Programming Logic & Design

Chapter 8 – Arrays – Part 3

Queen's College CSD

1133 – CPCM -2023S

What is Sorting?

Sorting: an operation that segregates items into groups according to specified criterion.

$A = \{ 3\ 1\ 6\ 2\ 1\ 3\ 4\ 5\ 9\ 0 \}$

$A = \{ 0\ 1\ 1\ 2\ 3\ 3\ 4\ 5\ 6\ 9 \}$

Why Sort and Examples

Consider:

Sorting Books in Library (Dewey system)

Sorting Individuals by Height (Feet and Inches)

Sorting Movies in Blockbuster (Alphabetical)

Sorting Numbers (Sequential)

Types of Sorting Algorithms

There are many, many different types of sorting algorithms, but the primary ones are:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Radix Sort
- Swap Sort

Sorting Classification

In memory sorting			External sorting
Comparison sorting $\Omega(N \log N)$		Specialized Sorting	
$O(N^2)$	$O(N \log N)$	$O(N)$	# of tape accesses
• Bubble Sort • Selection Sort • Insertion Sort • Shell Sort	• Merge Sort • Quick Sort • Heap Sort	• Bucket Sort • Radix Sort	• Simple External Merge Sort • Variations

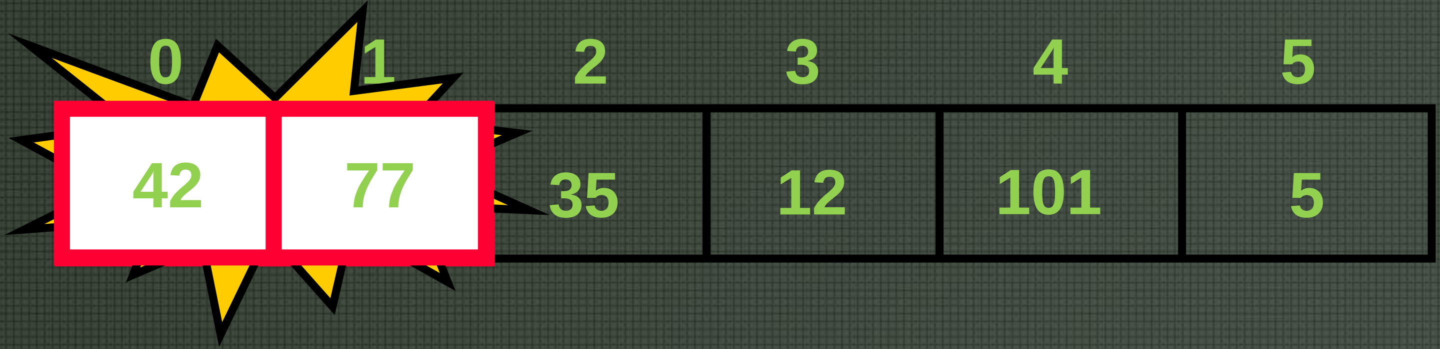
Bubble sort

- **bubble sort**: orders a list of values by repetitively comparing neighboring elements and swapping their positions if necessary

- more specifically:
  - scan the list, exchanging adjacent elements if they are not in relative order; this bubbles the highest value to the top
  - scan the list again, bubbling up the second highest value
  - repeat until all elements have been placed in their proper order

