CS 170: Algorithms

CS 170: Algorithms

Hello and ...

H . . . H . S . H H

CS 170: Algorithms

Hello and ...

H . . . H .   . H H
S

CS 170: Algorithms

Hello and ...

H . . . H .   . H

S H

CS 170: Algorithms

Hello and ...

H . . . . . H

S H H

Hello and ...

H . . . . .

S H H H

CS 170: Algorithms

Hello and ...

S H H H H

CS 170: Algorithms

Hello and ...

S H H H H .

Hello and ...

S H H H H . .

CS 170: Algorithms

Hello and ...

S H H H H . . .

CS 170: Algorithms

Hello and ...

S H H H H . . . .

CS 170: Algorithms

Hello and ...

S H H H H . . . . . .

CS 170: Algorithms

Hello and ...

S H H H H . . . . . .

Please,

CS 170: Algorithms

Hello and ...

S H H H H . . . . . .

Please, limit laptops (unless lecture draft slides),

CS 170: Algorithms

Hello and ...

S H H H H . . . . . .

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

CS 170: Algorithms

Hello and ...

S H H H H . . . . . .

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

Bad for your learning. Worse for your neighbors learning.

CS 170: Algorithms

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.

CS 170: Algorithms

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.

Lecture in a minute.

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

Lecture in a minute.

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

Sorting Lower Bound.

Lecture in a minute.

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

Also: iterative view.

Sorting Lower Bound.
- $n!$ possible output orderings.

Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Comparison splits outputs into 2.

Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Comparison splits outputs into 2.

$\Omega(\log n!) = \Omega(n \log n)$ time.

Lecture in a minute.

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

Sorting Lower Bound.
- $n!$ possible output orderings.
- Comparison splits outputs into 2.
- $\Omega(\log n!) = \Omega(n\log n)$ 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.
$T(n) = 2T(\frac{n}{2}) + O(n)$.
$T(n) = O(n\log n)$.
Also: iterative view.

Sorting Lower Bound.
$n!$ possible output orderings.
Comparison splits outputs into 2.
$\Omega(\log n!) = \Omega(n\log n)$ 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.
$T(n) = 2T(\frac{n}{2}) + O(n)$.
$T(n) = O(n \log n)$.
Also: iterative view.

Sorting Lower Bound.
$n!$ possible output orderings.
Comparison splits outputs into 2.
$\Omega(\log n!) = \Omega(n \log n)$ 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.
$T(n) = 2T(\frac{n}{2}) + O(n)$.
$T(n) = O(n\log n)$.
Also: iterative view.

Sorting Lower Bound.
$n!$ possible output orderings.
Comparison splits outputs into 2.
$\Omega(\log n!) = \Omega(n\log n)$ 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., $A = [5, 6, 7, 9, 10, 2, 3...]$.

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)

```
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., $A = [5, 6, 7, 9, 10, 2, 3...]$.

Mergesort(A)

```
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?

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)

```
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,

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)

```
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,

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)

```
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.

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)

```
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.

Sorted SubArray 1: $3, 7, 8, 10, 11, \ldots$
Sorted Subarray 2: $4, 5, 9, 19, 20, \ldots$

, , , ,

More divide and conquer: mergesort.

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

Mergesort(A)

```
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.

Sorted SubArray 1: $3, 7, 8, 10, 11, \ldots$
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., $A = [5,6,7,9,10,2,3...]$.

Mergesort(A)

```
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.

Sorted SubArray 1: ~~3~~, 7, 8, 10, 11, ...

Sorted Subarray 2: ~~4~~, 5, 9, 19, 20, ...

3, 4, , ,

More divide and conquer: mergesort.

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

Mergesort(A)

```
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.

Sorted SubArray 1: ~~3~~, 7, 8, 10, 11, ...
Sorted Subarray 2: ~~4~~, ~~5~~, 9, 19, 20, ...

3, 4, 5, ,

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)

```
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.

Sorted SubArray 1: $\cancel{3}, \cancel{7}, 8, 10, 11, \ldots$

Sorted Subarray 2: $\cancel{4}, \cancel{5}, 9, 19, 20, \ldots$

$3, 4, 5, 7,$

More divide and conquer: mergesort.

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

Mergesort(A)

```
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.

Sorted SubArray 1: ~~3~~, ~~7~~, ~~8~~, 10, 11, ...

Sorted Subarray 2: ~~4~~, ~~5~~, 9, 19, 20, ...

3, 4, 5, 7, 8

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)

```
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.

Sorted	SubArray 1:	$\cancel{3}, \cancel{7}, \cancel{8}, 10, 11, \ldots$
Sorted	Subarray 2:	$\cancel{4}, \cancel{5}, 9, 19, 20, \ldots$
$3, 4, 5, 7, 8$	$\ldots$	

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
```

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

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)

```
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)

```
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

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 : $O(n)$.

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 $n/2$.

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 $n/2$.

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

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 $n/2$.

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

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
        (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 $n/2$.

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

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

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 $n/2$.

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

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

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 $n/2$.

$$T(n) = 2T(\tfrac{n}{2}) + O(n).$$

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)

```
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 $n/2$.

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

Masters: $T(n) = aT(n/b) + O(n^d)$

with $\log_b a = d \implies O(n^d \log_b n)$

Apply Masters:

$a = 2$, $b = 2$, $d = 1 \implies \log_2 2 = 1$

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 comparision : $O(n)$.
Recursive: 2 subproblems of size $n/2$.

