


**ROYAL UNIVERSITY OF BHUTAN
GYALPOZHING COLLEGE OF INFORMATION TECHNOLOGY
GYALPOZHING : BHUTAN**

MERGESORT

Idea:

1. Divide the unsorted list into N sublists, each containing 1 element.
2. Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. N will now convert into $N/2$ lists of size 2.
3. Repeat the process till a single sorted list of obtained.

Program of MergeSort in JAVA

A screenshot of a Java code editor window titled 'Merge.java'. The code implements the MergeSort algorithm. It starts with a public class 'Merge' containing two static methods. The 'merge' method takes an array 'a', an auxiliary array 'aux', and indices 'lo', 'mid', and 'hi'. It first copies the elements from 'a' to 'aux' in the range [lo, hi]. Then, it merges the two sorted sub-arrays 'aux[lo..mid]' and 'aux[mid+1..hi]' back into 'a'. The 'sort' method is a recursive function that sorts the array 'a' in the range [lo, hi] by repeatedly dividing it into two halves, sorting each half, and then merging them. Line numbers 1 through 30 are visible on the left side of the editor.

```
1 public class Merge {
2
3     // stably merge a[lo .. mid] with a[mid+1 ..hi] using aux[lo .. hi]
4     private static void merge(int[] a, int[] aux, int lo, int mid, int hi) {
5         for (int k = lo; k <= hi; k++) {
6             aux[k] = a[k];
7         }
8
9         // merge back to a[]
10        int i = lo, j = mid+1;
11        for (int k = lo; k <= hi; k++) {
12            if (i > mid) a[k] = aux[j++];
13            else if (j > hi) a[k] = aux[i++];
14            else if (less(aux[j], aux[i])) a[k] = aux[j++];
15            else a[k] = aux[i++];
16        }
17    }
18
19    // mergesort a[lo..hi] using auxiliary array aux[lo..hi]
20    private static void sort(int[] a, int[] aux, int lo, int hi) {
21        if (hi <= lo) return;
22        int mid = lo + (hi - lo) / 2;
23        //System.out.println(mid);
24        sort(a, aux, lo, mid);
25        //System.out.println("left");
26        sort(a, aux, mid + 1, hi);
27        //System.out.println("hi"+mid+1);
28        merge(a, aux, lo, mid, hi);
29        //System.out.println("Merge");
30    }
```

Figure 1: code 1

```

31
32
33     /**
34     * Rearranges the array in ascending order, using the natural order.
35     * @param a the array to be sorted
36     */
37     public static void sort(int[] a) {
38         int[] aux = new int[a.length];
39         sort(a, aux, 0, a.length-1);
40     }
41
42     /**
43     * Helper sorting function.
44     */
45     // is v < w ?
46     private static boolean less(Comparable v, Comparable w) {
47         return v.compareTo(w) < 0;
48     }
49     // print array to standard output
50     private static void show(int[] a) {
51         for (int i = 0; i < a.length; i++) {
52             System.out.print(" "+a[i]);
53         }
54         System.out.println();
55     }
56     /**
57     * Reads in a sequence of strings from standard input; mergesorts them;
58     * and prints them to standard output in ascending order.
59     *
60     * @param args the command-line arguments
61     */
62     public static void main(String[] args) {
63         int[] a = {9,7,8,3,2,1};
64         Merge.sort(a);
65         show(a);
66     }
67 }

```

Line 29, Column 11

Figure 2: continue code 1

Merge Sort

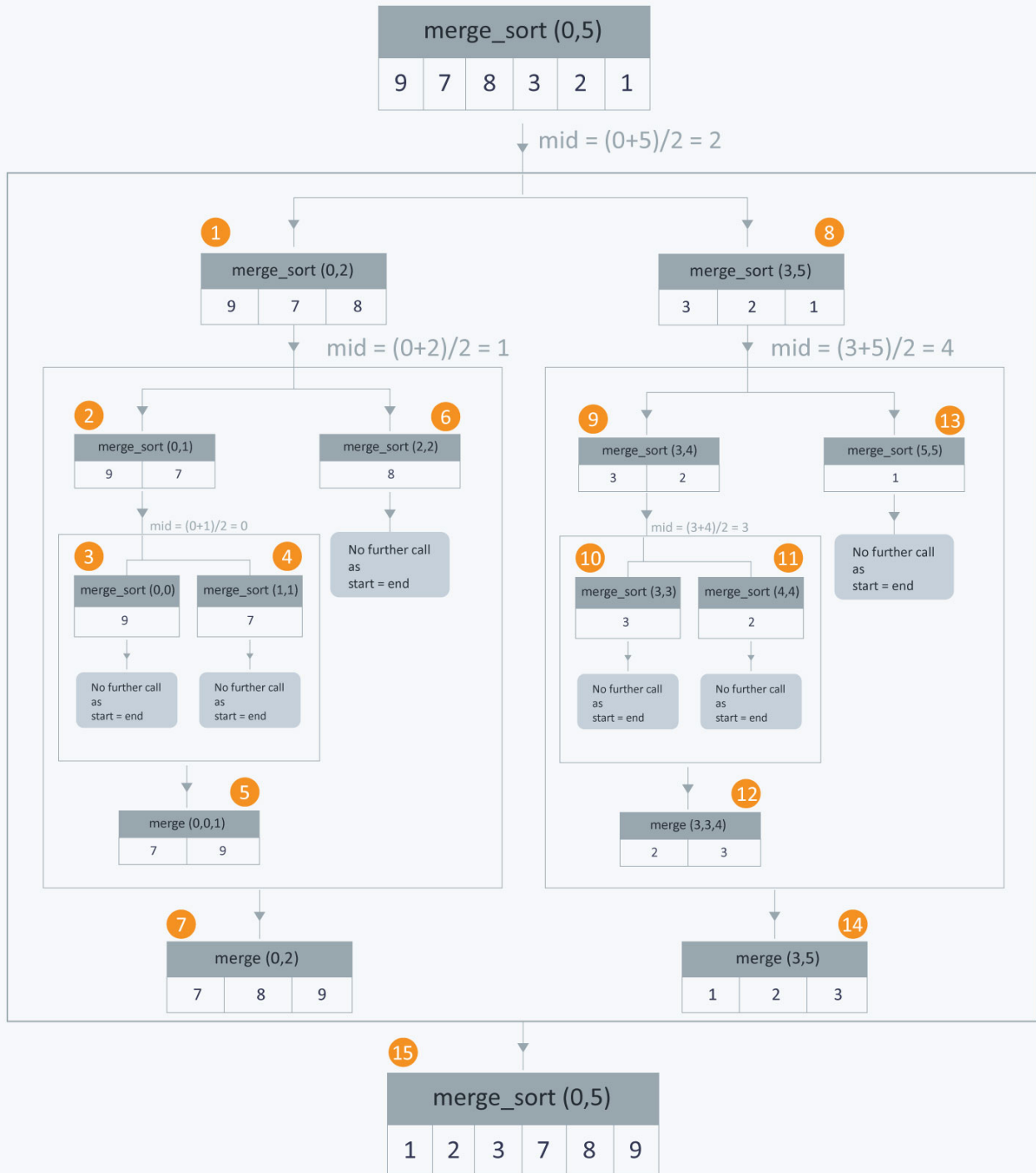


Figure 3: Visualization
Page 4