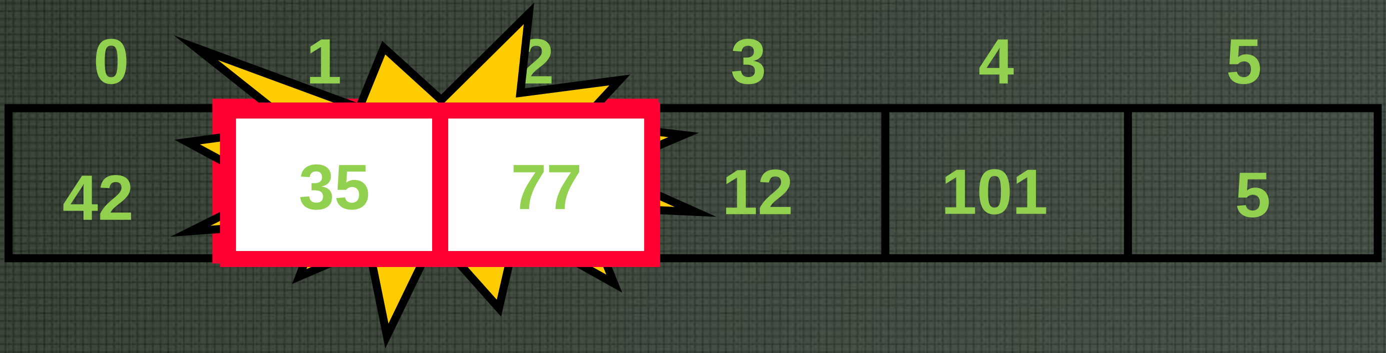
"Bubbling" largest element

Traverse a collection of elements
- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping



"Bubbling" largest element

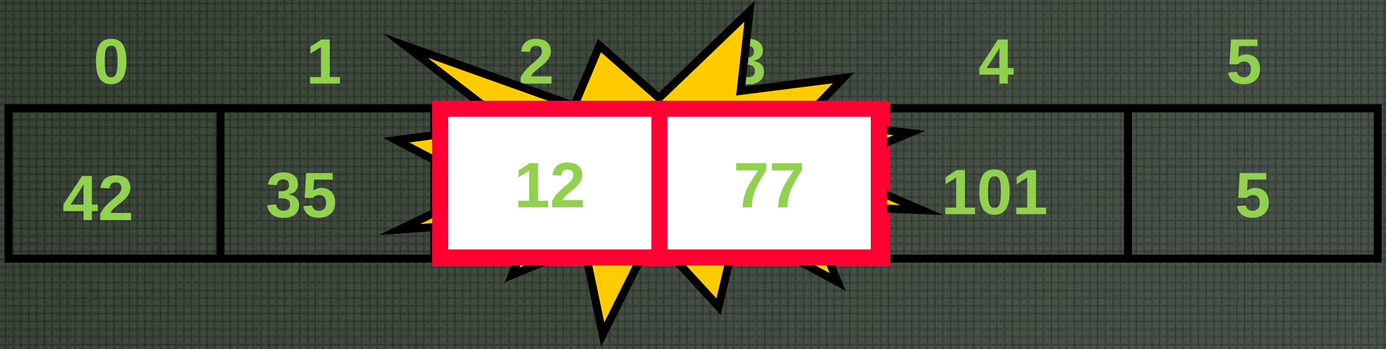
Traverse a collection of elements
- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping



