EX.NO : 12(A)	INSERTION SORT
DATE :	

PROGRAM STATEMENT:

To write a CPP Program to get the elements of an Array which needs to be sorted using insertion sort.

ALGORITHM:

1. Start the program.
2. Define the function input(int arr[]) that takes an array arr[] as an argument.
3. Loop from i = 0 to i = 4 (5 iterations):
   a. Prompt the user to input an integer.
   b. Store the entered integer in arr[i] using scanf.
4. End the program.

PROGRAM:

```
void    input(int    arr[]){
  for(int i=0;i<5;i++){
    scanf("%d",&arr[i]);
  }
}
```

	Input	Expected	Got	
✔	100 90 80 70 60	Before Sorting the Array: 100 90 80 70 60 After Sorting the Array: 60 70 80 90 100	Before Sorting the Array: 100 90 80 70 60 After Sorting the Array: 60 70 80 90 100	✔
✔	10 5 7 3 8	Before Sorting the Array: 10 5 7 3 8 After Sorting the Array: 3 5 7 8 10	Before Sorting the Array: 10 5 7 3 8 After Sorting the Array: 3 5 7 8 10	✔
✔	45 23 56 11 9	Before Sorting the Array: 45 23 56 11 9 After Sorting the Array: 9 11 23 45 56	Before Sorting the Array: 45 23 56 11 9 After Sorting the Array: 9 11 23 45 56	✔

Passed all tests! ✔

RESULT:

Thus, the C++ program to get the elements of an Array, which needs to be sorted using insertion sort, is created successfully.

EX.NO : 12(B)	QUICK SORT
DATE :	

PROGRAM STATEMENT:

To write the quickSort module of Quick Sort in CPP.

ALGORITHM:

1. **Start the program**.
2. If low < high, perform the following steps:
   a. Call the partition() function to find the pivot index pi.
   b. Recursively call quickSort() on the subarray to the left of pi (from low to pi-1).
   c. Recursively call quickSort() on the subarray to the right of pi (from pi+1 to high).
3. End the program**.**

PROGRAM:

```
voidquickSort(int array[], int low, int high)
{
if ( low < high )
{
   int pi =partition(array, low, high);
    quickSort(array, low, pi - 1);
   quickSort(array, pi + 1, high);
   }
}
```

OUTPUT:

	Input	Expected	Got	
✔	7 19 15 20 25 34 44 5	19 15 20 25 34 44 5 5 15 19 20 25 34 44	19 15 20 25 34 44 5 5 15 19 20 25 34 44	✔
✔	5 99 10 98 97 96	99 10 98 97 96 10 96 97 98 99	99 10 98 97 96 10 96 97 98 99	✔
✔	10 90 80 70 10 20 30 25 75 110 200	90 80 70 10 20 30 25 75 110 200 10 20 25 30 70 75 80 90 110 200	90 80 70 10 20 30 25 75 110 200 10 20 25 30 70 75 80 90 110 200	✔

Passed all tests! ✔

RESULT:

Thus, the C++ program to write the quickSort module of Quick Sort in CPP is created successfully.

EX.NO : 12(C)	MERGE SORT
DATE :	

PROGRAM STATEMENT:

To write the mergeSort Module of Merge Sort in CPP.
.

ALGORITHM:

1. Start the program.
2. If l < r, perform the following steps:
    a. Calculate the middle index m = l + (r - l) / 2.
    b. Recursively call mergeSort() on the left subarray (from l to m).
    c. Recursively call mergeSort() on the right subarray (from m + 1 to r).
    d. Call the merge() function to merge the two sorted subarrays (from l to m and from m+1 to r).
3. End the program

PROGRAM:

```
void mergeSort(int arr[], int l, int r)
{
if (l<r)
   {
   int m=l+(r-l)/2;
   mergeSort(arr,l,m);
    mergeSort(arr,m+1,r);
   merge(arr,l,m,r);
   }
}
```

OUTPUT:

	Input	Expected	Got	
✓	7 19 15 20 25 34 44 5	19 15 20 25 34 44 5 5 15 19 20 25 34 44	19 15 20 25 34 44 5 5 15 19 20 25 34 44	✓
✓	5 99 10 98 97 96	99 10 98 97 96 10 96 97 98 99	99 10 98 97 96 10 96 97 98 99	✓
✓	10 90 80 70 10 20 30 25 75 110 200	90 80 70 10 20 30 25 75 110 200 10 20 25 30 70 75 80 90 110 200	90 80 70 10 20 30 25 75 110 200 10 20 25 30 70 75 80 90 110 200	✓

Passed all tests! ✓

RESULT:

Thus, the C++ program To write the mergeSort Module of Merge Sort in CPP.is created successfully.

EX.NO : 12(D)	BINARY SEARCH ALGORITHM
DATE :	

PROGRAM STATEMENT:

To write the Binary Search Module of Binary Search Algorithm in CPP.

ALGORITHM:

1. Start the program.
2. Initialize beg = 0 and end = n.
3. While beg <= end, perform the following:
    a. Calculate the middle index mid = (beg + end) / 2.
    b. If a[mid] == search, print "Element found at position mid + 1", and return 1 (indicating the element was found).
    c. If a[mid] > search, update end = mid - 1 to search the left half.
    d. Otherwise, update beg = mid + 1 to search the right half.
4. If the loop ends without finding the element, return 0 (indicating the element was not found).
5. End the program.

PROGRAM:

```
int BS(int a[], int n, int search){
   int beg=0, end=n, mid;

   while(beg<=end)
   {
   mid=(beg+end)/2;
    if(a[mid]==search)
      {
      printf("Element found at %d position",mid + 1);
      return 1;
      break; }
  else if(a[mid] >search)
      end=mid-1;
   else
         beg=mid+1;
   }
   Return 0;
}
```

OUTPUT:

	Input	Expected	Got	
✔	5 10 20 30 40 50 40	Element found at 4 position	Element found at 4 position	✔
✔	9 10 20 30 40 50 60 70 80 90 80	Element found at 8 position	Element found at 8 position	✔
✔	10 1 2 3 4 5 6 7 80 90 99 100	Element Not Found	Element Not Found	✔

Passed all tests! ✔

RESULT:

Thus, the C++ program to write the Binary Search Module of Binary Search Algorithm in CP is created successfully.