Objectives

- In this session, you will learn to:
  - Identify the algorithms that can be used to sort data
  - Sort data by using bubble sort
  - Sort data by using selection sort
  - Sort data by using insertion sort
  - Sort data by using shell sort

Sorting Data

- Suppose, you have to invite your friends and relatives for your birthday party. For this, you need to give them a call.
- You are looking for the telephone number of a friend named Steve in a telephone directory with 10,000 records.
- However, the records are stored randomly and not in any particular order.

Sorting Data (Contd.)

- To search for the telephone number of your friend in such a directory, you would need to scan each entry in the directory in a sequential manner.
- This would be very time consuming.
- How can you solve this problem?

Sorting Data (Contd.)

- A simple solution to save time, and search records efficiently is sorting.
- 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 that stores the names starting with ‘S’, thereby reducing the number of records to be traversed.

Selecting a Sorting Algorithm

- Sorting is implemented in a program by using an algorithm.
- Some sorting algorithms are:
  - Bubble sort
  - Selection sort
  - Insertion sort
  - Shell sort
  - Merge sort
  - Quick sort
  - Heap sort
- To select an appropriate algorithm, you need to consider the following:
  - Execution time
  - Storage space
  - Programming effort

Sorting Data by Using Bubble Sort

- Bubble sort algorithm:
  - Is one of the simplest sorting algorithms
  - Has a quadratic order of growth and is therefore suitable for sorting small lists only
  - Works by repeatedly scanning through the list, comparing adjacent elements, and swapping them if they are in the wrong order

Implementing Bubble Sort Algorithm

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

	0	1	2	3	4
arr	5	2	6	7	3

Implementing Bubble Sort Algorithm (Contd.)

- Let us sort this unsorted list.

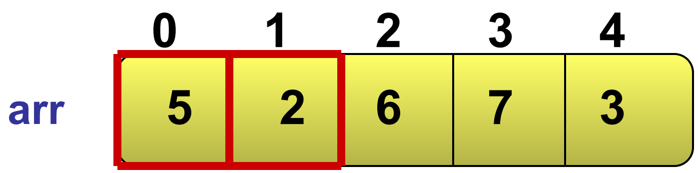
	0	1	2	3	4
arr	5	2	6	7	3

Implementing Bubble Sort Algorithm (Contd.)

Pass 1

n = 5

- Compare the element stored at index 0 with the element stored at index 1.

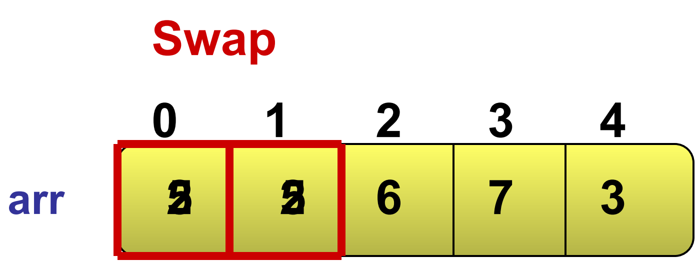


Implementing Bubble Sort Algorithm (Contd.)

Pass 1

n = 5

- Swap the values if they are not in the correct order.



Implementing Bubble Sort Algorithm (Contd.)

Pass 1

n = 5

- Compare the element stored at index 1 with the element stored at index 2 and swap the values if the value at index 1 is greater than the value at index 2.

No Change

	0	1	2	3	4
arr	2	5	6	7	3

Implementing Bubble Sort Algorithm (Contd.)

Pass 1

n = 5

- Compare the element stored at index 2 with the element stored at index 3 and swap the values if the value at index 2 is greater than the value at index 3.

No Change

	0	1	2	3	4
arr	2	5	6	7	3

Implementing Bubble Sort Algorithm (Contd.)

Pass 1

n = 5

- Compare the element stored at index 3 with the element stored at index 4 and swap the values if the value at index 3 is greater than the value at index 4.

Swap

	0	1	2	3	4
arr	2	5	6	7	3

Implementing Bubble Sort Algorithm (Contd.)

Pass 1

n = 5

- Compare the element stored at index 3 with the element stored at index 4 and swap the values if the value at index 3 is greater than the value at index 4.

	0	1	2	3	4
arr	2	5	6	3	7

Largest element is placed at its correct position after Pass 1

Implementing Bubble Sort Algorithm (Contd.)

Pass 2

n = 5

- Compare the element stored at index 0 with the element stored at index 1 and swap the values if the value at index 0 is greater than the value at index 1.

No Change

	0	1	2	3	4
arr	2	5	6	3	7

Implementing Bubble Sort Algorithm (Contd.)

Pass 2

n = 5

- Compare the element stored at index 1 with the element stored at index 2 and swap the values if the value at index 1 is greater than the value at index 2.

No Change

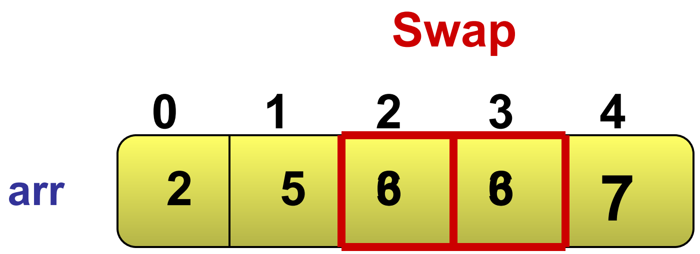
	0	1	2	3	4
arr	2	5	6	3	7

Implementing Bubble Sort Algorithm (Contd.)

Pass 2

n = 5

- Compare the element stored at index 2 with the element stored at index 3 and swap the values if the value at index 2 is greater than the value at index 3.



Implementing Bubble Sort Algorithm (Contd.)

Pass 2

n = 5

- Compare the element stored at index 2 with the element stored at index 3 and swap the values if the value at index 2 is greater than the value at index 3.

	0	1	2	3	4
arr	2	5	3	6	7

Second largest element is placed at its correct position after Pass 2

Implementing Bubble Sort Algorithm (Contd.)

Pass 3

n = 5

- Compare the element stored at index 0 with the element stored at index 1 and swap the values if the value at index 0 is greater than the value at index 1.

No Change

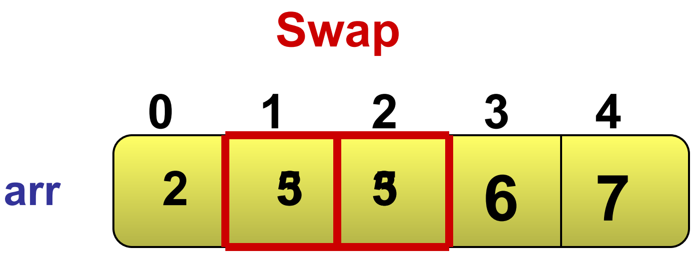
	0	1	2	3	4
arr	2	5	3	6	7

Implementing Bubble Sort Algorithm (Contd.)

