

a_2 that have already been copied into a . When we have copied either all components of a_1 or all components of a_2 , we complete the merge by copying all the remaining components of the other array into a .

Algorithm 3.23 captures this idea. Two points in the algorithm are particularly noteworthy. Firstly, the conditions in steps 2.1 and 2.2 are not mutually exclusive: if $a_1[i]$ is equal to $a_2[j]$, it does not matter whether we perform step 2.1.1 or 2.2.1. Secondly, only one of the loops at steps 3 and 4 will actually do any work: after the loop at step 2 terminates, one of the conditions $i \leq \text{right}_1$ and $j \leq \text{right}_2$ will no longer be satisfied.

Figure 3.24 shows the loop invariant for the merging algorithm. (Interestingly, all three loops have the same invariant.) Figure 3.25 illustrates the algorithm's behavior as it merges two arrays of words.

Let us analyze the merging algorithm's time efficiency. Let $n_1 = \text{right}_1 - \text{left}_1 + 1$ be the length of $a_1[\text{left}_1 \dots \text{right}_1]$, let $n_2 = \text{right}_2 - \text{left}_2 + 1$ be the length of $a_2[\text{left}_2 \dots \text{right}_2]$, and let $n = n_1 + n_2$ be the total number of merged components. For the characteristic operations we could choose either copies or comparisons.

Let us first analyze the number of copies. Each component of a_1 is copied exactly once, by either step 2.1.1 or step 3.1. Likewise, each component of a_2 is copied exactly once, by either step 2.2.1 or step 4.1. Therefore:

$$\text{No. of copies} = n_1 + n_2 = n \quad (3.6)$$

Let us now analyze the number of comparisons. The loop at step 2 is iterated at most $n - 1$ times. If we assume that steps 2.1 and 2.2 can be implemented with a single comparison, then:

```

static void merge
    (Comparable[] a1, int left1, int right1,
     Comparable[] a2, int left2, int right2,
     Comparable[] a3, int left3) {
    // Merge a1[left1...right1] and a2[left2...right2] into
    // a3[left3...] (where both a1 and a2 are sorted).
    int i = left1, j = left2, k = left3;
    while (i <= right1 && j <= right2) {
        int comp = a1[i].compareTo(a2[j]);
        if (comp <= 0)
            a3[k++] = a1[i++];
        else
            a3[k++] = a2[j++];
    }
    while (i <= right1)
        a3[k++] = a1[i++];
    while (j <= right2)
        a3[k++] = a2[j++];
}

```

Program 3.26 Implementation of the array merging algorithm as a Java method.