# Deterministic Selection

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

# Outline

## Topics and Learning Objectives

- Analyze an irregular, deterministic, and divide and conquer algorithm for the selection problem.

## Exercise

- Recursion Tree

# Resources

- Algorithms Illuminated Chapter 6

# Selection Problem

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

**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 <u>even</u>, then the medians are the $n/2$ and $n/2 + 1$ order statistics
  - If $n$ is <u>odd</u>, 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

Pivot

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 2 | 5 | 1 | 4 | 7 | 6 |

Pivot

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 6 | 7 | 4 | 5 | 8 |

<u>What if we are looking for the 5$^{th}$ order statistic?</u>

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

# Outline for today

- Describe the deterministic selection algorithm
- Prove the running time
- We will not prove correctness (see the randomized select proof)

# 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

1. Break input into groups of size 5 (n/5 total groups)
2. Sort each group
3. Copy the n/5 medians (middle elements) from each group
4. Recursively compute the median of medians
5. Use the median of medians as the pivot
6. Partition using this pivot
7. Return
   - the pivot element, or
   - recursively search the left or 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)
```

```
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)
```

Base Case

Recursively find median of medians

Partition

Recursion

n=20

`CreateGroupsOfFive(array)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 17 | 12 | 13 | 8 | 20 | 4 | 6 | 3 | 19 | 1 | 9 | 5 | 16 | 10 | 15 | 18 | 14 | 11 |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 17 | 12 | 13 | 8 | 20 | 4 | 6 | 3 | 19 | 1 | 9 | 5 | 16 | 10 | 15 | 18 | 14 | 11 |

n=20

`CreateGroupsOfFive(array)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 17 | 12 | 13 | 8 | 20 | 4 | 6 | 3 | 19 | 1 | 9 | 5 | 16 | 10 | 15 | 18 | 14 | 11 |

`SortGroupsOfFive(groups)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 7 | 12 | 13 | 17 | 3 | 4 | 6 | 8 | 20 | 1 | 5 | 9 | 16 | 19 | 10 | 11 | 14 | 15 | 18 |

`GetMediansGroupsOfFive(groups_sorted)`

What is the median of medians?

What is the running time T(n) of DSelect?

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

```
    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)
```

Recursively find median of medians

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

Partition

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

Recursion

# 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

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
n = array.length
IF n == 1, RETURN A[1]
```

Base Case

```
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)
```

Recursively find median of medians

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

Partition

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

Recursion

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

```
    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)
```

Recursively find median of medians

What must be the running time of this work?

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

Partition

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

Recursion

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

```
    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)
```

Recursively find median of medians

What must be the running time of this work?

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

Partition

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

Recursion

# Sorting 5 elements

| A | B | C | D | E |
|---|---|---|---|---|
| 19 | 11 | 17 | 7 | 13 |

1

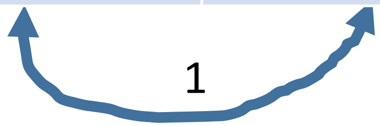
compare

Steps:

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

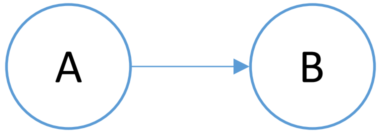
# Sorting 5 elements

| A | B | C | D | E |
|---|---|---|---|---|
| 11 | 19 | 17 | 7 | 13 |



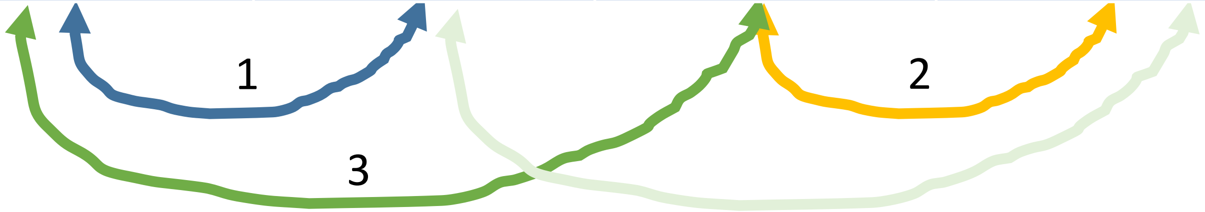

Steps:

