BUBBLE SORT

```
BUBBLE_SORT (A,N)

Step 1: Repeat Step 2 For I = 0 to N-1
step2:      Repeat For J = 0 to N-I
step3:              IF A[J] > A[J+1]
                    SWAP A[J] and A[J+1]
            [END OF INNER LOOP]
        [END OF OUTER LOOP]
Step 4: EXIT.
```

Complexity of Bubble Sort

Complexity of Any Sorting Algo. depends upon the number of comparisons. Therefore to compute Bubble Sort complexity we need to calculate the total number of comparisons.

It can be given as:-

$$f(n) = (n-1) + (n-2) + (n-3) + \cdots + 3 + 2 + 1$$

$$f(n) = n(n-1)/2$$

$$f(n) = n^2/2 + O(n) = O(n^2).$$

Therefore the complexity of Bubble Sort Algorithm is $O(n^2)$.

Bubble Sort Optimization

Even if the array is already sorted. In this situation No SWAPPING is done

→ but we still have to continue with all n-1 passes.

→ In the Best case, when the array is already sorted, the Optimized Bubble Sort will take O(n) times.

Code for Optimized Bubble Sort

```
Void bubble_sort (int *arr, int n)
{
 int i, j, temp, flag=0;
 for(i=0; i<n; i++)
  {
   for(j=0; j<n-i-1; j++)
    {
     if (arr[j] > arr[j+1])
      {
       flag = 1;
       temp = arr[j+1];
       arr[j+1] = arr[j];
       arr[j] = temp;
      }
    }
   if (flag==0) // array is Sorted.
     return;
  }
}
```

About Bubble Sort :→

→ Sorts the array elements by repeatedly moving the largest elements to the highest index position of the array segment (in case of arranging elements in ascending order).

→ In Bubble Sorting, consecutive adjacent pairs of elements in the array are compared with each other.

→ This procedure of sorting is called Bubble Sorting because elements 'Bubble' to the top of the list.

Note that at the end of the first pass, the largest element in the list will be placed at its proper position. (i.e at the end of the list).

NOTE:-

If the elements are to be sorted in Descending Order, then in first pass the Smallest element is moved to the highest index of the array.

```
#include <stdio.h>
#include <conio.h>
int main()
 {
  int i, n, temp, j, arr[10];
  clrscr();
  printf ("Enter the number of elements in the array :");
  scanf ("%d", &n);
  printf ("\n Enter the elements : ");
  for (i=0; i<n; i++)
    {
      scanf ("%d", &arr[i]);
    }
```

```
for (i=0; i<n; i++)
 {
   for(j=0; j<n-i-1; j++)
    {
        if (arr[j] > arr[j+1])
         {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
    }
 }
printf("\n The array sorted in ascending order is : \n");
 for(i=0; i<n; i++)
        printf("%d \t", arr[i]);
 getch();
 return 0;
}
```

SORTING BY EXCHANGE

→ Based on the principle of "sorting by exchange".

→ The Basic concept in this technique is to interchange (exchange) pairs of elements that are out of order until no such pair exists.

→ Following Sorting Technique based on this principle and included in our discussion in the following subsections.

- SORTING BY EXCHANGE
  - Bubble Sort
  - Shell Sort
  - Quick Sort

(A). Bubble Sort Algorithm.

