Outline

- **Minimum and maximum**
- Selection in expected linear time
- Selection in worst-case linear time
Order statistics

- The *ith order statistic* of a set of *n* elements is the *ith* smallest element.
- The **minimum** is the first order statistic (*i* = 1).
- The **maximum** is the *n*th order statistic (*i* = *n*).
- A **median** is the “halfway point” of the set.
- When *n* is odd, the median is unique, at *i* = (*n* + 1)/2.
- When *n* is even, there are two medians:
  - The **lower median**: *i* = \( \lfloor (n + 1)/2 \rfloor \), and
  - The **upper median**: *i* = \( \lceil (n + 1)/2 \rceil \).
  - We mean lower median when we use the phrase “the median”.

3
The selection problem

- How can we find the $i^{th}$ order statistic of a set and what is the running time?

- **Input:** A set $A$ of $n$ (distinct) number 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 $A$.

- The selection problem can be solved in $O(n \log n)$ time.
  - Sort the numbers using an $O(n \log n)$-time algorithm, such as heapsort or merge sort.
  - Then return the $i^{th}$ element in the sorted array.

- Are there faster algorithms?
  - An $O(n)$-time algorithm would be presented in this chapter.
Finding minimum

- We can easily obtain an upper bound of \( n - 1 \) comparisons for finding the minimum of a set of \( n \) elements.
  - Examine each element in turn and keep track of the smallest one.
  - The algorithm is optimal, because each element, except the minimum, must be compared to a smaller element at least once.

\[ \text{MINIMUM}(A) \]
1. \( \text{min} \leftarrow A[1] \)
2. \( \text{for } i \leftarrow 2 \text{ to } \text{length}[A] \)
3. \( \quad \text{do if } \text{min} > A[i] \)
4. \( \quad \text{then } \text{min} \leftarrow A[i] \)
5. \( \quad \text{return } \text{min} \)

- The maximum can be found in exactly the same way by replacing the > with < in the above algorithm.
Some applications need both the minimum and maximum.

- Find the minimum and maximum independently, using \( n-1 \) comparisons for each, for a total of \( 2n-2 \) comparisons.

In fact, at most \( 3 \left\lfloor \frac{n}{2} \right\rfloor \) comparisons are needed:

- Maintain the minimum and maximum of elements seen so far.
- Process elements in pairs.
- Compare the elements of a pair to each other.
- Then compare the larger element to the maximum so far, and compare the smaller element to the minimum so far.

This leads to only 3 comparisons for every 2 elements.
Simultaneous minimum and maximum\textsuperscript{2/3}

- An observation
  - If we compare the elements of a pair to each other, the larger can’t be the minimum and the smaller can’t be the maximum.
  - So we just need to compare the larger to the current maximum and the smaller to the current minimum.
  - It costs 3 comparisons for every 2 elements.
    - The previous method costs 2 comparisons for each element.

![Diagram of elements and comparisons]

larger elements: compare to the current maximum
smaller elements: compare to the current minimum
Setting up the initial values for the min and max depends on whether $n$ is odd or even.
- If $n$ is even, compare the first two elements and assign the larger to max and the smaller to min.
- If $n$ is odd, set both min and max to the first element.

If $n$ is even, # of comparisons $= \frac{3(n-2)}{2} + 1 = \frac{3n}{2} - 2$.
If $n$ is odd, # of comparisons $= \frac{3(n-1)}{2} = 3\lceil n/2 \rceil$.
In either case, the # of comparisons is $\leq 3\lceil n/2 \rceil$. 
Outline

- Minimum and maximum
- Selection in expected linear time
- Selection in worst-case linear time
Selection in expected linear time

- In fact, selection of the $i$th smallest element of the array $A$ can be done in $\Theta(n)$ time.

- We first present a randomized version in this section and then present a deterministic version in the next section.

