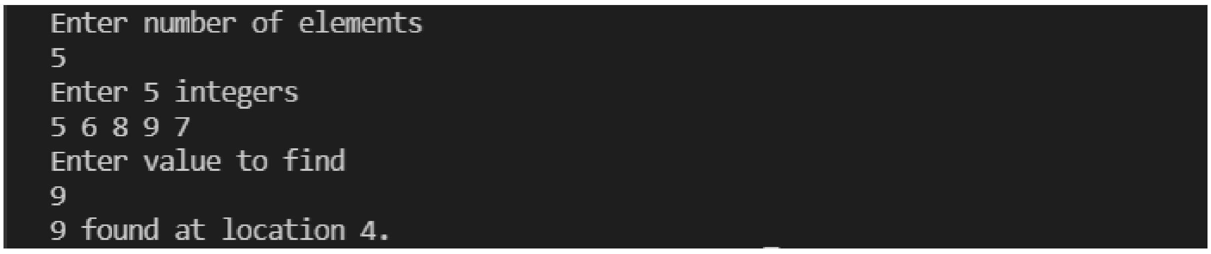
1. Binary Search:

CODE:

```c
/* Binary Search using Iteration */

/* Major Condition : Array must be sorted for Binary Search */

#include <stdio.h>
int main()
{
  int c, first, last, middle, n, search, array[100];

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)  // Taking the input
    scanf("%d", &array[c]);

  printf("Enter value to find\n");
  scanf("%d", &search);
/* In Binary Search, We split the array in two subarrays,
   if target is less than the mid, we traverse the left subarray
   if target is greater than mid, we traverse the right subarray
   and do the same procedure */
  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while (first <= last) {
    if (array[middle] < search)
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);

  return 0;
}
```

OUTPUT:



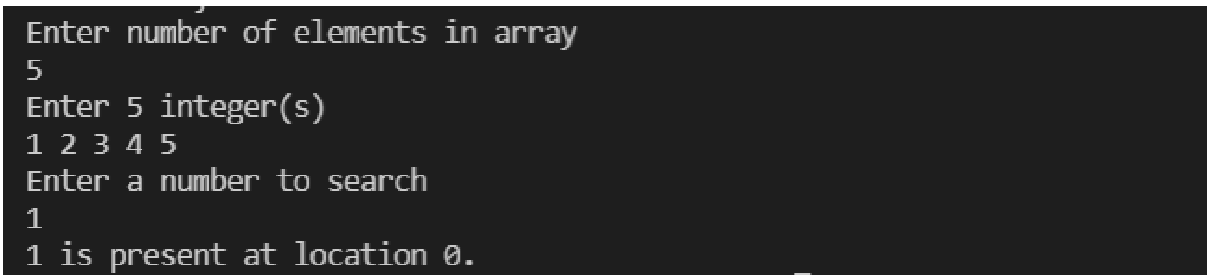
```
Enter number of elements
5
Enter 5 integers
5 6 8 9 7
Enter value to find
9
9 found at location 4.
```

2. Linear Search

CODE:

```c
/* Code For Linear Search*/

#include <stdio.h>
int main()
{
  int array[100], search, i, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n); //Taking the inputs.

  for (i = 0; i < n; i++)
    scanf("%d", &array[i]);

  printf("Enter a number to search\n");
  scanf("%d", &search);
/* In Linear search, We just Compare the required element with
   each element in the array */
  for (i = 0; i < n; i++)
  {
    if (array[i] == search)     /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, i);
      break;
    }
  }
  if (i == n)
    printf("Element not found.\n");

  return 0;
}
```

OUTPUT:



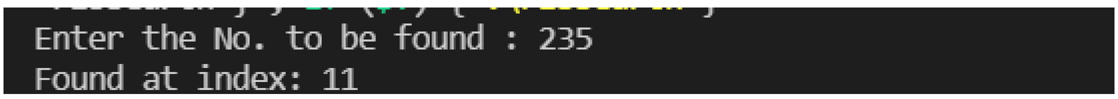
```
Enter number of elements in array
5
Enter 5 integer(s)
1 2 3 4 5
Enter a number to search
1
1 is present at location 0.
```

