Sorting Algorithms

Merge Sort

Overview

- Divide and Conquer
- Merge Sort

Divide and Conquer

1. Base Case, solve the problem **directly** if it is small enough

2. **Divide** the problem into two or more **similar and smaller** subproblems

3. **Recursively** solve the subproblems

4. **Combine** solutions to the subproblems

Divide and Conquer - Sort

Problem:

- Input: A[left..right] – **unsorted** array of integers

- Output: A[left..right] – **sorted** in non-decreasing order

Divide and Conquer - Sort

1. **Base case**

   at most one element (left ≥ right), return

2. **Divide** A into two subarrays: FirstPart, SecondPart

   Two Subproblems:

   sort the FirstPart

   sort the SecondPart

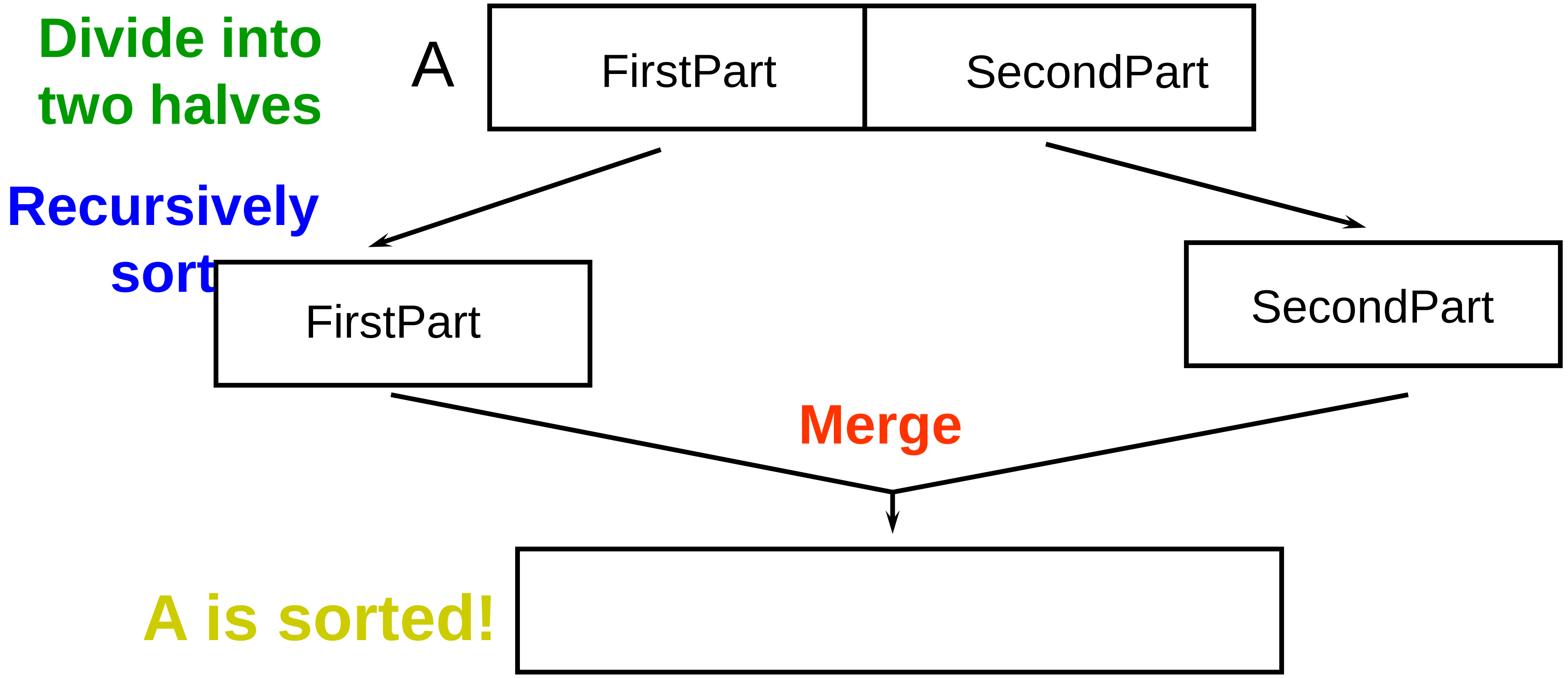
3. **Recursively**

   sort FirstPart

   sort SecondPart

4. **Combine** sorted FirstPart and sorted SecondPart

Merge Sort: Idea

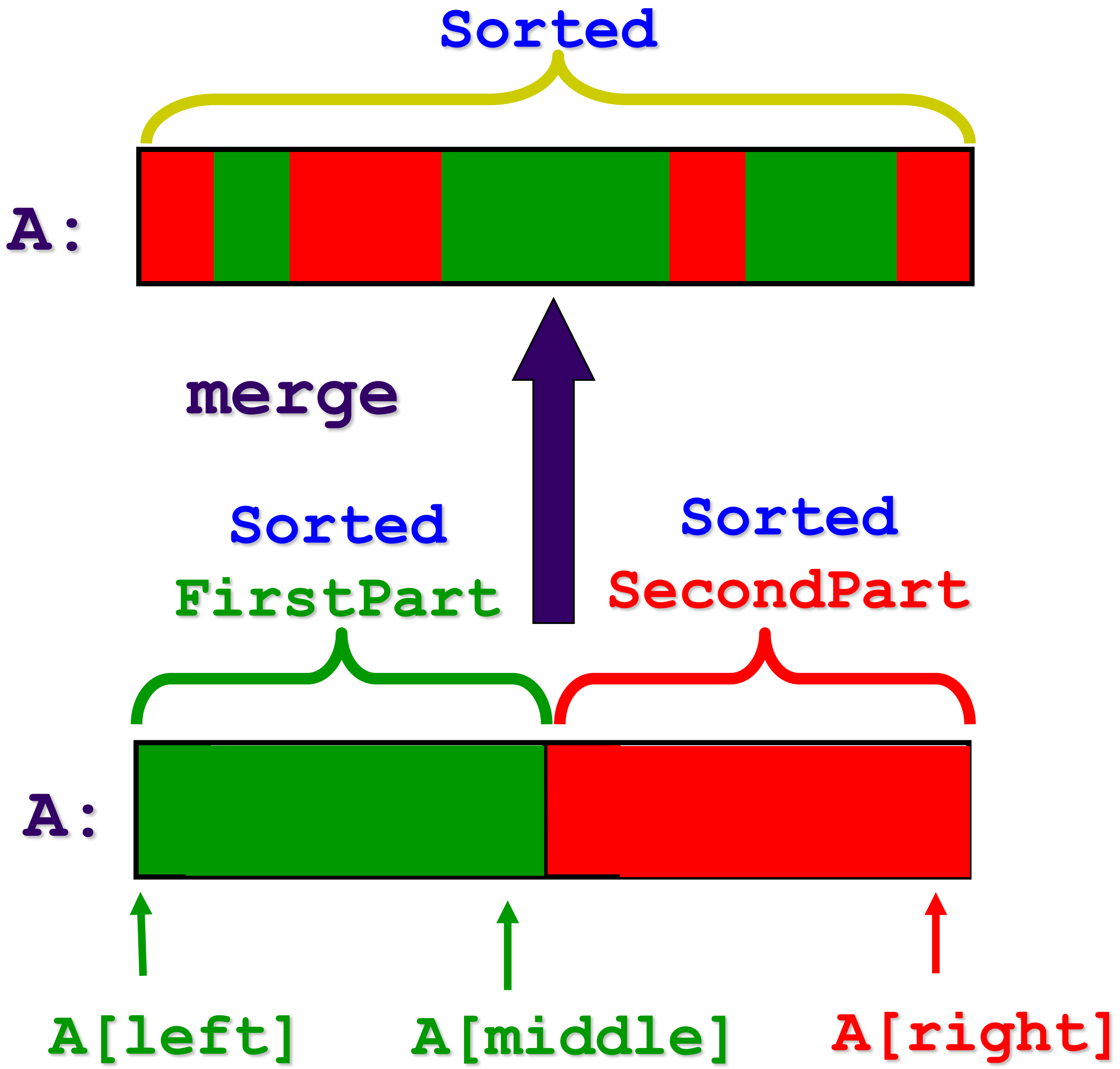


Merge Sort: Algorithm

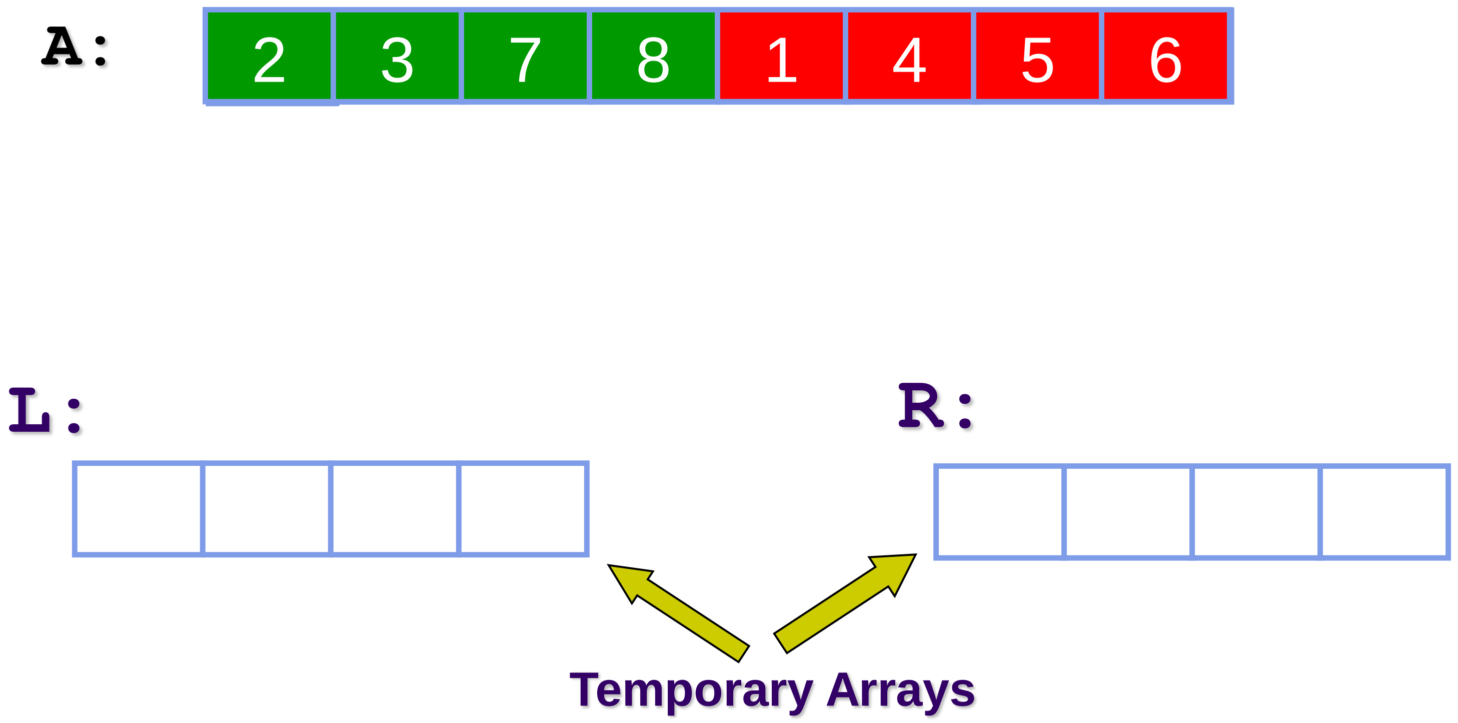
```
Merge-Sort (A, left, right)
  if    $left \geq right$   return
  else
        middle ← b(left+right)/2⌋
        Merge-Sort(A, left, middle)
        Merge-Sort(A, middle+1, right)
        Merge(A, left, middle, right)
```