- The function **RANDOMIZED-SELECT:**
  - is a divide-and-conquer algorithm,
  - uses **RANDOMIZED-PARTITION** from the quicksort algorithm in Chapter 7, and
  - recurses on one side of the partition only.
RANDOMIZED-SELECT  procedure 1/2

1. RANDOMIZED-SELECT \((A, p, r, i)\)
2. \[\text{if } p = r\]
3. \[\text{then return } A[p]\]
4. \[q \leftarrow \text{RANDOMIZED-PARTITION}(A, p, r)\]
5. \[k \leftarrow q - p + 1\]
6. \[\text{if } i = k \quad \text{/* the pivot value is the answer */}\]
7. \[\text{then return } A[q]\]
8. \[\text{elseif } i < k\]
9. \[\text{then return } \text{RANDOMIZED-SELECT}(A, p, q - 1, i)\]
10. \[\text{else return } \text{RANDOMIZED-SELECT}(A, q, r, i - k)\]
RANDOMIZED-SELECT procedure

To find the $i$th order statistic in $A[p...q-1]$

To find the $(i-k)$th order statistic in $A[q+1...r]$

A[\(\leq A[q]\)] is the answer
Algorithm analysis

- The **worst case**: always recurse on a subarray that is only 1 element smaller than the previous subarray.
  
  \[ T(n) = T(n - 1) + \Theta(n) \]
  \[ = \Theta(n^2) \]

- The **best case**: always recurse on a subarray that has half of the elements smaller than the previous subarray.
  
  \[ T(n) = T(n/2) + \Theta(n) \]
  \[ = \Theta(n) \text{ (Master Theorem, case 3)} \]
The **average case:**

- We will show that $T(n) = \Theta(n)$.
- For $1 \leq k \leq n$, the probability that the subarray $A[p .. q]$ has $k$ elements is $1/n$.
- To obtain an upper bound, we assume that $T(n)$ is monotonically increasing and that the $i$th smallest element is always in the larger subarray.
- So, we have

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(\max(k - 1, n - k) + O(n))).$$
Algorithm analysis

\[ T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(\max(k-1, n-k)) + O(n)) = \frac{1}{n} \sum_{k=1}^{n} (T(\max(k-1, n-k))) + O(n) \]

\[ \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} T(k) + O(n). \]

\[ \because \max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lceil n/2 \rceil \\ n-k & \text{if } k \leq \lceil n/2 \rceil \end{cases} \]

<table>
<thead>
<tr>
<th>( k )</th>
<th>1</th>
<th>2</th>
<th>\ldots</th>
<th>\lceil n/2 \rceil</th>
<th>\lceil n/2 \rceil + 1</th>
<th>\ldots</th>
<th>n-1</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \max(k-1, n-k) )</td>
<td>n-1</td>
<td>n-2</td>
<td>\ldots</td>
<td>n-\lceil n/2 \rceil</td>
<td>\lceil n/2 \rceil</td>
<td>\ldots</td>
<td>n-2</td>
<td>n-1</td>
</tr>
</tbody>
</table>

- If \( n \) is even, each term from \( T(\lceil n/2 \rceil) \) to \( T(n-1) \) appears exactly twice.
- If \( n \) is odd, each term from \( T(\lceil n/2 \rceil) \) to \( T(n-1) \) appears exactly twice and \( T(\lfloor n/2 \rfloor) \) appears once.
- Because \( k = \lceil n/2 \rceil, k-1 = n-k = \lfloor n/2 \rfloor \).
Solve this recurrence by substitution:

- Assume $T(n) \leq cn$ for sufficiently large $c$.
- The function described by the $O(n)$ term is bounded by $an$ for all $n > 0$.
- Then, we have

$$T(n) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} T(k) + O(n) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} ck + an$$

$$= \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an = \frac{2c}{n} \left( \frac{(n-1)n}{2} - \left( \frac{n/2 - 1}{2} \right) \frac{n/2}{2} \right) + an$$

$$\leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{n/2 - 2}{2} \frac{n/2 - 1}{2} \right) + an$$

$$= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an = c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an$$

$$\leq \frac{3cn}{4} + \frac{c}{2} + an = cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right).$$

Thus, if we assume that $T(n) = O(1)$ for $n < 2c/(c - 4a)$, we have $T(n) = O(n)$. 
Outline

- Minimum and maximum
- Selection in expected linear time
- Selection in worst-case linear time
**SELECT algorithm**

- **Idea:** Guarantee a good split when the array is partitioned.

