Hello and ... Please, limit laptops (unless lecture draft slides), ... Bad for your learning. Worse for your neighbors learning. If you must leave early, please sit by exit. Thank you!
Hello and ...

H . . . H . S . H H
Hello and ...

H . . . H . . . H H H
S
Hello and ...

H ... H . . H
S H
Hello and ...

Please, limit laptops (unless lecture draft slides), Bad for your learning. Worse for your neighbors learning. If you must leave early, please sit by exit. Thank you!
Hello and ...

H . . . . . . .
S H H H H
Hello and ...

S H H H H
Hello and ...

Please, limit laptops (unless lecture draft slides), ... Bad for your learning. Worse for your neighbors learning. If you must leave early, please sit by exit.
Thank you!
Hello and ...
Hello and ...
Hello and ...
Hello and ...

S H H H H H . . . . .
Hello and ...

Please,
Hello and ...

Please, limit laptops (unless lecture draft slides),
Hello and ...

S H H H H H . . . .

Please, limit laptops (unless lecture draft slides), ...
Hello and ...

Please, limit laptops (unless lecture draft slides), ...
Bad for your learning. Worse for your neighbors learning.
Hello and ...

S H H H H H . . . . .

Please, limit laptops (unless lecture draft slides), ...

Bad for your learning. Worse for your neighbors learning.

If you must leave early, please sit by exit.
Hello and ...

S H H H H . . . .

Please, limit laptops (unless lecture draft slides), ...

Bad for your learning. Worse for your neighbors learning.
If you must leave early, please sit by exit.
Thank you!
Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) \]

Also: iterative view.

Sorting Lower Bound.

\[ n! \text{ possible output orderings.} \]

Comparison splits outputs into 2.

\[ \Omega(\log n!) = \Omega(n \log n) \text{ time.} \]

Median finding.

Selection: more general, "strengthen induction."

Random pivot element to split elements.

Recurse on one subset.

Expected Time Analysis:

\[ O(n) \text{ time to decrease size by } \frac{3}{4}. \]

Extra: Deterministic Pivot Selection.
Lecture in a minute.

MergeSort.
   Sort two halves, put together.
       Merge: two pointer scan.
   \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
   \[ T(n) = O(n\log n). \]
Also: iterative view.
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
    Merge: two pointer scan.
    $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$.
    $T(n) = O(n\log n)$.
  Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.
Comparison splits outputs into 2.
$\Omega(\log n!) = \Omega(n\log n)$ time.

Median finding.
Selection: more general, "strengthen induction."
Random pivot element to split elements.
Recurse on one subset.

Expected Time Analysis:
$O(n)$ time to decrease size by $\frac{3}{4}$.

Extra: Deterministic Pivot Selection.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
Merge: two pointer scan.

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
\[ T(n) = O(n\log n). \]
Also: iterative view.

Sorting Lower Bound.
\[ n! \text{ possible output orderings.} \]
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) \]
  \[ T(n) = O(n\log n) \]
  Also: iterative view.

Sorting Lower Bound.
  \( n! \) possible output orderings.
  Comparison splits outputs into 2.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
  Merge: two pointer scan.
  $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$.
  $T(n) = O(n\log n)$.

Also: iterative view.

Sorting Lower Bound.
  $n!$ possible output orderings.
  Comparison splits outputs into 2.
  $\Omega(\log n!) = \Omega(n\log n)$ time.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n\log n). \]
Also: iterative view.

Sorting Lower Bound.
\( n! \) possible output orderings.
Comparison splits outputs into 2.
\( \Omega(\log n!) = \Omega(n\log n) \) time.

Median finding.
Selection: more general, “strengthen induction.”
Random pivot element to split elements.
Recurse on one subset.
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
  Merge: two pointer scan.
  \( T(n) = 2T\left(\frac{n}{2}\right) + O(n) \).
  \( T(n) = O(n \log n) \).
  Also: iterative view.

Sorting Lower Bound.
  \( n! \) possible output orderings.
  Comparison splits outputs into 2.
  \( \Omega(\log n!) = \Omega(n \log n) \) time.

Median finding.
  Selection: more general, “strengthen induction.”
  Random pivot element to split elements.
  Recurse on one subset.

Expected Time Analysis:
  \( O(n) \) time to decrease size by \( 3/4 \).
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n\log n). \]
  Also: iterative view.

Sorting Lower Bound.
  \( n! \) possible output orderings.
  Comparison splits outputs into 2.
  \( \Omega(\log n!) = \Omega(n\log n) \) time.

Median finding.
  Selection: more general, “strengthen induction.”
  Random pivot element to split elements.
  Recurse on one subset.

Expected Time Analysis:
  \( O(n) \) time to decrease size by \( 3/4 \).

Extra: Deterministic Pivot Selection.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n\log n). \]
Also: iterative view.

Sorting Lower Bound.
  \( n! \) possible output orderings.
  Comparison splits outputs into 2.
  \( \Omega(\log n!) = \Omega(n\log n) \) time.

Median finding.
  Selection: more general, “strengthen induction.”
  Random pivot element to split elements.
  Recurse on one subset.
Expected Time Analysis:
  \( O(n) \) time to decrease size by 3/4.

Extra: Deterministic Pivot Selection.
More divide and conquer: mergesort.

Sort items in $n$ elt array: $A = [a_1, \ldots, a_n]$,
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),

E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).

**Mergesort(A)**

if (length(A) > 1)
    return
    (merge(mergesort(a[1],...,a[n/2]),
        mergesort(a[n/2+1],...,a[n])))
else
    return a
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).

**Mergesort(A)**

```plaintext
if (length(A) >1)
    return 
    (merge(mergesort(a[1],\ldots,a[n/2]),
       mergesort(a[n/2+1],\ldots,a[n])))
else
    return a
```

How to merge?
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3, \ldots] \).

\textbf{Mergesort}(A)
\begin{verbatim}
if (length(A) >1)
    return
        (merge(mergesort(a[1],\ldots,a[n/2]),
               mergesort(a[n/2+1],\ldots,a[n]))
else
    return a
\end{verbatim}

How to merge?
Choose lowest from two lists,
More divide and conquer: mergesort.

Sort items in $n$ elt array: $A = [a_1, \ldots, a_n]$, E.g., $A = [5, 6, 7, 9, 10, 2, 3\ldots]$.

**Mergesort(A)**

if (length(A) > 1)
    return 
    (merge(mergesort(a[1],\ldots,a[n/2])),
     mergesort(a[n/2+1],\ldots,a[n]))
else
    return a

How to merge?

Choose lowest from two lists, cross out,
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3] \).

**Mergesort(A)**

\[
\text{if } (\text{length}(A) > 1) \quad \text{return} \\
\quad (\text{merge}(\text{mergesort}(a[1], \ldots, a[n/2])), \quad \\
\quad \text{mergesort}(a[n/2+1], \ldots, a[n])) \quad \\
\text{else} \\
\quad \text{return } a
\]

How to merge?

Choose lowest from two lists, cross out, repeat.
More divide and conquer: mergesort.

Sort items in $n$ elt array: $A = [a_1, \ldots, a_n]$,
E.g., $A = [5, 6, 7, 9, 10, 2, 3\ldots]$.

**Mergesort(A)**

if (length(A) >1)  
    return
        (merge(mergesort(a[1],\ldots,a[n/2]),
                mergesort(a[n/2+1],\ldots,a[n]))
     
else
    return a

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: 3, 7, 8, 10, 11,\ldots  
Sorted Subarray 2: 4, 5, 9, 19, 20,\ldots  
\ldots
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).

**Mergesort(A)**

```
if (length(A) > 1)
    return
    (merge(mergesort(a[1],\ldots,a[n/2])),
    mergesort(a[n/2+1],\ldots,a[n]))
else
    return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: 3
\( \times \), 7, 8, 10, 11,\ldots

Sorted Subarray 2: 4, 5, 9, 19, 20,\ldots
3, , , ,
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3 \ldots] \).

**Mergesort(A)**

if (length(A) > 1)
    return
    (merge(mergesort(a[1],\ldots,a[n/2]),
           mergesort(a[n/2+1],\ldots,a[n])))
else
    return a

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: \( x, 7, 8, 10, 11, \ldots \)
Sorted Subarray 2: \( x, 5, 9, 19, 20, \ldots \)
3, 4, , ,
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),

E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).

**Mergesort(A)**

```plaintext
if (length(A) >1)
    return
    (merge(mergesort(a[1],\ldots,a[n/2]),
           mergesort(a[n/2+1],\ldots,a[n]))
else
    return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: \( \times, 7, 8, 10, 11, \ldots \)

Sorted SubArray 2: \( \times, \times, 9, 19, 20, \ldots \)

3, 4, 5, \ldots
More divide and conquer: mergesort.

Sort items in $n$ elt array: $A = [a_1, \ldots, a_n]$, 
E.g., $A = [5, 6, 7, 9, 10, 2, 3\ldots]$.

**Mergesort(A)**

```plaintext
if (length(A) >1) return
  (merge(mergesort(a[1],\ldots,a[n/2]),
        mergesort(a[n/2+1],\ldots,a[n])))
else return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

Sorted SubArray 1: $\underline{3}, \underline{X}, 8, 10, 11,\ldots$

Sorted Subarray 2: $\underline{4}, \underline{X}, 9, 19, 20,\ldots$

$3, 4, 5, 7,$
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).

**Mergesort(A)**

```plaintext
if (length(A) > 1)
    return (merge(mergesort(a[1],...,a[n/2]),
                  mergesort(a[n/2+1],...,a[n])))
else
    return a
```

How to merge?

Choose lowest from two lists, cross out, repeat.

<table>
<thead>
<tr>
<th>Sorted SubArray 1:</th>
<th>Sorted Subarray 2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5, 6, 7, 9, 10, 11,...</td>
<td>4, 5, 9, 19, 20,...</td>
</tr>
<tr>
<td>3, 4, 5, 7, 8</td>
<td></td>
</tr>
</tbody>
</table>
More divide and conquer: mergesort.

Sort items in \( n \) elt array: \( A = [a_1, \ldots, a_n] \),
E.g., \( A = [5, 6, 7, 9, 10, 2, 3\ldots] \).

**Mergesort(A)**

\[
\text{if (length(A) > 1)} \\
\text{return} \\
(\text{merge(mergesort(a[1], \ldots, a[n/2]),} \\
\text{mergesort(a[n/2+1], \ldots, a[n])))} \\
\text{else} \\
\text{return a}
\]

How to merge?

Choose lowest from two lists, cross out, repeat.

<table>
<thead>
<tr>
<th>Sorted</th>
<th>SubArray 1: 3, 4, 5, 7, 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted</td>
<td>Subarray 2: 3, 4, 5, 7, 8</td>
</tr>
</tbody>
</table>
Mergesort: running time analysis

**Mergesort(A)**