3. Fibonacci Search

CODE:

```c
// C program for Fibonacci Search
// Average case Time Complexity: O(log n)
// Worst case Time Complexity: O(log n)

#include <stdio.h>

int min(int x, int y) { return (x <= y) ? x : y; }

int fibMonaccianSearch(int arr[], int x, int n)
{
    int fibMMm2 = 0; // (m-2)'th Fibonacci No.
    int fibMMm1 = 1; // (m-1)'th Fibonacci No.
    int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

       /*Number greater than or equal to n */
    while (fibM < n) {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    // Marks the eliminated range from front
    int offset = -1;

    /* while there are elements to be inspected. Note that
       we compare arr[fibMm2] with x. When fibM becomes 1,
       fibMm2 becomes 0 */
    while (fibM > 1) {
        // Check if fibMm2 is a valid location
        int i = min(offset + fibMMm2, n - 1);

        /* If x is greater than the value at index fibMm2,
           cut the subarray array from offset to i */
        if (arr[i] < x) {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }

        /* If x is greater than the value at index fibMm2,
           cut the subarray after i+1  */
        else if (arr[i] > x) {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        /* element found. return index */
        else
            return i;
    }

    if (fibMMm1 && arr[offset + 1] == x)
        return offset + 1;

    return -1;
}

int main(void)
{
    int arr[]
        = { 10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100,235};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x ;
    printf("Enter the No. to be found : ");
    scanf("%d",&x);
      int ind = fibMonaccianSearch(arr, x, n);
  if(ind>=0)
    printf("Found at index: %d",ind);
  else
    printf("%d isn't present in the array",x);
    return 0;
}
```

OUTPUT:



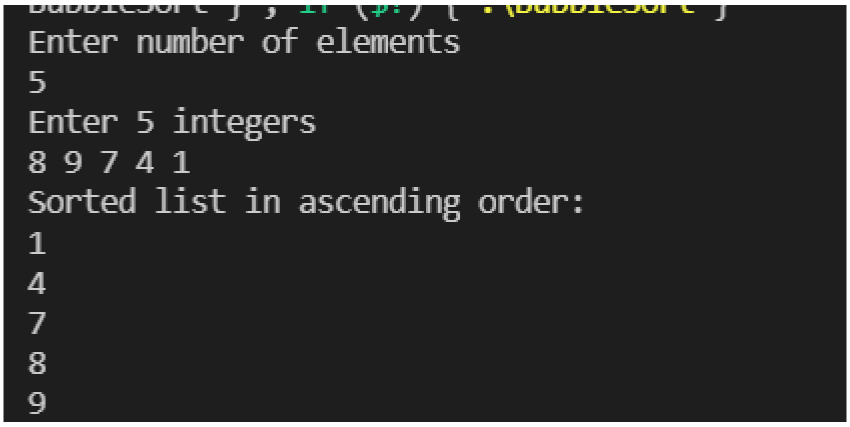
```
Enter the No. to be found : 235
Found at index: 11
```

4. Bubblesort

CODE:

```c
/* Bubble sort code */
#include <stdio.h>

int main()
{
  int array[100], n, i, j, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n); //taking the inputs

  for (i = 0; i < n; i++)
    scanf("%d", &array[i]);
/* In Bubble Sort, We just compare Elements one after the other
   and swap them, if needed */
  for (i = 0 ; i < n - 1; i++)
  {
    for (j = 0 ; j < n - i - 1; j++)
    {
      if (array[j] > array[j+1]) /* For decreasing order use '<' instead of
'>' */
      {
        swap       = array[j];
        array[j]   = array[j+1];
        array[j+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for (i = 0; i < n; i++)
     printf("%d\n", array[i]); // Printing the sorted array.

  return 0;
}
```

OUTPUT:

