Basic Sorting Algorithms

Instructor: Krishna Venkatasubramanian

CSC 212

Announcements

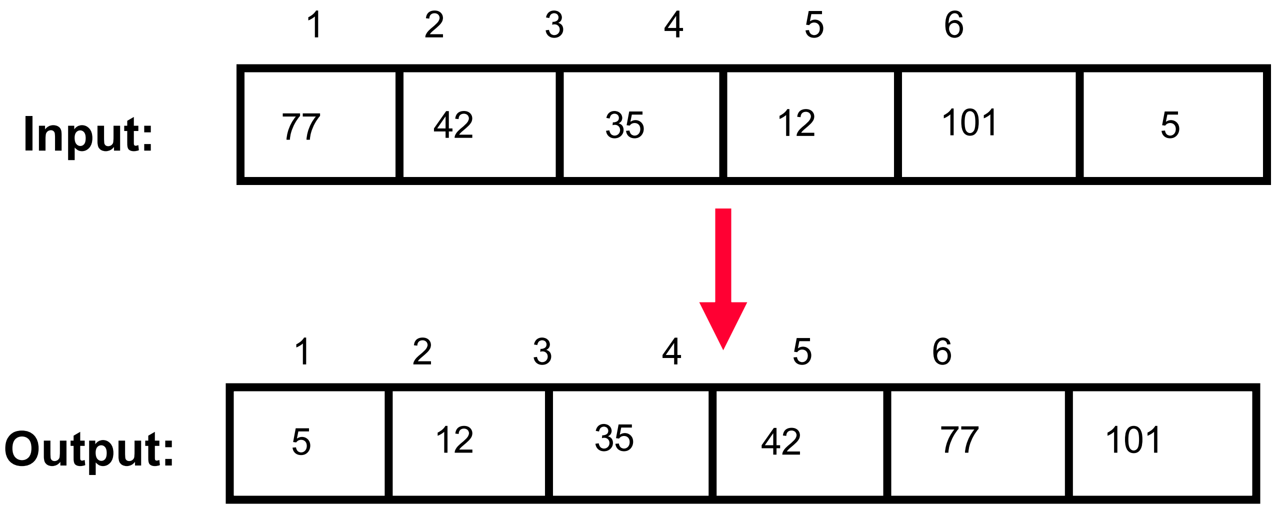
- Take the poll on office hours on Piazza
  - We need more people to respond
  - We will make a decision on changing office hours this Friday, so now is the time.