```plaintext
if (length(A) > 1)
    return
    (merge(mergesort( a[1],...,a[n/2]),
             mergesort( a[n/2+1],...,a[n]))
else
    return a
```
Mergesort: running time analysis

Mergesort(A)
    if \( \text{length}(A) > 1 \)
        return
        (merge(mergesort( a[1],...,a[n/2]),
                mergesort( a[n/2+1],...,a[n])))
    else
        return a

Split: \( O(n) \) time
Mergesort: running time analysis

**Mergesort(A)**

if (length(A) > 1)
  return
  (merge(mergesort( a[1], ..., a[n/2]),
          mergesort( a[n/2+1], ..., a[n])))
else
  return a

Split: $O(n)$ time
Could be $O(1)$, e.g., **mergeSort(A,start,finish)**.
Mergesort: running time analysis

**Mergesort**(*A*)

```plaintext
if (length(*A*) > 1)
    return
    (merge(mergesort(*a[1]*,...,*a[n/2]*)
        mergesort(*a[n/2+1]*,...,*a[n]*)
    
else
    return *a*
```

**Split**: $O(n)$ time
Could be $O(1)$, e.g., `MergeSort(A,start,finish)`.

**Merge**: 

Mergesort: running time analysis

**Mergesort(A)**

\[
\text{if } (\text{length}(A) > 1) \\
\quad \text{return} \\
\quad (\text{merge}(\text{mergesort}(a[1],...,a[n/2]), \\
\quad \text{mergesort}(a[n/2+1],...,a[n]))) \\
\text{else} \\
\quad \text{return } a
\]

Split: \(O(n)\) time

Could be \(O(1)\), e.g., \texttt{MergeSort(A,start,finish)}.

Merge: each element in output takes one comparision
Mergesort: running time analysis

Mergesort(A)
  if (length(A) > 1)
    return
    (merge(mergesort( a[1],...,a[n/2]),
            mergesort( a[n/2+1],...,a[n])))
  else
    return a

Split: $O(n)$ time
  Could be $O(1)$, e.g., MergeSort(A,start,finish).
Merge: each element in output takes one comparison : $O(n)$. 
Mergesort: running time analysis

\textbf{Mergesort(A)}

\begin{verbatim}
if (length(A) > 1)
  return
  (merge(mergesort(a[1],...,a[n/2]),
      mergesort(a[n/2+1],...,a[n]))
else
  return a
\end{verbatim}

Split: \textbf{O(n)} time

Could be \textbf{O(1)}, e.g., \textbf{MergeSort(A,start,finish)}.

Merge: each element in output takes one comparision: \textbf{O(n)}.

Recursive: \textbf{2} subproblems of size \textbf{n/2}.
Mergesort: running time analysis

Mergesort(A)
  if (length(A) >1)
    return
    (merge(mergesort( a[1],...,a[n/2]),
    mergesort( a[n/2+1],...,a[n])))
  else
    return a

Split: \(O(n)\) time
  Could be \(O(1)\), e.g., \texttt{MergeSort(A,start,finish)}.
Merge: each element in output takes one comparison : \(O(n)\).
Recursive: 2 subproblems of size \(n/2\).

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
Mergesort: running time analysis

Mergesort(A)
if (length(A) >1)
    return
    (merge(mergesort( a[1],...,a[n/2]),
     mergesort( a[n/2+1],...,a[n])))
else
    return a

Split: $O(n)$ time
    Could be $O(1)$, e.g., MergeSort(A,start,finish).
Merge: each element in output takes one comparision : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

Masters: $T(n) = aT(n/b) + O(n^d)$
    with $\log_b a = d$
Mergesort: running time analysis

Mergesort(A)
if (length(A) > 1)
  return
(m_merge(mergesort(a[1],...,a[n/2]),
           mergesort(a[n/2+1],...,a[n]))
else
  return a

Split: $O(n)$ time
  Could be $O(1)$, e.g., $\text{MergeSort}(A, \text{start}, \text{finish})$.
Merge: each element in output takes one comparison: $O(n)$.
Recursive: 2 subproblems of size $n/2$.

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]

Masters: \( T(n) = aT(n/b) + O(n^d) \)
with \( \log_b a = d \implies O(n^d \log_b n) \)
Mergesort: running time analysis

Mergesort(A)
  if (length(A) >1)
    return
      (merge(mergesort( a[1],...,a[n/2]),
    mergesort( a[n/2+1],...,a[n]))
  else
    return a

Split: \( O(n) \) time
  Could be \( O(1) \), e.g., MergeSort(A,start,finish).
Merge: each element in output takes one comparision : \( O(n) \).
Recursive: 2 subproblems of size \( n/2 \).

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]

Masters: \( T(n) = aT(n/b) + O(n^d) \)
  with \( \log_b a = d \) \( \Longrightarrow \) \( O(n^d \log_b n) \)
Mergesort: running time analysis

Mergesort(A)
if (length(A) >1)
  return
    (merge(mergesort( a[1],...,a[n/2]),
    mergesort( a[n/2+1],...,a[n]))
else
  return a

Split: $O(n)$ time
  Could be $O(1)$, e.g., MergeSort(A,start,finish).
Merge: each element in output takes one comparison : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

$T(n) = 2T(n/2) + O(n)$.

Masters: $T(n) = aT(n/b) + O(n^d)$
  with $\log_b a = d \implies O(n^d \log_b n)$

Apply Masters:
  $a = 2, b = 2, d = 1$
Mergesort: running time analysis

**Mergesort(A)**

```plaintext
if (length(A) > 1)
    return
    (merge(mergesort( a[1],...,a[n/2]),
        mergesort( a[n/2+1],...,a[n]))
else
    return a
```

Split: \( O(n) \) time
Could be \( O(1) \), e.g., \textbf{MergeSort(A,start,finish)}.

Merge: each element in output takes one comparision : \( O(n) \).

Recursive: 2 subproblems of size \( n/2 \).

\[
T(n) = 2T\left(\frac{n}{2}\right) + O(n).
\]

Masters: \( T(n) = aT(n/b) + O(n^d) \)
with \( \log_b a = d \implies O(n^d \log_b n) \)

Apply Masters:
\( a = 2, b = 2, d = 1 \implies \log_2 2 = 1 \)
Mergesort: running time analysis

**Mergesort**(*A*)

if (length(*A*) >1)
    return
    (merge(mergesort( *a*[1],...,*a*[*n*/2]*)),
     mergesort( *a*[*n*/2+1],...,*a*[*n*]))
else
    return *a*

Split: \(O(n)\) time
Could be \(O(1)\), e.g., **MergeSort**(*A*,start,finish).

Merge: each element in output takes one comparison : \(O(n)\).

Recursive: 2 subproblems of size \(n/2\).

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]

Masters: \( T(n) = aT\left(\frac{n}{b}\right) + O(n^d) \)
with \( \log_b a = d \) \(\implies\) \( O(n^d \log_b n) \)

Apply Masters:
\(a = 2, b = 2, d = 1 \implies \log_2 2 = 1 \implies T(n) = O(n \log n).\)
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 | ··· |··· |··· |··· |··· |··· |··· |··· |

Rinse.
Repeat.

Each pass through queue:
Each element touched once.
$O(n)$ time.
Each pass halves number of lists.
$= \Rightarrow O(\log n)$ passes
$= \Rightarrow O(n \log n)$ time
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

8  3  5  9  ·······

Merge first two lists, put in queue (at end).

5  9  ·······  3, 8
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

\[
\begin{array}{c}
 8 & 3 & 5 & 9 & \cdots \\
\end{array}
\]

Merge first two lists, put in queue (at end).

\[
\begin{array}{c}
 5 & 9 & \cdots & 3,8 \\
\end{array}
\]

Rinse.
Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

```
8  3  5  9  ··· ···
```

Merge first two lists, put in queue (at end).

```
5  9  ··· ···  3,8
```

Rinse. Repeat.
Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 | ⋯ ⋯ ⋯ |

Merge first two lists, put in queue (at end).

| 5 | 9 | ⋯ ⋯ ⋯ | 3,8 |

Rinse. Repeat.

| ⋯ ⋯ ⋯ | 3,8 | 5,9 |
Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 |    |

Merge first two lists, put in queue (at end).

| 5 | 9 |    | 3, 8 |

Rinse. Repeat.

|    |    | 3, 8 | 5, 9 |

And next pass through queue...
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

<table>
<thead>
<tr>
<th>8</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Merge first two lists, put in queue (at end).

<table>
<thead>
<tr>
<th>5</th>
<th>9</th>
<th></th>
<th></th>
<th>3</th>
<th>8</th>
</tr>
</thead>
</table>

Rinse. Repeat.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>3,8</th>
<th></th>
<th>5,9</th>
</tr>
</thead>
</table>

And next pass through queue...

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>3,5,8,9</th>
<th></th>
<th></th>
</tr>
</thead>
</table>
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

```
8 3 5 9  ·······
```

Merge first two lists, put in queue (at end).

```
5 9  ·····  3,8
```

Rinse. Repeat.

```
······  3,8  5,9
```

And next pass through queue...