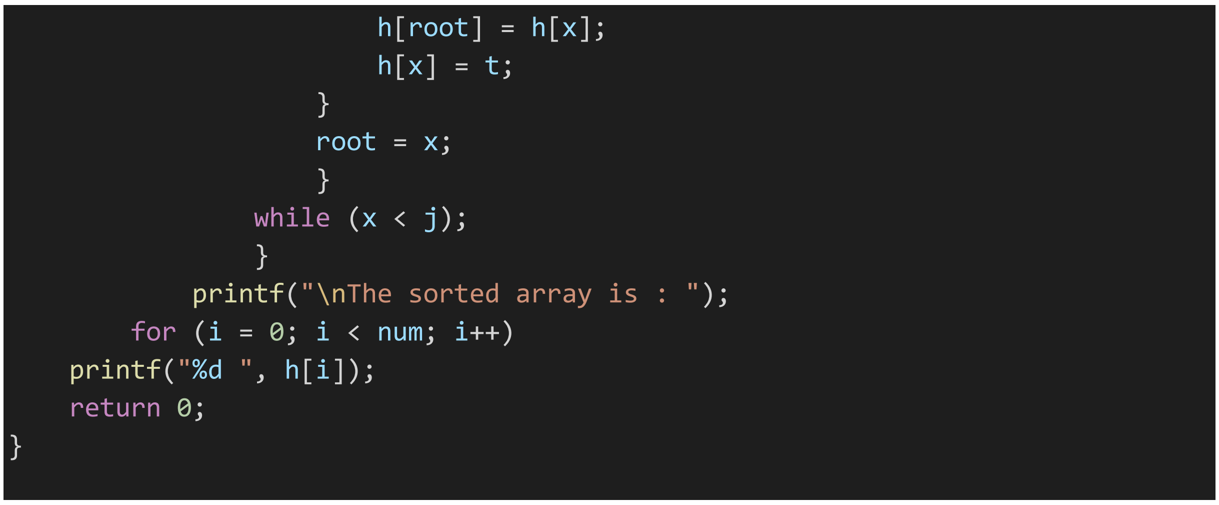


```
BubbleSort } ; if ($?) { .\BubbleSort }
Enter number of elements
5
Enter 5 integers
8 9 7 4 1
Sorted list in ascending order:
1
4
7
8
9
```

5. HeapSort

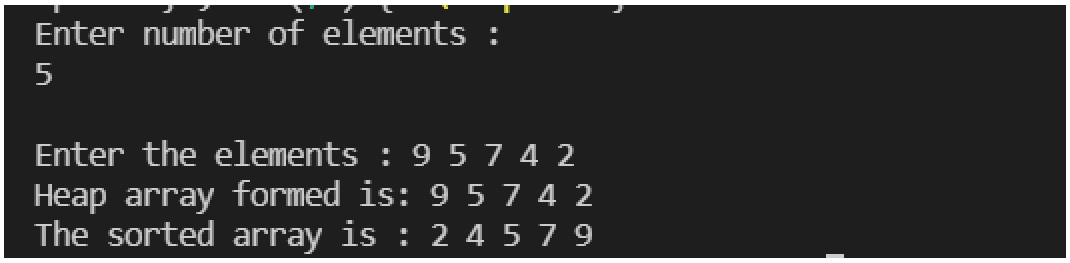
CODE:

```
// C program for Heap Sort
// worst case time complexity: O(n logn)

#include <stdio.h>
int main()
{
    int h[20],num,i,j,root,t,x;
    printf("Enter number of elements :");
    scanf("%d", &num);
    printf("\nEnter the elements : ");
    for (i = 0; i < num; i++)
        scanf("%d", &h[i]);
        for(i=0;i<num;i++)
        {
            x=i;
            do
            {
                root = (x - 1) / 2;
                if (h[root] < h[x])
                {
                                t= h[root];
                    h[root] = h[x];
                    h[x] = t;
                }
                x = root;
            }
            while (x != 0);
            }
        printf("Heap array formed is: ");
        for (i = 0; i < num; i++)
            printf("%d ", h[i]);
            for (j = num - 1; j >= 0; j--)
            {
                t = h[0];
                h[0] = h[j];
                h[j] = t;
                root = 0;
                do
                {
                    x = 2 * root + 1;
                    if ((h[x] < h[x + 1]) && x < j-1)
                    x++;
                    if (h[root]<h[x] && x<j)
                    {
                        t = h[root];
```



```
                        h[root] = h[x];
                        h[x] = t;
                    }
                    root = x;
                    }
                while (x < j);
                }
            printf("\nThe sorted array is : ");
        for (i = 0; i < num; i++)
    printf("%d ", h[i]);
    return 0;
}
```

