Data Structures

Divide and Conquer

- A technique to solve a computational problem by dividing it into one or more subproblems, recursively solve the subproblems, and then merge the solutions to the subproblems
  - **Divide:** The problem is divided into a number of subproblems which are smaller instances of the original problem
  - **Conquer:** The subproblems are solved recursively
  - **Combine:** Combine the solutions to the subproblems to get the solution to the original problem
- "n" is the size of the problem, "S(n)" is the problem to be solved
- $S(n)$ is divided into $S(n_1)$, $S(n_2)$, . . ., $S(n_k)$, where $n_i < n$ for $i = 1,2, . . ., k$
- Solve $S(n_1)$, $S(n_2)$, . . ., $S(n_k)$
- Combine the solutions of $S(n_1)$, $S(n_2)$, . . ., $S(n_k)$ to get the solution of $S(n)$

Merge Sort

- Given with an array S of elements/keys
- **Divide:** If S has zero or one element, return S directly. Otherwise (that is, if S has at least two elements), remove all elements from S and put them in two sequences, S_1 and S_2, each containing half of the elements of S (that is, S_1 contains the first $\lceil n/2 \rceil$ elements and S_2 contains the remaining $\lfloor n/2 \rfloor$)
- **Conquer:** Sort sequences S_1 and S_2 using Merge Sort
- **Combine:** Merge the sorted sequences S_1 and S_2 into one sorted sequence and put it in S

Merge Sort Algorithm

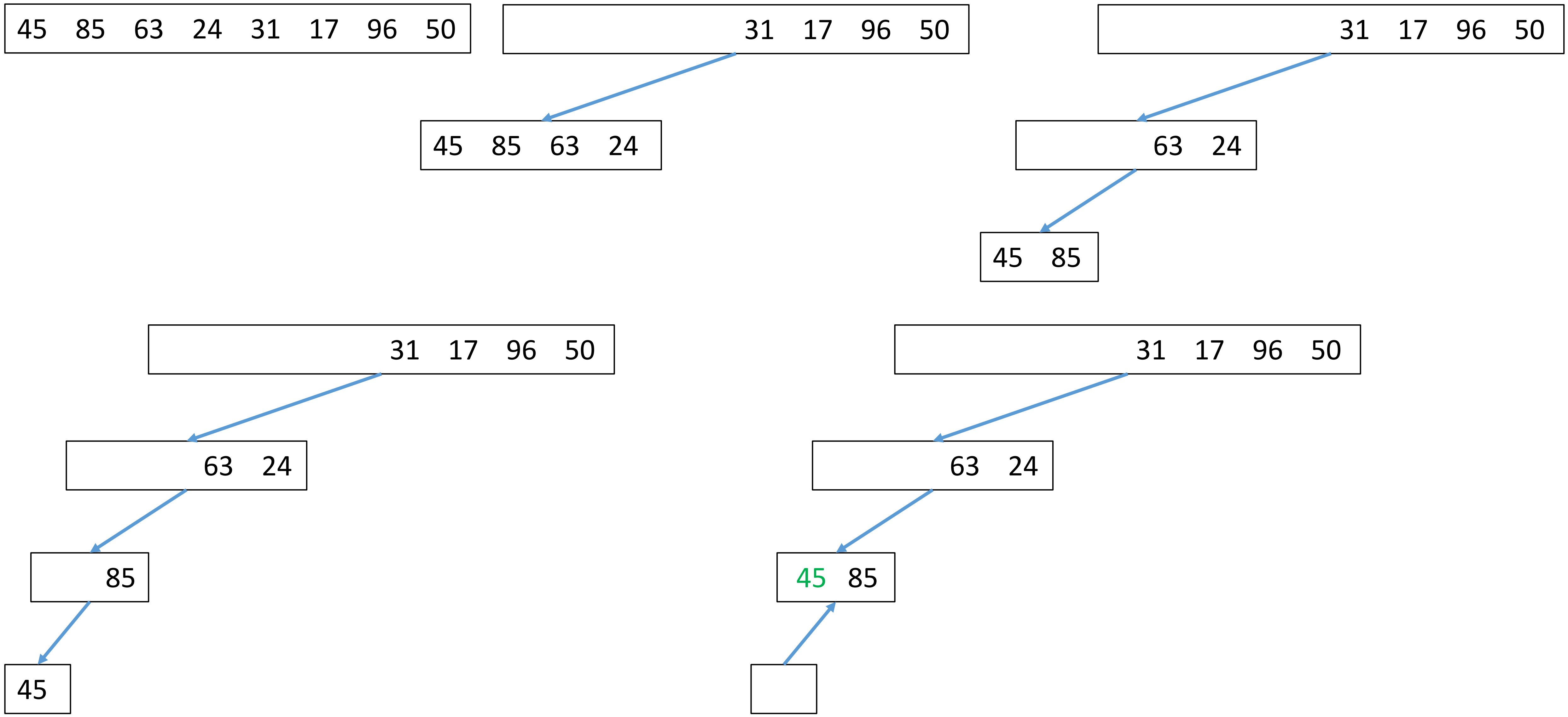
```
Algorithm Merge_Sort(A, l, r)
        If(l < r)
                center = (l+r)/2
                Merge_Sort(A, l, center)
                Merge_Sort(A, center + 1, r)
                Merge(A, l, center, r)
```