Recursive Call

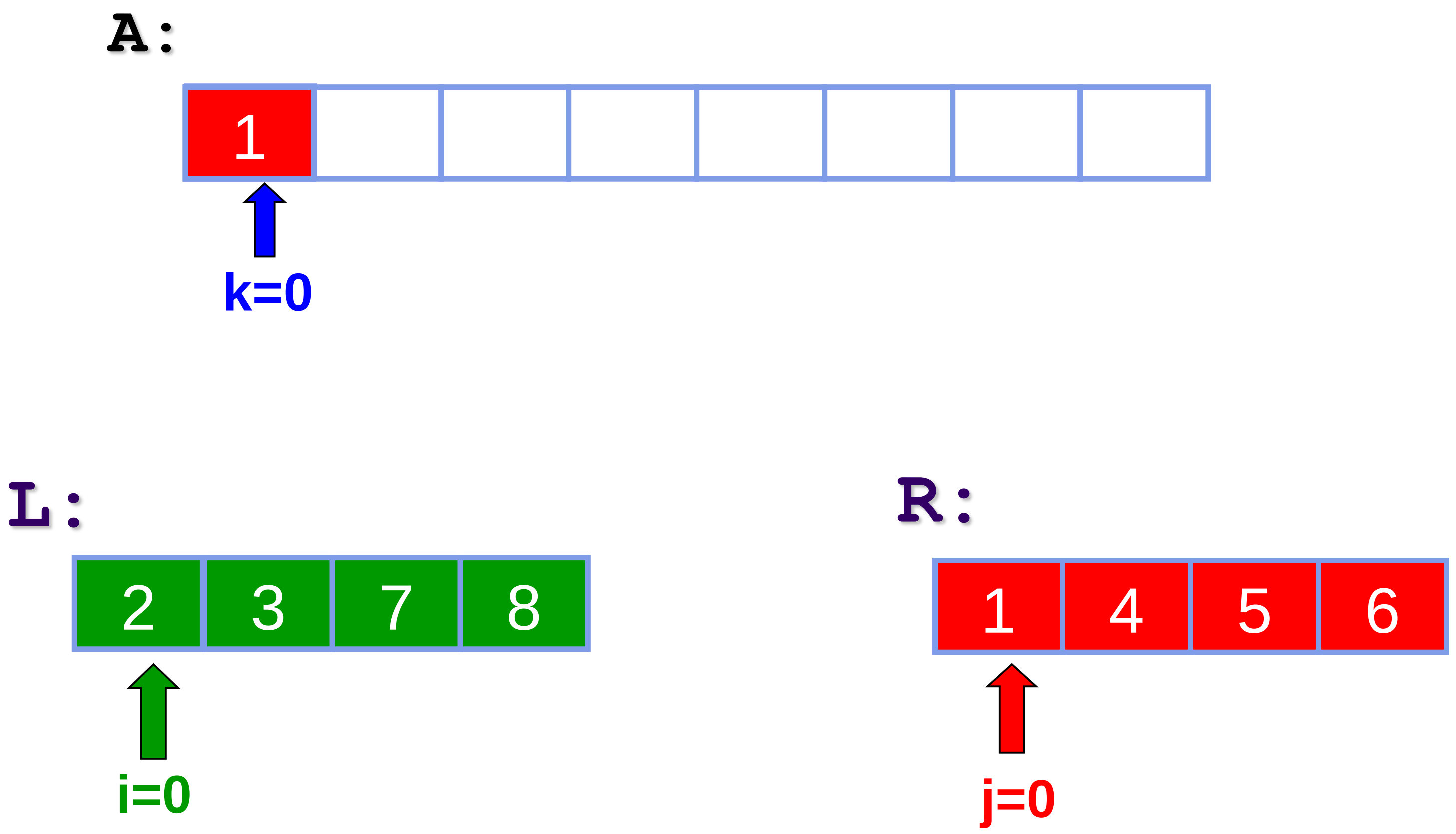
Merge-Sort: Merge



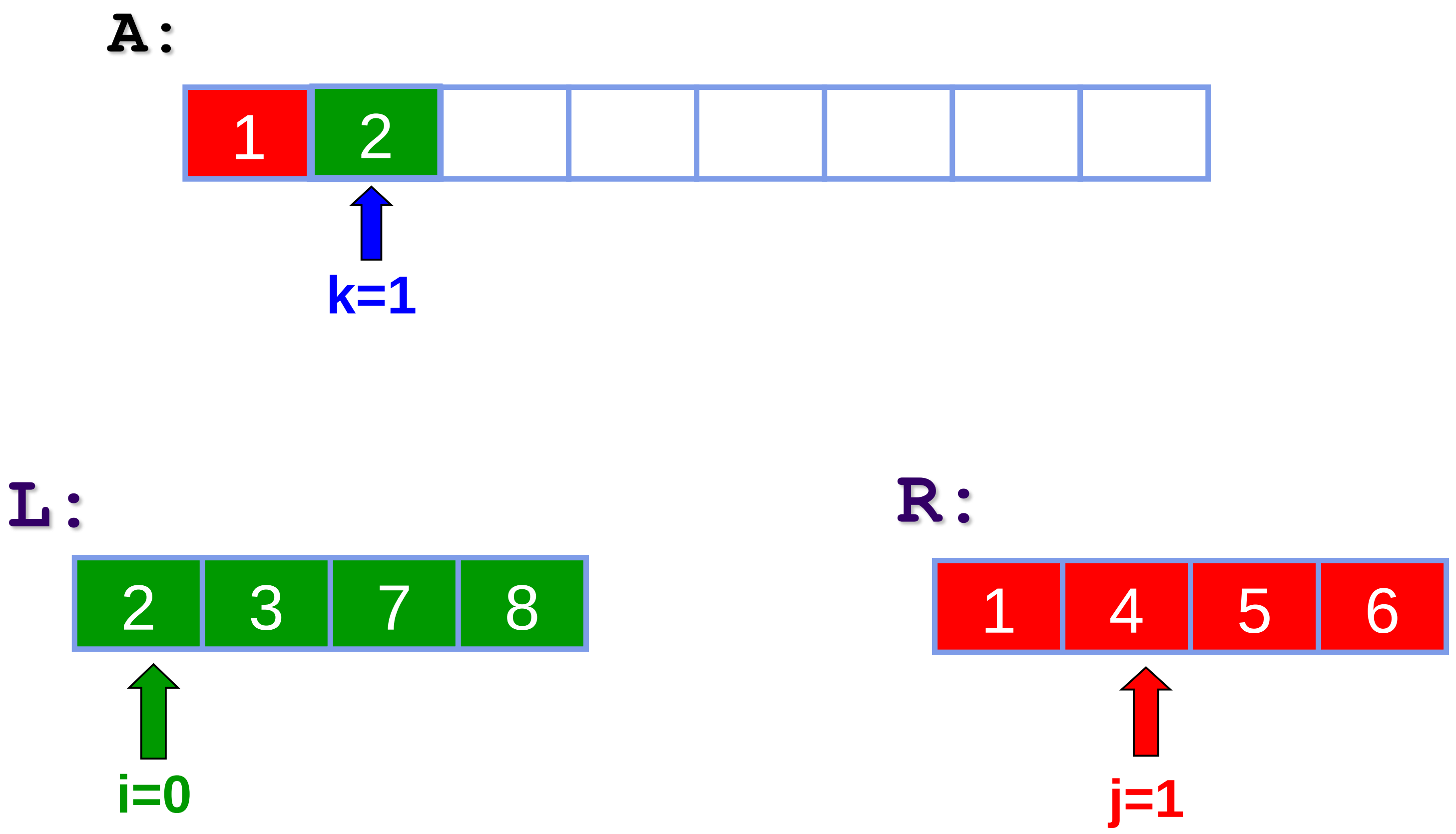
Merge-Sort: Merge Example



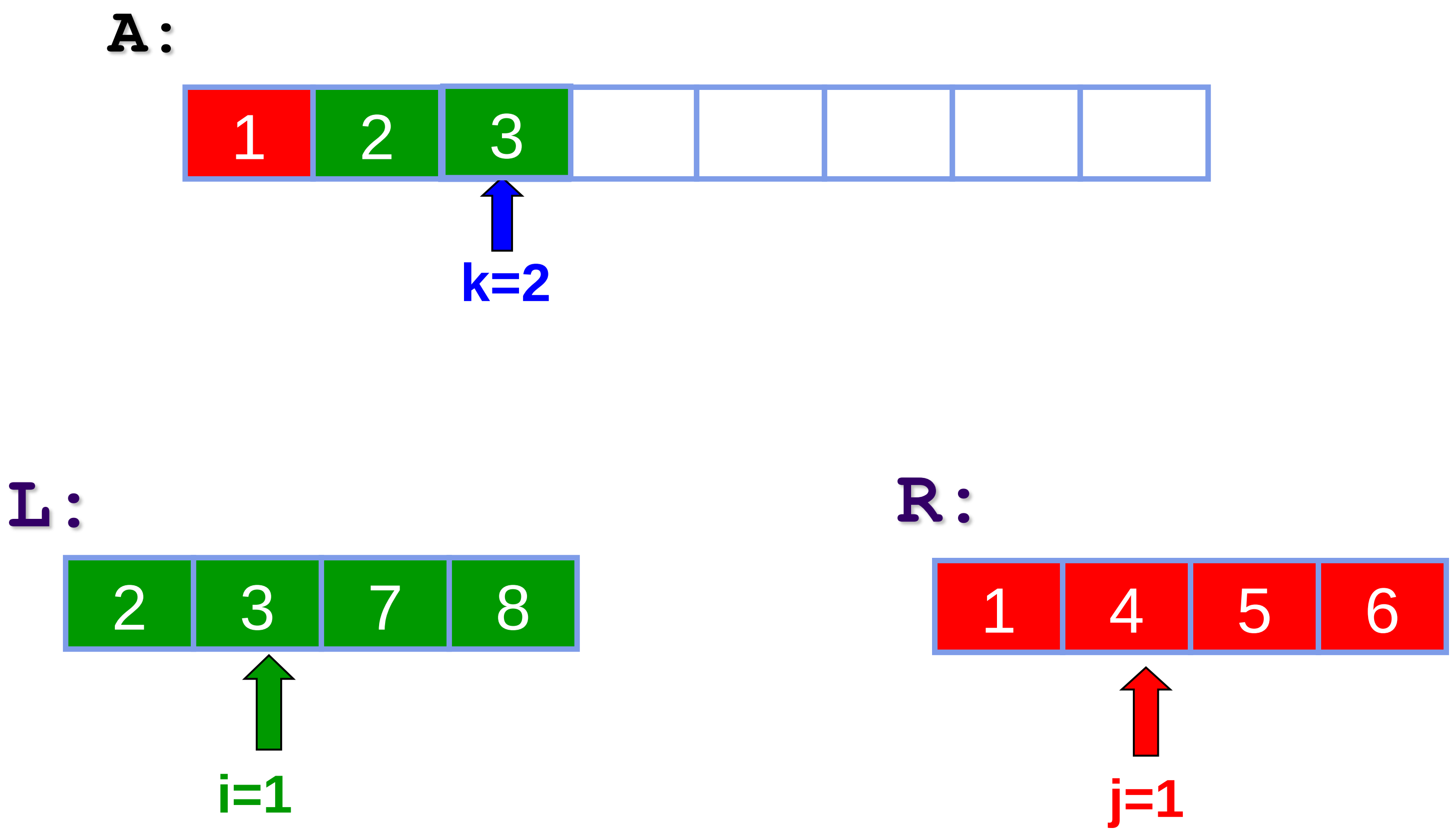
Merge-Sort: Merge Example



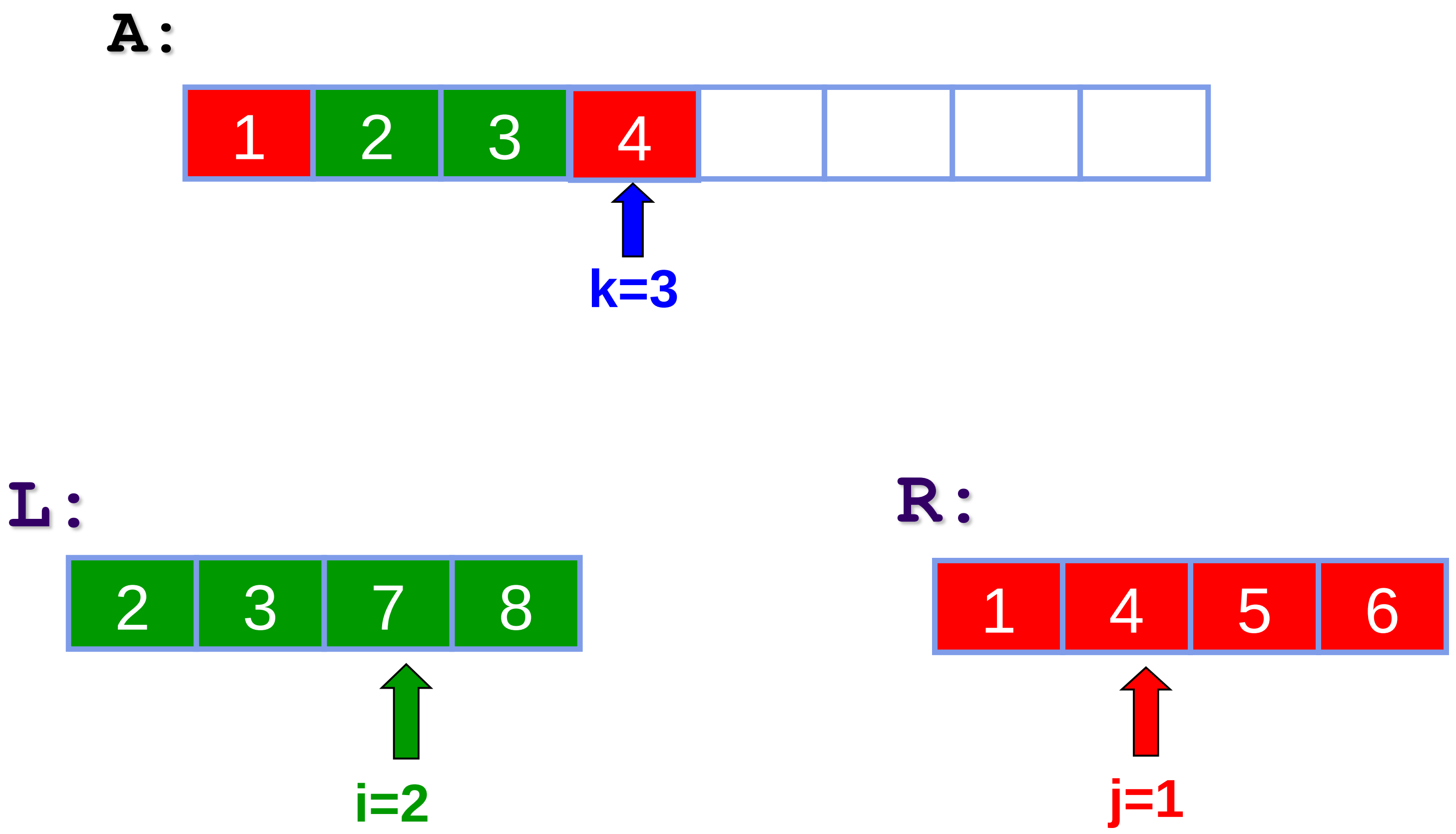
Merge-Sort: Merge Example



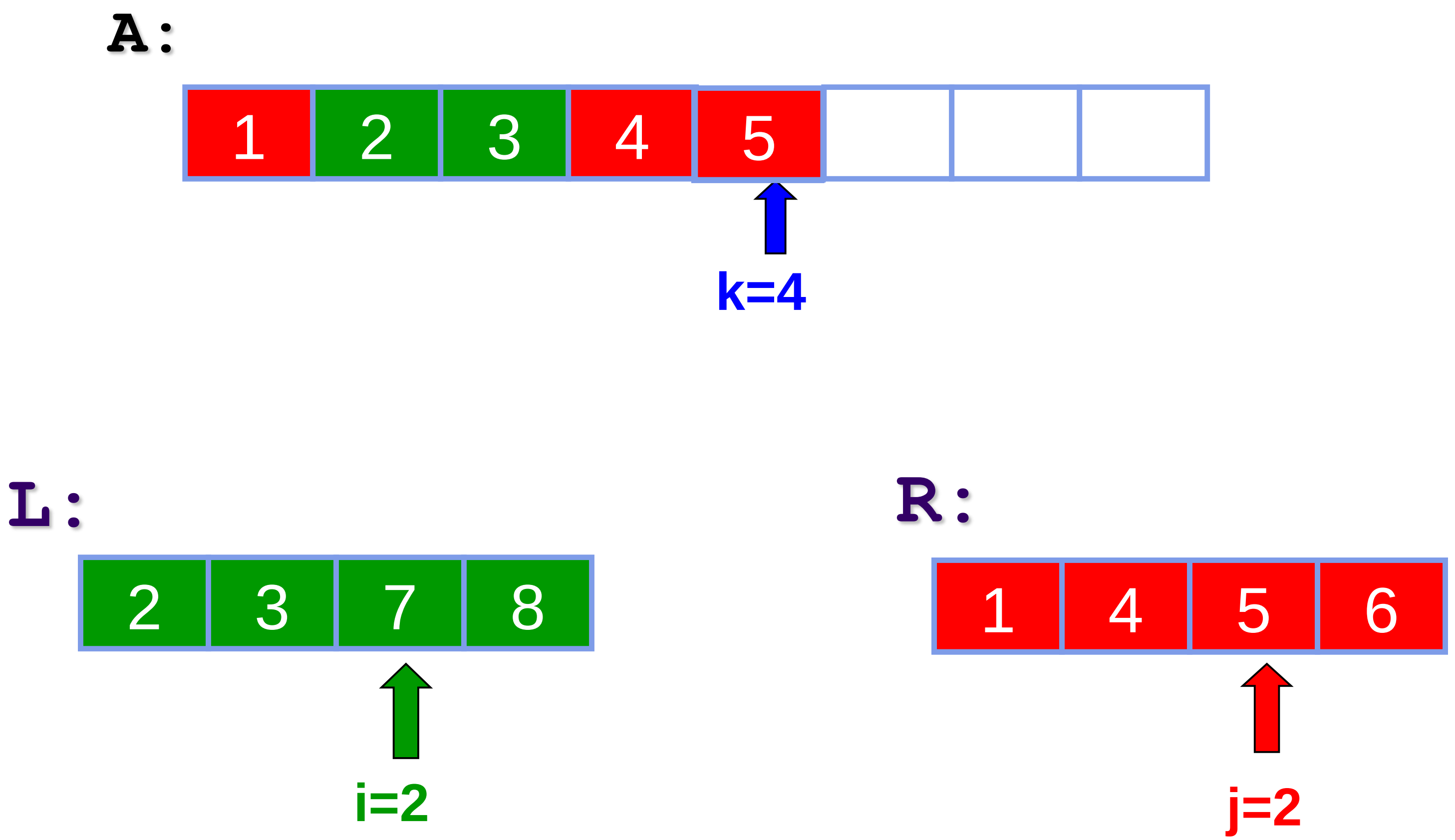
Merge-Sort: Merge Example