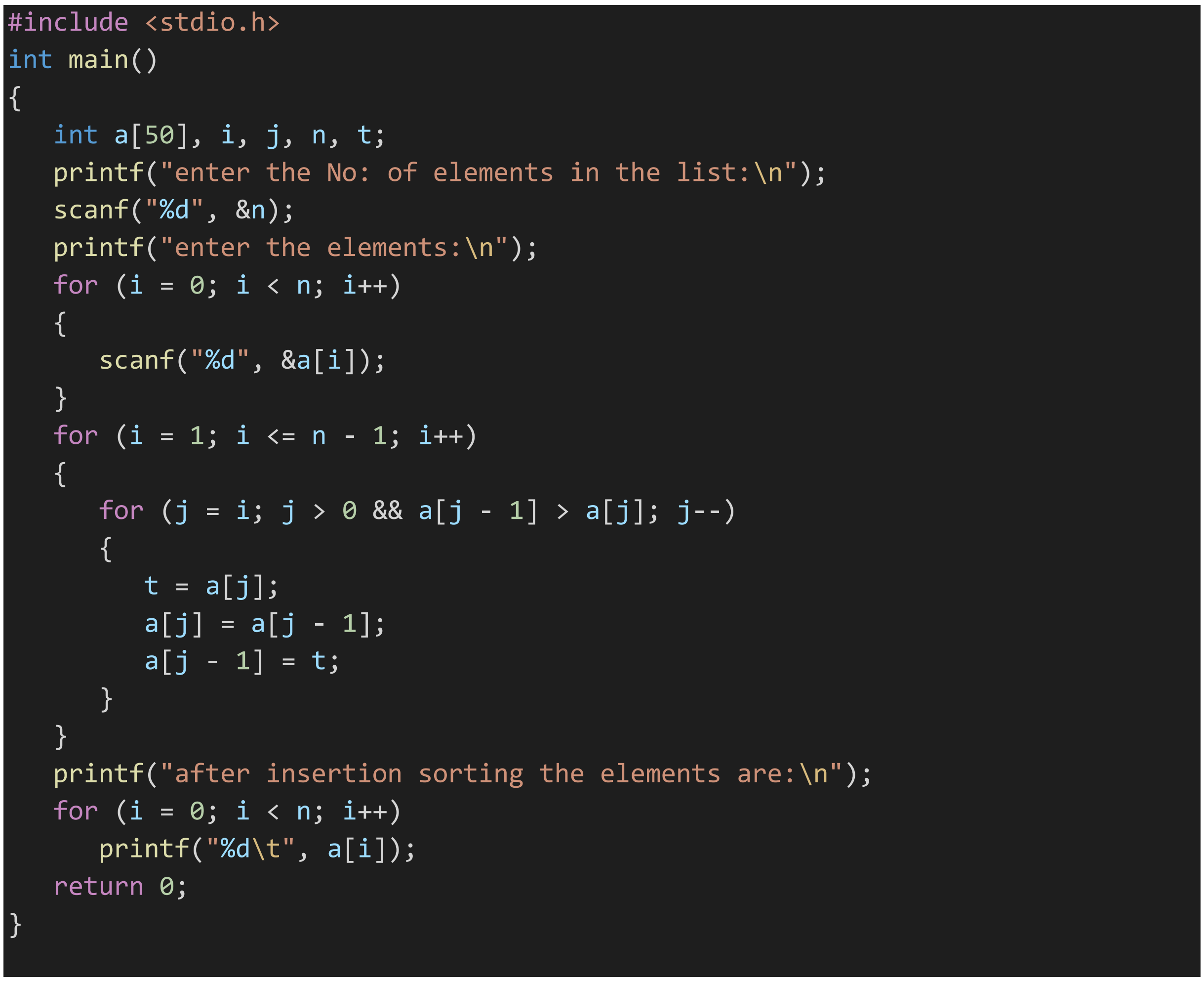
OUTPUT:



```
Enter number of elements :
5

Enter the elements : 9 5 7 4 2
Heap array formed is: 9 5 7 4 2
The sorted array is : 2 4 5 7 9
```

