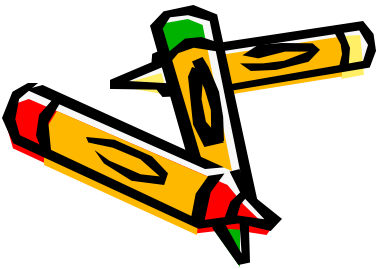
Brute Force Approach

What is Brute Force?

- The first algorithm design technique we shall explore
- A straightforward approach to solving problem, usually based on problem statement and definitions of the concepts involved
- "Force" comes from using computer power not intellectual power
- In short, "brute force" means "Just do it!"



Brute Force

A straightforward approach, usually based directly on the problem's statement and definitions of the concepts involved

Examples:

1. Computing a^n ($a > 0$, n a nonnegative integer)

2. Computing $n!$

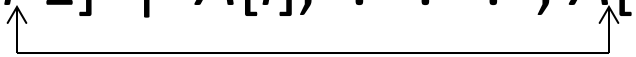
3. Multiplying two matrices

4. Searching for a key of a given value in a list

Brute-Force Sorting Algorithm

Selection Sort Scan the array to find its smallest element and swap it with the first element. Then, starting with the second element, scan the elements to the right of it to find the smallest among them and swap it with the second elements. Generally, on pass i ($0 \le i \le n-2$), find the smallest element in $A[i..n-1]$ and swap it with $A[i]$:

$A[0] \le \ldots \le A[i-1] \mid A[i], \ldots, A[min], \ldots, A[n-1]$



in their final positions

Example: 7 3 2 5

Analysis of Selection Sort

ALGORITHM *SelectionSort*$(A[0..n-1])$

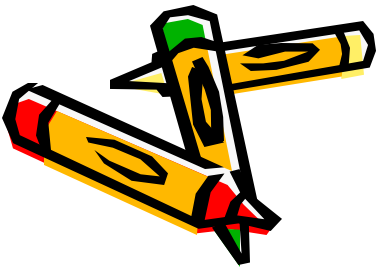
```
//Sorts a given array by selection sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in ascending order
for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
        if A[j] < A[min]   min ← j
    swap A[i] and A[min]
```

Time efficiency: $\Theta(n^2)$

Brute Force: Bubble Sort

- Compare adjacent elements and exchange them if out of order
- Essentially, it bubbles up the largest element to the last position

$$A_0, \ldots \ldots, A_j \overset{?}{<->} A_{j+1}, \ldots \ldots, A_{n-i-1} \mid A_{n-i} \leq \ldots \leq A_{n-1}$$



Brute Force: Bubble Sort (contd.)

ALGORITHM BubbleSort(A[0..n-1])

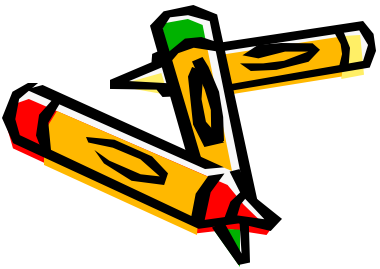
for i <- 0 to n-2 **do**

    for j <- 0 to n-2-i **do**

        if A[j+1] < A[j]

            swap A[j] and A[j+1]

