Shell sort:

```
// Java implementation of ShellSort
class ShellSort
{
	/* An utility function to print array of size n*/
	static void printArray(int arr[])
	{
		int n = arr.length;
		for (int i=0; i<n; ++i)
			System.out.print(arr[i] + " ");
		System.out.println();
	}
	/* function to sort arr using shellSort */
	int sort(int arr[])
	{
		int n = arr.length;
		// 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;
            }
        }
        return 0;
    }
    // Driver method
    public static void main(String args[])
    {
        int arr[] = {12, 34, 54, 2, 3};
        System.out.println("Array before sorting");
        printArray(arr);
        ShellSort ob = new ShellSort();
        ob.sort(arr);
        System.out.println("Array after sorting");
        printArray(arr);
    }
}
```

Quick sort:

```
// Java implementation of QuickSort
import java.io.*;
class GFG {
	// A utility function to swap two elements
	static void swap(int[] arr, int i, int j)
	{
		int temp = arr[i];
		arr[i] = arr[j];
		arr[j] = temp;
	}
	// This function takes last element as pivot,
	// places the pivot element at its correct position
	// in sorted array, and places all smaller to left
	// of pivot and all greater elements to right of pivot
	static int partition(int[] arr, int low, int high)
	{
		// Choosing the pivot
		int pivot = arr[high];
		// Index of smaller element and indicates
		// the right position of pivot found so far
		int i = (low - 1);

		for (int j = low; j <= high - 1; j++) {

			// If current element is smaller than the pivot
			if (arr[j] < pivot) {
```

```
                        // Increment index of smaller element
                        i++;
                        swap(arr, i, j);
                }
        }
        swap(arr, i + 1, high);
        return (i + 1);
}
// The main function that implements QuickSort
// arr[] --> Array to be sorted,
// low --> Starting index,
// high --> Ending index
static void quickSort(int[] arr, int low, int high)
{
        if (low < high) {
                // pi is partitioning index, arr[p]
                // is now at right place
                int pi = partition(arr, low, high);

                // Separately sort elements before
                // partition and after partition
                quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high);
        }
}
// To print sorted array
public static void printArr(int[] arr)
{
        for (int i = 0; i < arr.length; i++) {
```

```
                System.out.print(arr[i] + " ");
        }
    }
    // Driver Code
    public static void main(String[] args)
    {
        int[] arr = { 10, 7, 8, 9, 1, 5 };
        int N = arr.length;

        // Function call
        quickSort(arr, 0, N - 1);
        System.out.println("Sorted array:");
        printArr(arr);
    }
}
```
