Data Structures and Algorithm Analysis

14

Dr. Syed Asim Jalal
Department of Computer Science
University of Peshawar

Bubble Sort

- Bubble sort is a *simple* sorting algorithm.
- This sorting algorithm is a *comparison based* algorithm in which each pair of adjacent elements is compared and the elements are *swapped if they are not in order*.
- It "Bubbles" the largest value to the end
- Although the algorithm is simple, it is too slow and impractical for most problems

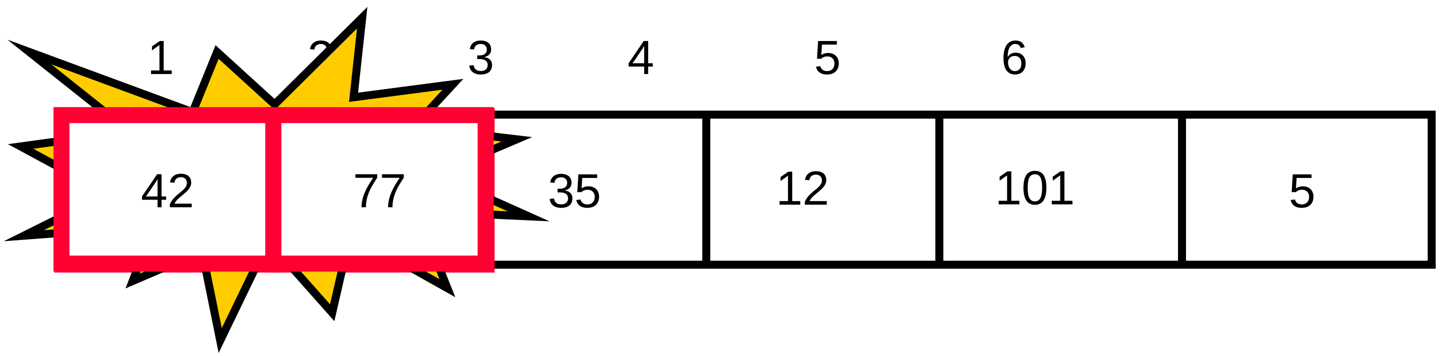
"Bubbling Up" the Largest Element

- **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

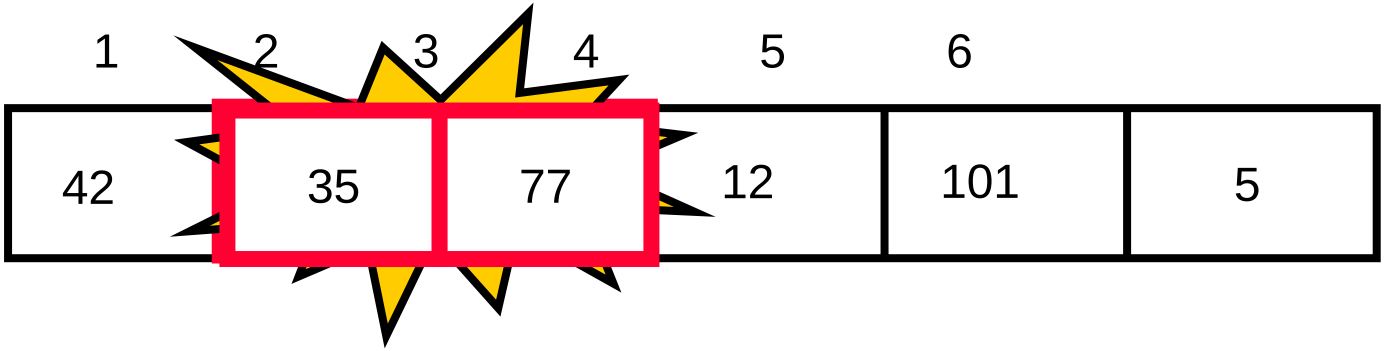
"Bubbling Up" the Largest Element

- **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



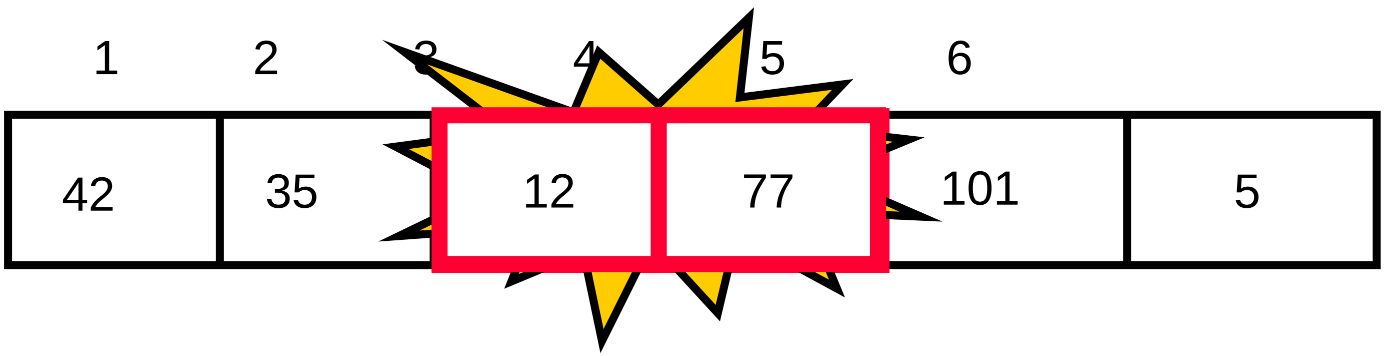
"Bubbling Up" the Largest Element

- **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

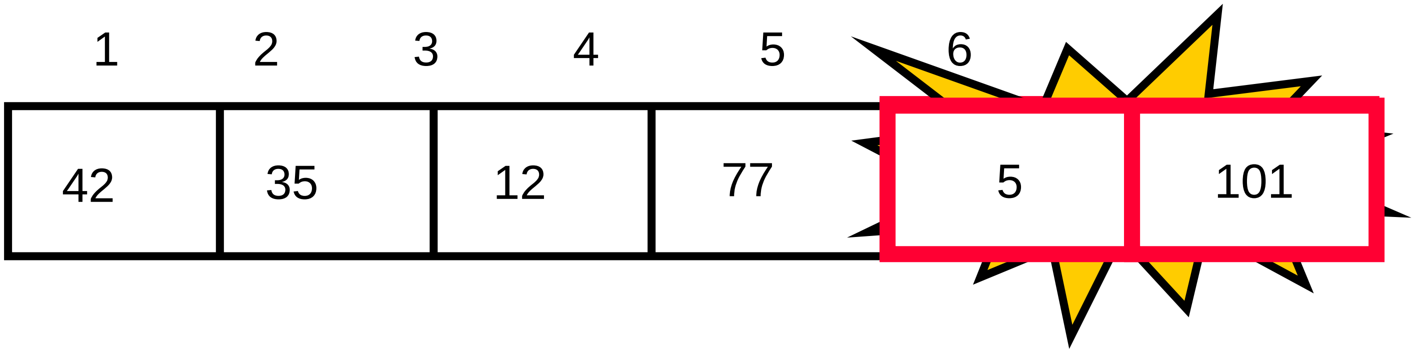
- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**