- Next Quiz (Oct 8)
  - Same format this quiz
  - 20 minutes

- Grades for Labs 1 and 2 and Quiz 0 on Gradescope

Sorting: Problem Definition

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



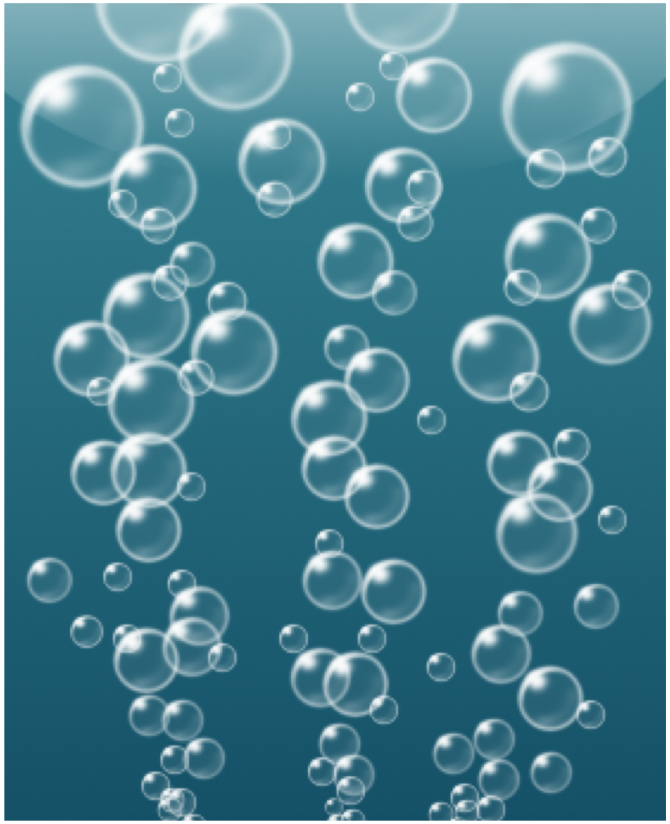
	1	2	3	4	5	6
Input:	77	42	35	12	101	5

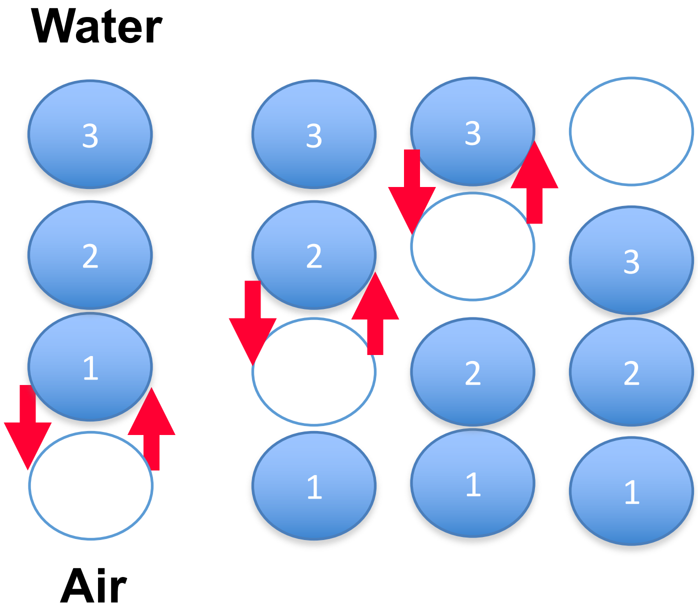
	1	2	3	4	5	6
Output:	5	12	35	42	77	101

Sorting Algorithms

- *Insertion Sort --- covered already*
- **Bubble Sort**
- Selection Sort
- Heap Sort
- Merge Sort
- Quick Sort
- …

