MERGE SORT

RECURSION + MERGE SORT ALGORITHM

References:
http://algs4.cs.princeton.edu/home/
http://homepages.math.uic.edu/~leon/cs-mcs401-s08/handouts/quicksort.pdf

RECURSIVE METHODS

Properties of good recursive methods

- **Identify a base case:**
  - When should the recursion stop?
  - The base case is a condition that makes the method return without calling itself.
- **Method arguments**
  - Recursive methods must have at least one argument as a means of communication.
- **Recursive calls must lead to the base case.**
  - Each call to the same function must modify the argument such that reaching the base case is guaranteed.

RECURSIVE ALGORITHMS

How to write a recursive algorithm:

- **Break down the original problem into smaller parts**
  - The parts should be of the same type as the original problem.
- **Identify the cases where the problem can be solved directly:**
  - When it can’t get any simpler
- **Combine the solutions of the smaller problems.**
  - This should be the solution to the original problem

MERGE SORT

- **Divide the array into two equal subarrays**
- **Recursively sort each subarray**
- **Merge the two sorted subarrays into one sorted array.**
  - Use an auxiliary array to hold the elements
  - Keep track of index in subarray 1 and 2 with i and j respectively
  - Compare elements at index i & j
  - Copy smaller element to the array at index k

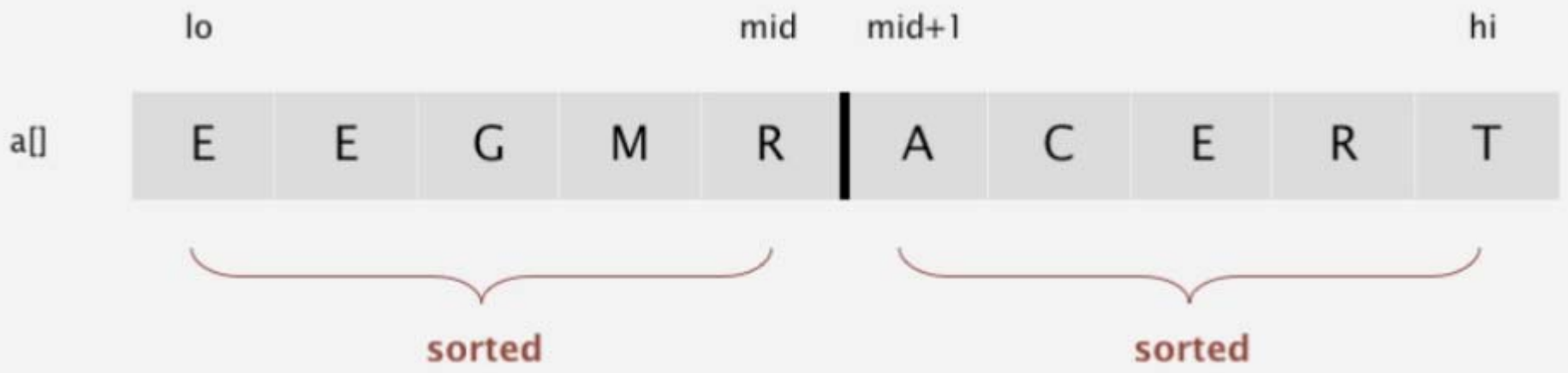
MERGE SORT

```
private static void mergeSort(int[] data, int lo, int hi) {
        if (lo<hi){
                //find index of middle element
                int mid=(lo+hi)/2;
                //recursive calls for first and second half of the array
                mergeSort(data, lo, mid);
                mergeSort(data, mid+1, hi);
                //merge the elements of each side of the array
                merge(data, lo, mid, hi);
        }
}
```

MERGE METHOD

```
Merge (data, lo, mid, hi)
Create a temporary array
Set i=lo, and j=mid+1
Loop while (i<=mid and j<=hi)
        compare elements at i and j
        copy smaller element to temp and increment i or j
while (i<=mid)
        copy element at i to temp, increment i
While (j<=hi)
        copy element at j to temp, increment j
Copy elements from temp back to data array
```

MERGE DEMO



MERGE DEMO

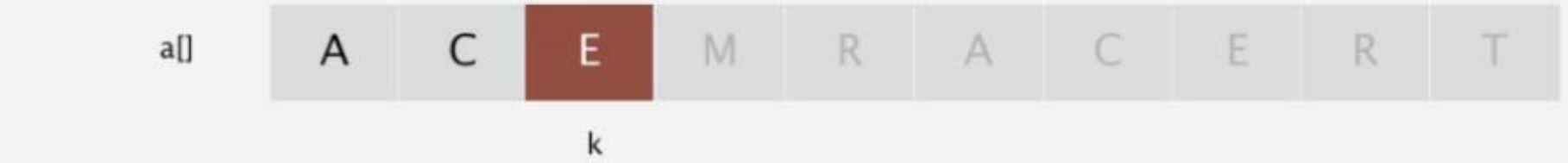
MERGE DEMO

compare minimum in each subarray

MERGE DEMO

compare minimum in each subarray

MERGE DEMO



compare minimum in each subarray

MERGE DEMO

compare minimum in each subarray

MERGE DEMO


a[]
A C E E E A C E R T
k
compare minimum in each subarray
aux[]
E E G M R A C E R T
i
j

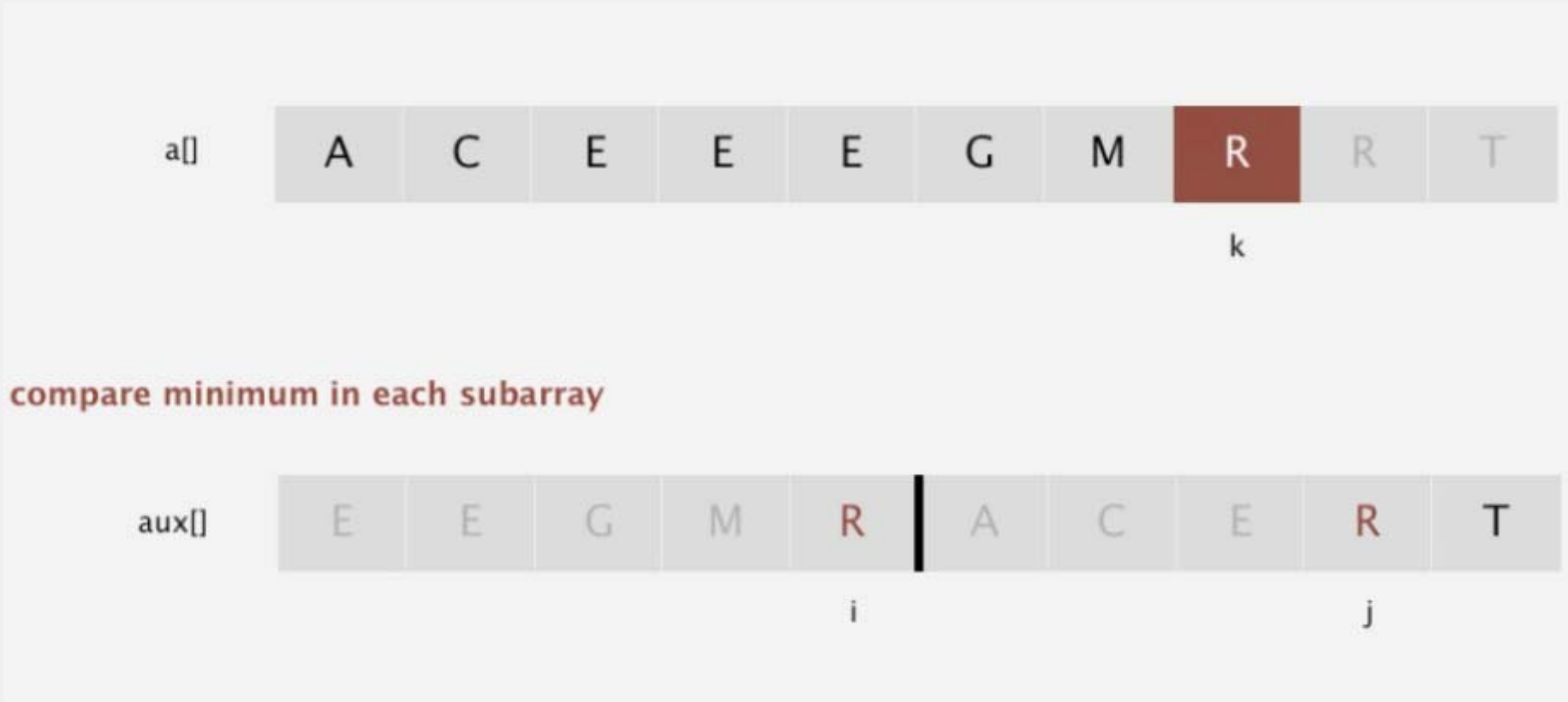
MERGE DEMO

compare minimum in each subarray

MERGE DEMO

compare minimum in each subarray

MERGE DEMO


a[]
A C E E E G M R R T
k
compare minimum in each subarray
aux[]
E E G M R A C E R T
i
j

MERGE DEMO

one subarray exhausted, take from other

MERGE METHOD

```
Merge (data, lo, mid, hi)
Create a temporary array
Set i=lo, and j=mid+1
Loop while (i<=mid and j<=hi)
        compare elements at i and j
        copy smaller element to temp and increment i or j
while (i<=mid)
        copy element at i to temp, increment i
While (j<=hi)
        copy element at j to temp, increment j
Copy elements from temp back to data array
```

MERGE DEMO

one subarray exhausted, take from other

MERGE DEMO

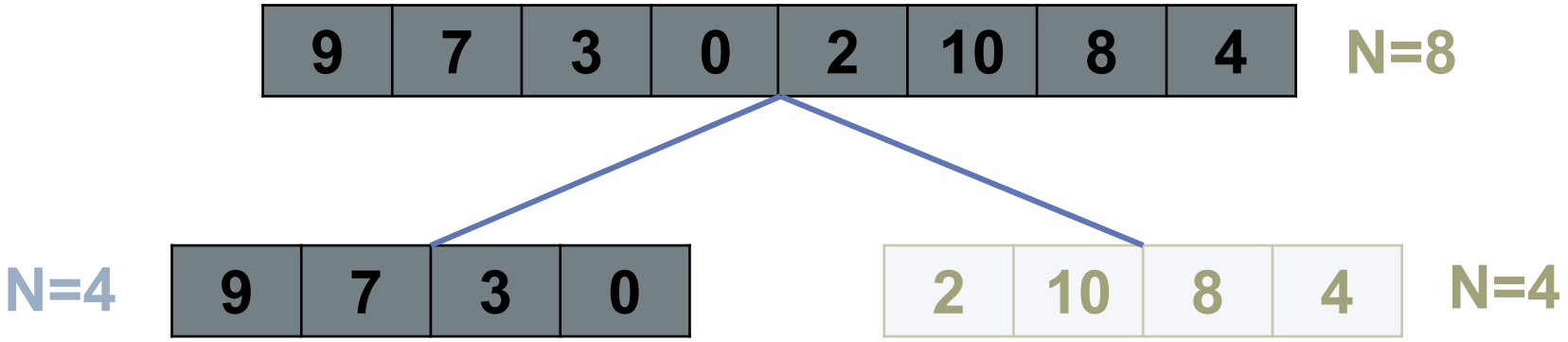

a[]
A C E E E G M R R T
k
one subarray exhausted, take from other
aux[]
E E G M R A C E R T
i
j

