Insertion Sort

Objectives

- In this Week, you will learn to:
  - Sort data by using insertion sort

Sorting Data

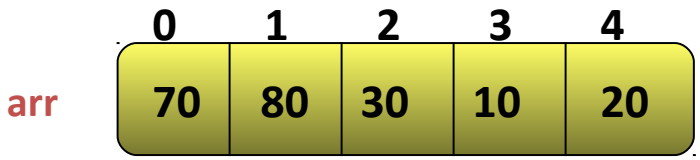
- Sorting is the process of arranging data in some pre-defined order or sequence. The order can be either ascending or descending.
- If the data is ordered, you can directly go to the section , thereby reducing the number of records to be traversed.

Sorting Data by Using Insertion Sort

- Insertion sort algorithm:
  - Has a quadratic order of growth and is therefore suitable for sorting small lists only
  - Is much more efficient than bubble sort, and selection sort, if the list that needs to be sorted is nearly sorted

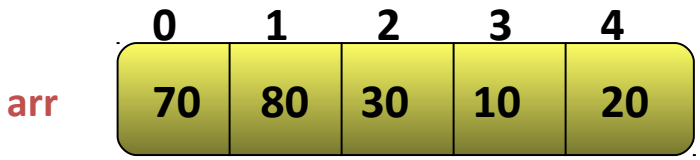
Implementing Insertion Sort Algorithm

- To understand the implementation of insertion sort algorithm, consider an unsorted list of numbers stored in an array.



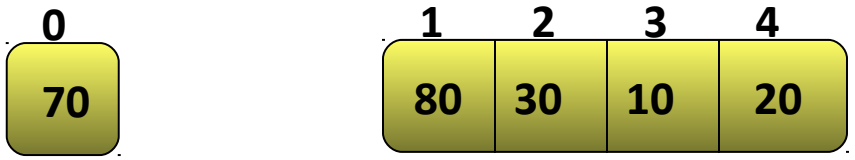
Implementing Insertion Sort Algorithm (Contd.)

- To sort this list by using the insertion sort algorithm:
  - You need to divide the list into two sublists, sorted and unsorted.



Implementing Insertion Sort Algorithm (Contd.)

- To sort this list by using the insertion sort algorithm:
  - You need to divide the list into two sublists, sorted and unsorted.
  - Initially, the sorted list has the first element and the unsorted list has the remaining 4 elements.



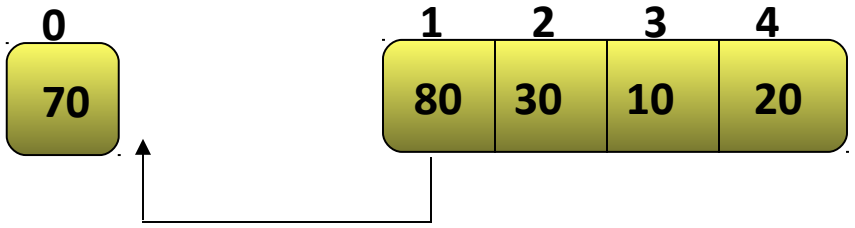
Sorted List

Unsorted List

Implementing Insertion Sort Algorithm (Contd.)

Pass 1

- Place the first element from the unsorted list at its correct position in the sorted list.



Implementing Insertion Sort Algorithm (Contd.)

Pass 1

- Place the first element from the unsorted list at its correct position in the sorted list.

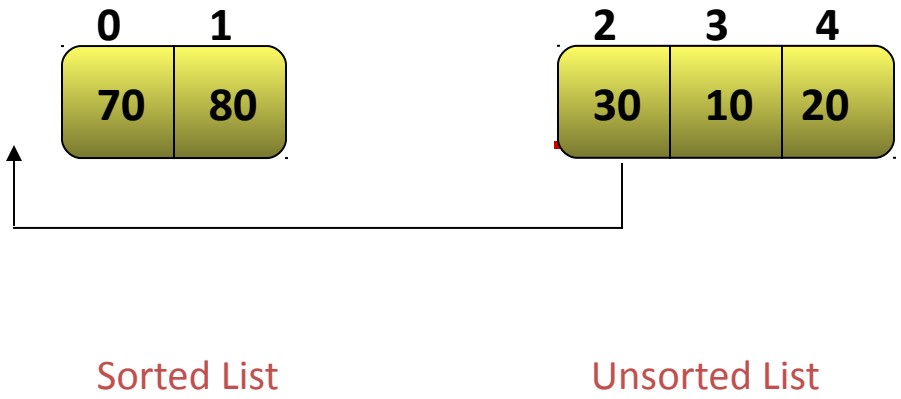
Sorted List

Unsorted List

Implementing Insertion Sort Algorithm (Contd.)