Bubble Sort





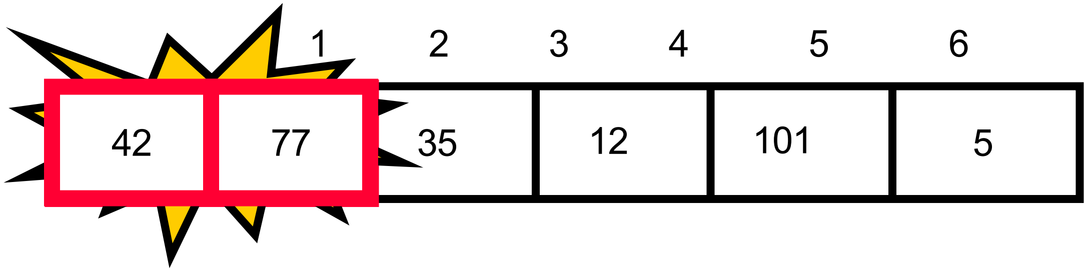
"Bubbling Up" the Largest Element

- **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

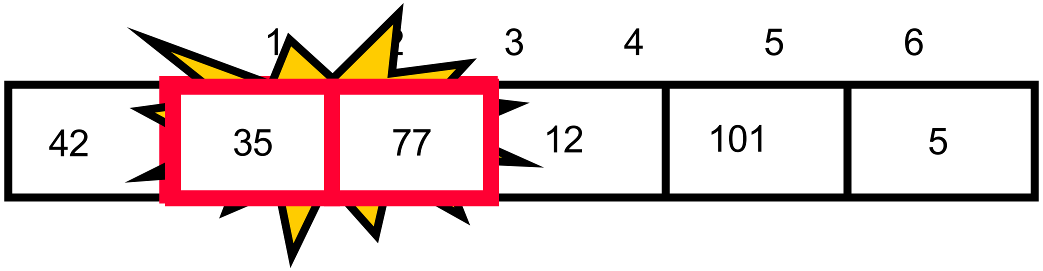
"Bubbling Up" the Largest Element

- **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**



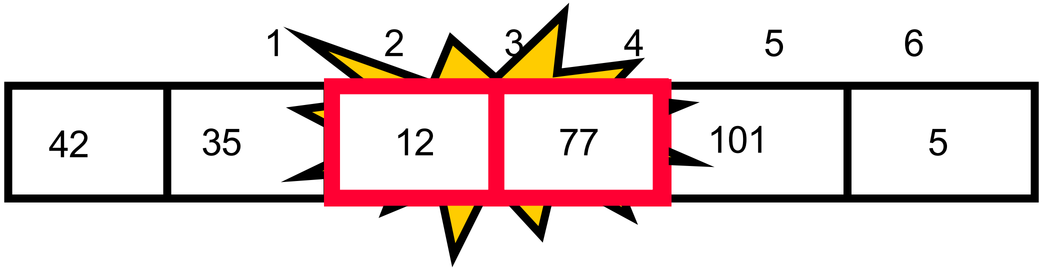
"Bubbling Up" the Largest Element

- **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

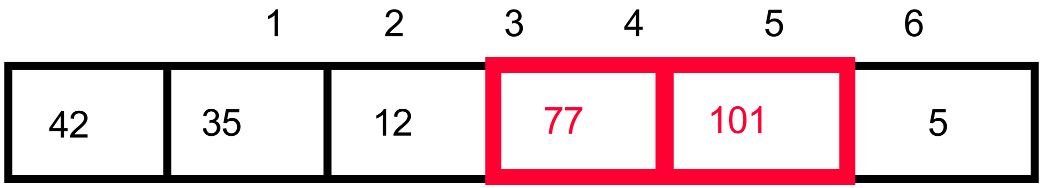
- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



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

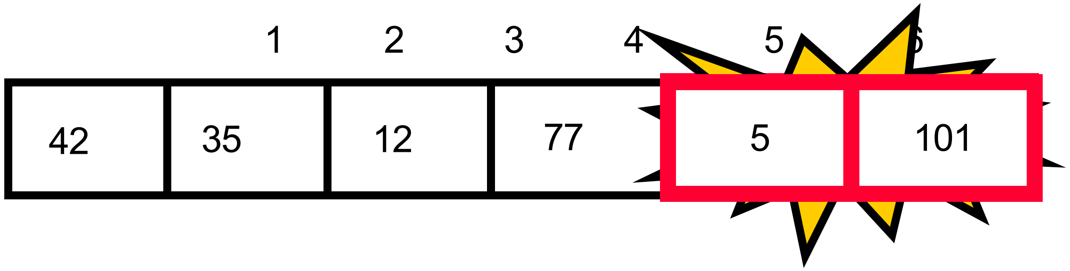


1	2	3	4	5	6
42	35	12	77	101	5

No need to swap

"Bubbling Up" the Largest Element

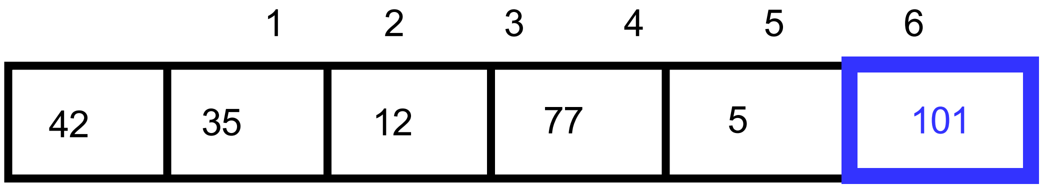
- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