MERGE SORT

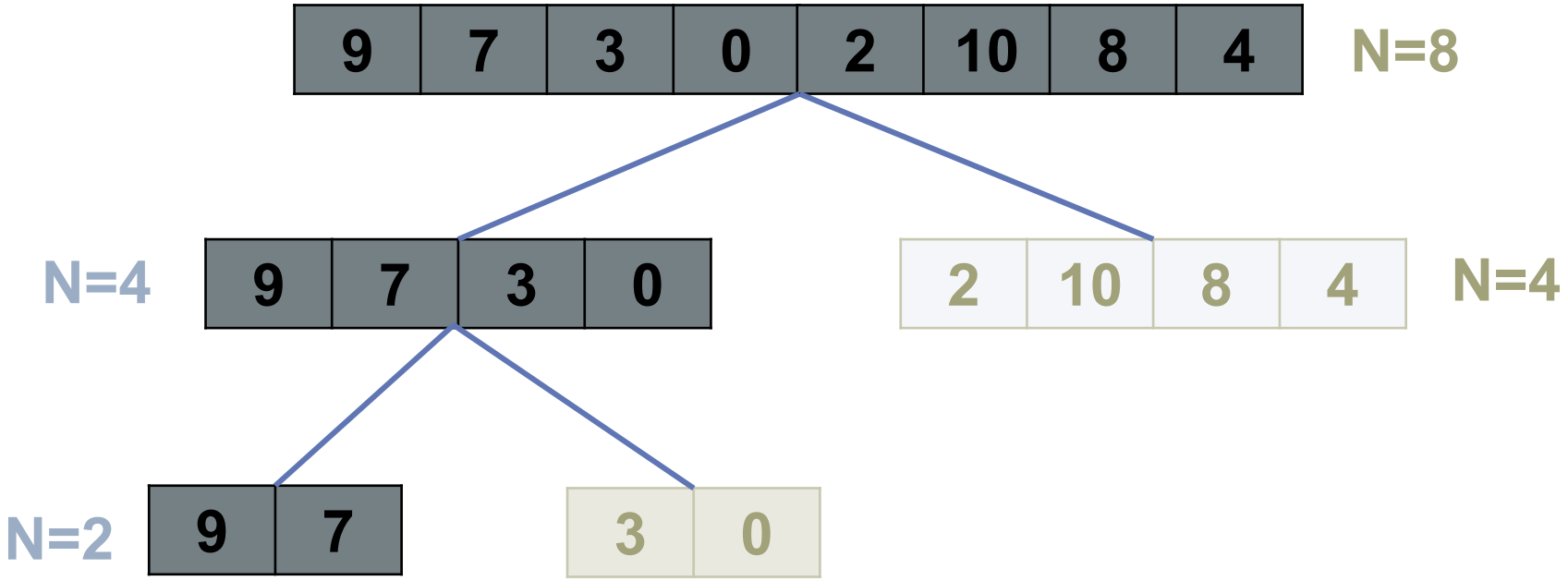
```
private static void mergeSort(int[] data, int lo, int hi) {
        if (lo<hi){
                //find index of middle element
                int mid=(lo+hi)/2;
                //recursive calls for first and second half of the array
                mergeSort(data, lo, mid);
                mergeSort(data, mid+1, hi);
                //merge the elements of each side of the array
                merge(data, lo, mid, hi);
        }
}
```

