**Selection sort**

// Moving small element to first

public static void selectionSort(int[] arr) {

for(int i=0;i<arr.length-1; i++) {

int minIndex=i;

for(int j=i+1; j<arr.length; j++) {

if(arr[j] < arr[minIndex]) minIndex=j;

}

swap(arr, i, minIndex);

}

Arrays.stream(arr).forEach(System.out::print);

}

// Moving large element to last

public static void selectionSort2(int[] arr) {

int lastUnsortedIndex = arr.length;

for(int i= lastUnsortedIndex-1; i>0; i--) {

int largest=0;

for(int j=0; j<=i; j++) {

if(arr[j]>arr[largest]) largest = j;

}

swap(arr, i, largest);

}

Arrays.stream(arr).forEach(System.out::print);

}

**Bubble Sort**

private static void bubbleSort(int[] arr) {

int length = arr.length;

for(int i= length-1; i>0; i--) {

for(int j=0; j<i; j++) {

if(arr[j]>arr[j+1]) swap(arr, j, j+1);

}

}

Arrays.stream(arr).forEach(System.out::print);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Merge Sort**

int[] array = new int[]{19, -0, 78, 0, 28, -33};

**mergeSort**(array, 0, array.length);

private static void **mergeSort**(int[] array, int start, int end) {

if(end - start < 2){

return;

}

int mid = (start+end)/2;

mergeSort(array, start, mid);

mergeSort(array, mid, end);

merge(array, start, mid, end);

}

private static void merge(int[] input, int start,int mid, int end) {

if(input[mid-1]<=input[mid]) return;

int i =start, j= mid, tempIndex=0;

int[] temp = new int[end-start];

while(i<mid && j<end){

temp[tempIndex++] = input[i]<input[j] ? input[i++] : input[j++];

}

System.arraycopy(input, i, input, start+tempIndex, mid-i);

System.arraycopy(temp, 0, input, start, tempIndex);

}

**Quick Sort**

int[] arr = new int[]{10, -9, 0 ,29, -44};

quickSort(arr, 0, arr.length);

private static void quickSort(int[] arr, int start, int end) {

if(end - start < 2) return;

int pivotIndex = partition(arr, start, end);

quickSort(arr, start, pivotIndex);

quickSort(arr, pivotIndex+1, end);

}

private static int partition(int[] arr, int start, int end) {

int pivot = arr[start];

int i = start, j= end;

while(i<j){

while(i<j && arr[--j] >= pivot) ;

if(i<j) arr[i]= arr[j];

while(i<j && arr[++i] <= pivot) ;

if(i<j) arr[j] = arr[i];

}

arr[j] = pivot;

return j;

}