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



"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	$\boxed{101}$

Largest value correctly placed

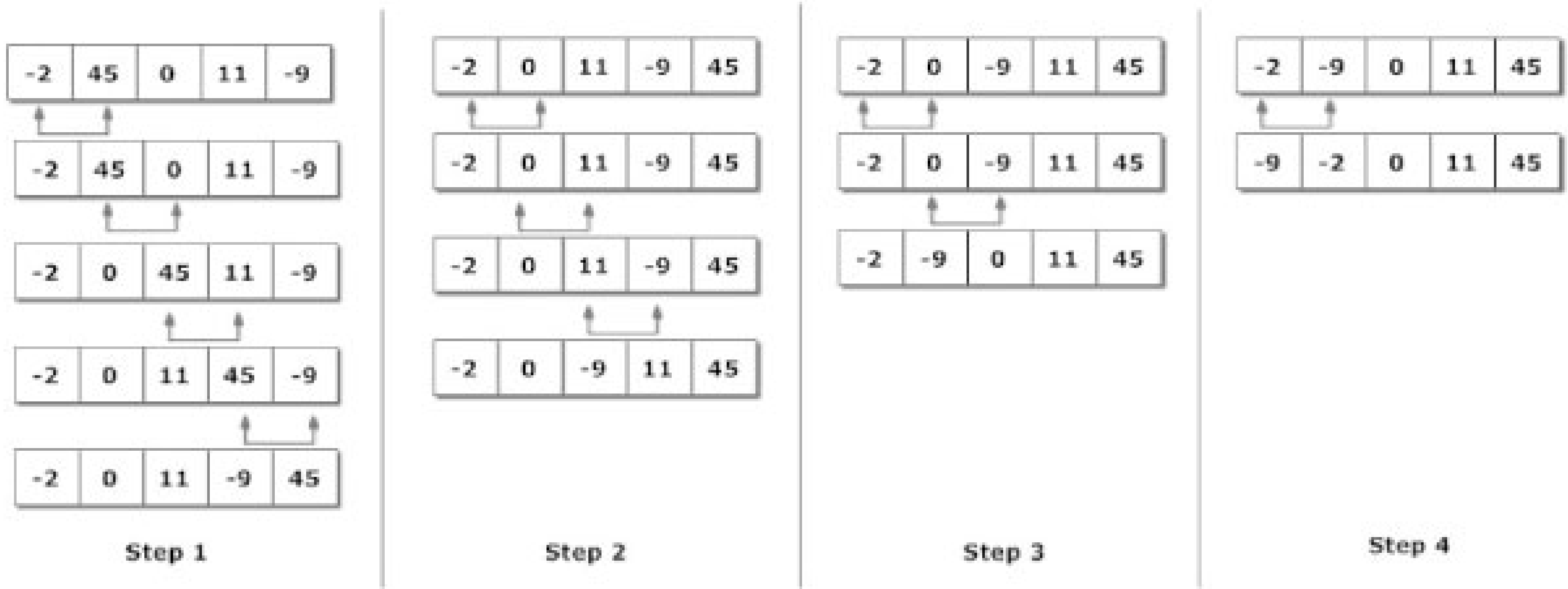


Figure: Working of Bubble sort algorithm

Reducing the Number of Comparisons in each pass

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

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

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

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

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

- Bubble pass requires n-1 passes over the array to sort the array.
- In each pass every adjacent elements a[i] **and** a[i+1] are compared.
- In each pass we have n-k comparisons, where k is the kth pass.

- Total comparisons are
  - ✓ 1^{st} pass: n-1 comparisons
  - ✓ 2^{nd} pass: n-2, comparisons
  - ✓ ..
  - ✓ Last pass : 1 comparison

Bubble Sort Algorithm

Algorithm for sorting an Array ‘A’ of size ‘N’.

BUBBLESORT(A, N)

for passNo = 1 **to** N - 1

{

for j = 0 **to** N - 1 - passNo

if A[j] > A[j+1]

then exchange A[j] $\leftrightarrow$ A[j+1]

}

```c
int main()
{
    int a[5] = {5,4,3,2,1};
    int n=5;
    int temp=0;

    for (int x=0; x<n; x++)
    printf("\n%d", a[x]);


    for (int i=0; i<n-1; i++)
        for (int j=0; j<n-i-1; j++)
         {
            if (a[j] > a[j+1])
            {
                temp = a[j+1];
                a[j+1] = a[j];
                a[j] = temp;

            }
         }
    printf("\nAfter Sorting");
    for (int x=0; x<n; x++)
    printf("\n%d", a[x]);

    printf("\n");
    return 0;
}
```

```
5
4
3
2
1
After Sorting
1
2
3
4
5
```

Time Complexity of Bubble Sort

(Analysis of Bubble Sort)

Bubble Sort - Analysis

Algorithm for sorting an Array ‘A’ of size ‘N’.

How many times each statement is executed?

BUBBLESORT(A, N)

for passNo = 1 **to** N - 1 — N times

{

for j = 0 **to** N - 1 - passNo — ???

if A[j] > A[j+1] — ???

then exchange A[j] ↔ A[j+1] — ???

}

Bubble Sort - Analysis

Algorithm for sorting an Array 'A' of size 'N'.

```
BUBBLESORT(A, N)
for  passNo = 1    to   N  - 1
{
   for  j  =  0   to   N -  1 - passNo
   if    A[j]  >  A[j+1]
         then exchange A[j] ↔ A[j+1]
}
```

The number of times the inner **IF statement** is executed will give us the running time or time complexity of the algorithm.

