as the pivot for partitioning
// returns location of pivot after partitioning
x ⟵ A[r];
i ⟵ p − 1;
for j ⟵ p to r − 1
   do if A[j] ≤ x
      then  i ⟵ i + 1
            exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i+1;
```

Run-time of PARTITION

$O(n)$

Illustration of PARTITION (1/2)

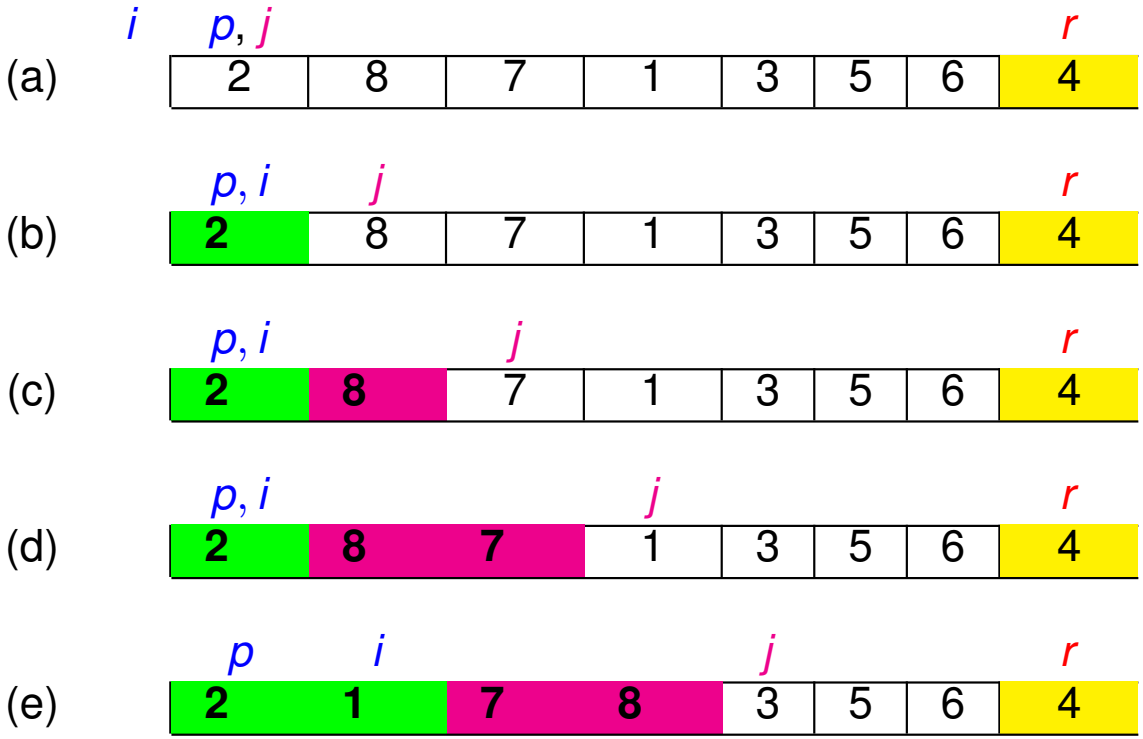


Illustration of PARTITION (2/2)

Quicksort: The divide-and-conquer paradigm (1/3)

For a given subarray $[p..r]$

- **Divide.** Partition (rearrange) the array $A[p..r]$ into two (possibly empty) subarrays $A[p...q-1]$ and $A[q+1..r]$ such that each element of $A[p...q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1..r]$. Compute the index q as part of this partitioning procedure.
- **Conquer.** Sort the two subarrays $A[p...q-1]$ and $A[q+1..r]$ by recursive calls to quicksort.
- **Combine.** Since the subarrays are sorted in place, no work is needed to combine them: the entire array $A[p..r]$ is now sorted.