MERGE SORT

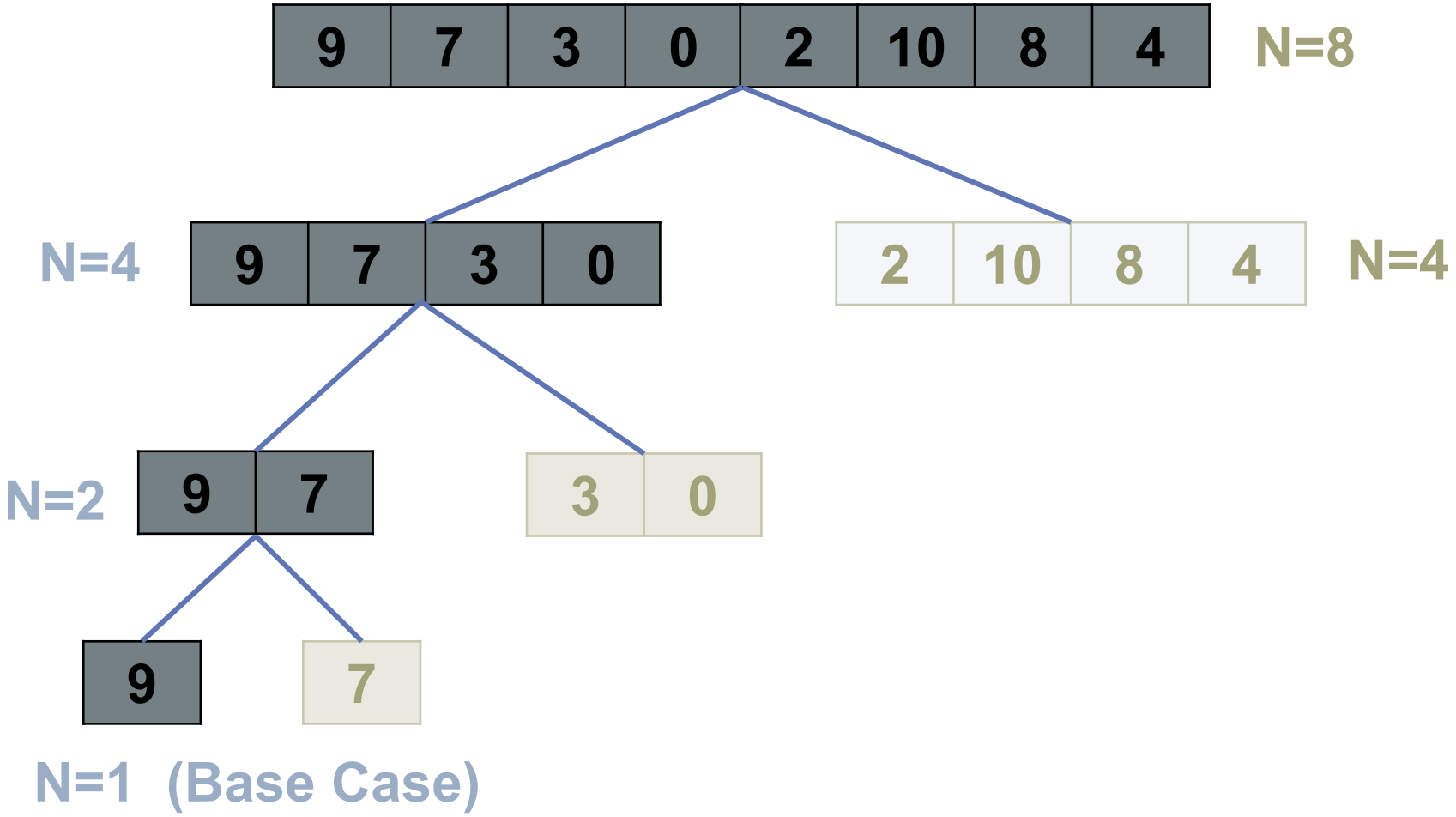
MERGE SORT



MERGE SORT



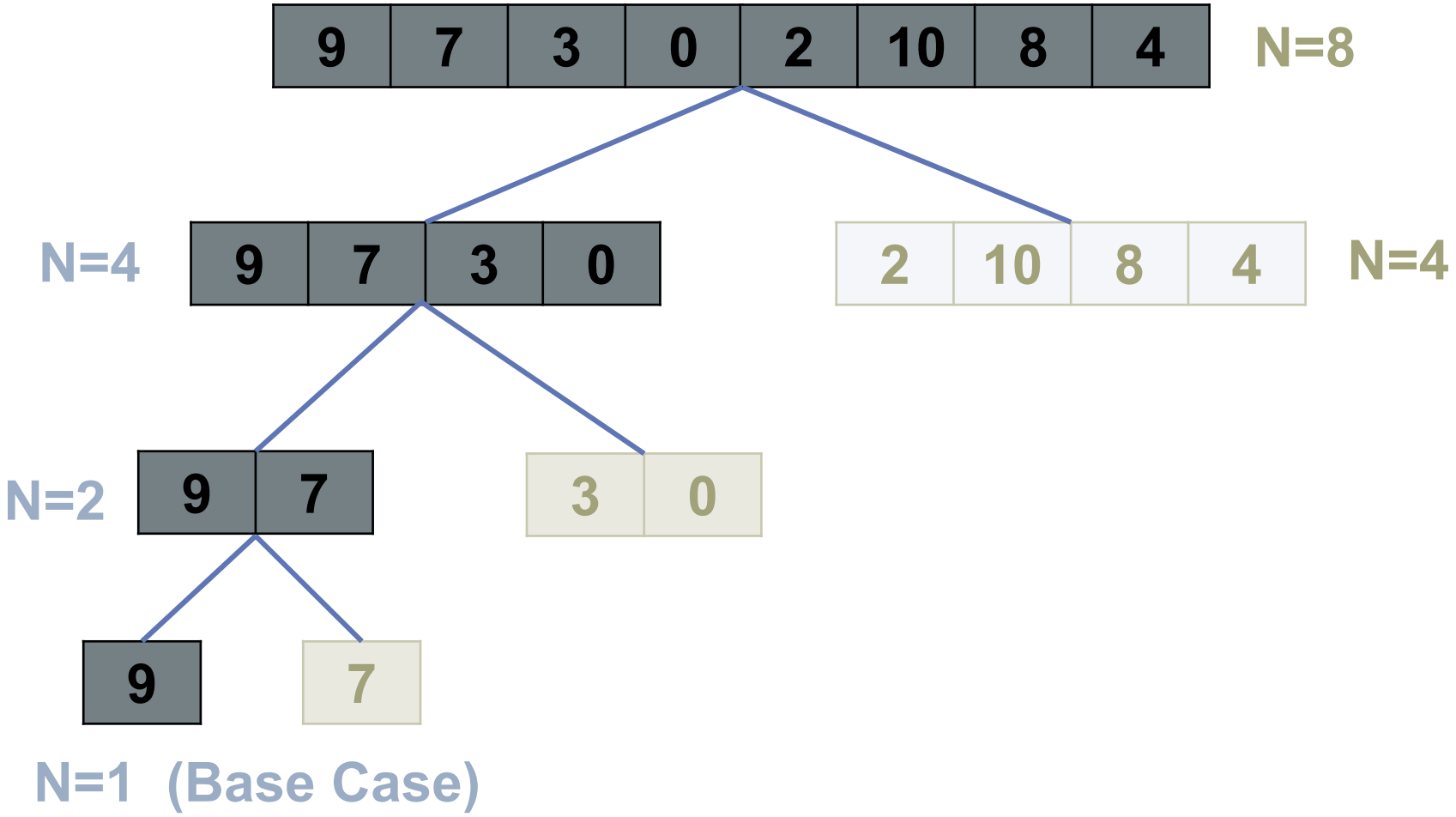
MERGE SORT



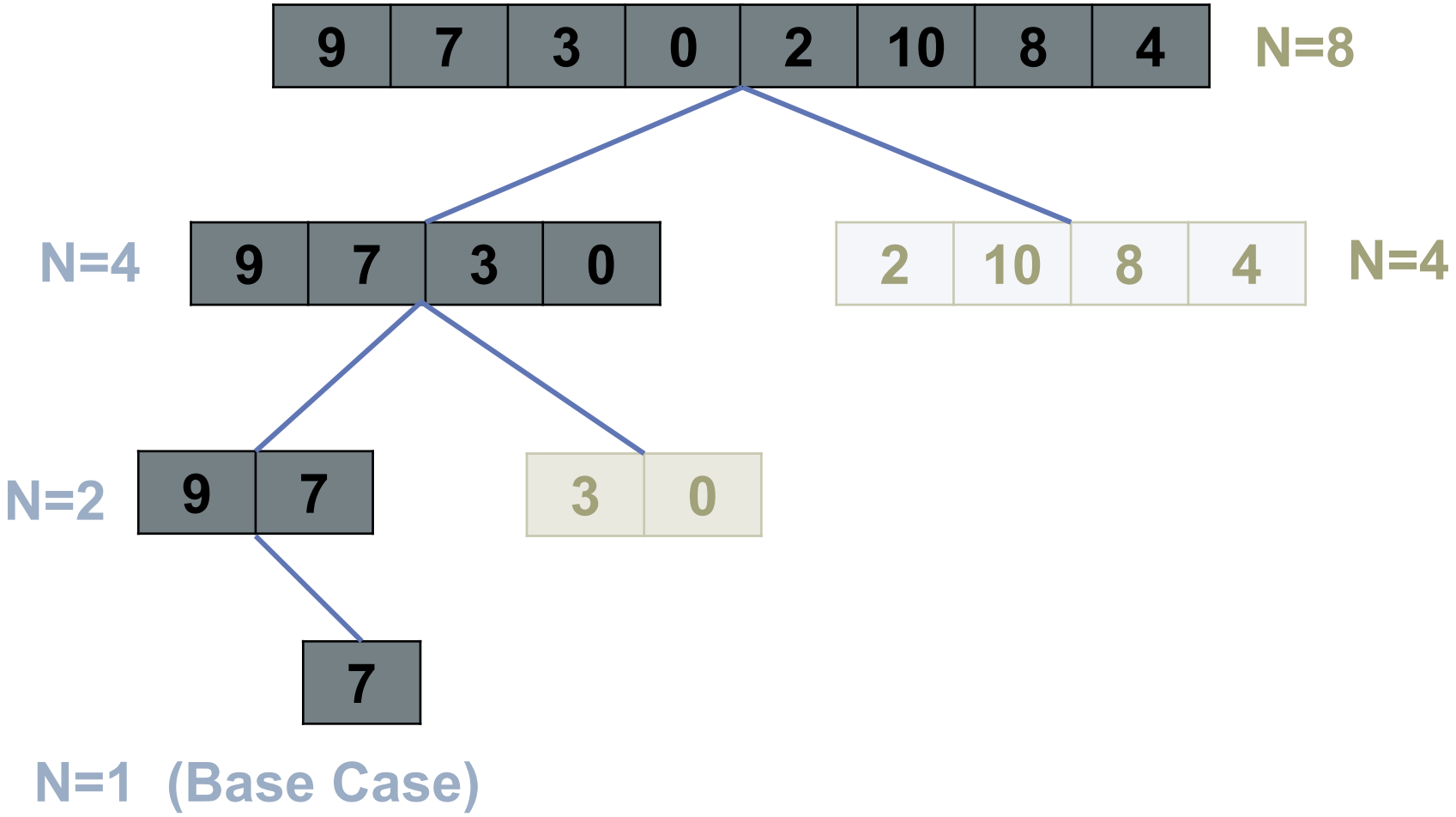
MERGE SORT

```
private static void mergeSort(int[] data, int lo, int hi) {
        if (lo<hi){
                //find index of middle element
                int mid=(lo+hi)/2;
                //recursive calls for first and second half of the array
                mergeSort(data, lo, mid);
                mergeSort(data, mid+1, hi);
                //merge the elements of each side of the array
                merge(data, lo, mid, hi);
        }
}
```

