COMP 2611, DATA STRUCTURES
LECTURE 20

SORTING

- **Review of Selection Sort, Bubble Sort, and Insertion Sort**
- **Performance Analysis**

Sorting: Selection Sort

Think of the array as having a sorted section and an unsorted section.

Find the smallest value in the unsorted section.

0	1	2	3	4	5	6	7	8	9
56	34	38	71	28	10	91	18	22	75

sorted unsorted

0	1	2	3	4	5	6	7	8	9
10	34	38	71	28	56	91	18	22	75

sorted unsorted

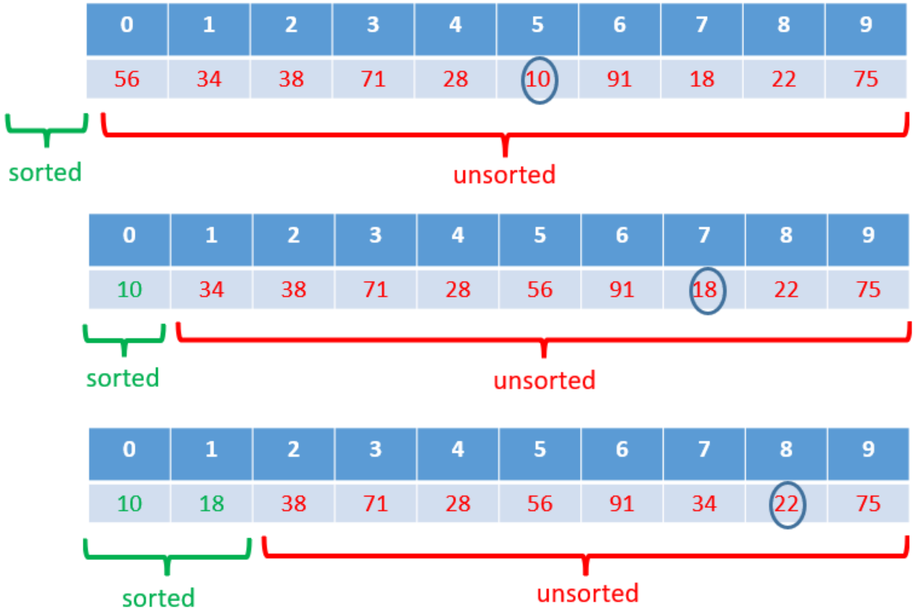
As the algorithm progresses, the unsorted section gets smaller and the sorted section gets larger.

0	1	2	3	4	5	6	7	8	9
10	18	38	71	28	56	91	34	22	75

sorted unsorted

Selection Sort

```
void selectionSort (int A [], int numElements) {
        int min, temp;

        for (int i=0; i<numElements-1; i++) {
                min = i;
                for (int j=i+1; j<numElements; j++) {
                        if (A[j] < A[min])
                                min = j;
                }
                temp = A[i];
                A[i] = A[min];
                A[min] = temp;
        }
}
```



0	1	2	3	4	5	6	7	8	9
56	34	38	71	28	10	91	18	22	75

0	1	2	3	4	5	6	7	8	9
10	34	38	71	28	56	91	18	22	75

0	1	2	3	4	5	6	7	8	9
10	18	38	71	28	56	91	34	22	75

Performance: O (n^2)

Sorting Algorithms

- Selection sort
- Bubble sort
- Insertion sort
- Heap sort
- Merge sort
- Quick sort

Inserting an Element “In Place”

0	1	2	3	4	5	6	7	8	9	10
32	37	50	56	63	70	74	86			

A (8 elements)

Assume that *A* is sorted.

