********** Algorithm for the randomize quick sort **********

```
#include <iostream>
#include <cstdlib>
#include<ctime>
using namespace std;

int dividing(int arr[], int first, int last)
{

   int pivot = arr[last]; //taking the pivot

   int a = (first - 1);  // taking the lower element

   for (int b = first; b <= last - 1; b++)
   {
      if (arr[b] <= pivot)
      {
         a++;
         swap(arr[a], arr[b]);
      }
   }
   swap(arr[a + 1], arr[last]);
   return (a + 1);
}


int dividing_r(int arr[], int first, int last)
{
```

```
   // Taking the random index using srand such that it sets the starting point of producing the series of random numbers generated.
   srand(time(NULL));
   int random = first + rand() % (last - first); //using rand function to generate random numbers

   // Swaping the index with last element as per this algorithm
   swap(arr[random], arr[last]);

   return dividing(arr, first, last);
}

void quicksort(int arr[], int first, int last)
{
   if (first < last) {

      /* di is dividing index,
      arr position point at right most position */
      int di = dividing_r(arr, first, last);

      // Separately sort elements before
      // dividing and after dividing
      quicksort(arr, first, di - 1);
      quicksort(arr, di + 1, last);
   }
}
void printsortedArray(int B[])
{
   for (int i = 1; i <=100; i++)
   cout<< B[i]<<" ";
```

```
}

void generatearray(int A[])
{
  double time_req1;
  for(int i=0;i<100;i++)
  {
     A[i]=i+1;
  }
  for(int j=0 ;j<5;j++)
  {
     time_req1 = clock();
     quicksort(A, 0, 100);
     time_req1 = clock() - time_req1;
     cout<<" The time required for  "<<j+1<<" iteration  "<<time_req1<<"seconds"<<" \n ";
  }

}

int main() {
       int c[500];
       generatearray(c);
       cout<<" \n ";
       cout <<" sorted array ";
       cout<<" \n ";
       printsortedArray(c);

return 0;
}
```

OUTPUT

The time required for 1 iteration 12seconds

The time required for 2 iteration 9seconds

The time required for 3 iteration 10seconds

The time required for 4 iteration 9seconds

The time required for 5 iteration 8seconds

sorted array

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**********Algorithm for the heap sort for the random generation of the numbers from 1 to 100*****

```
#include <iostream>
#include <cstdlib>
#include<ctime>
using namespace std;

void Max_Heap(int Y[], int m, int d)
{
   int biggest = d;
   int left = 2 * d + 1;
   int right = 2 * d + 2;

   // if left is greater than root node
   if (left < m && Y[left] > Y[biggest])
      biggest = left;
```

```
    // If right child is greater than root node
    if (right < m && Y[right] > Y[biggest])
        biggest = right;

    // If biggest node is not root node
    if (biggest != d) {
        swap(Y[d], Y[biggest]);

        // Recursively Max_Heap gets called
        Max_Heap(Y, m, biggest);
    }
}

void Sort_heap(int D[], int m)
{
    for (int v = m / 2 - 1; v >= 0; v--)//Building the heap array
        Max_Heap(D, m, v);
    for (int w = m - 1; w >= 0; w--) //Extract the element from array
    {
        swap(D[0], D[w]);  //Max element is at the end of the array
        Max_Heap(D, w, 0); // Maximum heap array
    }
}
void FinalArray(int B[])
{
    for (int i = 0; i<100; i++)
    cout<< B[i]<<" ";
}
```

```
void genrandomarray(int Z[])
{
   cout<<"Random numbers generated from 1 and 100:"<<endl;
  for ( int x=0;x<100;x++)
  {
   int random = (rand() % 100) ;
      swap(Z[x],Z[random]);
  }

  for(int s=0;s<100;s++)
   cout<<Z[s]<<" ";

   Sort_heap(Z,100);
}
int main()
{
       int c[500];
       cout<<" Input the elements of array between 1 to 100 ";
       for(int i=0;i<100;i++)
       {
         c[i]=i+1;
         cout<<c[i]<<" ";
       }
       cout<<” \n’;
       genrandomarray(c);
       cout<<" \n ";
       cout <<" sorted array ";
       cout<<"\n";
       FinalArray(c);
```

return 0;