Merge algorithm

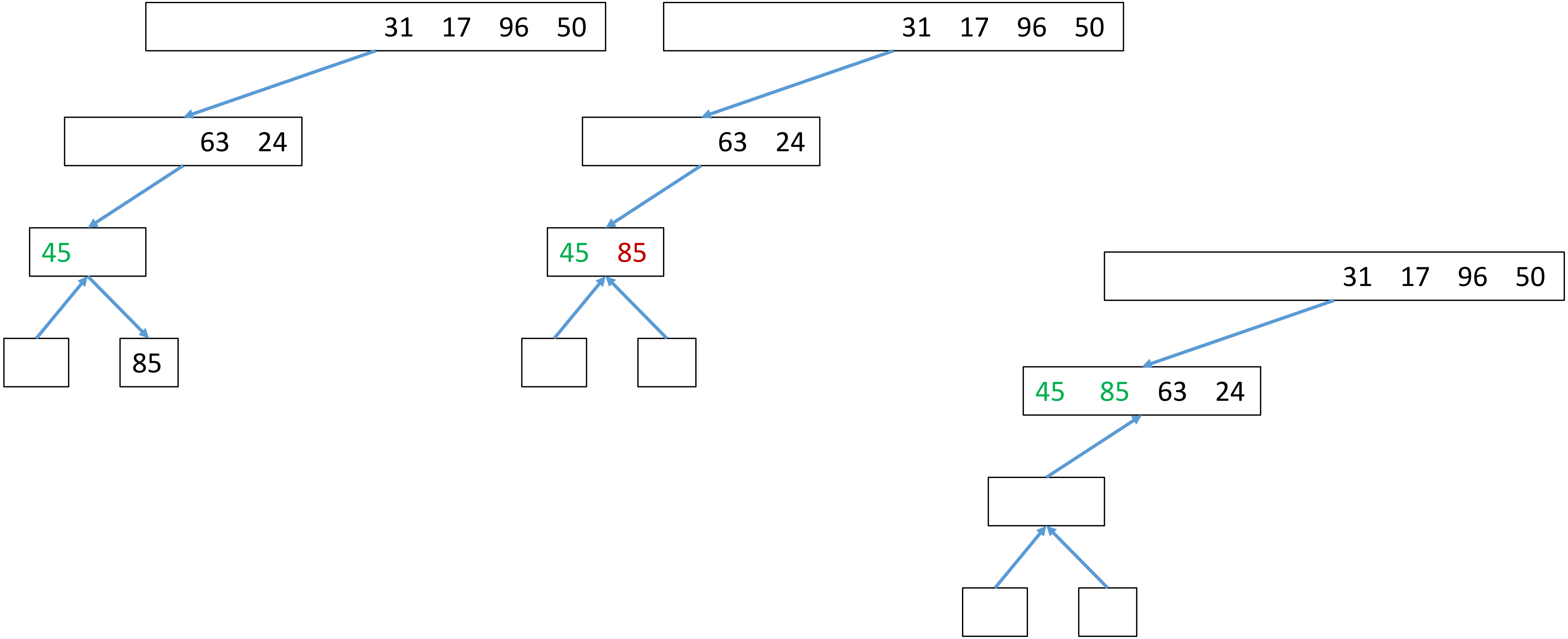
Algorithm Merge(A, l ,center, r)

```
k ← l, n1 ← center-l+1, n2 ← r-center
S1 ← A[l . . .center]
S2 ← A[center+1 . . .r]
i ← 0, j ← 0
while (i < n1 and j < n2) do
        if S1 [i] ≤ S2[j] then
                A[k] = S1 [i]
                i ← i+1
                k ← k+1
        else
                A[k] = S2[j]
                j ← j+1
                k ← k+1
if (i < n1)
        Copy the remaining elements from S1 to A
if(j < n2)
        Copy the remaining elements from S2 to A
```