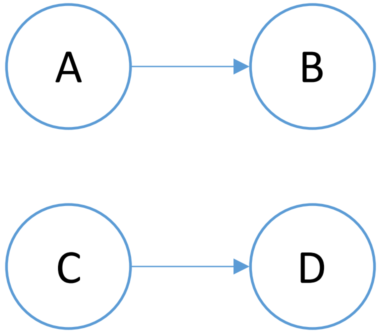
1. A > B
   - Swap(A, B)
2. C > D
   - Swap(C, D)



# Sorting 5 elements

| A | B | C | D | E |
|---|---|---|---|---|
| 11 | 19 | 7 | 17 | 13 |

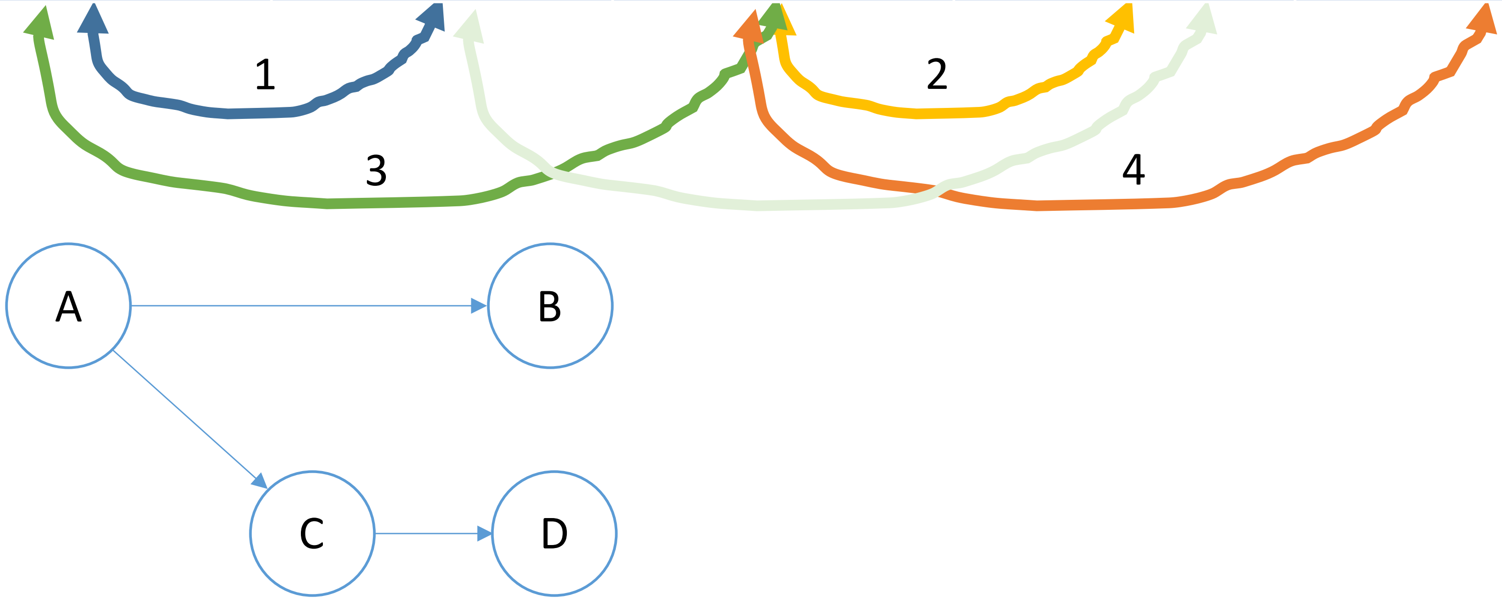



Steps:
1. A > B
   - Swap(A, B)
2. C > D
   - Swap(C, D)
3. A > C
   - Swap(A, C)
   - Swap(B, D)

