THE SELECTION PROBLEM

- Input: An array \( A \) containing \( n \) distinct integer values, and an integer \( k \) between 1 and \( n \)
- Output: The \( k \)-th smallest integer in \( A \)
- Minimum is a special case where \( k = 1 \)
- Median is a special case where \( k = \frac{n}{2} \)
- Maximum is a special case where \( k = n \)

Simple algorithm for solving selection?

- **Reconstruction** \( (A, y) \)

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c} \hline
A & 1 & 2 & 3 & \cdots & i_y & \cdots & n \ \\
\hline
A_L & 1 & 2 & 3 & \cdots & i_y \ \\
\hline
A_R & y & \cdots & n \ \\
\hline
\end{array} \]

- Number of elements in this range = \( i_y \)
- **What's the \( k \)-th smallest element of \( A_L \)?**
  - If \( k = i_y \), then \( y \)
  - If \( k < i_y \), then the \( k \)-th smallest in \( A_L \)
  - If \( k > i_y \), then the \((k - i_y)\)-th smallest in \( A_R \)

Recursive calls

Suppose we choose a pivot element \( y \) in the array \( A \), and we **restructure** \( A \) so that all elements less than \( y \) precede \( y \) in \( A \), and all elements greater than \( y \) occur after \( y \) in \( A \). (This is exactly what is done in **QuickSort**, and it takes linear time.)

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c} \hline
A & 12 & 4 & 6 & 27 & 23 & 17 & 40 & 9 \ \\
\hline
\end{array} \]

**Restructure** \( (A, y) \)

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c} \hline
A & 12 & 4 & 6 & 27 & 23 & 17 & 40 & 9 \ \\
\hline
\end{array} \]

Number of elements on each side depend on the value \( y \).

**Recursive calls**

- Number of elements in this range = \( i_y \)
- **What's the \( k \)-th smallest element of \( A_L \)?**
  - If \( k = i_y \), then \( y \)
  - If \( k < i_y \), then the \( k \)-th smallest in \( A_L \)
  - If \( k > i_y \), then the \((k - i_y)\)-th smallest in \( A_R \)

Precondition: \( 1 \leq k \leq n \)

```python
1. QuickSelect(A, 1, n)      # base case
2. if n == 1 then return A[1] # if n = 1 then return A[1]
3. y = A[1]                 # pick an arbitrary pivot
4. (A_L, A_R, i_y) = Restructure(A, y)
5. if k == i_y then return y # if k = i_y, then return y
6. if k < i_y then return QuickSelect(A_L, 1, i_y) # if k < i_y, then return QuickSelect(A_L, 1, i_y)
7. if k > i_y then return QuickSelect(A_R, i_y + 1, n) # if k > i_y, then return QuickSelect(A_R, i_y + 1, n)
8. Restructure(A_L, i_y, y)  # allocate more than enough
9. A_R = new array(1, n)    # to avoid need for expansion
10. nL = 0, nR = 0
11. for i = 1..n
14. return (A_L, A_R, nL+1) # nL+1 is the new index of y
```
OVERLY OPTIMISTIC ANALYSIS 😊

If \( i_y = \frac{n}{2} \), then we recurse on \( \frac{n}{2} \) elements.
- If we could always recurse on \( \frac{n}{2} \) elements then
  - We would get \( T(n) = T\left(\frac{n}{2}\right) + \Theta(n) \)
  - Which would yield \( a = 1, b = 2, y = 1, x = \log_2 1 = 0 \), \( y > x \) and \( T(n) \in \Theta(n^2) \) by the Master theorem.

But we don't always recurse on \( \frac{n}{2} \) elements!

WORST-CASE ANALYSIS

- If we always get \( i_y = 1 \) and recurse on the right, then
  - We get \( T(n) = T(n-1) + \Theta(n) \)
  - By the substitution method this is \( \Theta(n^2) \)
  - So, sometimes the pivot is good, sometimes it's bad...
  - What about the average case?

AVERAGE-CASE ANALYSIS

Definition: we say a pivot \( y \) is good if \( i_y \in \left[\frac{3n}{4}, \frac{3n}{2}\right] \)

For any good pivot
- we recurse on at most \( \frac{3n}{4} \) elements
- Probability of an arbitrary pivot being good? \( \frac{1}{2} \)

PROOF SKETCH

Since probability of a good pivot is \( \frac{1}{2} \),
- on average, every two recursive calls, we will encounter a good pivot
- Encountering a good pivot reduces problem size to at most \( \frac{3n}{4} \)
- So, problem size is reduced to \( \frac{3n}{4} \) after expected linear work
- Average case recurrence: \( T(n) = T\left(\frac{n}{4}\right) + \Theta(n) \)
  \( T(n) \in \Theta(n) \)

TAKING SELECTION FURTHER

We just showed: QuickSelect with average case runtime in \( \Theta(n) \)
Next up:
- Median-of-medians QuickSelect (MOMQuickSelect) word case runtime in \( \Theta(n) \)
- The algorithm we will use picks a good pivot in every recursive call
- Must get a good pivot within \( \Theta(n) \) recursive calls always
- Relies on getting a good pivot within \( \Theta(n) \) recursive calls on average
HIGH LEVEL ALGORITHM

- Similar to QuickSelect
  - Choose a pivot
  - Move smaller elements to the left of the pivot, and larger elements to the right of the pivot
  - Recursively call MOMQuickSelect on one subarray
- Only difference is how we choose the pivot
  - Always want to pick a good pivot

HOW GOOD IS THE PIVOT 23?

