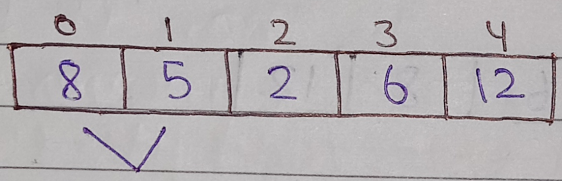
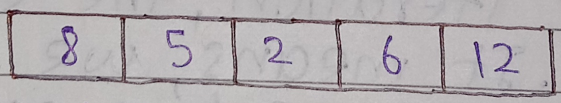
Bubble Sort

Bubble sort is a sorting algorithm which iterates through a given array of eleme-nts & compares each pair of adjacent elements one after the other.
In any of the adjacent pairs, if the first element is greater than the second element, than it swaps the elements & if not, then it moves on to the next pair of elements.

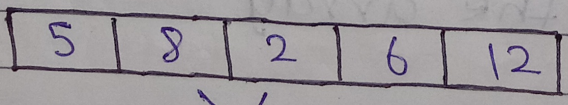


compares the adjacent elements. Here, 8 > 5, so they get swapped.

Example,



start with the first element & compare it with the adjacent element. 8 > 5, so swap these elements.



now we compare 2nd & 3rd element. 8 > 2, so swap again.

5	2	8	6	12

now we compare 3rd & 4th element. 8 > 6, so swap again.

5	2	6	8	12

now compare 4th & 5th element. 8 < 12, no need to swap.

5	2	6	8	12

this is the result after 1st iteration.

After the 1st iteration, the elements are not sorted. It means, we have to keep repeating the set of actions again & again until the entire array of elements are sorted.

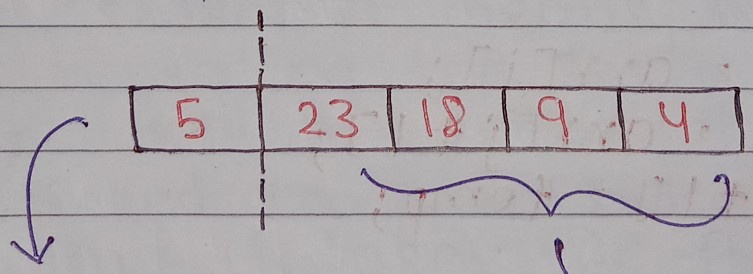
Generally, it takes n-1 iterations in order to sort a given array using Bubble sort algorithm, where n is the number of elements in the array.

Bubble sort :-

```
#include <stdio.h>
int main()
{
  int arr[] = {8,5,2,6,12};
  int n = 5, temp;
  printf("before sorting ");
  for(int i=0; i<n; i++)
  {
    printf("%d ", arr[i]);
  }

  for(int i=0; i<n; i++)
  {
    for(int j=0; j<n; j++)
    {
      if(arr[j] > arr[j+1])
      {
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
      }
    }
  }

  printf("after sorting ");
  for(int i=0; i<n; i++)
  {
    printf("%d ", arr[i]);
  }
}
```

Insertion Sort

Insertion Sort is based on the idea that each element in the array is consumed in each iteration to find its right positi -on in the sorted array such that the entire array is sorted at the end of all the iterations.
In other words, it compares the current element with the elements on the left hand side (sorted array). If the current element is greater than all the elements on its left hand side, than it leaves the element in its place & moves on to the next element. Else it find its correct position by shifting all the elements which are larger than the current element, in the sorted array to one position ahead.

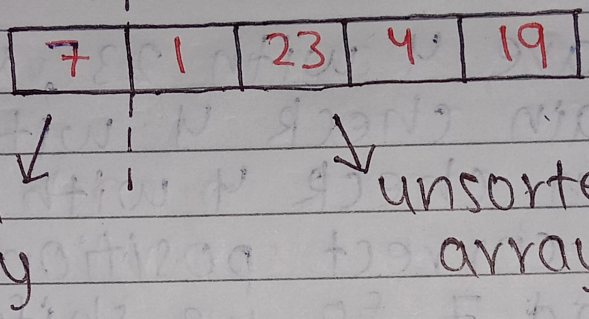


initially, only the first element is considered to be a part of the sorted array.

other array is considered as unsorted array.

Each iteration of insertion sort causes the sorted subset to grow & the unsor -ted subset to shrink.

