Lecture in a minute.

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

Sorting Lower Bound.
Can we do better?
Comparison sorting algorithm only compares numbers.
How to merge?
Choose lowest from two lists, cross out, repeat.

Extra: Deterministic Pivot Selection.

More divide and conquer: mergesort.

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

Mergesort(\( A \))
if (\( \text{length}(A) > 1 \))
    return
    (merge(mergesort(\( a_1, \ldots, a_n/2 \)),
            mergesort(\( a_n/2+1, \ldots, a_n \))))
else 
    return \( a \)

Check it out...

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

Mergesort: running time analysis

Mergesort(\( A \))
if (\( \text{length}(A) > 1 \))
    return
    (merge(mergesort(\( a_1, \ldots, a_n/2 \)),
            mergesort(\( a_n/2+1, \ldots, a_n \))))
else 
    return \( a \)

Split: \( O(n) \) time
Could be \( O(1) \), e.g., MergeSort(\( A, \text{start}, \text{finish} \)).
Mergesort: time to decrease size by \( 3/4 \).
Expected Time Analysis:
\[ T(n) = O(n) \] 

Apply Masters:
\[ a = 2, b = 2, d = 1 \implies \log_2 2 = 1 \implies T(n) = O(n\log n) \]

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.
\[ \implies O(n\log n) \text{ passes} \implies O(n\log n) \text{ time} \]

CS 170: Algorithms

Hello and ...

H ... H . S . H H
S H H H H H . . . .

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

Bad for your learning. Worse for your neighbors learning.

If you must leave early, please sit by exit.

Thank you!
Sorting lower bound.

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

**Proof idea:** Input: \( a_1, a_2, \ldots, a_n \)
Possible Output: \( a_1, a_2, \ldots, a_n \)
Represent output as permutation of \([1, \ldots, n]\).
Output: \( 8, n-8, \ldots, 15 \).
How many possible outputs? \( n! \)
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.
After a sequence of comparisons get to termination or 1 permutation.

\( S \) is set of possible permutations at some point in Algorithm.
Example: After no comparisons, any output is possible.
Do some comparison: \( a_i > a_j \)?
If Yes, Alg "could" return subset of permutations: \( S_1 \).
If No, Alg "could" return subset of permutations: \( S_2 \).
\( S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2 \).
Each comparison divides possible outputs by at most 2.
Need at least \( \log_2(n!) \) comparisons to get to just 1 permutation.
...to get to termination.
\( n! \geq \left( \frac{n}{2} \right)^n \implies \log n! = \Omega(n \log n) \).

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

???

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 \( \frac{\sum_{i=1}^{n} a_i}{n} \) time.
Compute median? Sort to get \( a_1, \ldots, a_n \). Output element \( a_{\lfloor n/2 \rfloor + 1} \).
\( O(n \log n) \) time.
Better algorithm?

Solve a harder Problem: Selection.

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

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

Base Case: \( k = 1 \) and \( |S| = 1 \), return elt.
Choose rand. pivot elt \( v \) from \( A \).
Form \( S_1 \) containing all ets < \( v \).
Form \( S_2 \) containing all ets > \( v \).
Form \( S_0 \) containing all ets = \( v \).

If \( k \leq |S_2| \): Select \( k \), \( S_2 \).
else if \( k \leq |S_1| + |S_2| \): return \( v \).
else Select \( k - |S_1| - |S_2| \), \( S_0 \).

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.

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

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

k-th element in first \([S_L]\) elts.

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

\( k \)-th elt of \( S \) is \( k \)-th elt of \( S_L \)

else if \( k \leq |S_L| + |S_v| + |S_R| \), return \( v \)

\( k \)-th elt of \( S \) is in \( S_v \), all have value \( v \)

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

\( k \)-th element is in \( S_R \) and

\( k \)-th elt of \( S \) is \( k - |S_L| - |S_v| \) after elts of \( S_L \cup S_v \).

Correct in all cases.

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.)

Extra Example: Deterministic Selection.

Recall Selection of “pivot”:

Choose random elt \( b \) from \( A \).

Expected to be “in the middle”.

Instead: find elt that must be “in the middle.”

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?

Expected (average) Time?

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

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

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

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

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

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

Expected time recurrence:

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

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

\( \implies T(n) = O(n) \)
SelectPivot: runtime recurrence.

SelectPivot: A.
Split into groups of 5.
\( S = \text{medians of each group.} \)
Return \( \text{median} (S) \).
Calls median! Runtime \( P(n) \)?
\[ P(n) \leq T\left(\frac{5}{7}n\right) + O(n) \]
where \( T(\cdot) \) is runtime for median.
Middle Lemma: \( x \geq \frac{3}{11}n \) elements.
\[ T(n) \leq P(n) + T\left(\frac{7}{10}n\right) + O(n) \]
Or,
\[ T(n) \leq T\left(\frac{5}{7}n\right) + T\left(\frac{7}{10}n\right) + O(n) \]
\( O(n) \)-compute medians \( S \) and for partitioning.
\( T\left(\frac{5}{7}n\right) \) for computing median of \( S \).
\( T\left(\frac{7}{10}n\right) \) for the recursive call in Select.

Bound Recurrence.

\[ T(n) \leq T\left(\frac{5}{7}n\right) + T\left(\frac{7}{10}n\right) + cn \]
Idea: \( \frac{5}{7} + \frac{7}{10} < 1 \). Problem sizes decrease geometrically.
Prove \( T(n) \leq c'n \) for some \( c' \).
Induction Hypothesis: \( T(n') \leq c'n' \) for \( n' < n \).
\[ T(n) \leq T\left(\frac{5}{7}n\right) + T\left(\frac{7}{10}n\right) + cn \leq c'n + c'n + cn \leq c'n + (c - c') \frac{1}{10}n \]
Choose \( c' \geq 10c \Rightarrow c - c' \frac{1}{10} < 0 \Rightarrow T(n) \leq c'n. \]
Base Case: \( c' \geq c. \) \( \square \)
Selection is \( O(n) \) deterministic time!

Lecture in a minute.

MergeSort.
Sort two halves, put together.
Merge: two pointer scan.
\[ T(n) = 2T\left(\frac{n}{2}\right) + O(n) \]
Also: iterative view.
Sorting Lower Bound.
\( n! \) possible output orderings.
Comparison splits outputs into 2.
\( \Omega(n \log n) = \Omega(n \log n) \) 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.