Pass 3

n = 5

- Compare the element stored at index 1 with the element stored at index 2 and swap the values if the value at index 1 is greater than the value at index 2.



Implementing Bubble Sort Algorithm (Contd.)

Pass 3

n = 5

- Compare the element stored at index 2 with the element stored at index 3 and swap the values if the value at index 2 is greater than the value at index 3.

	0	1	2	3	4
arr	2	3	5	6	7

Third largest element is placed at its correct position after Pass 3

Implementing Bubble Sort Algorithm (Contd.)

Pass 4

n = 5

- Compare the element stored at index 0 with the element stored at index 1 and swap the values if the value at index 0 is greater than the value at index 1.

No Change

	0	1	2	3	4
arr	2	3	5	6	7

Implementing Bubble Sort Algorithm (Contd.)

Pass 4

n = 5

- Compare the element stored at index 0 with the element stored at index 1 and swap the values if the value at index 0 is greater than the value at index 1.

	0	1	2	3	4
arr	**2**	**3**	**5**	**6**	**7**

Fourth largest element is placed at its correct position after Pass 4

Implementing Bubble Sort Algorithm (Contd.)

Pass 4

n = 5

- At the end of Pass 4, the elements are sorted.

	0	1	2	3	4
arr	2	3	5	6	7

Implementing Bubble Sort Algorithm (Contd.)

- Write an algorithm to implement bubble sort.
- Algorithm for bubble sort:
  1. Set pass = 1.
  2. Repeat step 3 varying j from 0 to n – 1 – pass.
  3. If the element at index j is greater than the element at index j + 1, swap the two elements.
  4. Increment pass by 1.
  5. If pass <= n – 1, go to step 2.

Determining the Efficiency of Bubble Sort Algorithm

- The efficiency of a sorting algorithm is measured in terms of number of comparisons.
- In bubble sort, there are n – 1 comparisons in Pass 1, n – 2 comparisons in Pass 2, and so on.
- Total number of comparisons = (n – 1) + (n – 2) + (n – 3) + … + 3 + 2 + 1 = n(n – 1)/2.
- n(n – 1)/2 is of $O(n^2)$ order. Therefore, the bubble sort algorithm is of the order $O(n^2)$.

Just a minute

- What is the order of growth of the bubble sort algorithm?

- Answer:
  - The bubble sort algorithm has a quadratic order of growth.

Just a minute

- While implementing bubble sort algorithm, how many comparisons will be performed in Pass 1.

- Answer:
  - n – 1 comparisons

Activity: Sorting Data by Using Bubble Sort Algorithm

- Problem Statement:
  - Write a program to store the marks of 10 students in an array. Include a function to sort the elements of the array by using the bubble sort algorithm. After sorting the array, display the sorted list.

Sorting Data by Using Selection Sort

- Selection sort algorithm:
  - Has a quadratic order of growth and is therefore suitable for sorting small lists only
  - Scans through the list iteratively, selects one item in each scan, and moves the item to its correct position in the list

Implementing Selection Sort Algorithm

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

	0	1	2	3	4
arr	105	120	10	200	20

Implementing Selection Sort Algorithm (Contd.)

- Let us sort this unsorted list.

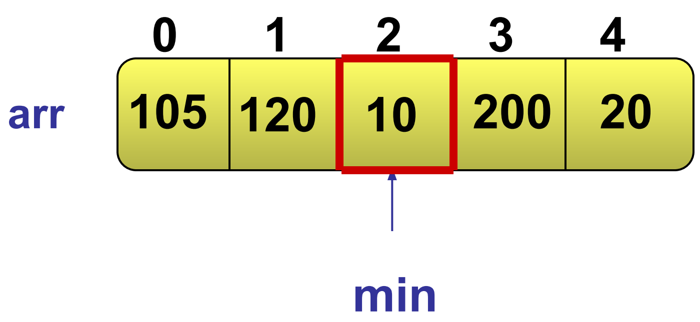
	0	1	2	3	4
arr	105	120	10	200	20

Implementing Selection Sort Algorithm (Contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1].

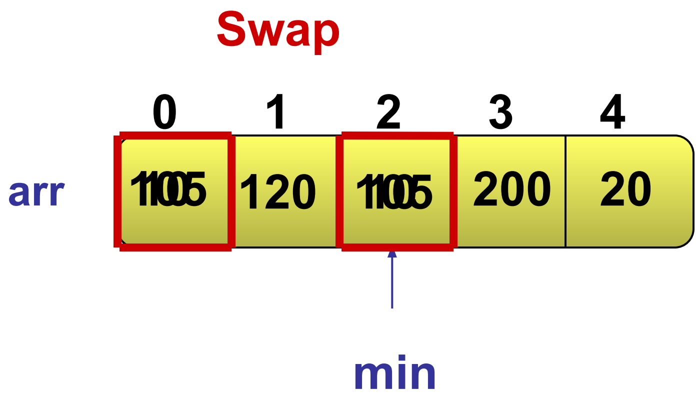


Implementing Selection Sort Algorithm (Contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1].
- Swap the minimum value with the value at index 0.



Implementing Selection Sort Algorithm (Contd.)

Pass 1

n = 5

- Search the minimum value in the array, arr[0] to arr[n – 1].
- Swap the minimum value with the value at index 0.

	0	1	2	3	4
arr	10	120	105	200	20

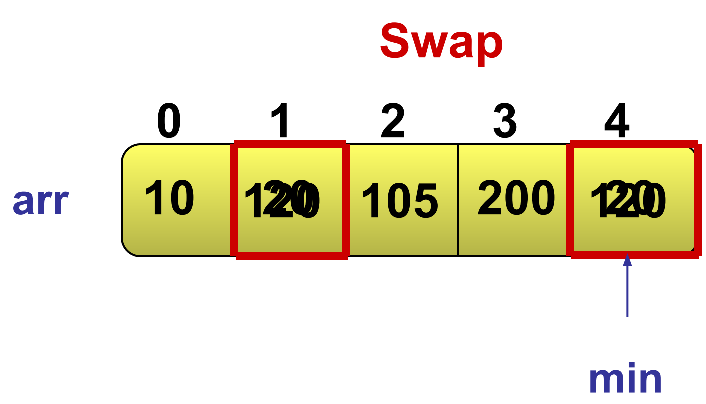
The smallest value is placed at its correct location after Pass 1

Implementing Selection Sort Algorithm (Contd.)