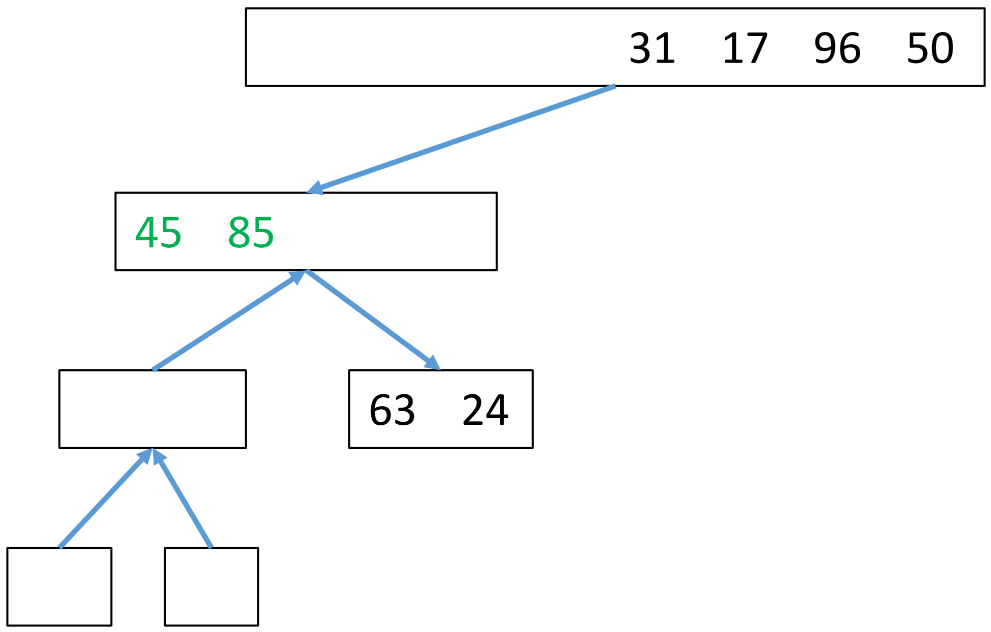
Merge Sort: Example

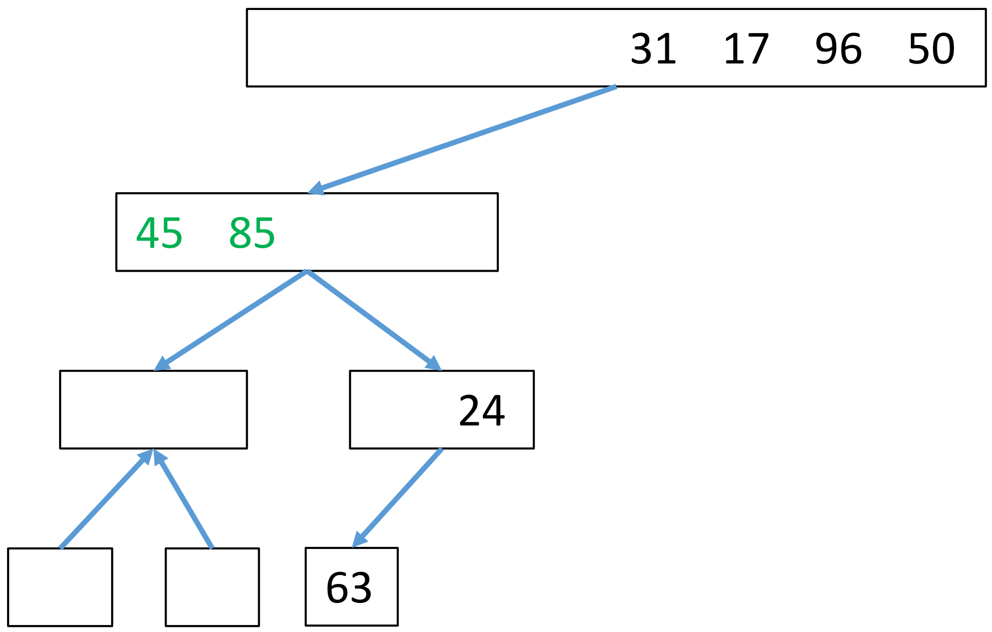


Merge Sort: Example

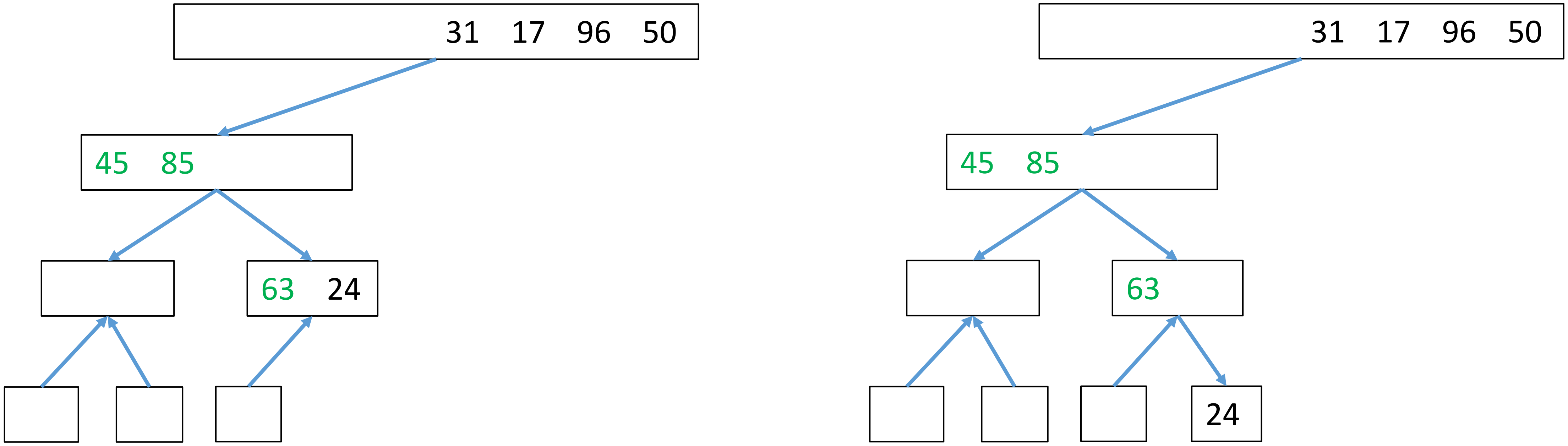


Merge Sort: Example

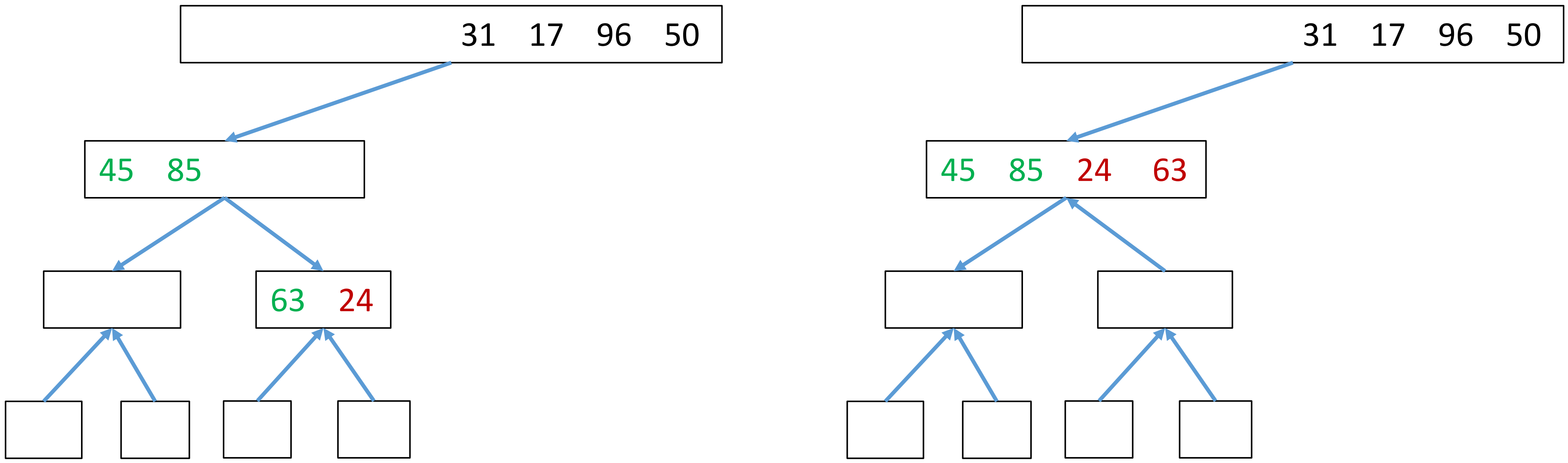