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

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

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

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Check it out...

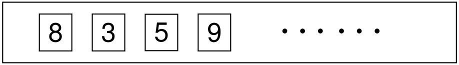
Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

Check it out...

Iterative Mergesort: Bottom up, use of queues.

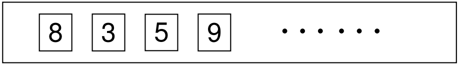
Make each element into list and put lists in queue.



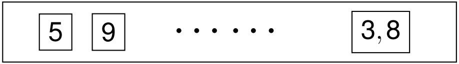
Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.



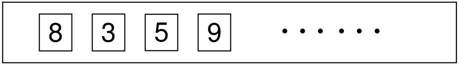
Merge first two lists, put in queue (at end).



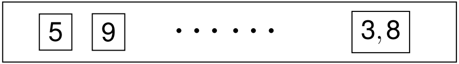
Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.



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

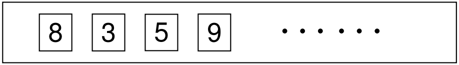


Rinse.

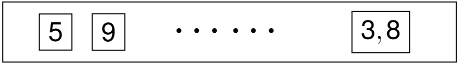
Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.



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

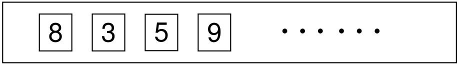


Rinse. Repeat.

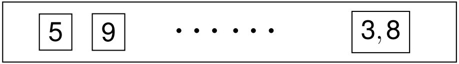
Check it out...

Iterative Mergesort: Bottom up, use of queues.

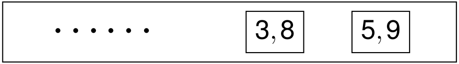
Make each element into list and put lists in queue.



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



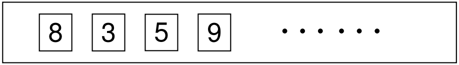
Rinse. Repeat.



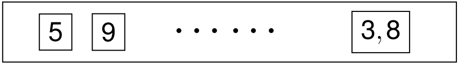
Check it out...

Iterative Mergesort: Bottom up, use of queues.

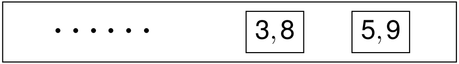
Make each element into list and put lists in queue.



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



Rinse. Repeat.

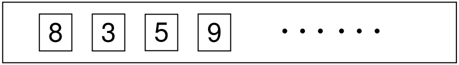


And next pass through queue...

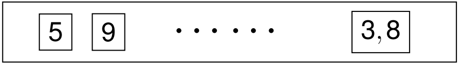
Check it out...

Iterative Mergesort: Bottom up, use of queues.

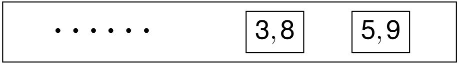
Make each element into list and put lists in queue.



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



Rinse. Repeat.

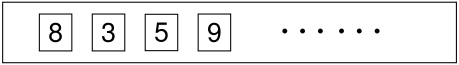


And next pass through queue...

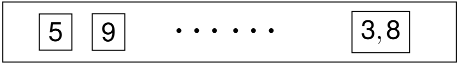
Check it out...

Iterative Mergesort: Bottom up, use of queues.

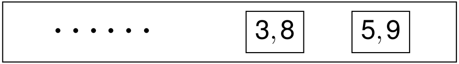
Make each element into list and put lists in queue.



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



Rinse. Repeat.



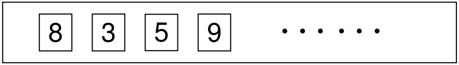
And next pass through queue...

Each pass through queue:

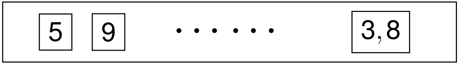
Check it out...

Iterative Mergesort: Bottom up, use of queues.

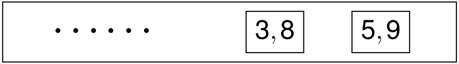
Make each element into list and put lists in queue.



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



Rinse. Repeat.



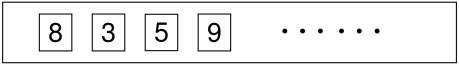
And next pass through queue...

Each pass through queue: each element touched once.

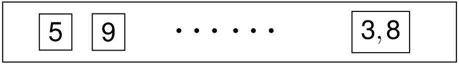
Check it out...

Iterative Mergesort: Bottom up, use of queues.

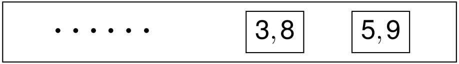
Make each element into list and put lists in queue.



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



Rinse. Repeat.



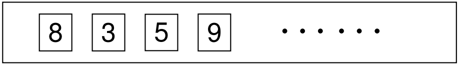
And next pass through queue...

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

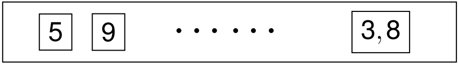
Check it out...

Iterative Mergesort: Bottom up, use of queues.

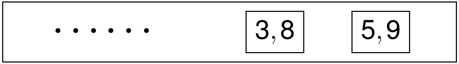
Make each element into list and put lists in queue.



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



Rinse. Repeat.