Quicksort: The divide-and-conquer paradigm (2/3)

For a given subarray $[p..r]$ with n elements

QUICKSORT(A, p, r)

if $p < r$
  then
    $q \leftarrow$ PARTITION(A, p, r)
    QUICKSORT$(A, p, q-1)$
    QUICKSORT$(A, q+1, r)$

- Worst Case Analysis: $O(n^2)$
- Expected Running Time: $O(n \log n)$
- Heapsort and Mergesort: $O(n \log n)$ (worst and average cases)
- Bubblesort: $O(n^2)$ (worst and average cases)

Quicksort: The divide-and-conquer paradigm (3/3)

- Worst Case Analysis: $T(n) = T(n-k) + T(k) + \Theta(n)$
- Worst case: when the pivot happened to be the least element of the array

$$T(n) = T(n-1) + T(1) + \Theta(n)$$

- General solution of $T(n) = T(n-1)$: $T(n) = A \times 1^n$
- Particular solution of $T(n) = T(n-1) + T(1) + \Theta(n)$: $T(n) = A_1 n^2 + B_1 n + C_1$
- Solution of $T(n) = T(n-1) + T(1) + \Theta(n)$: $T(n) = O(n^2)$

Randomized Partition Algorithm

For a given subarray $A = [p..r]$

RANDOMIZE-PARTITION(A, p, r)

// a random element of A is used as the pivot
// returns location of this pivot after partitioning
$i \longleftarrow$ Random(p, r);
exchange $A[r] \leftrightarrow A[i]$;
return PARTITION(A, p, r);

Randomized Select Algorithm (1/2)

For a given array $A = [1..n]$ (and subarray $[p..r]$)

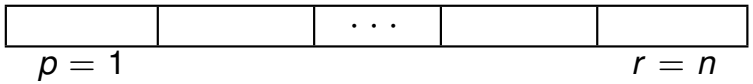
RANDOMIZED-SELECT(A, p, r, i)

// find i^{th} smallest element in $A[p..r]$
IF $p = r$ RETURN $A[p]$;
$q \longleftarrow$ RANDOMIZE-PARTITION(A, p, r); $k \longleftarrow q - p + 1$;
IF $i = k$
// we have found the k-th smallest element
THEN RETURN $A[q]$;
ELSEIF $i < k$
THEN RANDOMIZE-SELECT$(A, p, q - 1, i)$;
ELSE RETURN RANDOMIZE-SELECT$(A, q + 1, r, i - k)$;

Initially, call RANDOMIZE-SELECT$(A, 1, n, i)$

Randomized Select Algorithm (2/2)

Initial Call:



$q_1 \leftarrow$ pivot position at the end of RANDOMIZE-PARTITION$(A, 1, n, i)$

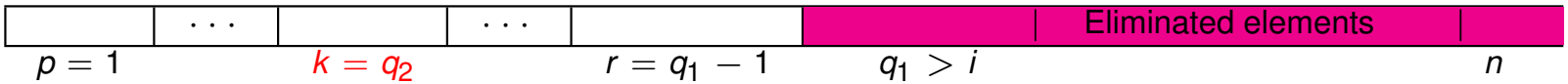
$k \leftarrow q_1 - p + 1 = q_1 - 1 + 1 = q_1$

We have found for the kth = q_1th order in the interval $[p = 1,\ r = n]$

Case 1: $i = k \rightarrow$ Easy

Case 2: $i < k$

$q_2 \leftarrow$ pivot position at the end of RANDOMIZE-PARTITION$(A, 1, q_1 - 1, i)$

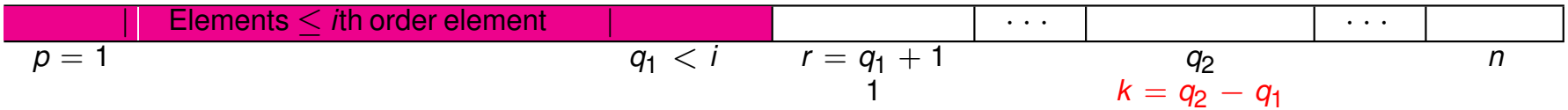


We are looking for the kth = q_2th order in the interval

$[p = 1,\ r = q_1 - 1] \quad \rightarrow \quad k \leftarrow q_2 - p + 1 = q_2 - 1 + 1 = q_2$

Case 3: $i > k$

$q_2 \leftarrow$ pivot position at the end of RANDOMIZE-PARTITION$(A, q_1 + 1, n, i - q_1)$



After a recursive call, as a result of the PARTITION algorithm, we will find the kth = $(q_2 - q_1)$th order in the interval

$[p = q_1 + 1,\ r = n] \quad \rightarrow \quad k \leftarrow q_2 - q_1 - 1 + 1 = q_2 - q_1$

Expected value of a random variable (1/2)

- A (discrete) random variable X is a function from a finite or countably infinite sample space S to the real numbers.
