Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES
(COMMON TO CSE & IT)

UNIT No. 5

SEARCHING,SORTING AND HASHING TECHNIQUES

5.2 Sorting - Bubble sort - Selection sort

Version: 1.XX

4 QUALITY EDUCATION

NBA NAAC (A+) nirf

\

SORTING

A sorting algorithm is an algorithm made up of a series of instructions that takes an array as input, performs specified operations on the array, sometimes called a list, and outputs a sorted array.

EXAMPLES OF SORTING IN REAL-LIFE SCENARIOS:

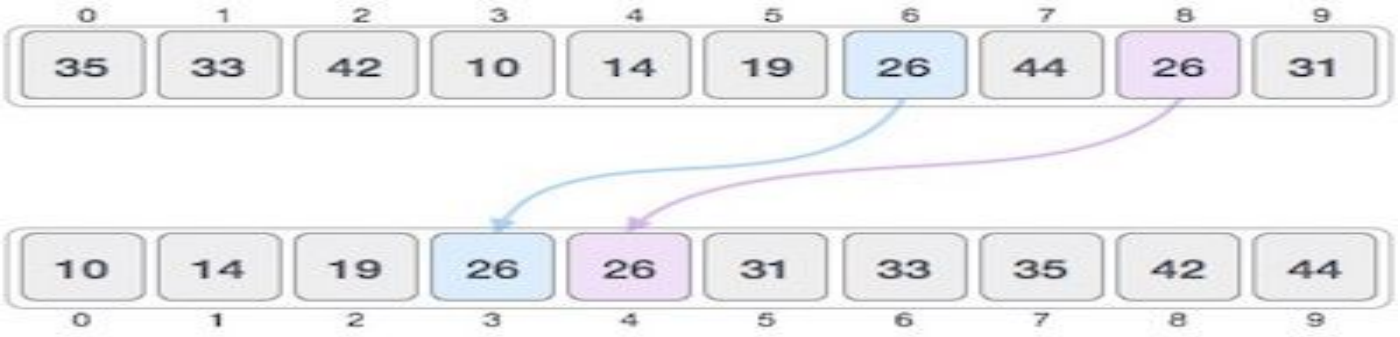
- Telephone Directory
- Dictionary

TYPES:

- stable sorting.
- NOT stable sorting.

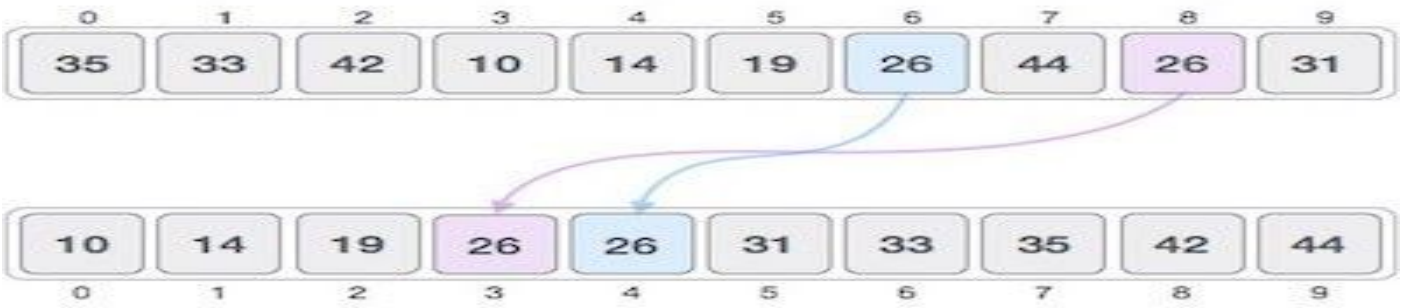
STABLE SORTING:

If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called **stable sorting**.