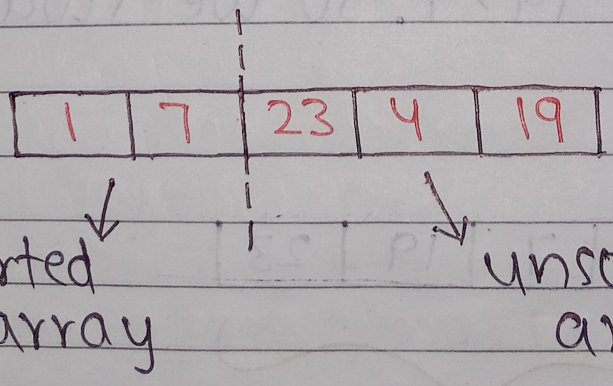
Example,



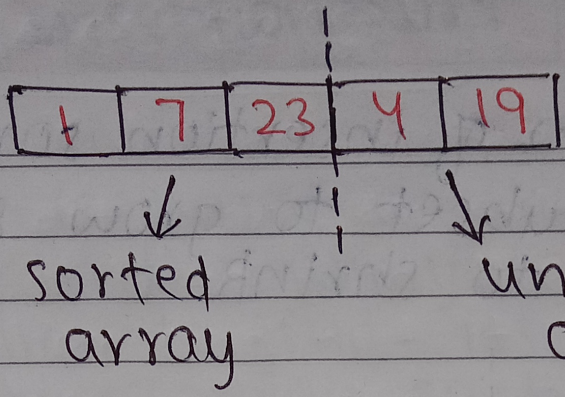
consider the first element, i.e., 7, as sorted as there are no other elements on its left hand side.

Now, we look at the next unsorted element, i.e., 1 and compare it with sorted element, i.e., 7.

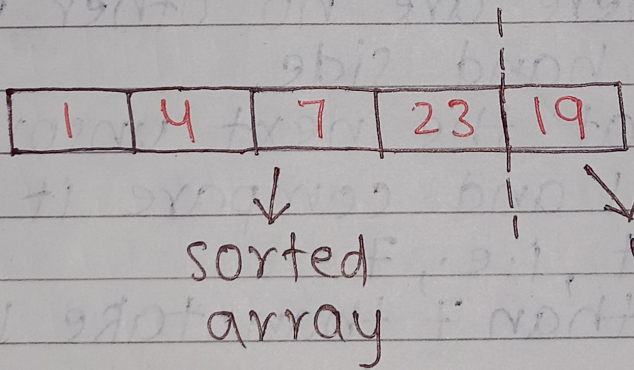
If 1 is less than 7 than take 1 into sorted array.



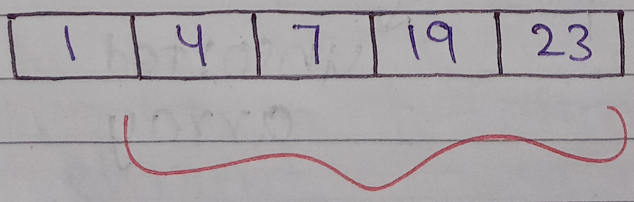
Now, we compare 23 with 7, 23 > 7, so no need to swap & we normally shift 23 into sorted array part.



Now, we compare 4 with 23, $4 < 23$
then we again check 4 with 7, $4 < 7$
then we again check 4 with 1, $4 > 1$.
It means the correct position of 4 is between 1 and 7. So, we shift all the elements from 7 to one position ahead.



Now, $19 < 23$ but $19 > 7$. So we swap 19 with 23.



the entire array is sorted

* Insertion sort :-

```
#include<stdio.h>
int main()
{
    int arr[] = {7, 1, 23, 4, 19};
    int n = 5;
    int i, j, temp;
    printf("before sort ");
    for(i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
    for(i=1; i<5; i++){
        j = i;
        while(j>0 && arr[j-1] > arr[j]){
            temp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = temp;
            j--;
        }
    }
    printf("after sort ");
    for(i=0; i<n; i++){
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

selection sort

Selection sort is based on the idea that, find the smallest number and put in the first place and then find the second smallest number and put it in the second place and so on.
This is the simple technique on which selection sort is based.
It is named so because in each pass it selects the smallest element and keeps it in its exact place.
For example :-

82	42	49	8	25	52	36	93	59
0	1	2	3	4	5	6	7	8

→ first we find the minimum element in the array & place it in the first position.

8	42	49	82	25	52	36	93	59
0	1	2	3	4	5	6	7	8

→ now 8 is in our correct position. Now we find the smallest element from the rest of the array and place it in the second position.

This process will continue until we come up with the sorted array.

* Selection Sort :-

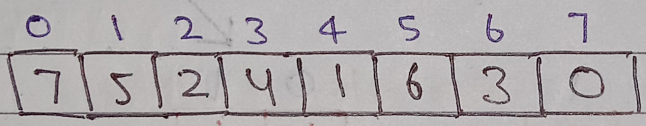
```
#include <stdio.h>
int main()
{
    int i, j, temp, min;
    int arr[] = {10, 69, 22, 45, 55, 80, 88, 90,
                                        14, 8};

