Merge Algorithm

Given:
- **list**, an array split into two segments each of which are sorted.
  - The top segment consists of items \( \text{list[topFirst]} \) to \( \text{list[topLast]} \).
  - The bottom segment consists of items \( \text{list[bottomFirst]} \) to \( \text{list[bottomLast]} \).
- **aux**, an auxiliary array in which to merge the items.

Steps:
- **start** at the first items in the top segment, the bottom segment and **aux**
- **while** haven’t reached the end of a segment **do**
  - compare the current item in the top segment to that in the bottom segment
  - move the smaller of the two into the auxiliary array
  - increment the appropriate position variables
- **end while**
- whichever segment hasn’t ended
  - move the remaining items into the auxiliary array
- **copy** the entire auxiliary array back into **list**
public void merge( double [] aux, double [] list,
     int topFirst, int topLast,
     int bottomFirst, int bottomLast )
// Precondition: aux is not null AND list is not null
//    AND list[topFirst] to list[topLast] are in order
//    AND list[bottomFirst] to list[bottomLast] are in order.
// Postcondition:
//    list[topFirst] to list[bottomLast] are in order.
{
    int auxPos = topFirst;  // position within auxiliary array
    int top = topFirst;     // position within top segment
    int bottom = bottomFirst;  // position within bottom segment
    // copy until reaching the end of one segment
    while( top <= topLast && bottom <= bottomLast )
    {
        // copy smaller item to auxiliary array and
        // increment the appropriate position variables
        if ( list[top] < list[bottom] )
            aux[auxPos++] = list[top++];
        else
            aux[auxPos++] = list[bottom++];
    }
    // which segment ended?
    if ( top <= topLast ) // bottom segment
        // copy remainder of top segment to aux
        while ( top <= topLast )
            aux[auxPos++] = list[top++];
    else // top segment
        // copy remainder of bottom segment to aux
        while ( bottom <= bottomLast )
            aux[auxPos++] = list[bottom++];
    // copy aux back into the primary array
    for ( int k = topFirst; k <= bottomLast; k++ )
        list[k] = aux[k];
}
Recursive Mergesort Algorithm

Given:
- `list`, an array.
- `first` and `last`, the items to be sorted are `list[first]` to `list[last]`.
- `aux`, an auxiliary array.

Steps:
- If `first == last` then
  - there's only one item to be sorted so return
- else
  - calculate the midpoint between `first` and `last`
  - call `mergesort` recursively to sort the segment above the midpoint
  - call `mergesort` recursively to sort the segment below the midpoint
  - call `merge` to merge the two sorted segments
- end if

```java
public void recmergesort( double [] aux, double [] list,
                          int first, int last )

// Precondition: aux is not null AND list is not null
//               AND first <= last.
// Postcondition: list[first] to list[last] are in order.
{
    if( first == last ) // there's only 1 item
        return;         // nothing to sort
    else
        {
            // find middle of segment
            int mid = ( first + last ) / 2;
            // sort list[first] to list[mid]
            recmergesort( aux, list, first, mid );
            // sort list[mid+1] to list[last]
            recmergesort( aux, list, mid+1, last );
            // merge the sorted halves
            merge( aux, list, first, mid, mid+1, last );
        }
}
```
public void mergesort( double [] list )
// Precondition: list is not null.
// Postcondition: list is in ascending order.
{
    // create auxiliary list
    double [] aux = new double[list.length];
    // perform recursive mergesort
    recmergesort( aux, list, 0, list.length-1 );
}

Example
Trace the execution of the Mergesort algorithm on the array given below.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>21</td>
<td>33</td>
<td>70</td>
<td>12</td>
<td>85</td>
<td>44</td>
<td>3</td>
<td>67</td>
<td>24</td>
<td>51</td>
<td>40</td>
</tr>
</tbody>
</table>

Show the recursion tree generated by the recursive calls to Mergesort and the contents of the array after each call to the merge method.
**Legend**

- **0,11 5** represents call to `recmergesort` with `first = 0, last = 11` and `mid = 5`
- **0,0** represents call to `recmergesort` with `first = 0, last = 0` and no further recursion
- **3** represents call to `merge`
Original array:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64</td>
<td>21</td>
<td>33</td>
<td>70</td>
<td>12</td>
<td>85</td>
<td>44</td>
<td>3</td>
<td>67</td>
<td>24</td>
<td>51</td>
<td>40</td>
</tr>
</tbody>
</table>

Sequence of merges made during execution of Mergesort:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>21</td>
<td>64</td>
<td>33</td>
<td>70</td>
<td>12</td>
<td>85</td>
<td>44</td>
<td>3</td>
<td>67</td>
<td>24</td>
<td>51</td>
</tr>
<tr>
<td>b</td>
<td>21</td>
<td>33</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>70</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td>70</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>12</td>
<td>21</td>
<td>33</td>
<td>64</td>
<td>70</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>44</td>
<td>67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
<td>51</td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
<td>40</td>
</tr>
<tr>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>k</td>
<td>3</td>
<td>12</td>
<td>21</td>
<td>24</td>
<td>33</td>
<td>40</td>
<td>44</td>
<td>51</td>
<td>64</td>
<td>67</td>
<td>70</td>
</tr>
</tbody>
</table>