Pass 2

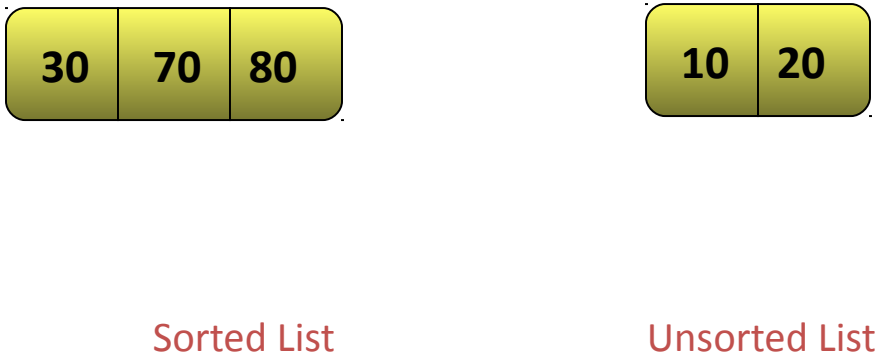
- Place the first element from the unsorted list at its correct position in the sorted list.



Implementing Insertion Sort Algorithm (Contd.)

Pass 2

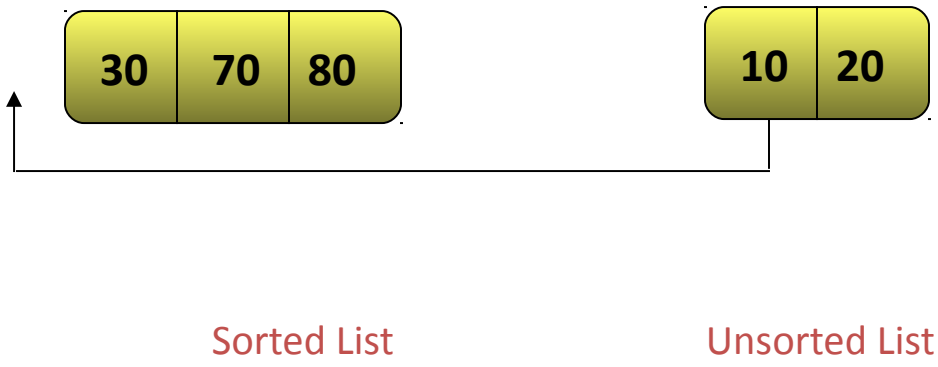
- Place the first element from the unsorted list at its correct position in the sorted list.



Implementing Insertion Sort Algorithm (Contd.)

Pass 3

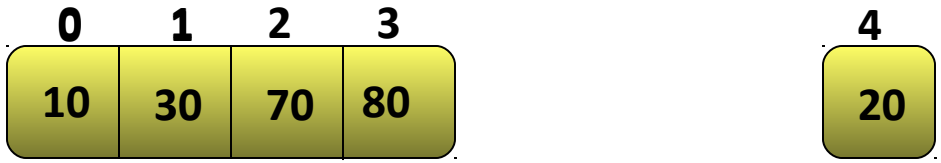
- Place the first element from the unsorted list at its correct position in the sorted list.



Implementing Insertion Sort Algorithm (Contd.)

Pass 3

- Place the first element from the unsorted list at its correct position in the sorted list.



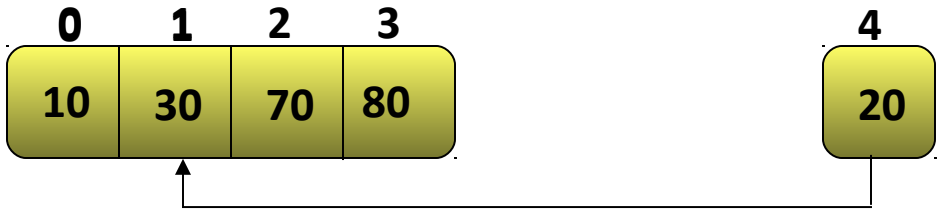
Sorted List

Unsorted List

Implementing Insertion Sort Algorithm (Contd.)

Pass 4

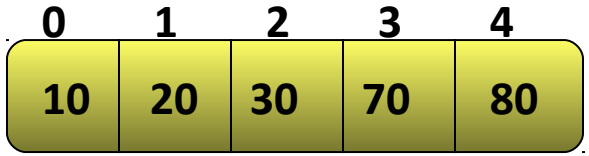
- Place the first element from the unsorted list at its correct position in the sorted list.



Implementing Insertion Sort Algorithm (Contd.)