    int n = 10;
    printf("unsorted array: ");
    for(i=0; i<n; i++)
    {
        printf("%d ", arr[i]);
    }
    for(i=0; i<n; i++)
    {
        min = i;
        for(j=i+1; j<n; j++)
        {
            if(arr[min] > arr[j])
            {
                min = j;
            }
        }
        if(i != min)
        {
            temp = arr[i];
            arr[i] = arr[min];
            arr[min] = temp;
        }
    }
```

```
printf("sorted list: ");
for(i=0; i<n; i++)
{
    printf("%d ", arr[i]);
}
printf("\n");
return 0;
}
```

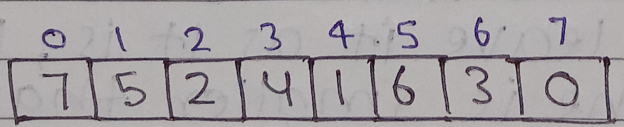
Merge Sort

Merge sort is one of the most efficient sorting algorithms. It works on the principle of divide & conquer. Merge sort repeatedly break down a array into several arrays until each array consist of a single element & merging those arrays in a manner that results into a sorted array.

For example, let suppose we have given an unsorted array,



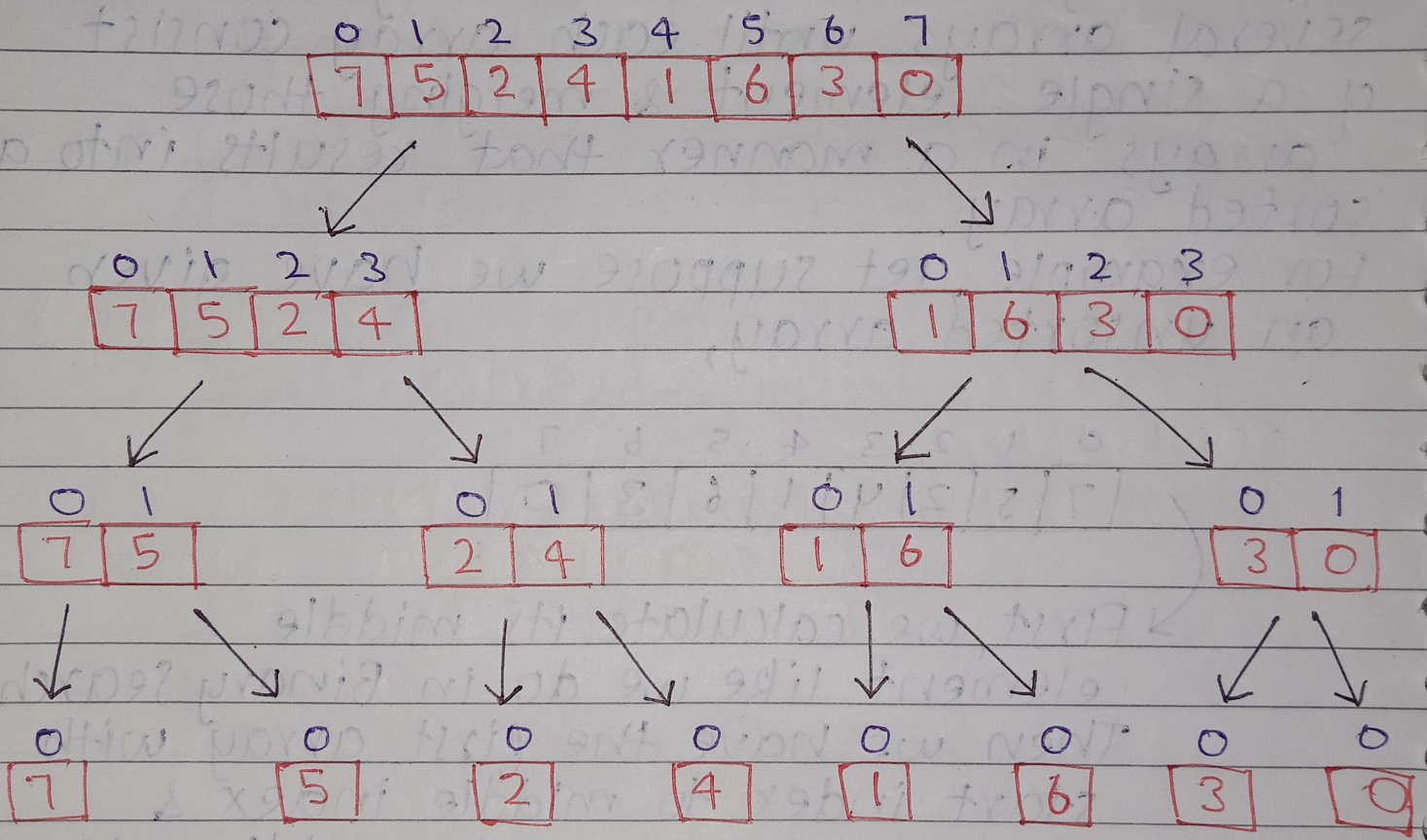