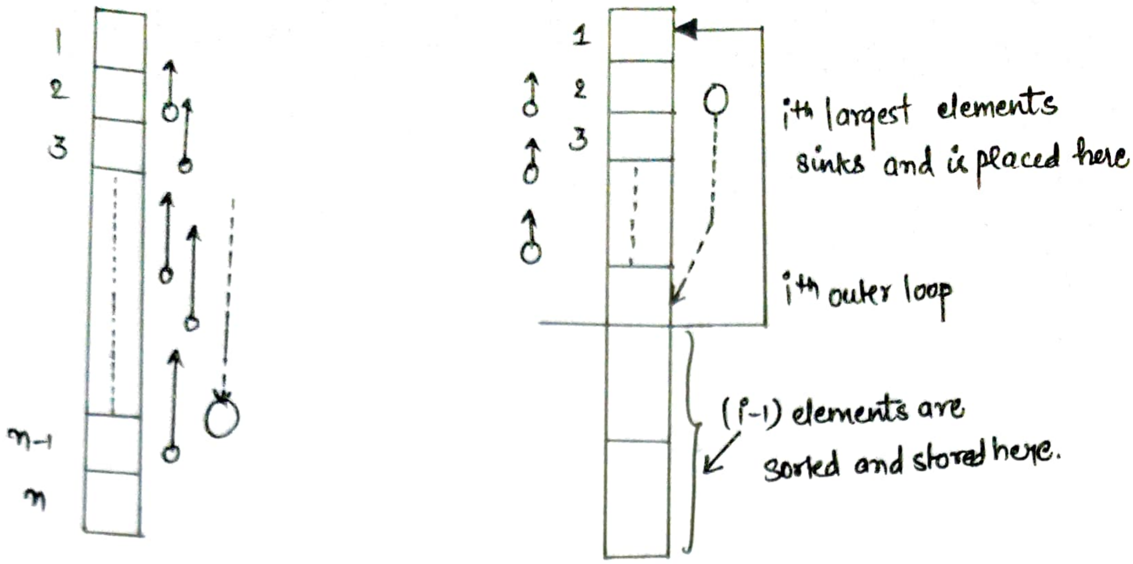
The Bubble Sort derive its name from the fact that the smaller data items "bubble up" to the top of the list.

This method is also alternatively termed "SINKING SORT" because the larger elements "sink down" to the Bottom.

For example → in the first pass of the outer loop, the largest key in the list moves to the end. Thus first pass assures that the list will have the largest element at the last location.

The same operation is repeated for the remaining (n-1) element other than this largest element, so that the second largest element is placed in its position.

Repeating the process for a total of (n-1) passes will eventually guarantee that all items are placed in sorted order.



The principle of Bubble Sort

ALGORITHM BUBBLE SORT

Input: An array A[1, 2, ... n], where n is the no. of element.

Output: An array A with all elements in sorted order.

Remark: Sort the elements in ascending order.

Steps:-

1. For i=1 to n-1 do
2. → For j=1 to n-i do
3. → If (A[j] > A[j+1]) then
4. → Swap (A[j], A[j+1])
5. → EndIf
6. → End For
7. → End for
8. Stop

I/p list	Pass1	PASS2	Pass3	Pass4	Pass5	Pass6	Pass7
55	55	55	33	22	22	22	22
77	77	33	22	33	33	33	33
99	33	22	55	55	44	44	44
33	22	77	66	44	55	55	55
22	88	66	44	66	66	66	66
88	66	44	77	77	77	77	77
66	44	88	88	88	88	88	88
44	99	99	99	99	99	99	99

22
33
44
55
66
77
88
99

O/p list.

Analysis of the algorithm Bubble Sort

- Illustration of Bubble Sort technique, it is evident that the bubble sort algorithm, it is evident that the Bubble Sort algorithm does not require any additional storage space other than the list itself.

- Hence, the algorithm Bubble Sort is an in place sorting method. Therefore, if there are n elements in the input list, then additional storage space requirement is

$$\boxed{S(n) = 0}$$

- Storage space requirement is irrespective of the ordering of the elements in the input list.

  The no. of comparisons and the no. of movements, on the other hand, depend on the ordering of the elements.

Case 1. The input list is already in sorted order.