# Sorting 5 elements

| A | B | C | D | E |
|---|---|---|---|---|
| 7 | 17 | 11 | 19 | 13 |

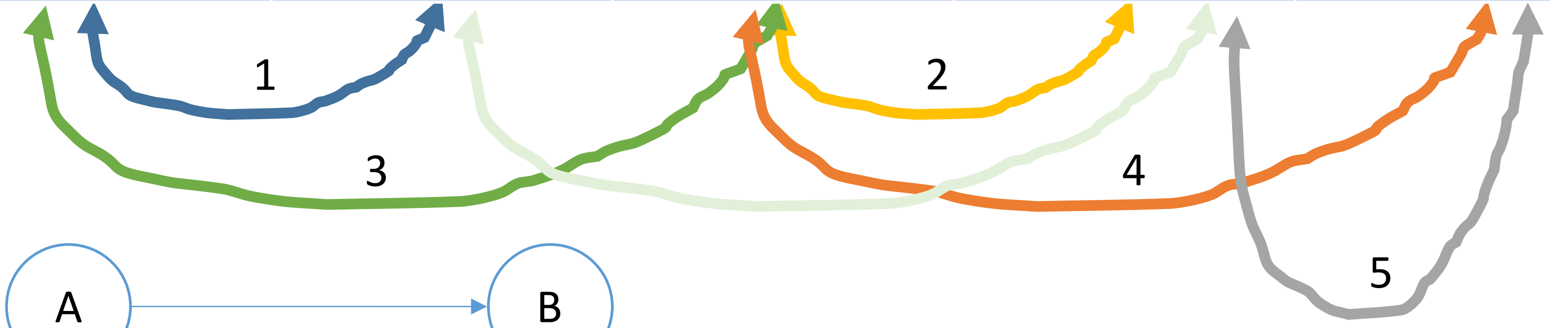


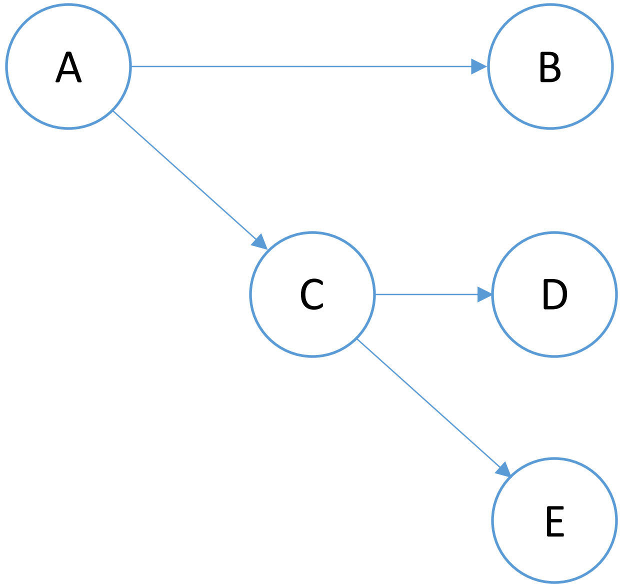

Steps:

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

| A | B | C | D | E |
|---|---|---|---|---|
| 7 | 17 | 11 | 19 | 13 |



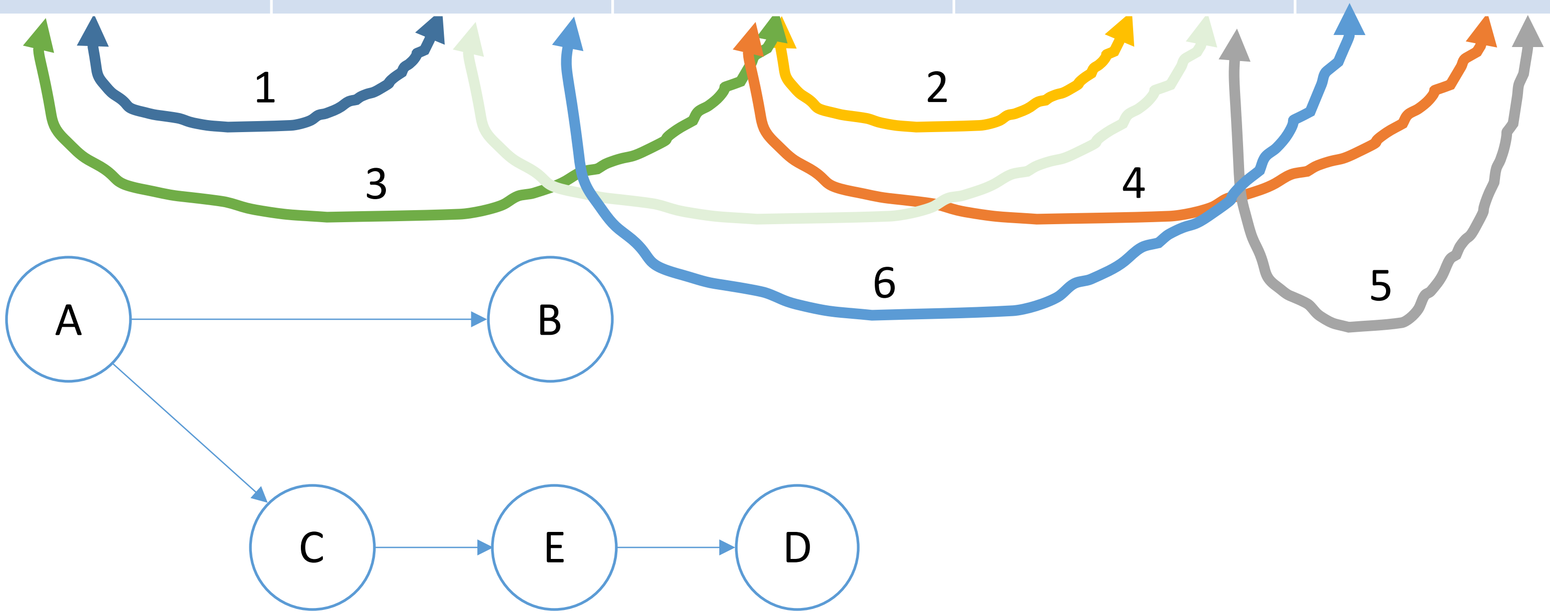




Steps:

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

| A | B | C | D | E |
|---|---|---|---|---|
| 7 | 17 | 11 | 19 | 13 |

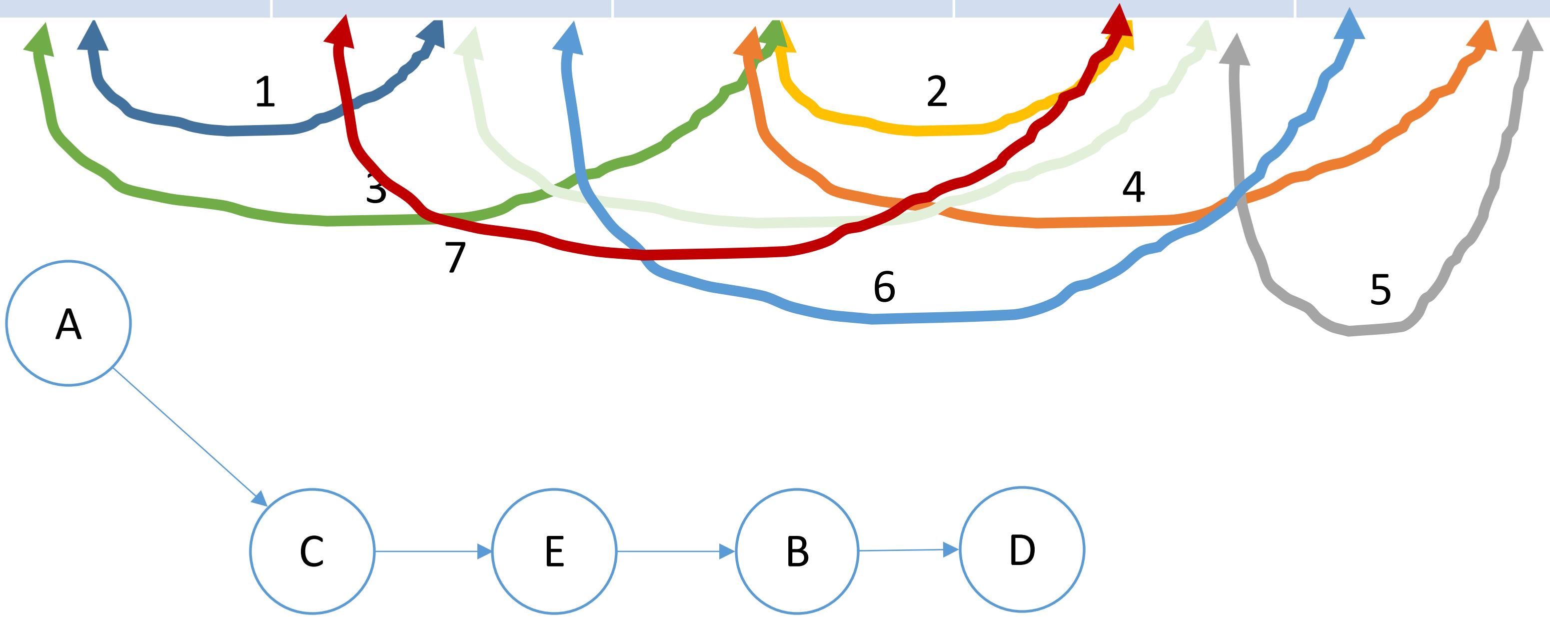



Steps:

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

# Sorting 5 elements

| A | B | C | D | E |
|---|---|---|---|---|
| 7 | 17 | 11 | 19 | 13 |




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]

At most 7 comparisons!

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

O(n)

```
    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)
```

Recursively find median of medians

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

Partition

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

Recursion

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

O(n)

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

Recursively find median of medians

T(n/5)

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

How can we denote this recursive running time?

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

Partition

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

Recursion

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

O(n)

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

T(n/5)

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

Recursively find median of medians

O(n)

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

Partition

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

Recursion

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1) — Base Case

```
    n = array.length
    IF n == 1, RETURN A[1]
```

O(n) — Recursively find median of medians

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

T(n/5)

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

O(n) — Partition

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

T(?) — Recursion

```
    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) = maximum # of operations required for input of length n

$$T(1) = O(1)$$

Finding a good pivot

$$T(n) \leq cn + T\left(\frac{n}{5}\right) + T(?\,)$$

Sorting, partitioning, copying, etc.

Recursively searching one side

On what does the “?” depend?

# DSelect Running Time

T(n) = maximum # of operations required for input of length n

$T(1) = O(1)$

Finding a good pivot

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

Sorting, partitioning, copying, etc.

Lemma:
the recursive search is guaranteed to be on an array of size ≤ 7n/10

T(n)

```
FUNCTION DSelect(array, i)
    # Base 1 indexing
```

O(1)

