Assignment – `mergeSort()`

©2025 Chris Nielsen – www.nielsenedu.com

1. Complete the methods in order to implement a recursive mergeSort algorithm.
   Once complete, show your code and be able to answer the question below.

```
1  public static void mergeSort(int[] arr) {
2     if (arr == null || arr.length <= 1) {
3        return; // Already sorted
4     }
5     int[] scratch = new int[arr.length];
6     mergeSort(arr, 0, arr.length - 1, scratch);
7  }
.  private static void mergeSort(int[] arr, int low, int high,
.                                int[] scratch) {
.     // Implement this method
.  }
   private static void merge(int[] arr, int low, int mid, int high,
                             int[] scratch) {
      // Implement this method
   }
```

   The temporary array, `scratch`, is allocated in the one-parameter `mergeSort` method. It is passed as a parameter to the `mergeSort` method, which then passes it as a parameter to the `merge` method. The array variable is not used in either `mergeSort` method, but is only used in the `merge` method. If the array were to be created within the `merge` method, rather than within the single-parameter `mergeSort` method, the `scratch` parameter could be removed from both the four-parameter `mergeSort` method and the `merge` method.

   What advantage does the current implementation, with `scratch` allocated in the one-parameter `mergeSort` method, have over the alternative method described immediately above?

Assignment – `mergeSort()`

©2025 Chris Nielsen – www.nielsenedu.com

It is computationally expensive (it takes time) to repeatedly allocate a new array. The current implementation needs only allocate the `scratch` ***array one time.***

```
public static void mergeSort(int[] arr) {
   if (arr == null || arr.length <= 1) {
      return; // Already sorted
   }
   int[] scratch = new int[arr.length];
   mergeSort(arr, 0, arr.length - 1, scratch);
}
private static void mergeSort(int[] arr,
                              int low, int high,
                              int[] scratch) {
   if (low < high) {
      int mid = low + (high - low) / 2;
      mergeSort(arr, low, mid, scratch);
      mergeSort(arr, mid + 1, high, scratch);
      merge(arr, low, mid, high, scratch);
   }
}
private static void merge(int[] arr,
                          int low, int mid, int high,
                          int[] scratch) {
   for (int i = low; i <= high; i++) {
      scratch[i] = arr[i];
   }
   int i = low, j = mid + 1, k = 0;
   while (i <= mid && j <= high) {
      if (scratch[i] < scratch[j]) {
         arr[k++] = scratch[i++];
      } else {
         arr[k++] = scratch[j++];
      }
   }
   while (i <= mid) {
      arr[k++] = scratch[i++];
   }
}
```