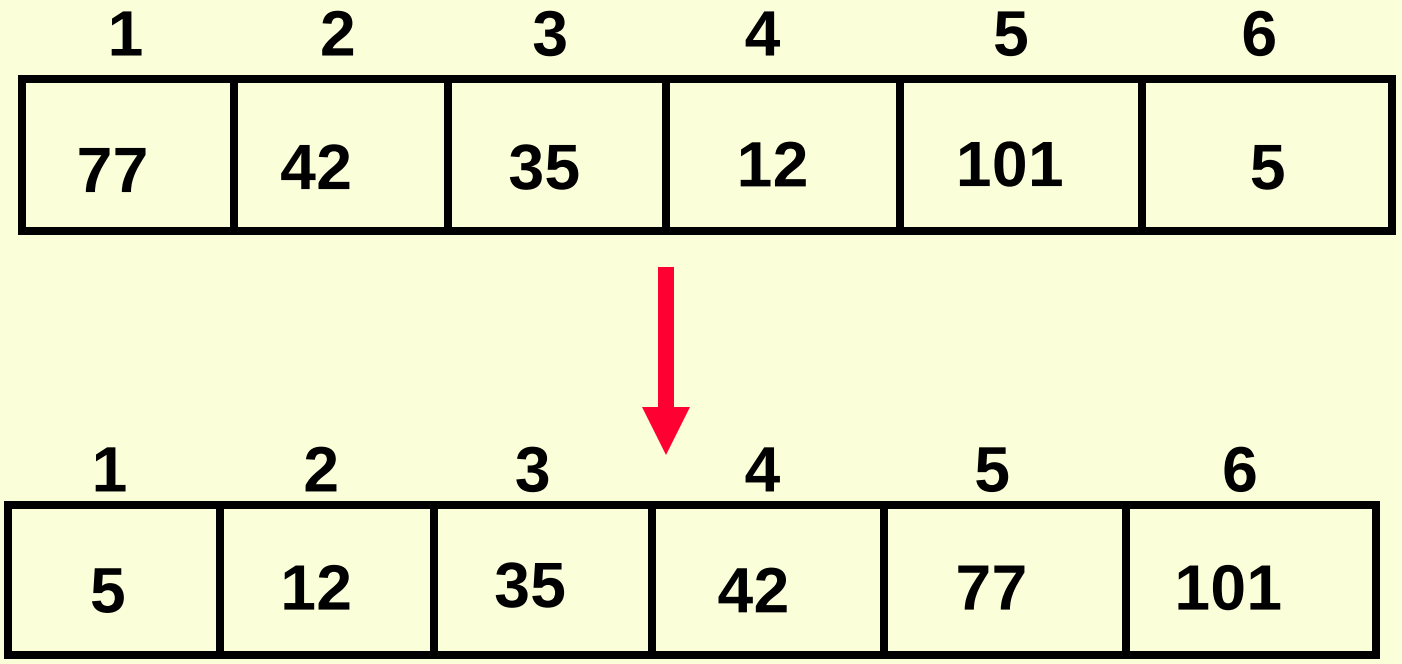
What about 89, 45, 68, 90, 29, 34, 17 ?