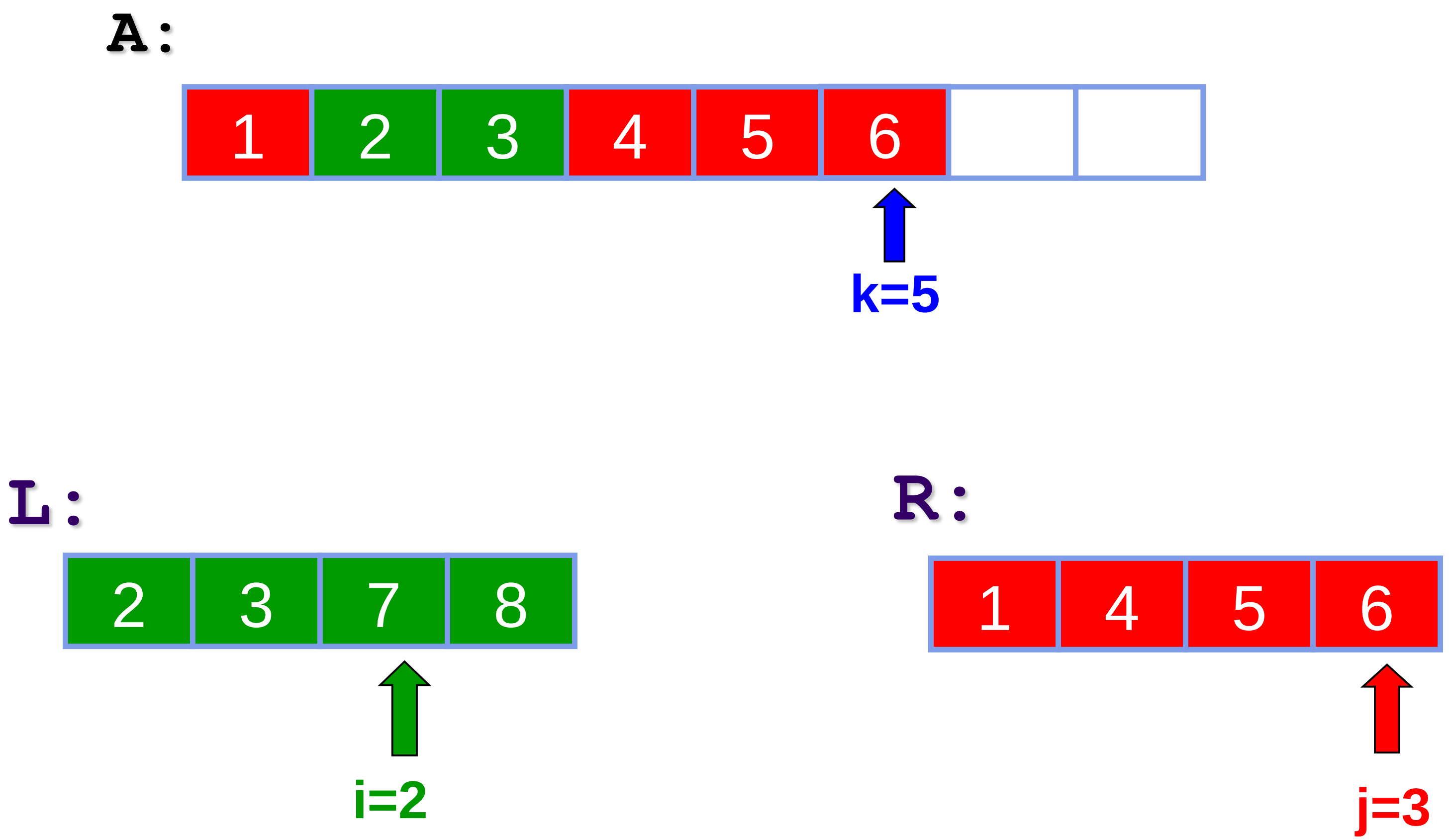
Merge-Sort: Merge Example



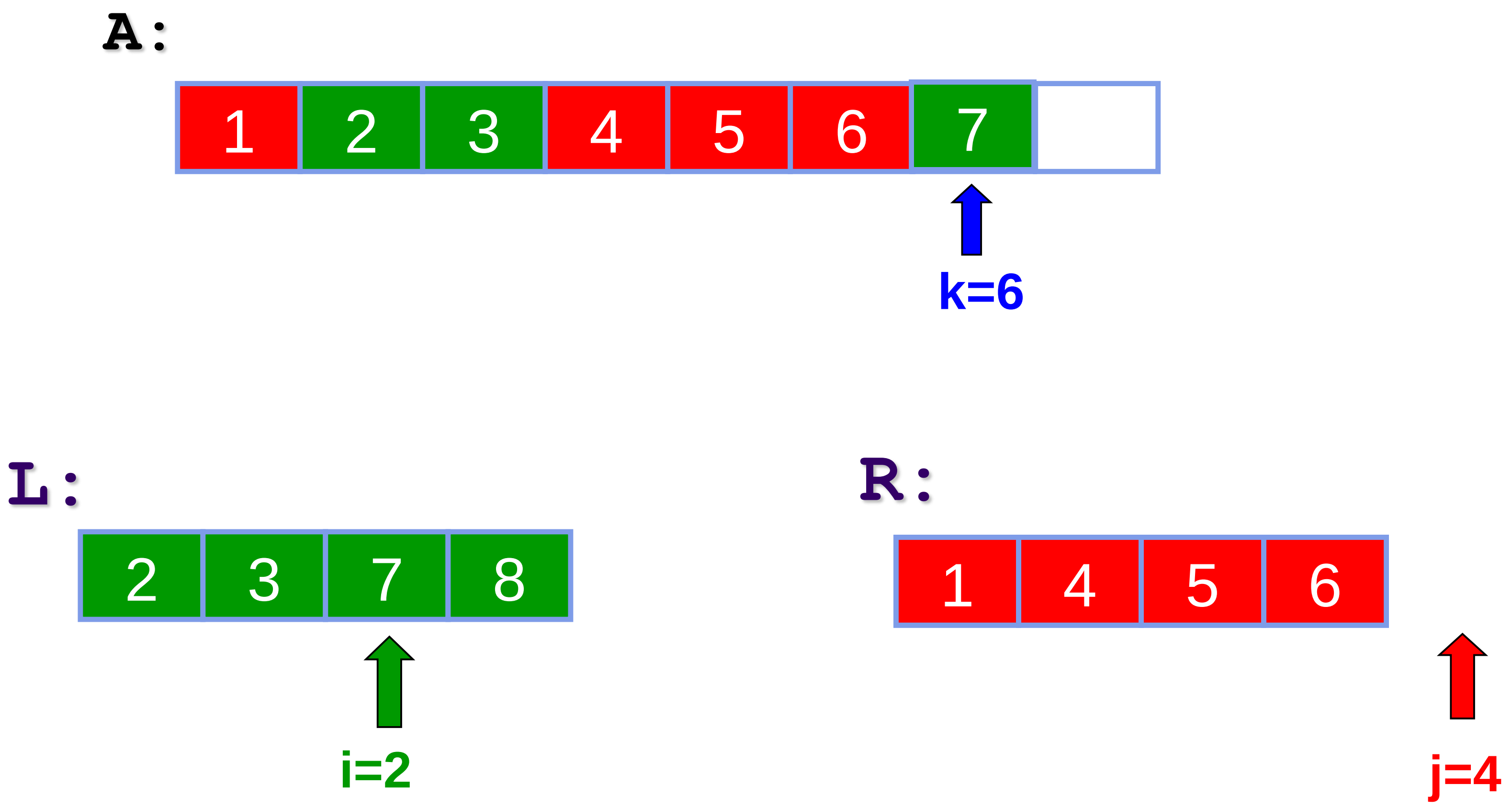
Merge-Sort: Merge Example



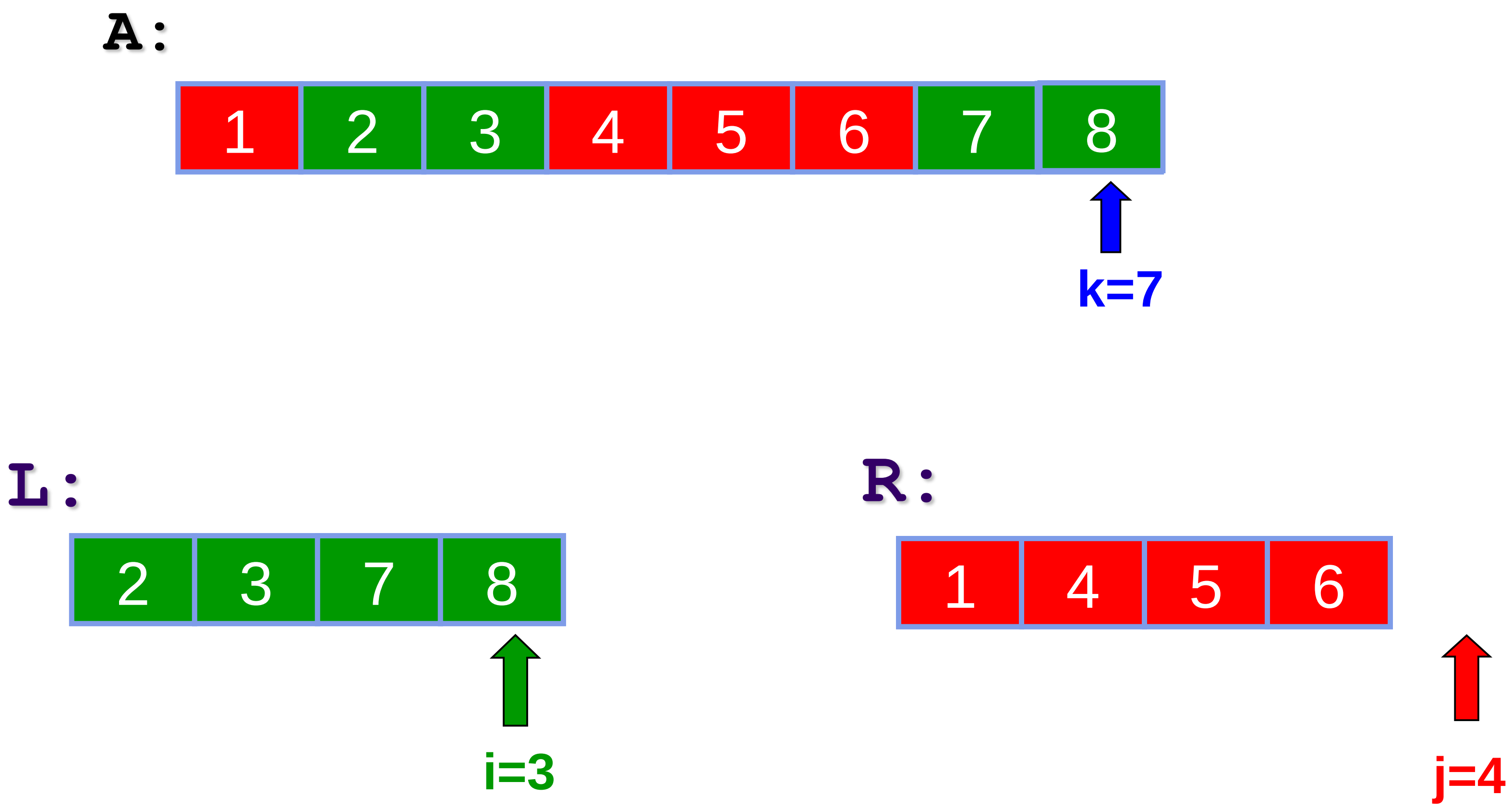
Merge-Sort: Merge Example



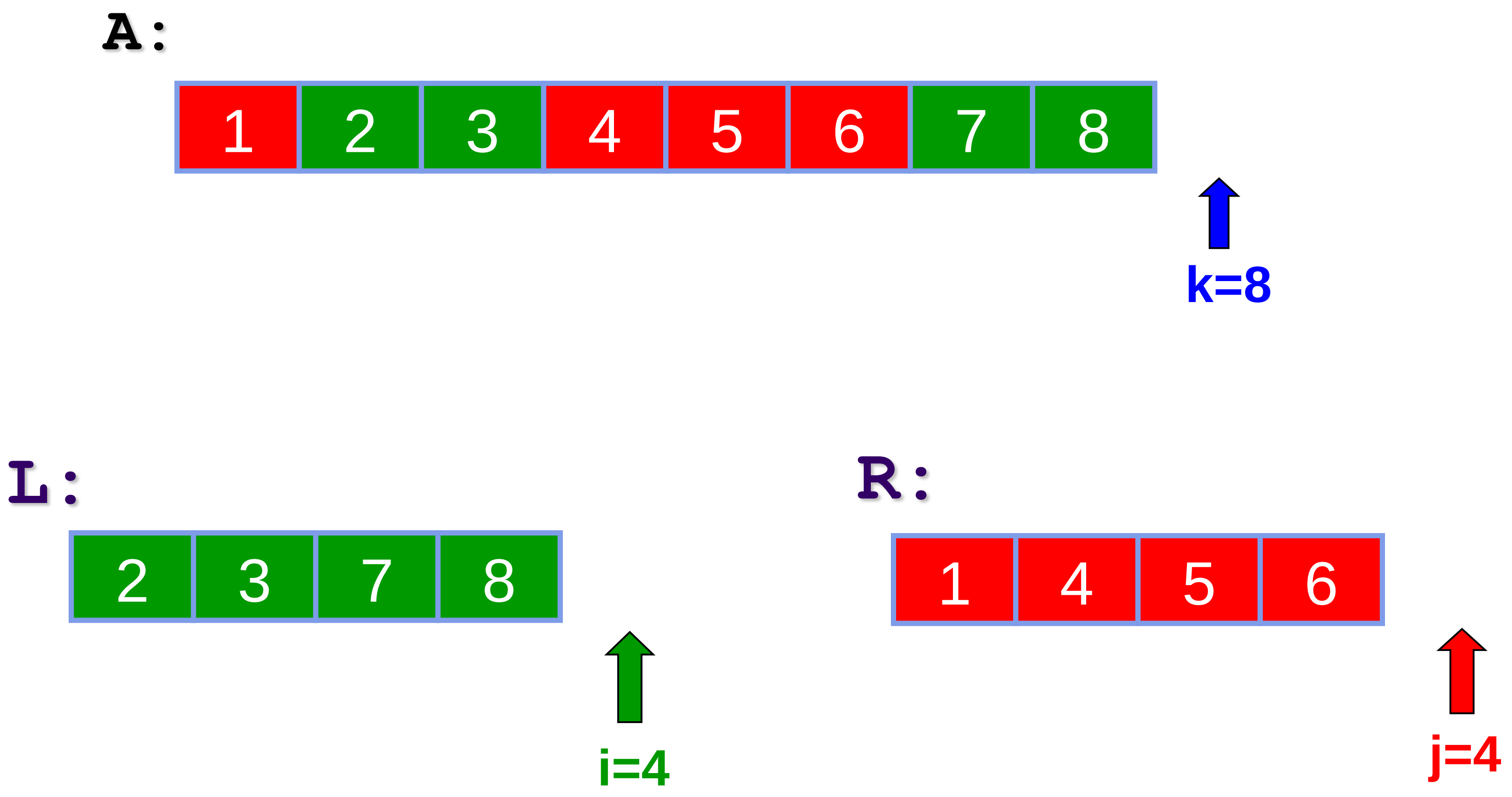
Merge-Sort: Merge Example



Merge-Sort: Merge Example



Merge-Sort: Merge Example



Merge(A, left, middle, right)

```
$n_1$ ← middle - left + 1