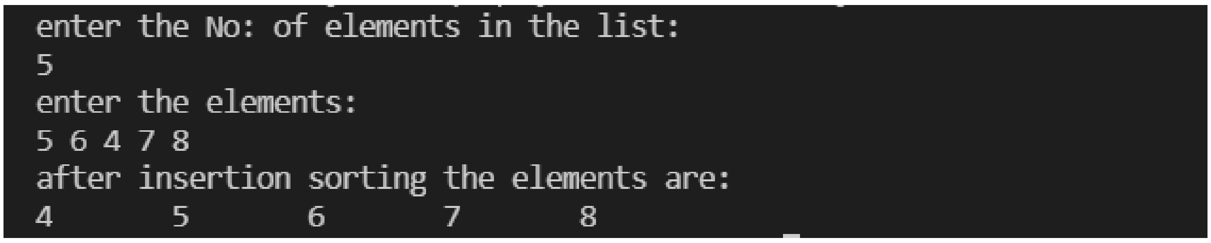
6. Insertion Sort

CODE:



```c
#include <stdio.h>
int main()
{
   int a[50], i, j, n, t;
   printf("enter the No: of elements in the list:\n");
   scanf("%d", &n);
   printf("enter the elements:\n");
   for (i = 0; i < n; i++)
   {
      scanf("%d", &a[i]);
   }
   for (i = 1; i <= n - 1; i++)
   {
      for (j = i; j > 0 && a[j - 1] > a[j]; j--)
      {
         t = a[j];
         a[j] = a[j - 1];
         a[j - 1] = t;
      }
   }
   printf("after insertion sorting the elements are:\n");
   for (i = 0; i < n; i++)
      printf("%d\t", a[i]);
   return 0;
}
```

OUTPUT:

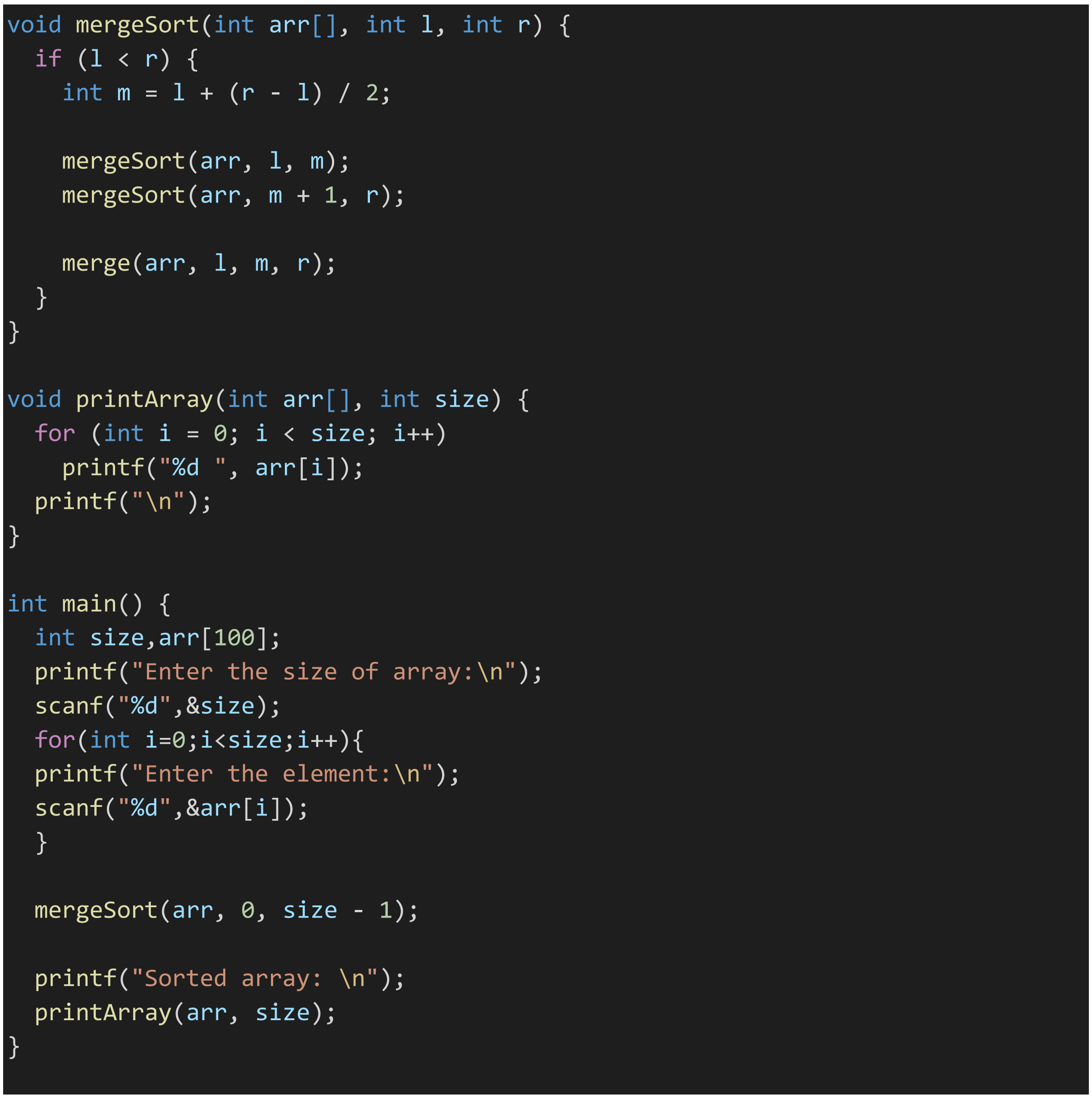


```
enter the No: of elements in the list:
5
enter the elements:
5 6 4 7 8
after insertion sorting the elements are:
4       5       6       7       8
```