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

"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
42	35	12	77	5	101

Largest value correctly placed

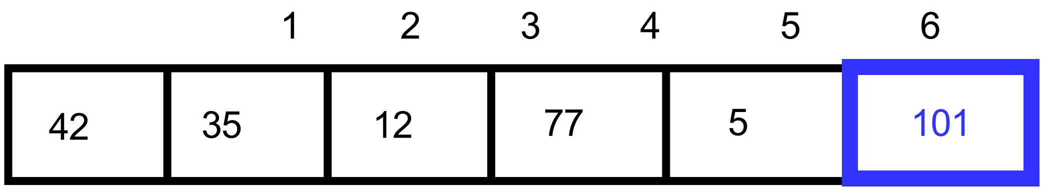
The “Bubble Up” Algorithm

```
Bubble-Sort-step1( A ):
     N = len (A)       # N = 7
   for k in range(1,N):
       if A[k-1] > A[k]:
              Swap( A, k-1, k)

Swap( A, x, y ):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp
```

Need More Iterations

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



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

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 will correctly place all N elements.**

The “Bubble Up” Algorithm: Repetition

```
Bubble-Sort( A ):
   N = len(A)
   for k in  range(1,N):
        if A[k-1] > A[k]:
               Swap( A, k-1, k )
```

The “Bubble Up” Algorithm

```
Bubble-Sort(A):
   N = len (A)
   for i in range(N):

       for k in range(1,N):
              if A[k-1] > A[k]:
                     Swap(A, k-1, k)
```