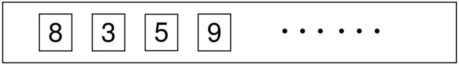
And next pass through queue...

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

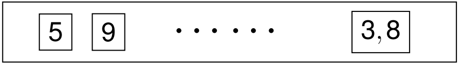
Check it out...

Iterative Mergesort: Bottom up, use of queues.

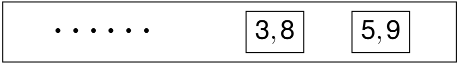
Make each element into list and put lists in queue.



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



Rinse. Repeat.



And next pass through queue...

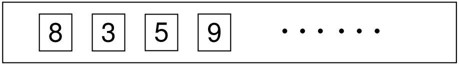
Each pass through queue: each element touched once. $O(n)$ time.
Each pass halves number of lists.

$\implies O(\log n)$ passes

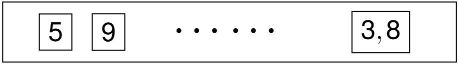
Check it out...

Iterative Mergesort: Bottom up, use of queues.

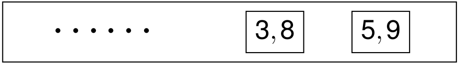
Make each element into list and put lists in queue.



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



Rinse. Repeat.



And next pass through queue...

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?

Sorting lower bound.

Can we do better?

Comparison sorting algorithm only compares numbers.

Sorting lower bound.

Can we do better?

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

Sorting lower bound.

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.

Sorting lower bound.

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?

Sorting lower bound.

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.

Sorting lower bound.

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.

Sorting lower bound.

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"....

Sorting lower bound.

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.

Sorting lower bound.

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...

Sorting lower bound.

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”.

Sorting lower bound.

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.!

Sorting lower bound.

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

Sorting lower bound.

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: $a_8, a_{n-8}, \ldots, a_{15}$

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 $[1, \ldots, n]$.

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: $8, n-8, \ldots, 15$.

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?

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? $n!$

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.

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.

Algorithm must output just 1 permutation at termination.

Sorting lower bound: ...proof

Algorithm must be able to output any of n! permutations.

Sorting lower bound: ...proof

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

Sorting lower bound: ...proof

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

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.

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.

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

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:

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.

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: $a_i > a_j$?

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: $a_i > a_j$?

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.

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.

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.

$S_1 \cup S_2 = S$

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.

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

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.

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

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

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

$n! \geq (\frac{n}{e})^n$

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

$n! \geq (\frac{n}{e})^n \implies \log n!$

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 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.

$n! \geq (\frac{n}{e})^n \implies \log n! = \Omega(\log(n^n))$

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

$n! \geq (\frac{n}{e})^n \implies \log n! = \Omega(\log(n^n)) = \Omega(n \log n)$.

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 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.

$n! \geq (\frac{n}{e})^n \implies \log n! = \Omega(\log(n^n)) = \Omega(n \log n)$. □

Figure for proof.

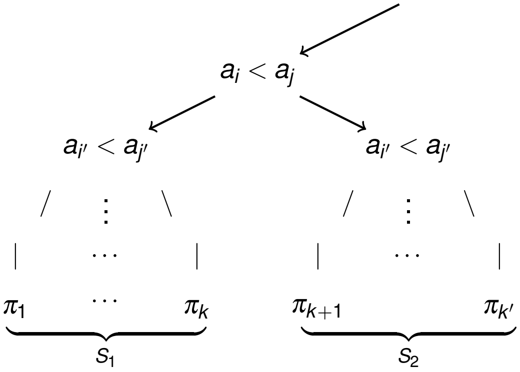
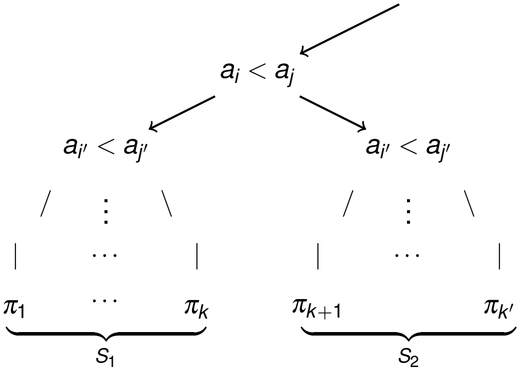
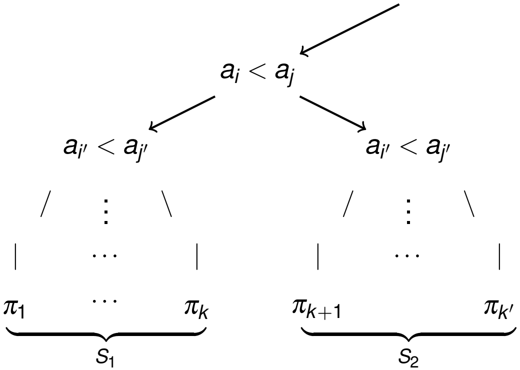


Figure for proof.



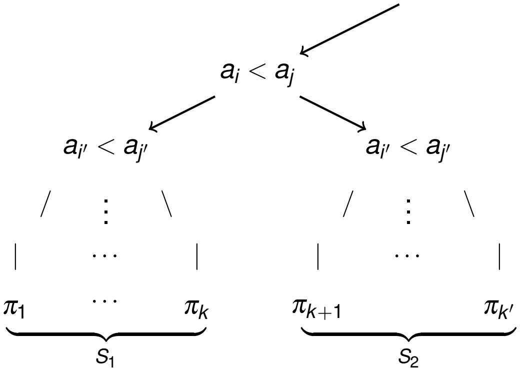
Either the set of permutations S_1 or S_2 is larger.