7. Merge Sort

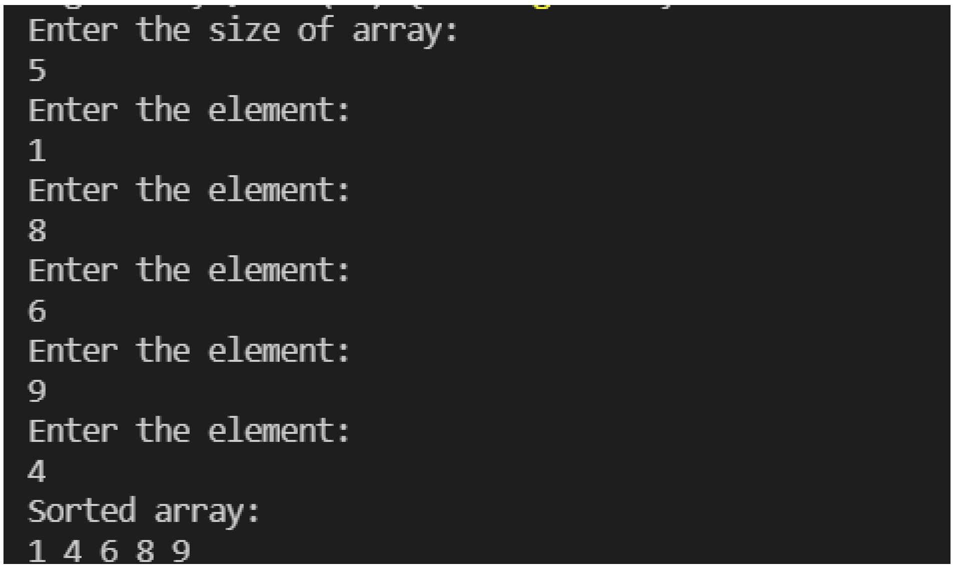
CODE:

```c
// Merge Sort Program in C
// Worst case Time Complexity: O(n logn)
#include <stdio.h>

void merge(int arr[], int p, int q, int r) {

  int n1 = q - p + 1;
  int n2 = r - q;

  int L[n1], M[n2];

  for (int i = 0; i < n1; i++)
    L[i] = arr[p + i];
  for (int j = 0; j < n2; j++)
    M[j] = arr[q + 1 + j];

  int i, j, k;
  i = 0;
  j = 0;
  k = p;

  while (i < n1 && j < n2) {
    if (L[i] <= M[j]) {
      arr[k] = L[i];
      i++;
    } else {
      arr[k] = M[j];
      j++;
    }
    k++;
  }

  while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
  }

  while (j < n2) {
    arr[k] = M[j];
    j++;
    k++;
  }
}
```



```c
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);
  }
}

void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++)
    printf("%d ", arr[i]);
  printf("\n");
}

int main() {
  int size,arr[100];
  printf("Enter the size of array:\n");
  scanf("%d",&size);
  for(int i=0;i<size;i++){
  printf("Enter the element:\n");
  scanf("%d",&arr[i]);
  }

  mergeSort(arr, 0, size - 1);

  printf("Sorted array: \n");
  printArray(arr, size);
}
```