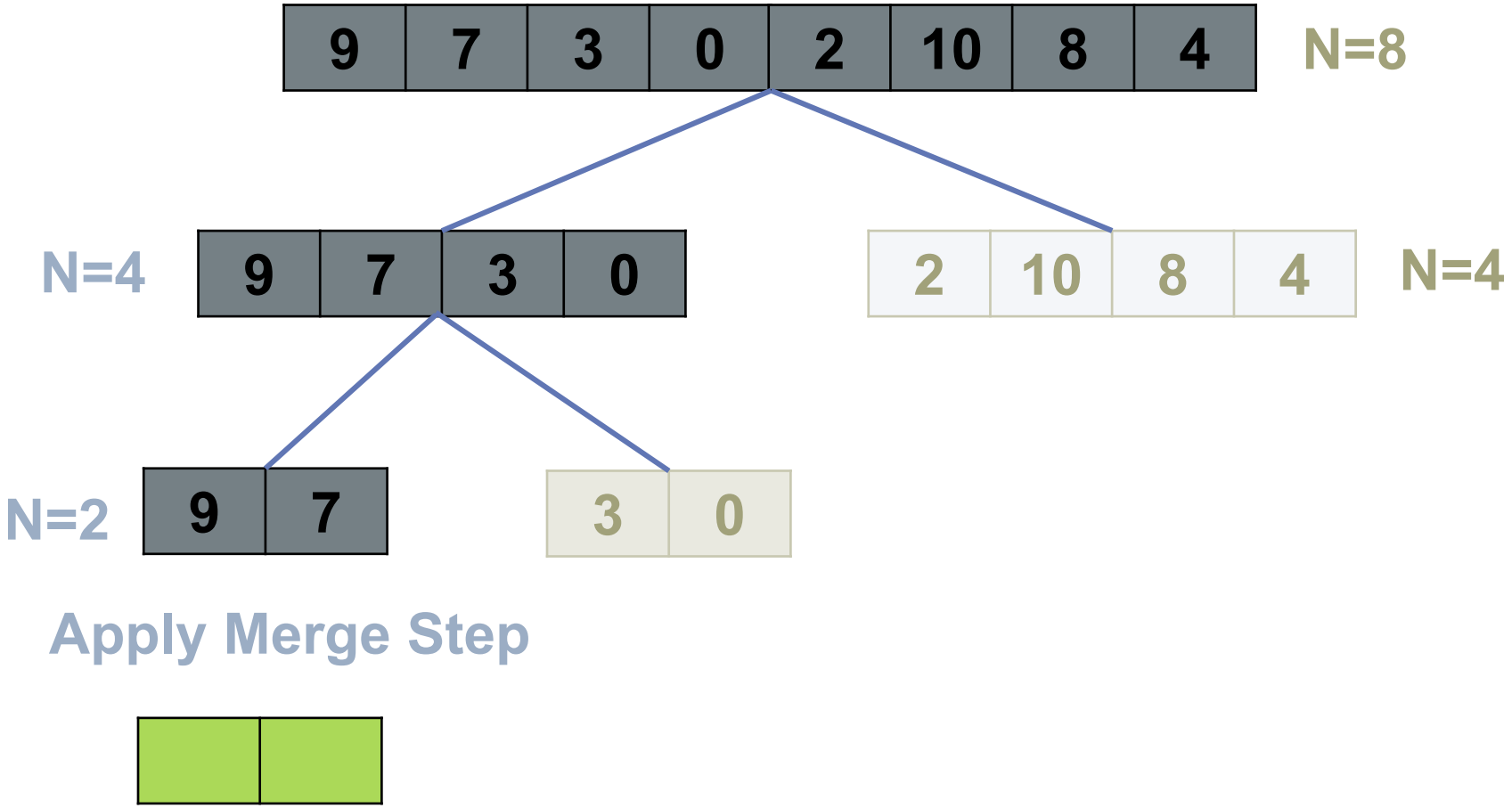
MERGE SORT



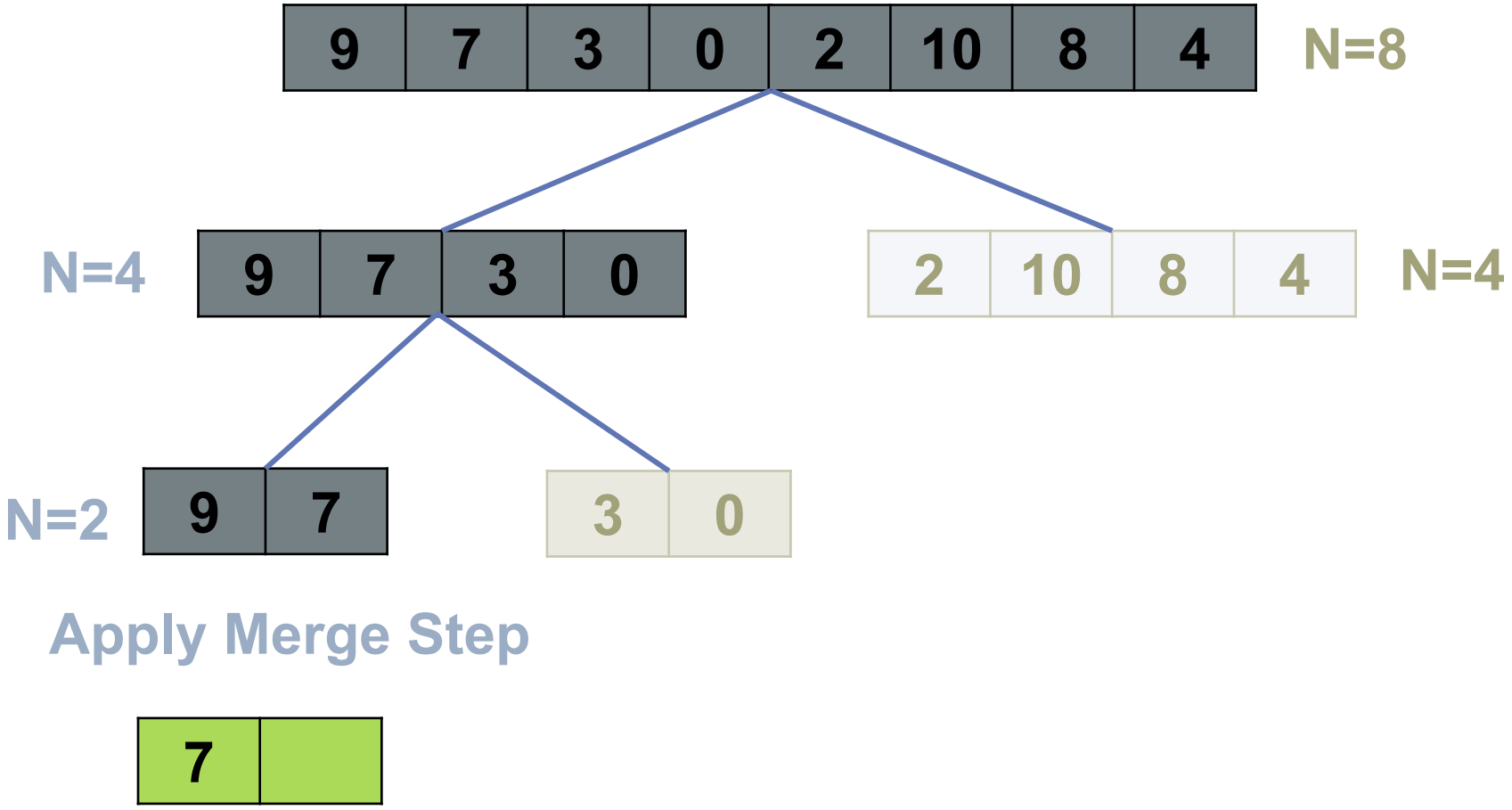
MERGE SORT



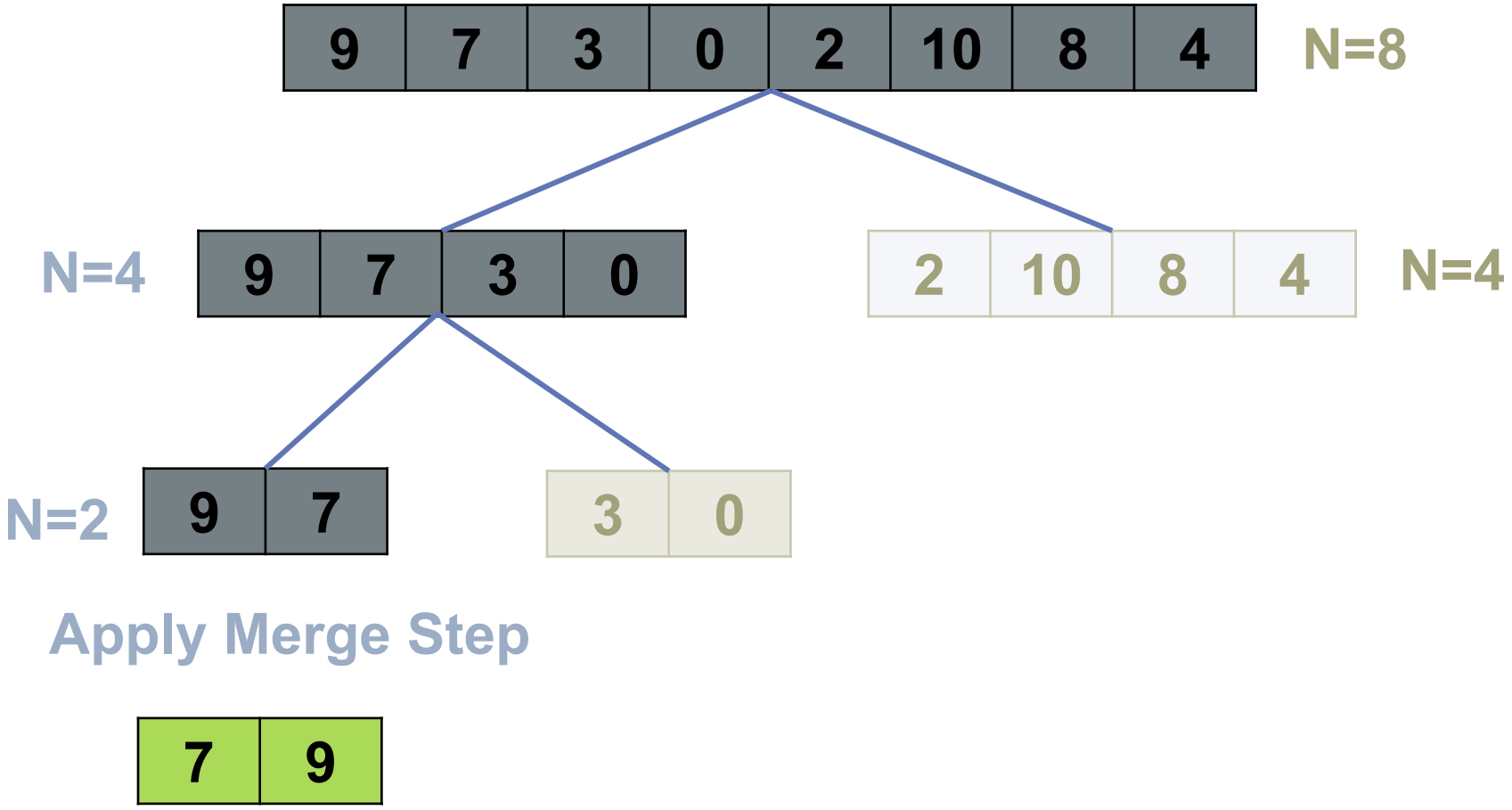
MERGE SORT



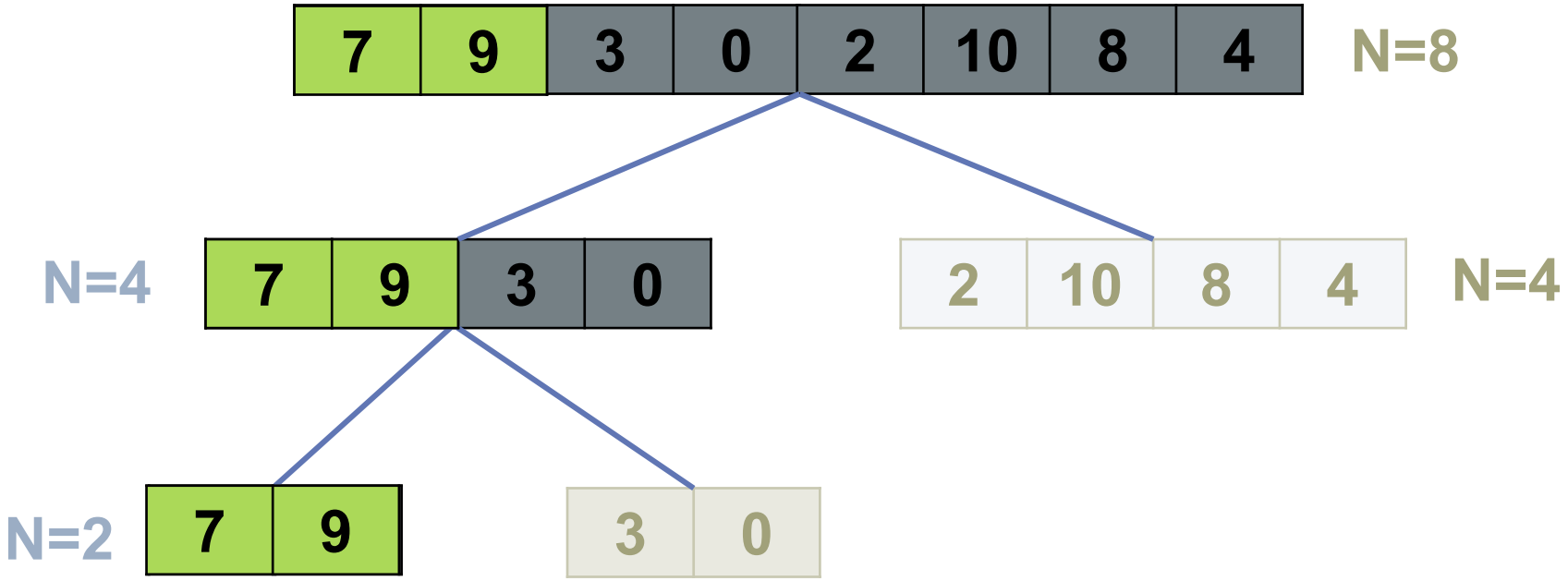
MERGE SORT



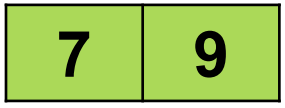
MERGE SORT



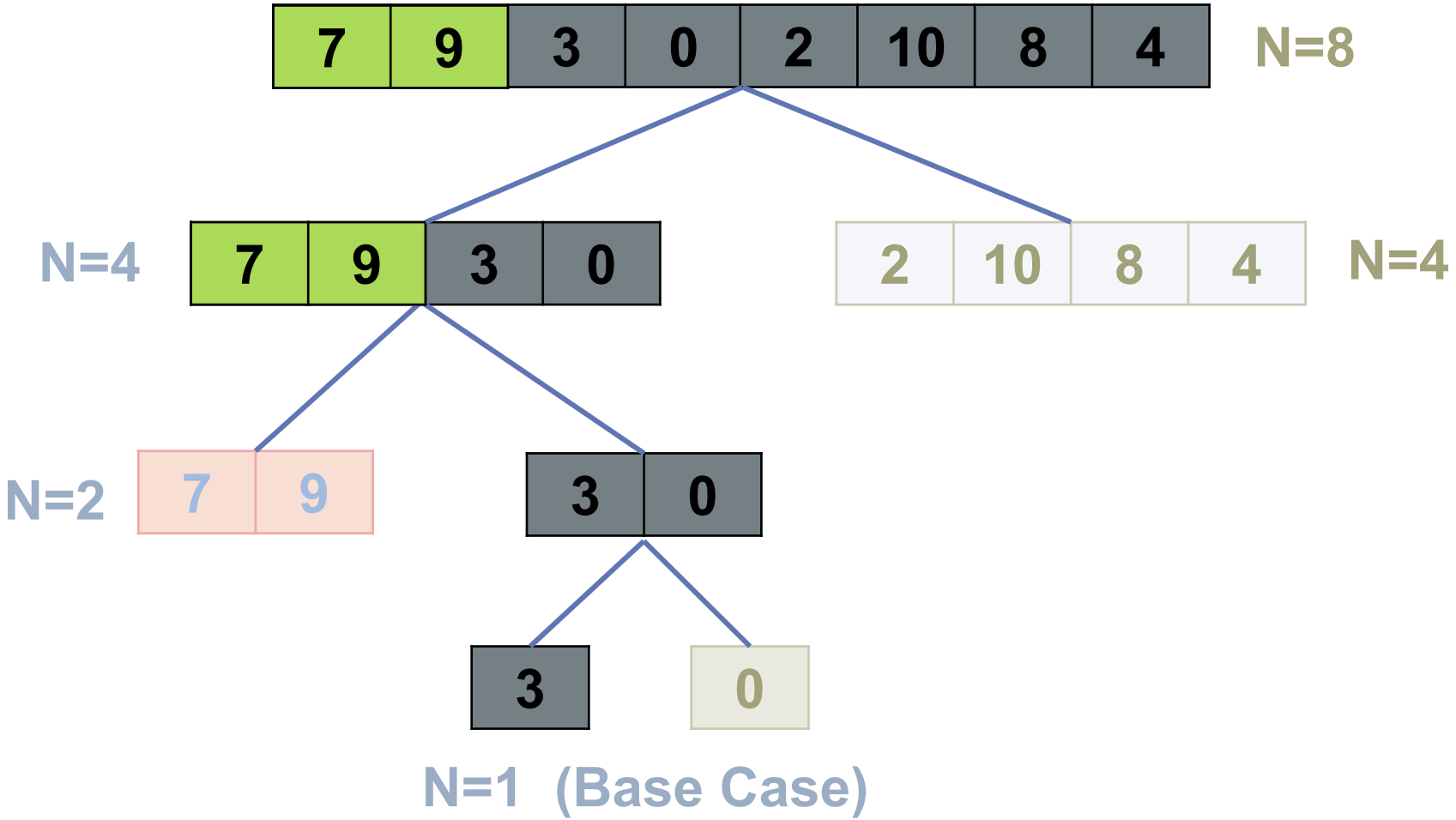
MERGE SORT



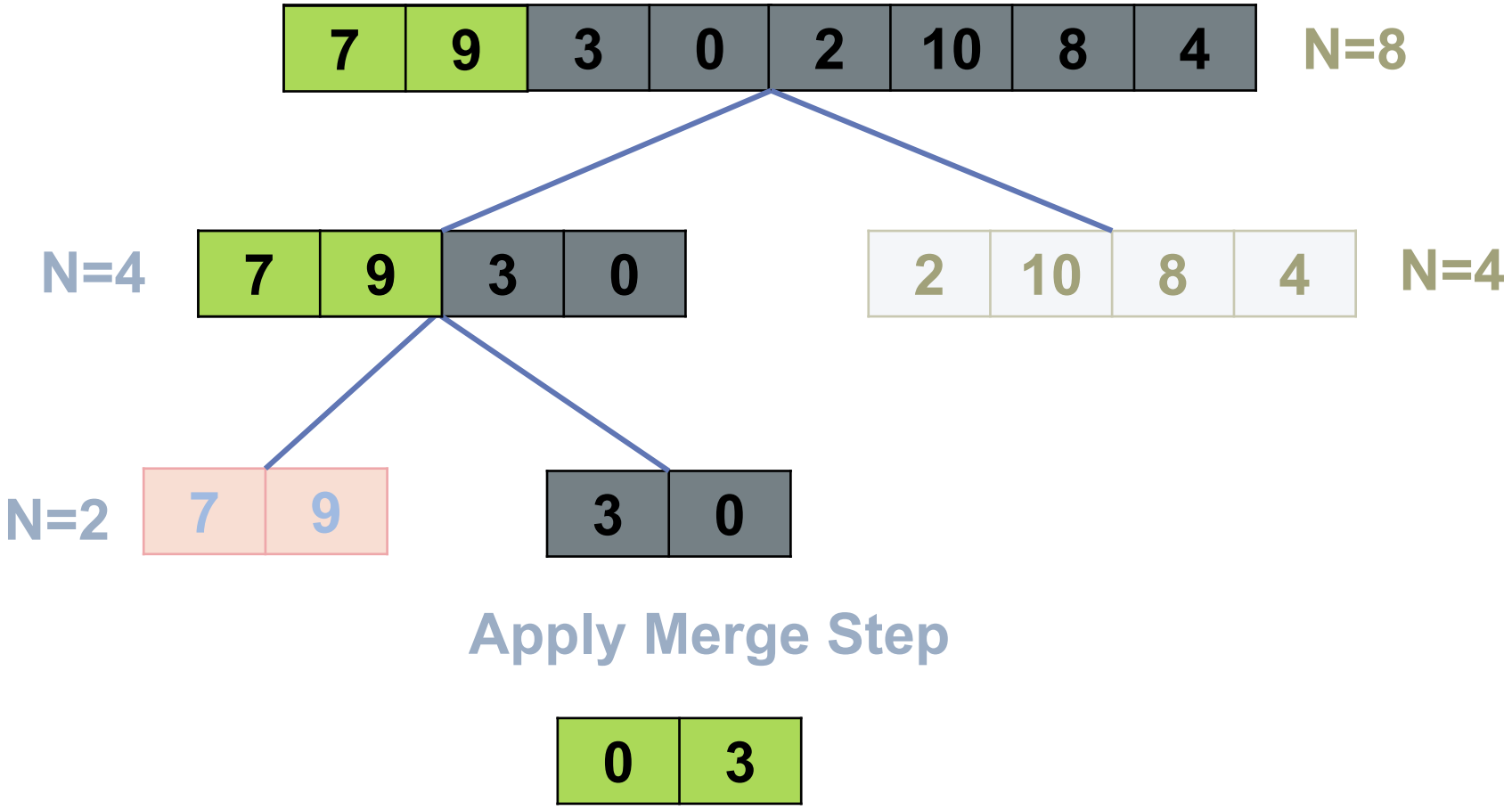
Copy Merged Elements