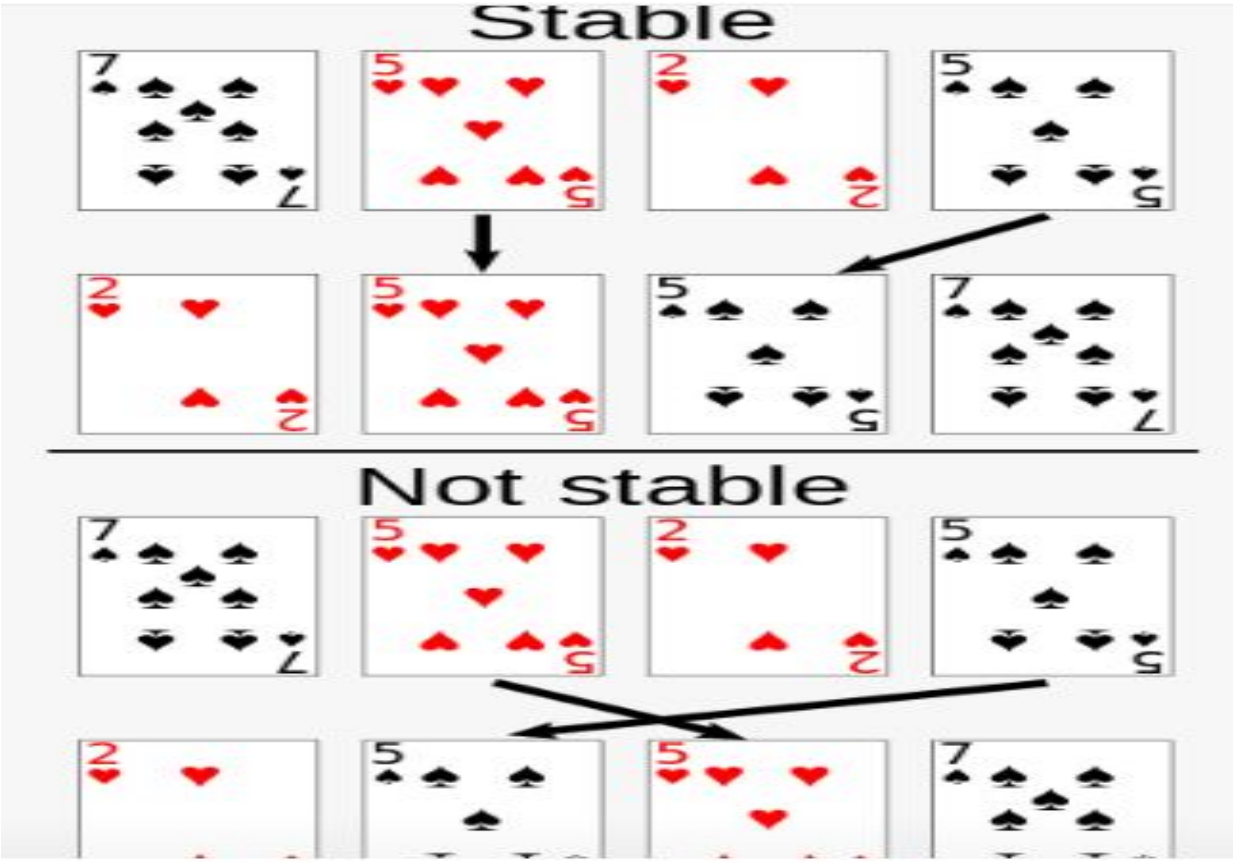


NOT STABLE SORTING:

If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called **unstable sorting**.



EXAMPLE FOR STABLE & UNSTABLE:



TYPES OF SORTING:

i. Internal sorting

ii. External sorting

Internal Sorting:

If all the data that is to be sorted can be adjusted at a time in the main memory, the internal sorting method is being performed.

External Sorting:

When the data that is to be sorted cannot be accommodated in the memory at the same time and some has to be kept in auxiliary memory such as hard disk, floppy disk, magnetic tapes etc, then external sorting methods are performed.

Complexity of Sorting Algorithms

The complexity of sorting algorithm calculates the running time of a function in which 'n' number of items are to be sorted. The most noteworthy of these considerations are:

- The length of time spent by the programmer in programming a specific sorting program
- Amount of machine time necessary for running the program
- The amount of memory necessary for running the program
- The Efficiency of Sorting Techniques

The Efficiency of Sorting Techniques

- To get the amount of time required to sort an array of 'n' elements by a particular method, the normal approach is to analyze the method to find the number of comparisons (or exchanges) required by it.
- Most of the sorting techniques are data sensitive, and so the metrics for them depends on the order in which they appear in an input array.
- Various sorting techniques are analysed, that are

  Best case

  Worst case

  Average case

Types of sorting Techniques:

- Bubble Sort
- Selection Sort
- Merge Sort
- Insertion Sort
- Quick Sort
- Heap Sort

BUBBLE SORT:

- Bubble Sort Algorithm is used to arrange N elements in ascending order, and for that,
- Begin with 0^{th} element and compare it with the first element.
- If the 0^{th} element is found greater than the 1^{st} element, then the swapping operation will be performed, i.e., the two values will get interchanged. In this way, all the elements of the array get compared.