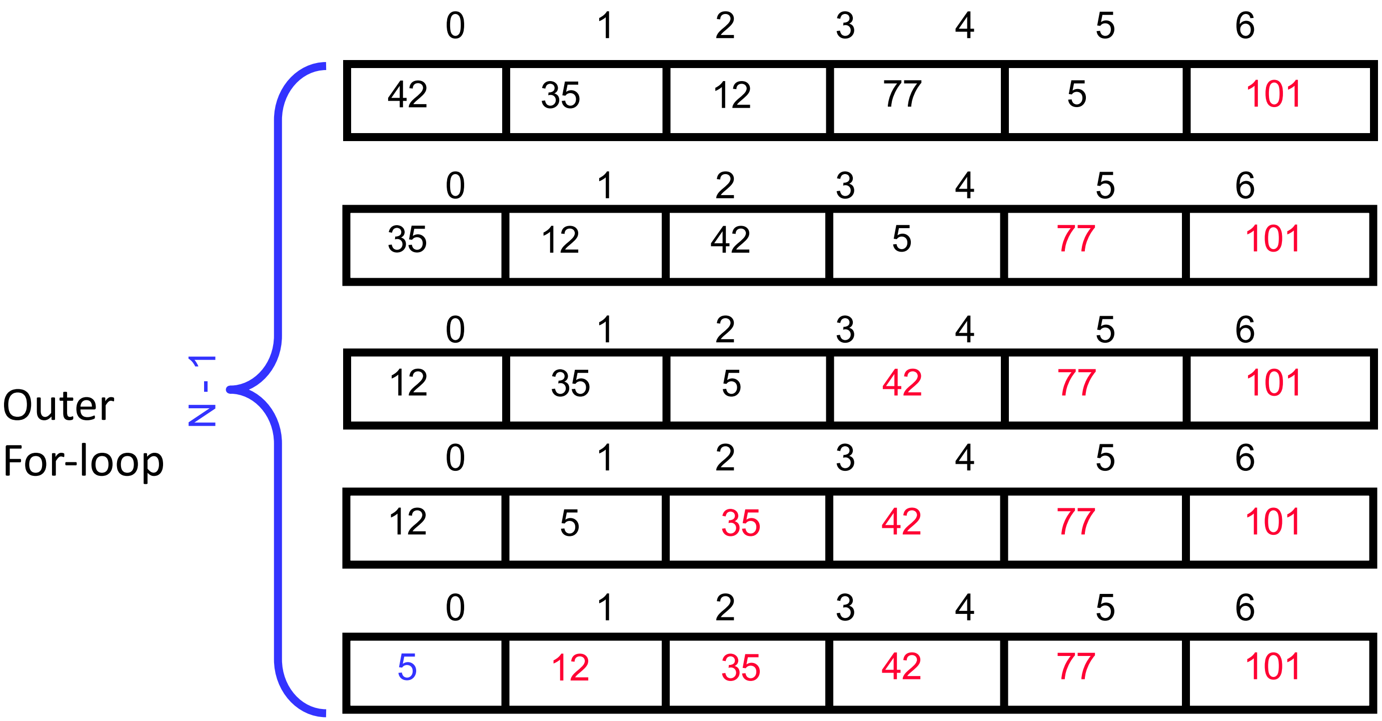
To do N-1 iterations

To bubble a value

Inner loop

Outer loop

“Bubbling” All the Elements



0	1	2	3	4	5	6
42	35	12	77	5	101	

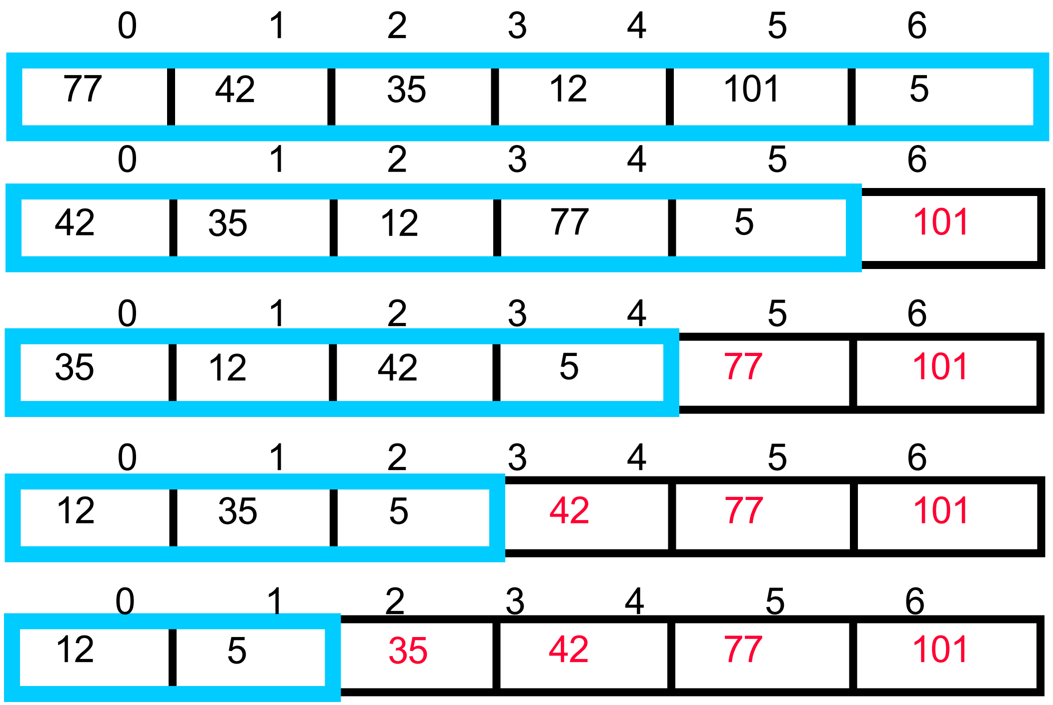
0	1	2	3	4	5	6
35	12	42	5	77	101	

0	1	2	3	4	5	6
12	35	5	42	77	101	

0	1	2	3	4	5	6
12	5	35	42	77	101	

0	1	2	3	4	5	6
5	12	35	42	77	101	

Reducing the Number of Comparisons



0	1	2	3	4	5	6
77	42	35	12	101	5	

0	1	2	3	4	5	6
42	35	12	77	5	101	

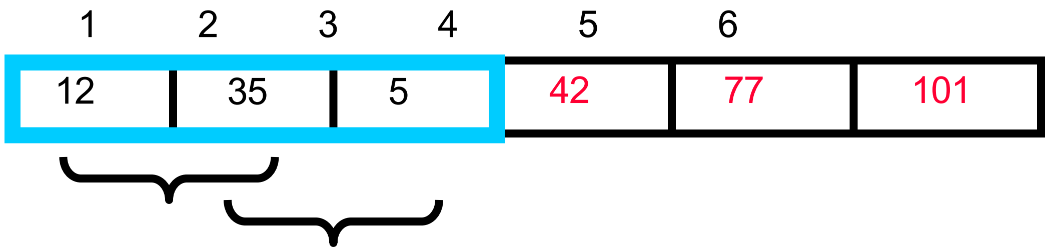
0	1	2	3	4	5	6
35	12	42	5	77	101	

0	1	2	3	4	5	6
12	35	5	42	77	101	

0	1	2	3	4	5	6
12	5	35	42	77	101	

Reducing the Number of Comparisons

- **Assume the array size → N**
- **On the ith iteration, we only need to do N-i comparisons.**
- **For example:**
  - **N = 6**
  - **i = 4 (4th iteration)**
  - **Thus, we have 2 comparisons to do**



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

