**Iterative Algorithm: Merge Sort**

Please build and test a recursive version of the merge sort, and copy and paste it into the box below.

|  |
| --- |
| public static void mergeSort(int[] a) {  if (a.length <= 1) {  return;  }    int mid = a.length / 2; // find mid point  int[] leftArray = new int[mid]; // create the left side of the merge sort  int[] rightArray = new int[a.length - mid]; // create the right side of the merge sort    System.*arraycopy*(a, 0, leftArray, 0, mid); // copy the input array into each side    if (a.length - mid >= 0) {  System.*arraycopy*(a, mid, rightArray, 0, a.length - mid);  }    *mergeSort*(leftArray); // sort the left side  *mergeSort*(rightArray); // sort the right side  *merge*(a, leftArray, rightArray); // recur until length is less than or equal to 1  }  public static void merge(int[] a, int[] leftArray, int[] rightArray) {  int i=0, j=0, k=0;  while (i < leftArray.length && j < rightArray.length) { // shift numbers in ascending or  if (leftArray[i] <= rightArray[j]) {  a[k++] = leftArray[i++];  } else {  a[k++] = rightArray[j++];  }  }  while (i < leftArray.length) {  a[k++] = leftArray[i++];  }  while (j < rightArray.length) {  a[k++] = rightArray[j++];  }  } |

1. Please add developer comments throughout your methods, identifying the elements contributing to recursion (base case/ parameters/ loops/ etc.) Please briefly explain these elements in your comment.
2. In the box below, please provide a written description of how this sort moves through an array sorting it. This Description should highlight the elements of recursion, discussing what elements it changes in order to reuse the code in an efficient manner.

|  |
| --- |
| The array is split in half and assigned as separate, individual arrays. Then each side is sorted in ascending order through comparing each element with the next and swapping them if it is greater. This repeats for each merge through the function recurring until it reaches the end of the array. |

**Iterative Algorithm: Quick Sort**

Please build and test a recursive version quick sort, and copy and paste it into the box below.

|  |
| --- |
| static void swap(int[] arr, int i, int j)  {  int temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }    static int partition(int[] arr, int low, int high)  {  int pivot = arr[high]; // chooses pivot at the end index  int i = (low - 1); // sets index at start index    for (int j = low; j <= high - 1; j++) { // sorts through by swapping lower values for higher values  if (arr[j] < pivot) {  i++;  *swap*(arr, i, j);  }  }  *swap*(arr, i + 1, high);  return (i + 1);  }  static void quickSort(int[] arr, int low, int high)  {  if (low < high) { // splits array in half and sorts each side with the partition function  int pi = *partition*(arr, low, high);  *quickSort*(arr, low, pi - 1);  *quickSort*(arr, pi + 1, high);  }  } |

1. Please add developer comments throughout your methods, identifying the elements contributing to recursion specifically how the partition is determined and when it is used (base case/ parameters/ loops/ etc.). Please briefly explain these elements in your comment.
2. In the box below, please provide a written description of how this sort moves through an array sorting it. This Description should highlight the elements of recursion, discussing the partition value, and how it is determined, and then when it is used to separate the index values.

|  |
| --- |
| The quick sorts works similarly to the merge sort by dividing the array into two then sorting each side, but a pivot point is selected instead of using the middle of the array. |