Pass 4

- Place the first element from the unsorted list at its correct position in the sorted list.

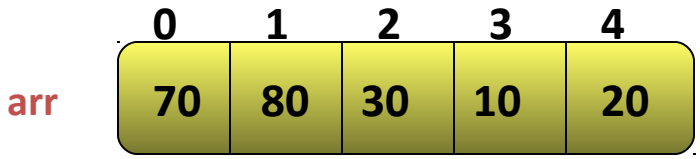


Sorted List

Unsorted List

Implementing Insertion Sort Algorithm (Contd.)

- Let us now write an algorithm to implement insertion sort algorithm.



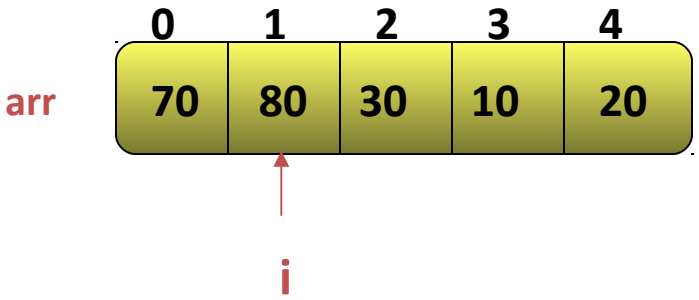

```
Algorithm ( A [],n )
 {
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
 }
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80



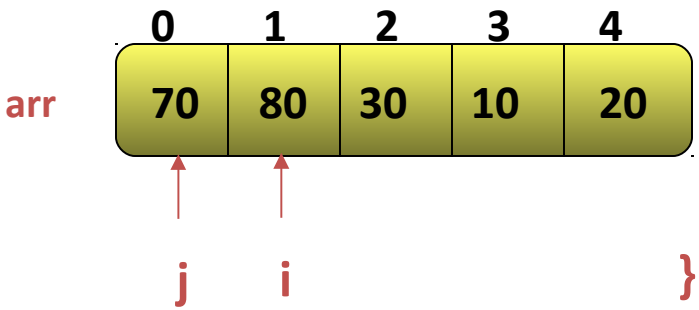

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

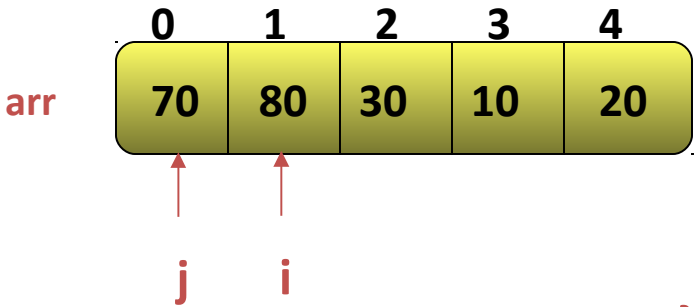
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80

A [j] < temp




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

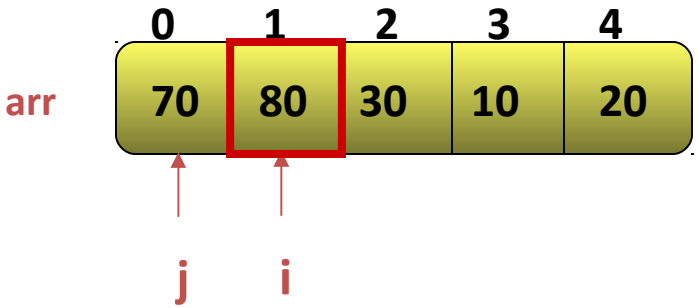
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80

A [j] < temp



Value temp is stored at its correct position in the sorted list

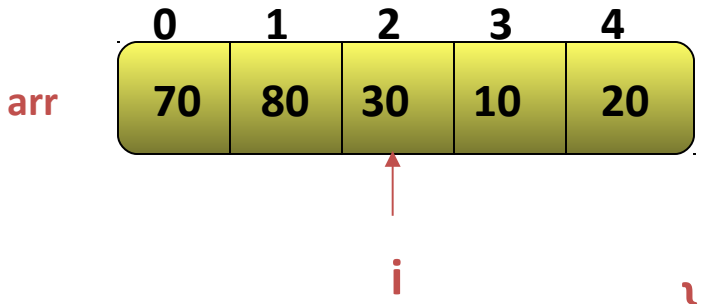
```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

