Welcome to CSE21!

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Instructor</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture B</td>
<td>Miles Jones</td>
<td>MWF 9-9:50pm</td>
<td>PCYN 109</td>
</tr>
<tr>
<td>Lecture D</td>
<td>Russell (Impagliazzo)</td>
<td>MWF 4-4:50pm</td>
<td>Center 101</td>
</tr>
</tbody>
</table>

http://cseweb.ucsd.edu/classes/sp16/cse21-bd/

March 30, 2016
* Assume elements of the set to be sorted have some underlying order
A TA facing a stack of exams needs to input all 400 scores into a spreadsheet where the students are listed in alphabetical order.

OR

You want to find all the duplicate values in a long list.

It's easier to access data when it is sorted because you know exactly where to find it.
General questions to ask about algorithms

1) **What** problem are we solving?
2) **How** do we solve the problem?
3) **Why** do these steps solve the problem?
4) **When** do we get an answer?
General questions to ask about algorithms

1) **What** problem are we solving? _PROBLEM SPECIFICATION_

2) **How** do we solve the problem? _ALGORITHM DESCRIPTION_

3) **Why** do these steps solve the problem? _CORRECTNESS_

4) **When** do we get an answer? _RUNNING TIME PERFORMANCE_
Given a list

\[ a_1, a_2, \ldots, a_n \]

rearrange the values so that

\[ a_1 \leq a_2 \leq \ldots \leq a_n \]

Values can be any type (with underlying total order). For simplicity, use integers.
Given a list

\[ a_1, a_2, \ldots, a_n \]

rearrange the values so that

\[ a_1 \leq a_2 \leq \ldots \leq a_n \]

What is the correct sorted ordering of 5, 7, 5, 3, 9
A. 9,7,5,3
B. 9,7,5,5,3
C. 3,5,7,9
D. 3,5,5,7,9
Your approaches: HOW

- Selection (min) sort
- Bubble sort
- Insertion sort
- Bucket sort
- Merge sort
- Bogo sort
- Quick sort
- Binary search tree traversal

"Find the first name alphabetically, move it to the front. Then look for the next one, move it, etc."
procedure selection sort($a_1, a_2, ..., a_n$: real numbers with $n \geq 2$)
for $i := 1$ to $n-1$
  $m := i$
  for $j := i+1$ to $n$
    if ($a_j < a_m$) then $m := j$
  interchange $a_i$ and $a_m$

{$a_1, ..., a_n$ is in increasing order}
Bubble Sort

"Compare the first two cards, and if the first is bigger, keep comparing it to the next card in the stack until we find one larger than it. Repeat until the stack is sorted."
procedure bubble sort(a₁, a₂, ..., aₙ: real numbers with n >=2 )
for i := 1 to n-1
    for j:= 1 to n-i
        if ( aⱼ > aⱼ₊₁ ) then interchange aⱼ and aⱼ₊₁ 

{ a₁, ..., aₙ is in increasing order}
"We passed the cards from right to left, each individual inserting their own card in the correct position as they relayed the pile."
procedure insertion sort(a₁, a₂, ..., aₙ: real numbers with n ≥ 2)
for j := 2 to n
  i := 1
  while aₐ > aᵢ
    i := i+1
  m := aⱼ
  for k := 0 to j-i-1
    aⱼ-k := aⱼ-k-1
    aᵢ := m

{ a₁, ..., aₙ is in increasing order}
"Call out from A to Z, collecting cards by first letter. If there are more than one with the same first letter, repeat with the second letter, and so on."
Bucket Sort – Pseudo pseudo code

• Create empty buckets that have an ordering.
• Put each of the elements of the list into the correct bucket.
• Sort within each bucket.
• Concatenate the buckets in order.
"We split into two groups and organized each of the groups, then got back together and figured out how to interleave the groups in order."
Merge Sort – Pseudo pseudo code

• If the list has just one element, return.
• Otherwise,
  • Divide list into two pieces:
    \[ L_1 = a_1 \ldots a_{n/2} \text{ and } L_2 = a_{n/2+1} \ldots a_n \]
  • \( M_1 = \text{Merge sort}(L_1) \)
  • \( M_2 = \text{Merge sort}(L_2) \)
  • Merge the two (sorted) lists \( M_1 \) and \( M_2 \)
Why so many algorithms?
Why so many algorithms?

Practice for homework / exam / job interviews.

Some algorithms are better than others. Wait, *better*?

Sorting is ubiquitous because it organizes data.

Knuth: “Computer manufacturers of the 1960’s estimated that more than 25% of the running time of their computers was spent on sorting.”