"Bubbling" largest element

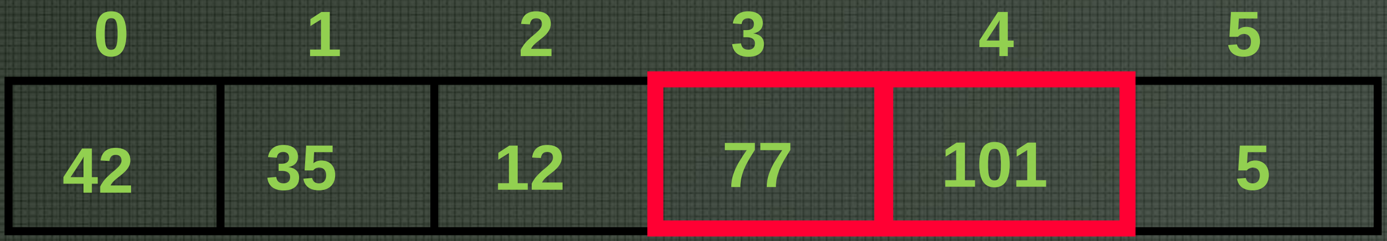
Traverse a collection of elements

- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping



"Bubbling" largest element

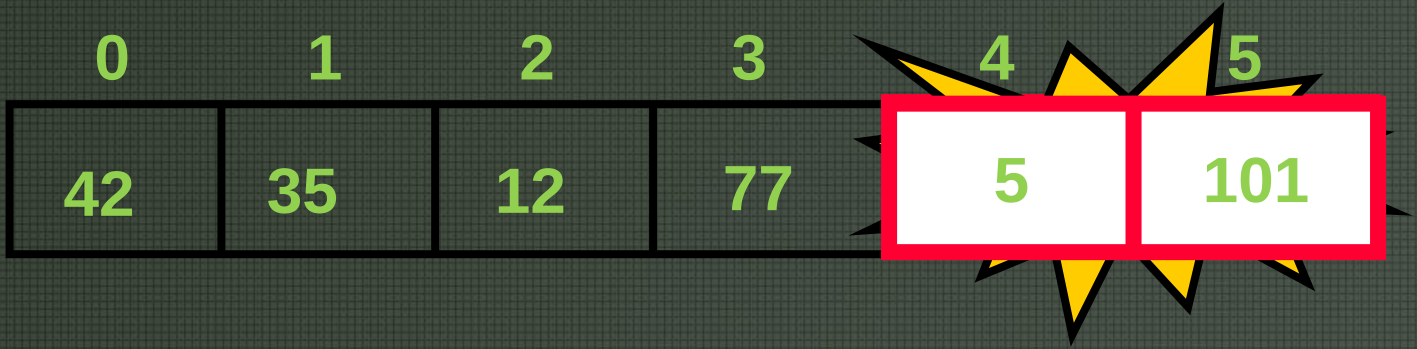
Traverse a collection of elements
- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping



No need to swap

"Bubbling" largest element

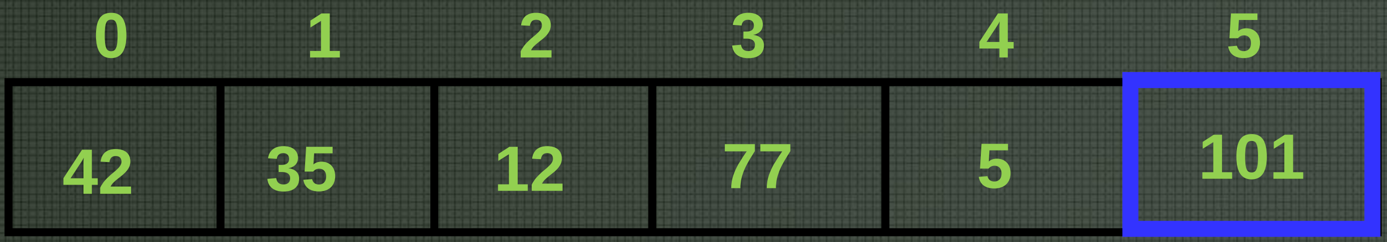
Traverse a collection of elements
- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping



"Bubbling" largest element

Traverse a collection of elements

- Move from the front to the end
- "Bubble" the largest value to the end using pair-wise comparisons and swapping



Largest value correctly placed

Bubble sort code

```
public static void bubbleSort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 1; j < a.length - i; j++) {
            // swap adjacent out-of-order elements
            if (a[j-1] > a[j]) {
                swap(a, j-1, j);
            }
        }
    }
}
```

Selection sort

- **selection sort**: orders a list of values by repetitively putting a particular value into its final position

- more specifically:
  - find the smallest value in the list
  - switch it with the value in the first position
  - find the next smallest value in the list
  - switch it with the value in the second position
  - repeat until all values are in their proper places