Figure for proof.



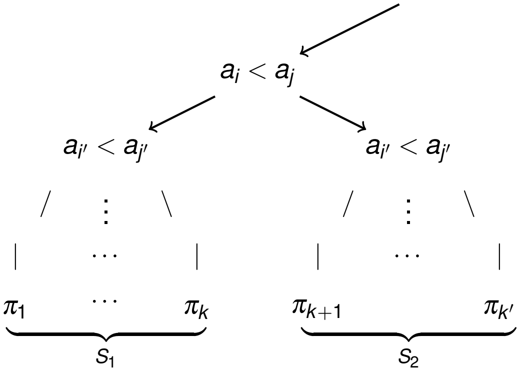
Either the set of permutations S_1 or S_2 is larger.
One must be at least half.

Figure for proof.



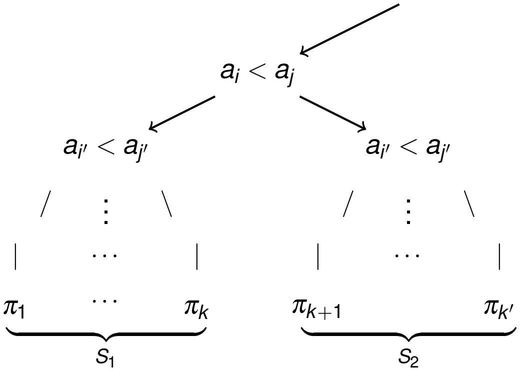
Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#permutations))$

Figure for proof.



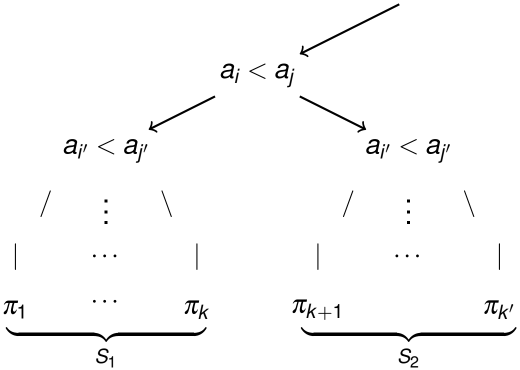
Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!)$

Figure for proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

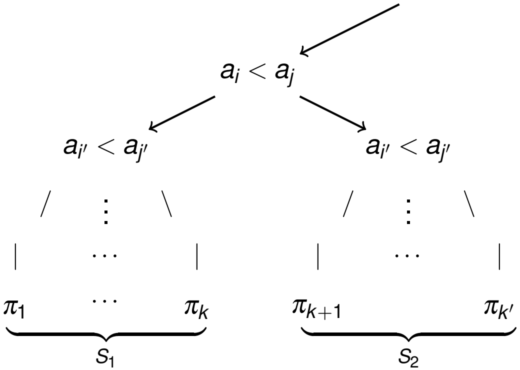
Figure for proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

Can we do better than mergesort?

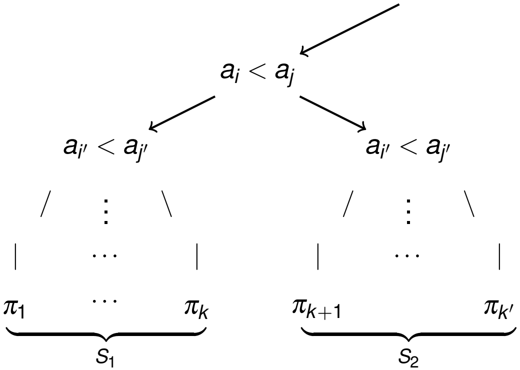
Figure for proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

Can we do better than mergesort? Yes?

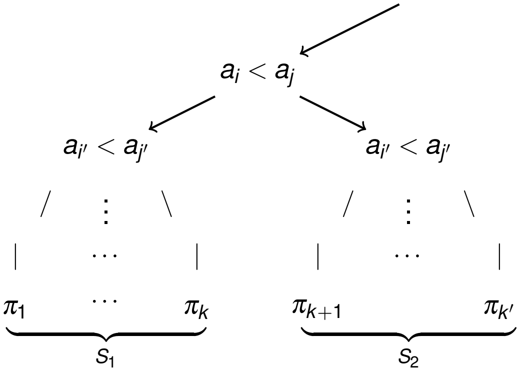
Figure for proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

Can we do better than mergesort? Yes? No?

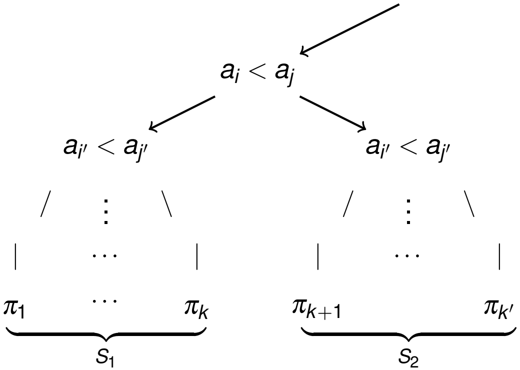
Figure for proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

Can we do better than mergesort? Yes? No?