First, we calculate its middle element like we do in Binary Search Then we have the first array with start index to middle index & another array with middle + 1 index to end. The same procedure will follow until we have all the individual element left.



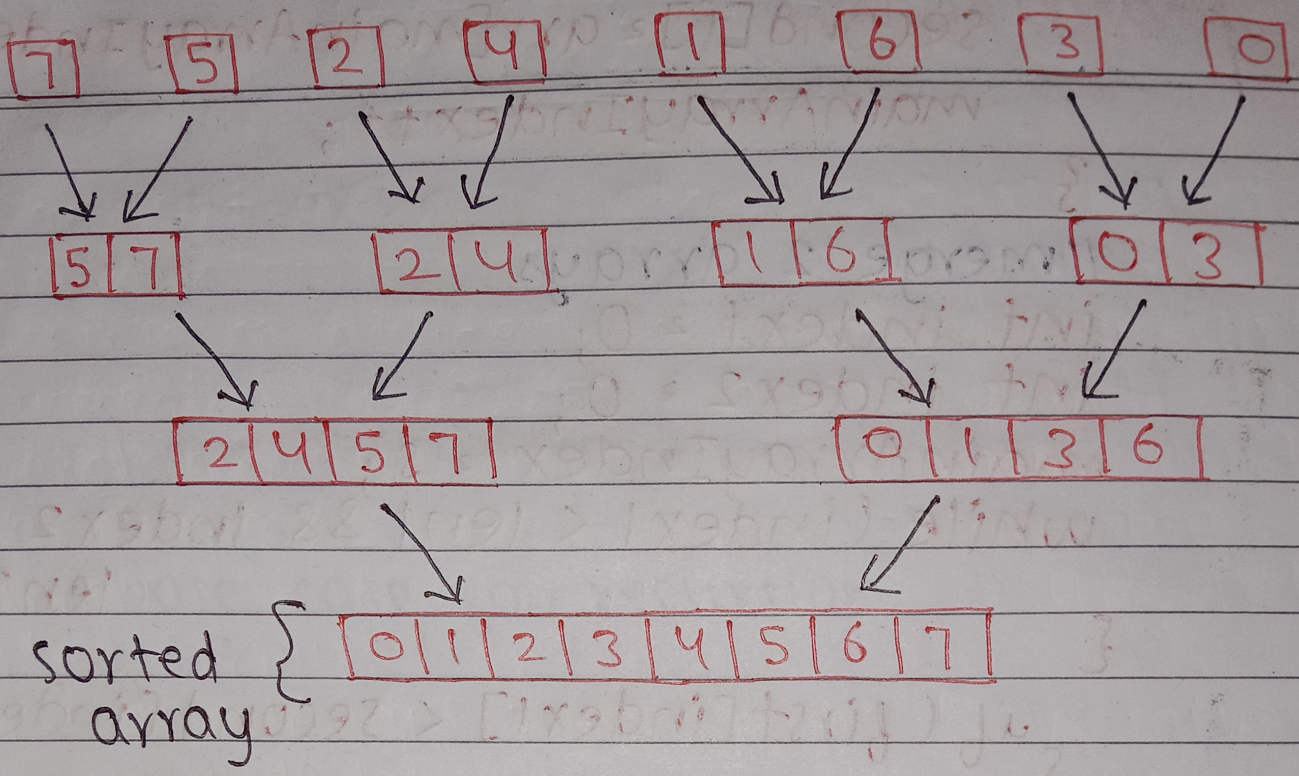
$$\frac{\text{start index} + \text{last index}}{2} = \frac{0+7}{2}$$

$$= 3$$

i.e., 3 is our middle element. Now we divide this array into 2 parts, first part is from (0 to 3) & second ~~part is from (3+1) to 7~~, i.e., 4 to 7 and the same strategy will follow until we have only single element left.



Now, we have single elements left. After this step we do the next method, i.e., merge sort, we compare the 2-2 pairs, if the left one is smaller than the right one, we leave it as it is, but if the right one is smailer than the left one than we swap the elements & the same procedure will follow until we find the proper sorted array.



Merge Sort :-