- Random variable X, real number x.
- Event $X = x$ is defined by: $\{s \in S : X(s) = x\}$
- $Pr[X = x] = \sum_{\{s \in S : X(s) = x\}} Pr\{s\}$
- Probability axioms entail:

$$Pr[X = x] \geq 0 \quad \text{and} \quad \sum_x Pr[X = x] = 1.$$

- Expected value (or expectation or mean) of a discrete random variable X:

$$E[X] = \sum_x x \, Pr\{X = x\}$$

Expected value of a random variable (2/2)

- **Linearity of Expectation**. The expectation of the sum of two random variables is the sum of their expectations:

$$E[X + Y] = E[X] + E[Y].$$

- $E[a\,X] = a\,E[X]$
- When n random variables $X_1, X_2, \ldots, X_n$ are **mutually independent**

$$E[X_1 X_2 \ldots X_n] = E[X_1] \times E[X_2] \times \cdots \times E[X_n]$$

Complexity of RANDOMIZED-SELECT (1/7)

The worst-case analysis of **Randomize-Select** is $\Theta(n^2)$

Theorem

*The expected run-time $T(n)$ of **Randomize-Select** is linear, i.e., $T(n) = \Theta(n)$*

Time required by RANDOMIZED-SELECT on an input array $A[p..r]$ of n elements is a random variable $T(n)$.

Aim: Compute an upper bound $\mathbf{E}[T(n)]$ on $T(n)$.

Complexity of RANDOMIZED-SELECT (2/7)

- RANDOMIZE-PARTITION returns any value with the same probability as the pivot.
- For any k, $1 \leq k \leq n$, the sub-array $A[p..q]$ has k elements (all less than or equal to the pivot) with probability $1/n$.
- Indicator random variable

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split} \\ 0 & \text{otherwise} \end{cases}$$

- Expected value: $E[X_k] = 1/n$
- When $X_k = 1$ ($X_{k'} = 0$ for all $k' \neq k$), we recurse on sub-arrays of size $k-1$ or $n-k$. Worst case: Take the largest one.

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot \left(T(\max\{k-1, n-k\}) + O(n) \right)$$

Complexity of RANDOMIZED-SELECT (3/7)

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot \Big(T(\max\{k-1, n-k\}) + O(n) \Big)$$

$$= \sum_{k=1}^{n} \Big(X_k \cdot T(\max\{k-1, n-k\}) \Big) + O(n)$$

$$\mathbf{E}[T(n)] \leq \mathbf{E}\left[\sum_{k=1}^{n} X_k \cdot T(\max\{k-1, n-k\}) + O(n)\right]$$

$$\leq \sum_{k=1}^{n} \mathbf{E}\left[X_k \cdot T(\max\{k-1, n-k\})\right] + O(n) \qquad \text{Linearity of expectation}$$

$$\leq \sum_{k=1}^{n} \mathbf{E}[X_k] \cdot \mathbf{E}\left[T(\max\{k-1, n-k\})\right] + O(n) \qquad E[XY] = E[X] \cdot E[Y]$$

Home exercise: Show that X_k and $T(\max\{k-1, n-k\})$ are independent random variables

Complexity of RANDOMIZED-SELECT (4/7)

$$\begin{aligned}\mathbf{E}[T(n)] &\leq \sum_{k=1}^{n} \mathbf{E}[X_k] \cdot \mathbf{E}[T(\max\{k-1, n-k\})] + O(n) \\ &\leq \sum_{k=1}^{n} \frac{1}{n} \cdot \mathbf{E}[T(\max\{k-1, n-k\})] + O(n) \qquad \text{recall that } E[X_k] = 1/n\end{aligned}$$

Observe that:

$$\max\{k-1, n-k\} = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil \\ n-k & \text{if } k \leq \lfloor n/2 \rfloor \end{cases}$$

- n is even: Each term from $T(\lceil n/2 \rceil)$ up to $T(n-1)$ appears exactly twice in the summation
- n is odd: All these terms appear twice and $T(\lfloor n/2 \rfloor)$ appears once

$$\mathbf{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbf{E}[T(k)] + O(n)$$

Complexity of RANDOMIZED-SELECT (5/7)

$$\mathbf{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathbf{E}[T(k)] + O(n)$$

- We solve the recurrence by substitution, this involves an induction proof.