$n_2$ ← right - middle
create array L[$n_1$], R[$n_2$]
for i ← 0 to $n_1$-1 do L[i] ← A[left +i]
for j ← 0 to $n_2$-1 do R[j] ← A[middle+j]
k ← i ← j ← 0
while i < $n_1$ &  j < $n_2$
      if L[i] < R[j]
            A[k++] ← L[i++]
      else
            A[k++] ← R[j++]
while i < $n_1$
   A[k++] ← L[i++]
while j < $n_2$
   A[k++] ← R[j++]
```

$n = n_1+n_2$

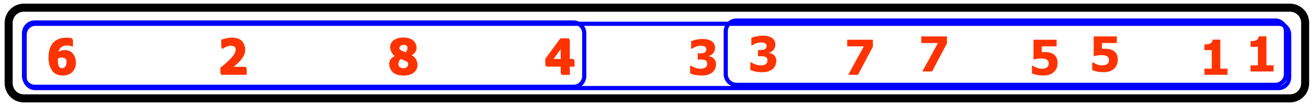
Space: n

Time : cn for some constant c

Merge-Sort(A, 0, 7)

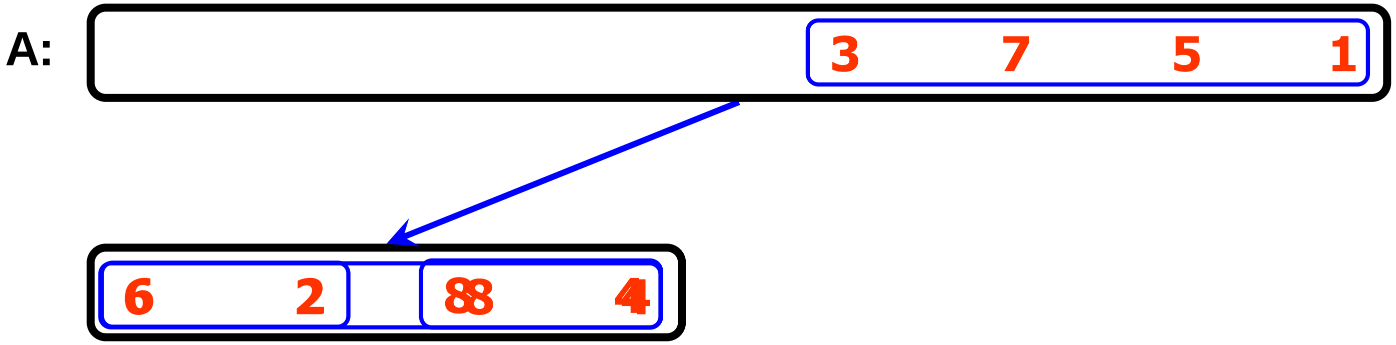
Divide

A:



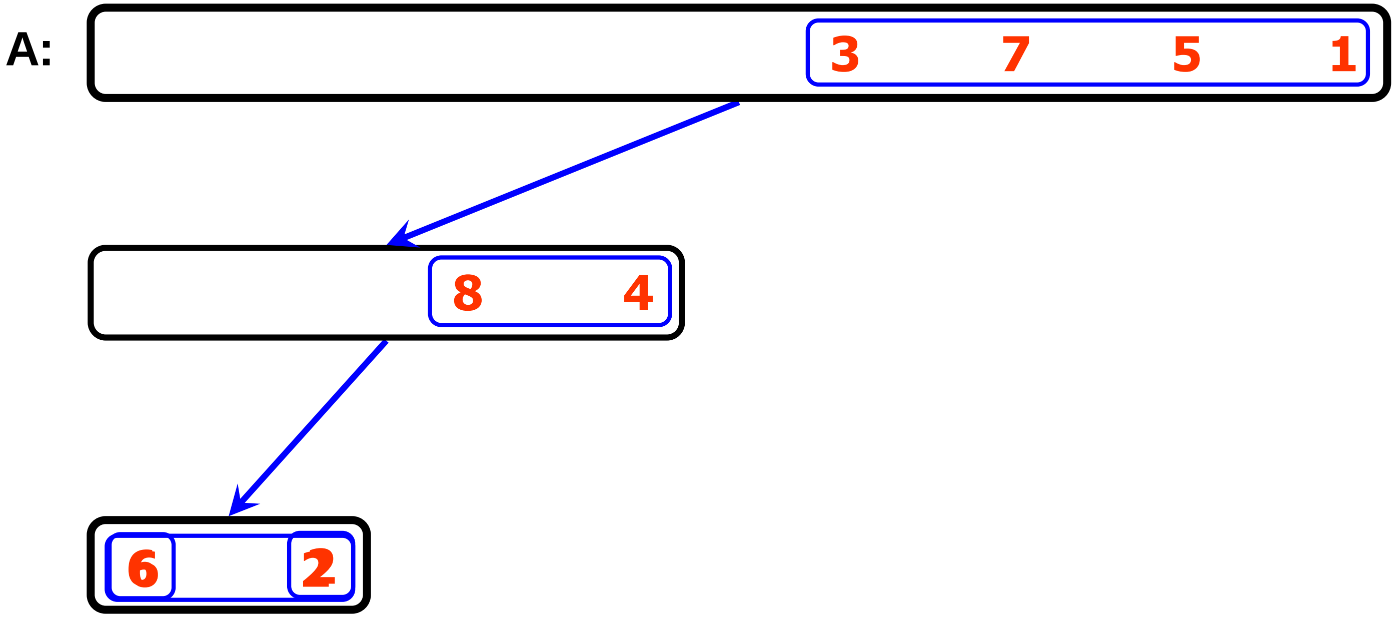
Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 3) , divide



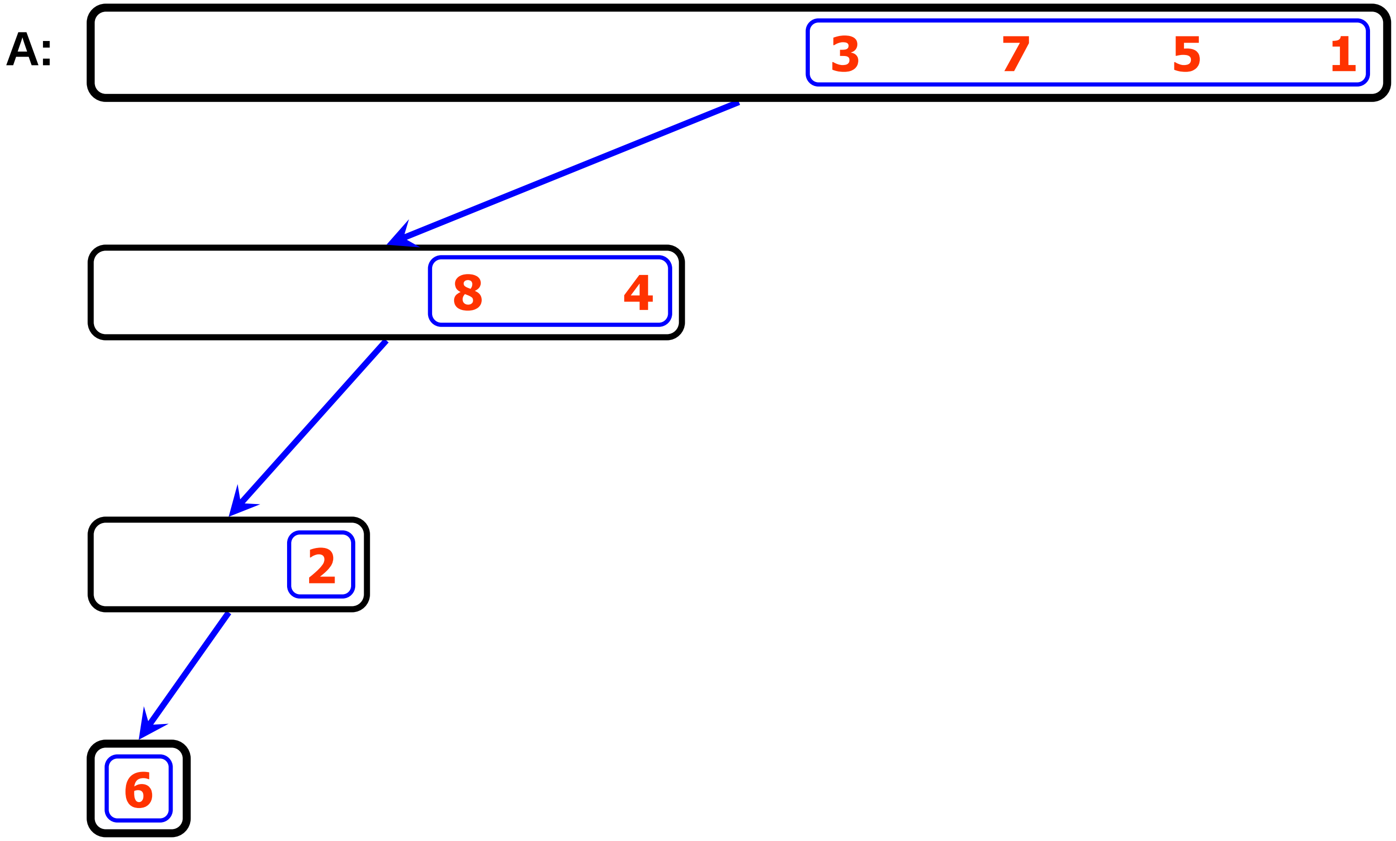
Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 1), divide