OUTPUT:



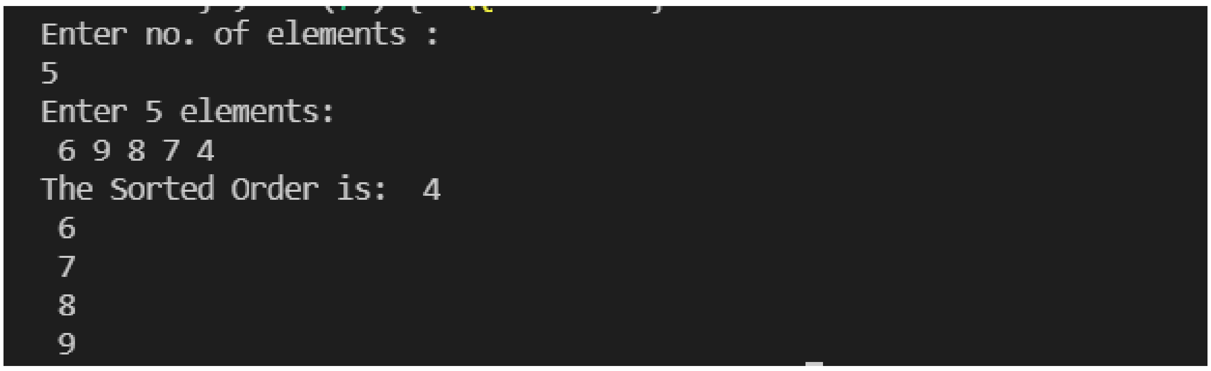
```
Enter the size of array:
5
Enter the element:
1
Enter the element:
8
Enter the element:
6
Enter the element:
9
Enter the element:
4
Sorted array:
1 4 6 8 9
```

8. QuickSort

CODE:

```c
// Quick Sort Program in C
// Best case Time Complexity: O(n logn)
// Average case Time Complexity: O(n logn)
// Worst case time complexity: O(n^2)
#include <stdio.h>

void quicksort(int number[100], int first, int last)
{
    int i, j, pivot, temp;
    if (first < last)
    {
        pivot = first;
        i = first;
        j = last;
        while (i < j)
        {
            while (number[i] <= number[pivot] && i < last)
                i++;
            while (number[j] > number[pivot])
                j--;
            if (i < j)
            {
                temp = number[i];
                number[i] = number[j];
                number[j] = temp;
            }
        }
        temp = number[pivot];
        number[pivot] = number[j];
        number[j] = temp;
        quicksort(number, first, j - 1);
        quicksort(number, j + 1, last);
    }
}

int main()
{
    int i, count, number[25];
    printf("Enter no. of elements : \n");
    scanf("%d", &count);
    printf("Enter %d elements:\n ", count);
    for (i = 0; i < count; i++)
        scanf("%d", &number[i]);
    quicksort(number, 0, count - 1);
    printf("The Sorted Order is: ");
    for (i = 0; i < count; i++)
        printf(" %d\n", number[i]);
    return 0;
}
```

OUTPUT:



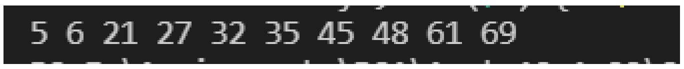
```
Enter no. of elements :
5
Enter 5 elements:
 6 9 8 7 4
The Sorted Order is:  4
 6
 7
 8
 9
```

9. SelectionSort

CODE:

```
#include <stdio.h>
int main()
{
   int arr[10] = {5, 48, 69, 35, 21, 61, 27, 6, 32, 45};
   int n = 10;
   int i, j, position, swap;
   for (i = 0; i < (n - 1); i++)
   {
      position = i;
      for (j = i + 1; j < n; j++)
      {
         if (arr[position] > arr[j])
            position = j;
      }
      if (position != i)
      {
         swap = arr[i];
         arr[i] = arr[position];
         arr[position] = swap;
      }
   }
   for (i = 0; i < n; i++)
      printf("%d\t", arr[i]);
   return 0;
}
```

OUTPUT:



```
5 6 21 27 32 35 45 48 61 69
```