<table>
<thead>
<tr>
<th>Recall median of each row</th>
<th>Imagine sorting each row</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 38 22 20 19 17 21 14</td>
<td>13 17 19 20 21 22 23 24</td>
</tr>
<tr>
<td>11 34 9 7 5 3 2 1</td>
<td>11 1 2 3 5 7 9 13</td>
</tr>
<tr>
<td>8 34 49 47 28</td>
<td>8 28 34 47 49</td>
</tr>
<tr>
<td>18 44 31 44 48</td>
<td>18 31 44 48 44</td>
</tr>
<tr>
<td>24 4 2 50 23</td>
<td>24 2 4 23 50</td>
</tr>
<tr>
<td>10 32 35 41 24</td>
<td>10 24 32 35 41</td>
</tr>
<tr>
<td>1 50 12 12 28</td>
<td>1 12 12 28 50</td>
</tr>
<tr>
<td>19 36 33 31</td>
<td>19 31 33 36</td>
</tr>
<tr>
<td>25 40 20 40 40</td>
<td>25 20 40 40 40</td>
</tr>
</tbody>
</table>

# elements ≤ 23 is at least 31.5. This is at least 1/20th of our 50-element input, or 3/10th.

So, after restructuring, pivot 23 must have at least 3/10ths of our 50 elements before and after it.

We recurse on $A_1$ or $A_2$, and both have size at most $7.5$.

ALWAYS PICKING A GOOD PIVOT

Example input:

```
11, 38, 6, 21, 20, 19, 17, 5, 8, 34, 49, 47, 28, 18, 44, 31, 44, 48, 19, 44, 31, 44, 48, 17, 19, 36, 33, 37, 10, 24, 32, 35, 41, 24, 50, 23, 40, 27, 4, 2, 50, 23, 10, 32, 35, 41, 24, 16, 19, 36, 33, 37, 15, 23, 38, 20, 40, 40, 40
```

Group into rows of 5

<table>
<thead>
<tr>
<th>Find median of each row</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 38 6 21 20</td>
</tr>
<tr>
<td>12 14 9 7 5</td>
</tr>
<tr>
<td>8 34 49 47 28</td>
</tr>
<tr>
<td>18 44 31 44 48</td>
</tr>
<tr>
<td>24 4 2 50 23</td>
</tr>
<tr>
<td>10 32 35 41 24</td>
</tr>
<tr>
<td>16 19 36 33 37</td>
</tr>
<tr>
<td>15 23 38 20 40</td>
</tr>
</tbody>
</table>

Time complexity for this step:

<table>
<thead>
<tr>
<th>Build array of medians</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 38 6 21 20</td>
</tr>
<tr>
<td>12 14 9 7 5</td>
</tr>
<tr>
<td>8 34 49 47 28</td>
</tr>
<tr>
<td>18 44 31 44 48</td>
</tr>
<tr>
<td>24 4 2 50 23</td>
</tr>
<tr>
<td>10 32 35 41 24</td>
</tr>
<tr>
<td>16 19 36 33 37</td>
</tr>
<tr>
<td>15 23 38 20 40</td>
</tr>
</tbody>
</table>

Time complexity for this step:

Recursive finding medians of these medians: 23

Recursive pruning used
How much does the problem shrink?

- Shrinks by at least 3(r + 1)
- Problem size = n = 10r + 5
- Subproblem size = n – Shrink = n – 3(r + 1) = 10r + 5 – 3r – 3 = 7r + 2
- Express in terms of n using r = \[ \frac{n-5}{10} \]
  Subproblem size ≤ 7 \[ \frac{n-5}{10} \] + 2 ≤ \[ \frac{n-5}{10} \] + 2
  = \[ \frac{2n}{10} - \frac{7}{10} \] + 2 = \[ \frac{2n}{10} - \frac{3}{10} \]

Time complexity

Let \( T(n) = c'n + \frac{T(n/5)}{5} + \frac{T(7n/10)}{10} \) where \( c' > 0 \)

Want to prove: \( T(n) = cn \) for some \( c > 0 \)

Note \( c \) and \( c' \) are independent constants
- \( c' \) comes from the work at each level of recursion being \( O(n) \)
- \( c \) is a positive constant we are trying to show exists

I.H.: Suppose \( 3c > 0 \) : \( T(n') = cn' \) for \( 15 \leq n' < n \)

\[ T(n) = c'n + \frac{c'n}{5} + \frac{7n}{10} \] (by inductive hypoth.)

\[ T(n) = cn \] (want this to be true)

\[ \Leftrightarrow c'n + \frac{c'n}{5} + \frac{7n}{10} = cn \] (equivalently)

\[ \Leftrightarrow c' + \frac{c'}{5} + \frac{7}{10} = c \Leftrightarrow c = 10c' \] (by algebra)