Sorting!

Sorting is the process of ordering a bunch of items based on one or more rules, subject to one or more constraints…

Items – what are we sorting, and how many are there?

- Strings, numbers, student records, C++ objects (e.g., Circles, Robots)
- Thousands, millions or trillions?

Rules – how do we order them?

- Ascending ↗ vs. Descending ↘ order
- Based on Circle radius? Student GPA?
- Based on multiple criteria, e.g.: by last name, then first name

Constraints?

- Are the items in RAM or on disk?
- Is the data in an array or a linked list?

Carey's 2 Rules of Sorting

Rule #1:

Don't choose a sorting algorithm until you understand the requirements of your problem.

Rule #2:

Always choose the simplest sorting algorithm possible that meets your requirements.

The Selection Sort

- Look at all N books, select the shortest book
- Swap this with the first book
- Look at the remaining N-1 books, and select the shortest
- Swap this book with the second book
- Look at the remaining N-2 books, and select the shortest
- Swap this book with the third book and so on…

So, is our sort efficient?

If we have N books, how many steps does it take to sort them?

Let's assume a step is any time we either swap a book or point our finger at a book.

The Selection Sort- Speed

- Look at all N books, select the shortest book — **N steps**
- Swap this with the first book — **1 step**
- Look at the remaining N-1 books, and select the shortest — **N-1 steps**
- Swap this book with the second book — **1 step**
- Look at the remaining N-2 books, and select the shortest — **N-2 steps**
- Swap this book with the third book and so on… — **1 step**

So this comes to:

N swap steps

PLUS

N + N-1 + N-2 + … + 2 + 1 steps to find the smallest item

So Selection Sort is $O(N^2)$

Or, for N books, you need roughly N^2 steps to sort them.

(It's considered pretty slow)

Selection Sort – Better or Worse?

Are there any kinds of input data where Selection Sort is either more or less efficient?

For example, what if all of the books are mostly in order before our sort starts?

```
void selectSort(shelf of N books)
{
  for i = 1 to N
  {
     find the smallest book
       between slots i and N

     swap this smallest book
       with book i;
  }
}
```

No! Selection sort takes just as many steps either way!

The Se

And here's the C++ source code to sort a bunch of numbers…

```
void selectionSort(int A[], int n)
{
  for (int i = 0; i < n; i++)
  {
    int minIndex = i;
    for (int j = i+1; j < n; j++)
    {
      if (A[j] < A[minIndex])
        minIndex = j;
    }
    swap(A[i], A[minIndex]);
  }
}
```

For each of the n array elements…

Locate the smallest item in the array between the i^{th} slot and slot n-1.

Swap the smallest item found with slot A[i].

Selection Sort Questions

Can Selection Sort be applied easily to sort items within a linked list?

Is Selection Sort "stable" or "unstable"?

What's a Stable Sort?

Imagine that N old people line up to buy laxatives at a drugstore.

And the drugstore wants to sort them and serve them based on urgency.

The drugstore needs to pick a sort algorithm to re-order the guests. They can choose between a "stable" sort or an "unstable" sort.

An "unstable" sorting algorithm re-orders the items without taking into account their initial ordering.

A "stable" sorting algorithm does take into account the initial ordering when sorting, maintaining the order of similar-valued items.

As you solve problems (in class or at work) you should choose your sort depending on whether stability is important. If you forget the concept, just remember the laxatives! ☺

People in line	Unstable Sort Results	Stable Sort Results
Ebeneezer - 8 days	Steve – 8 days	Ebeneezer - 8 days
Carey – 5 days	Vicki – 8 days	Steve – 8 days
David – 2 days	Ebeneezer - 8 days	Vicki – 8 days
Michael – 4 days	Andrea – 5 days	Carey – 5 days
Steve – 8 days	Carey – 5 days	Andrea – 5 days
Vicki – 8 days	Michael – 4 days	Michael – 4 days
Andrea – 5 days	David – 2 days	David – 2 days

The Se

And here's the C++ source code to sort a bunch of numbers…

```
void selectionSort(int A[], int n)
{
  for (int i = 0; i < n; i++)
  {
    int minIndex = i;
    for (int j = i+1; j < n; j++)
    {
      if (A[j] < A[minIndex])
        minIndex = j;
    }
    swap(A[i], A[minIndex]);
  }
}
```

Selection Sort Questions

Can Selection Sort be applied easily to sort items within a linked list?

Is Selection Sort "stable" or "unstable"?

When might you use Selection Sort?

Here's a hint – consider this array:

10	10	1

When Selection Sort finds the 1, it swaps it with the first 10.

Then our array ends up like this:

1	10	10

The Insertion Sort

Well, we couldn't just teach you one sort, right?

Let's learn another!

The insertion sort is probably the most common way…

to sort playing cards!

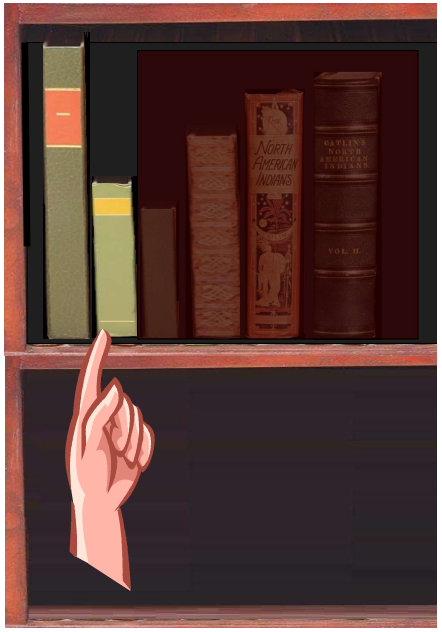
(But I'll still explain the sort with library books)

The Insertion Sort

Let's focus on the first two books – ignore the rest.