$C(n) \in \Theta(n^2)$

Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**



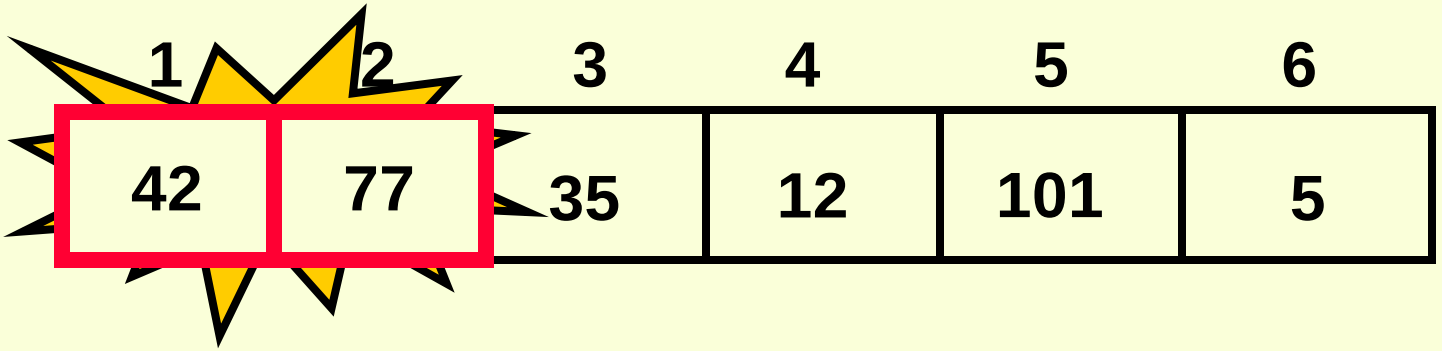
"Bubbling Up" the Largest Element

- 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

1	2	3	4	5	6
77	42	35	12	101	5

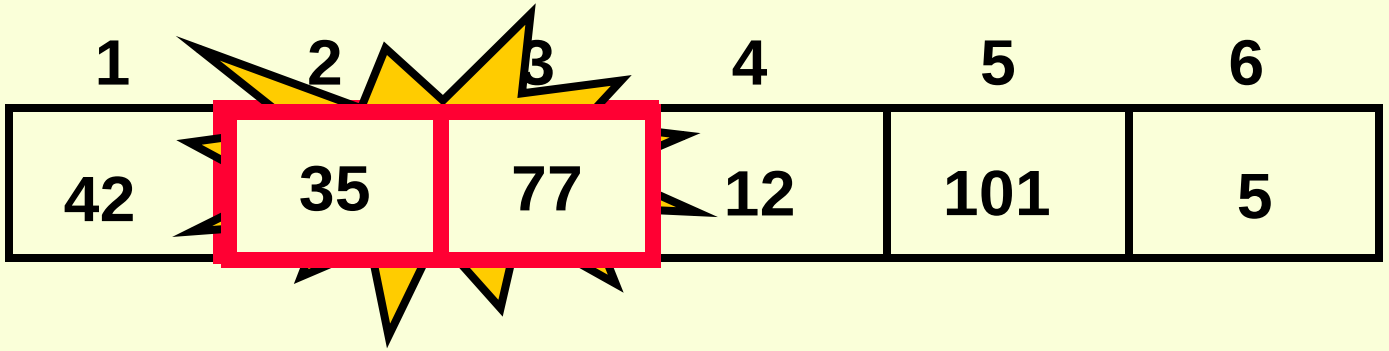
"Bubbling Up" the Largest Element

- 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



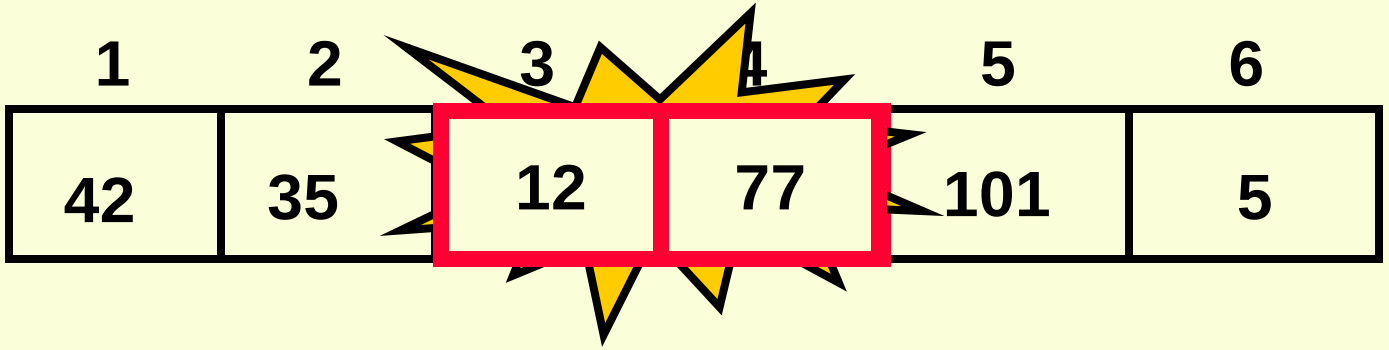
"Bubbling Up" the Largest Element

- 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



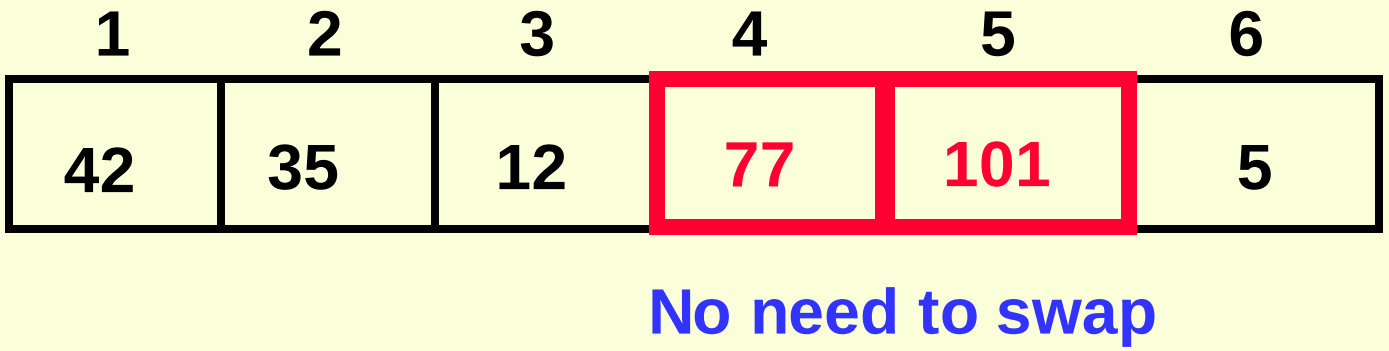
"Bubbling Up" the Largest Element

- 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



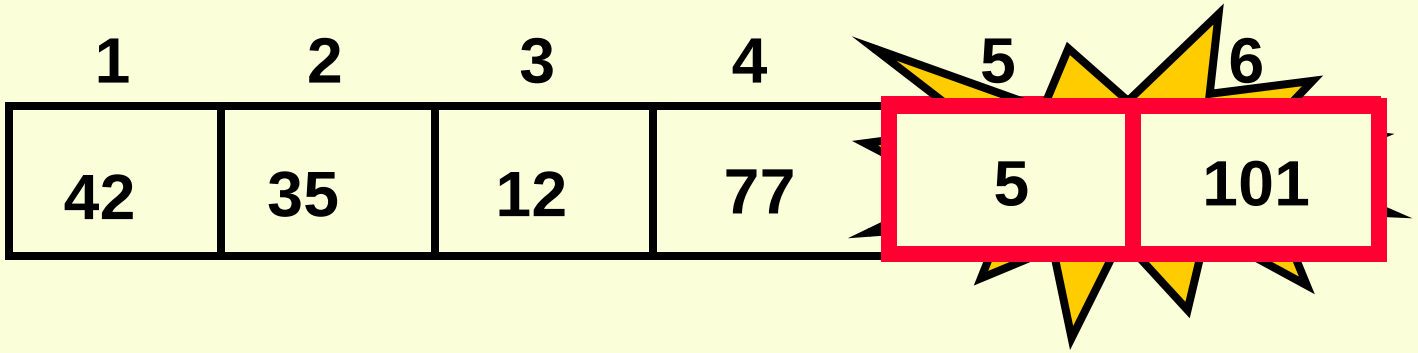
"Bubbling Up" the Largest Element

- 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