List is unsorted. We are going to sort this in ASCENDING order using Bubble Sort

5	4	3	2	1

First Pass of the first For loop.

1=0, a[j]=5, a[j+1] =4 (Second For loop starts working)

5	4	3	2	1

Here a[j]=5 and a[j+1]=4 are compared. Their positions are then interchanged.

Second For loop gets iterated. Now j=1 a[j]=5 and a[j+1]=3

4	5	3	2	1

j=2 a[j]=5 and a[j+1]=2

4	3	5	2	1

j=3 a[j]=5 and a[j+1]=1

4	3	2	5	1

Largest element of the list is moved to last position after first pass.

4	3	2	1	5

Example 02:

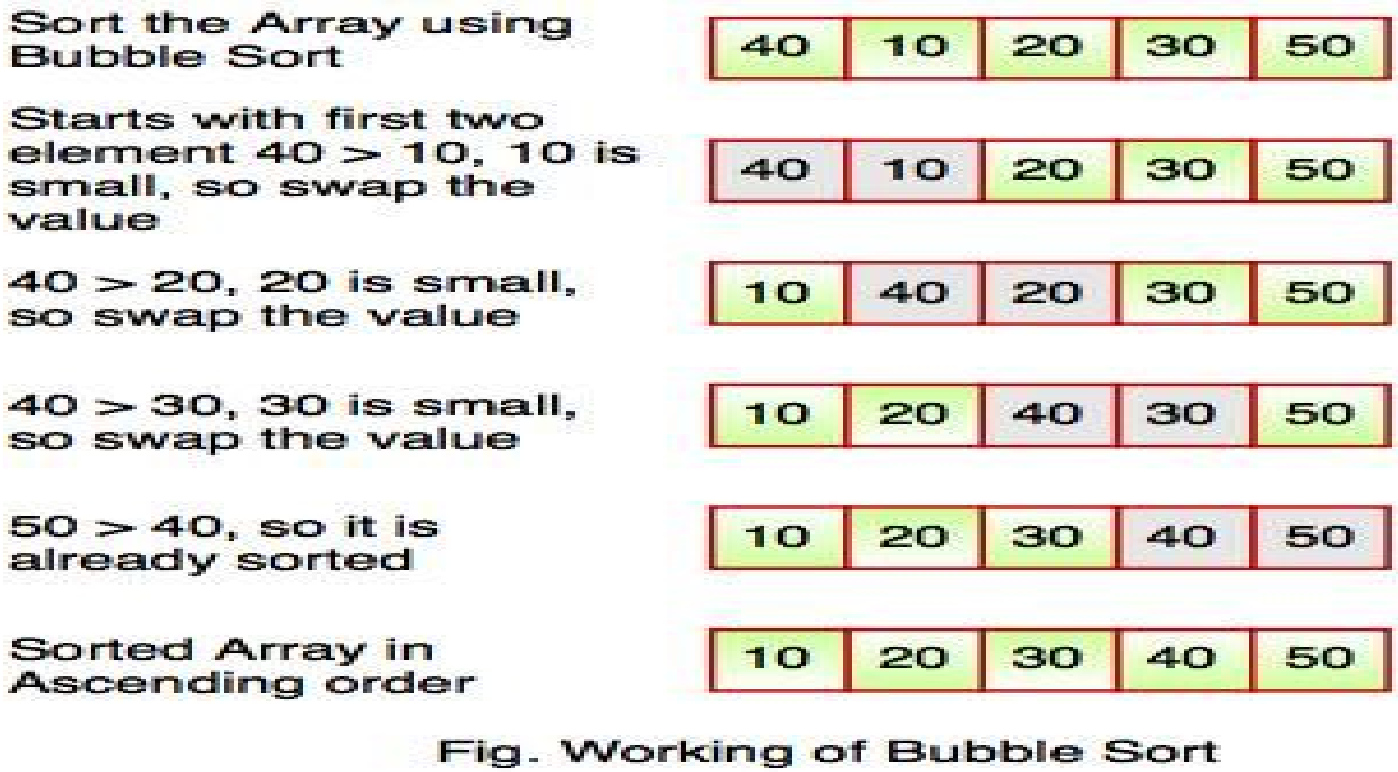


Fig. Working of Bubble Sort

- The above diagram represents how bubble sort actually works. This sort takes O(n2) time. It starts with the first two elements and sorts them in ascending order.
- Bubble sort starts with first two elements. It compares the element to check which one is greater.
- In the above diagram, element 40 is greater than 10, so these values must be swapped. This operation continues until the array is sorted in ascending order.