The “Bubble Up” Algorithm: Optimized

```
Bubble-Sort(A):
   N = len (A)
   for i in range(N):


       for k in range(1,N):
              if A[k-1] > A[k]:
                     Swap(A, k-1, k)


       N = N-1
```

To do N-1, N-2… iterations

To bubble a value

Inner loop

Outer loop

Reduce the size of N, to reduce the number of iterations

Bubble Sort Time Complexity

- Best-Case Time Complexity
  - The scenario under which the algorithm will do the least amount of work (finish the fastest)

- Worst-Case Time Complexity
  - The scenario under which the algorithm will do the largest amount of work (finish the slowest)

Bubble Sort Time Complexity

- **Best-Case Time Complexity**
  - **Array is already sorted**
  - **(N-1) + (N-2) + ... + 1 = (N-1)* N / 2 comparisons**

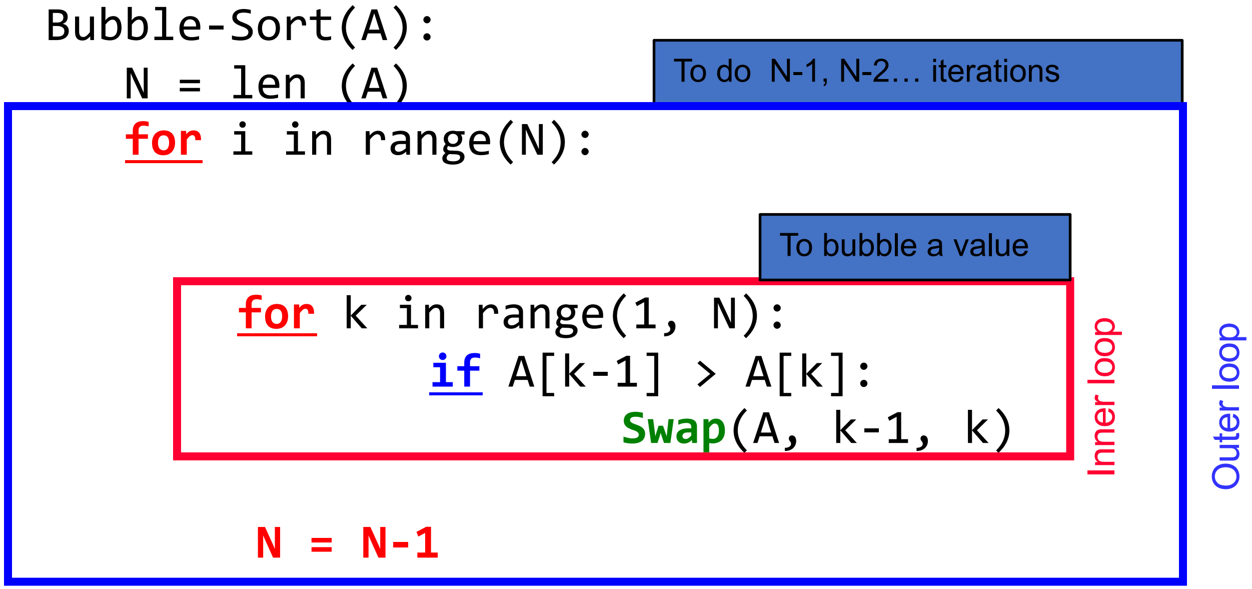
Called Quadratic Time
$O(N^2)$
Order-of-N-square