```
······  3,5,8,9
```

Each pass through queue:
Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 | ... | ...

Merge first two lists, put in queue (at end).

| 5 | 9 | ... | ... | 3, 8 |

Rinse. Repeat.

| ... | 3, 8 | 5, 9 |

And next pass through queue...

| ... | 3, 5, 8, 9 |

Each pass through queue: each element touched once.
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 | ... | ... |

Merge first two lists, put in queue (at end).

| 5 | 9 | ... | ... | 3,8 |

Rinse. Repeat.

| ... | 3,8 | 5,9 |

And next pass through queue...

| ... | 3,5,8,9 |

Each pass through queue: each element touched once. $O(n)$ time.
Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 | · · · · · |

Merge first two lists, put in queue (at end).

| 5 | 9 | · · · · · | 3, 8 |

Rinse. Repeat.

| · · · · · | 3, 8 | 5, 9 |

And next pass through queue...

| · · · · · | 3, 5, 8, 9 |

Each pass through queue: each element touched once. $O(n)$ time. Each pass halves number of lists.
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 | · · · · · |

Merge first two lists, put in queue (at end).

| 5 | 9 | · · · · · | 3, 8 |

Rinse. Repeat.

| · · · · · | 3, 8 | 5, 9 |

And next pass through queue...

| · · · · · | 3, 5, 8, 9 |

Each pass through queue: each element touched once. \( O(n) \) time.
Each pass halves number of lists.

\[ \implies O(\log n) \text{ passes} \]
Check it out...

Iterative Mergesort: Bottom up, use of queues.
Make each element into list and put lists in queue.

| 8 | 3 | 5 | 9 |      |

Merge first two lists, put in queue (at end).

| 5 | 9 |      | 3,8 |

Rinse. Repeat.

|      | 3,8 | 5,9 |

And next pass through queue...

|      | 3,5,8,9 |

Each pass through queue: each element touched once. $O(n)$ time.
Each pass halves number of lists.

$\implies O(\log n)$ passes $\implies O(n \log n)$ time
Sorting lower bound.

Can we do better?
Can we do better?

Comparison sorting algorithm only compares numbers.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort?
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”....
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first els and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”.... Repeat in each bucket with next characters.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”....
Repeat in each bucket with next characters.
Looks at characters...
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”.... Repeat in each bucket with next characters. Looks at characters... or looks at “bits”.
Can we do better?

Comparison sorting algorithm only compares numbers. Does not look at bits only uses result of comparison.

Merge:
Compare two first elts and then output first.

Comparison sort? Yes.

“Radix” Sort.
Bucket according to whether begins with “A”, “B”.... Repeat in each bucket with next characters.
Looks at characters... or looks at “bits”.
Not a comparision sort!
Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.
Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permutation of $[1, \ldots, n]$. 
Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permutation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$. 

Sorting lower bound.
Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permutation of [1, ..., $n$].
Output: 8, $n - 8$, ..., 15.

How many possible outputs?
Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permutation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs? $n!$
Sorting lower bound.

Thm: Comparison sort requires $\Omega(n \log n)$ comparisons.

**Proof idea:** Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permutation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs? $n!$
Algorithm must be about to output any of $n!$ permutations.
Thm: Comparison sort requires $\Omega(n\log n)$ comparisons.

Proof idea: Input: $a_1, a_2, \ldots, a_n$
Possible Output: $a_8, a_{n-8}, \ldots, a_{15}$
Represent output as permutation of $[1, \ldots, n]$.
Output: $8, n-8, \ldots, 15$.

How many possible outputs? $n!$

Algorithm must be about to output any of $n!$ permutations.
Algorithm must output just 1 permutation at termination.
Algorithm must be able to output any of $n!$ permutations.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination. Algorithm as tree of comparisons.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
  After a sequence of comparisons get to termination or 1 permutation.
Algorithm must be able to output any of $n!$ permutations.
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm

$S_1 \cup S_2 = S \Rightarrow \max(|S_1|, |S_2|) \geq |S| / 2.$
Each comparison divides possible outputs by at most 2.
Need at least $\log_2(n!)$ comparisons to get to just 1 permutation.

$n! \geq (n/e)^n \Rightarrow \log n! = \Omega(n \log n).$
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination. Algorithm as tree of comparisons. After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm

Example:
Sorting lower bound: ...proof

Algorithm must be able to output any of \( n! \) permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons. After a sequence of comparisons get to termination or 1 permutation.

\( S \) is set of possible permutations at some point in Algorithm

Example: After no comparisons, any output is possible.
Sorting lower bound: ...proof

Algorithm must be able to output any of \( n! \) permutations. 
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
   After a sequence of comparisons get to termination or 1 permutation.

\( S \) is set of possible permutations at some point in Algorithm
   Example: After no comparisons, any output is possible.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons. After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm

Example: After no comparisons, any output is possible.

Do some comparison: $a_i > a_j$?
Algorithm must be able to output any of $n!$ permutations.
Algorithm must output just 1 permutation at termination.
Algorithm as tree of comparisons.
After a sequence of comparisons get to termination or 1 permutation.
$S$ is set of possible permutations at some point in Algorithm.
Example: After no comparisons, any output is possible.
Do some comparison: $a_i > a_j$?
Algorithm must be able to output any of $n!$ permutations.
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm
Example: After no comparisons, any output is possible.

Do some comparison: $a_i > a_j$?

If Yes, Alg “could” return subset of permutations: $S_1$. 
Algorithm must be able to output any of $n!$ permutations. 
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons. 
After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm
Example: After no comparisons, any output is possible.

Do some comparision: $a_i > a_j$?

If Yes, Alg “could” return subset of permutations: $S_1$.
If No, Alg “could” return subset of permutations: $S_2$. 

$S_1 \cup S_2 = S \Rightarrow \max(|S_1|, |S_2|) \geq |S|/2.$

Each comparision divides possible outputs by at most 2.
Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.

$n! \geq (n \ln n)^n \Rightarrow \log n! = \Omega(n \log n)$.
Algorithm must be able to output any of \( n! \) permutations.
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
After a sequence of comparisons get to termination or 1 permutation.

\( S \) is set of possible permutations at some point in Algorithm
Example: After no comparisons, any output is possible.

Do some comparison: \( a_i > a_j \)?

If Yes, Alg “could” return subset of permutations: \( S_1 \).
If No, Alg “could” return subset of permutations: \( S_2 \).

\[ S_1 \cup S_2 = S \]
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons. After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm

Example: After no comparisons, any output is possible.

Do some comparison: $a_i > a_j$?

If Yes, Alg “could” return subset of permutations: $S_1$.
If No, Alg “could” return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2.$
Sorting lower bound: ...proof

Algorithm must be able to output any of \( n! \) permutations.
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
   After a sequence of comparisons get to termination or 1 permutation.

\( S \) is set of possible permutations at some point in Algorithm
   Example: After no comparisons, any output is possible.

Do some comparison: \( a_i > a_j \)?
   If Yes, Alg “could” return subset of permutations: \( S_1 \).
   If No, Alg “could” return subset of permutations: \( S_2 \).

\( S_1 \cup S_2 = S \iff \max(|S_1|, |S_2|) \geq |S|/2. \)

Each comparison divides possible outputs by at most 2.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons. After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm

Example: After no comparisons, any output is possible.

Do some comparison: $a_i > a_j$?

If Yes, Alg “could” return subset of permutations: $S_1$.
If No, Alg “could” return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.

Each comparison divides possible outputs by at most 2. Need at least $\log_2(n!)$ comparisons to get to just 1 permutation.
Algorithm must be able to output any of $n!$ permutations.
Algorithm must output just 1 permutation at termination.
Algorithm as tree of comparisons.
   After a sequence of comparisons get to termination or 1 permutation.
$S$ is set of possible permutations at some point in Algorithm
   Example: After no comparisons, any output is possible.
Do some comparison: $a_i > a_j$?
   If Yes, Alg “could” return subset of permutations: $S_1$.
   If No, Alg “could” return subset of permutations: $S_2$.
$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.
Each comparison divides possible outputs by at most 2.
Need at least $\log_2(n!)$ comparisons to get to just 1 permutation.
...to get to termination.
Algorithm must be able to output any of $n!$ permutations.
Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
  After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm
  Example: After no comparisons, any output is possible.

Do some comparison: $a_i > a_j$?
  If Yes, Alg “could” return subset of permutations: $S_1$.
  If No, Alg “could” return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.

Each comparison divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisons to get to just 1 permutation.
...to get to termination.

$n! \geq \left(\frac{n}{e}\right)^n$
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
    After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm
    Example: After no comparisons, any output is possible.

Do some comparision: $a_i > a_j$?

    If Yes, Alg “could” return subset of permutations: $S_1$.
    If No, Alg “could” return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation. ...

\[ n! \geq \left( \frac{n}{e} \right)^n \implies \log n! \]
Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
   After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm
   Example: After no comparisons, any output is possible.

