SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Sorting

Somaiya TRUST

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Outline

- Sorting- concept
- Sorting Terms
- Bubble sort
- Insertion sort
- Counting sort
- Sorting applications

Somaiya TRUST

Sorting

- **Sorting** is any process of arranging items systematically in a particular order
  - Sorting in ascending order :arrange n keys in such a way that $key_i < key_j$ for any i & j such that i<j
  - Sorting in descending order: arrange n keys in such a way that $key_i > key_j$ for any i & j any i & j such that i<j

Sorting Terms

- Stable sort
- Inplace sort
- Number of Passes

Bubble Sort

- *Compares adjacent* array elements
  - *Exchanges* their values if they are *out of order*
- Smaller values *bubble up* to the top of the array
  - Larger values sink to the bottom

FIGURE 10.1
One Pass of Bubble Sort

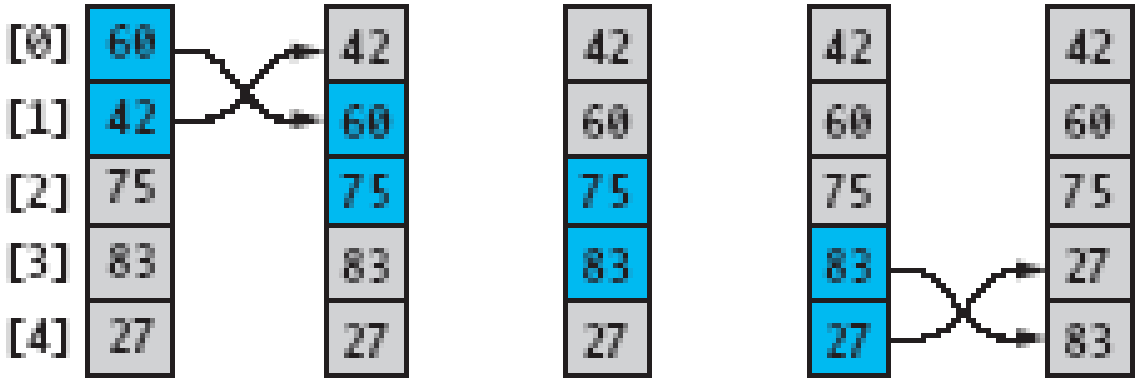
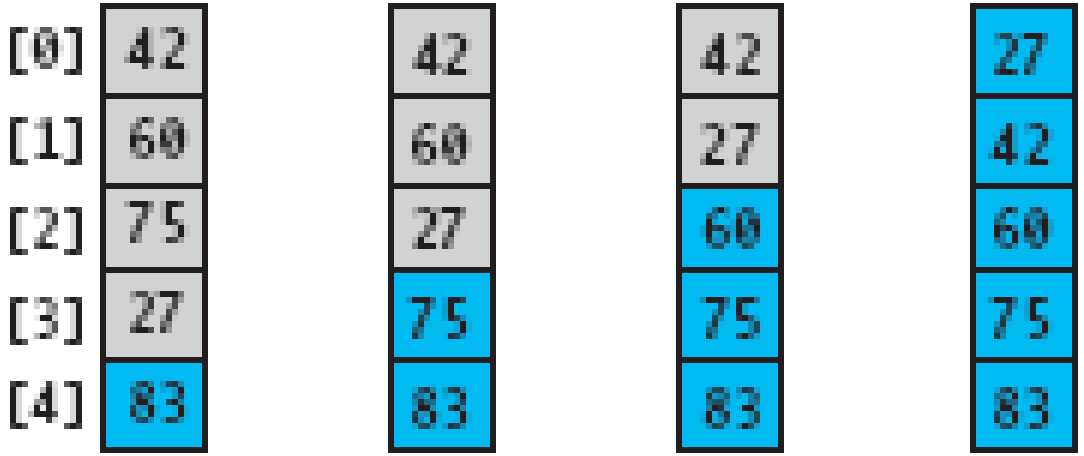


FIGURE 10.2
Array After Completion of Each Pass



Bubble Sort Algorithm

1. do
2. for each pair of adjacent array elements
3. if values are out of order
4. Exchange the values
5. while the array is not sorted

Bubble Sort Algorithm, Refined

1. do
2. Initialize **exchanges** to **false**
3. for each pair of adjacent array elements
4. if values are out of order
5. Exchange the values
6. Set **exchanges** to **true**
7. while **exchanges**

Analysis of Bubble Sort

- Excellent performance *in some cases*
  - But *very poor* performance in others!
- Works ***best*** when array is nearly sorted to begin with
- Worst case number of comparisons: $O(n^2)$
- Worst case number of exchanges: $O(n^2)$
- *Best case* occurs when the array is already sorted:
  - O(n) comparisons
  - O(1) exchanges (none actually)

```
bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)

    // Last i elements are already in place
    for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
```