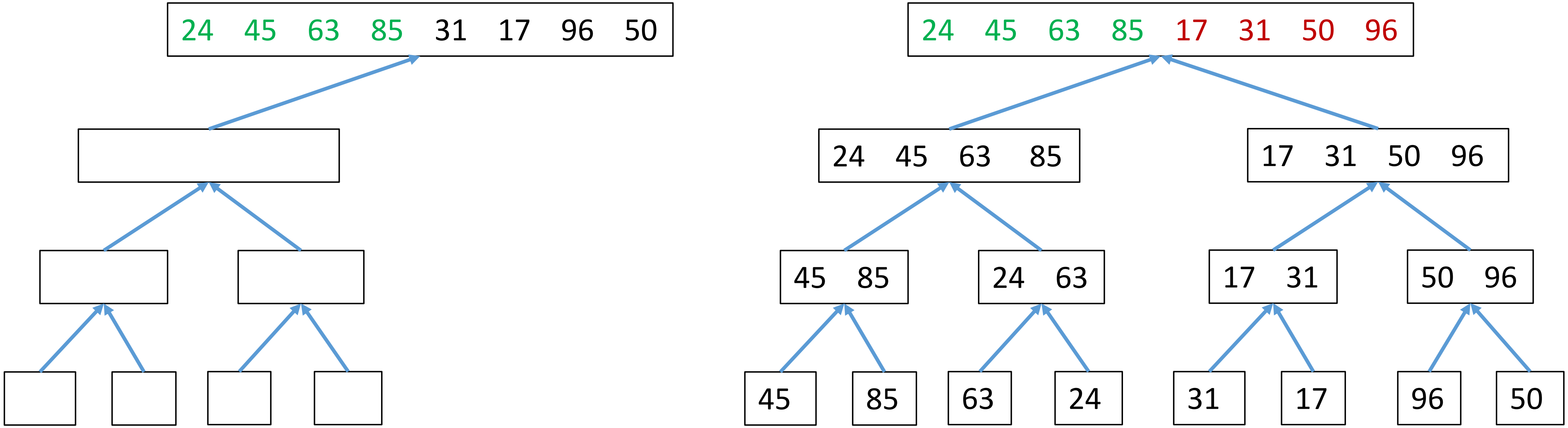
Merge Sort: Example



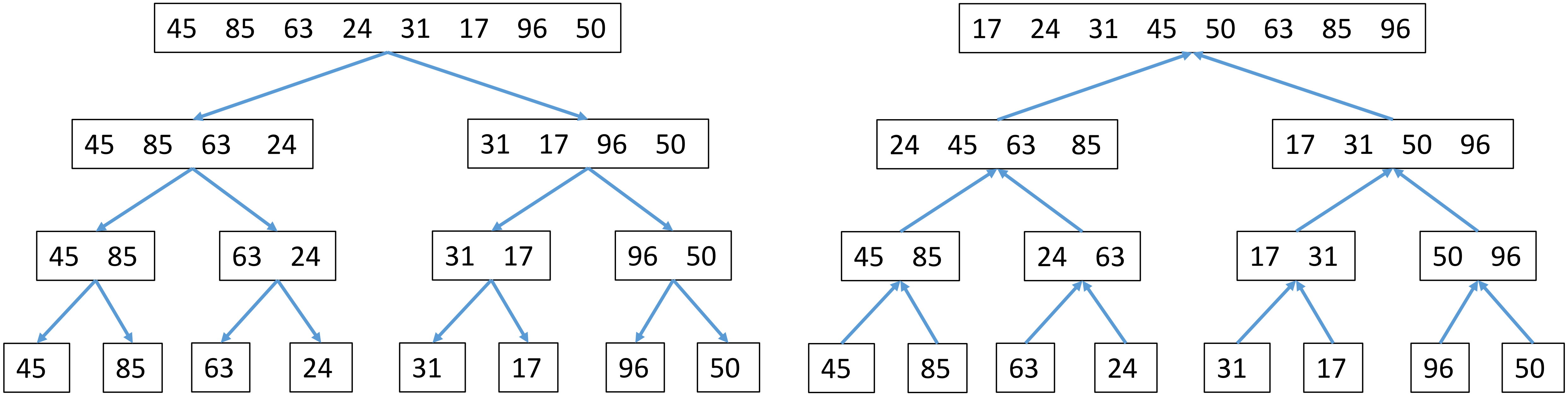
Merge Sort: Example



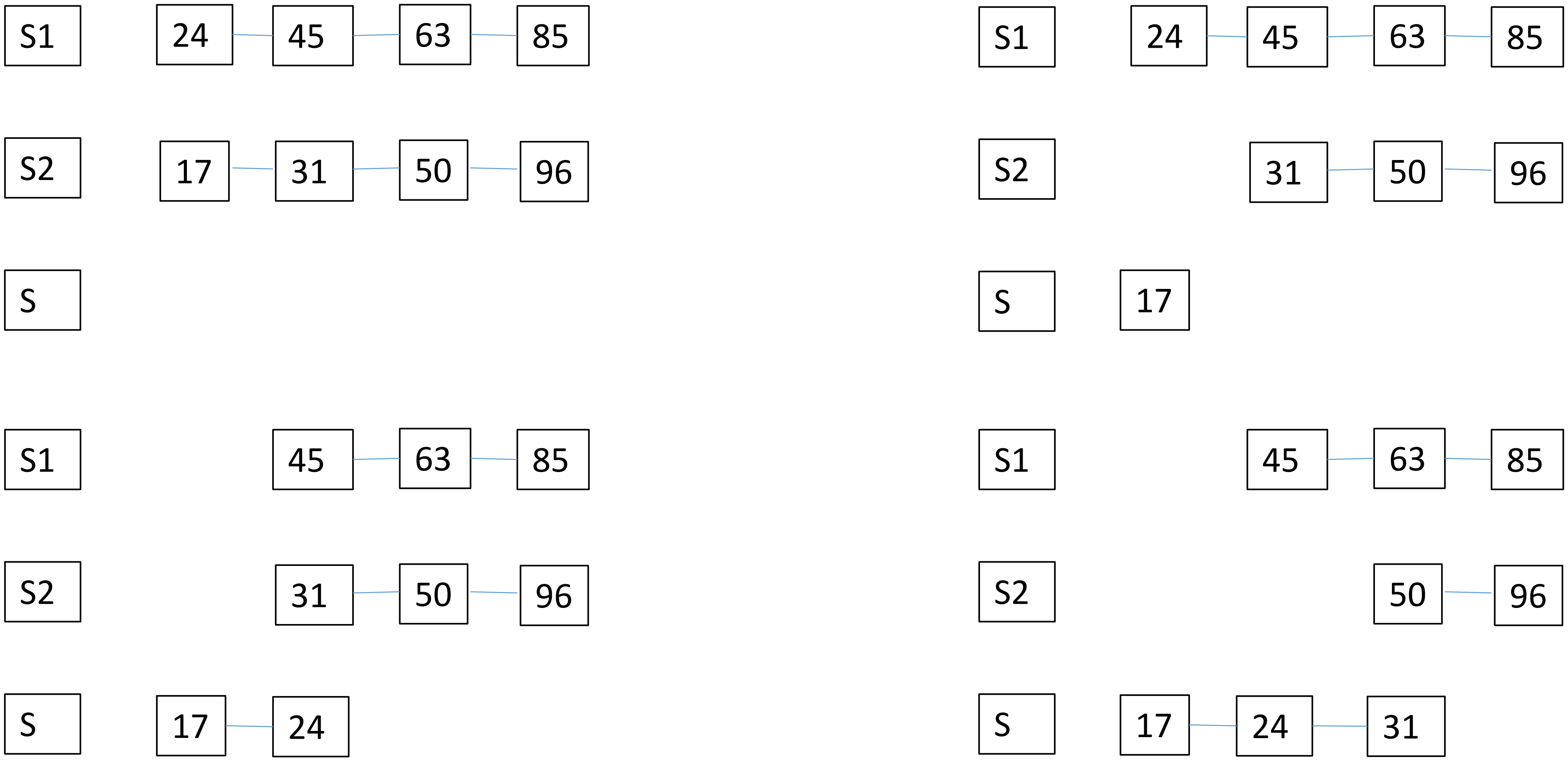
Merge Sort: Example