Do some comparison: $a_i > a_j$?
   If Yes, Alg “could” return subset of permutations: $S_1$.
   If No, Alg “could” return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
   ...to get to termination.

$n! \geq \left(\frac{n}{e}\right)^n \implies \log n! = \Omega(\log(n^n))$
Sorting lower bound: ...proof

Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons. After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm

Example: After no comparisons, any output is possible.

Do some comparision: $a_i > a_j$?

If Yes, Alg "could" return subset of permutations: $S_1$.
If No, Alg "could" return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisons to get to just 1 permutation.
...to get to termination.

$n! \geq \left(\frac{n}{e}\right)^n \implies \log n! = \Omega(\log(n^n)) = \Omega(n \log n)$.
Algorithm must be able to output any of $n!$ permutations. Algorithm must output just 1 permutation at termination.

Algorithm as tree of comparisons.
After a sequence of comparisons get to termination or 1 permutation.

$S$ is set of possible permutations at some point in Algorithm
Example: After no comparisons, any output is possible.

Do some comparision: $a_i > a_j$?

If Yes, Alg “could” return subset of permutations: $S_1$.
If No, Alg “could” return subset of permutations: $S_2$.

$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2.$

Each comparision divides possible outputs by at most 2.

Need at least $\log_2(n!)$ comparisions to get to just 1 permutation.
...to get to termination.

$n! \geq \left(\frac{n}{e}\right)^n \implies \log n! = \Omega(\log(n^n)) = \Omega(n \log n).$
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!)= \Omega(n \log n)$. Can we do better than mergesort? Yes? No. For comparison sort. (Recall from 61b: radix sort may be faster: $O(n)$.) A research area: “bit complexity” versus “word complexity”.
Either the set of permutations $S_1$ or $S_2$ is larger.
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half.
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#\text{permutations}))$.
Either the set of permutations $S_1$ or $S_2$ is larger.
One must be at least half.
Depth must be $\Omega(\log(\#permutations)) = \Omega(\log n!)$. 
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half.
Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n \log n)$. 

Can we do better than mergesort? Yes? No? No. For comparison sort. (Recall from 61b: radix sort may be faster: $O(n)$.) 

A research area: “bit complexity” versus “word complexity.”
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

Can we do better than mergesort?
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n \log n)$. Can we do better than mergesort? Yes?
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half.
Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.
Can we do better than mergesort? Yes? No?
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half.
Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.
Can we do better than mergesort? Yes? No?
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n \log n)$. Can we do better than mergesort? Yes? No? 
No.
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n\log n)$.

Can we do better than mergesort? Yes? No? No. For comparison sort.
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half.

Depth must be $\Omega(\log(\#\text{permutations})) = \Omega(\log n!) = \Omega(n\log n)$.

Can we do better than mergesort? Yes? No?

No. For comparison sort.

(Recall from 61b: radix sort may be faster: $O(n)$.)
Either the set of permutations $S_1$ or $S_2$ is larger. One must be at least half. Depth must be $\Omega(\log(\#permutations)) = \Omega(\log n!) = \Omega(n\log n)$. Can we do better than mergesort? Yes? No? No. For comparison sort. (Recall from 61b: radix sort may be faster: $O(n)$.) A research area: “bit complexity” versus “word complexity”.
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$. 
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)
Median finding.

Find the median element of a set of elements: \( a_1, \ldots, a_n \).
Median is value, \( v \), where \( \frac{n}{2} \) elts are less than \( v \) (if \( n \) is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200
Why so different?
Bill Gates and Jeff Bezos.
Why use average?
Find average?
Compute \( \sum_{i} a_i \n \)
O(\( n \)) time.
Compute median?
Sort to get \( s_1, \ldots, s_n \).
Output element \( s_n / 2 + 1 \).
O(\( n \log n \)) time.
Better algorithm?
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200
Median finding.

Find the median element of a set of elements: \( a_1, \ldots, a_n \).
Median is value, \( \nu \), where \( \frac{n}{2} \) elts are less than \( \nu \) (if \( n \) is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different?
Median finding.

Find the median element of a set of elements: \( a_1, \ldots, a_n \).
Median is value, \( v \), where \( \frac{n}{2} \) elts are less than \( v \) (if \( n \) is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos.
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.
Why use average?

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average?
Median finding.

Find the median element of a set of elements: \( a_1, \ldots, a_n \).

Median is value, \( v \), where \( \frac{n}{2} \) elts are less than \( v \) (if \( n \) is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average? Compute \( \frac{\sum_i a_i}{n} \).
Median finding.

Find the median element of a set of elements: \( a_1, \ldots, a_n \).
Median is value, \( v \), where \( \frac{n}{2} \) elts are less than \( v \) (if \( n \) is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1\%, perhaps.

Why use average?

Find average? Compute \( \frac{\sum a_i}{n} \).

\( O(n) \) time.
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Compute median?
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.

Median is value, $v$, where $\frac{n}{2}$ elts are less than $v$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum a_i}{n}$.

$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$. 

Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$.
Median is value, $\nu$, where $\frac{n}{2}$ elts are less than $\nu$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.
$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$. Output element $s_{n/2+1}$. 
Median finding.

Find the median element of a set of elements: $a_1, \ldots, a_n$. Median is value, $v$, where $\frac{n}{2}$ elts are less than $v$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average? Compute $\frac{\sum_i a_i}{n}$.
$O(n)$ time.

Compute median? Sort to get $s_1, \ldots, s_n$. Output element $s_{n/2+1}$. $O(n \log n)$ time.
Find the median element of a set of elements: $a_1, \ldots, a_n$. Median is value, $v$, where $\frac{n}{2}$ elts are less than $v$ (if $n$ is odd.)

Versus Average?

Average household income (2004): $70,700
Median household income (2004): $43,200

Why so different? Bill Gates and Jeff Bezos. The 1%, perhaps.

Why use average?

Find average? Compute $\sum_i a_i / n$.

$O(n)$ time.

Compute median? Sort to get $s_1, \ldots s_n$. Output element $s_{n/2+1}$.

$O(n \log n)$ time.

Better algorithm?
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \), select the \( k \)th smallest element.

Median: select \( \lceil n / 2 \rceil + 1 \) elt.

Example. \( k = 7 \) for items \{11, 48, 5, 21, 2, 15, 17, 19, 15\}.

Output?

(A) 19  
(B) 15  
(C) 21  

???
Solve a harder Problem: Selection.

For a set of $n$ items $S$. 
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.
Solve a harder Problem: Selection.

For a set of $n$ items $S$. Select $k$th smallest element.

Median: select $\lceil\frac{n}{2}\rceil + 1$ elt.
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

Example.
\( k = 7 \) for items \( \{11, 48, 5, 21, 2, 15, 17, 19, 15\} \)
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

Example.
\( k = 7 \) for items \( \{11, 48, 5, 21, 2, 15, 17, 19, 15\} \)
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lceil n/2 \rceil + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Output?
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

Example.
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$

Output?

(A) 19
(B) 15
(C) 21
Solve a harder Problem: Selection.

For a set of $n$ items $S$. 
Select $k$th smallest element. 
Median: select $\lfloor n/2 \rfloor + 1$ elt. 

Example. 
$k = 7$ for items $\{11, 48, 5, 21, 2, 15, 17, 19, 15\}$ 
Output? 

(A) 19 
(B) 15 
(C) 21 

????
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \), select \( k \)th smallest element.

**Median:** select \( \left\lfloor \frac{n}{2} \right\rfloor + 1 \) elt.

Select \((k, S)\):

\[ k = 7 \]

\[ S: \{11, 48, 5, 21, 2, 15, 17, 19, 15\} \]

**Base Case:**

\( k = 1 \) and \(|S| = 1\), return elt.

Choose rand. pivot elt \( b \) from \( A \).

\[ v = 15 \]

Form \( S_L \) containing all elts < \( v \):

\[ S_L: \{11, 5, 2\} \]

Form \( S_v \) containing all elts = \( v \):

\[ S_v: \{15, 15\} \]

Form \( S_R \) containing all elts > \( v \):

\[ S_R: \{48, 21, 17, 19\} \]

If \( k \leq |S_L| \), Select \((k, S_L)\).

\[ 7 \leq 3? \]

elseif \( k \leq |S_L| + |S_v| \), return \( v \).

\[ 7 \leq 5? \]

else Select \((k - |S_L| - |S_v|, S_R)\).

Select \((2, [48, 21, 17, 19])\)

Will eventually return 19, which is 7th element of list.

**Correctness:**

Induction.

Idea: Subroutine returns correct answer, and so will I.

Base case is good. Subroutine calls..by design.
Solve a harder Problem: Selection.

For a set of $n$ items $S$. 
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**$(k, S)$: $k = 7$  \hspace{1cm} S : 11, 48, 5, 21, 2, 15, 17, 19, 15
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lceil n/2 \rceil + 1$ elt.

**Select**$(k, S)$: $k = 7$  

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. pivot elt $b$ from $A$.

$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

$v = 15$
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**$(k, S)$: $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.

Form $S_L$ containing all elts $< v$

$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

$\nu = 15$

Will eventually return 19, which is 7th element of list.
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lceil n/2 \rceil + 1$ elt.

**Select**($k$, $S$): $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.

Form $S_L$ containing all els $< v$

Form $S_v$ containing all els $= v$

Form $S_R$ containing all els $> v$

If $k \leq |S_L|$, Select ($k$, $S_L$).

elseif $k \leq |S_L| + |S_v|$, return $v$.

else Select ($k - |S_L| - |S_v|$, $S_R$)

Will eventually return 19, which is 7th element of list.

Correctness:
Induction.
Idea: Subroutine returns correct answer, and so will I.

Base case is good. Subroutine calls..by design.
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lceil n/2 \rceil + 1$ elt.