}

OUTPUT :

```
 Input the elements of array between 1 to 100 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 6
5 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Random numbers generated from 1 and 100:
84 87 32 7 94 66 2 93 14 22 63 39 72 10 75 49 41 4 73 44 28 40 33 12 58 62 34 15 23 24 6 78 13 59 77 29 65 46 21 90
 26 74 69 79 85 56 67 83 27 71 80 54 92 81 55 42 18 35 97 82 53 31 45 30 3 61 99 76 20 9 48 96 25 57 64 17 8 5 1 60
 88 50 70 37 86 89 98 11 38 100 68 36 19 51 47 91 52 16 43 95
  sorted array
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 4
2 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

**********Algorithm for the counting sort **********

```
#include<iostream>
using namespace std;

void counting_sort(int A[], int k, int n)
{
   int i, j;
   int B[n], C[k + 1];
   //Initializing counting array C[i] to 0
   for (i = 0; i <= k; i++)
      C[i] = 0;
   //Store count of each element in array C
   for (j = 0; j < n; j++)
      C[A[j]] = C[A[j]] + 1;
   /* Change C[i] such that it contains actual
   position of these elements in output array*/
```

```
    for (i = 1; i < k + 1; i++)
        C[i] = C[i] + C[i - 1];
    //Creating Output array from C[i]
    //and decrementing value of C[i].
    for (j = n - 1; j >= 0; j--)
    {
        B[C[A[j]] - 1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }
    //Printing sorted array
    cout<<"The Sorted array using the counting sort is : ";
    for (i = 0; i < n; i++)
        cout<<B[i]<<" ";
}
int main()
{
    int n, max = 0, i;
    int A[11]={20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0};
    cout << (" The array list to be sorted is 20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0\n");
    /*Storing element in a array. And finding max of elements to set range of counting array C[]*/
    for (i = 0; i < 11; i++)
    {
        if (A[i] > max) {
            max = A[i];
        }
    }
    //calling counting-sort function
    counting_sort(A, max, 11);
    cout << "\n";
```

```
    return 0;
}
```

OUTPUT :

```
 The array list to be sorted is 20, 18, 5, 7, 16, 10, 9, 3, 12, 14, 0
The Sorted array using the counting sort is : 0 3 5 7 9 10 12 14 16 18 20
```

********** Algorithm for the Radix sort array **********

```
#include<iostream>
using namespace std;
void counting_sort(int A[], int k, int n)
{
    int i, j;
    int B[n], C[k + 1];

    for (i = 0; i <= k; i++)
        C[i] = 0;

    for (j = 0; j < n; j++)
        C[A[j]] = C[A[j]] + 1;

    for (i = 1; i < k + 1; i++)
        C[i] = C[i] + C[i - 1];

    for (j = n - 1; j >= 0; j--)
    {
        B[C[A[j]] - 1] = A[j];
        C[A[j]] = C[A[j]] - 1;
    }
```

```
   cout << "The Sorted array using the radix sort of every iteration while moving from lower order to
higher ";
   cout<<" \n ";
   for (i = 0; i < n; i++)
    {
       cout << B[i] << " ";
    }
   cout<<" \n ";
}

void radix_sort(int A[])
{
   int max;
   for (int i = 0; i < 7; i++)
   {
      if (A[i] > max) {
         max = A[i];
      }
   }
   cout<<" The radix sort array using  counting sort for decimal digits from the low order to high order.";
   cout << "\n";
   cout<<" The last part of iteration prints the sorted array ";
   cout << "\n";
   for (int mul = 1; max / mul > 0; mul *= 10)
      counting_sort(A,mul,7);
}

int main()
```

```
{
   int A[7] = { 329, 457, 657, 839, 436, 720, 353 };
   cout << (" The array list to be sorted is 329, 457, 657, 839, 436, 720, 353 \n");
   //calling radix-sort function
   radix_sort(A);
   cout << "\n";
   return 0;
}
```

OUTPUT

```
 The array list to be sorted is 329, 457, 657, 839, 436, 720, 353
 The radix sort array using  counting sort for decimal digits from the low order to high order.
 The last part of iteration prints the sorted array
The Sorted array using the radix sort of every iteration while moving from lower order to higher
 329 21932 0 0 67 0 0
 The Sorted array using the radix sort of every iteration while moving from lower order to higher
 329 21932 0 0 67 0 0
 The Sorted array using the radix sort of every iteration while moving from lower order to higher
 329 21932 0 0 67 0 0
 The Sorted array using the radix sort of every iteration while moving from lower order to higher
 329 353 436 457 657 720 839
 The Sorted array using the radix sort of every iteration while moving from lower order to higher
 329 353 436 457 657 720 839
```