(a). No. of Comparison

- Let us consider the case of the i^{th} pass, when $1 \leq i \leq n-1$.
- In the i^{th} pass, $i-1$ elements are already sorted. and the inner loop iterates comparing with 1 to $n-(i-1)-1$ elements.
- Total no. of key comparisons in the i^{th} pass is equal to $n-i$.
- Total no. of comparisons is

$$C(n) = \sum_{i=1}^{n-1} (n-i) = (n-1) + (n-2) + \cdots + 2 + 1$$

$$C(n) = \frac{n(n-1)}{2}$$

(b). No. of movement

In this case, no data swap operation takes place in any pass.

Hence, the number of movements $M(n)$ is:

$$\boxed{M(n) = 0}$$

Case 2 The input list is sorted but in reverse order

(a) No. of comparisons

$$C(n) = \frac{n(n-1)}{2}$$

(b) No. of movements

The number of movements is same as the no. of comparisons.
That is, in ith iteration of the algorithm, the number of movements is $n-i$.

Therefore, the total number of key movement is

$$M(n) = \sum_{i=1}^{n-1} (n-i) = (n-1) + (n-2) + \cdots + 2 + 1$$

$$M(n) = \frac{n(n-1)}{2}$$

Case 3. Elements in the input list are in Random Order.

(a). No. of Comparisons

$$C(n) = \frac{n(n-1)}{2}$$

(b). **No. of movements**

- To calculate the no. of movt. in this case, let us consider the ith pass of the algorithm.
- We know that in the ith $(1 \le i \le n-1)$ pass, $(i-1)$ elements are present in the sorted part (bottom part) and $(n-i+1)$ elements are in unsorted part (top) in the array.
- Let P_j be the probability that the largest element in the unsorted part is in the j^{th} $(1 \le j \le n-i+1)$ location.
- If the largest element is in the j^{th} location, then the expect number of swap operation is

$P_j \times (\overline{n-i+1}-j)$. Therefore, the average no. of swaps in the ith pass is given by

$$= \sum_{j=1}^{n-i+1} (\overline{n-i+1}-j) \cdot P_j$$

To simplify the above calculation, assume that the largest element is equally probable at any place and hence,

$$P_1 = P_2 = P_{n-i+1} = \frac{1}{n-i+1}$$

with the assumption, the average no. of swaps in the ith pass stand as

$$= \sum_{j=1}^{n-i+1} \frac{1}{n-i+1} \cdot (\overline{n-i+1}-j)$$

$$= \frac{1}{n-i+1}\left[(n-i+1)(n-i+1) - (1+2+3+\cdots+(n-i+1))\right]$$

$$= (n-i+1) - \frac{1}{n-i+1} \cdot \frac{(n-i+1)(n-i+2)}{2}$$

$$= \frac{n-i}{2}$$

Consider all the passes b/w $i=1$ to $n-1$, the average no. of movement, $M(n)$ is given by

$$M(n) = \sum_{j=1}^{n-1} \frac{n-i}{2}$$

$$= \frac{n(n-1)}{4}$$

Analysis of the Algo. Bubble Sort

Case	Comparisons	Movement	Memory	Remark
Case 1	$C(n) = \frac{n(n-1)}{2}$	$M(n)=0$	$S(n)=0$	I/p list in sorted order
Case 2	$C(n) = \frac{n(n-1)}{2}$	$M(n) = \frac{n(n-1)}{2}$	$S(n)=0$	I/p list in reverse order
Case 3	$C(n) = \frac{n(n-1)}{2}$	$M(n) = \frac{n(n-1)}{4}$	$S(n)=0$	i/p list in random order

Time complexity $= t_1 \cdot C(n) + t_2 \cdot M(n)$

TIME COMPLEXITY

RunTime, T(n)	Complexity	Remark
$T(n) = c\frac{n(n-1)}{2}$	$T(n) = O(n^2)$	Best Case
$T(n) = c\,n(n-1)$	$T(n) = O(n^2)$	Worst case
$T(n) = c\frac{3}{4}n(n-1)$	$T(n) = O(n^2)$	Average case.