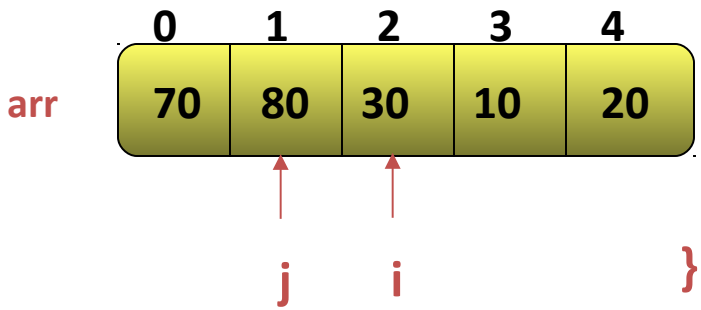
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

A [j] > temp




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

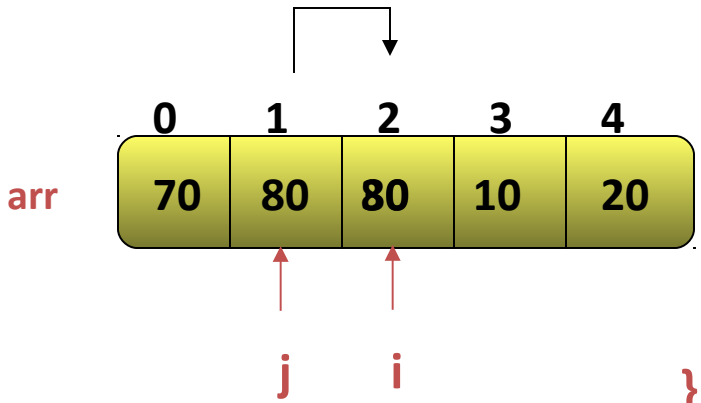
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

A [j] > temp




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

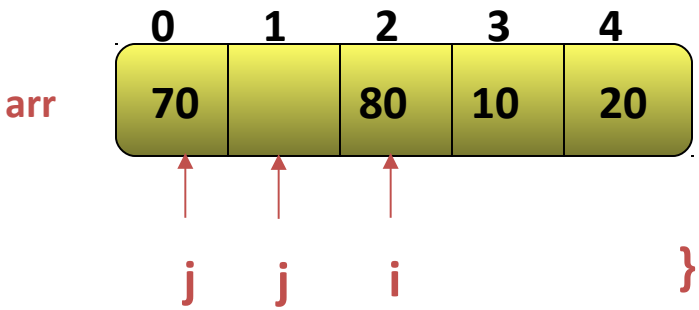
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

A [j] > temp



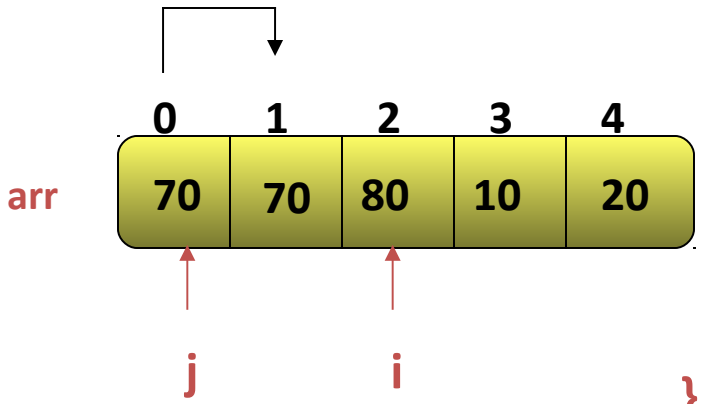

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

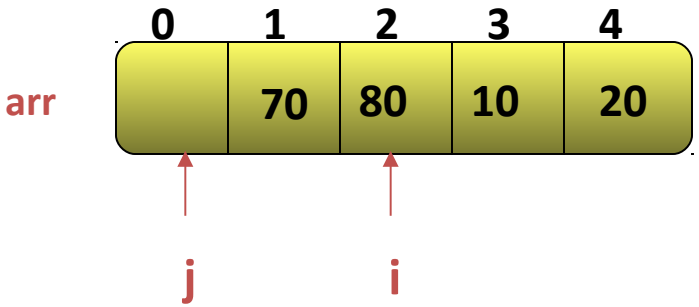
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

j = –1




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

             1.3.1 A [j+1] = A [ j ]
             1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

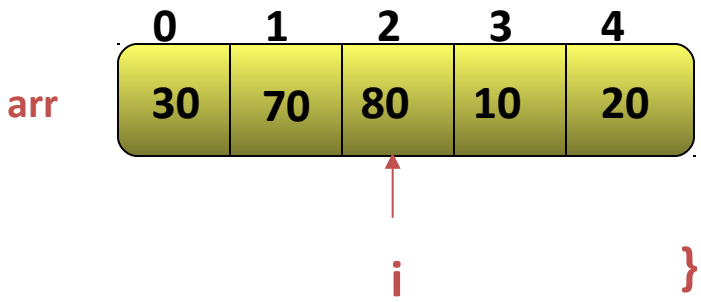
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

