Sorting

Insertion sort
Bubble sort
Divide and conquer sorting

Last time: introduction to sorting

- Big “O” notation: method for calculating which algorithms are fastest
- Sorting problem: organizing data in order
- Simple sorts
  - Selection sort: pick the item that goes next from the remainder of the set
  - Somewhat slow — O(n^2) — but easy to understand and program
- Today: more sorting algorithms
  - Bubble sort
  - Insertion sort
  - (If time): divide and conquer sorting
Bubble sort

- Basic idea: run through the array, exchanging values that are out of order
  - May have to make multiple “passes” through the array
  - Eventually, we will have exchanged all out-of-order values, and the list will be sorted
  - Easy to code!
- Unlike selection sort, bubble sort doesn’t have an outer loop that runs once for each item in the array
- Bubble sort works well with either linked lists or arrays

Bubble sort: code

```c
do {
    done = 1;
    for (j = 0; j < nItems-1; j++) {
        if (arr[j] > arr[j+1]) {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
            done = 0;
        }
    }
} while (!done);
```

- Code is very short and simple
- Will it ever finish?
  - Keeps going as long as at least one swap was made
  - How do we know it’ll eventually end?
- Guaranteed finish: finite number of swaps possible
  - Large elements “bubble” to the end of the array
  - Outer loop runs at most nItems-1 times
- Generally not a good sort
  - OK if a few items slightly out of order
**Bubble sort: running time**

- How long does bubble sort take to run?
  - Outer loop can execute a maximum of nItems-1 times
  - Inner loop can execute a maximum of nItems-1 times
- **Answer:** $O(n^2)$
  - Best case time could be much faster
  - Array nearly sorted would run very quickly with bubble sort
- Beginning to see a pattern: sorts seem to take time proportional to $n^2$
  - Is there any way to do better?
  - Let’s check out insertion sort

---

**What is insertion sort?**

- Selection sort: find the next element in the sorted list from the rest of the unsorted list
- Insertion sort: place the next element in the unsorted list where it “should” go in the sorted list
  - Other elements may need to shift to make room
  - May be best to do this with a linked list…
Pseudocode for insertion sort

```
while (unsorted list not empty) {
  pop item off unsorted list
  for (cur = sorted.first;
       cur is not last && cur.value < item.value;
       cur = cur.next) {
    ;
  }
  if (cur.value < item.value) {
    insert item after cur // last on list
  } else {
    insert item before cur
  }
}
```

```
for (j = 1; j < nItems; j++) {
  curr = arr[j];
  for (k = j-1; (k>=0) && (arr[k]>curr); k--) {
    arr[k+1] = arr[k];
  }
  arr[k+1] = curr;
}
```

How fast is insertion sort?

- Insertion sort has two nested loops
  - Outer loop runs once for each element in the original unsorted loop
  - Inner loop runs through sorted list to find the right insertion point
    - Average time: 1/2 of list length
- The timing is similar to selection sort: $O(n^2)$
- Can we improve this time?
  - Inner loop has to find element just past the one we want to insert
  - We know of a way to this in $O(\log n)$ time: binary search!
    - Requires arrays, but insertion sort works best on linked lists…
    - Maybe there’s hope for faster sorting
How can we improve sorting speed?

- Why is sorting so slow?
  - We have to iterate over the entire unsorted array
  - Operations for each item we’re sorting take time O(n)
- Attack the problem by breaking it into smaller pieces: divide and conquer
  - Break the array in half
  - Sort each half
  - Merge the two halves together
- Is this any faster?
  - May be faster because merging is easier than sorting…

Sorting by merging: mergesort

- Break the data into two (equal) halves
- Recursively sort each half the same way
- How many levels of splits do we have?
  - Each level takes time O(n)
  - We have O(log n) levels!
- This means that total required time is O(n log n)!
  - It works because we can sort smaller pieces
  - Merging is very fast: O(n)