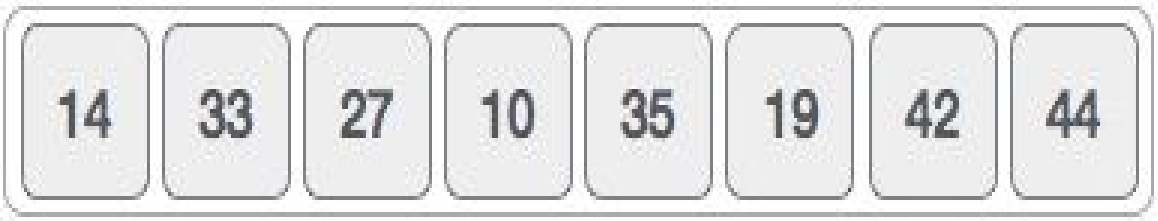
PROGRAM

```
void bubble_sort(long list[], long n)
{
  long c, d, t;
  for (c = 0 ; c < ( n - 1 ); c++)
  {        for (d = 0 ; d < n - c - 1; d++)
           {
              if (list[d] > list[d+1])
                 {        /* Swapping */
                          t        = list[d];
                          list[d]   = list[d+1];
                          list[d+1] = t;
                 } }
  } }
```

SELECTION SORT:

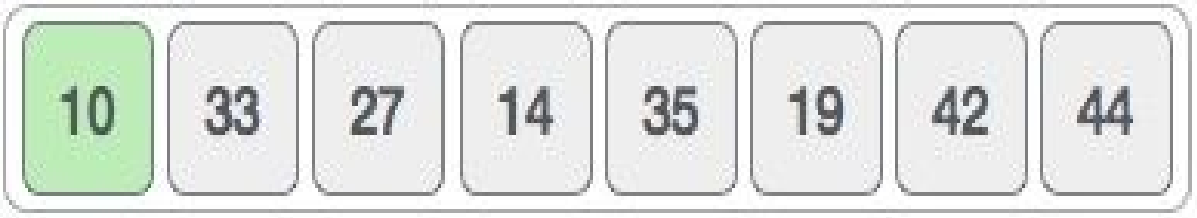
- Selection sort is a simple sorting algorithm.
- This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

- Consider the following depicted array as an example.



For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

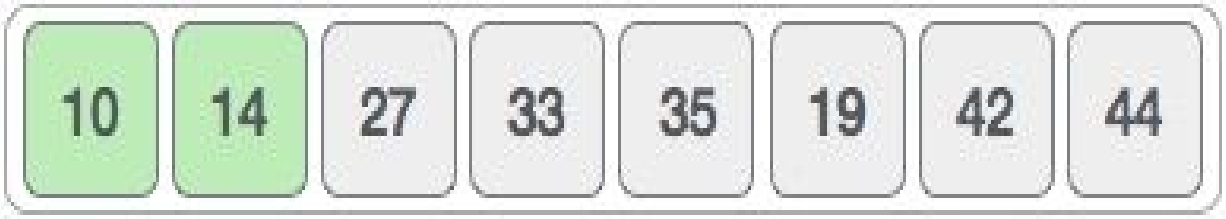
So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.



For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner.