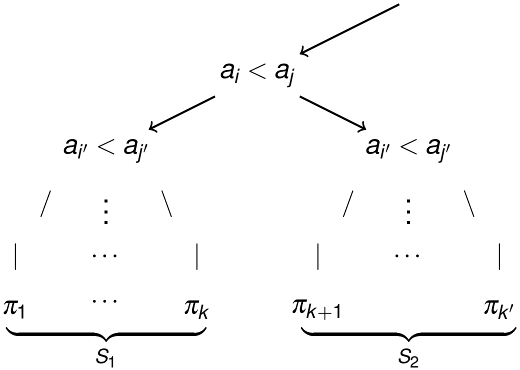
Figure for proof.



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.

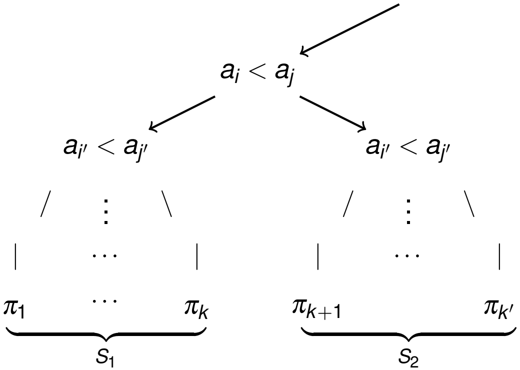
Figure for proof.



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 comparision sort.

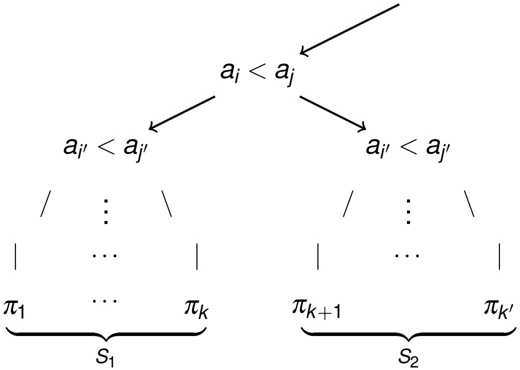
Figure for proof.



Either the set of permutations S_1 or S_2 is larger.
One must be at least half.
Depth must be $\Omega(\log(\#\textit{permutations})) = \Omega(\log n!) = \Omega(n \log n)$.

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

Figure for proof.



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 comparision 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, v, where $\frac{n}{2}$ elts are less than v (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?

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 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

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

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?

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, 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.

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?

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?

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 $\frac{\sum_i a_i}{n}$.

$O(n)$ time.

Median finding.

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

Median is value, 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?

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 $s_1, \ldots s_n$.

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 $s_{n/2+1}$.

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.

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.

Better algorithm?

Solve a harder Problem: Selection.

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 kth smallest element.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

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

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 kth 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 kth 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?

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 kth 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.

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 kth smallest element.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

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

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

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

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

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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. $v = 15$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $< v$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $< v$ $S_L : 11, 5, 2$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $= v$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $= v$ $S_v : 15, 15$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $> v$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $> v$ $S_R : 48, 21, 17, 19$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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). $7 \leq 3$?

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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. $7 \leq 5$?

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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$(k - |S_L| - |S_v|, S_R)$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $\vert S \vert = 1$, 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 $k \leq \vert S_L \vert$, Select(k, S_L).	$7 \leq 3$?
elseif $k \leq \vert S_L \vert + \vert S_v \vert$, return v.	$7 \leq 5$?
else Select$(k - \vert S_L \vert - \vert S_v \vert, S_R)$	Select$(2, [48, 21, 17, 19])$

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

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

Select(k, S)**:** $\quad k = 7$	$S: 11, 48, 5, 21, 2, 15, 17, 19, 15$
Base Case: $k = 1$ and $\lvert S \rvert = 1$, 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 $k \leq \lvert S_L \rvert$, Select(k, S_L).	$7 \leq 3$?
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.	$7 \leq 5$?
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$	Select$(2, [48, 21, 17, 19])$

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

Correctness:

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 kth 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, and

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $\lvert S \rvert = 1$, 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 $k \leq \lvert S_L \rvert$, Select(k, S_L).		$7 \leq 3$?
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.		$7 \leq 5$?
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$		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

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $\lvert S \rvert = 1$, 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 $k \leq \lvert S_L \rvert$, Select(k, S_L).	$7 \leq 3$?
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.	$7 \leq 5$?
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$	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

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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, and so will I

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

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

Select(k, S): $\quad k = 7$	$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$
Base Case: $k = 1$ and $\lvert S \rvert = 1$, 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 $k \leq \lvert S_L \rvert$, Select(k, S_L).	$7 \leq 3$?
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.	$7 \leq 5$?
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$	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 !

Solve a harder Problem: Selection.

For a set of n items S.
Select kth smallest element.

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

Select(k, S): $\quad k = 7$	$S : 11, 48, 5, 21, 2, 15, 17, 19, 15$
Base Case: $k = 1$ and $\lvert S \rvert = 1$, 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 $k \leq \lvert S_L \rvert$, Select(k, S_L).	$7 \leq 3$?
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.	$7 \leq 5$?
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$	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.