**Select**$(k, S)$: $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.

Form $S_L$ containing all elts $< v$

Form $S_v$ containing all elts $= v$

$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

$v = 15$

$S_L : 11, 5, 2$

Will eventually return 19, which is 7th element of list.

Correctness:
Induction.
Idea: Subroutine returns correct answer, and so will $I$!
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

\[
\text{Select}(k, S): \quad k = 7
\]

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.

 Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts \( < v \)
Form \( S_v \) containing all elts \( = v \)

\[
S : 11, 48, 5, 21, 2, 15, 17, 19, 15
\]

\[
S_L : 11, 5, 2
\]

\[
S_v : 15, 15
\]
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

**Select** \((k, S)\): \( k = 7 \)

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.

Choose rand. pivot elt \( b \) from \( A \).

Form \( S_L \) containing all elts < \( v \)

Form \( S_v \) containing all elts = \( v \)

Form \( S_R \) containing all elts > \( v \)

\[ S : 11, 48, 5, 21, 2, 15, 17, 19, 15 \]

\[ v = 15 \]

\[ S_L : 11, 5, 2 \]

\[ S_v : 15, 15 \]

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I.

Base case is good.
Subroutine calls..by design.
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lceil n/2 \rceil + 1 \) elt.

**Select** \((k, S)\): \( k = 7 \)  

\( S : 11, 48, 5, 21, 2, 15, 17, 19, 15 \)  

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.
Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts < \( v \)  
Form \( S_v \) containing all elts = \( v \)  
Form \( S_R \) containing all elts > \( v \)

\( v = 15 \)  
\( S_L : 11, 5, 2 \)  
\( S_v : 15, 15 \)  
\( S_R : 48, 21, 17, 19 \)
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**($k, S$): $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.
Form $S_L$ containing all elts $< v$
Form $S_V$ containing all elts $= v$
Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select($k, S_L$).

$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$
$\nu = 15$
$S_L : 11, 5, 2$
$S_V : 15, 15$
$S_R : 48, 21, 17, 19$

$7 \leq 3?$
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

\textbf{Select}(k, S): \quad k = 7

\text{Base Case: } k = 1 \text{ and } |S| = 1, \text{ return elt.}

Choose \text{rand. pivot} elt \( b \) from \( A \).
Form \( S_L \) containing all elts < \( v \)
Form \( S_v \) containing all elts = \( v \)
Form \( S_R \) containing all elts > \( v \)

If \( k \leq |S_L| \), \text{Select}(k, S_L).
elseif \( k \leq |S_L| + |S_v| \), return \( v \).
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

\[
\text{Select}(k, S):
\]
\[
k = 7
\]

**Base Case:** \( k = 1 \) and \( |S| = 1 \), return elt.

Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts \( < v \)
Form \( S_v \) containing all elts \( = v \)
Form \( S_R \) containing all elts \( > v \)

If \( k \leq |S_L| \), Select\((k, S_L)\).
elseif \( k \leq |S_L| + |S_v| \), return \( v \).
else Select\((k - |S_L| - |S_v|, S_R)\)

\[
S : 11, 48, 5, 21, 2, 15, 17, 19, 15
\]

\[
S : 11, 48, 5, 21, 2, 15, 17, 19, 15
\]

\[
\nu = 15
\]

\[
S_L : 11, 5, 2
\]

\[
S_L : 11, 5, 2
\]

\[
S_V : 15, 15
\]

\[
S_V : 15, 15
\]

\[
S_R : 48, 21, 17, 19
\]

\[
S_R : 48, 21, 17, 19
\]
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lceil n/2 \rceil + 1 \) elt.

**Select** \((k, S)\): \( k = 7 \)

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.

Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts \(< v \)
Form \( S_V \) containing all elts \( = v \)
Form \( S_R \) containing all elts \( > v \)

If \( k \leq |S_L| \), Select\((k, S_L)\).
elseif \( k \leq |S_L| + |S_V| \), return \( v \).
else Select\((k - |S_L| - |S_V|, S_R)\)

\( S : 11, 48, 5, 21, 2, 15, 17, 19, 15 \)
\( v = 15 \)
\( S_L : 11, 5, 2 \)
\( S_V : 15, 15 \)
\( S_R : 48, 21, 17, 19 \)

Will eventually return 19, which is 7th element of list.

Correctness:
Induction.
Idea: Subroutine returns correct answer, and so will I!
Base case is good.
Subroutine calls by design.
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**$(k, S)$: $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.

Form $S_L$ containing all elts $< v$
Form $S_v$ containing all elts $= v$
Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select$(k, S_L)$.
elseif $k \leq |S_L| + |S_v|$, return $v$.
else Select$(k - |S_L| - |S_v|, S_R)$

Will eventually return 19, which is 7th element of list.
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

**Select**\((k, S)\): \( k = 7 \)

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.
Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts \( < v \)
Form \( S_V \) containing all elts \( = v \)
Form \( S_R \) containing all elts \( > v \)

If \( k \leq |S_L| \), Select\((k, S_L)\).
elseif \( k \leq |S_L| + |S_V| \), return \( v \).
else Select\((k − |S_L| − |S_V|, S_R)\)

Will eventually return 19, which is 7th element of list.

Correctness:
Solve a harder Problem: Selection.

For a set of $n$ items $S$. Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**$(k, S)$:  
$k = 7$  
$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.  
$v = 15$

Form $S_L$ containing all elts $< v$  
$S_L : 11, 5, 2$

Form $S_v$ containing all elts $= v$  
$S_v : 15, 15$

Form $S_R$ containing all elts $> v$  
$S_R : 48, 21, 17, 19$

If $k \leq |S_L|$, Select$(k, S_L)$.  
$7 \leq 3?$

elseif $k \leq |S_L| + |S_v|$, return $v$.  
$7 \leq 5?$

else Select$(k − |S_L| − |S_v|, S_R)$  
Select$(2, [48, 21, 17, 19])$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.  
Idea: Subroutine returns correct answer,
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**($k, S$): $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.

Form $S_L$ containing all elts $< v$

Form $S_v$ containing all elts $= v$

Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select($k, S_L$).

elseif $k \leq |S_L| + |S_v|$, return $v$.

else Select($k - |S_L| - |S_v|, S_R$)

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lceil n/2 \rceil + 1 \) elt.

**Select(\( k, S \)):** \( k = 7 \)

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.

Choose rand. pivot elt \( b \) from \( A \).

Form \( S_L \) containing all elts \( < v \)
Form \( S_v \) containing all elts \( = v \)
Form \( S_R \) containing all elts \( > v \)

If \( k \leq |S_L| \), Select(\( k, S_L \)).
elseif \( k \leq |S_L| + |S_v| \), return \( v \).
else Select(\( k - |S_L| - |S_v|, S_R \))

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

**Select**\((k, S)\): \( k = 7 \) \( S : 11, 48, 5, 21, 2, 15, 17, 19, 15 \)
Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.
Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts < \( v \) \( S_L : 11, 5, 2 \)
Form \( S_V \) containing all elts = \( v \) \( S_V : 15, 15 \)
Form \( S_R \) containing all elts > \( v \) \( S_R : 48, 21, 17, 19 \)

If \( k \leq |S_L| \), Select\((k, S_L)\). \( 7 \leq 3 \)?
elseif \( k \leq |S_L| + |S_V| \), return \( v \). \( 7 \leq 5 \)?
else Select\((k − |S_L| − |S_V|, S_R)\) \( \text{Select}(2, [48, 21, 17, 19]) \)

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will
Solve a harder Problem: Selection.

For a set of \( n \) items \( S \).
Select \( k \)th smallest element.

Median: select \( \lfloor n/2 \rfloor + 1 \) elt.

**Select**\((k, S)\): \( k = 7 \)

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.
Choose rand. pivot elt \( b \) from \( A \).
Form \( S_L \) containing all elts < \( v \)
Form \( S_v \) containing all elts = \( v \)
Form \( S_R \) containing all elts > \( v \)

If \( k \leq |S_L| \), Select\((k, S_L)\).
elseif \( k \leq |S_L| + |S_v| \), return \( v \).
else Select\((k − |S_L| − |S_v|, S_R)\)

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select($k, S$):**

$k = 7$

- Base Case: $k = 1$ and $|S| = 1$, return elt.
- Choose rand. pivot elt $b$ from $A$.
- Form $S_L$ containing all elts $< v$
- Form $S_V$ containing all elts $= v$
- Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select($k, S_L$).
elseif $k \leq |S_L| + |S_V|$, return $v$.
else Select($k - |S_L| - |S_V|, S_R$)

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I!
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**$(k, S)$: $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. pivot elt $b$ from $A$.
Form $S_L$ containing all elts $< v$
Form $S_v$ containing all elts $= v$
Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select$(k, S_L)$.
elseif $k \leq |S_L| + |S_v|$, return $v$.
else Select$(k - |S_L| - |S_v|, S_R)$

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I!
Base case is good.
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lfloor n/2 \rfloor + 1$ elt.

**Select**($k, S$): $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.

Choose rand. pivot elt $b$ from $A$.

Form $S_L$ containing all elts $< v$
Form $S_V$ containing all elts $= v$
Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select($k, S_L$).
elseif $k \leq |S_L| + |S_V|$, return $v$.
else Select($k - |S_L| - |S_V|, S_R$)

Will eventually return 19, which is 7th element of list.

Correctness: Induction.

Idea: Subroutine returns correct answer, and so will I!

Base case is good. Subroutine calls
Solve a harder Problem: Selection.

For a set of $n$ items $S$.
Select $k$th smallest element.

Median: select $\lceil n/2 \rceil + 1$ elt.

**Select**($k, S$): $k = 7$

Base Case: $k = 1$ and $|S| = 1$, return elt.
Choose rand. pivot elt $b$ from $A$.
Form $S_L$ containing all elts $< v$
Form $S_v$ containing all elts $= v$
Form $S_R$ containing all elts $> v$

If $k \leq |S_L|$, Select($k, S_L$).
elseif $k \leq |S_L| + |S_v|$, return $v$.
else Select($k - |S_L| - |S_v|, S_R$)

Will eventually return 19, which is 7th element of list.

Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I!
Base case is good. Subroutine calls ..by design.
The Induction.

Base Case: $k = 1$, $|S| = 1$. Trivial.
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

| \( S_L \) | \( S_v \) | \( S_R \) |
The Induction.

Base Case: $k = 1, |S| = 1$. Trivial.

| $S_L$ | $S_v$ | $S_R$ |

If $k \leq |S_L|$, Select($k, S_L$)
The Induction.

Base Case: $k = 1$, $|S| = 1$. Trivial.

<table>
<thead>
<tr>
<th>$S_L$</th>
<th>$S_v$</th>
<th>$S_R$</th>
</tr>
</thead>
</table>

If $k \leq |S_L|$, Select($k, S_L$)

$k$th element in first $|S_L|$ elts.
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

<table>
<thead>
<tr>
<th></th>
<th>( S_L )</th>
<th>( S_v )</th>
<th>( S_R )</th>
</tr>
</thead>
</table>

If \( k \leq |S_L| \), Select\((k, S_L)\)

\( k \)th element in first \(|S_L|\) elts.

Correct in all cases.
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

<table>
<thead>
<tr>
<th></th>
<th>( S_L )</th>
<th>( S_V )</th>
<th>( S_R )</th>
</tr>
</thead>
</table>

If \( k \leq |S_L| \), Select(\( k, S_L \))

- \( k \)th element in first \(|S_L|\) elts.
- \( k \)th elt of \( S \) is \( k \)th elt of \( S_L \)