- ➢ Suppose we want to insert 52.
- ➢ We need to “pull down” all the elements from location 3:

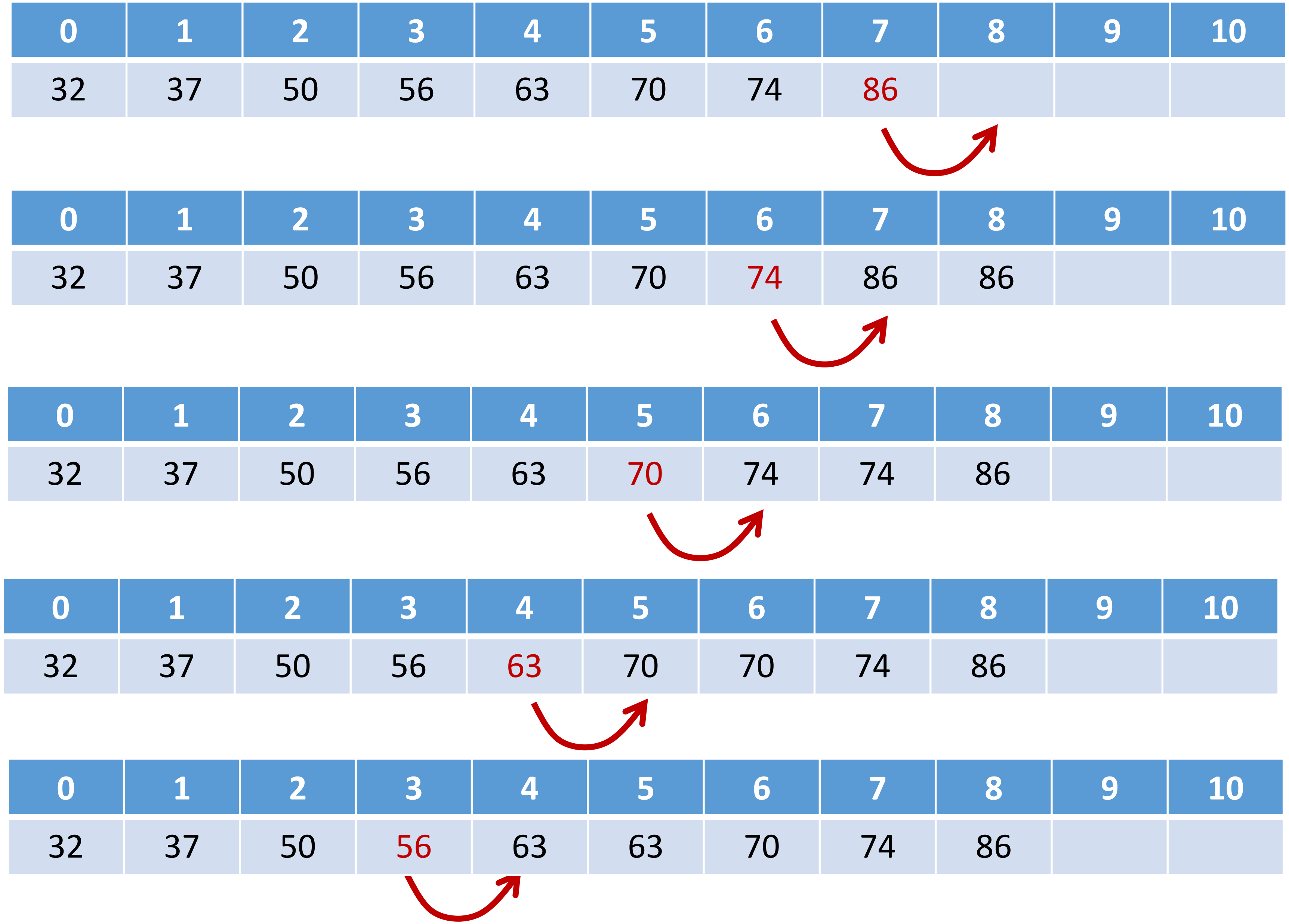
0	1	2	3	4	5	6	7	8	9	10
32	37	50		56	63	70	74	86		

A

52

- ➢ 52 is then inserted in the “space” created at location 3.

How to “Pull Down” Elements From a Location



A (8 elements)

0	1	2	3	4	5	6	7	8	9	10
32	37	50	56	63	70	74	86			

0	1	2	3	4	5	6	7	8	9	10
32	37	50	56	63	70	74	86	86		

0	1	2	3	4	5	6	7	8	9	10
32	37	50	56	63	70	74	74	86		

0	1	2	3	4	5	6	7	8	9	10
32	37	50	56	63	70	70	74	86		

0	1	2	3	4	5	6	7	8	9	10
32	37	50	56	63	63	70	74	86		

Code to “Pull Down” An Element

- Suppose *A* has *n* elements before inserting the new element [0..*n*-1].

