Sorting

- Arrange array elements to be in ascending (or descending) order by key

- Sorting algorithms differ by:
    - **ease of implementation**
    - **efficiency (in best, worst, and average cases)**
    - **adaptability for memory limitations** (i.e., internal vs. external sorting)

Bubblesort

- Imagine array turned vertically (A[0] at “top”, A[n-1] at “bottom”)
- Make repeated passes over the array, from bottom to top
- If 2 adjacent elements out of order, then reverse them (i.e., “lighter” one moves up)
- On 0^{th} pass, “lightest” element rises all the way to the top
- On i^{th} pass, no element will have to rise above position i

```
for (i = 0; i < n - 1; i++)               // n-2 times
  for (j = n - 1; j >= i + 1; j--)        // n-1 times, n-2 times, …, 1 time
    if (A[ j ].key < A[ j – 1].key)          // constant time
      swap(A[ j ], A[ j – 1 ]);             //  constant time
```

When	What j loop will do	# times if stmt will be executed
i = 0	j = (n-1)..1	n-1
i = 1	j = (n-1)..2	n-2
i = 2	j = (n-1)..3	n-3
…		
i = n-2	j = (n-1)..(n-1)	n – (n – 1) = 1

So $T(n) = (n-1) + (n-2) + (n-3) + \ldots + (n-(n-1)) = ((n-1) * n)) - \sum_{i=1}^{n-1} i$ = $\mathbf{O(n^2)}$

…What if array is already in sorted order???

Selection Sort

- On the i^{th} pass, select the lowest key among A[i], …, A[n-1], and swap it with A[i]

```
for (i = 0; i < n-1; i++) {            // n-2 times
  min = A[ i ].key; minPos = i;
  for (j = i + 1; j < n; j++)         // n-1 times, n-2 times, …, 1 time
    if (A[ j ].key < min) {
        min = A [ j ].key;
        minPos = j;
      }
  if (i != minPos) swap(A[ i ], A[ minPos ]);
  }
```

This algorithm is also $\mathbf{O(n^2)}$ ***…Again, what if array is already in sorted order???***

Insertion Sort

- On the i^{th} pass, insert A[i] into its rightful place among A[0], A[1], …, A[i-1]
- After that A[0], A[1], …, A[i] are in sorted order

```
for (i = 1; i < n; i++)
  insert A[ i ] into the elements before A [ i ] (which are already in sorted order)
```

This algorithm is also **O(n^2)**   ***…What if array is already in sorted order???***

Characteristics of Quadratic Time Sorting Algorithms

- **Simple to implement**
- **Not very efficient, but O.K. if n not too big**
- **Require all elements of array to be in memory at one time**