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

| \( S_L \) | \( S_v \) | \( S_R \) |

If \( k \leq |S_L| \), Select\( (k, S_L) \)
- \( k \)th element in first \( |S_L| \) elts.
- \( k \)th elt of \( S \) is \( k \)th elt of \( S_L \)

elseif \( k \leq |S_L| + |S_v| \), return \( v \),
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

\[
\begin{array}{ccc}
S_L & S_v & S_R \\
\end{array}
\]

If \( k \leq |S_L| \), Select\((k, S_L)\)
- \( k \)th element in first \(|S_L|\) els.
- \( k \)th elt of \( S \) is \( k \)th elt of \( S_L \)

elseif \( k \leq |S_L| + |S_v| \), return \( v \),
- \( k \in [|S_L|, \ldots, |S_L| + |S_v|] \).

Correct in all cases.
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

<table>
<thead>
<tr>
<th>( S_L )</th>
<th>( S_v )</th>
<th>( S_R )</th>
</tr>
</thead>
</table>

If \( k \leq |S_L| \), Select\((k, S_L)\)

- \( k \)th element in first \(|S_L|\) elts.
- \( k \)th elt of \( S \) is \( k \)th elt of \( S_L \)

elseif \( k \leq |S_L| + |S_v| \), return \( v \),

\( k \in [|S_L|, \ldots, |S_L| + |S_v|] \).
The Induction.

Base Case: $k = 1$, $|S| = 1$. Trivial.

<table>
<thead>
<tr>
<th>$S_L$</th>
<th>$S_v$</th>
<th>$S_R$</th>
</tr>
</thead>
</table>

If $k \leq |S_L|$, Select($k$, $S_L$)
   $k$th element in first $|S_L|$ elts.
   $k$th elt of $S$ is $k$th elt of $S_L$
elseif $k \leq |S_L| + |S_v|$, return $v$,
   $k \in [|S_L|, \ldots, |S_L| + |S_v|]$.
   $k$th elt of $S$ is in $S_v$, all have value $v$
The Induction.

Base Case: $k = 1$, $|S| = 1$. Trivial.

<table>
<thead>
<tr>
<th>$S_L$</th>
<th>$S_v$</th>
<th>$S_R$</th>
</tr>
</thead>
</table>

If $k \leq |S_L|$, Select($k$, $S_L$)
- $k$th element in first $|S_L|$ elts.
- $k$th elt of $S$ is $k$th elt of $S_L$
elseif $k \leq |S_L| + |S_v|$, return $v$,
- $k \in [|S_L|, \ldots, |S_L| + |S_v|]$.
- $k$th elt of $S$ is in $S_v$, all have value $v$
else Select($k - |S_L| - |S_v|$, $S_R$)
The Induction.

Base Case: \( k = 1, |S| = 1. \) Trivial.

<table>
<thead>
<tr>
<th></th>
<th>( S_L )</th>
<th>( S_v )</th>
<th>( S_R )</th>
</tr>
</thead>
</table>

If \( k \leq |S_L| \), Select\( (k, S_L) \)
  \( k \)th element in first \( |S_L| \) elts.
  \( k \)th elt of \( S \) is \( k \)th elt of \( S_L \)

elseif \( k \leq |S_L| + |S_v| \), return \( v \),
  \( k \in [|S_L|, \ldots, |S_L| + |S_v|] \).
  \( k \)th elt of \( S \) is in \( S_v \), all have value \( v \)

else Select\( (k - |S_L| - |S_v|, S_R) \)
  \( k \)th element is in \( S_R \) and
  \( k \)th elt of \( S \) is \( k - |S_L| - |S_v| \) after elts of \( S_L \cup S_v \).
The Induction.

Base Case: \( k = 1, |S| = 1 \). Trivial.

<table>
<thead>
<tr>
<th>( S_L )</th>
<th>( S_v )</th>
<th>( S_R )</th>
</tr>
</thead>
</table>

If \( k \leq |S_L| \), Select\((k, S_L)\)
  
  \( k \)th element in first \(|S_L|\) elts.
  
  \( k \)th elt of \( S \) is \( k \)th elt of \( S_L \)

elseif \( k \leq |S_L| + |S_v| \), return \( v \),
  
  \( k \in [|S_L|, \ldots, |S_L| + |S_v|] \).
  
  \( k \)th elt of \( S \) is in \( S_v \), all have value \( v \)

else Select\((k − |S_L| − |S_v|, S_R)\)
  
  \( k \)th element is in \( S_R \) and
  
  \( k \)th elt of \( S \) is \( k − |S_L| − |S_v| \) after elts of \( S_L \cup S_v \).

Correct in all cases.
The Induction.

Base Case: $k = 1$, $|S| = 1$. Trivial.

<table>
<thead>
<tr>
<th></th>
<th>$S_L$</th>
<th>$S_v$</th>
<th>$S_R$</th>
</tr>
</thead>
</table>

If $k \leq |S_L|$, Select($k, S_L$)
   $k$th element in first $|S_L|$ elts.
   $k$th elt of $S$ is $k$th elt of $S_L$
elseif $k \leq |S_L| + |S_v|$, return $v$,
   $k \in [|S_L|, \ldots, |S_L| + |S_v|]$.
   $k$th elt of $S$ is in $S_v$, all have value $v$
else Select($k - |S_L| - |S_v|, S_R$)
   $k$th element is in $S_R$ and
   $k$th elt of $S$ is $k - |S_L| - |S_v|$ after elts of $S_L \cup S_v$.

Correct in all cases.
Selection: runtime.

Worst case runtime?

Let $k = n$.

Partition element is smallest every time.

Size of list decrease by 1 in each recursive call.

Time for partition is $O(i)$ time when $i$ elements.

$\Theta(n + (n - 1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)

Worse than sorting!

On average?
Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$. 
Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$
(B) $\Theta(n)$
(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.
Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$
(B) $\Theta(n)$
(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.
Size of list decrease by 1 in each recursive call.
Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$
(B) $\Theta(n)$
(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.
Size of list decrease by 1 in each recursive call.
Time for partition is $O(i)$ time when $i$ elements.
Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$
(B) $\Theta(n)$
(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.
Size of list decrease by 1 in each recursive call.
Time for partition is $O(i)$ time when $i$ elements.
$\Theta(n + (n - 1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)
Selection: runtime.

Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.
Size of list decrease by 1 in each recursive call.
Time for partition is $O(i)$ time when $i$ elements.
$\Theta(n + (n - 1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)
Worse than sorting!
Worst case runtime?

(A) $\Theta(n \log n)$

(B) $\Theta(n)$

(C) $\Theta(n^2)$

Let $k = n$.
Partition element is smallest every time.
Size of list decrease by 1 in each recursive call.
Time for partition is $O(i)$ time when $i$ elements.
$\Theta(n + (n - 1) + \cdots + 2 + 1) = \Theta(n^2)$ time. or (C)
Worse than sorting!
On average?
Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?
Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two

(B) three
Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two

(B) three

(C) Could go forever!
Flip a coin, what is average number of tosses to get a heads?

(A) two
(B) three
(C) Could go forever!

(A) ..
Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two
(B) three
(C) Could go forever!

(A) ..and (C)
Average time to get a heads?

Flip a coin, what is average number of tosses to get a heads?

(A) two

(B) three

(C) Could go forever!

(A) ..and (C) (but not relevant.)
Expected (average) Time?

Lemma:

Expected number of coin tosses to get a heads is 2.

Proof:

\[ E[X] = 1 + \frac{1}{2} E[X] \Rightarrow \frac{1}{2} E[X] = 1 \Rightarrow E[X] = 2. \]

Probability that random pivot elt in the middle half is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).

Expected time recurrence:

\[ T(n) \leq T\left( \frac{3}{4} n \right) + O(n) \]

Masters or just thinking:

\( n + \left( \frac{3}{4} n \right) + \left( \frac{3}{4} n \right)^2 + \cdots = O(n) \)

\[ \Rightarrow T(n) = O(n) \]
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.
Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: \( E[X] = 1 + \frac{1}{2} E[X] \)
Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: \[ E[X] = 1 + \frac{1}{2} E[X] \]
\[ \Rightarrow \frac{1}{2} E[X] = 1 \]
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:** $E[X] = 1 + \frac{1}{2} E[X]$

$\implies \frac{1}{2} E[X] = 1 \implies E[X] = 2.$

\[\Box\]
Lemma: Expected number of coin tosses to get a heads is 2.

Proof: $E[X] = 1 + \frac{1}{2} E[X]$

$\implies \frac{1}{2} E[X] = 1 \implies E[X] = 2.$

Probability that random pivot elt in the middle half is $\geq \frac{1}{2}$. 
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:** $E[X] = 1 + \frac{1}{2} E[X]$

$\implies \frac{1}{2} E[X] = 1 \implies E[X] = 2.$

Probability that random pivot elt in the middle half is $\geq \frac{1}{2}$. 

---

Pick in the middle half subproblem size is $\leq \frac{3}{4} n$.

Expected time recurrence:

$T(n) \leq T\left(\frac{3}{4} n\right) + O(n)$. 

Masters or just thinking: 

$(n + (\frac{3}{4} n + (\frac{3}{4} n)^2 + \cdots) = O(n) \implies T(n) = O(n)$
Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: \( E[X] = 1 + \frac{1}{2} E[X] \)
\[ \implies \frac{1}{2} E[X] = 1 \implies E[X] = 2. \]

Probability that random pivot elt in the middle half is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:**
\[ E[X] = 1 + \frac{1}{2} E[X] \]
\[ \implies \frac{1}{2} E[X] = 1 \implies E[X] = 2. \]

Probability that random pivot elt in the middle half is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).
**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:** \( E[X] = 1 + \frac{1}{2} E[X] \)

\[ \implies \frac{1}{2} E[X] = 1 \implies E[X] = 2. \]

Probability that random pivot elt in the **middle half** is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).

Expected time recurrence:

\[ T(n) \leq \]
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:** \[ E[X] = 1 + \frac{1}{2} E[X] \]
\[ \implies \frac{1}{2} E[X] = 1 \implies E[X] = 2. \]

Probability that random pivot elt in the middle half is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).

Expected time recurrence:
\[ T(n) \leq T\left(\frac{3}{4} n\right) + \]
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:** \( E[X] = 1 + \frac{1}{2} E[X] \)

\[ \implies \frac{1}{2} E[X] = 1 \implies E[X] = 2. \]

Probability that random pivot elt in the *middle half* is \( \geq \frac{1}{2} \).

![Diagram of middle half pick in Quicksort process]

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).

Expected time recurrence:

\[ T(n) \leq T\left(\frac{3}{4} n\right) + O(n). \]
Expected (average) Time?

Lemma: Expected number of coin tosses to get a heads is 2.

Proof: \( E[X] = 1 + \frac{1}{2} E[X] \)

\[ \implies \frac{1}{2} E[X] = 1 \implies E[X] = 2. \]

Probability that random pivot elt in the middle half is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).

Expected time recurrence:

\[ T(n) \leq T\left(\frac{3}{4} n\right) + O(n). \]

Masters or just thinking:
**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:**
\[
E[X] = 1 + \frac{1}{2} E[X]
\]
\[
\implies \frac{1}{2} E[X] = 1 \implies E[X] = 2.
\]

Probability that random pivot elt in the *middle half* is \( \geq \frac{1}{2} \).

Expected time to get a middle element is \( E[X] \times O(n) = O(n) \).

Pick in the middle half subproblem size is \( \leq \frac{3}{4} n \).

Expected time recurrence:
\[
T(n) \leq T\left(\frac{3}{4} n\right) + O(n).
\]

Masters or just thinking: \( (n + (3/4)n + (3/4)^2 n + \cdots = O(n)) \)
Expected (average) Time?

**Lemma:** Expected number of coin tosses to get a heads is 2.

**Proof:** $E[X] = 1 + \frac{1}{2} E[X]$

$\implies \frac{1}{2} E[X] = 1 \implies E[X] = 2.$

Probability that random pivot elt in the middle half is $\geq \frac{1}{2}$.

Expected time to get a middle element is $E[X] \times O(n) = O(n)$. 
Pick in the middle half subproblem size is $\leq \frac{3}{4} n$.

Expected time recurrence:

$$T(n) \leq T\left(\frac{3}{4} n\right) + O(n).$$

Masters or just thinking: $(n + (3/4)n + (3/4)^2 n + \cdots = O(n))$

$\implies T(n) = O(n).$