j = –1



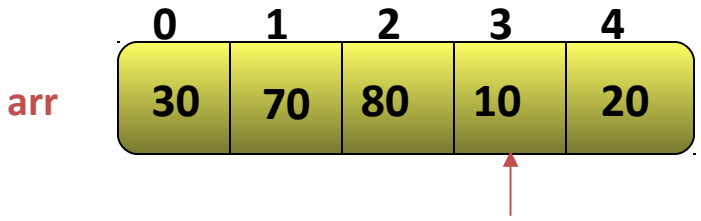

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

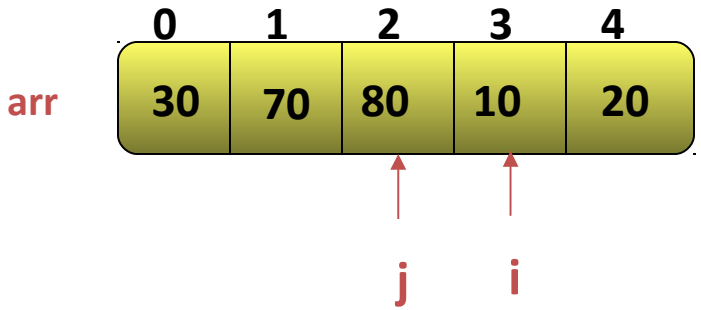
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

A [j] > temp



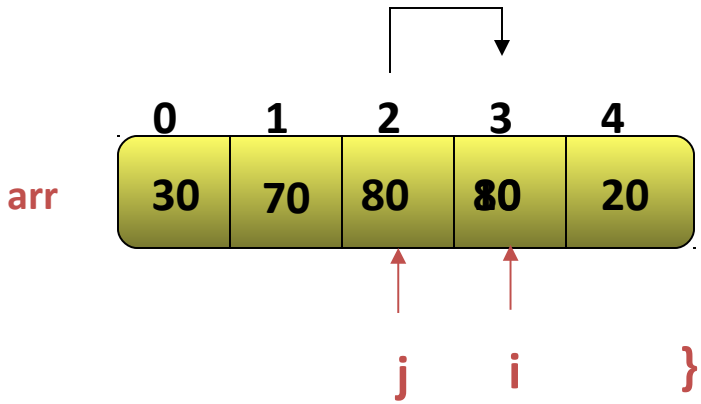

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

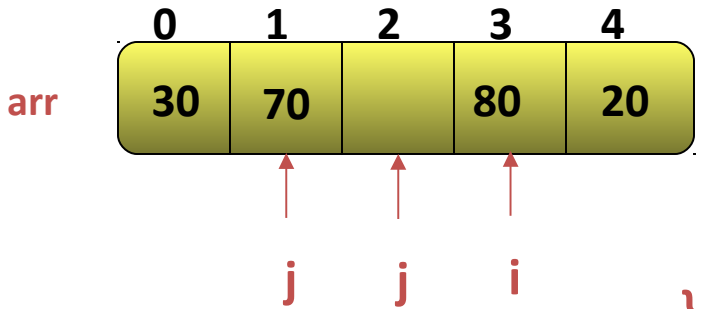
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

A [j] > temp



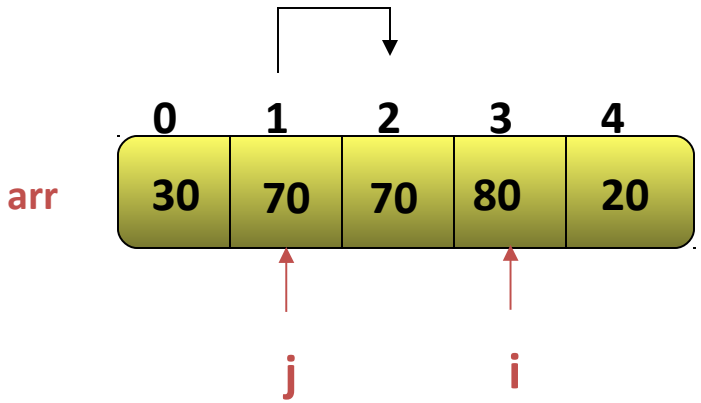

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

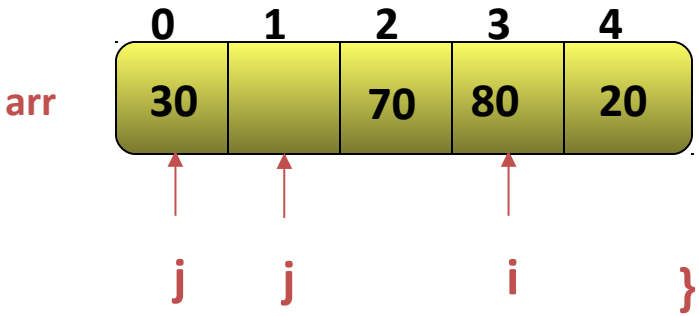
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