Called Quadratic Time
$O(N^2)$
Order-of-N-square

- **Worst-Case Time Complexity**
  - **Need N-1 iterations**
  - **(N-1) + (N-2) + ... + 1 = (N-1)* N / 2**

Loop Invariant

- **It is a condition or property that is guaranteed to be correct with each iteration in the loop**
- **Usually used to prove the correctness of the algorithm**



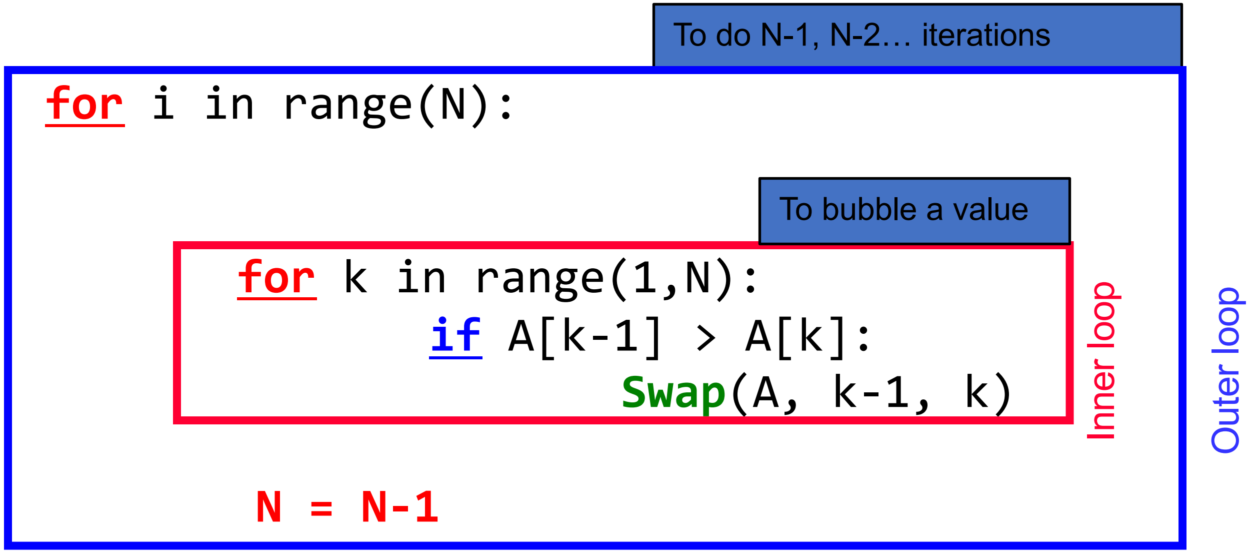
```
Bubble-Sort(A):
   N = len (A)
   for i in range(N):


       for k in range(1, N):
              if A[k-1] > A[k]:
                     Swap(A, k-1, k)


       N = N-1
```

Loop Invariant for Bubble Sort

- ***By the end of** iteration i ➔ the right-most i items (largest) are sorted and in place*