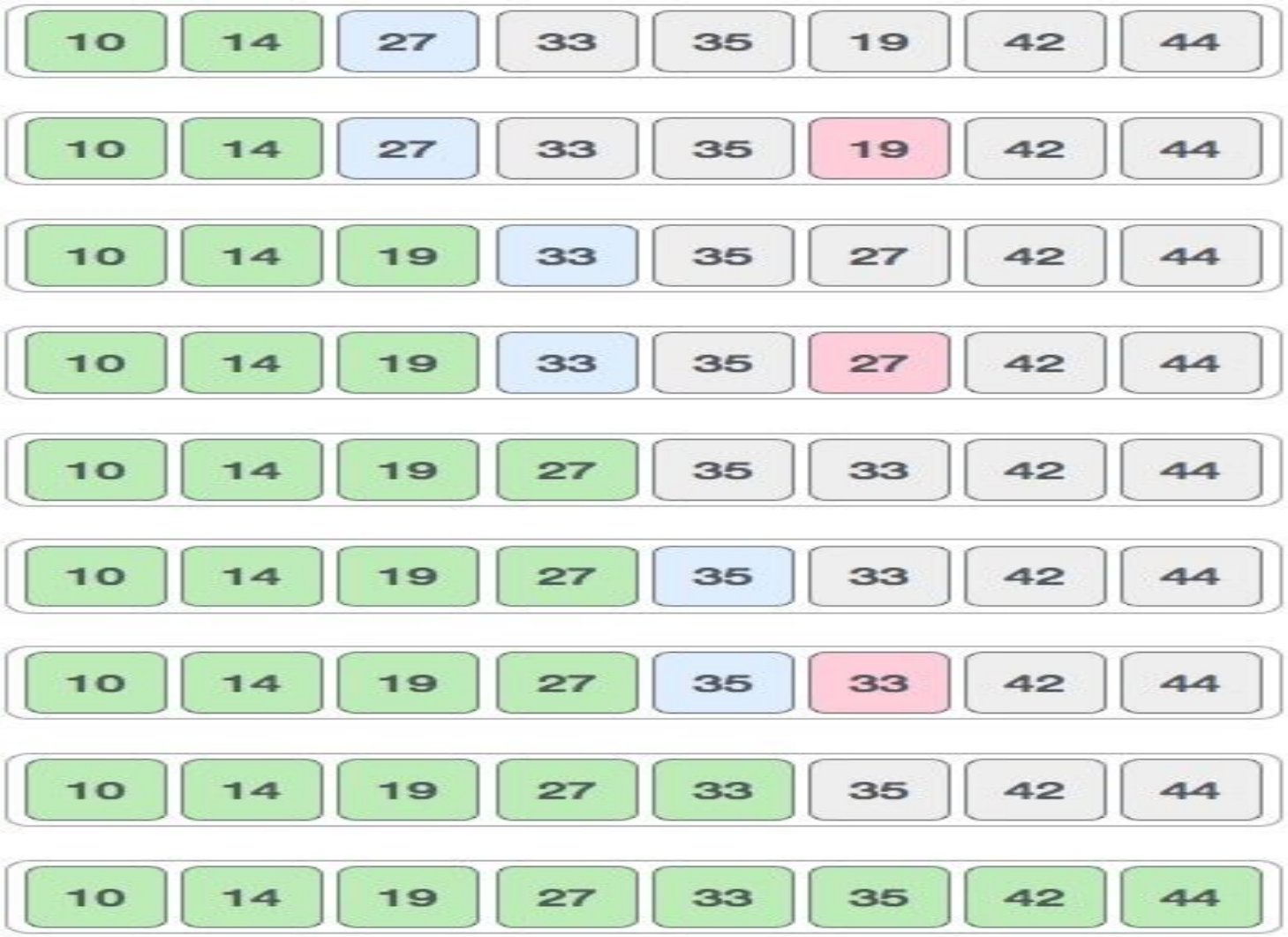
We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two least values are positioned at the beginning in a sorted manner.



The same process is applied to the rest of the items in the array.