A [j] > temp



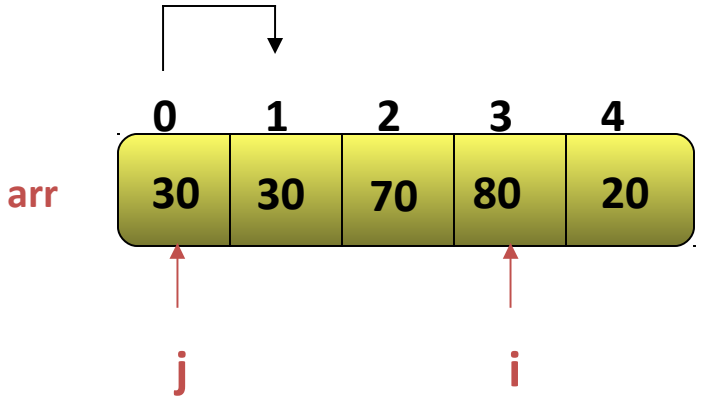

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

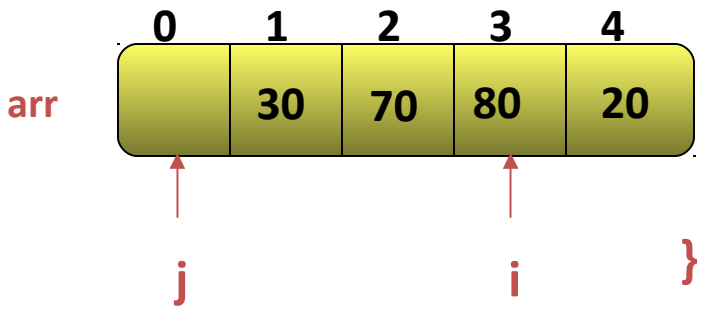
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

j = –1




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

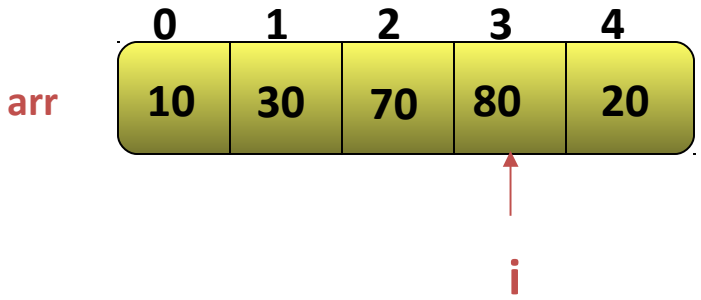
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

j = –1



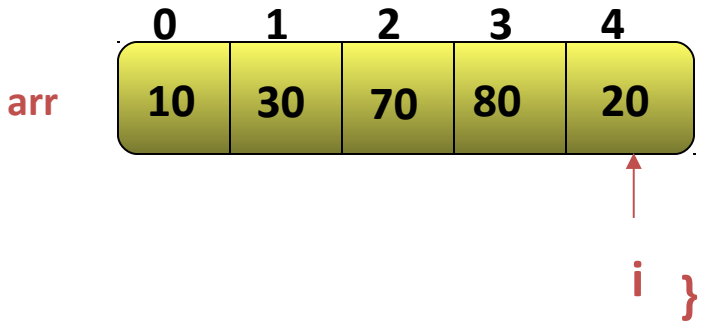

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

             1.3.1 A [j+1] = A [ j ]
             1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

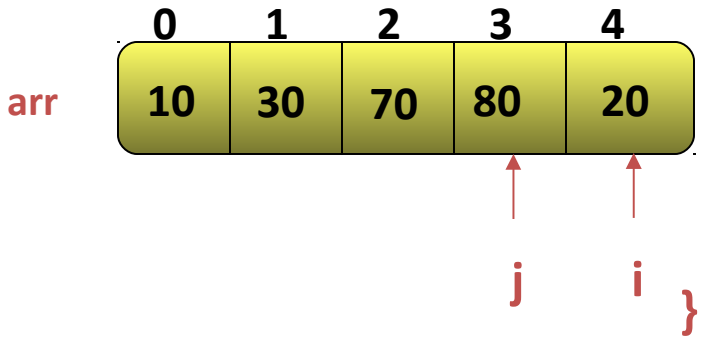
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