Pass 2

n = 5

- Search the minimum value in the array, arr[1] to arr[n – 1].
- Swap the minimum value with the value at index 1.

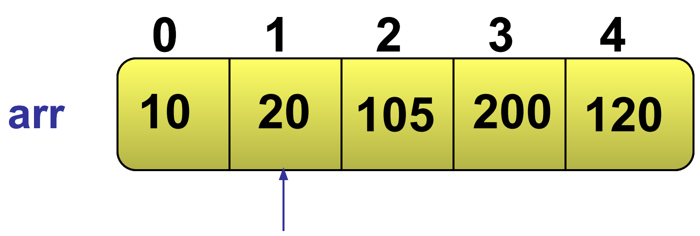


Implementing Selection Sort Algorithm (Contd.)

Pass 2

n = 5

- Search the minimum value in the array, arr[1] to arr[n – 1].
- Swap the minimum value with the value at index 1.



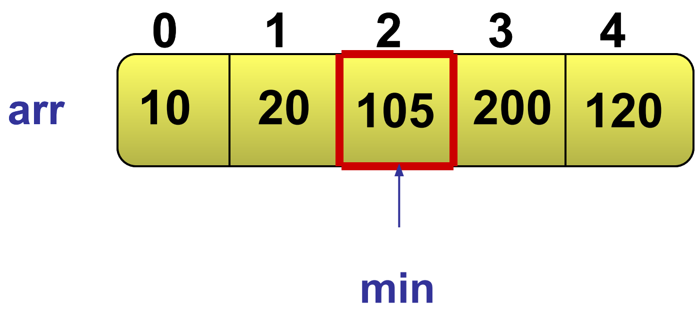
The second smallest value is placed at its correct location after Pass 2

Implementing Selection Sort Algorithm (Contd.)

Pass 3

n = 5

- Search the minimum value in the array, arr[2] to arr[n – 1].
- Swap the minimum value with the value at index 2.

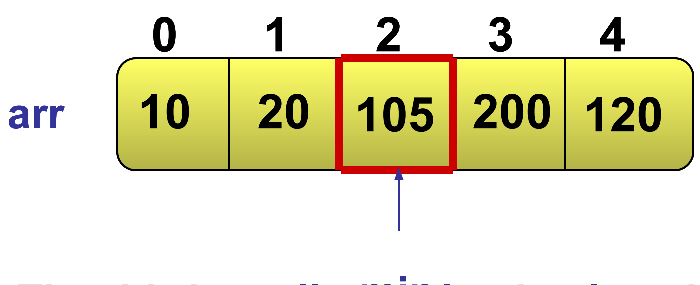


Implementing Selection Sort Algorithm (Contd.)

Pass 3

n = 5

- Search the minimum value in the array, arr[2] to arr[n – 1].
- Swap the minimum value with the value at index 2.



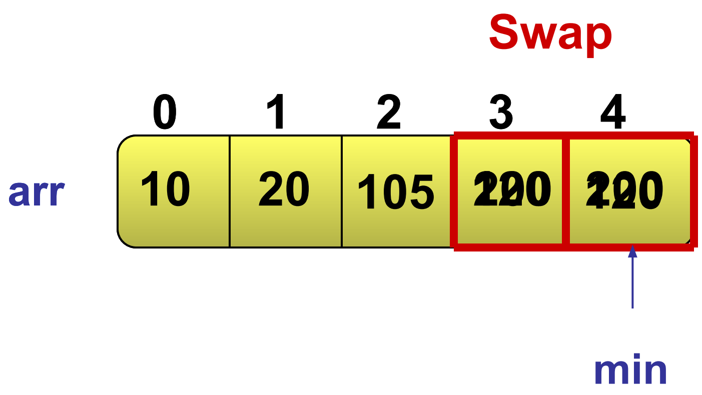
The third smallest value is placed at its correct location after Pass 3

Implementing Selection Sort Algorithm (Contd.)

Pass 4

n = 5

- Search the minimum value in the array, arr[3] to arr[n – 1].
- Swap the minimum value with the value at index 3.



Implementing Selection Sort Algorithm (Contd.)

Pass 4

n = 5

- Search the minimum value in the array, arr[3] to arr[n – 1].
- Swap the minimum value with the value at index 3.

	0	1	2	3	4
arr	**10**	**20**	**105**	**120**	**200**

The fourth smallest value is placed at its correct location after Pass 4

Implementing Selection Sort Algorithm (Contd.)

Pass 4

n = 5

- The list is now sorted.

	0	1	2	3	4
arr	10	20	105	120	200

Implementing Selection Sort Algorithm (Contd.)

- Write an algorithm to implement selection sort.
- Algorithm for selection sort:
  1. Repeat steps 2 and 3 varying j from 0 to n – 2
  2. Find the minimum value in arr[j] to arr[n – 1]:
     a. Set min_index = j
     b. Repeat step c varying i from j + 1 to n – 1
     c. If arr[i] < arr[min_index]:
        i. min_index = i
  3. Swap arr[j] with arr[min_index]

Determining the Efficiency of Selection Sort Algorithm

- In selection sort, there are n – 1 comparisons during Pass 1 to find the smallest element, n – 2 comparisons during Pass 2 to find the second smallest element, and so on.
- Total number of comparisons = $(n-1) + (n-2) + (n-3) + \ldots + 3 + 2 + 1 = n(n-1)/2$
- $n(n-1)/2$ is of $O(n^2)$ order. Therefore, the selection sort algorithm is of the order $O(n^2)$.

Just a minute

- Read the following statement and specify whether it is true or false:
  - The efficiency of selection sort algorithm is same as that of the bubble sort algorithm.

- Answer:
  - True

Just a minute

- How many comparisons are performed in the second pass of the selection sort algorithm?

- Answer:
  - n – 2

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.

	0	1	2	3	4
arr	**70**	**80**	**30**	**10**	**20**

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.

	0	1	2	3	4
arr	70	80	30	10	20

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.

0
70

Sorted List

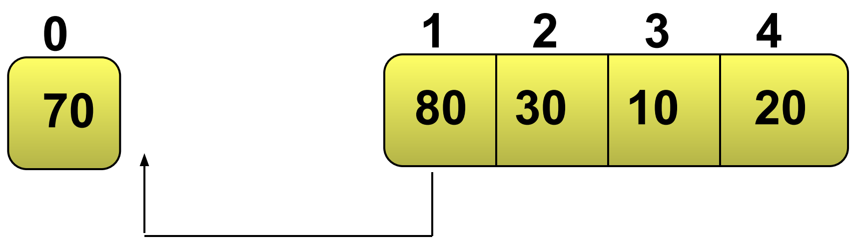
1	2	3	4
80	30	10	20

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.



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.