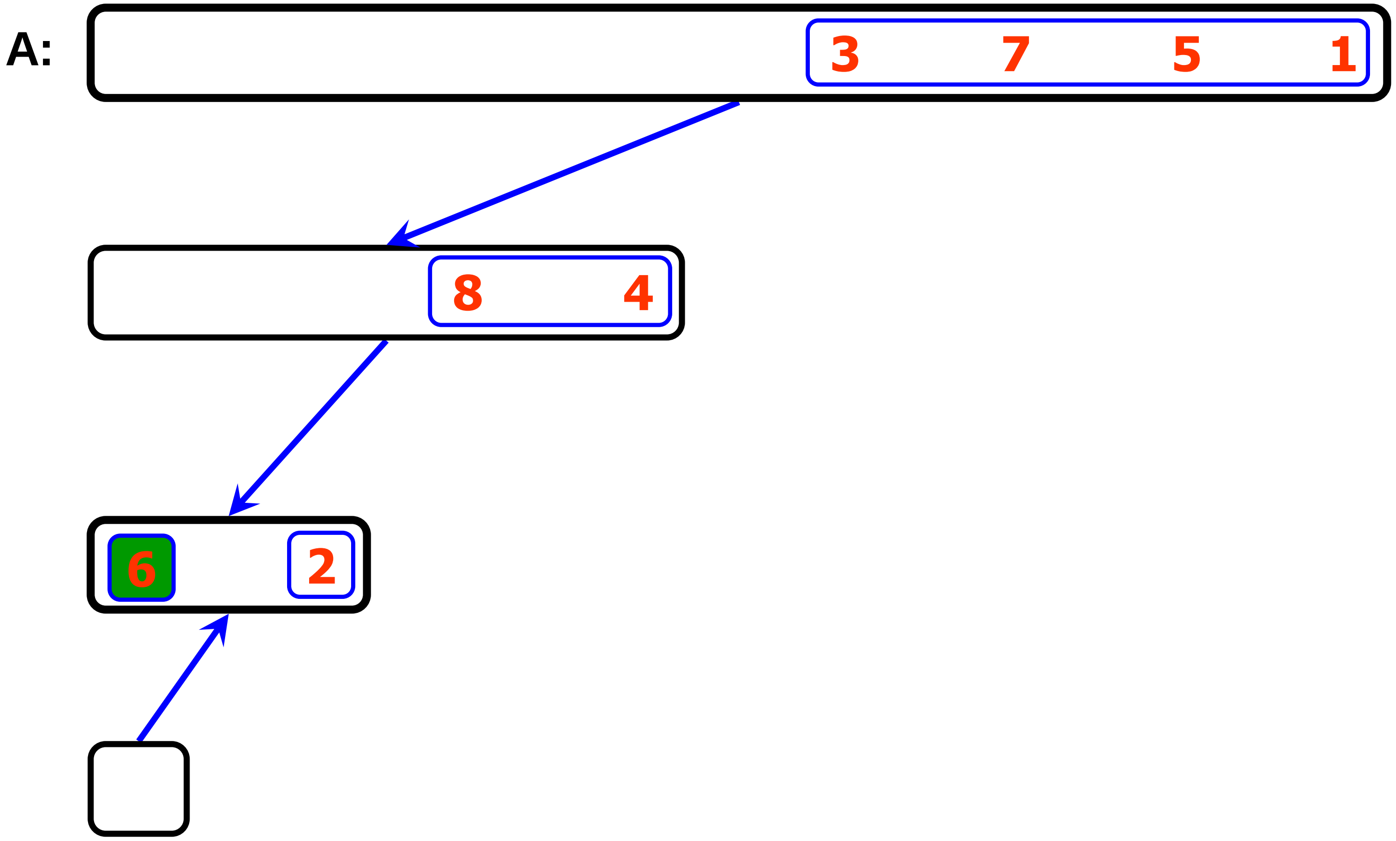
Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 0) , base case



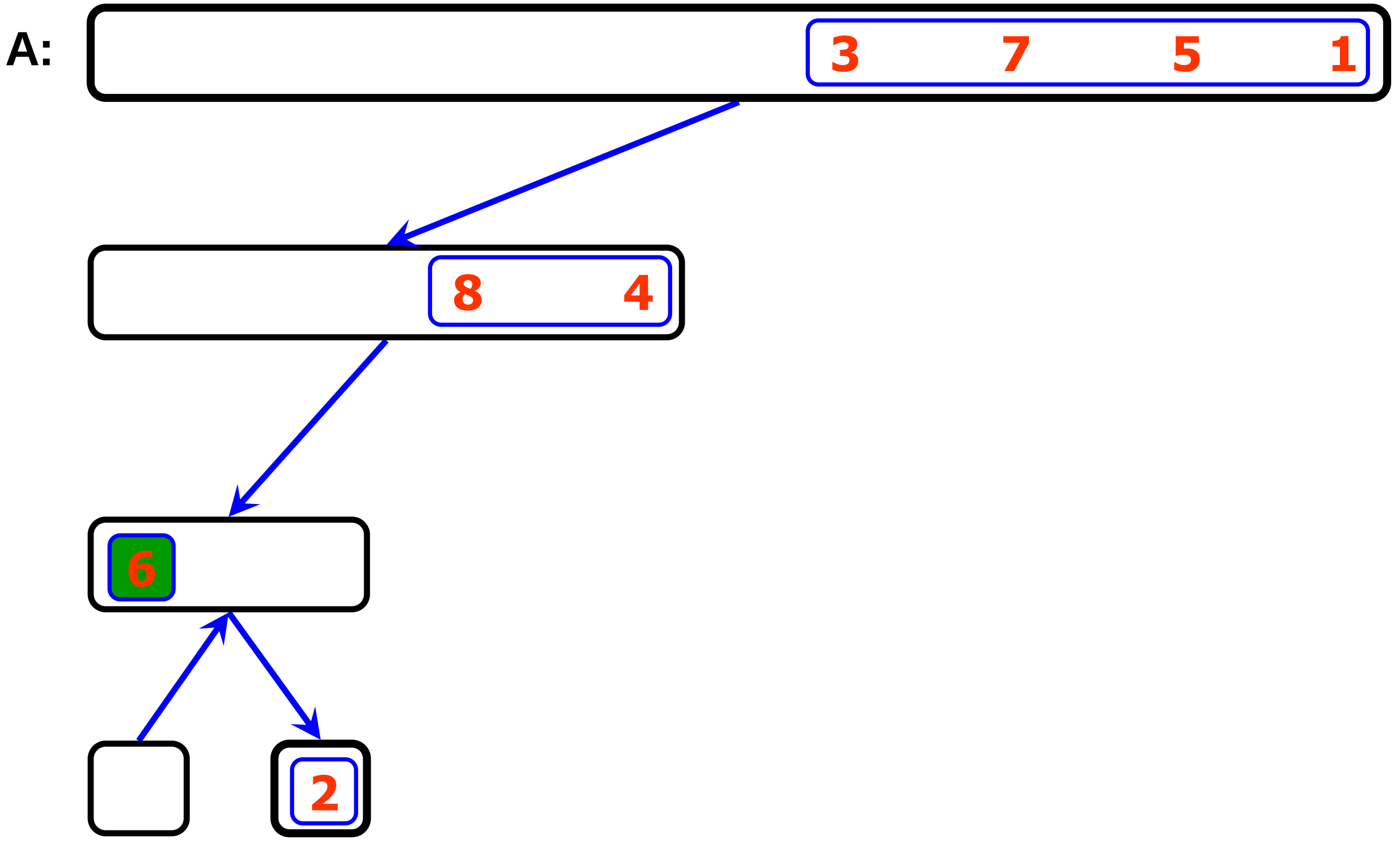
Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 0), return