A [j] > temp



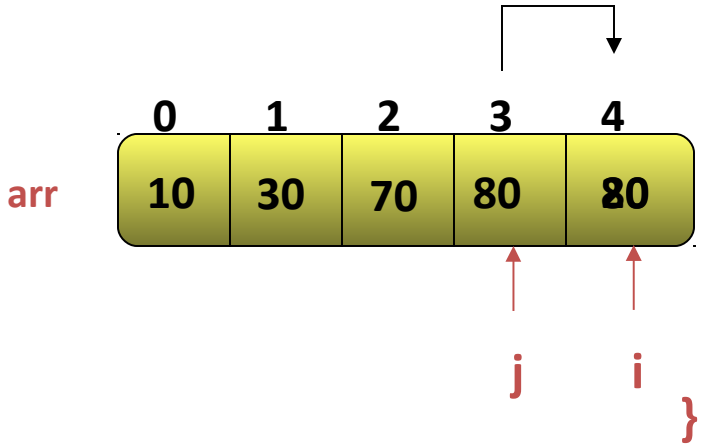

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

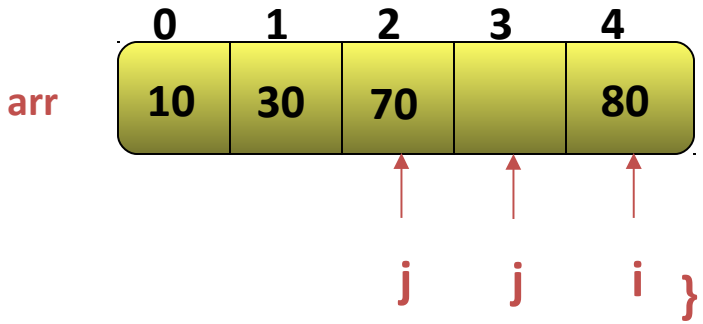
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

A [j] > temp



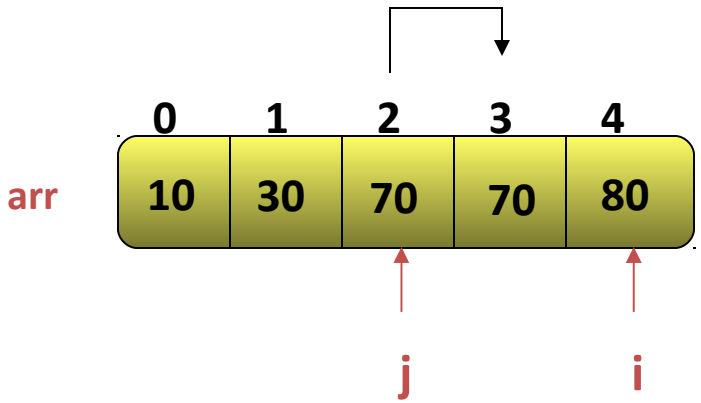

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

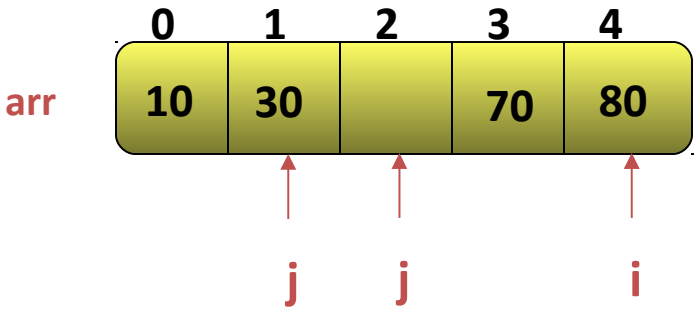
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

A [j] > temp



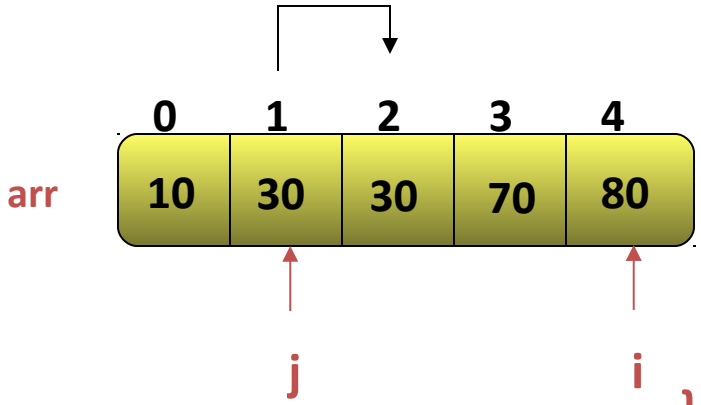

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20




```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
       1.1 Set temp = A [i]

       1.2 Set j = i – 1

       1.3 while ( A [ j ]  > temp && j >= 0 )

            1.3.1 A [j+1] = A [ j ]
            1.3.2  j = j – 1

       1.4 Store temp at index j + 1
}
```