- After the new element is inserted, it has *n*+1 elements [0..*n*].

```
void insertInPlace (int A [], int n, int newElement) {

  int k = n-1;

  while (k >= 0 && newElement < A[k]) {
    A[k+1] = A[k];
    k--;
  }

  A[k+1] = newElement;
}
```

Calling *insertInPlace* with a Set of Random Data

- What happens if *insertInPlace* is called repeatedly with a set of random numbers read from a file?

- Compile and run `InsertInPlace.cpp` with the data file, `data.txt`.

- Even though the data is not sorted in `data.txt`, by calling *insertInPlace* as each value is read from the file, the array is automatically sorted!

Insertion Sort

Think of the array as having a sorted section and an unsorted section.

0	1	2	3	4	5	6	7	8	9
56	**34**	38	71	28	10	91	18	22	75

sorted

unsorted

- ➢ Take the first element from the unsorted section (A[1]) and insert it in sorted section (which has one element, 56).
- ➢ Call insertInPlace (A, 1, A[1]):

0	1	2	3	4	5	6	7	8	9
34	56	**38**	71	28	10	91	18	22	75

sorted

unsorted

0	1	2	3	4	5	6	7	8	9
34	56	**38**	71	28	10	91	18	22	75

- ➢ Call insertInPlace (A, 2, A[2]):

0	1	2	3	4	5	6	7	8	9
34	38	56	**71**	28	10	91	18	22	75

- ➢ Call insertInPlace (A, 3, A[3]):

0	1	2	3	4	5	6	7	8	9
34	38	56	71	**28**	10	91	18	22	75

- ➢ Call insertInPlace (A, 4, A[4]):

0	1	2	3	4	5	6	7	8	9
28	34	38	56	71	**10**	91	18	22	75

- ➢ Call insertInPlace (A, 5, A[5]):

0	1	2	3	4	5	6	7	8	9
10	28	34	38	56	71	**91**	18	22	75

- ➢ Call insertInPlace (A, 6, A[6]):

0	1	2	3	4	5	6	7	8	9
10	28	34	38	56	71	91	**18**	22	75

- ➢ Call insertInPlace (A, 7, A[7]):

0	1	2	3	4	5	6	7	8	9
10	18	28	34	38	56	71	91	**22**	75

- Call insertInPlace (A, 8, A[8]):

0	1	2	3	4	5	6	7	8	9
10	18	22	28	34	38	56	71	91	**75**

- Call insertInPlace (A, 9, A[9]):

0	1	2	3	4	5	6	7	8	9
10	18	22	28	34	38	56	71	75	91

The array is now completely sorted since there are no elements in the unsorted section.

Insertion Sort Function

- Repeatedly add elements from the unsorted section to the sorted section using the *insertInPlace* function.

