front

CS1134 Data Structures

Sorting

Overview

- Why?
- Simple Implementations (i.e. already covered, well mostly)
- More advanced

front

Why?

- Why sort?
  - That can't seriously be a question.
  - We often need things in order.
  - Some algorithms even depend on them being in order.
- Why *study* sorting?
  - Ah! You may well ask!
  - After all, you will always have access to a function or utility to sort for you.
  - But the different designs of sorting algorithms show interesting aspects of problem solving.

Basic sort algorithms

- **Insertion sort**
  - assume lower items already sorted and insert another item “from the top”
- **Bubble sort**
  - Repeatedly start from 0 and “bubble” everything up.
- **Selection sort**
  - From index 0 on up, find the smallest item and swap with index.
- **Heap sort**
  - Ok, not so basic, but if you already have a heap...

front

Insertion Sort

```
def insertion_sort(seq):
    """Sort the sequence in place using insertion sort"""
    # Note that we start at index of one because the single
    # element at index zero is already "sorted"
    for cur in range(1, len(seq)):
        # swap your way down till something's not smaller.
        for j in range(cur, 0, -1):
            if seq[j] < seq[j-1]:
                seq[j-1], seq[j] = seq[j], seq[j-1] # swap
            else: break  # We are not smaller
```

Bubble Sort

```
def bubble_sort(seq):
    """Sort the sequence in place using bubble sort"""
    for i in range(len(seq)):
        # Bubble up
        for j in range(len(seq)-1):
            if (seq[j] > seq[j+1]):
                seq[j], seq[j+1] = seq[j+1], seq[j]  # swap
```

front

Bubble Sort (did we change anything?)

```
def bubble_sort(seq):
    """Sort the sequence in place using bubble sort"""
    for i in range(len(seq)):
        done = True  # Switch to False if we swap anything
        # Bubble up
        for j in range(len(seq)-1):
            if (seq[j] > seq[j+1]):
                seq[j], seq[j+1] = seq[j+1], seq[j]
                done = False
        if done: break
```

Best case, i.e. already sorted, is now linear instead of quadratic. Wow!

Bubble Sort (small optimization)

```
def bubble_sort(seq):
    """Sort the sequence in place using bubble sort"""
    for i in range(len(seq)):
        done = True  # Switch to False if we swap anything
        # Bubble up
        for j in range(len(seq)-1 - i): # Stop early!!!
            if (seq[j] > seq[j+1]):
                seq[j], seq[j+1] = seq[j+1], seq[j]
                done = False
        if done: break
```

Run time in all cases is cut in half. Double wow!

front

Selection Sort

```
def selection_sort(seq):
    """Sort the sequence in place using selection sort"""
    for i in range(len(seq)):
        # Find the smallest item starting at index i
        min_index = i
        for j in range(i+1, len(seq)):
            if seq[j] < seq[min_index]:
                min_index = j
        # Swap the smallest item found with that at index i
        seq[i], seq[min_index] = seq[min_index], seq[i]
```

Heap Sort

```
def heap_sort(seq):
    """Sort the sequence in place using heap sort"""
    # Create a max heap from the sequence
    heap = Heap(seq, operator.gt)
    # As you remove the largest item remaining in the heap,
    # place it in the free up space in the sequence.
    for i in range(len(seq)-1, -1, -1):
        largest = heap.remove_min()
        seq[i] = largest
```