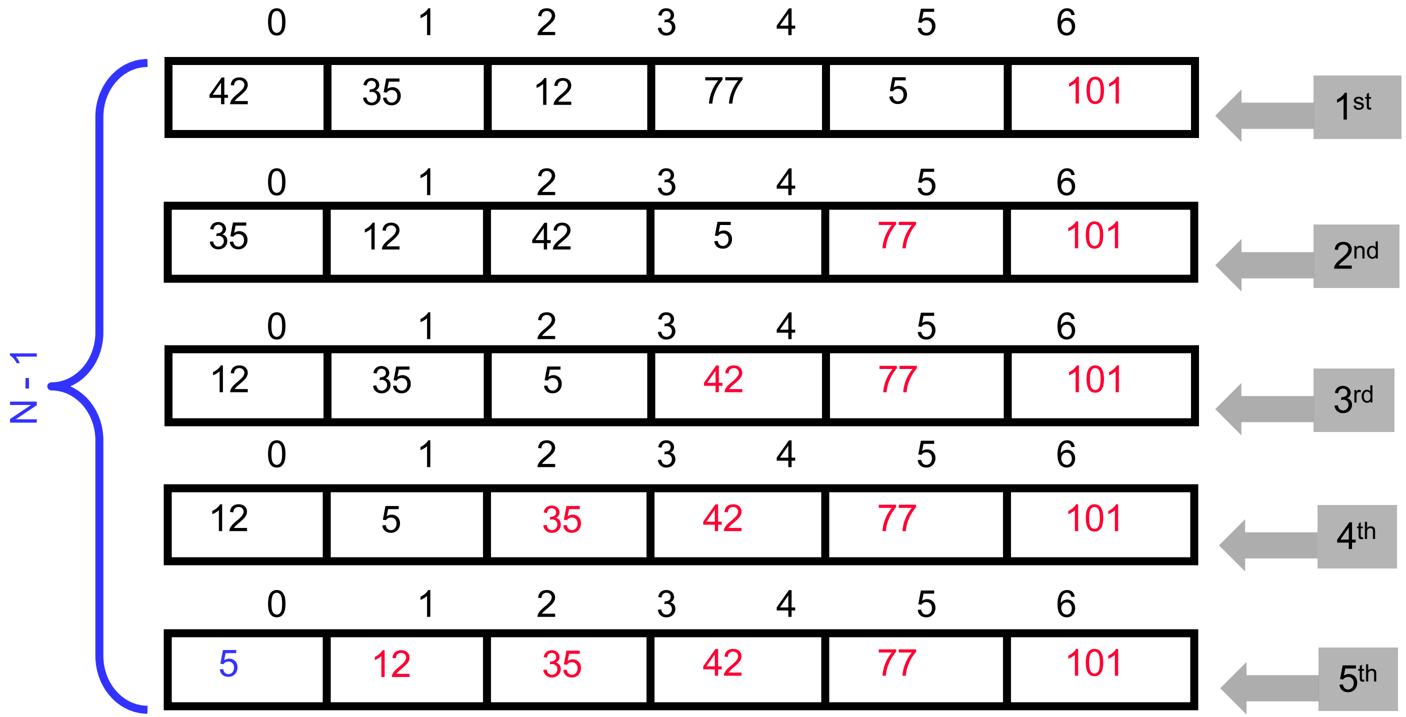


```
for i in range(N):

    for k in range(1,N):
        if A[k-1] > A[k]:
            Swap(A, k-1, k)

    N = N-1
```

N-1 Iterations



0	1	2	3	4	5	6
42	35	12	77	5	101	1st
0	1	2	3	4	5	6
35	12	42	5	77	101	2nd
0	1	2	3	4	5	6
12	35	5	42	77	101	3rd
0	1	2	3	4	5	6
12	5	35	42	77	101	4th
0	1	2	3	4	5	6
5	12	35	42	77	101	5th

Correctness of Bubble Sort

- Bubble sort has N-1 Iterations

- ***Invariant:*** *By the end of iteration i ➔ the right-most i items (largest) are sorted and in place*

- **Then:** After the N-1 iterations ➔ The right-most N-1 items are sorted
    - This implies that all the N items are sorted

That's all Folks!
Any Question ?