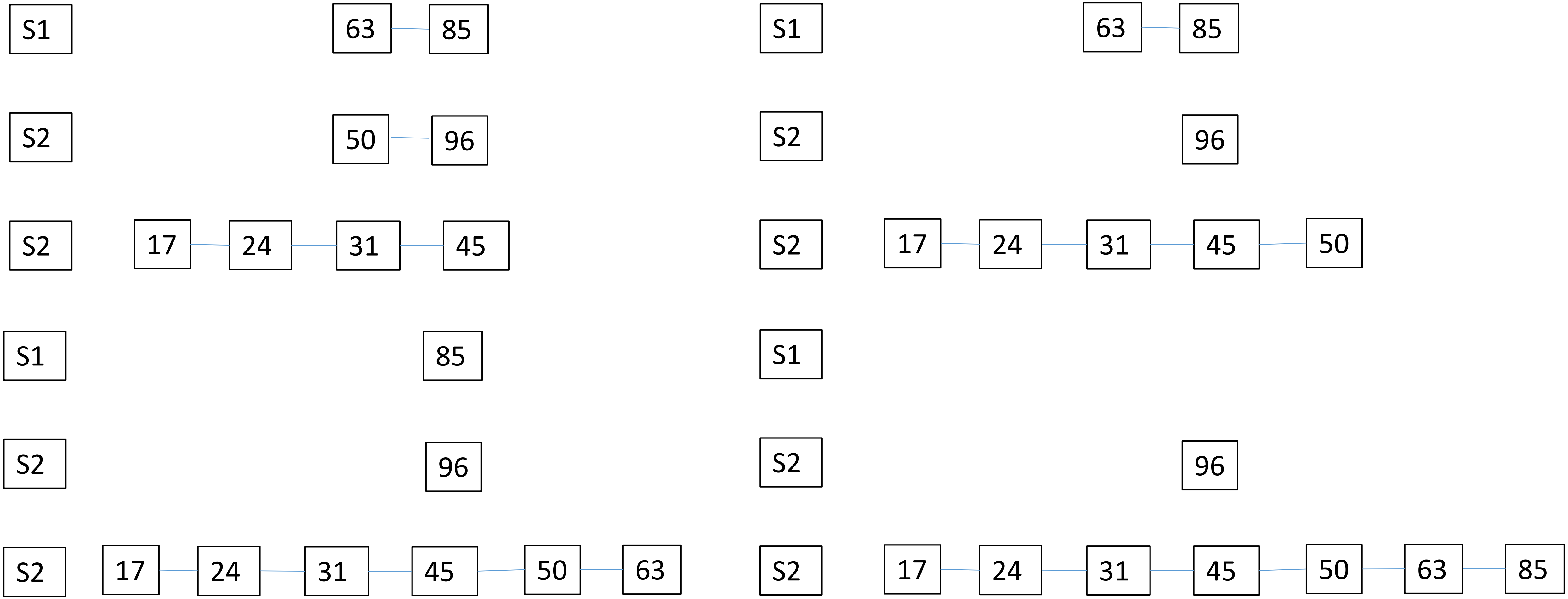
Merge Sort: Example



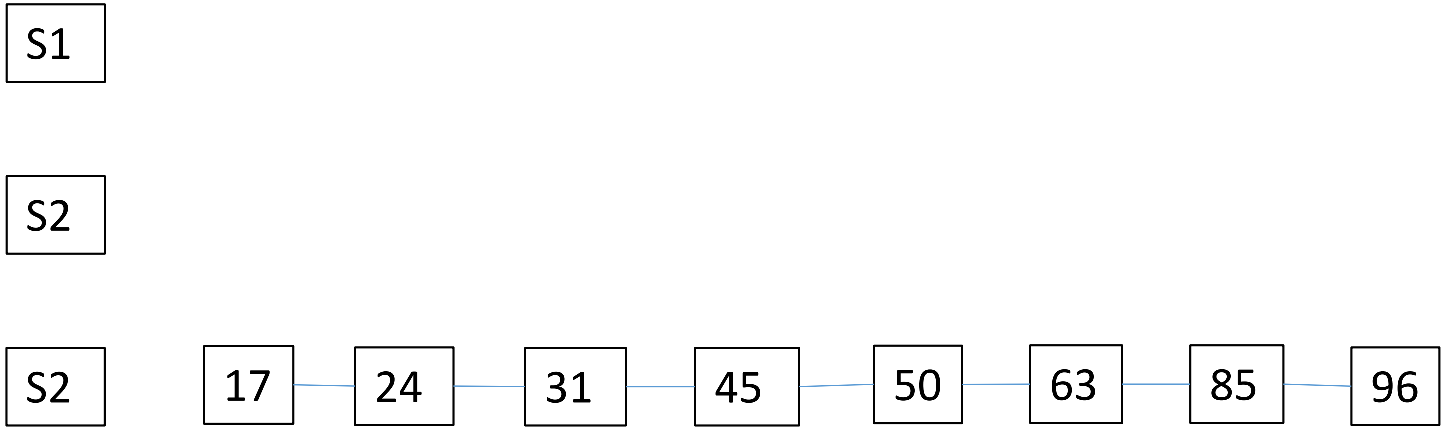
Merge Sequences



Merge Sequences



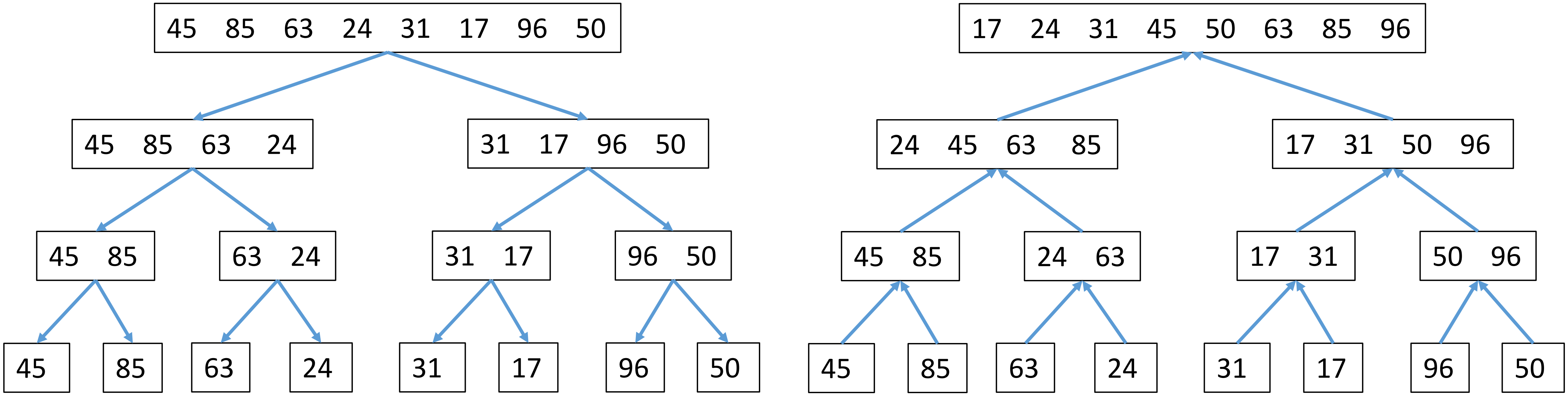
Merge Sequences



Analysis of Merge method

- The merge algorithm is of order O(n1 + n2), where n1 and n2 are sizes of sequences S1 and S2, respectively
- If n1 and n2 are sizes of sequences S1 and S2, respectively, then the number of comparisons in the worst case?
- The number of comparisons in the best case?

Analysis of Merge Sort



Analysis of Merge Sort

- The size of input, n, is a power of 2
- The time spent at node v:
  - Time spent in division
  - Time spent in merge
- The time spent at node v which is at depth "i" is $O(n/2^i)$
- The time spent at a depth is: $O(2^i\, n/2^i)$
- The height tree is log(n)
- Running time is: O(n log n)
- We log n to represent $\log_2 n$