This represents total comparisons

Bubble Sort - Analysis

```
BUBBLESORT(A, N)
for  passNo = 1    to   N  - 1
{
    for  j  =  0   to   N -  1 - passNo
    if    A[j]  >  A[j+1]
          then exchange A[j] ↔ A[j+1]
}
```

Total comparisons (IF statement executed) in each pass:

- ✓ 1^{st} pass: n-1
- ✓ 2^{nd} pass: n-2
- ✓ 3^{rd} pass: n-3
- ✓ ..
- ✓ $N-1^{th}$ pass : n-(n-1) = 1

It makes the following series.

(n-1), (n-2), (n-3), …….., 3, 2, 1

What will be the total comparisons in n-1 passes?

We will sum the following series to get the total comparison in n-1 passes of bubble sort.

(n-1), (n-2), (n-3), …….., 3, 2, 1

$$
\begin{aligned}
&(n-1)+(n-2)+(n-3)+\cdots+(n-(n-2))+(n-(n-1)) \\
&=1+2+3+\cdots+n-1 \\
&=\sum_{1}^{n-1} i \\
&=\frac{(n-1)((n-1)+1)}{2} \\
&=\frac{n(n-1)}{2} \\
&=O(n^2)
\end{aligned}
$$

Selection sort

Selection Sort

Basic Concept

- Find the smallest element in the array
- Exchange it with the element in the first position
- Find the second smallest element and exchange it with the element in the second position
- Continue until the array is sorted

Example

8	4	6	9	2	3	1

1	4	6	9	2	3	8

1	2	6	9	4	3	8

1	2	3	9	4	6	8

1	2	3	4	9	6	8

1	2	3	4	6	9	8

1	2	3	4	6	8	9

1	2	3	4	6	8	9

Selection Sort

```
SELECTION-SORT(A)
   n ← length[A]
   for j ← 0 to n - 2
   {         IndexOfsmallest ← j
             for i ← j+1 to n-1
             {     if A[i]   <   A[IndexOfsmallest]
                        then IndexOfsmallest ← i
             }
             exchange A[j] ↔ A[IndexOfsmallest]
   }
```

Selection Sort

```
SELECTION-SORT(A)
   n ← length[A]
   for j ← 0 to n - 2
   {        IndexOfsmallest ← j
            for i ← j+1 to n-1
            {     if A[i]   <   A[IndexOfsmallest]
                        then IndexOfsmallest ← i
            }
            exchange A[j] ↔ A[IndexOfsmallest]
   }
```

SELECTION-SORT*(A)*

n ← length[A]

for j ← 0 **to** **n - 2**

{ IndexOfsmallest ← j

for i ← **j+1** **to** **n-1**

{ **if** A[i] < A[IndexOfsmallest]

then IndexOfsmallest ← i

}

exchange A[j] ↔ A[IndexOfsmallest]

}

Selection Sort - Analysis

Iteration of outer loop	No. times array comparison performed during this iteration of outer loop
0	n-1
1	n-2
2	n-3
. . .	
last	1

So number of comparisons is

$$1 + 2 + 3 + \ldots + (n-2) + (n-1) = n * (n-1) / 2 = n^2/2 - n/2$$

As n gets large, the term n^2 dominates. We say the number if comparisons *is proportional to* n^2 and that this is a *quadratic* algorithm.

Suppose there are total 6 values, that is , n = 6

for j ← 0 to n - 2	N-1			
IndexOfsmallest ← j	N-2			
for (i= j+1, n-1)	**j**	**i**		Series
	J=0	**1,2,3,4,5,6**	6	**n-1**
	J=1	**2,3,4,5,6**	5	
	J=2	**3,4,5,6**	4	
	J=3	**4,5,6**	3	
	J=4	**5,6**	2	**2**
				Series
if A[i] < A[IndexOfsmallest]	**J=0**	**1,2,3,4,5**	5	**(N-1)**
then IndexOfsmallest ← i	**J=1**	**2,3,4,5**	4	
	J=2	**3,4,5**	3	
	J=3	**4,5**	2	
	J=4	**5**	1	**1**
exchange A[j] ↔ A[IndexOfsmallest]	N-2			