0	1
70	80

Sorted List

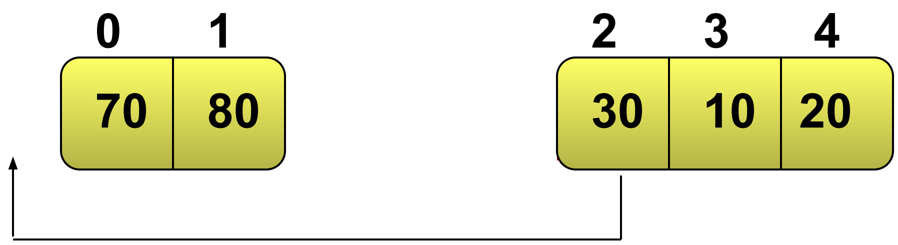
2	3	4
30	10	20

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.



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.

30	70	80

Sorted List

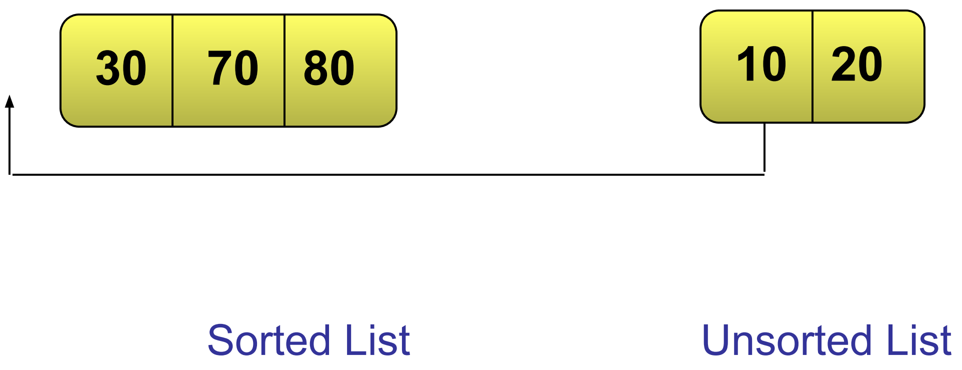
10	20

Unsorted 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.

0	1	2	3
10	30	70	80

4
20

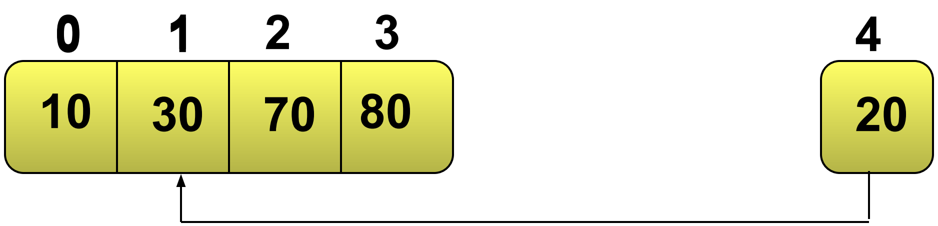
Sorted List

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.



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.

0	1	2	3	4
10	20	30	70	80

Sorted List

Unsorted List

Implementing Insertion Sort Algorithm (Contd.)

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

	0	1	2	3	4
arr	70	80	30	10	20

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

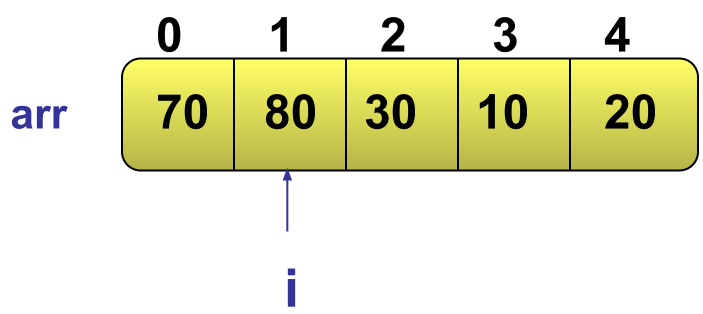
	0	1	2	3	4
arr	**70**	**80**	**30**	**10**	**20**

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1



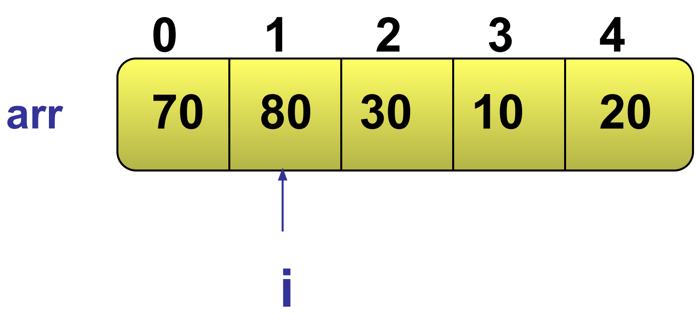
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80



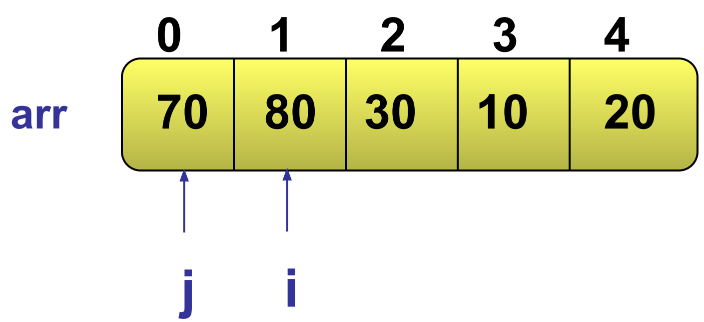
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80



1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

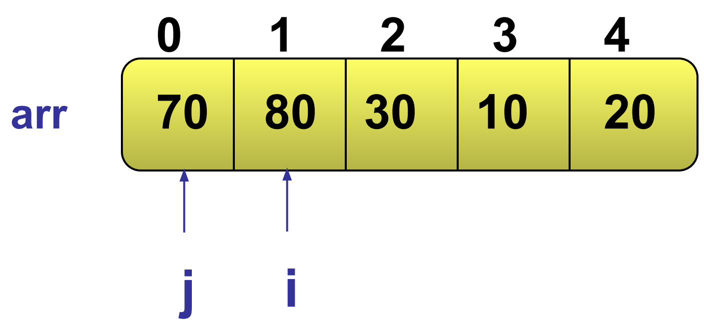
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80