- If the last book in this set is in the wrong order
  - Remove it from the shelf
  - Shift the book before it to the right
  - Insert our book into the proper slot

Great! Now our first two books are in sorted order (ignoring the others)



The Insertion Sort

Ok, now focus on the first three books – ignore the rest.

- If the last book in this set is in the wrong order
  - Remove it from the shelf
  - Shift the books before it to the right, as necessary
  - Insert our book into the proper slot

Great! Now our first three books are in sorted order (ignoring the others)

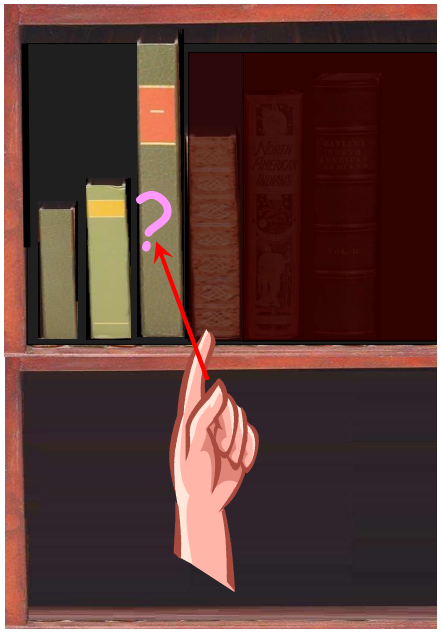
The Insertion Sort

Ok, now focus on the first four books – ignore the rest.

- If the last book in this set is in the wrong order
  - Remove it from the shelf
  - Shift the books before it to the right, as necessary
  - Insert our book into the proper slot

Great! Now our first four books are in sorted order!

We just keep repeating this process until the entire shelf is sorted!



The Insertion Sort

So what's the complete algorithm?

Start with set size s = 2

While there are still books to sort:

- Focus on the first s books
- If the last book in this set is in the wrong order
  - Remove it from the shelf
  - Shift the books before it to the right, as necessary
  - Insert our book into the proper slot
- s = s + 1

The Insertion Sort - Speed

So what's the Big-O of our Insertion Sort?

During each round of the algorithm we consider a larger set of books.

During the first round, we may need to shift up to one book to find the right spot.

The Insertion Sort - Speed

So what's the Big-O of our Insertion Sort?

During each round of the algorithm we consider a larger set of books.

During the first round, we may need to shift up to one book to find the right spot.

During the second round, we may need to shift up to two books to find the right spot.

2 shifts →

The Insertion Sort - Speed

So what's the Big-O of our Insertion Sort?

During each round of the algorithm we consider a larger set of books.

During the first round, we may need to shift up to one book to find the right spot.

During the second round, we may need to shift up to two books to find the right spot.

• • •

During the last round, we may need to shift up to N-1 books to find the right spot.

N-1 shifts →

1 step in round 1
+ 2 steps in round 2
+ …
+ N-1 steps in last rnd
= roughly N^2 steps

Thus, Insertion Sort is $O(N^2)$, and is generally quite slow!

Insertion Sort – Better or Worse?

Are there any kinds of input data where Insertion Sort is either more or less efficient?

Any ideas?

Right! If all books are in the proper order…

then Insertion Sort never needs to do any shifting!

In this case, it just takes roughly ~N steps to sort the array! O(N)

Conversely, a perfectly mis-ordered set of books is the worst case.

Since every round requires the maximum shifts!

The Insertio

And here's the C++ version sorts an array in ascending

Insertion Sort Questions

Can Insertion Sort be applied easily to sort items within a linked list?
Is Insertion Sort a "stable" sort?
When might you use Insertion Sort?

```
void insertionSort(int A[], int n)
{
  for(int s = 2; s <= n; s++)
  {
     int sortMe = A[ s - 1 ];

     int i = s - 2;
     while (i >= 0 && sortMe < A[i])
     {
         A[i+1] = A[i];
         --i;
     }

    A[i+1] = sortMe;
  }
}
```

Focus on successively larger prefixes of the array. Start with the first s=2 elements, then the first s=3 elements…

Make a copy of the last val in the current set – this opens up a slot in the array for us to shift items!

Shift the values in the focus region right until we find the proper slot for sortMe.

Store the sortMe value into the vacated slot.

Everyone loves to make fun of the:

Sort

But it's actually quite simple… And sometimes simple is good!

Ok, what's the algorithm?

Start at the top element of your array

Compare the first two elements: A[0] and A[1]
If they're out of order, then swap them

Then advance one element in your array
Compare these two elements: A[1] and A[2]
If they're out of order, swap them

…

Repeat this process until you hit the end of the array

When you hit the end, if you made at least one swap, then repeat the whole process again!

The Bubble Sort

Bubble Sort Questions

Can Bubble Sort be applied easily to sort items within a linked list?
Is Bubble Sort a "stable" sort?
Is Bubble Sort ever faster than $O(n^2)$?
When might you use Bubble Sort?

```
void bubbleSort(int Arr[], int n)
{
  bool atLeastOneSwap;

  do
  {
    atLeastOneSwap = false;

    for (int j = 0; j < (n-1); j++)
    {
      if (Arr[j] > Arr[j + 1])
      {
        Swap(Arr[j],Arr[j+1]);
        atLeastOneSwap = true;
      }
    }
  }
  while (atLeastOneSwap == true);
}
```

Start by assuming that we won't do any swaps

Compare each element with its neighbor and swap them if they're out-of-order.

Don't forget-we swapped!

If we swapped at least once, then start back at the top and repeat the whole process.