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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, and so will I !
Base case is good. Subroutine calls

Solve a harder Problem: Selection.

For a set of n items S.
Select kth 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 $\lvert S \rvert = 1$, 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 $k \leq \lvert S_L \rvert$, Select(k, S_L).		$7 \leq 3$?
elseif $k \leq \lvert S_L \rvert + \lvert S_v \rvert$, return v.		$7 \leq 5$?
else Select$(k - \lvert S_L \rvert - \lvert S_v \rvert, S_R)$		Select$(2, [48, 21, 17, 19])$

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

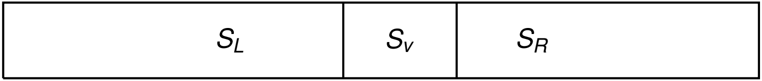
Correctness: Induction.
Idea: Subroutine returns correct answer, and so will I !
Base case is good. Subroutine calls ..by design.

The Induction.

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

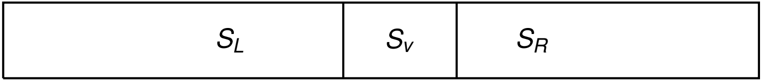
The Induction.

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



The Induction.

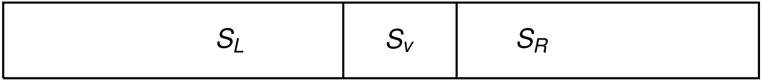
Base Case: $k = 1$, $|S| = 1$. Trivial.



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

The Induction.

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

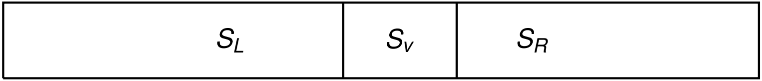


If $k \leq |S_L|$, Select(k, S_L)

kth element in first $|S_L|$ elts.

The Induction.

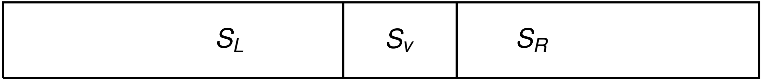
Base Case: $k = 1$, $|S| = 1$. Trivial.



If $k \leq |S_L|$, Select(k, S_L)
kth element in first $|S_L|$ elts.

The Induction.

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



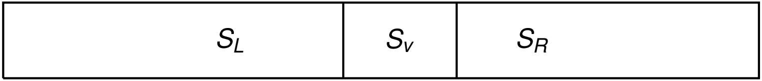
If $k \leq |S_L|$, Select(k, S_L)

kth element in first $|S_L|$ elts.

kth elt of S is kth elt of S_L

The Induction.

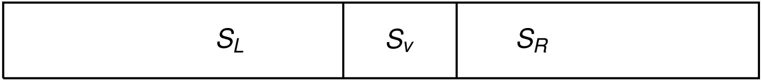
Base Case: $k = 1$, $|S| = 1$. Trivial.



If $k \leq |S_L|$, Select(k, S_L)
kth element in first $|S_L|$ elts.
kth elt of S is kth elt of S_L
elseif $k \leq |S_L| + |S_v|$, return v,

The Induction.

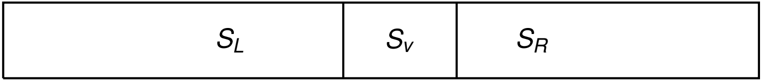
Base Case: $k = 1$, $|S| = 1$. Trivial.



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



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

kth element in first $|S_L|$ elts.

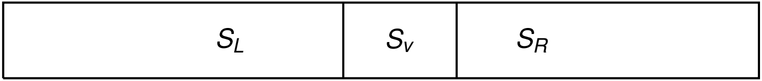
kth elt of S is kth elt of S_L

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.



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

kth element in first $|S_L|$ elts.

kth elt of S is kth elt of S_L

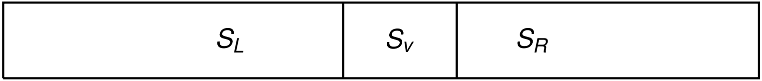
elseif $k \leq |S_L| + |S_v|$, return v,

$k \in [|S_L|, \ldots, |S_L| + |S_v|]$.

kth elt of S is in S_v, all have value v

The Induction.

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



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

kth element in first $|S_L|$ elts.

kth elt of S is kth elt of S_L

elseif $k \le |S_L| + |S_v|$, return v,

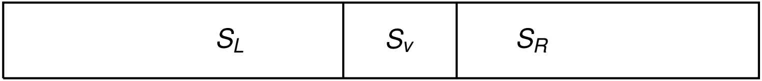
$k \in [|S_L|, \ldots, |S_L| + |S_v|]$.

kth 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.



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

kth element in first $|S_L|$ elts.

kth elt of S is kth elt of S_L

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

$k \in [|S_L|, \ldots, |S_L| + |S_v|]$.

kth elt of S is in S_v, all have value v

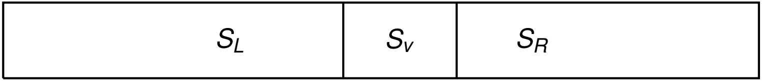
else Select$(k - |S_L| - |S_v|, S_R)$

kth element is in S_R and

kth 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.



