What You’ll Learn Today

– What is sorting?
– Why does sorting matter?
– How is sorting accomplished?
– Why are there different sorting algorithms?
– On what basis should we compare algorithms?
Sorting

**Sorting**
Arranging items in a collection so that there is a natural ordering on one (or more) of the fields in the items.

**Sort Key**
The field (or fields) on which the ordering is based.

**Sorting Algorithms**
Algorithms that order the items in the collection based on the sort key.

Why Does Sorting Matter?

We as humans have come to expect data to be presented with a natural ordering (e.g. alphabetic, numeric, etc.).

Finding an item is much easier when the data is sorted. *Why?*
Why Does Sorting Matter?

Sorting is an operation which takes up a lot of computer cycles.

*How many cycles?*

It depends:
- *How many items to be sorted*
- *Choice of sorting algorithm*

*What does this mean to the user?*

---

How would you sort it?

Suppose you have these 7 cards, and you need to put them in ascending order:

![Card images]

Describe your process in pseudo code.
Take a minute or two to write it down.
Sorting Example (1/7)

Find the card with the minimum value:

5♣ 7♣ 3♣ 6♣ 4♣ 2♣ 

Put it in the “ordered” set:

A♣ 

Sorting Example (2/7)

Find the card with the minimum value:

5♣ 7♣ 3♣ 6♣ 4♣ 2♣ 

Put it in the “ordered” set:

A♣ 2♣
Sorting Example (3/7)

Find the card with the minimum value:

```
5 7 8 6 4 2
```

Put it in the “ordered” set:

```
A 2 3
```

---

Sorting Example (4/7)

Find the card with the minimum value:

```
5 7 8 6 4 2
```

Put it in the “ordered” set:

```
A 2 3 4
```
Sorting Example (5/7)

Find the card with the minimum value:

```
X X X X X
```

Put it in the “ordered” set:

```
A 2 3 4 5
```
Sorting Example (7/7)

Find the card with the minimum value:

Put it in the “ordered” set:

Sorting: pseudo code

Given a set of values, put in ascending order:

Start a new pile for the “sorted” list
While length of “original” list > 0:
    Find minimum, copy to “sorted” list
    Remove value from the “original” list

This is called a selection sort.
Another Selection Sort

A slight adjustment to this manual approach does away with the need for a second list:

- As you cross a value off the original list, a free space opens up.
- Instead of writing the value found on a second list, exchange it with the value currently in the position where the crossed-off item should go.

Selection Sort Example

Find the card with the minimum value:

```
5 ♠  7 ♠  5 ♠  A ♠  6 ♠  4 ♠  2 ♠
```

Exchange it with “first” unsorted item:

```
A ♠  7 ♠  3 ♠  5 ♠  6 ♠  4 ♠  2 ♠
```

Unsorted portion
Selection Sort Example

Find the card with the minimum value:

Exchange it with “first” unsorted item:

Unsorted portion
Selection Sort Example

Find the card with the minimum value:

Exchange it with “first” unsorted item:

Unsorted portion

Which item comes first?

Think about writing the code for this.
How do you find the first element in a list?

\[
\text{min} = \text{first item in list} \\
\text{Go through each item in the list:} \\
\quad \text{if item} < \text{min:} \\
\quad \quad \text{min} = \text{item}
\]

How many comparisons does it take to find min?
Consider a list of size 10.
Calculating the Running Time

We measure the running time of an algorithm by the number of operations it requires.

Most of the work of sorting is making comparisons between pairs of items to see which comes first.

Thus our basic question:

*How many comparisons must be done?*

Calculating the Running Time

How can we determine the number of steps required to sort a list of n items?

- Selection Sort requires n comparisons to find the next unsorted item.*
- This process must be repeated n times, to sort all items on the list.*

Thus, we can say that it will require n passes through n items to complete the sort.

\[ n \text{ times } n = n^2 \text{ steps} \]

We call Selection Sort an \(O(n^2)\) algorithm.

* A mathematical simplification has been made. An explanation follows for those who care.
* A Mathematical Footnote

Of course, we don’t really need to always compare every item in the list. Once part of the list is sorted, we can ignore that part and do comparisons against the unsorted part of the list. So for a list of size $n$, we really need to make:

$$n + (n-1) + (n-2) + \ldots + 1$$

comparisons.

This series simplifies to:

$$\frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

comparisons.

This is indeed less than $n^2$. However, as $n$ becomes sufficiently large, it is the $n^2$ part which dominates the equation’s result. We make a simplification in notation and say that these algorithms are “on the order of magnitude of” $n^2$.

Hence the notation of $O(n^2)$ algorithm.

* A Mathematical Footnote

Actually, the running time is $(n^2-n)/2$, but as $n$ becomes sufficiently large, the $n^2$ part of this equation dominates the outcome. Hence the notation of $O(n^2)$ algorithm.
Another Algorithm: Bubble Sort

Bubble Sort uses the same strategy:
- Find the next item.
- Put it into its proper place.

But uses a different scheme for finding the next item:
- Starting with the last list element, compare successive pairs of elements, swapping whenever the bottom element of the pair is smaller than the one above it.

The minimum “bubbles up” to the top (front) of the list.

Bubblesort Example (1/6)

First pass: comparing last two items:

Swap if needed:
Bubblesort Example (2/6)

First pass: compare next pair of items:

Swap if needed:

Bubblesort Example (3/6)

First pass: compare next pair of items:

Swap if needed:
Bubblesort Example (4/6)

First pass: compare next pair of items:

Swap if needed:

Bubblesort Example (5/6)

First pass: compare next pair of items:

Swap if needed:
Bubblesort Example (6/6)

First pass: compare next pair of items:

Swap if needed:

Swap if needed:

Bubblesort Example

After first pass:
We have 1 sorted item and 6 unsorted items:

Notice: all other items are slightly “more sorted” then they were at the start.
After second pass:
We have 2 sorted items and 5 unsorted items:

![Unsorted cards]

Notice: all other items are slightly “more sorted” then they were at the start.

---

After third pass:
We have 3 sorted items and 4 unsorted items:

![Unsorted cards]

Notice: all other items are slightly “more sorted” then they were at the start.
Bubblesort Example

After fourth pass:
We have 4 sorted items and 3 unsorted items:

Notice: all other items are slightly “more sorted” then they were at the start.

Bubblesort Example

After fifth pass:
We have 5 sorted items and 2 unsorted items:

No, the last two items are not sorted yet!
Why not?
After sixth pass, all items have been sorted:

```
A 2 3 4 5 6 7

Bubblesort Example
```

Calculating the Running Time

How do we calculate the running time for Bubble Sort? Determine the number of comparisons.

For a list of size $n$:
- Bubble Sort will go through the list $n$ times
- Each time compare $n$ adjacent pairs of numbers.*

$n \text{ times } n = n^2 \text{ steps}$

Bubble Sort is also an $O(n^2)$ algorithm.

* A mathematical simplification has been made. See previous footnote.
Sorting Algorithm Demo

A really cool graphical demo of different sorting algorithms running side-by-side:
http://www.cs.bu.edu/courses/cs101/demos/sorts.html

(with thanks to Penny Ellard for the original page)

Also, check this out:
http://www.sorting-algorithms.com/

What You Learned Today

– Sorting, sort key, sort algorithms
– Selection sort
– Bubble sort
– Running time analysis
Announcements and To Do List

- HW11 due TUE 11/23
- Readings:
  - http://www.sorting-algorithms.com/ (today)
  - http://www.heroku.cc/teach/HO/dbms/dbms.htm (Tuesday)