- Hypothesis (Strong Principle of Mathematical Induction): Assume $E[T(k)] \leq ck$ for some constant c that satisfies the initial conditions of the recurrence, $k < n$.
- Base case: $T(n) = O(1)$ for n less than some constant; we shall pick this constant later.
- Let a be a constant such that the function described by $O(n)$ is bounded from above by an for all $n \geq 0$.

$$\mathbf{E}[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an$$

Complexity of RANDOMIZED-SELECT (6/7)

$$
\begin{aligned}
\mathbf{E}[T(n)] \leq & \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an \\
\leq & \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\
\leq & \frac{2c}{n} \left(n(n-1)/2 - \lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)/2 \right) + an \\
\leq & \frac{2c}{n} \left((n-1)n/2 - (n/2-1)(n/2-2)/2 \right) + an \\
\leq & \frac{c}{n} \left(3n^2/4 + n/2 - 2 \right) + an \\
\leq & \; 3cn/4 + c/2 - 2c/n + an \leq cn - (cn/4 - c/2 - an)
\end{aligned}
$$

Complexity of RANDOMIZED-SELECT (7/7)

$$\mathbf{E}[T(n)] \leq cn - (cn/4 - c/2 - an)$$

We need $(cn/4 - c/2 - an) \geq 0$ for sufficiently large n, i.e., $n(c/4 - a) \geq c/2$ for sufficiently large n.

Choose $c > 4a$, and we have $n \geq 2c/(c - 4a)$.

All is fine if $T(n) = O(1)$ for small values of n.

Third analysis: Selection in worst-case linear time

Can we make the Selection algorithm to be linear in the worst case?

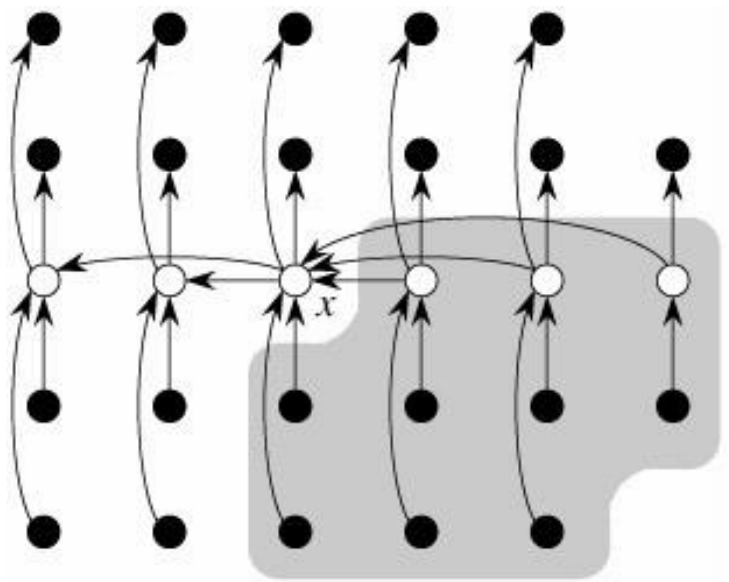
Idea: Make the split "good" in all cases by using all the time a "good" pivot.