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

kth element in first $|S_L|$ elts.

kth elt of S is kth elt of S_L

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

$k \in [|S_L|, \ldots, |S_L| + |S_v|]$.

kth elt of S is in S_v, all have value v

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

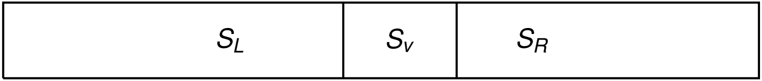
kth element is in S_R and

kth 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.



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

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!

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!

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!

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) ..

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?

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

Expected (average) Time?

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

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

Expected (average) Time?

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

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

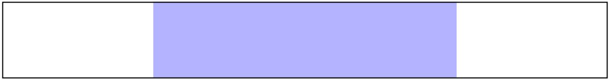
Probability that random pivot elt in the middle half is $\geq \frac{1}{2}$.

Expected (average) Time?

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

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

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

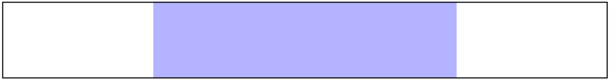


Expected (average) Time?

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 $\geq \frac{1}{2}$.



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

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 (average) Time?

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

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

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

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:

$$T(n) \leq T(\frac{3}{4}n) +$$

Expected (average) Time?

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

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

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:

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

Expected (average) Time?

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

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

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:

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

Masters or just thinking:

Expected (average) Time?

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

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

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:

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

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

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:

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

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 rand. elt b from A.

Extra Example: Deterministic Selection.

Recall Selection of “pivot”:

Choose rand. elt b from A.

Extra Example: Deterministic Selection.

Recall Selection of “pivot”:

Choose 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.

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.
  $S =$ medians of each group.
    $|S|$?

SelectPivot

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

SelectPivot

SelectPivot: A.

Split into groups of size 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

Return **median**(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.

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:

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.

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.

$S =$ 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

5 distinct elements of A.

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
5 distinct elements of A.
Argument picture:

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
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.
$|S|$? $|S| = \frac{n}{5}$.
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:
$A = (\cdots m_1 \cdots) \cdots (a, b, , m_i \cdots) \cdots (\cdots x \cdots) \cdots$
$x \geq m_i \implies x \geq a, b, m_i$

SelectPivot

SelectPivot: A.
Split into groups of size 5.
$S =$ medians of each group.
$|S|$? $|S| = \frac{n}{5}$.
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:
$A = (\cdots m_1 \cdots) \cdots (a, b, , m_i \cdots) \cdots (\cdots x \cdots) \cdots$
$x \geq m_i \implies x \geq a, b, m_i$ or $x \geq 3$ elements of $\frac{1}{2}$ of $\frac{n}{5}$ groups

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
    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

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
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

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
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.
$S =$ 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
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.
  $S =$ 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
    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

$$\frac{1}{2} \times \frac{n}{5} \times 3$$

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
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.
  $S =$ 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
    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

$$\frac{1}{2} \times \frac{n}{5} \times 3 = \frac{3}{10} n \text{ elements.} \qquad \square$$

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.
$S =$ medians of each group.
$|S|$?

SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

SelectPivot: runtime recurrence.

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

SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

Return **median**(S).

Calls median! Runtime $P(n)$?

SelectPivot: runtime recurrence.

SelectPivot: A.

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

Calls median! Runtime $P(n)$?

$P(n) \leq T(\frac{n}{5}) + O(n)$

SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

Return **median**(S).

Calls median! Runtime $P(n)$?

$P(n) \leq T(\frac{n}{5}) + O(n)$

where $T(\cdot)$ is runtime for median.

SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

Return **median**(S).

Calls median! Runtime $P(n)$?

$$P(n) \leq T(\frac{n}{5}) + O(n)$$

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.

  $S =$ medians of each group.

    $|S|$? $|S| = \frac{n}{5}$.

  Return **median**(S).

Calls median! Runtime $P(n)$?

$$P(n) \leq T(\tfrac{n}{5}) + O(n)$$

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

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

$$T(n) \leq P(n) + T(\tfrac{7}{10}n) + O(n).$$

SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

Return **median**(S).

Calls median! Runtime $P(n)$?

$$P(n) \leq T(\tfrac{n}{5}) + O(n)$$

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

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

$$T(n) \leq P(n) + T(\tfrac{7}{10}n) + O(n).$$

Or,

$$T(n) \leq T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + O(n).$$

SelectPivot: runtime recurrence.

SelectPivot: A.

Split into groups of 5.

$S =$ medians of each group.

$|S|$? $|S| = \frac{n}{5}$.

Return **median**(S).

Calls median! Runtime $P(n)$?

$$P(n) \leq T(\tfrac{n}{5}) + O(n)$$

where $T(\cdot)$ is runtime for median.

$T(n)$ recurrence?

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

$$T(n) \leq P(n) + T(\tfrac{7}{10}n) + O(n).$$

Or,

$$T(n) \leq T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + O(n).$$

$O(n)$- compute medians S and for partitioning.

SelectPivot: runtime recurrence.

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

Calls median! Runtime $P(n)$?

$$P(n) \leq T(\frac{n}{5}) + O(n)$$

where $T(\cdot)$ is runtime for median.
$T(n)$ recurrence?
  Middle Lemma: x is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

$$T(n) \leq P(n) + T(\frac{7}{10}n) + O(n).$$

Or,

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + O(n).$$

$O(n)$- compute medians S and for partitioning.