Extra Example: Deterministic Selection.

Recall Selection of “pivot”:
Extra Example: Deterministic Selection.

Recall Selection of “pivot”:
Choose \texttt{rand. elt} \texttt{b} from \texttt{A}.
Extra Example: Deterministic Selection.

Recall Selection of “pivot”:
Choose \textit{rand.} elt $b$ from $A$. 

Extra Example: Deterministic Selection.

Recall Selection of “pivot”:
Choose \textit{rand.} elt $b$ from $A$.
   Expected to be “in the middle”.
Extra Example: Deterministic Selection.

Recall Selection of “pivot”:
Choose rand. elt $b$ from $A$.
    Expected to be “in the middle”.
Instead: find elt that must be “in the middle.”
SelectPivot

SelectPivot: $A$. 

$S = \text{medians of each group.}$ 

$|S| = n/5$. 

Return median ($S$).

"In Middle" Lemma: $x$ is $\geq$ (also $\leq$) at least $3 = \frac{1}{10} n$ elements.

Proof: $x$ is at least as large as half of $S$. Each distinct elt of $S$ is at least as large as 5 distinct elements of $A$.

Argument picture: $A = (\ldots m_1 \ldots (a, b, \ldots m_i \ldots) \ldots x \ldots) \ldots x \geq m_i \Rightarrow x \geq a, b, m_i$ or $x \geq 3$ elements of 1/2 of $n/5 \times 3 = \frac{3}{10} n$ elements.
SelectPivot

SelectPivot: A.
  Split into groups of size 5.
SelectPivot

SelectPivot: A.
Split into groups of size 5.
$S =$ medians of each group.
SelectPivot

SelectPivot: $A$.
   Split into groups of size 5.
   $S = \text{medians of each group.}$
   $|S|?$. 
SelectPivot

SelectPivot: A.
   Split into groups of size 5.
   $S =$ medians of each group.
   $|S| = \frac{n}{5}$. 

"In Middle" Lemma: $x \geq (\text{also } \leq)$ at least $\frac{3}{10}n$ elements.

Proof: $x$ is at least as large as half of $S$.
Each distinct elt of $S$ is at least as large as 5 distinct elements of $A$.
Argument picture:
$A = (\ldots m_1 \ldots \cdot \cdot \cdot (a, b, \ldots m_i \ldots \cdot \cdot \cdot (\ldots x \ldots) \cdot \cdot \cdot x \geq m_i \Rightarrow x \geq a, b, m_i$ or $x \geq \frac{3}{10}n$ elements.
SelectPivot

SelectPivot: $A$.
- Split into groups of size 5.
- $S =$ medians of each group.
- $|S| \leq \frac{n}{5}$.
- Return \textbf{median}(S).
SelectPivot

SelectPivot: $A$.
  Split into groups of size 5.
  $S = \text{medians of each group}$. 
  $|S|? \ |S| = \frac{n}{5}$.
  Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10} n$ elements.
SelectPivot

SelectPivot: A.
   Split into groups of size 5.
   $S = \text{medians of each group.}$
   $|S|\frac{}{} |S| = \frac{n}{5}.$
   Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
SelectPivot

SelectPivot: $A$.
   Split into groups of size 5.
   $S = \text{medians of each group}.$
   $|S| \approx |S| = \frac{n}{5}$.
   Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
   $x$ is at least as large as half of $S$. 
SelectPivot

SelectPivot: \( A \).
Split into groups of size 5.
\( S = \) medians of each group.
\(|S|?|S| = \frac{n}{5} \).
Return median\( (S) \).

“In Middle” Lemma: \( x \) is \( \geq \) (also \( \leq \)) at least \( \frac{3}{10}n \) elements.

Proof:
\( x \) is at least as large as half of \( S \).
Each distinct elt of \( S \) is at least as large as
SelectPivot

SelectPivot: $A$.
- Split into groups of size 5.
- $S = \text{medians of each group}$.
  - $|S|? |S| = \frac{n}{5}$.
- Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
- $x$ is at least as large as half of $S$.
- Each distinct elt of $S$ is at least as large as
  - 5 distinct elements of $A$. 
SelectPivot

SelectPivot: $A$.
   Split into groups of size 5.
   $S = \text{medians of each group.}$
   $|S| = \frac{n}{5}$.
   Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
   $x$ is at least as large as half of $S$.
   Each distinct elt of $S$ is at least as large as
      5 distinct elements of $A$.
   Argument picture:
SelectPivot

SelectPivot: A.
  Split into groups of size 5.
  $S = \text{medians of each group}$.
  $|S|? \ 
  |S| = \frac{n}{5}$.
  Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10} n$ elements.

Proof:
$x$ is at least as large as half of $S$.
Each distinct elt of $S$ is at least as large as
  5 distinct elements of $A$.
Argument picture:
$A = (\cdots m_1 \cdots) \cdots (a, b, , m_i \cdots) \cdots (\cdots x \cdots) \cdots$
SelectPivot

SelectPivot: $A$.
Split into groups of size 5.
$S = $ medians of each group.
$|S| = \frac{n}{5}$.
Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
$x$ is at least as large as half of $S$.
Each distinct elt of $S$ is at least as large as
5 distinct elements of $A$.

Argument picture:
$A = (\ldots m_1 \ldots) \ldots (a, b, , m_i \ldots) \ldots (\ldots x \ldots) \ldots$
$x \geq m_i \implies x \geq a, b, m_i$
SelectPivot

SelectPivot: \( A \).
- Split into groups of size 5.
- \( S = \) medians of each group.
  \[ |S| = \frac{n}{5}. \]
- Return median\((S)\).

“In Middle” Lemma: \( x \) is \( \geq \) (also \( \leq \)) at least \( \frac{3}{10}n \) elements.

Proof:
- \( x \) is at least as large as half of \( S \).
- Each distinct elt of \( S \) is at least as large as
  - 5 distinct elements of \( A \).

Argument picture:
\[ A = (\cdots m_1 \cdots) \cdots (a, b, m_1 \cdots) \cdots (\cdots x \cdots) \cdots \]
\[ x \geq m_i \implies x \geq a, b, m_i \text{ or } x \geq 3 \text{ elements of } \frac{1}{2} \text{ of } \frac{n}{5} \text{ groups} \]
SelectPivot

SelectPivot: $A$.
- Split into groups of size 5.
- $S =$ medians of each group.
  - $|S| \leq \frac{n}{5}$.
- Return $\text{median}(S)$.

"In Middle" Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
- $x$ is at least as large as half of $S$.
- Each distinct elt of $S$ is at least as large as
  - 5 distinct elements of $A$.

Argument picture:
- $A = (\cdots m_1 \cdots) \cdots (a, b, , m_i \cdots) \cdots (\cdots x \cdots) \cdots$
- $x \geq m_i \iff x \geq a, b, m_i$ or $x \geq 3$ elements of $\frac{1}{2}$ of $\frac{n}{5}$ groups
SelectPivot

SelectPivot: \( A \).
  Split into groups of size 5.
  \( S = \) medians of each group.
  \(|S| \geq \frac{n}{5} \).
  Return \text{median}(S).

“In Middle” Lemma: \( x \) is \( \geq \) (also \( \leq \)) at least \( \frac{3}{10} n \) elements.

Proof:
  \( x \) is at least as large as half of \( S \).
  Each distinct elt of \( S \) is at least as large as
    5 distinct elements of \( A \).

Argument picture:
\[
A = (\cdots m_1 \cdots) \cdots (a, b, \cdots) \cdots (\cdots x \cdots) \cdots
\]
\( x \geq m_i \implies x \geq a, b, m_i \) or \( x \geq 3 \) elements of \( \frac{1}{2} \) of \( \frac{n}{5} \) groups

\implies x \) is at least as large as
SelectPivot

SelectPivot: \( A \).
Split into groups of size 5.
\( S = \text{medians of each group.} \)
\(|S|? |S| = \frac{n}{5}. \)
Return \textbf{median}(S).

“In Middle” Lemma: \( x \) is \( \geq \) (also \( \leq \)) at least \( \frac{3}{10} n \) elements.

Proof:
\( x \) is at least as large as half of \( S \).
Each distinct elt of \( S \) is at least as large as
5 distinct elements of \( A \).
Argument picture:
\( A = (\cdots m_1 \cdots) \cdots (a, b, , m_i \cdots) \cdots (\cdots x \cdots) \cdots \)
\( x \geq m_i \implies x \geq a, b, m_i \) or \( x \geq 3 \) elements of \( \frac{1}{2} \) of \( \frac{n}{5} \) groups
\( \implies \) \( x \) is at least as large as
\[ \frac{1}{2} \]
SelectPivot

SelectPivot: $A$.
Split into groups of size 5.
$S = \text{medians of each group.}$
$|S|?\ |S| = \frac{n}{5}$.
Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10} n$ elements.

Proof:
$x$ is at least as large as half of $S$.
Each distinct elt of $S$ is at least as large as
5 distinct elements of $A$.

Argument picture:
$A = (\cdots m_1 \cdots) \cdots (a, b, , m_i \cdots) \cdots (\cdots x \cdots) \cdots$
$x \geq m_i \implies x \geq a, b, m_i$ or $x \geq 3$ elements of $\frac{1}{2}$ of $\frac{n}{5}$ groups

$\implies x$ is at least as large as
$\frac{1}{2} \times \frac{n}{5} \times$
SelectPivot

SelectPivot: $A$.
Split into groups of size 5.
$S = \text{medians of each group.}$
$|S|?|S| = \frac{n}{5}$.
Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10} n$ elements.

Proof:
$x$ is at least as large as half of $S$.
Each distinct elt of $S$ is at least as large as
5 distinct elements of $A$.
Argument picture:
$A = (\ldots m_1 \ldots) \ldots (a, b, , m_i \ldots) \ldots (\ldots x \ldots) \ldots$
$x \geq m_i \implies x \geq a, b, m_i$ or $x \geq 3$ elements of $\frac{1}{2}$ of $\frac{n}{5}$ groups
$
\implies x$ is at least as large as
$\frac{1}{2} \times \frac{n}{5} \times 3$
SelectPivot

SelectPivot: $A$.
   Split into groups of size 5.
   $S =$ medians of each group.
   $|S| \leq \frac{n}{5}$.
   Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
   $x$ is at least as large as half of $S$.
   Each distinct elt of $S$ is at least as large as
      5 distinct elements of $A$.
   Argument picture:
   $A = (\cdots m_1 \cdots) \cdots (a, b, m_i \cdots) \cdots (\cdots x \cdots) \cdots$
   $x \geq m_i \implies x \geq a, b, m_i$ or $x \geq 3$ elements of $\frac{1}{2}$ of $\frac{n}{5}$ groups
   $\implies x$ is at least as large as
   \[
   \frac{1}{2} \times \frac{n}{5} \times 3 = \frac{3}{10}n \text{ elements.}
   \]
SelectPivot

SelectPivot: $A$.
Split into groups of size 5.
$S = \text{medians of each group}.$
$|S| \leq |S| = \frac{n}{5}$.
Return $\text{median}(S)$.

“In Middle” Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

Proof:
$x$ is at least as large as half of $S$.
Each distinct elt of $S$ is at least as large as
5 distinct elements of $A$.
Argument picture:
$A = (\cdots m_1 \cdots) \cdots (a, b, m_i \cdots) \cdots (\cdots x \cdots) \cdots$
$x \geq m_i \implies x \geq a, b, m_i \text{ or } x \geq 3 \text{ elements of } 12 \text{ of } \frac{n}{5} \text{ groups}$
$\implies x$ is at least as large as
$\frac{1}{2} \times \frac{n}{5} \times 3 = \frac{3}{10}n$ elements.
SelectPivot: runtime recurrence.

SelectPivot: A.
SelectPivot: runtime recurrence.

SelectPivot: A.
Split into groups of 5.
SelectPivot: runtime recurrence.

SelectPivot: $A$.
  Split into groups of 5.
  $S =$ medians of each group.
SelectPivot: runtime recurrence.

SelectPivot: $A$.
   Split into groups of 5.
   $S = \text{medians of each group.}$
      $|S|?
SelectPivot: runtime recurrence.

SelectPivot: A.
Split into groups of 5.
$S = \text{medians of each group.}$
$|S| \leq \frac{n}{5}.$
SelectPivot: runtime recurrence.

SelectPivot: A.
Split into groups of 5.
$S = \text{medians of each group.}$
$|S| \leq \frac{n}{5}$.
Return $\text{median}(S)$. 
SelectPivot: runtime recurrence.

SelectPivot: A.
  Split into groups of 5.
  \( S = \text{medians of each group.} \)
  \(|S| = \frac{n}{5} \).
  Return \texttt{median}(S).

Calls median! Runtime \( P(n) \)?
SelectPivot: runtime recurrence.

SelectPivot: A.
- Split into groups of 5.
- $S = \text{medians of each group.}$
  - $|S| = \frac{n}{5}$.
- Return median$(S)$.

Calls median! Runtime $P(n)$?
- $P(n) \leq T\left(\frac{n}{5}\right) + O(n)$
SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S = \text{medians of each group.}$

$|S| = \frac{n}{5}.$