That was 50 years ago but now modern data centers spend much of their time sorting.
Why so many algorithms?

Sorting is important in different environments with different constraints. E.g. low power for your iphone or needing to sort in place when there is no excess memory.

Different algorithms are optimized for different information, e.g. Bucket sort is useful when data is uniformly distributed.

The more you know about your data set, the better you can optimize
From "How" to "Why"

What makes this algorithm work?

How do you know that the resulting list will be sorted?

*For loop-based algorithms:*

What's the effect of each loop iteration on the list?

Have we made progress?
A loop invariant is a property that remains true after each time the body of a loop is executed.

For an iterative algorithm:

- **Step 1: Look for a loop invariant**
  - State the property precisely
- **Step 2: Prove that it is invariant**
  - It must be true after any number of loop iterations
- **Step 3: Use the invariant to prove correctness**
  - Show that when the loop is finished, the invariant guarantees that we've reached a solution
Selection Sort (MinSort)

"Find the first name alphabetically, move it to the front. Then look for the next one, move it, etc."
procedure selection sort($a_1$, $a_2$, ..., $a_n$: real numbers with $n \geq 2$)
for $i := 1$ to $n-1$
    $m := i$
    for $j := i+1$ to $n$
        if ($a_j < a_m$) then $m := j$
    interchange $a_i$ and $a_m$

{ $a_1$, ..., $a_n$ is in increasing order}
procedure selection sort(a_1, a_2, ..., a_n: real numbers with n >=2 )
for i := 1 to n-1
    m := i
    for j:= i+1 to n
        if (a_j < a_m) then m := j
    interchange a_i and a_m

{ a_1, ..., a_n is in increasing order}
Selection Sort

In groups of 3 or so …. Discuss possible invariants for Selection sort. Remember that an invariant is true for all iterations, not just at the end.
Selection Sort (MinSort) Correctness: WHY

**Loop invariant:** After the $k$-th time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.
Selection Sort (MinSort) Correctness: WHY

Loop invariant: After the $k^{th}$ time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.

How can we show that this loop invariant is true?

Once we do, why can we conclude that the program is correct?
Loop invariant: After the $k^{th}$ time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.

How can we show that this loop invariant is true?

Once we do, why can we conclude that the program is correct?
**Loop invariant:** After the $k^{th}$ time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.

Therefore, **pick a correct conclusion that proves the correctness of MinSort**

A. since the first $k$ elements are the $k$ smallest elements, MinSort is correct.
B. since we execute the loop at least once, the first element at the end of the algorithm is the smallest and so MinSort is correct.
C. since we execute the loop exactly $n$ times, the loop invariant with $k=n$ guarantees that all $n$ elements of the list will be in correct order at the end of the algorithm, as required.
D. None of the above.
Loop invariant: After the $k^{th}$ time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.

How can we show that this loop invariant is true?

Once we do, why can we conclude that the program is correct?
Loop invariant: After the $k$th time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.

How can we show that this loop invariant is true?

What will be the relevant induction variable?

A. $n$, the number of elements in the list
B. $k$, the number of times we go through the outer loop
C. $i$, the index of the smallest element in the list
D. None of the above.
Statement: After the $k^{th}$ time through the outer loop, the first $k$ elements of the list are the $k$ smallest list elements in order.

Induction variable ($k$): *the number of times through the loop.*

**Base case:** Need to show the statement holds for $k=0$, before the loop.

**Inductive step:** Let $k$ be a positive integer.

**Induction hypothesis:** Suppose the statement holds after $k-1$ times through the loop.

**Need to show** that the statement holds after $k$ times through the loop.
procedure selection sort(a₁, a₂, ..., aₙ: real numbers with n ≥ 2)
for i := 1 to n-1
    m := i
    for j := i+1 to n
        if (aⱼ < aₘ) then m := j
    interchange aᵢ and aₘ
{ a₁, ..., aₙ is in increasing order}
Proving Loop Invariants

Induction variable (k): *the number of times through the loop.*

**Base case:** Need to show the statement holds for k = 0, before the loop begins.

**Inductive step:** Let k be a positive integer.

**Induction hypothesis:** Suppose the statement holds after k - 1 times through the loop.

Need to show that the statement holds after k times through the loop.
Why sort?

A TA facing a stack of exams needs to input all 400 scores into a spreadsheet where the students are listed in alphabetical order.

OR

You want to find all the duplicate values in a long list.

It's easier to access data when it is sorted because you know exactly where to find it.
Two searching algorithms:

One that works for any data, sorted or not

One that is much faster, but relies on the data being sorted

More on this next time!
Reminders

HW 1 due **tonight 11:59pm.**