Deterministic Selection

https://cs.pomona.edu/classes/cs140/
Selection Problem

**Input:** A set of $n$ numbers and an integer $i$, with $1 \leq i \leq n$

**Output:** The element that is larger than exactly $i - 1$ other elements

- Known as the $i^{th}$ order statistic or the $i^{th}$ smallest number
- The minimum element is the $1^{st}$ order statistic ($i = 1$)
- The maximum element is the $n^{th}$ order statistic ($i = n$)

- What is “$i$” for the median? (an expression base on $n$)
  - If $n$ is **even**, then the medians are the $n/2$ and $n/2 + 1$ order statistics
  - If $n$ is **odd**, then the median is the $(n + 1)/2$ order statistic
Selection Problem

*Find the $i^{th}$ smallest number in an array*

We can reduce this to sorting:
- $O(n \lg n)$

We can use Quickselect (randomized selection):
- Best Case: $O(n)$
- Average Case: $O(n)$
- Worst Case: $O(n^2)$
Key Component of Quickselect: Partitioning

What if we are looking for the 5th order statistic?

- What is the fifth order statistic?
- Do we need to recursively look on both sides of the pivot?
Deterministic Selection

Works like Quicksort

**Deterministically** choose “good” pivot (*close* to a 50-50 split)

- The pivot is some value near to the median

Goal: select a pivot that is *guaranteed* to be pretty good

Key idea: find the *median of medians*
Deterministic Selection, Pivot Selection

• Break input into groups of size 5 (n/5 total groups)
• Sort each group
• Copy the n/5 medians (middle elements) from each group
• Recursively compute the median of medians
• Use the median of medians as the pivot
• Partition using this pivot

Return
• the pivot element, or
• recursively search the left and right

You can call this higher-level pseudocode
FUNCTION DSelect(array, i)
    # Base 1 indexing (makes it easier to interpret indices)
    n = array.length
    IF n == 1, RETURN A[1]

    groups = CreateGroupsOfFive(array)
    groups_sorted = SortGroupsOfFive(groups)
    medians = GetMediansGroupsOfFive(groups_sorted)

    # Get median of medians and call it the pivot
    pivot = DSelect(medians, n/5/2)

    left, right, pivot_index = Partition(array, pivot)

    IF pivot_index == i, RETURN pivot
    IF pivot_index < i, RETURN DSelect(left, i)
    IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
n=20

CreateGroupsOfFive(array)

[7, 2, 17, 12, 13, 8, 20, 4, 6, 3, 19, 1, 9, 5, 16, 10, 15, 18, 14, 11]
n=20

CreateGroupsOfFive(array)

SortGroupsOfFive(groups)

GetMediansGroupsOfFive(groups_sorted)

What is the median of medians?
FUNCTION DSelect(array, i)
# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
DSelect Running Time

**Theorem:**
- for every input array of length $n$, DSelect returns the $i^{th}$ order statistic in $O(n)$

Seems impossible since (compared with quicksort) we’ve added
- another recursive call (to find the pivot) and
- a bunch of work to find the median of medians
FUNCTION DSelect(array, i)

# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
FUNCTION DSelect(array, i)

# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
FUNCTION DSelect(array, i)

# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
Sorting 5 elements

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>11</td>
<td>17</td>
<td>7</td>
<td>13</td>
</tr>
</tbody>
</table>

Steps:
1. A > B
   - Swap(A, B)

compare
Sorting 5 elements

Steps:
1. A > B
   • Swap(A, B)
2. C > D
   • Swap(C, D)
### Sorting 5 elements

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11</td>
<td>19</td>
<td>7</td>
<td>17</td>
<td>13</td>
</tr>
</tbody>
</table>

**Steps:**

1. **A > B**
   - Swap(A, B)

2. **C > D**
   - Swap(C, D)

3. **A > C**
   - Swap(A, C)
   - Swap(B, D)
Sorting 5 elements

Steps:
1. A > B
   • Swap(A, B)
2. C > D
   • Swap(C, D)
3. A > C
   • Swap(A, C)
   • Swap(B, D)
4. E into A-C-D
5. E into A-C-D
Steps:
1. A > B
   • Swap(A, B)
2. C > D
   • Swap(C, D)
3. A > C
   • Swap(A, C)
   • Swap(B, D)
4. E into A-C-D
5. E into A-C-D
### Sorting 5 elements

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>17</td>
<td>11</td>
<td>19</td>
<td>13</td>
</tr>
</tbody>
</table>

**Steps:**

1. A > B  
   - Swap(A, B)

2. C > D  
   - Swap(C, D)

3. A > C  
   - Swap(A, C)  
   - Swap(B, D)

4. E into A-C-D

5. E into A-C-D

6. B into C-E-D

7. B into C-E-D
At most 7 comparisons!

Steps:
1. A > B  
   • Swap(A, B)
2. C > D  
   • Swap(C, D)
3. A > C  
   • Swap(A, C)  
   • Swap(B, D)
4. E into A-C-D
5. E into A-C-D
6. B into C-E-D
7. B into C-E-D