$T(\frac{n}{5})$ for computing median of S.

SelectPivot: runtime recurrence.

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

Calls median! Runtime $P(n)$?

$$P(n) \leq T(\tfrac{n}{5}) + O(n)$$

where $T(\cdot)$ is runtime for median.
$T(n)$ recurrence?
  Middle Lemma: x is $\geq$ (also $\leq$) at least $\frac{3}{10}n$ elements.

$$T(n) \leq P(n) + T(\tfrac{7}{10}n) + O(n).$$

Or,

$$T(n) \leq T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + O(n).$$

$O(n)$- compute medians S and for partitioning.

$T(\frac{n}{5})$ for computing median of S.

$T(\frac{7}{10}n)$ 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.

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$$

Bound Recurrence.

$$T(n) \leq T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$.

Bound Recurrence.

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Bound Recurrence.

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Prove $T(n) \leq c'n$ for some c'.

Bound Recurrence.

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Prove $T(n) \leq c'n$ for some c'.

Bound Recurrence.

$$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$$

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.

$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$

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 $n' < n$.

$$\begin{aligned}
T(n) &\leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\
&\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn \\
&\leq c'\frac{9}{10}n + cn \\
&\leq c'n + (c - c'\frac{1}{10})n
\end{aligned}$$

Bound Recurrence.

$T(n) \le T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Prove $T(n) \le c'n$ for some c'.

Induction Hypothesis: $T(n') \le c'n'$ for $n' < n$.

$$\begin{aligned} T(n) &\le T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\ &\le c'\frac{n}{5} + c'\frac{7}{10}n + cn \\ &\le c'\frac{9}{10}n + cn \\ &\le c'n + (c - c'\frac{1}{10})n \end{aligned}$$

Choose $c' \ge 10c$

Bound Recurrence.

$$T(n) \le T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Prove $T(n) \le c'n$ for some c'.

Induction Hypothesis: $T(n') \le c'n'$ for $n' < n$.

$$\begin{aligned} T(n) &\le T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\ &\le c'\frac{n}{5} + c'\frac{7}{10}n + cn \\ &\le c'\frac{9}{10}n + cn \\ &\le c'n + (c - c'\frac{1}{10})n \end{aligned}$$

Choose $c' \ge 10c \implies c - c'\frac{1}{10} < 0$

Bound Recurrence.

$$T(n) \le T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Prove $T(n) \le c'n$ for some c'.

Induction Hypothesis: $T(n') \le c'n'$ for $n' < n$.

$$\begin{aligned}
T(n) &\le T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\
&\le c'\frac{n}{5} + c'\frac{7}{10}n + cn \\
&\le c'\frac{9}{10}n + cn \\
&\le c'n + (c - c'\frac{1}{10})n
\end{aligned}$$

Choose $c' \ge 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \le c'n$.

Bound Recurrence.

$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$

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 $n' < n$.

$$\begin{aligned}
T(n) &\leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\
&\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn \\
&\leq c'\frac{9}{10}n + cn \\
&\leq c'n + (c - c'\frac{1}{10})n
\end{aligned}$$

Choose $c' \geq 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \leq c'n$.

Base Case: $c' \geq c$.

Bound Recurrence.

$T(n) \leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn. \quad T(1) = c.$

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 $n' < n$.

$$\begin{aligned} T(n) &\leq T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\ &\leq c'\frac{n}{5} + c'\frac{7}{10}n + cn \\ &\leq c'\frac{9}{10}n + cn \\ &\leq c'n + (c - c'\frac{1}{10})n \end{aligned}$$

Choose $c' \geq 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \leq c'n.$

Base Case: $c' \geq c$. □

Bound Recurrence.

$$T(n) \le T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + cn. \quad T(1) = c.$$

Idea: $\frac{1}{5} + \frac{7}{10} < 1$. Problem sizes decrease geometrically.

Prove $T(n) \le c'n$ for some c'.

Induction Hypothesis: $T(n') \le c'n'$ for $n' < n$.

$$\begin{aligned} T(n) &\le T(\frac{n}{5}) + T(\frac{7}{10}n) + cn \\ &\le c'\frac{n}{5} + c'\frac{7}{10}n + cn \\ &\le c'\frac{9}{10}n + cn \\ &\le c'n + (c - c'\frac{1}{10})n \end{aligned}$$

Choose $c' \ge 10c \implies c - c'\frac{1}{10} < 0 \implies T(n) \le c'n$.

Base Case: $c' \ge c$. $\square$

Selection is $O(n)$ deterministic time!

Lecture in a minute.

MergeSort.

Lecture in a minute.

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

Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Comparison splits outputs into 2.

Lecture in a minute.

MergeSort.

Sort two halves, put together.

Merge: two pointer scan.

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Comparison splits outputs into 2.

$\Omega(\log n!) = \Omega(n \log n)$ time.

Lecture in a minute.

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

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

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Comparison splits outputs into 2.

$\Omega(\log n!) = \Omega(n \log n)$ 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.
$T(n) = 2T(\frac{n}{2}) + O(n)$.
$T(n) = O(n \log n)$.
Also: iterative view.

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

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

$T(n) = O(n \log n)$.

Also: iterative view.

Sorting Lower Bound.

$n!$ possible output orderings.

Comparison splits outputs into 2.

$\Omega(\log n!) = \Omega(n \log n)$ 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.