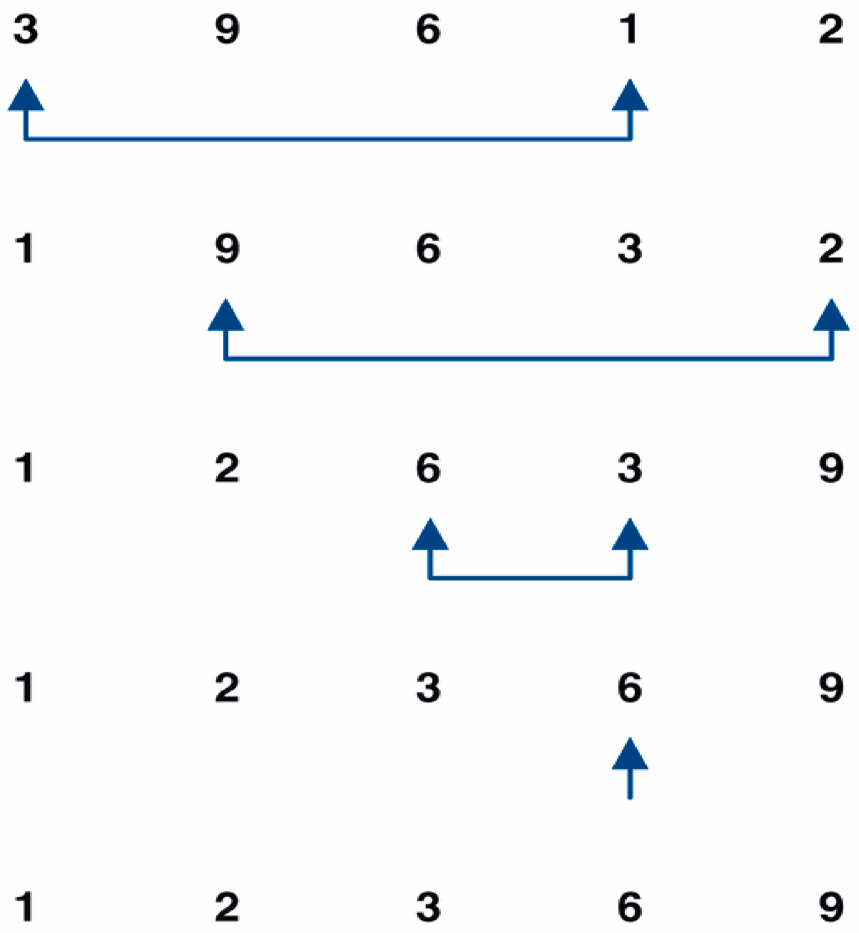
Selection sort example

Scan right starting with 3.
1 is the smallest. Exchange 1 and 3.

Scan right starting with 9.
2 is the smallest. Exchange 9 and 2.

Scan right starting with 6.
3 is the smallest. Exchange 6 and 3.

Scan right starting with 6.
6 is the smallest. Exchange 6 and 6.



Selection sort example 2

Index	0	1	2	3	4	5	6	7
Value	27	63	1	72	64	58	14	9
1st pass	**1**	63	27	72	64	58	14	9
2nd pass	1	**9**	27	72	64	58	14	63
3rd pass	1	9	**14**	72	64	58	27	63
…								

Selection sort code

```
public static void selectionSort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        // find index of smallest element
        int minIndex = i;
        for (int j = i + 1; j < a.length; j++) {
            if (a[j] < a[minIndex]) {
                minIndex = j;
            }
        }

        // swap smallest element with a[i]
        swap(a, i, minIndex);
    }
}
```

Insertion sort

- **insertion sort**: orders a list of values by repetitively inserting a particular value into a sorted subset of the list

- more specifically:
  - consider the first item to be a sorted sublist of length 1
  - insert the second item into the sorted sublist, shifting the first item if needed
  - insert the third item into the sorted sublist, shifting the other items as needed
  - repeat until all values have been inserted into their proper positions

Insertion sort

Simple sorting algorithm.

- n-1 passes over the array
- At the end of pass *i*, the elements that occupied A[0]…A[*i*] originally are still in those spots and in sorted order.

