**Iterative Algorithm: Merge Sort**

Below is a method which will sort an array of integers in ascending order using the Iterative Merge sort. Please copy this method into your own personal file to test it with the array, consisting of 50+ integers. (You can randomly roll these integers, but make sure you print the array unsorted first prior)

|  |
| --- |
| **public** **static** **void** mergeSort(**int**[] numList) {  **int** high = numList.length - 1;  **int** low = 0;    **int**[] temp = Arrays.copyOf(numList, numList.length); // copies array into a temporary array    **for** (**int** size = 1; size <= high - low; size = 2 \* size) { // while size is less than or equal to the highest element of the main array, double the size  **for** (**int** i = low; i < high; i += 2 \* size) {// while i is less than the highest element of the main array, add double of size to i  **int** begin = i;  **int** mid = i + size - 1;  **int** end = Integer.min(i + 2 \* size - 1, high);    merge(numList, temp, begin, mid, end); // apply the array, the temporary array, the middle of the array and the end of the array to the merge sort  }  }  }    **public** **static** **void** merge(**int**[] numList, **int**[] temp, **int** start, **int** mid, **int** end) {  **int** k = start, i = start, j = mid + 1;    // splitting into 2 parts to divide and conquer  **while** (i <= mid && j <= end) { // from the start to the middle of the array (first segment) check if the main array has an element of index i that is less than the main array of index j  **if**(numList[i] < numList[j]) { // first segment sort  temp[k++] = numList[i++];  }  **else** {  temp[k++] = numList[j++];  }  }    **while** (i < numList.length && i <= mid) { // while the index is less than the last element of the main array, switch the places of the temporary array and main array  temp[k++] = numList[i++]; // second segment sort  }    **for** (i = start; i <= end; i++) { // third segment sort  numList[i] = temp[i];  }  } |

1. Please add developer comments at the points indicated with a **“//”**. These comments should briefly outline what each line of code is for, and what it accomplishes.
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 process, it does not need to outline every single adjustment that is made.

|  |
| --- |
| The array is split in half and the indexes of the start, middle and end are stored to variables. And the each index is compared to the next and swapped if the lower index is greater than the higher index. |

1. Can you please the adjustments necessary to sort the array in descending order instead of ascending order. Please highlight your adjustments in **red bold text.** (can be done in original box)

|  |
| --- |
| **public** **static** **void** mergeSort(**int**[] numList) {  **int** high = numList.length - 1;  **int** low = 0;    **int**[] temp = Arrays.copyOf(numList, numList.length); // copies array into a temporary array    **for** (**int** size = 1; size <= high - low; size = 2 \* size) { // while size is less than or equal to the highest element of the main array, double the size  **for** (**int** i = low; i < high; i += 2 \* size) {// while i is less than the highest element of the main array, add double of size to i  **int** begin = i;  **int** mid = i + size - 1;  **int** end = Integer.min(i + 2 \* size - 1, high);    merge(numList, temp, begin, mid, end); // apply the array, the temporary array, the middle of the array and the end of the array to the merge sort  }  }  }    **public** **static** **void** merge(**int**[] numList, **int**[] temp, **int** start, **int** mid, **int** end) {  **int** k = start, i = start, j = mid + 1;    // splitting into 2 parts to divide and conquer  **while** (i <= mid && j <= end) { // from the start to the middle of the array (first segment) check if the main array has an element of index i that is less than the main array of index j  **if**(numList[i] **>** numList[j]) { // switch places of the indexes  temp[k++] = numList[i++];  }  **else** {  temp[k++] = numList[j++];  }  }    **while** (i < numList.length && i <= mid) { // while the index is less than the last element of the main array, switch the places of the temporary array and main array  temp[k++] = numList[i++]; //  }    **for** (i = start; i <= end; i++) { //  numList[i] = temp[i];  }  } |