Insertion Sort

- Based on technique of card players to arrange a hand
  - Player keeps cards picked up so far in *sorted order*
  - When the player picks up a new card

FIGURE 10.3
Picking Up a Hand of Cards

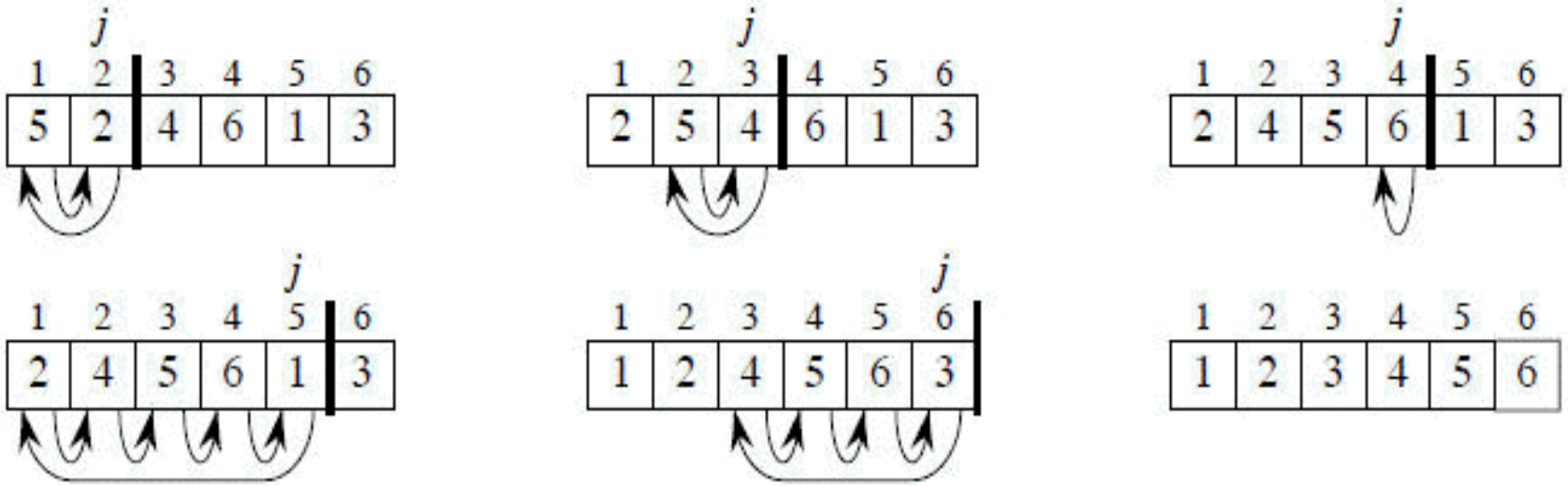
Insertion sort algorithm

INSERTION_SORT takes as parameters an array $A[1..n]$ and the length n of the array. The array A is sorted in place: the numbers are rearranged within the array, with at most a constant number outside the array at any time.

INSERTION_SORT (A)

1. **FOR** j ← 2 **TO** length[A]
2. **DO** key ← $A[j]$
3. {Put $A[j]$ into the sorted sequence $A[1 . . j - 1]$}
4. $i \leftarrow j - 1$
5. **WHILE** $i > 0$ and $A[i] >$ key
6. **DO** $A[i+1] \leftarrow A[i]$
7. $i \leftarrow i - 1$
8. $A[i + 1] \leftarrow$ key

Insertion sort algorithm



Stability :

Since multiple keys with the same value are placed in the sorted array in the same order that they appear in the input array, Insertion sort is stable.

Courtesy: Analysis of Algorithm, Coreman

Counting sort

- sorting is based on keys between a specific range.
- It works by counting the number of objects having distinct key values
- Followed by computation of position of each object in the output sequence.

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Counting sort

- Initialize count array of the size of input range
- Update the count array to store the count of each unique key.
- Further update the count array with cumulative additions of previous counts
- Shift the count array to right by one position; no circular shift
- Initialize sort array of the size of input sequence
- Update sort array by entering keys from input array at location from count array and increment the count by 1

Somaiya TRUST

Counting sort example

- i/p : 2 3 1 2 4 5 2 1 5 4
- N= 10, range: 1:5

Initialize count array of the size of input range

count array	0	1	2	3	4	5
	0	0	0	0	0	0

Update the count array to store the count of each unique key.