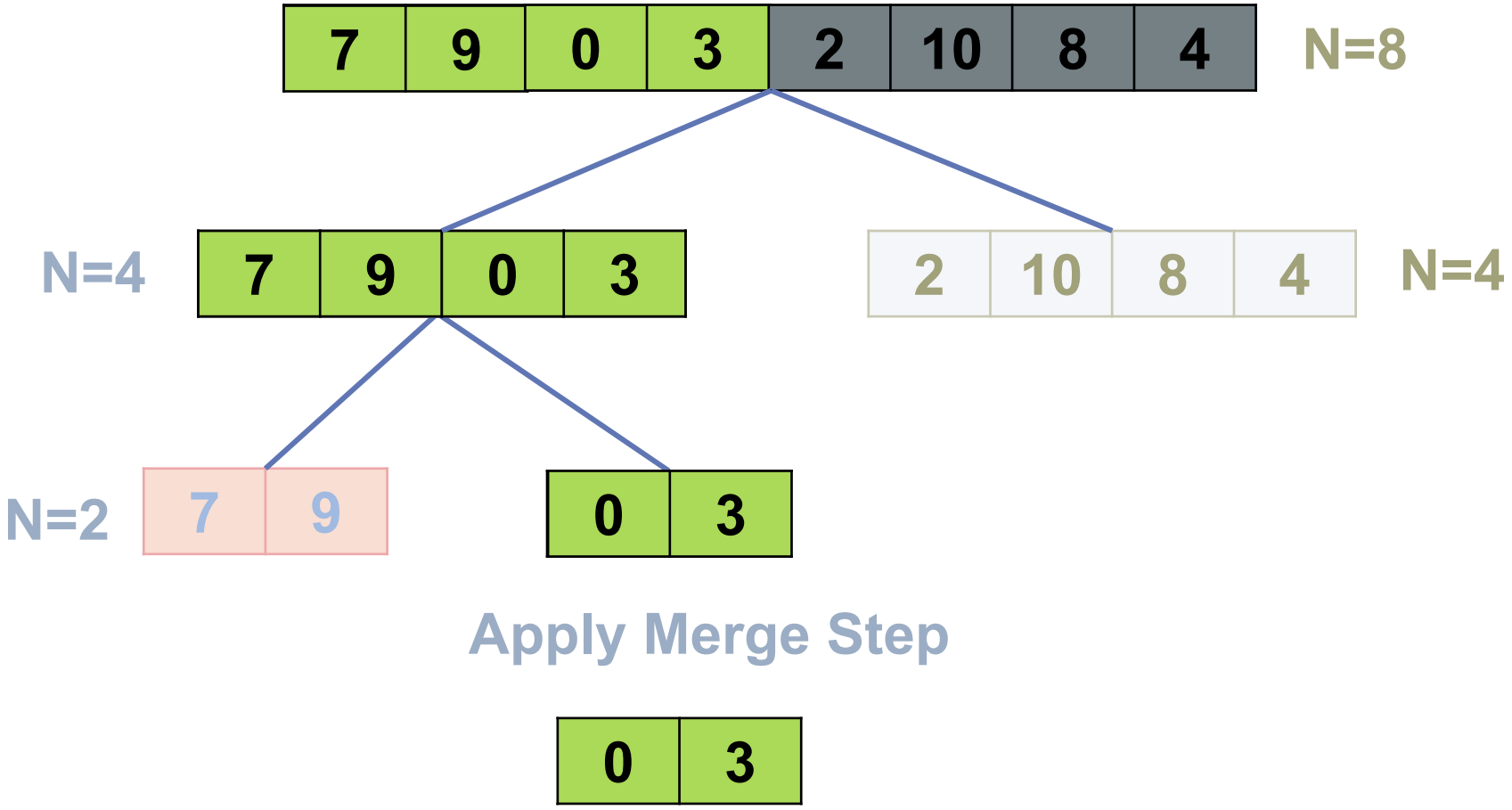
MERGE SORT



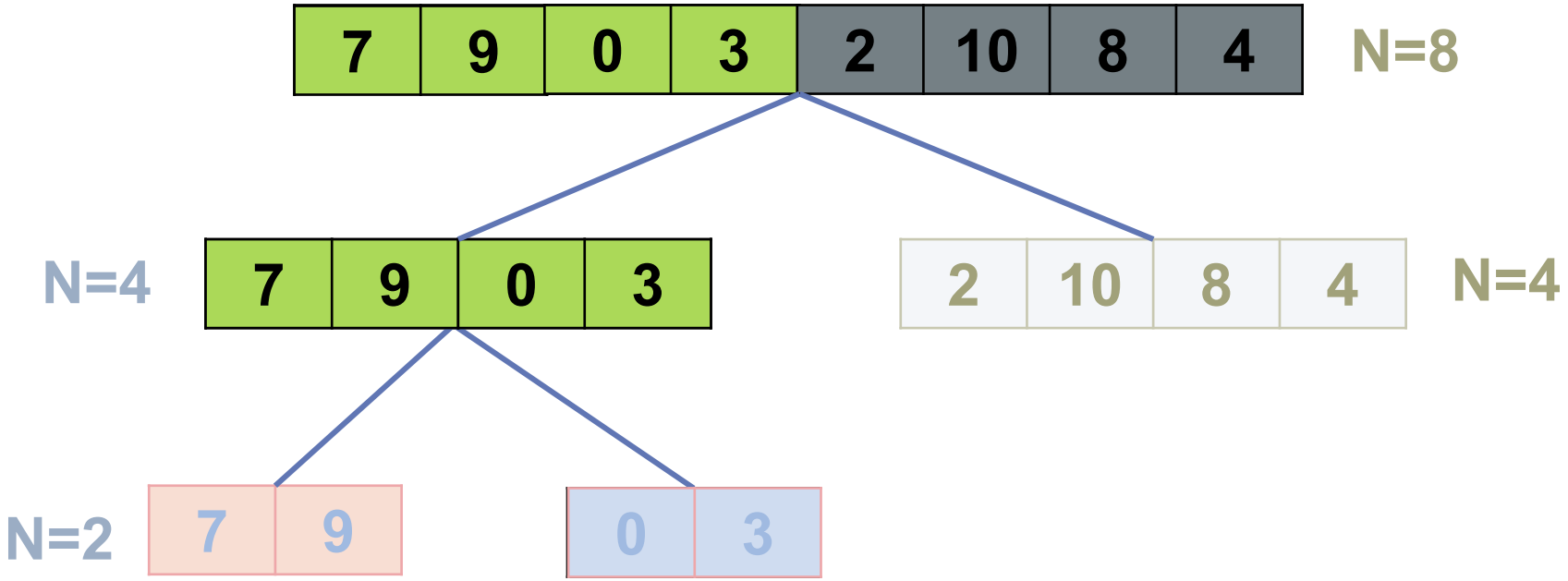
MERGE SORT



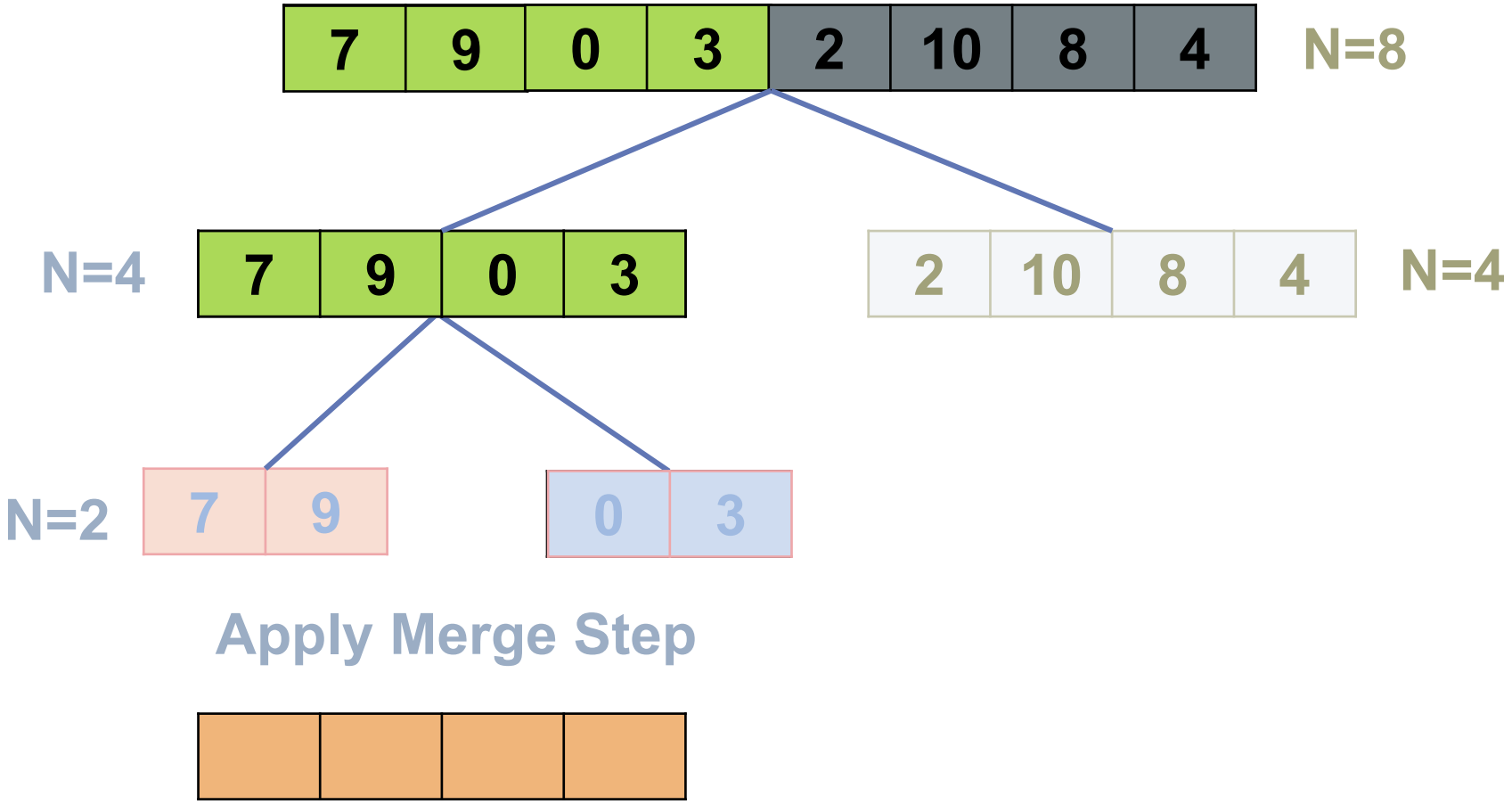
MERGE SORT



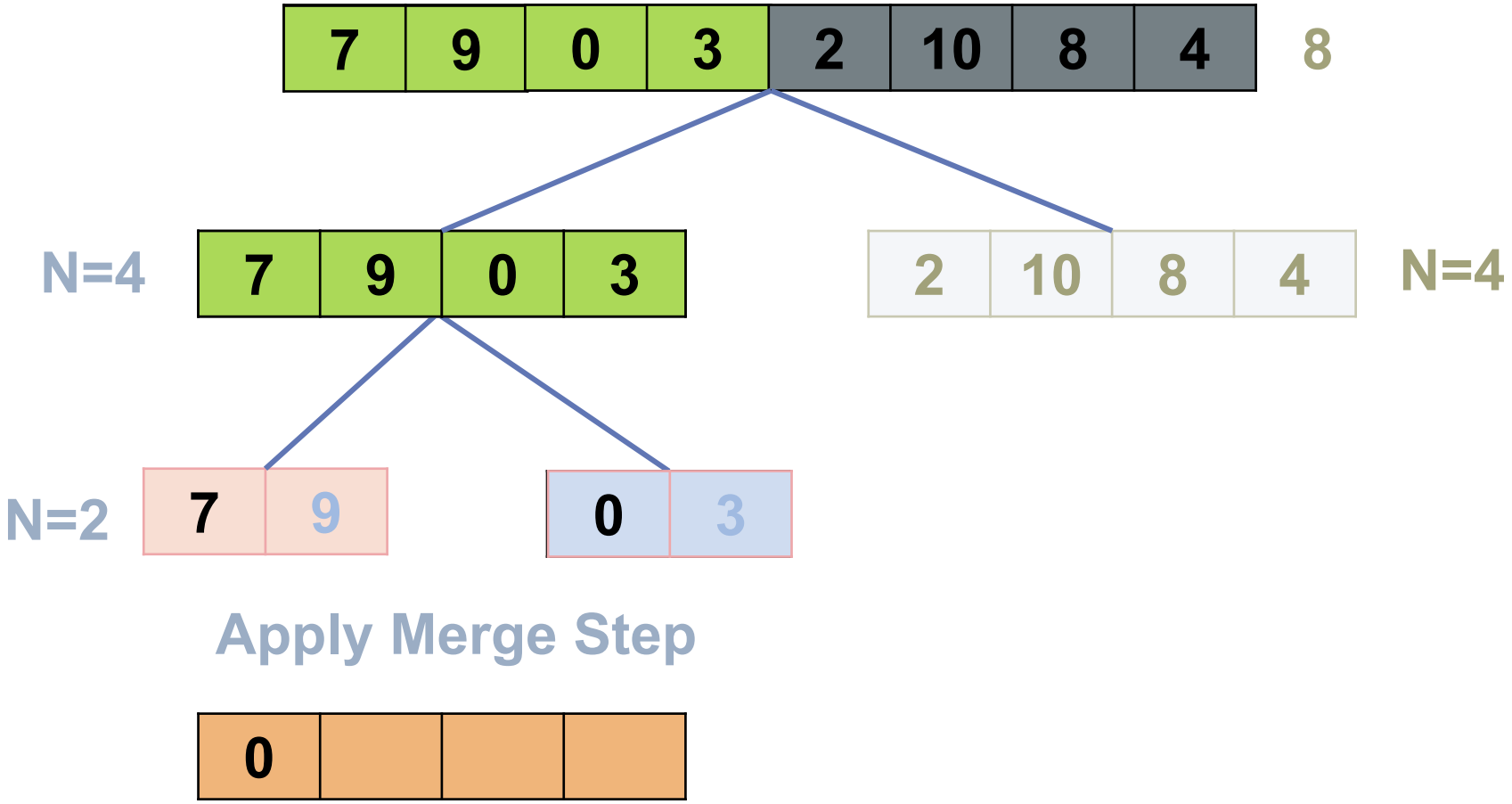
MERGE SORT



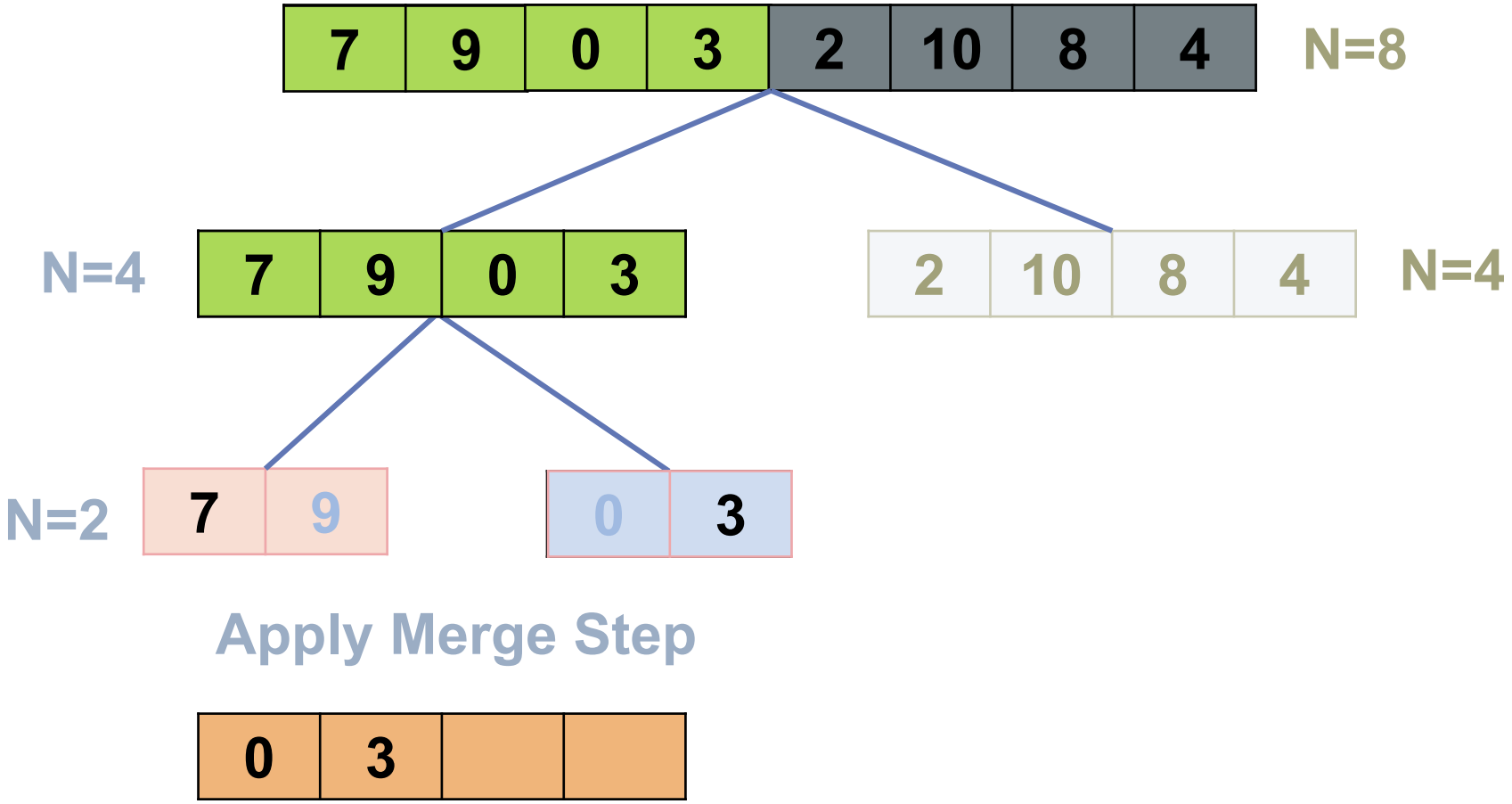
MERGE SORT



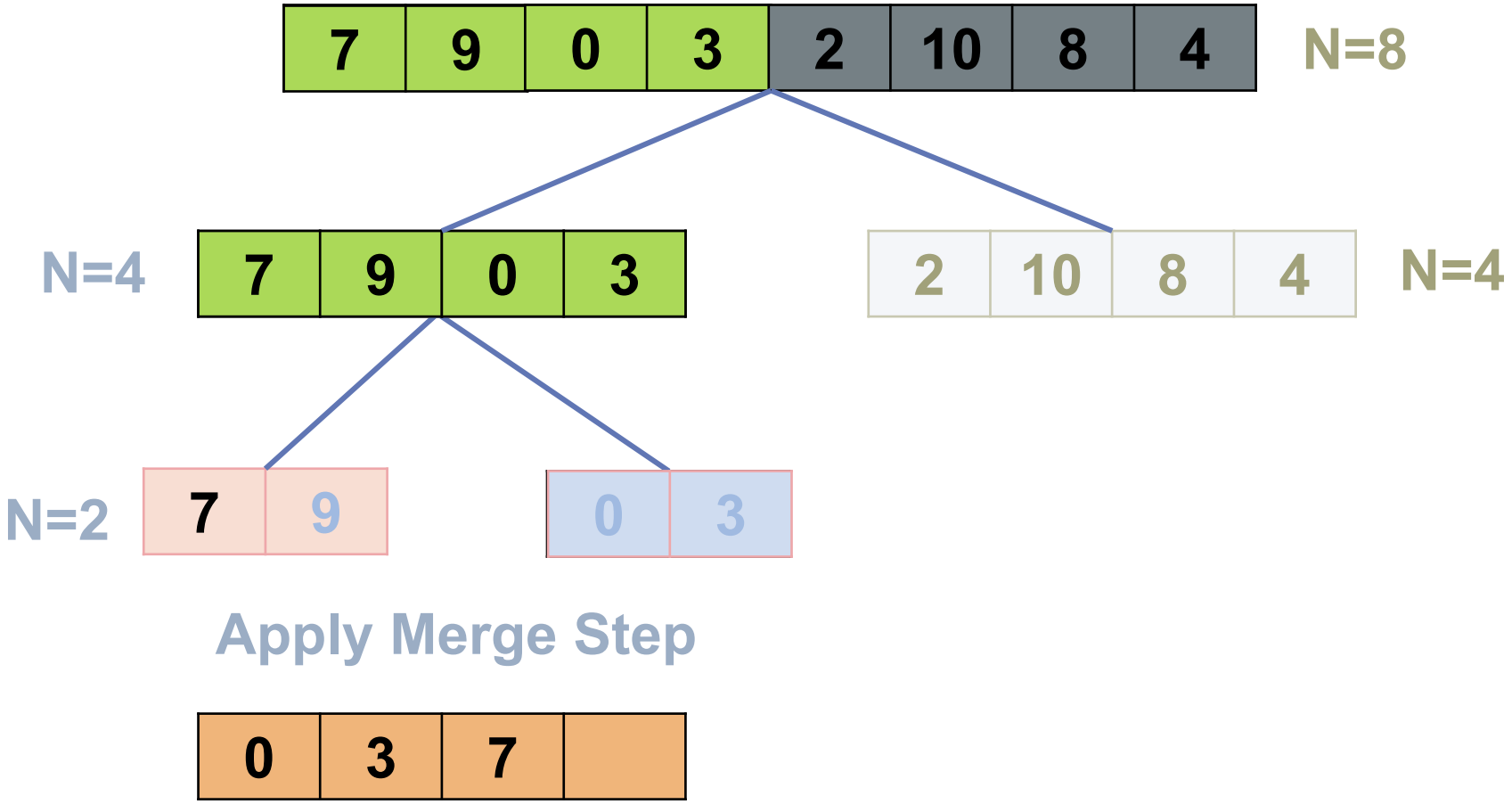
MERGE SORT



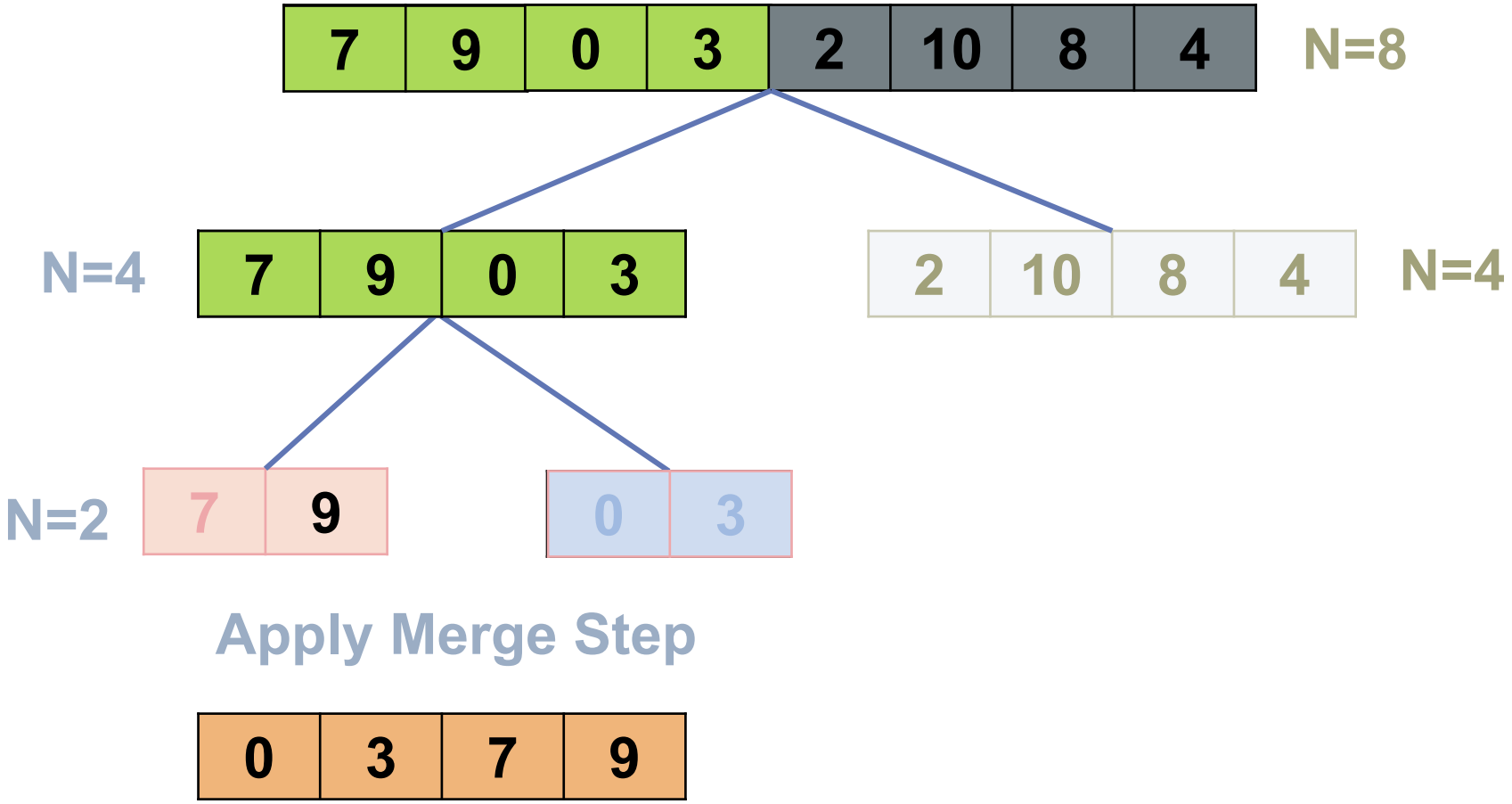
MERGE SORT