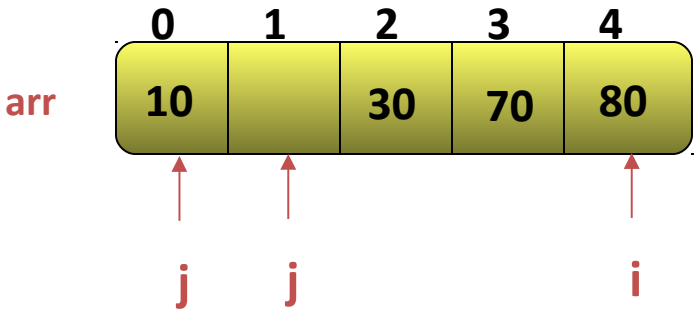
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

A [j] < temp



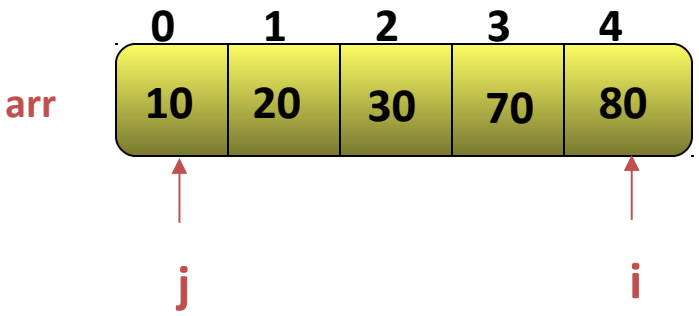

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

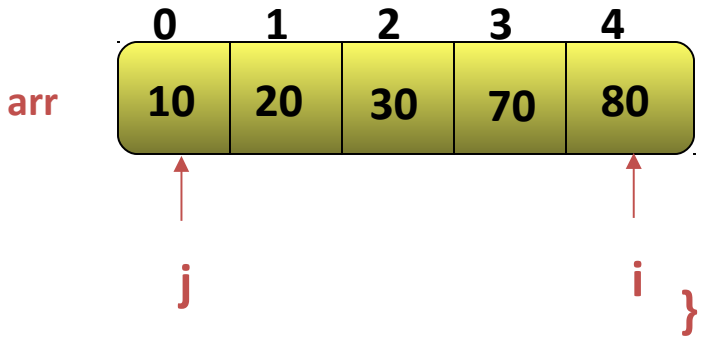



```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4



The list is now sorted

```
Algorithm ( A [],n )
{
  1. for ( i=1 to n-1 )
        1.1 Set temp = A [i]

        1.2 Set j = i – 1

        1.3 while ( A [ j ]  > temp && j >= 0 )

              1.3.1 A [j+1] = A [ j ]
              1.3.2  j = j – 1

        1.4 Store temp at index j + 1
}
```

Determining the Efficiency of Insertion Sort Algorithm

- Best Case Efficiency:
  - Best case occurs when the list is already sorted.
  - In this case, you will have to make only one comparison in each pass.
  - In n – 1 passes, you will need to make n – 1 comparisons.
  - The best case efficiency of insertion sort is of the order O(n).

To sort a list of size n by using insertion sort, you need to perform (n – 1) passes.

- Worst Case Efficiency:
  - Worst case occurs when the list is sorted in the reverse order.
  - In this case, you need to perform one comparison in the first pass, two comparisons in the second pass, three comparisons in the third pass, and n – 1 comparisons in the $(n - 1)^{th}$ pass.
  - The worst case efficiency of insertion sort is of the order $O(n^2)$.

Just a minute

- A sales manager has to do a research on best seller cold drinks in the market for the year 2004-2006. David, the software developer, has a list of all the cold drink brands along with their sales figures stored in a file. David has to provide the sorted data to the sales manager. The data in the file is more or less sorted. Which sorting algorithm will be most efficient for sorting this data and why?

- Answer:
  - Insertion sort provides better efficiency than bubble sort and selection sort when the list is partially sorted. Therefore, David should use the insertion sort algorithm.