Following is a pictorial depiction of the entire sorting process −



Example 02

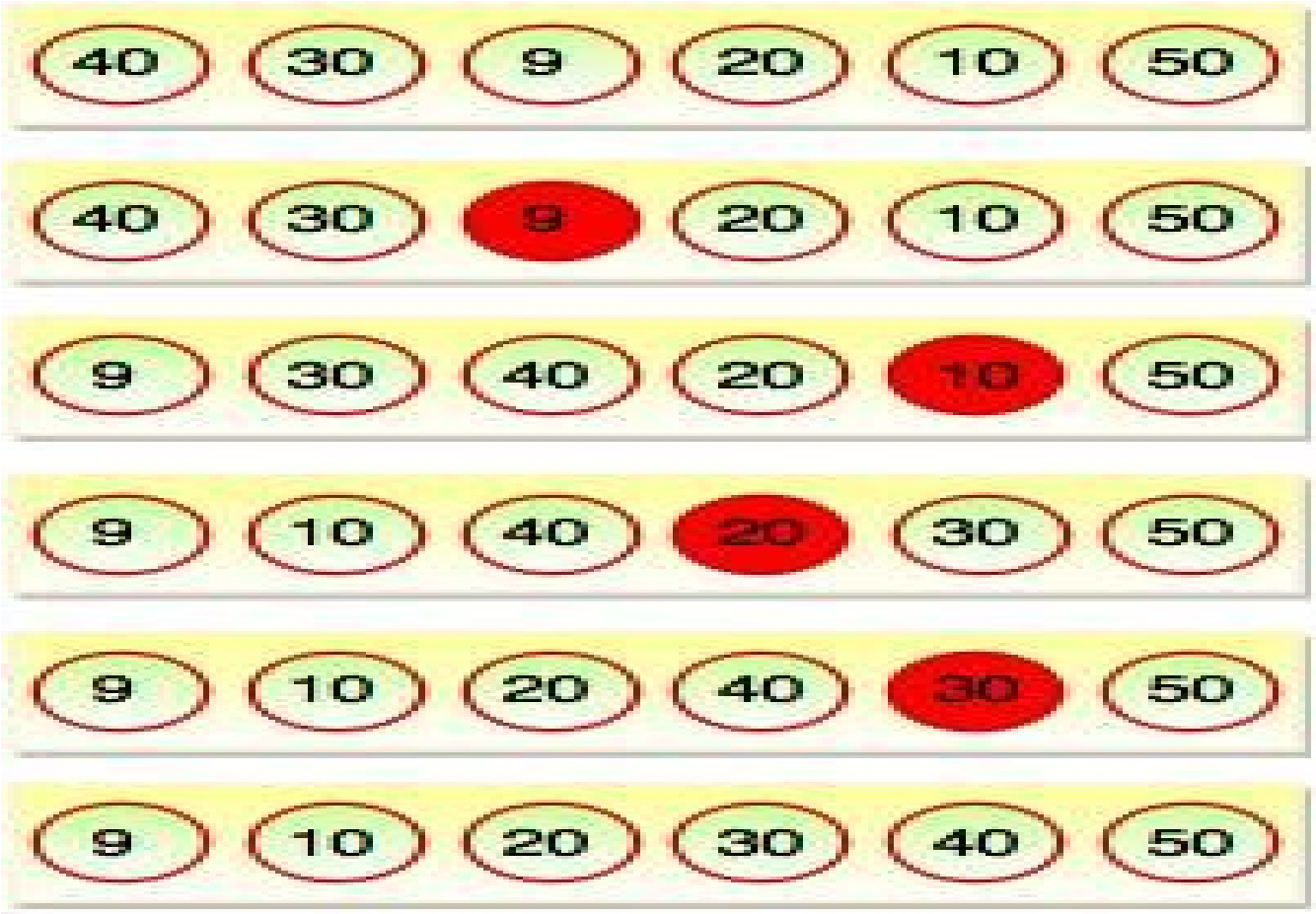


Fig. Working of Selection Sort

In the above diagram, the smallest element is found in first pass that is 9 and it is placed at the first position. In second pass, smallest element is searched from the rest of the element excluding first element. Selection sort keeps doing this, until the array is sorted.

Program for Selection Sort

```
#include <stdio.h>
int main()
{
   int array[100], n, c, d, position, swap;
   printf("Enter number of elements\n");
   scanf("%d", &n);
   printf("Enter %d integers\n", n);
   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);
   for ( c = 0 ; c < ( n - 1 ) ; c++ )
   {  position = c;
      for ( d = c + 1 ; d < n ; d++ )
      {     if ( array[position] > array[d] )
            position = d;
      }
      if ( position != c )
      {  swap = array[c];
         array[c] = array[position];
         array[position] = swap;
      }
  }
   printf("Sorted list in ascending order:\n");
   for ( c = 0 ; c < n ; c++ )
      printf("%d\n", array[c]);
   return 0;
}
```

```
Enter number of elements
5
Enter 5 integers
60
10
40
50
30
Sorted list in ascending order:
10
30
40
50
60
```