Algorithm SELECT(A, i) with A a set of n elements ith order statistics

Step 1. Divide n elements into $\lfloor n/5 \rfloor$ groups of 5 elements. Note that one group may have less than 5 elements.

Step 2. Find the median of each group by first insertion sorting the elements of each group, and then picking the median from the sorted list of group elements. $M \leftarrow$ set of medians.

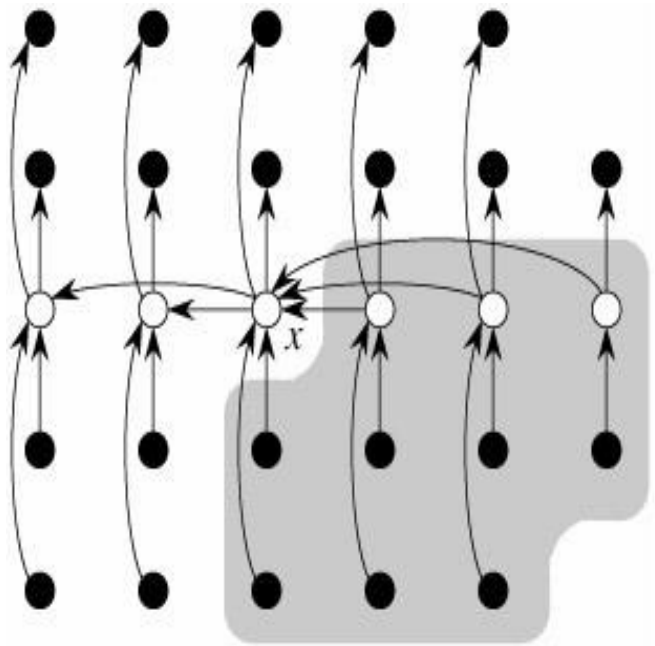
Step 3. Use SELECT recursively, i.e., SELECT($M, \lfloor \frac{\lceil n/5 \rceil}{2} \rfloor$), to find the median x of the $\lceil n/5 \rceil$ medians found in Step 2.



Algorithm SELECT(A, i) - ith order statistics

Step 4. Partition the input array A around the median-of-medians x using PARTITION (Slide 7). Let i be one more than the number of elements on the low side of the partition, so that x is the ith smallest element and there are $n - i$ elements on the high side of the partition.

Step 5. If $i = k$, then return x.
Otherwise, use SELECT recursively to find the ith smallest element on the low side if $i < k$
i.e., SELECT($A[1..k-1], i$)),
or the $(i - k)$th smallest elements on the high side if $i > k$,
i.e., SELECT($A[k+1..n], i-k$)).



Complexity Analysis of the SELECT Algorithm

$T(n)$ = complexity of SELECT Algorithm when there are n elements

Step 1. $O(n)$.

Step 2. $O(n)$ as computing the median of each group can be done in $O(1)$.

Step 3. $T(\lceil n/5 \rceil)$ assuming $T(n)$ denotes the running time of SELECT(n), i.e., of determining the ith smallest of an input array of $n > 1$ elements.

Step 4. $O(n)$ using the PARTITION algorithm described on Slide 8.

Step 5. $\leq T(7n/10 + 6)$, see the justification in the following slides.

Overall complexity:

$$\leq \underbrace{O(n)}_{\text{Step 1}} + \underbrace{O(n)}_{\text{Step 2}} + \underbrace{T(\lceil n/5 \rceil)}_{\text{Step 3}} + \underbrace{O(n)}_{\text{Step 4}} + \underbrace{T(7n/10 + 6)}_{\text{Step 5}}$$

How good is the partition around x?

At least half of the medians $\geq x$

$\Rightarrow$ At least half of $\lceil n/5 \rceil$ groups contribute 3 elements $\geq x$, except for 2 groups (last group + group containing x)

$\Rightarrow$ $3\left(\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil - 2\right) \geq \frac{3n}{10} - 6$ elements $\geq x$

$\Rightarrow$ SELECT called recursively on at most $\frac{7n}{10} + 6$ elements.