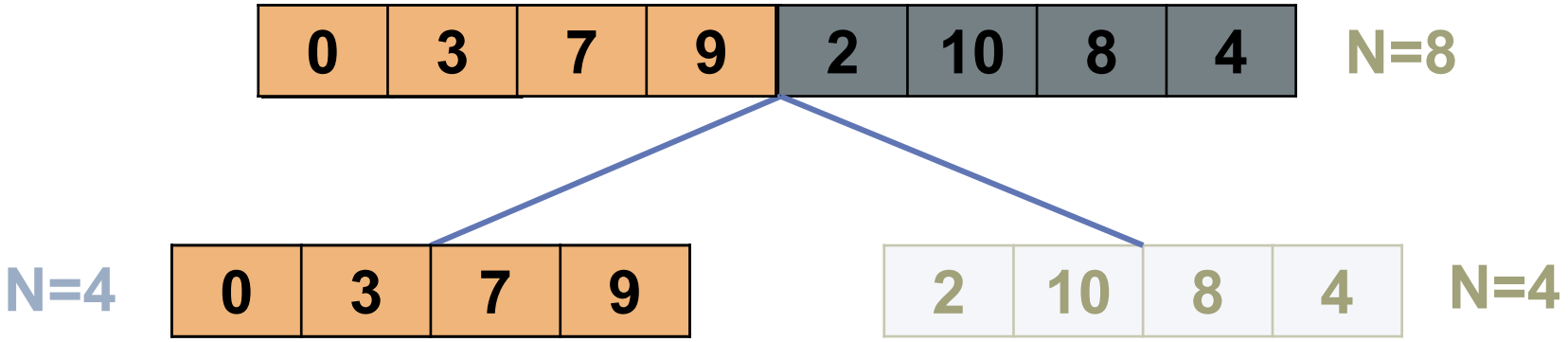
MERGE SORT



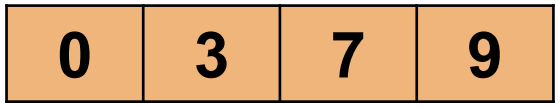
MERGE SORT



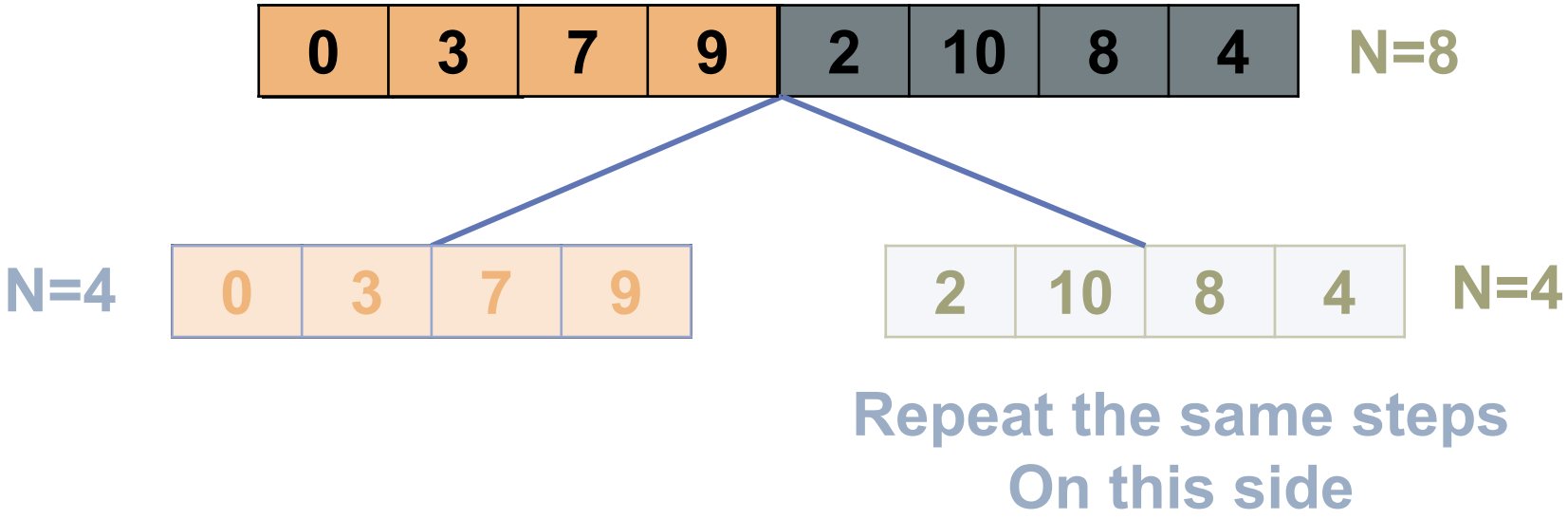
MERGE SORT



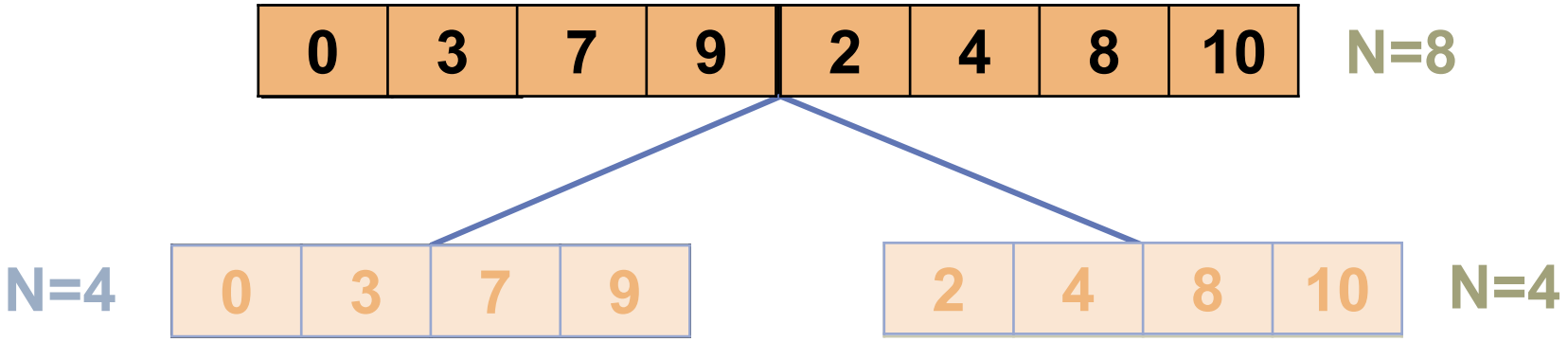
Copy Merged Elements



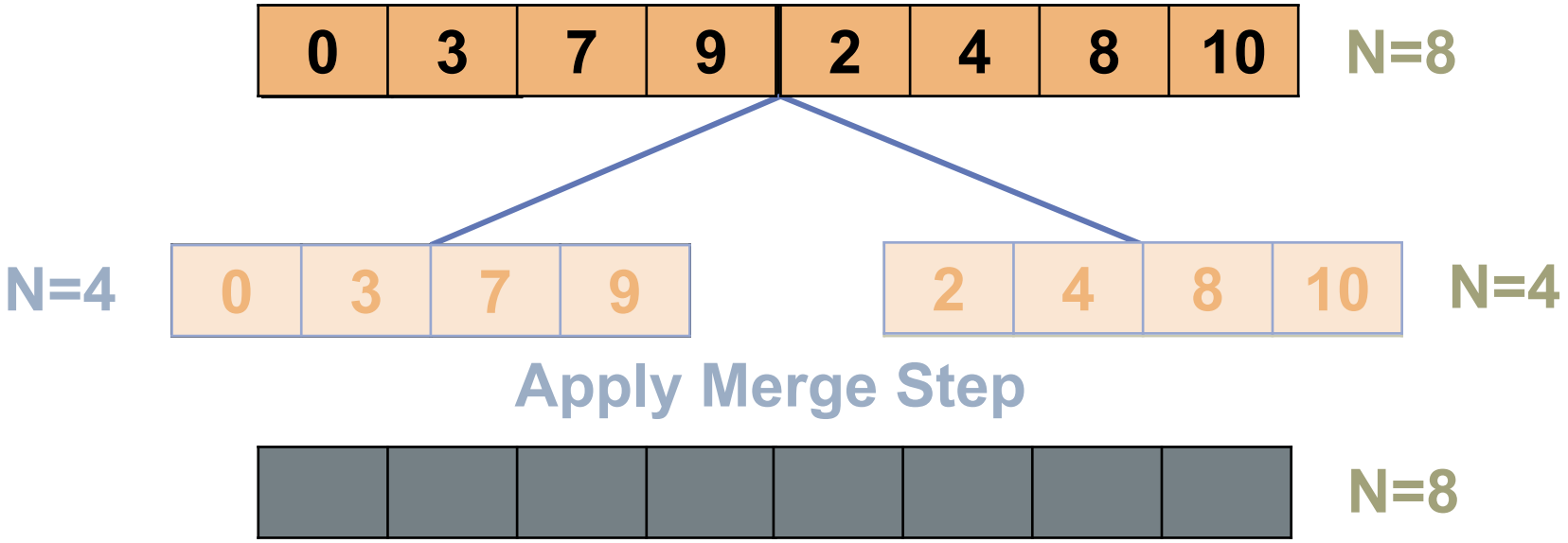
MERGE SORT


0 3 7 9 2 10 8 4
N=8
N=4
0 3 7 9
2 10 8 4
N=4
Repeat the same steps
On this side