arr[j] < temp



1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

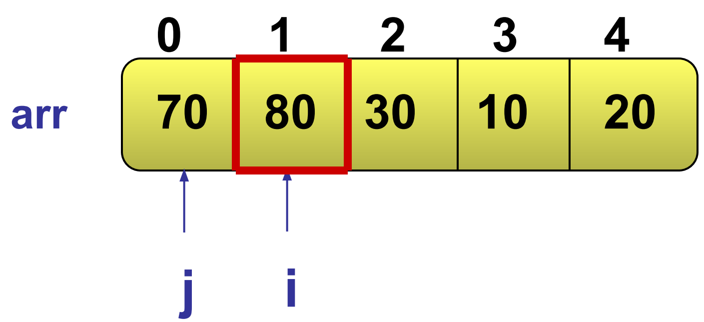
Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

temp = 80

arr[j] < temp



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

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 1

	0	1	2	3	4
arr	70	80	30	10	20

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

arr	0	1	2	3	4
	70	80	30	10	20

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

	0	1	2	3	4
arr	70	80	30	10	20

↑

i

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

	0	1	2	3	4
arr	**70**	**80**	**30**	**10**	**20**

i

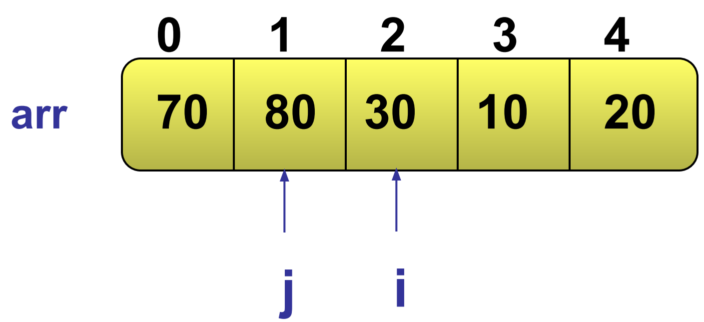
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30



1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

	0	1	2	3	4
arr	70	80	30	10	20

j i

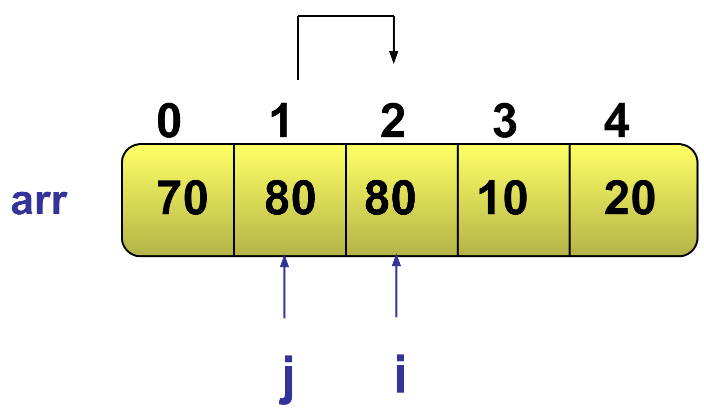
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30



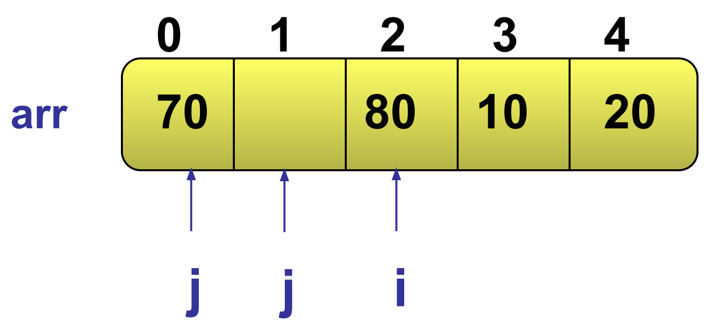
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30



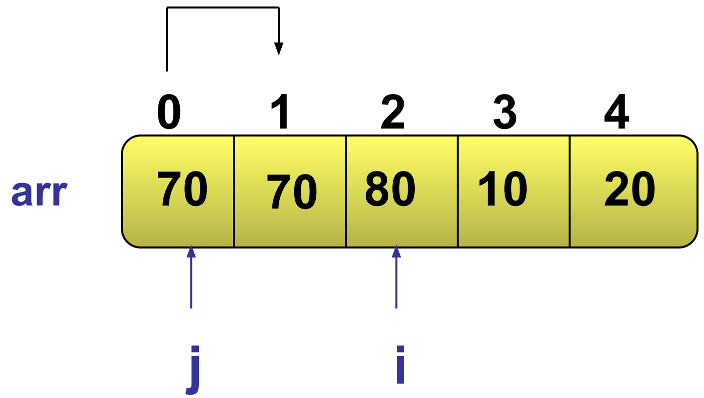
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30



1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

j = –1

	0	1	2	3	4
arr		70	80	10	20
	j		i		

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

temp = 30

j = –1

	0	1	2	3	4
arr	30	70	80	10	20

i

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 2

	0	1	2	3	4
arr	**30**	**70**	**80**	**10**	**20**

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

	0	1	2	3	4
arr	30	70	80	10	20
				i	

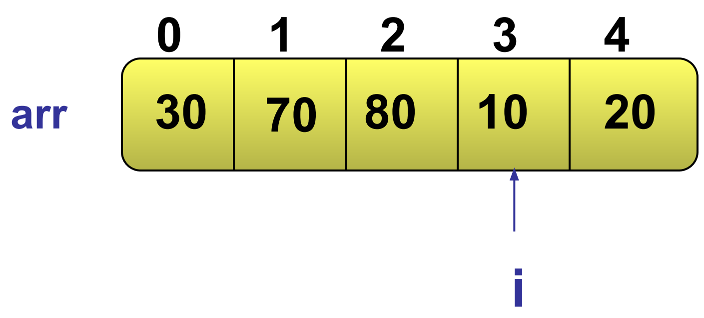
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



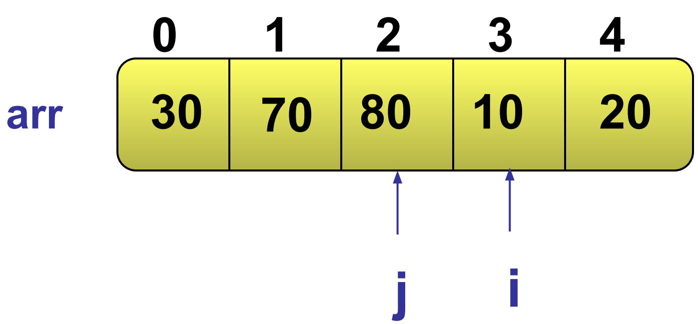
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