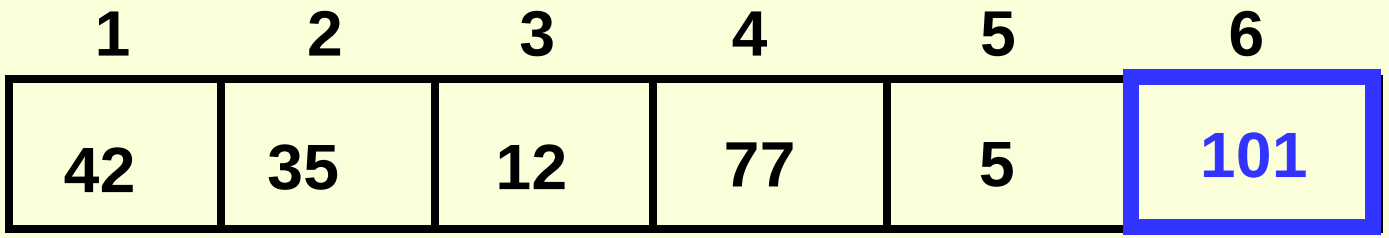
"Bubbling Up" the Largest Element

- 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 Up" the Largest Element

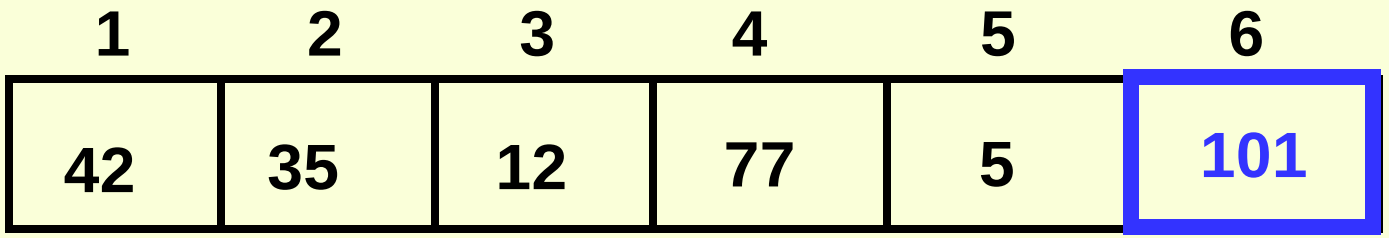
- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



Largest value correctly placed

Items of Interest

- Notice that only the largest value is correctly placed
- All other values are still out of order
- So we need to repeat this process



Largest value correctly placed

Repeat “Bubble Up” How Many Times?

- If we have N elements…
- And if each time we bubble an element, we place it in its correct location…
- Then we repeat the “bubble up” process N – 1 times.
- This guarantees we’ll correctly place all N elements.

“Bubbling” All the Elements