2	15	8	1	17	10	12	5
0	1	2	3	4	5	6	7

after pass 2

2	8	15	1	17	10	12	5
0	1	2	3	4	5	6	7

after pass 3

1	2	8	15	17	10	12	5
0	1	2	3	4	5	6	7

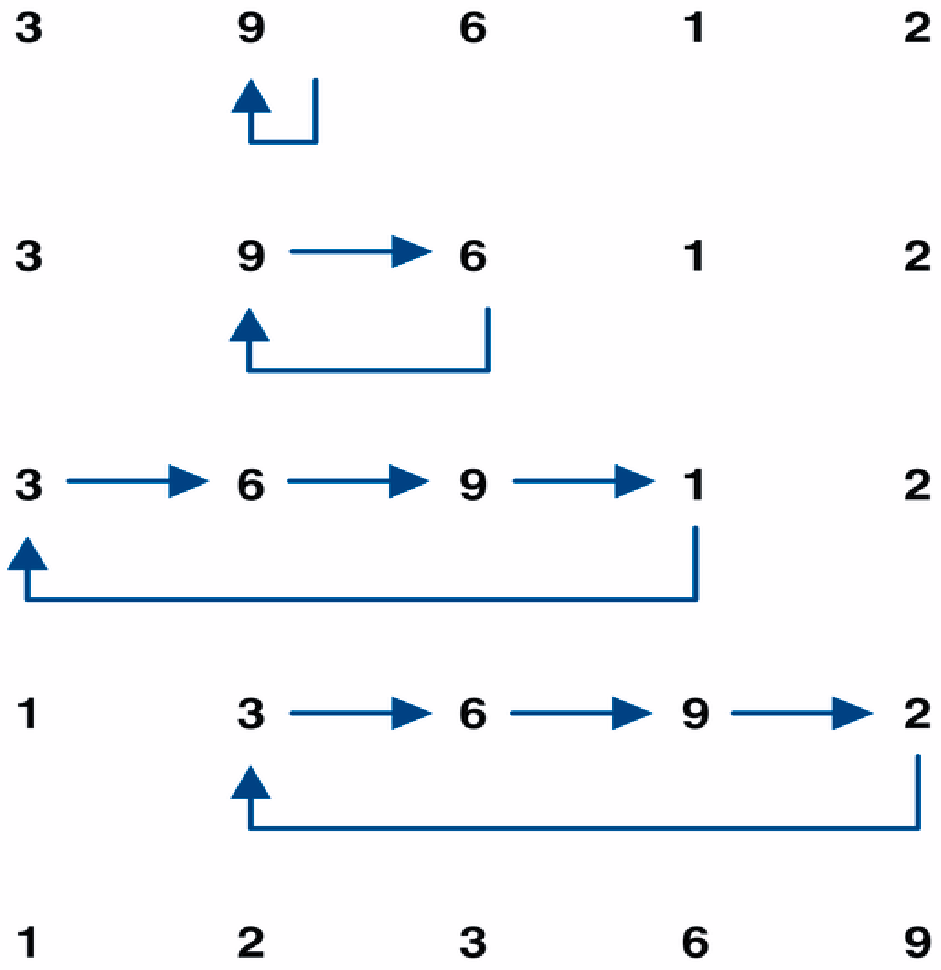
Insertion sort example

3 is sorted.
Shift nothing. Insert 9.

3 and 9 are sorted.
Shift 9 to the right. Insert 6.

3, 6, and 9 are sorted.
Shift 9, 6, and 3 to the right. Insert 1.

1, 3, 6, and 9 are sorted.
Shift 9, 6, and 3 to the right. Insert 2.



Insertion sort code

```
public static void insertionSort(int[] a) {
    for (int i = 1; i < a.length; i++) {
        int temp = a[i];

        // slide elements down to make room for a[i]
        int j = i;
        while (j > 0 && a[j - 1] > temp) {
            a[j] = a[j - 1];
            j--;
        }

        a[j] = temp;
    }
}
```