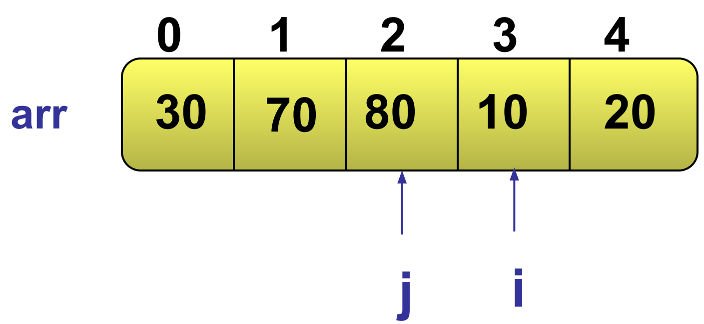
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



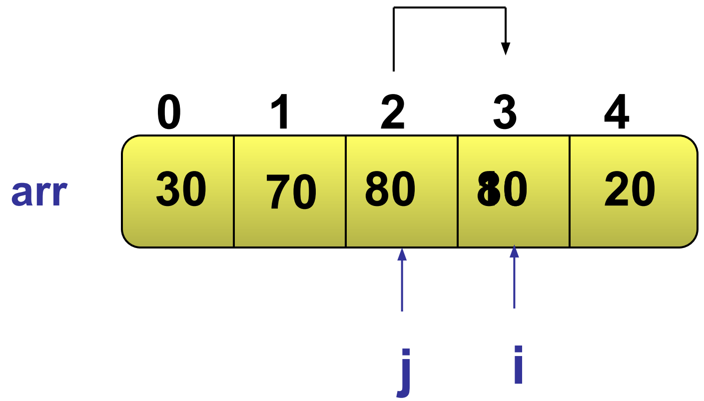
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



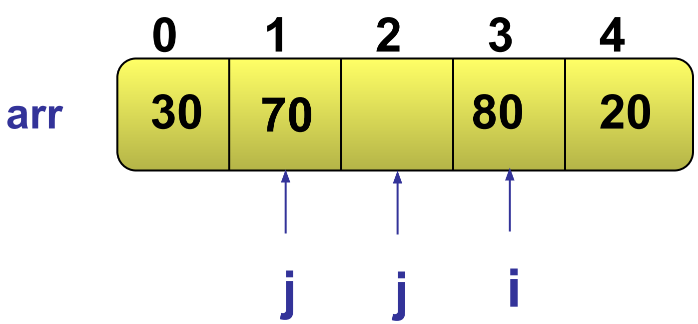
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



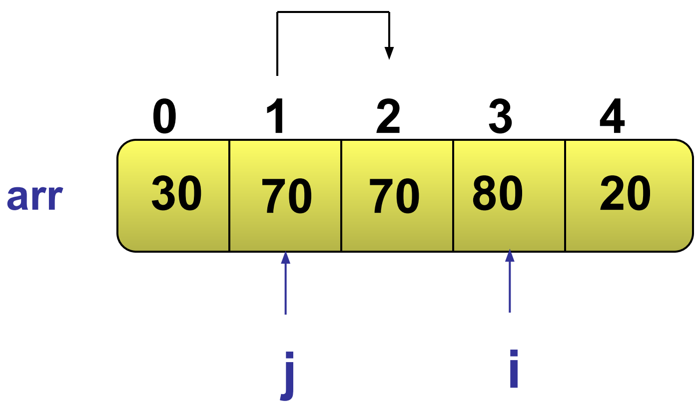
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



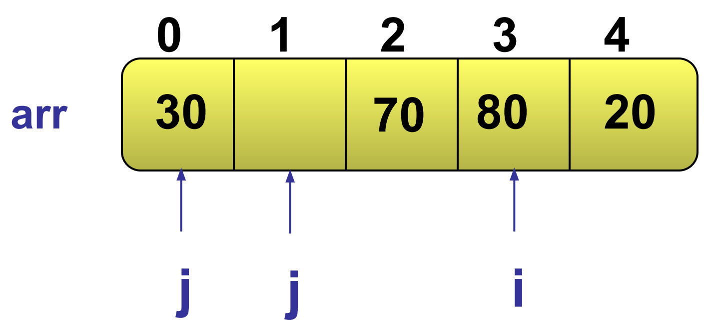
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



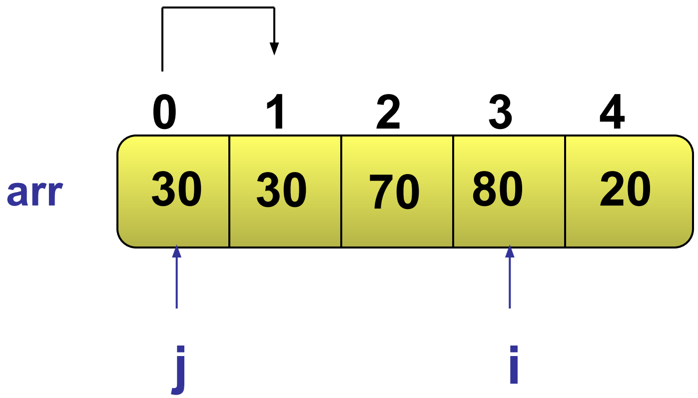
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10



1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

j = –1

	0	1	2	3	4
arr		**30**	**70**	**80**	**20**
	j			**i**	

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

temp = 10

j = –1

	0	1	2	3	4
arr	10	30	70	80	20
				i	

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 3

	0	1	2	3	4
arr	10	30	70	80	20

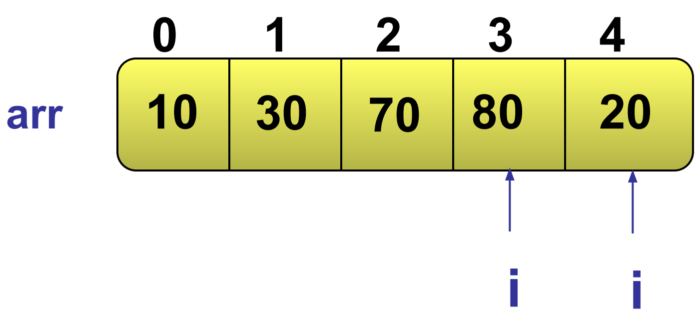
i

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4



1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

	0	1	2	3	4
arr	**10**	**30**	**70**	**80**	**20**

i

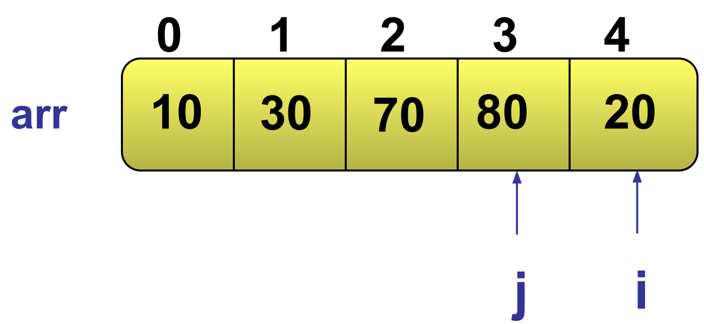
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