$$T(n) \leq \Theta(1) \qquad \text{if } n \leq n_0$$
$$T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) \qquad \text{if } n > n_0$$

Induction proof: $T(n) \leq cn$.

Base case: ...
Induction hypothesis: $\forall k : n_0 \leq k \leq n-1 \quad T(k) \leq ck$
Induction step

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(\frac{7n}{10} + 6) + an \\ &\leq c(n/5+1) + \frac{7nc}{10} + 6c + an \\ &= \frac{9nc}{10} + 7c + an \\ &= cn + (-cn/10 + 7c + an) \end{aligned}$$

Look for n such that: $-cn/10 + 7c + an \leq 0$

$c \geq \frac{10an}{n-70}$ if $n > 70$

$n \geq 140 \Rightarrow \frac{n}{n-70} \leq 2 \Rightarrow c \geq 20a$

Example (1/8)

Let us consider a set of $n = 43$ distinct integers.
Step 1. Divide n elements into $\lceil n/5 \rceil = 9$ groups, 8 groups of 5 elements, and one group with 3 elements.

Step 2. Find the median of each group by first insertion sorting the elements of each group, and then picking the median from the sorted list of group elements.

Example (2/8)

Step 2. Find the median of each group by sorting the elements of each group, and then picking the median from the sorted list of group elements.

17	110	112	199	149	19	13	49	112	111	...	...

Another way to represent the array

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9
17	19	...	...	...	...	...	...	
110	13	...	...	...	...	...	...	7
112	49	...	...	...	...	...	...	21
199	112	...	...	...	...	...	...	33
149	111	...	...	...	...	...	...	

Example (3/8)

Step 2. Find the median of each group by first sorting the elements of each group, and then picking the median from the sorted list of group elements. Median are located in the red positions of the array.

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9
17	19	...	...	...	...	...	...	
110	13	...	...	...	...	...	...	7
112	49	70	210	24	9	141	125	21
199	112	...	...	...	...	...	...	33
149	111	...	...	...	...	...	...	

Example (4/8)

Step 3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil = 9$ medians found in Step 2.

Set of medians:

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9
112	49	70	210	24	9	141	125	21

Median value: 70

Example (5/8)

Step 4. Partition the input array around the median-of-medians x using the modified version of PARTITION. Let k be one more than the number of elements on the low side of the partition, so that x is the kth smallest element and there are $n - k$ elements on the high side of the partition.

Let us apply the PARTITION procedure on the PIVOT = 70. We obtain:

Group 1'	Group 2'	Group 3'	Group 4'	Group 5'	Group 6'	Group 7'	Group 8'	Group 9'
...	...	...	...	...	...	...	...	...
...	...	...	70	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Note the groups are most probably completely different from what they were before, as the PARTITION algorithm is applied on the "linear" representation of the array, see the next slide.

Example (6/8)

Let us assume we had the following array after Step 3.

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8	Group 9
17	19	35	123	7	2	111	92	
110	13	23	119	3	5	88	77	7
112	49	70	210	24	9	141	125	21
199	112	71	234	81	81	145	130	33
149	111	82	215	168	78	168	160	

Example (7/8)

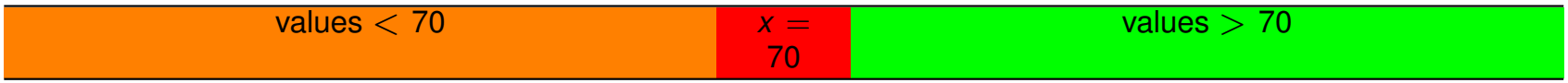
The array will look like as below about running PARTITION.