```
#include <stdio.h>
void merge(int arr[], int s, int e)
{
    int mid = (s+e)/2;
    int len1 = mid - s + 1;
    int len2 = end - mid;
    int first[len1], second[len2];
    //copy values from main array to
        temp array.
    int mainArrayIndex = s;
    for(int i=0; i<len1; i++)
    {
        first[i] = arr[mainArrayIndex];
        mainArrayIndex++;
    }
```

```
    mainArrayIndex = mid + 1;
    for(int i=0; i<len2; i++)
    {
        second[i] = arr[mainArrayIndex];
        mainArrayIndex++;
    }
    //merge 2 arrays
    int index1 = 0;
    int index2 = 0;
    mainArrayIndex = s;
    while (index1 < len1 && index2 <
                                len2)
    {
        if (first[index1] < second[index2])
        {
            arr[mainArrayIndex] =
                            first[index1];
            mainArrayIndex++;
            index1++;
        }
        else
        {
            arr[mainArrayIndex] =
                            second[index2];
            mainArrayIndex++;
            index2++;
        }
    }
    while (index1 < len1)
    {
        arr[mainArrayIndex] = first[index1];
        mainArrayIndex++;
        index1++;
    }
```

```
    while(index2 < len2)
    {
        arr[mainArrayIndex] =
                        second[index2];

        mainArrayIndex++;
        index2++;
    }
}

void mergesort(int arr[], int s, int e)
{
    //base case for recursion
    if (s >= e)
    {
        return;
    }
    int mid = (s+e)/2;
    //left part sort.
    mergeSort(arr, s, mid);
    //right part sort.
    merge sort(arr, mid+1, e);
    //merge function.
    merge(arr, s, e);
}

int main()
{
    int arr[] = {3,7,0,1,5,8,9,2,23,12};
    int n = 10;
    printf("Before sorting ");
    for(int i=0; i<n; i++)
    {
        printf("%d", arr[i]);
    }
```

```
mergeSort(arr, 0, n-1);
printf("after sorting ");
for(int i=0; i<n; i++)
{
  printf("%d ", arr[i]);
}
printf("\n");
return 0;
}
```

Quick sort

The name "Quick sort" comes from the fact that, Quick sort is capable of sorti -ng a array of data elements twice or thrice faster than any of the common algorithms. It is one of the most efficient sorting algorithm & is based on the splitting of an array (parti -tion) into smaller ones & swapping (exchange) based on the comparison with "pivot" element selected.
Due to this, Quick sort is also called as "Partition Exchange" sort.
Like Merge sort, Quick sort also falls into the category of divide & conquer approach of problem solving methodology.

Quick sort is a sorting algorithm based on Divide & conquer approach where :-

i) An array is divided into subarray by selecting a pivot element (element selected from the array).
while dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side & elements greater than pivot are on the right side of the pivot.

ii) The left & right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.

iii) At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

Quick sort :-

```
#include<stdio.h>
void quickSort(int arr[], int start,
                          int end)
{
  int i, j, pivot, temp;
  if(start < end)
  {
    pivot = start;
    i = start;
    j = end;
    while (i < j)
    {
      while(arr[i] <= arr[pivot])
      {
        i++;
      }
      while(arr[j] > arr[pivot])
      {
      }  j--;
      if(i < j)
      {
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
    }
    temp = arr[pivot];
```

```
        arr[pivot] = arr[j];
        arr[j] = temp;
        quickSort(arr, start, j-1);
        quickSort(arr, j+1, end);
    }
}

int main()
{
    int arr[] = {15, 20, 11, 17, 9, 69, 55,
                 41, 33, 37, 2, 10};
    int n = 12;

    quickSort(arr, 0, n-1);

    printf("after sorting");
    for(int i=0; i<n; i++)
    {
        printf("%d ", arr[i]);
    }

    printf("\n");
    return 0;
}
```