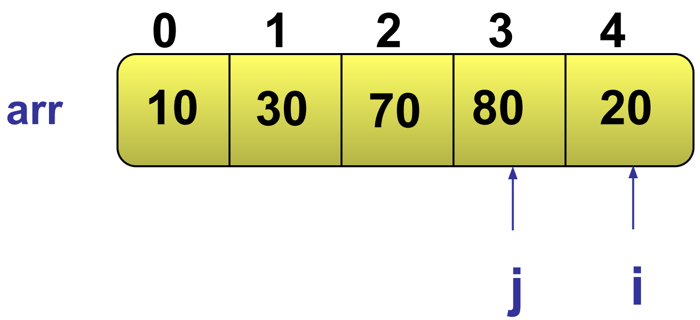
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



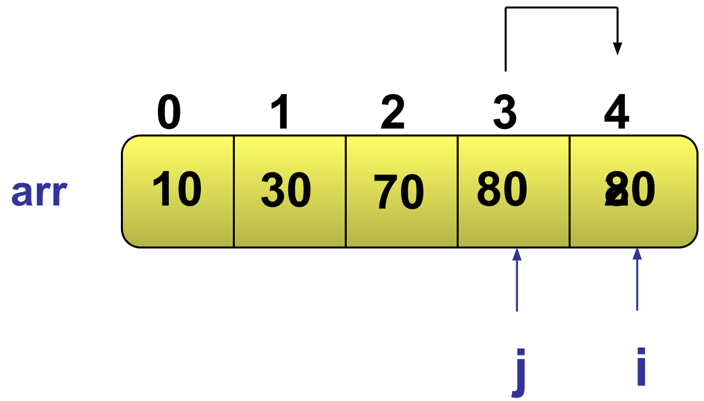
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



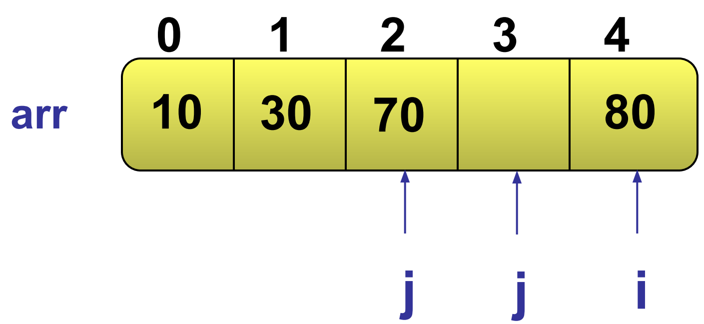
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



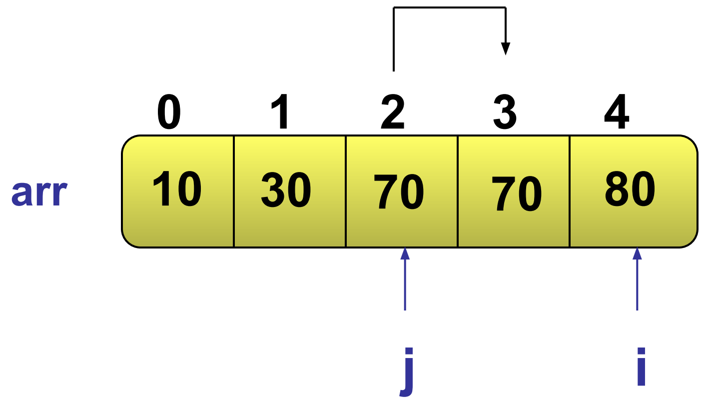
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



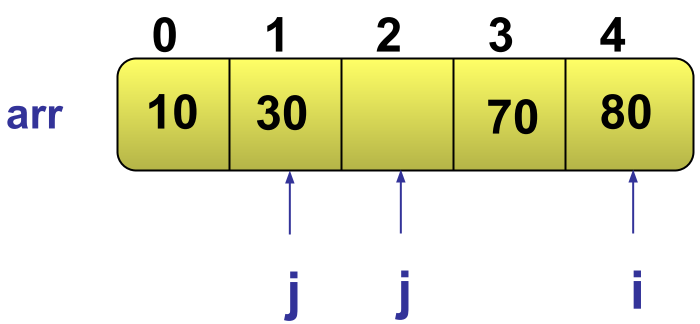
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



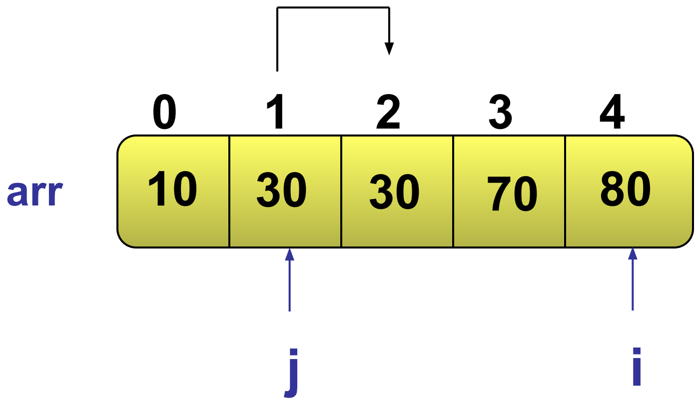
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



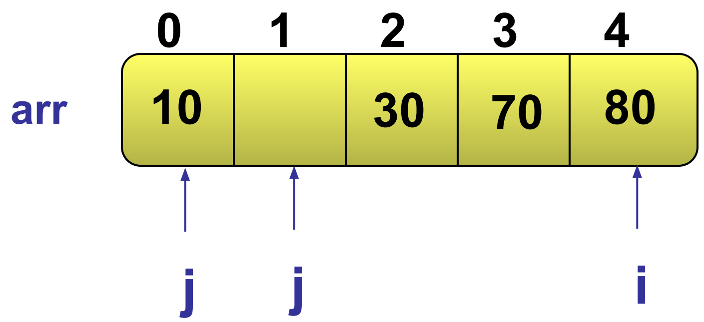
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20



