Sorting

COP 3502
Let’s say we have a list of the names of people in the class and we want to sort alphabetically.

- We are going to describe an algorithm (or systematic methods) for putting these names in order.
- The algorithms we will cover today:
  - Selection Sort
  - Insertion Sort
  - Bubble Sort

Image of people named BOB, JOE, ABE, SAM, and ANN.
Sorting a List

- **Selection Sort**
  - Finds the smallest element (alphabetically the closest to a)
    - Swaps it with the element in the first position
  - Then finds the second smallest element
    - Swaps it with the element in the second position
  - Etc. until we get to the last position, and then we’re done!
Selection Sort

Min = “Abe”

Sorting a List

- **Selection Sort**
  - Finds the smallest element (alphabetically the closest to a)
    - Swaps it with the element in the first position
  - Then finds the second smallest element
    - Swaps it with the element in the second position
  - Etc. until we get to the last position, and then we’re done!
Selection Sort

Min = “Ann”

Sorting a List

- **Selection Sort**

  Min = “Bob”

  “Sam” < “Bob”? “Joe” < “Bob”?
Selection Sort

Notice that now the list is **sorted**!
So we can stop when \texttt{Curr} is on the 2\textsuperscript{nd} to last element.
Sorting a List

- **Insertion Sort**
  - Take each element one by one, starting with the second and “insert” it into the already sorted list to its left in the correct order.

BOB  JOE  ABE  ANN  SAM
Sorting a List

**Insertion Sort**

- **PRE V**
- **CUR R**

“Joe” < “Bob”?  
Pos = 1
Sorting a List

- **Insertion Sort**

  - “Abe” < “Bob”?  
  - “Abe” < “Joe”?  
  - Pos = 2
Sorting a List

- **Insertion Sort**

  "Sam" < "Joe"?

  Pos = 3

ABE  BOB  JOE  SAM  ANN
Sorting a List

- **Insertion Sort**

  "Ann" < "Abe"?  "Ann" < "Bob"?  "Ann" < "Joe"?  "Ann" < "Sam"?

  Pos = 4

  ABE  BOB  BOH  JOE  SAN
Sorting a List

**Bubble Sort**
- The basic idea behind bubble sort is that you always compare consecutive elements, going left to right.
  - Whenever two elements are out of place, swap them.
  - At the end of a single iteration, the max element will be in the last spot.
- Now, just repeat this n times
- On each pass, one more maximal element will be put in place.
- As if the maximum elements are slowly “bubbling” up to the top.
Sorting a List

- **Bubble Sort**

  - “Bob” > “Joe”?
  - “Joe” > “Abe”?
  - “Joe” > “Sam”?
  - “Sam” > “Ann”?
Sorting a List

- **Bubble Sort**

  \[ \text{Curr} \quad \text{Curr} \quad \text{Curr} \quad \text{Next} \]

  “Bob” > “Abe”? \quad “Bob” > “Joe”? \quad “Joe” > “Ann”?
Sorting a List

- **Bubble Sort**

  - “Abe” > “Bob”?
  - “Bob” > “Ann”?

  ![Comparer](abe.png) ![Comparer](bob.png) 

  ![Comparer](ann.png)
Sorting a List

- **Bubble Sort**

  "Abe" > "Anne"?

  - ABE
  - ANN
  - BOB
  - JOE
  - SAM
Limitation of Sorts that only swap adjacent elements

A sorting algorithm that only swaps adjacent elements can only run so fast.

- In order to see this, we must first define an inversion:
  - An inversion is a pair of numbers in a list that is out of order.
  - In the following list: 3, 1, 8, 4, 5
  - the inversions are the following pairs of numbers: (3, 1), (8, 4), and (8, 5).
- When we swap adjacent elements in an array, we can remove at most one inversion from that array.
Limitation of Sorts that only swap adjacent elements

- Note that if we swap non-adjacent elements in an array, we can remove multiple inversions. Consider the following:
  - 8 2 3 4 5 6 7 1
    - Swapping 1 and 8 in this situation removes every inversion in this array (there are 13 of them total).

- Thus, the run-time of an algorithm that swaps adjacent elements only is constrained by the total number of inversions in an array.
Limitation of Sorts that only swap adjacent elements

- Let's consider the average case.
  - There are \( \binom{n}{2} = \frac{(n-1)n}{2} \) pairs of numbers in a list of \( n \) numbers.
    - Of these pairs, on average, half of them will be inverted.
  - Thus, on average, an unsorted array will have
    \[
    \frac{(n-1)n}{4} = \Omega(n^2)
    \]
    number of inversions,
    - and any sorting algorithm that swaps adjacent elements only will have a \( \Omega(n^2) \) run-time.