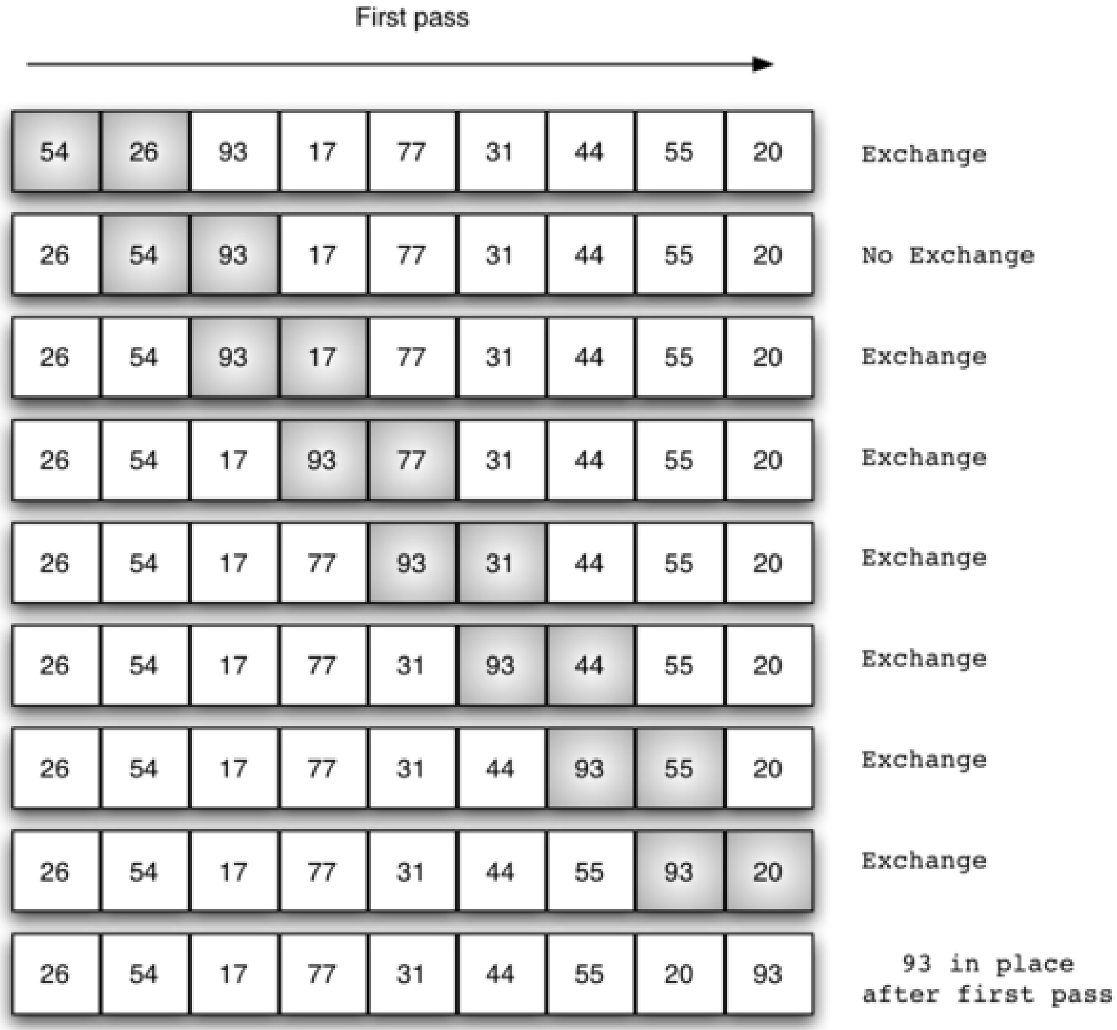
```
void insertionSort (int A[], int numElements) {

  int i;

  for (i=1; i<numElements; i++)
     insertInPlace (A, i, A[i]);

}
```

This function uses a *while* loop which could iterate up to *numElements*.

Performance: O (n^2)

BubbleSort Algorithm

The sorted section is at the right of the array and the unsorted section is at the left.



First pass

54	26	93	17	77	31	44	55	20	Exchange
26	54	93	17	77	31	44	55	20	No Exchange
26	54	93	17	77	31	44	55	20	Exchange
26	54	17	93	77	31	44	55	20	Exchange
26	54	17	77	93	31	44	55	20	Exchange
26	54	17	77	31	93	44	55	20	Exchange
26	54	17	77	31	44	93	55	20	Exchange
26	54	17	77	31	44	55	93	20	Exchange
26	54	17	77	31	44	55	20	93	93 in place after first pass

In the second pass, 77 will make its way to location 7. This continues for 8 passes or until there are no swaps.

BubbleSort Algorithm

- Repeatedly “bubble down” biggest element towards the right side of the array.

```
void bubbleSort (int A[], int lengthA) {

  int i, j, temp;

  for (i=0; i<lengthA-1; i++) {
    for (j=0; j<lengthA-i-1; j++) {
      if (A[j] > A[j+1]) {
        temp = A[j];
        A[j] = A[j+1];
        A[j+1] = temp;
      }
    }
  }
}
```

Performance: O (n^2)