: Quick sort algorithm

```
// Quick sort in C++

#include <iostream>
using namespace std;

// function to print the array
void printArray(int array[], int size) {
  int i;
  for (i = 0; i < size; i++)
    cout << array[i] << " ";
  cout << endl;
}

// function to rearrange array (find the partition point)
int partition(int array[], int low, int high) {

  // select the rightmost element as pivot
  int pivot = array[high];

  // pointer for greater element
  int i = (low - 1);

  // traverse each element of the array
  // compare them with the pivot
  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {

      // if element smaller than pivot is found
      // swap it with the greater element pointed by i
      i++;

      // swap element at i with element at j
      swap(array[i],array[j]);
    }
  }

  // swap pivot with the greater element at i
  swap(array[i + 1], array[high]);

  // return the partition point
  return (i + 1);
```

```
}

void quickSort(int array[], int low, int high) {
  if (low < high) {

    // find the pivot element such that
    // elements smaller than pivot are on left of pivot
    // elements greater than pivot are on righ of pivot
    int pi = partition(array, low, high);

    // recursive call on the left of pivot
    quickSort(array, low, pi - 1);

    // recursive call on the right of pivot
    quickSort(array, pi + 1, high);
  }
}

// Driver code
int main() {

 int n, i;
 cout<<"\nEnter the number of data element to be sorted: ";
cin>>n;
 int arr[n];
 for(i = 0; i < n; i++)
 {
      cout<<"Enter element "<<i+1<<": ";
      cin>>arr[i];
 }

   // perform quicksort on data
  quickSort(arr, 0, n - 1);

  cout << "Sorted array in ascending order: \n";
  printArray(arr, n);
}
```

: Counting sort algorithm

```
// Counting sort in C++ programming

#include <iostream>
using namespace std;

void countSort(int array[], int size) {
  // The size of count must be at least the (max+1) but
  // we cannot assign declare it as int count(max+1) in C++ as
  // it does not support dynamic memory allocation.
  // So, its size is provided statically.
  int output[20];
  int count[20];
  int max = array[0];

  // Find the largest element of the array
  for (int i = 1; i < size; i++) {
    if (array[i] > max)
      max = array[i];
  }

  // Initialize count array with all zeros.
  for (int i = 0; i <= max; ++i) {
    count[i] = 0;
  }

  // Store the count of each element
  for (int i = 0; i < size; i++) {
    count[array[i]]++;
  }

  // Store the cummulative count of each array
  for (int i = 1; i <= max; i++) {
    count[i] += count[i - 1];
  }

  // Find the index of each element of the original array in
//count array, and
  // place the elements in output array
  for (int i = size - 1; i >= 0; i--) {
    output[count[array[i]] - 1] = array[i];
    count[array[i]]--;
  }

  // Copy the sorted elements into original array
  for (int i = 0; i < size; i++) {
```

```
    array[i] = output[i];
  }
}

// Function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++)
    cout << array[i] << " ";
  cout << endl;
}

// Driver code
int main() {

 int n, i;
 cout<<"\nEnter the number of data element to be sorted: ";
    cin>>n;
 int arr[n];
 for(i = 0; i < n; i++)
 {
      cout<<"Enter element "<<i+1<<": ";
      cin>>arr[i];
 }
    cout<<"\nSorted Data\n ";

   countSort(arr, n);
   printArray(arr, n);
}
```

: Shell sort algorithm

```
// C++ implementation of Shell Sort

#include <iostream>
using namespace std;

/* function to sort arr using shellSort */
void shellSort(int arr[], int n)
{
 // Start with a big gap, then reduce the gap
 for (int gap = n/2; gap > 0; gap /= 2)
 {
  // Do a gapped insertion sort for this gap size.
 // The first gap elements a[0..gap-1] are already in gapped
order
 //keep adding one more element until the entire array is gap
sorted
      for (int i = gap; i < n; i += 1)
      {
      // add a[i] to the elements that have been gap sorted
     // save a[i] in temp and make a hole at position i
           int temp = arr[i];

    // shift earlier gap-sorted elements up until the correct
    // location for a[i] is found
      int j;
      for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

      // put temp (the original a[i]) in its correct location
           arr[j] = temp;
      }
 }

}

void printArray(int arr[], int n)
{
 for (int i=0; i<n; i++)
      cout << arr[i] << " ";
}

int main()
{

 int n, i;
```

```
cout<<"\nEnter the number of data element to be sorted: ";
    cin>>n;
int arr[n];
for(i = 0; i < n; i++)
{
     cout<<"Enter element "<<i+1<<": ";
     cin>>arr[i];
}


shellSort(arr, n);

cout << "\nArray after sorting: \n";
printArray(arr, n);

return 0;
}
```