- **The SELECT algorithm:**
  1. Divide $n$ elements into groups of 5 elements.
  2. Find median of each of the $\left\lceil n/5 \right\rceil$ groups.
     - Run insertion sort on each group.
     - Then just pick the median from each group.
  3. Use SELECT recursively to find median $x$ of the $\left\lceil n/5 \right\rceil$ medians.
  4. Partition the $n$ elements around $x$.
     - Let $x$ be the $k$th element of the array after partitioning.
     - There are $k - 1$ elements on the low side of the partition and $n - k$ elements on the high side.
SELECT algorithm_2/2

5. Now there are three possibilities:

- If $i = k$, then return $x$.
- If $i < k$, then use SELECT recursively to find $i$th smallest element on the low side.
- If $i > k$, then use SELECT recursively to find $(i-k)$th smallest element on the high side.

![Diagram](image)

- $\bigcirc$: The median of a group.
- $\rightarrow$: From larger to smaller.
- $\text{Gray}$: Elements in this region are greater than $x$.
- $\text{Red}$: Elements in this region are smaller than $x$. 
At least half of the medians are $\geq x$.
- Precisely, at least $\lceil n/5 \rceil / 2$ medians $\geq x$.

These group contributes 3 elements that are $> x$, except for 2 of the groups:
- the group containing $x$, and
- the group with $< 5$ elements.

The number of elements greater than $x$ is at least:
- $3 \left( \lceil n/5 \rceil / 2 - 2 \right) \geq \frac{3n}{10} - 6$.

Similarly, at least $\frac{3n}{10} - 6$ elements are less than $x$.

Thus, SELECT is called recursively on $\leq \frac{7n}{10} + 6$ elements in step 5.
Time complexity

The **Select** algorithm:

1. Divide \( n \) elements into groups of 5 elements. \( O(n) \)
2. Find median of each of the \( \lceil n/5 \rceil \) groups. \( O(n) \)
   - Run insertion sort on each group.
   - Then just pick the median from each group.
3. Use **Select** recursively to find median \( x \) of the \( \lceil n/5 \rceil \) medians. \( T(\lceil n/5 \rceil) \)
4. Partition the \( n \) elements around \( x \). \( O(n) \)
   - Let \( x \) be the \( k \)th element of the array after partitioning.
   - There are \( k - 1 \) elements on the low side of the partition and \( n - k \) elements on the high side.
5. Now there are three possibilities: \( T(7n/10 + 6) \)
   - If \( i = k \), then return \( x \).
   - If \( i < k \), then use **Select** recursively to find \( i \)th smallest element on the low side.
   - If \( i > k \), then use **Select** recursively to find \( (i-k) \)th smallest element on the high side.

Time complexity: \( T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) \).
Solve this recurrence by substitution:

- Assume $T(n) \leq cn$ for sufficiently large $c$.
- The function described by the $O(n)$ term is bounded by $an$ for all $n > 0$.
- Then, we have

$$T(n) \leq c\left\lceil n/5 \right\rceil + c(7n/10 + 6) + an$$
$$\leq cn/5 + c + 7cn/10 + 6c + an$$
$$= 9cn/10 + 7c + an$$
$$= cn + (-cn/10 + 7c + an)$$

This last quantity is $\leq cn$ if we choose $c \geq 20a$.

$$-cn/10 + 7c + an \leq 0$$
$$cn/10 - 7c \geq an$$

Notice: $(n/n-70) \leq 2$ for $n \geq 140$. 

$$c(n-70) \geq 10an$$
$$c \geq 10a(n/(n-70))$$