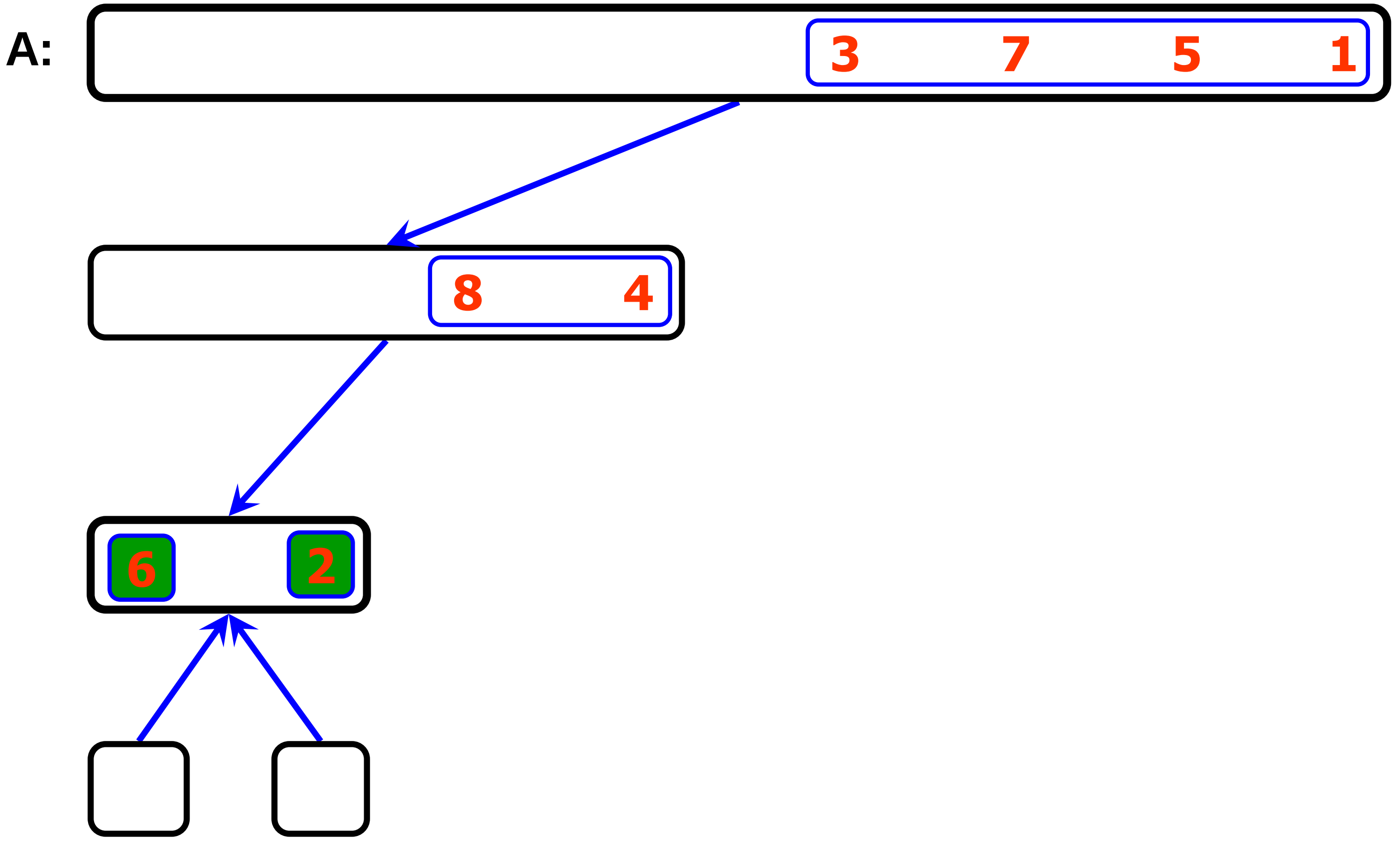
Merge-Sort(A, 0, 7)

Merge-Sort(A, 1, 1), base case



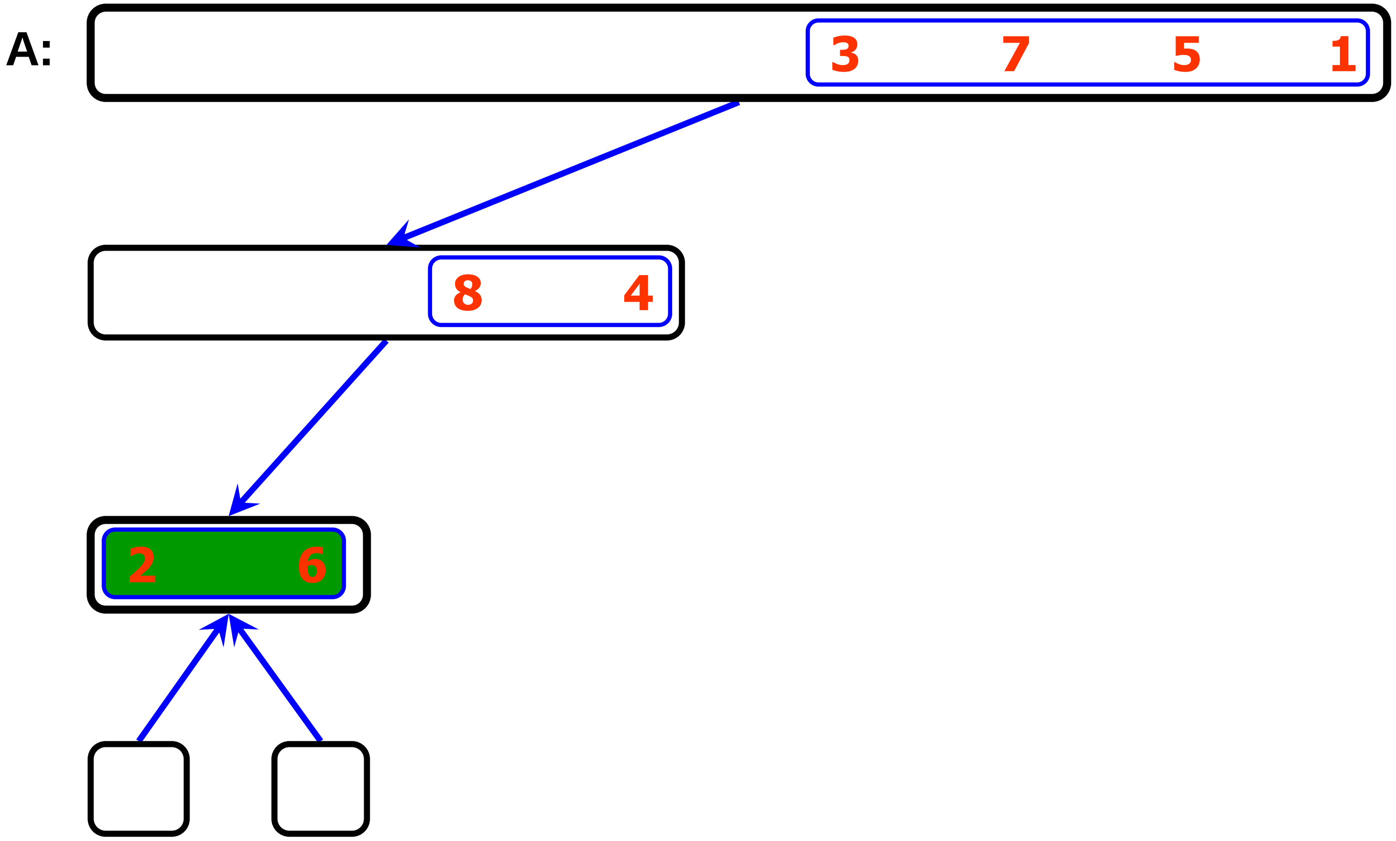
Merge-Sort(A, 0, 7)

Merge-Sort(A, 1, 1), return



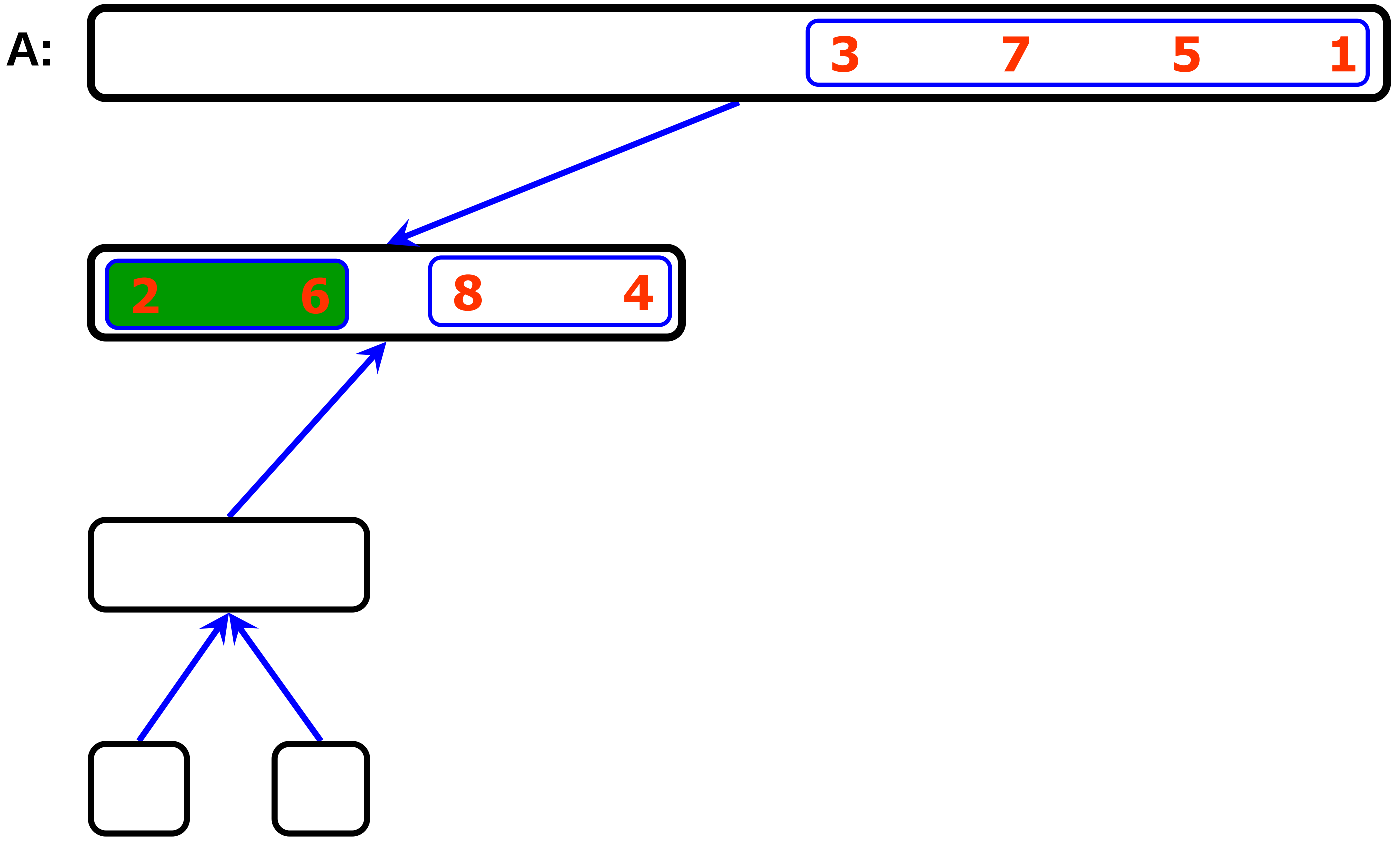
Merge-Sort(A, 0, 7)

Merge(A, 0, 0, 1)