Return: [A, C, E, B, D]
FUNCTION DSelect(array, i)
    # Base 1 indexing
    n = array.length
    IF n == 1, RETURN A[1]

    groups = CreateGroupsOfFive(array)
    groups_sorted = SortGroupsOfFive(groups)
    medians = GetMediansGroupsOfFive(groups_sorted)

    # Get median of medians and call it the pivot
    pivot = DSelect(medians, n/5/2)

    left, right, pivot_index = Partition(array, pivot)

    IF pivot_index == i, RETURN pivot
    IF pivot_index < i, RETURN DSelect(left, i)
    IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
FUNCTION DSelect(array, i)
# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]
groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)
# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)
left, right, pivot_index = Partition(array, pivot)
IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
FUNCTION DSelect(array, i)

# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
FUNCTION DSelect(array, i)

# Base 1 indexing
n = array.length
IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
DSelect Running Time

\[ T(n) = \text{maximum \# of operations required for input of length } n \]

\[ T(1) = O(1) \]

\[ T(n) \leq cn + T\left(\frac{n}{5}\right) + T(?) \]

- Finding a good pivot
- Recursively searching one side
- Sorting, partitioning, copying, etc.

On what does the “?” depend?
DSelect Running Time

\[ T(n) = \text{maximum \# of operations required for input of length } n \]

\[ T(1) = O(1) \]

\[ T(n) \leq cn + T \left( \frac{n}{5} \right) + T(?) \]

Finding a good pivot

Sorting, partitioning, copying, etc.

Lemma:
the recursive search is guaranteed to be on an array of size \( \leq 7n/10 \)
FUNCTION DSelect(array, i)

# Base 1 indexing
n = array.length

IF n == 1, RETURN A[1]

groups = CreateGroupsOfFive(array)
groups_sorted = SortGroupsOfFive(groups)
medians = GetMediansGroupsOfFive(groups_sorted)

# Get median of medians and call it the pivot
pivot = DSelect(medians, n/5/2)

left, right, pivot_index = Partition(array, pivot)

IF pivot_index == i, RETURN pivot
IF pivot_index < i, RETURN DSelect(left, i)
IF pivot_index > i, RETURN DSelect(right, i - pivot_index)
Selecting the pivot

- We can now replace the “?” with $7n/10$
- Let $k = n/5$ be the number of groups of size 5
- Let $x_i = i^{th}$ smallest element of the $k$ medians
- So, the pivot is $x_{k/2}$ (the median of medians)

- Our goal is to show that:
  - $\leq 30\%$ of the input array is smaller than $x_{k/2}$
  - $\leq 30\%$ of the input array is larger than $x_{k/2}$

This means that we must search at most $70\%$ ($7/10^{ths}$) of the remaining input

\[ T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \]
What is the median of medians?
n=20

CreateGroupsOfFive(array)

SortGroupsOfFive(groups)

GetMediansGroupsOfFive(groups_sorted)

What is the median of medians?

From where do we get 30%?
This is just a diagram to show what we're looking at.
Guaranteed bigger than (or equal to) (at least) 3/5 of 1/2 of the groups = \(30\%\)

<table>
<thead>
<tr>
<th></th>
<th>20</th>
<th>19</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Guaranteed smaller than (or equal to) (at least) $\frac{3}{5}$ of $\frac{1}{2}$ of the groups = **30%**

<table>
<thead>
<tr>
<th>$x_{k/2}$</th>
<th>20</th>
<th>19</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Guaranteed **bigger** than (or equal to) (at least) 3/5 of 1/2 of the groups = 30%

Guaranteed **smaller** than (or equal to) (at least) 3/5 of 1/2 of the groups = 30%
So, we need to search either the (at most) **upper** 70% of the array or the (at most) **lower** 70% of the array.

<table>
<thead>
<tr>
<th>$x_{k/2}$</th>
<th>20</th>
<th>19</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>16</td>
<td>13</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>12</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Total Running Time

\[ T(n) \leq cn + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \]

• Can we use the master method?
• Not all subproblems are the same size
• We are going to use the substitution method
Guess and Check

Claim: \( T(n) = O(n) \)

\[
c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \leq an \quad \forall n \geq n_0
\]

Let \( a = 10c_{DS}, \) and \( n_0 = 1 \)

Proof by induction
1. Base Case: \( T(1) \leq a \cdot n = a \cdot 1 = 10c_{DS} \)
2. Inductive Hypothesis: Assume \( T(k) \leq ak \) for \( k < n \)
3. Induction Step

\[
T(n) \leq c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \leq c_{DS}n + a \cdot \frac{n}{5} + a \cdot \frac{7n}{10} \leq an
\]
\[ T(n) \leq c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \leq c_{DS}n + a\frac{n}{5} + a\frac{7n}{10} \leq an \]

Let \( a = 10c_{DS} \), and \( n_0 = 1 \)
Selection

Randomized selection (average \(O(n)\) runtime)
- Fast and practical
- All operations done in-place
- Small constant factors

Deterministic selection (\textit{guaranteed} \(O(n)\) runtime)
- Slower in practice
- Extra memory required
- Large constant factors (extra non-recursive work)