```
    n = array.length
    IF n == 1, RETURN A[1]
```

Base Case

O(n)

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

T(n/5)

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

Recursively find median of medians

O(n)

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

Partition

T(7n/10)

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

Recursion

# Selecting the pivot

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

- 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

n=20

`CreateGroupsOfFive(array)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 17 | 12 | 13 | 8 | 20 | 4 | 6 | 3 | 19 | 1 | 9 | 5 | 16 | 10 | 15 | 18 | 14 | 11 |

`SortGroupsOfFive(groups)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 7 | 12 | 13 | 17 | 3 | 4 | 6 | 8 | 20 | 1 | 5 | 9 | 16 | 19 | 10 | 11 | 14 | 15 | 18 |

`GetMediansGroupsOfFive(groups_sorted)`

What is the median of medians?

n=20

`CreateGroupsOfFive(array)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 17 | 12 | 13 | 8 | 20 | 4 | 6 | 3 | 19 | 1 | 9 | 5 | 16 | 10 | 15 | 18 | 14 | 11 |

`SortGroupsOfFive(groups)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 7 | 12 | 13 | 17 | 3 | 4 | 6 | 8 | 20 | 1 | 5 | 9 | 16 | 19 | 10 | 11 | 14 | 15 | 18 |

`GetMediansGroupsOfFive(groups_sorted)`

What is the median of medians?

# From where do we get 30%?

n=20

`CreateGroupsOfFive(array)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 2 | 17 | 12 | 13 | 8 | 20 | 4 | 6 | 3 | 19 | 1 | 9 | 5 | 16 | 10 | 15 | 18 | 14 | 11 |

`SortGroupsOfFive(groups)`

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 7 | 12 | 13 | 17 | 3 | 4 | 6 | 8 | 20 | 1 | 5 | 9 | 16 | 19 | 10 | 11 | 14 | 15 | 18 |

`GetMediansGroupsOfFive(groups_sorted)`

| | | | |
|---|---|---|---|
| 20 | 19 | 17 | 18 |
| 8 | 16 | 13 | 15 |
| 6 | 9 | 12 | 14 |
| 4 | 5 | 7 | 11 |
| 3 | 1 | 2 | 10 |

This is a just a diagram to show what we're looking at; we don't actually sort the groups among each other.

Guaranteed bigger than (or equal to) (at least) 3/5 of 1/2 of the groups = **<u>30%</u>**

$x_{k/2}$

| | | | |
|---|---|---|---|
| 20 | 19 | 17 | 18 |
| 8 | 16 | 13 | 15 |
| 6 | 9 | 12 | 14 |
| 4 | 5 | 7 | 11 |
| 3 | 1 | 2 | 10 |

Guaranteed smaller than (or equal to) (at least) 3/5 of 1/2 of the groups = **30%**

| | | | | |
|---|---|---|---|---|
| | 20 | 19 | 17 | 18 |
| | 8 | 16 | 13 | 15 |
| $x_{k/2}$ | 6 | 9 | 12 | 14 |
| | 4 | 5 | 7 | 11 |
| | 3 | 1 | 2 | 10 |

Guaranteed **bigger** than (or equal to) (at least)
3/5 of 1/2 of the groups = **<u>30%</u>**
Guaranteed **smaller** than (or equal to) (at least)
3/5 of 1/2 of the groups = **<u>30%</u>**

| | | | | |
|---|---|---|---|---|
| | 20 | 19 | 17 | 18 |
| | 8 | 16 | 13 | 15 |
| $x_{k/2}$ | 6 | 9 | 12 | 14 |
| | 4 | 5 | 7 | 11 |
| | 3 | 1 | 2 | 10 |

So, we need to search either the (at most) upper 70% of the array or the (at most) lower 70% of the array.

| | | | | |
|---|---|---|---|---|
| | 20 | 19 | 17 | 18 |
| | 8 | 16 | 13 | 15 |
| $x_{k/2}$ | 6 | 9 | 12 | 14 |
| | 4 | 5 | 7 | 11 |
| | 3 | 1 | 2 | 10 |

# 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

$$T(n) \le c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

# Guess and Check

**Claim**: $T(n) = O(n)$

$$c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \le an \qquad \forall n \ge n_0$$

Guess: $a = 10c_{DS}$, and $n_0 = 1$

Proof by induction

1. Base Case: $T(1) \le a \cdot n = a \cdot 1 = 10c_{DS}$
2. Inductive Hypothesis: Assume $T(k) \le ak$ for $k < n$
3. Induction Step (apply hypothesis)

$$T(n) \le c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \le c_{DS}n + a\frac{n}{5} + a\frac{7n}{10} \le an$$

$$T(n) \le c_{DS}n + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) \le c_{DS}n + a\frac{n}{5} + a\frac{7n}{10} \le an$$

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 (**guaranteed** O(n) runtime)

- Slower in practice
- Extra memory required
- Large constant factors (extra non-recursive work)