

## Merge Sort

The Merge Sort Algorithm below sorts an array of integers into ascending order as follows:

### **mergeSort**

This top-level method creates the necessary temporary array and calls the `mergeSortHelper` recursive helper method.

### **mergeSortHelper**

This recursive helper method uses the Merge Sort Algorithm to sort `elements[from] ... elements[to]` inclusive into ascending order:

1. If there is more than one item in this range,
  - a. divide the items into two adjacent parts, and
  - b. call `mergeSortHelper` to recursively sort each part, and
  - c. call the `merge` helper method to merge the two parts into sorted order.
2. Otherwise, exit because these items are sorted.

### **merge**

This helper method merges two adjacent array parts, each of which has been sorted into ascending order, into one array part that is sorted into ascending order:

1. As long as both array parts have at least one item that hasn't been copied, compare the first un-copied item in each part and copy the minimal item to the next position in `temp`.
2. Copy any remaining items of the first part to `temp`.
3. Copy any remaining items of the second part to `temp`.
4. Copy the items from `temp[from] ... temp[to]` inclusive to the respective locations in `elements`.

```
/**
 * Sort an array of integers into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 *
 * Postcondition: elements contains its original items and items in elements
 *                  are sorted in ascending order.
 */
public static void mergeSort(int[] elements)
{
    int n = elements.length;
    int[] temp = new int[n];
    mergeSortHelper(elements, 0, n - 1, temp);
}
```

```

/**
 * Sorts elements[from] ... elements[to] inclusive into ascending order.
 *
 * @param elements an array containing the items to be sorted.
 * @param from the beginning index of the items in elements to be sorted.
 * @param to the ending index of the items in elements to be sorted.
 * @param temp a temporary array to use during the merge process.
 *
 * Precondition:
 * (elements.length == 0 or
 *  0 <= from <= to <= elements.length) and
 * elements.length == temp.length
 * Postcondition: elements contains its original items and the items in elements
 * [from] ... <= elements[to] are sorted in ascending order.
 */
private static void mergeSortHelper(int[] elements,
                                     int from, int to, int[] temp)
{
    if (from < to)
    {
        int middle = (from + to) / 2;
        mergeSortHelper(elements, from, middle, temp);
        mergeSortHelper(elements, middle + 1, to, temp);
        merge(elements, from, middle, to, temp);
    }
}

```

```

/**
 * Merges two adjacent array parts, each of which has been sorted into ascending
 * order, into one array part that is sorted into ascending order.
 *
 * @param elements an array containing the parts to be merged.
 * @param from the beginning index in elements of the first part.
 * @param mid the ending index in elements of the first part.
 *             mid+1 is the beginning index in elements of the second part.
 * @param to the ending index in elements of the second part.
 * @param temp a temporary array to use during the merge process.
 *
 * Precondition: 0 <= from <= mid <= to <= elements.length and
 * elements[from] ... <= elements[mid] are sorted in ascending order and
 * elements[mid + 1] ... <= elements[to] are sorted in ascending order and
 * elements.length == temp.length
 * Postcondition: elements contains its original items and
 * elements[from] ... <= elements[to] are sorted in ascending order and
 * elements[0] ... elements[from - 1] are in original order and
 * elements[to + 1] ... elements[elements.length - 1] are in original order.
 */
private static void merge(int[] elements,
                          int from, int mid, int to, int[] temp)
{
    int i = from;
    int j = mid + 1;
    int k = from;

    while (i <= mid && j <= to)
    {
        if (elements[i] < elements[j])
        {
            temp[k] = elements[i];
            i++;
        }
        else
        {
            temp[k] = elements[j];
            j++;
        }
        k++;
    }
}

```

```

while (i <= mid)
{
    temp[k] = elements[i];
    i++;
    k++;
}

while (j <= to)
{
    temp[k] = elements[j];
    j++;
    k++;
}

for (k = from; k <= to; k++)
{
    elements[k] = temp[k];
}
}

```