Return \text{median}(S).

Calls median! Runtime $P(n)$?

$P(n) \leq T\left(\frac{n}{5}\right) + O(n)$

where $T(\cdot)$ is runtime for median.
SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S = \text{medians of each group.}$

$|S| \sim |S| = \frac{n}{5}.$

Return $\text{median}(S).$

Calls median! Runtime $P(n)\,$?

\[ P(n) \leq T\left(\frac{n}{5}\right) + O(n) \]

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

Middle Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.
SelectPivot: runtime recurrence.

SelectPivot: A.
Split into groups of 5.
$S = \text{medians of each group.}$
$|S| = \frac{n}{5}.$
Return \textbf{median}(S).

Calls median! Runtime $P(n)$?

$$P(n) \leq T\left(\frac{n}{5}\right) + O(n)$$

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

Middle Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10} n$ elements.

$$T(n) \leq P(n) + T(\frac{7}{10} n) + O(n).$$
SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S = \text{medians of each group.}$

$|S|? \quad |S| = \frac{n}{5}.$

Return $\text{median}(S).$

Calls median! Runtime $P(n)?$ 

$P(n) \leq T\left(\frac{n}{5}\right) + O(n)$

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

Middle Lemma: $x$ is $\geq$ (also $\leq$) at least $\frac{3}{10} n$ elements.

$T(n) \leq P(n) + T\left(\frac{7}{10} n\right) + O(n).$

Or,

$T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10} n\right) + O(n).$
SelectPivot: runtime recurrence.

SelectPivot: A.
  Split into groups of 5.
  \( S = \text{medians of each group.} \)
  \(|S| = \frac{n}{5} \).
  Return \texttt{median}(S).

Calls median! Runtime \( P(n) \)?

\[
P(n) \leq T\left(\frac{n}{5}\right) + O(n)
\]

where \( T(\cdot) \) is runtime for median.

\( T(n) \) recurrence?
  Middle Lemma: \( x \) is \( \geq \) (also \( \leq \)) at least \( \frac{3}{10} n \) elements.

\[
T(n) \leq P(n) + T\left(\frac{7}{10}n\right) + O(n).
\]

Or,
\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + O(n).
\]

\( O(n) \)- compute medians \( S \) and for partitioning.
SelectPivot: runtime recurrence.

SelectPivot: A.
Split into groups of 5.
$S = \text{medians of each group.}$

$|S| = \frac{n}{5}.$
Return $\text{median}(S)$.

Calls median! Runtime $P(n)$?

$P(n) \leq T\left(\frac{n}{5}\right) + O(n)$

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

Middle Lemma: $x$ is $\geq (\text{also } \leq)$ at least $\frac{3}{10}n$ elements.

$T(n) \leq P(n) + T\left(\frac{7}{10}n\right) + O(n)$.

Or,

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

$O(n)$- compute medians $S$ and for partitioning.

$T\left(\frac{n}{5}\right)$ for computing median of $S$. 
SelectPivot: runtime recurrence.

SelectPivot: A.
- Split into groups of 5.
- $S = \text{medians of each group}.$
  - $|S| \geq \frac{n}{5}.$
- Return $\text{median}(S).$

Calls median! Runtime $P(n)$?

\[
P(n) \leq T\left(\frac{n}{5}\right) + O(n)
\]

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?
- Middle Lemma: $x$ is \( \geq \) (also \( \leq \)) at least \( \frac{3}{10} n \) elements.

\[
T(n) \leq P(n) + T\left(\frac{7}{10}n\right) + O(n).
\]

Or,

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

$O(n)$- compute medians $S$ and for partitioning.

$T\left(\frac{n}{5}\right)$ for computing median of $S$.

$T\left(\frac{7}{10}n\right)$ for the recursive call in Select.
Bound Recurrence.

\[ T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10} n) + cn. \quad T(1) = c. \]
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10} n\right) + cn. \quad T(1) = c. \]
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10} n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1. \)
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \(\frac{1}{5} + \frac{7}{10} < 1\). Problem sizes decrease geometrically.
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).
Bound Recurrence.

\[ T(n) \leq T\left( \frac{n}{5} \right) + T\left( \frac{7}{10} n \right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10} n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c' n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c' n' \) for \( n' < n \).
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c' n' \) for \( n' < n \).

\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn \\
\leq c' \cdot \frac{n}{5} + c' \cdot \frac{7}{10}n + cn \\
\leq c' \cdot \frac{9}{10}n + cn \\
\leq c' n + (c - c' \cdot \frac{1}{10})n
\]
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c'n' \) for \( n' < n \).

\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn \\
\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn \\
\leq c'\frac{9}{10}n + cn \\
\leq c'n + (c - c'\frac{1}{10})n
\]

Choose \( c' \geq 10c \)
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c'n' \) for \( n' < n \).

\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn
\]
\[
\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn
\]
\[
\leq c'\frac{9}{10}n + cn
\]
\[
\leq c'n + (c - c'\frac{1}{10})n
\]

Choose \( c' \geq 10c \implies c - c'\frac{1}{10} < 0 \)
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c'n' \) for \( n' < n \).

\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn \\
\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn \\
\leq c'\frac{9}{10}n + cn \\
\leq c'n + (c - c'\frac{1}{10})n
\]

Choose \( c' \geq 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \leq c'n. \]
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c'n' \) for \( n' < n \).

\[
\begin{align*}
T(n) &\leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn \\
&\leq c'n + c'\frac{7}{10}n + cn \\
&\leq c'n + c'\frac{9}{10}n + cn \\
&\leq c'n + (c - c'\frac{1}{10})n
\end{align*}
\]

Choose \( c' \geq 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \leq c'n. \)

Base Case: \( c' \geq c. \)
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \( \frac{1}{5} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.

Prove \( T(n) \leq c'n \) for some \( c' \).

Induction Hypothesis: \( T(n') \leq c'n' \) for \( n' < n \).

\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn \\
\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn \\
\leq c'\frac{9}{10}n + cn \\
\leq c'n + (c - c'\frac{1}{10})n
\]

Choose \( c' \geq 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \leq c'n. \)

Base Case: \( c' \geq c. \)
Bound Recurrence.

\[ T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn. \quad T(1) = c. \]

Idea: \(\frac{1}{5} + \frac{7}{10} < 1\). Problem sizes decrease geometrically.

Prove \(T(n) \leq c'n\) for some \(c'\).

Induction Hypothesis: \(T(n') \leq c'n'\) for \(n' < n\).

\[
T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + cn
\]
\[
\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn
\]
\[
\leq c'\frac{9}{10}n + cn
\]
\[
\leq c'n + (c - c'\frac{1}{10})n
\]

Choose \(c' \geq 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \leq c'n.\)

Base Case: \(c' \geq c.\)

Selection is \(O(n)\) deterministic time!
Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) \]

Also: iterative view.

Sorting Lower Bound.

\( n! \) possible output orderings.

Comparison splits outputs into 2.

\( \Omega(\log n!) = \Omega(n \log n) \) time.

Median finding.

Selection: more general, "strengthen induction."

Random pivot element to split elements.

Recurse on one subset.

Expected Time Analysis:

\( O(n) \) time to decrease size by 3/4.

Extra: Deterministic Pivot Selection.
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
    Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n\log n). \]
Also: iterative view.
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
  Merge: two pointer scan.
  $T(n) = 2T(n/2) + O(n)$.
  $T(n) = O(n \log n)$.
Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.
Comparison splits outputs into 2.
$\Omega(n!) = \Omega(n \log n)$ time.
Median finding.
Selection: more general, "strengthen induction."
Random pivot element to split elements.
Recurs on one subset.
Expected Time Analysis: $O(n)$ time to decrease size by $3/4$.
Extra: Deterministic Pivot Selection.
Lecture in a minute.

MergeSort.
- Sort two halves, put together.
  - Merge: two pointer scan.
  
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n\log n). \]

Also: iterative view.

Sorting Lower Bound.
- \( n! \) possible output orderings.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
   Merge: two pointer scan.
   \( T(n) = 2T\left(\frac{n}{2}\right) + O(n) \).
   \( T(n) = O(n \log n) \).
Also: iterative view.

Sorting Lower Bound.
   \( n! \) possible output orderings.
   Comparison splits outputs into 2.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n\log n). \]
Also: iterative view.

Sorting Lower Bound.
\( n! \) possible output orderings.
Comparison splits outputs into 2.
\( \Omega(\log n!) = \Omega(n\log n) \) time.
Lecture in a minute.

MergeSort.
  Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) \].
  \[ T(n) = O(n \log n) \].
  Also: iterative view.

Sorting Lower Bound.
  \( n! \) possible output orderings.
  Comparison splits outputs into 2.
  \[ \Omega(\log n!) = \Omega(n \log n) \] time.

Median finding.
  Selection: more general, “strengthen induction.”
  Random pivot element to split elements.
  Recurse on one subset.
Lecture in a minute.

MergeSort.
Sort two halves, put together.
  Merge: two pointer scan.
  \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
  \[ T(n) = O(n \log n). \]
Also: iterative view.

Sorting Lower Bound.
  \( n! \) possible output orderings.
  Comparison splits outputs into 2.
  \( \Omega(\log n!) = \Omega(n \log n) \) time.

Median finding.
  Selection: more general, “strengthen induction.”
  Random pivot element to split elements.
  Recurse on one subset.
Expected Time Analysis:
  \( O(n) \) time to decrease size by \( 3/4 \).
Lecture in a minute.

MergeSort.
Sort two halves, put together.
  Merge: two pointer scan.
  \( T(n) = 2T\left(\frac{n}{2}\right) + O(n) \).
  \( T(n) = O(n\log n) \).
Also: iterative view.

Sorting Lower Bound.
\( n! \) possible output orderings.
Comparison splits outputs into 2.
\( \Omega(\log n!) = \Omega(n\log n) \) time.

Median finding.
  Selection: more general, “strengthen induction.”
  Random pivot element to split elements.
  Recurse on one subset.
Expected Time Analysis:
  \( O(n) \) time to decrease size by 3/4.

Extra: Deterministic Pivot Selection.
Lecture in a minute.

MergeSort.
    Sort two halves, put together.
    Merge: two pointer scan.
    \[ T(n) = 2T\left(\frac{n}{2}\right) + O(n). \]
    \[ T(n) = O(n\log n). \]
Also: iterative view.

Sorting Lower Bound.
    \( n! \) possible output orderings.
    Comparison splits outputs into 2.
    \( \Omega(\log n!) = \Omega(n\log n) \) time.

Median finding.
    Selection: more general, “strengthen induction.”
    Random pivot element to split elements.
    Recurse on one subset.
Expected Time Analysis:
    \( O(n) \) time to decrease size by 3/4.

Extra: Deterministic Pivot Selection.