Group 1'	Group 2'	Group 3'	Group 4'	Group 5'	Group 6'	Group 7'	Group 8'	Group 9'
17	23	2	70	199	149	111	92	71
19	33	5	119	112	110	88	77	82
13	7	9	210	111	112	141	125	123
49	3	7	234	81	81	145	130	
35	24	21	215	168	78	168	160	

Example (8/8)

One more way to view the output of Step 3

We next apply the PARTITION algorithm using $x = 70$ as the pivot element. We get an array that can be represented as follows.



According to the comparison of i and k, we call the SELECT procedure recursively on either the left or the right part of the array produced as an output of the PARTITION algorithm using $x = 70$ as the pivot element

Insertion Sort (1/3)

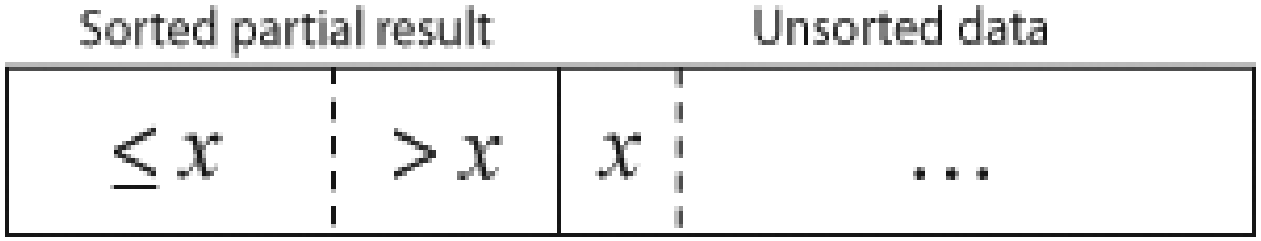
Insertion sort is a simple sorting algorithm, i.e., a comparison sort in which the sorted array (or list) is built one entry at a time.

Every iteration of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list, until no input elements remain. The choice of which element to remove from the input is arbitrary, and can be made using almost any choice algorithm.

Sorting is typically done in-place. The resulting array after k iterations has the property where the first $k + 1$ entries are sorted. In each iteration the first remaining entry of the input is removed, inserted into the result at the correct position, thus extending the result:

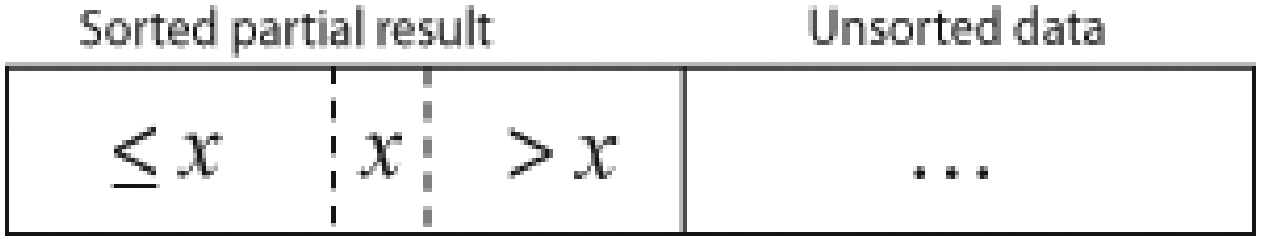
Insertion Sort (2/3)

Array prior to the insertion of x



becomes

Array after the insertion of x



with each element greater than x copied to the right as it is compared against x.

Insertion Sort (3/3)

INSERTIONSORT(array A)

```
for i = 1 to length[A] − 1 do
    value ← A[i]
    j ← i − 1
    while A[j] > value do
        A[j + 1] ← A[j]
        j ← j − 1
        if j < 0 then exit while
    endwhile
    A[j + 1] ← value
endfor
```

Worst case performance $O(n^2)$
Best case performance $O(n)$
Average case performance $O(n^2)$

- **Readings.** Chapter 9 (Sections 9.2 and 9.3) of Goodrich and Tamassia (2015).
- **Readings.** Chapter II 9 of Cormen *et al.*, 3rd edition (2009).
- **Suggested Exercises in the Coursepack.** 5.1, 5.2, 5.3.