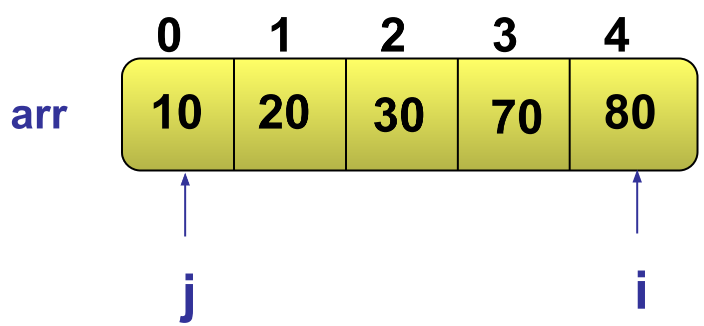
1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4

temp = 20

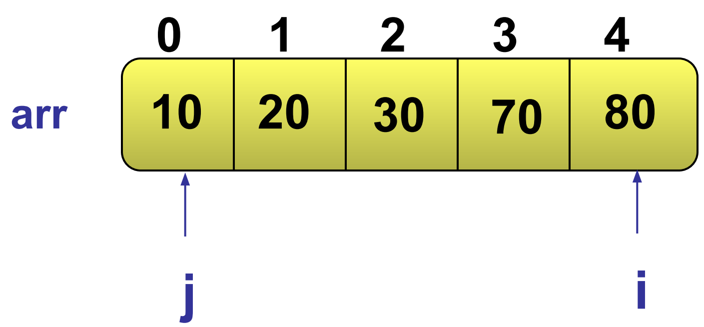


1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Implementing Insertion Sort Algorithm (Contd.)

n = 5

i = 4



The list is now sorted

1. Repeat steps 2, 3, 4, and 5 varying i from 1 to n – 1
2. Set temp = arr[i]
3. Set j = i – 1
4. Repeat until j becomes less than 0 or arr[j] becomes less than or equal to temp:
   a. Shift the value at index j to index j + 1
   b. Decrement j by 1
5. Store temp at index j + 1

Determining the Efficiency of Insertion Sort Algorithm

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

Sorting Data by Using Shell Sort

- Shell sort algorithm:
  - Insertion sort is an efficient algorithm only if the list is already partially sorted and results in an inefficient solution in an average case.
  - To overcome this limitation, a computer scientist, D.L. Shell proposed an improvement over the insertion sort algorithm.
  - The new algorithm was called shell sort after the name of its proposer.

Implementing Shell Sort Algorithm

- Shell sort algorithm:
  - Improves insertion sort by comparing the elements separated by a distance of several positions to form multiple sublists
  - Applies insertion sort on each sublist to move the elements towards their correct positions
  - Helps an element to take a bigger step towards its correct position, thereby reducing the number of comparisons

Implementing Shell Sort Algorithm (Contd.)

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

	0	1	2	3	4	5	6	7	8	9	10
arr	70	30	40	10	80	20	90	110	75	60	45

Implementing Shell Sort Algorithm (Contd.)

- To apply shell sort on this array, you need to:
  - Select the distance by which the elements in a group will be separated to form multiple sublists.
  - Apply insertion sort on each sublist to move the elements towards their correct positions.

	0	1	2	3	4	5	6	7	8	9	10
arr	70	30	40	10	80	20	90	110	75	60	45

Implementing Shell Sort Algorithm (Contd.)

Increment = 3

Pass =

List 1 =

0	3	6	9
70	10	90	60

List 2 =

1	4	7	10
30	80	110	45

List 3 =

2	5	8
40	20	75

arr

0	1	2	3	4	5	6	7	8	9	10
70	30	40	10	80	20	90	110	75	60	45

Implementing Shell Sort Algorithm (Contd.)

List 1 =

0	3	6	9
10	60	70	80

List 2 =

1	4	7	10
30	45	110	150

List 3 =

2	5	8
20	40	75

Apply insertion sort to sort the three lists

The lists are sorted

Implementing Shell Sort Algorithm (Contd.)

List 1 =

0	3	6	9
10	60	70	90

List 2 =

1	4	7	10
30	45	80	110

List 3 =

2	5	8
20	40	75

arr

0	1	2	3	4	5	6	7	8	9	10
10	30	20	60	45	40	70	80	75	90	110

Implementing Shell Sort Algorithm (Contd.)

List 1 =

0	2	4	6	8	10
10	20	45	70	75	110

List 2 =

1	3	5	7	9
30	60	40	80	90

arr

0	1	2	3	4	5	6	7	8	9	10
10	30	20	60	45	40	70	80	75	90	110

Implementing Shell Sort Algorithm (Contd.)

List 1 =

0	2	4	6	8	10
10	20	45	70	75	110

List 2 =

1	3	5	7	9
30	60	40	80	90

Apply insertion sort on each sublist

Implementing Shell Sort Algorithm (Contd.)

List 1 =

0	2	4	6	8	10
10	20	45	70	75	110

List 2 =

1	3	5	7	9
30	40	60	80	90

The lists are now sorted

Implementing Shell Sort Algorithm (Contd.)

List 1 =

0	2	4	6	8	10
10	20	45	70	75	110

List 2 =

1	3	5	7	9
30	40	60	80	90

arr

0	1	2	3	4	5	6	7	8	9	10
10	30	20	40	45	60	70	80	75	90	110

Implementing Shell Sort Algorithm (Contd.)

Increment = 1

Pass = 3

	0	1	2	3	4	5	6	7	8	9	10
arr	10	30	20	40	45	60	70	80	75	90	110

Apply insertion sort to sort the list

Implementing Shell Sort Algorithm (Contd.)

Increment = 1

Pass = 3

	0	1	2	3	4	5	6	7	8	9	10
arr	10	20	30	40	45	60	70	75	80	90	110

The list is now sorted

Just a minute

- Which of the following sorting algorithms compares the elements separated by a distance of several positions to sort the data? The options are:
  1. Insertion sort
  2. Selection sort
  3. Bubble sort
  4. Shell sort

- Answer:
  4. Shell sort

Summary

- In this session, you learned that:
  - Sorting is a process of arranging data in some pre-defined order of a key value. The order can be either ascending or descending.
  - There are various sorting algorithms that are used to sort data. Some of them are:
    - Bubble sort
    - Selection sort
    - Insertion sort
    - Shell sort
    - Merge sort
    - Quick sort
    - Heap sort

Summary (Contd.)

- To select an appropriate algorithm, you need to consider the following:
  - Execution time
  - Storage space
  - Programming effort
- Bubble sort and selection algorithms have a quadratic order of growth, and are therefore suitable for sorting small lists only.
- Insertion sort performs different number of comparisons depending on the initial ordering of elements. When the elements are already in the sorted order, insertion sort needs to make very few comparisons.
- Insertion sort is an efficient algorithm than bubble sort and selection sort if the list that needs to be sorted is nearly sorted.

Summary (Contd.)

- Shell sort improves insertion sort by comparing the elements separated by a distance of several positions. This helps an element to take a bigger step towards its correct position, thereby reducing the number of comparisons.