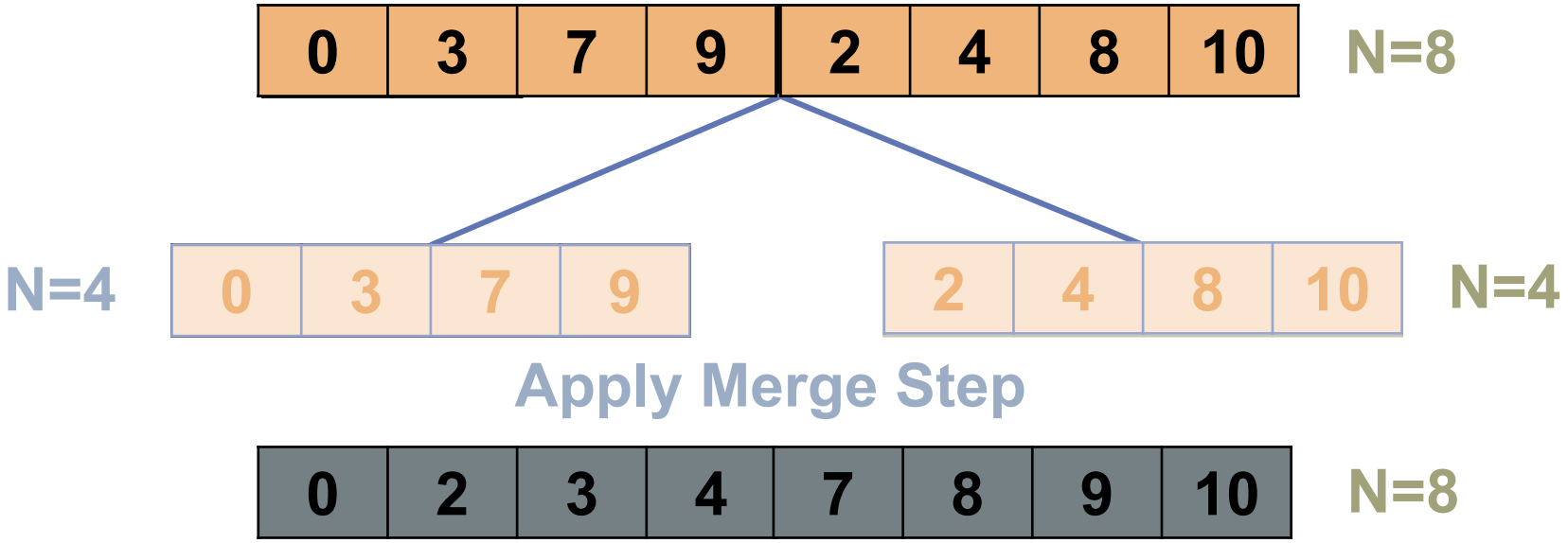
MERGE SORT



MERGE SORT



MERGE SORT



MERGE SORT

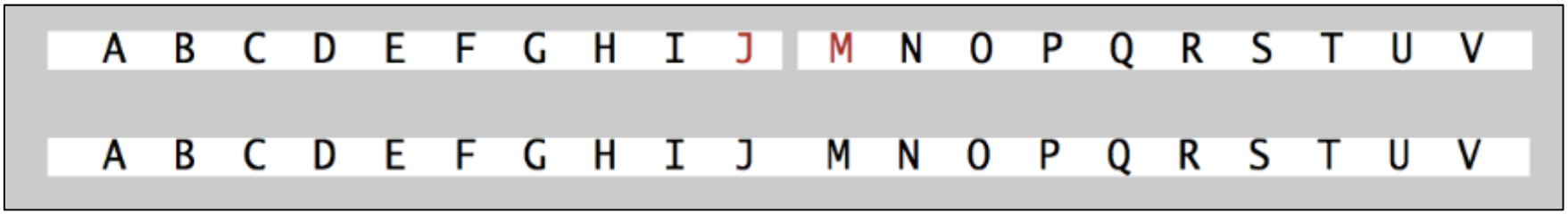
N=8

Copy Merged Elements

N=8

IMPROVING MERGE SORT

- **Stop if array is already sorted.**
  - If largest item in first half <= smallest item in second half
    - Array is already sorted, return.



MERGE SORT

```
private static void mergeSort(int[] data, int lo, int hi) {
        if (lo<hi){
                //find index of middle element
                int mid=(lo+hi)/2;
                //recursive calls for first and second half of the array
                mergeSort(data, lo, mid);
                mergeSort(data, mid+1, hi);
                //merge the elements of each side of the array
                merge(data, lo, mid, hi);
        }
}
```