count array	0	1	2	3	4	5
	0	2	3	1	2	2

Further update the count array with cumulative additions of previous counts

count array	0	1	2	3	4	5
	0	2	5	6	8	10

Shift the count array to right by one position; no circular shift

count array	0	1	2	3	4	5
	0	0	2	5	6	8

Initialize sort array of the size of input sequence

Sort Array	0	1	2	3	4	5	6	7	8	9

Update sort array by entering keys from input array at location from count array and increment the count by 1

i/p	2	3	1	2	4	5	2	1	5	4
count array	0	1	2	3	4	5				
	0	0	2	5	6	8				
Output Sorted y	0	1	2	3	4	5	6	7	8	9
	1	1	2	2	2	3	4	4	5	5

Shell sort introduction

- **shell sort**: orders a list of values by comparing elements that are separated by a gap of >1 indexes
  - a generalization of insertion sort
  - invented by computer scientist Donald Shell in 1959

- based on some observations about insertion sort:
  - insertion sort runs fast if the input is almost sorted
  - insertion sort's weakness is that it swaps each element just one step at a time, taking many swaps to get the element into its correct position

Shell sort

Shell Sort compares elements separated by a **gap** of several positions, allowing elements to make **larger jumps** towards their final position.

- **Multiple Passes:** Elements are sorted using progressively **smaller gap sizes**.
- **Final Step:** The last step is a **plain insertion sort**, but by this stage, the elements are **almost sorted**, leading to improved performance.

Shell sort vs insetion sort and bubble sort

Scenario:

- When the **smallest element** is at the opposite end of the array, **bubble sort** or **insertion sort** will take $O(n^2)$ time.
    - Requires approximately **n comparisons and exchanges** to move the smallest element to its correct position.

Shell Sort Advantage:

- Uses **large step sizes** in early passes.
    - Allows small elements to **move quickly** toward their final position with **fewer comparisons and exchanges**.

Technique

To visualize the way in which shell sort works, perform the following steps:

- *Step 1:* Arrange the elements of the array in the form of a table and sort the columns (using insertion sort).
- *Step 2:* Repeat Step 1, each time with smaller number of longer columns in such a way that at the end, there is only one column of data to be sorted.

Example 14.8 Sort the elements given below using shell sort.

63, 19, 7, 90, 81, 36, 54, 45, 72, 27, 22, 9, 41, 59, 33

Solution

Arrange the elements of the array in the form of a table and sort the columns.

63	19	7	90	81	36	54	45
72	27	22	9	41	59	33	

Result:

63	19	7	9	41	36	33	45
72	27	22	90	81	59	54	

The elements of the array can be given as:

63, 19, 7, 9, 41, 36, 33, 45, 72, 27, 22, 90, 81, 59, 54

Repeat Step 1 with smaller number of long columns.

63	19	7	9	41
36	33	45	72	27
22	90	81	59	54

Result:

22	19	7	9	27
36	33	45	59	41
63	90	81	72	54

The elements of the array can be given as:

22, 19, 7, 9, 27, 36, 33, 45, 59, 41, 63, 90, 81, 72, 54

Repeat Step 1 with smaller number of long columns.

			Result:		
22	19	7	9	19	7
9	27	36	22	27	36
33	45	59	33	45	54
41	63	90	41	63	59
81	72	54	81	72	90

The elements of the array can be given as:

9, 19, 7, 22, 27, 36, 33, 45, 54, 41, 63, 59, 81, 72, 90

Finally, arrange the elements of the array in a single column and sort the column.

	Result:
9	7
19	9
7	19
22	22
27	27
36	33
33	36
45	41
54	45
41	54
63	59
59	63
81	72
72	81
90	90

Finally, the elements of the array can be given as:

7, 9, 19, 22, 27, 33, 36, 41, 45, 54, 59, 63, 72, 81, 90

The algorithm to sort an array of elements using shell sort is shown in Fig. 14.13. In the algorithm, we sort the elements of the array `Arr` in multiple passes. In each pass, we reduce the `gap_size` (visualize it as the number of columns) by a factor of half as done in Step 4. In each iteration of the `for` loop in Step 5, we compare the values of the array and interchange them if we have a larger value preceding the smaller one.

```
Shell_Sort(Arr, n)

Step 1: SET FLAG = 1, GAP_SIZE = N
Step 2: Repeat Steps 3 to 6 while FLAG = 1 OR GAP_SIZE > 1
Step 3:     SET FLAG = 0
Step 4:     SET GAP_SIZE = (GAP_SIZE + 1) / 2
Step 5:     Repeat Step 6 for I = 0 to I < (N - GAP_SIZE)
Step 6:           IF Arr[I + GAP_SIZE] > Arr[I]
                        SWAP Arr[I + GAP_SIZE], Arr[I]
                        SET FLAG = 0
Step 7: END
```

Figure 14.13 Algorithm for shell sort

Analysis of sorting algorithms

Sr. no.	Algorithm	Stable?	Inplace?	#passes?
1	Bubble			
2	Insertion			
3	Counting			
4	Shell Sort			

Thank you