Problem solving.

How do you sort a set of objects?

In high school. Before you came to Berkeley.

How would you sort by yourself?

Talk to your neighbor!

How would you sort with two of you?

Do some work splitting up?

Do some work putting together?

Talk to your neighbor.

By yourself: $O(n^2)$

Pick the largest one,
set it aside,
sort the rest,
and put the largest last.

Choose the largest one: $O(n)$.

Do this n times.

$O(n^2)$.

Two of you: divide and conquer.

Split the objects in half.

Each of you sort.

Merge the sorted piles.

Divide and conquer.

Two of you: divide and conquer.

Split the objects into a larger and smaller group.

One sorts the larger one, the other the small one.

Put together.

Divide and conquer.

What's the point of all this?

You are already problem solvers!

We are giving tools to be more precise ...
and powerful (with recursion and programming and algorithms.)

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 ...

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

Check it out...

Iterative Mergesort: Bottom up, use of queues.

Make each element into list and put lists in queue.

[8] [3] [5] [9] ......

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

[5] [9] ...... [3,8]

Rinse. Repeat.

...... [3,8] [5,9]

And next pass through queue...

...... [3,5,8,9]

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

Sorting lower bound.

Can we do better?

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.

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.
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.

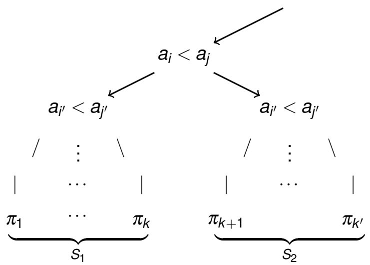
$S_1 \cup S_2 = S \implies \max(|S_1|, |S_2|) \geq |S|/2$.

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.



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".

Radix sort and lower bound.

Why is radix sort not subject to lower bound?

It buckets by 'letter'.

So, the degree of the tree is not 2. It is larger.

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.

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.

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

S_L	S_v	S_R

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?
(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?

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

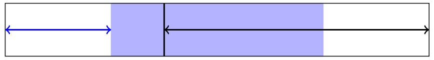
(A) ..and (C) (but not relevant.)

Expected (average) Time?

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

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

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: $(n + (3/4)n + (3/4)^2 n + \cdots = O(n))$
$\implies T(n) = O(n)$.

Other sorting method.

Find median.
Partition elements into halves according to median.
Sort each half.

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

Randomized sorting method: quicksort.

Choose a random element to be partition elt, p.
Split into larger and smaller elements based on p.

Recurse on each group.

$T(n) =????$

How do you analyse this?

Master Theorem doesn't apply...

Also, how does one do average time?

Expected Time of algorithm on n elements.

One method: amortized analysis.

From the point of view of an element.

Begin: size of elt's group is n.

Each iteration: size of elt's group gets smaller.

Size of the group become $\leq 3n/4$ if ...
an elt in middle fourth is partition elt.

X - number of iterations for group to be small enough.
We know $X \sim G(\frac{1}{2})$.
We did this before.

$E[X] = 1/2 \times (1 + E(X)) + 1/2 \times (1)$
or $E[X] = 2$.

How much expected work on this element?
$E[n] = 2 + E[3/4n]$
Unfold: $E[n] = 2 + 2 + \cdots + 2$
The number of 2's is $\log_{4/3} n$.
So, $O(\log n)$ per element, or $O(n \log n)$ overall elements.

Conclusion: More divide and conquer.

You came up with iterative,
and divide and conquer algorithms.

Iterative: $O(n^2)$.

Mergesort: $O(n \log n)$.
Divide in two, sort each, merge.

Median: $O(n)$ expected time.
Do selection.
Pick random partition, recurse on the correct side.
Analysis: after an average of two steps, problem smaller by $3/4$.
Geometric series of runtimes.

Quicksort: $O(n \log n)$ expected time.
Random partition.
Work per/element is constant in each iteration.
Expected $O(\log n)$ iterations.
$O(n \log n)$ work for n elements.