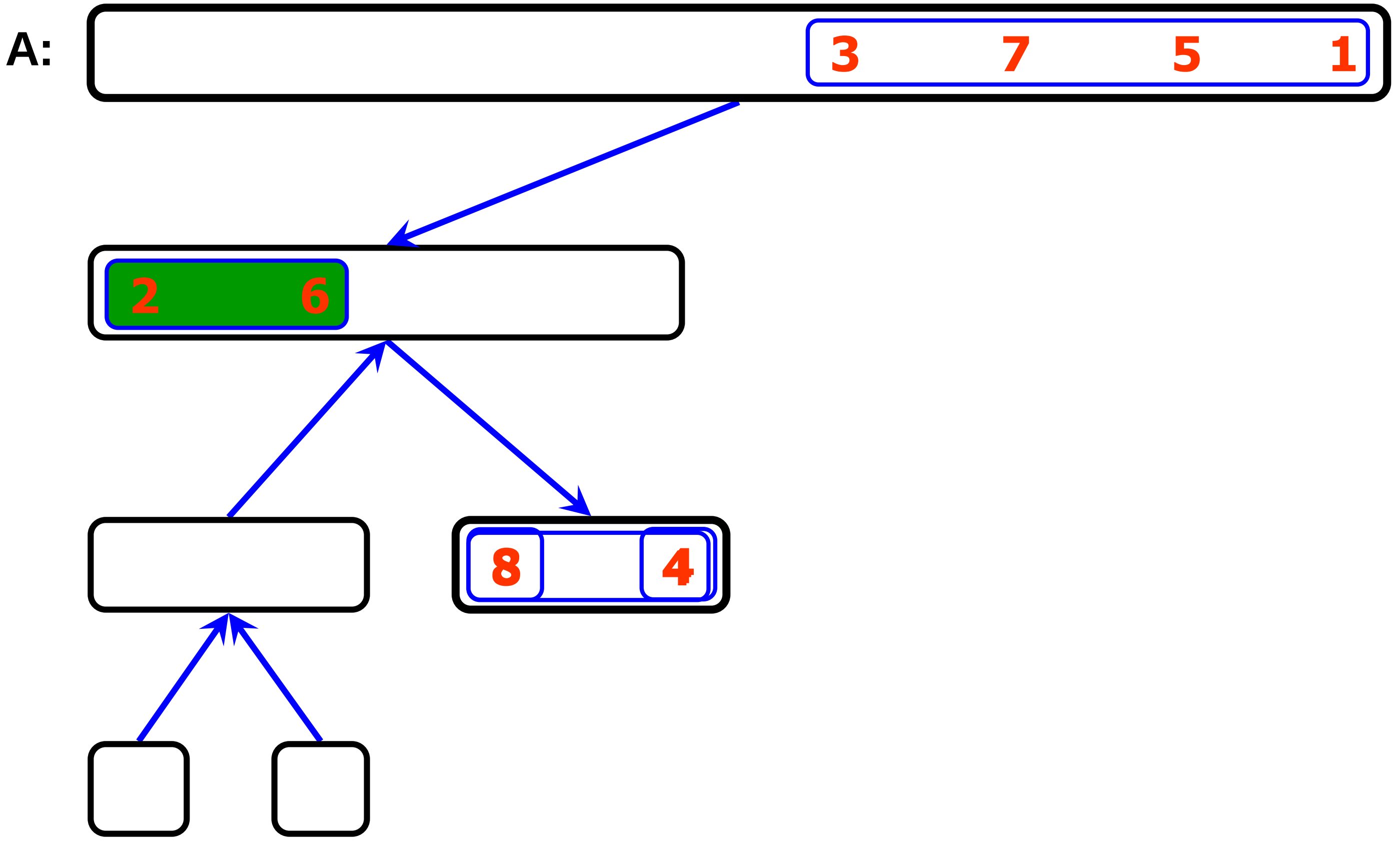
Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 1), return



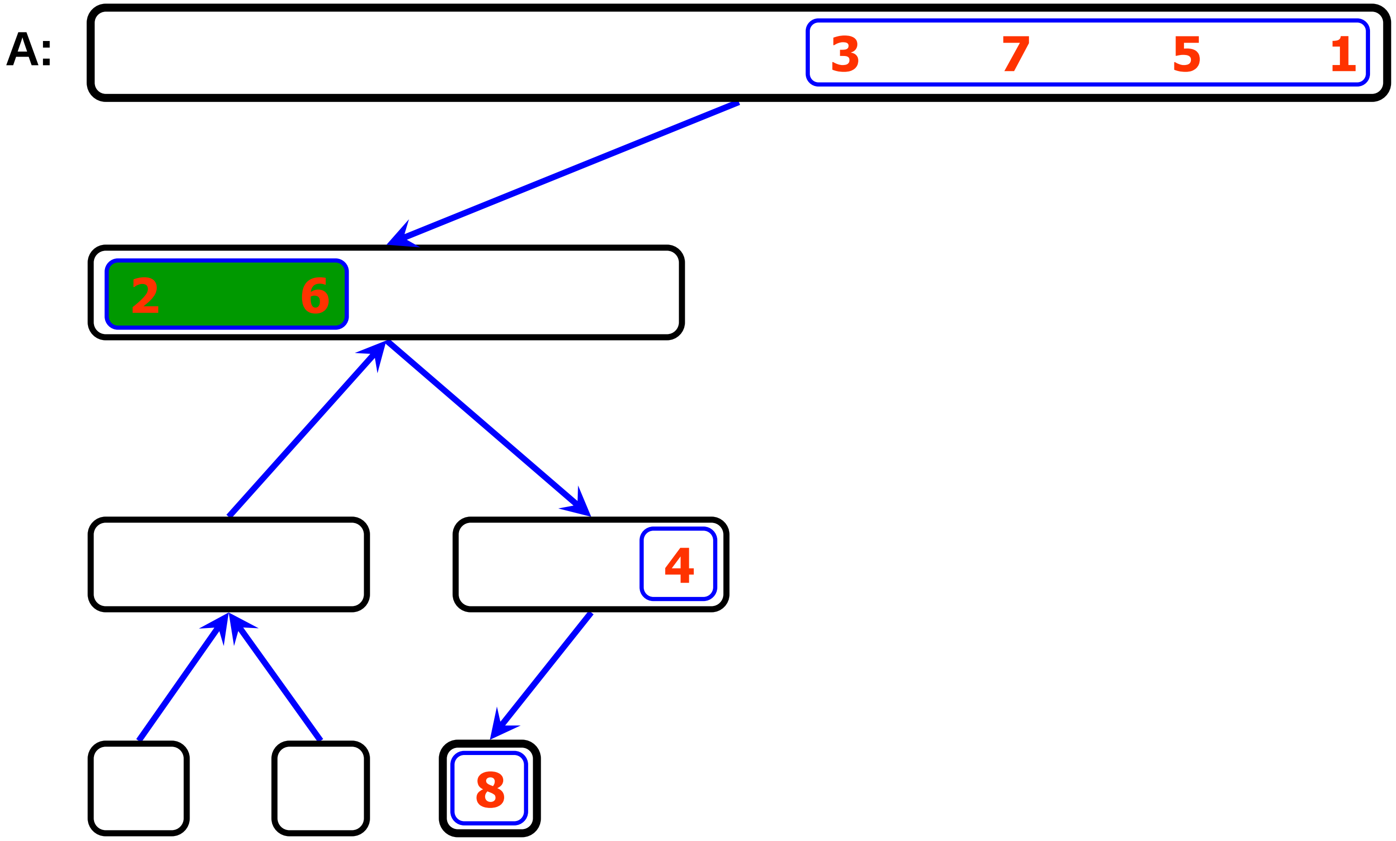
Merge-Sort(A, 0, 7)

Merge-Sort(A, 2, 3) , divide



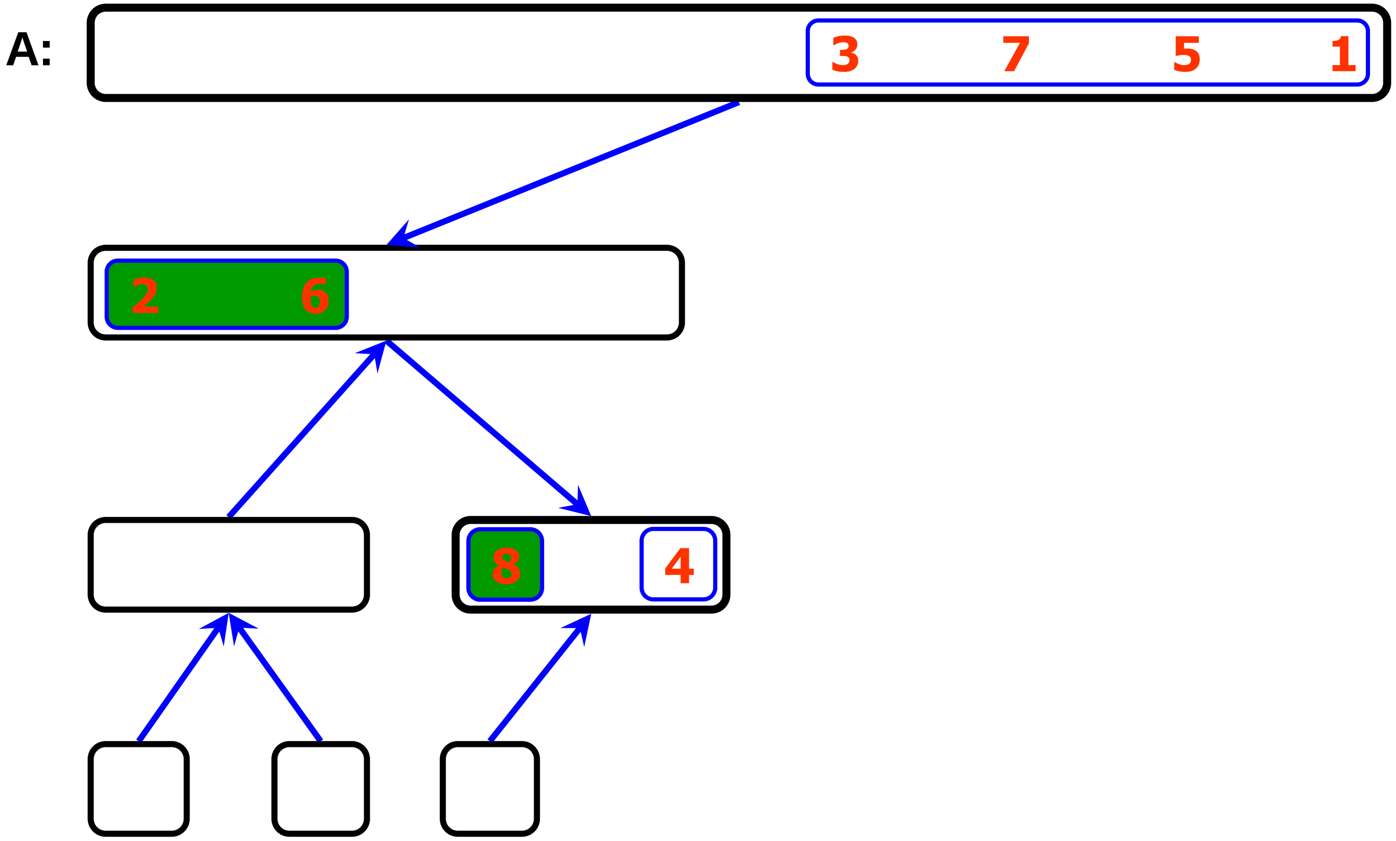
Merge-Sort(A, 0, 7)

Merge-Sort(A, 2, 2), base case



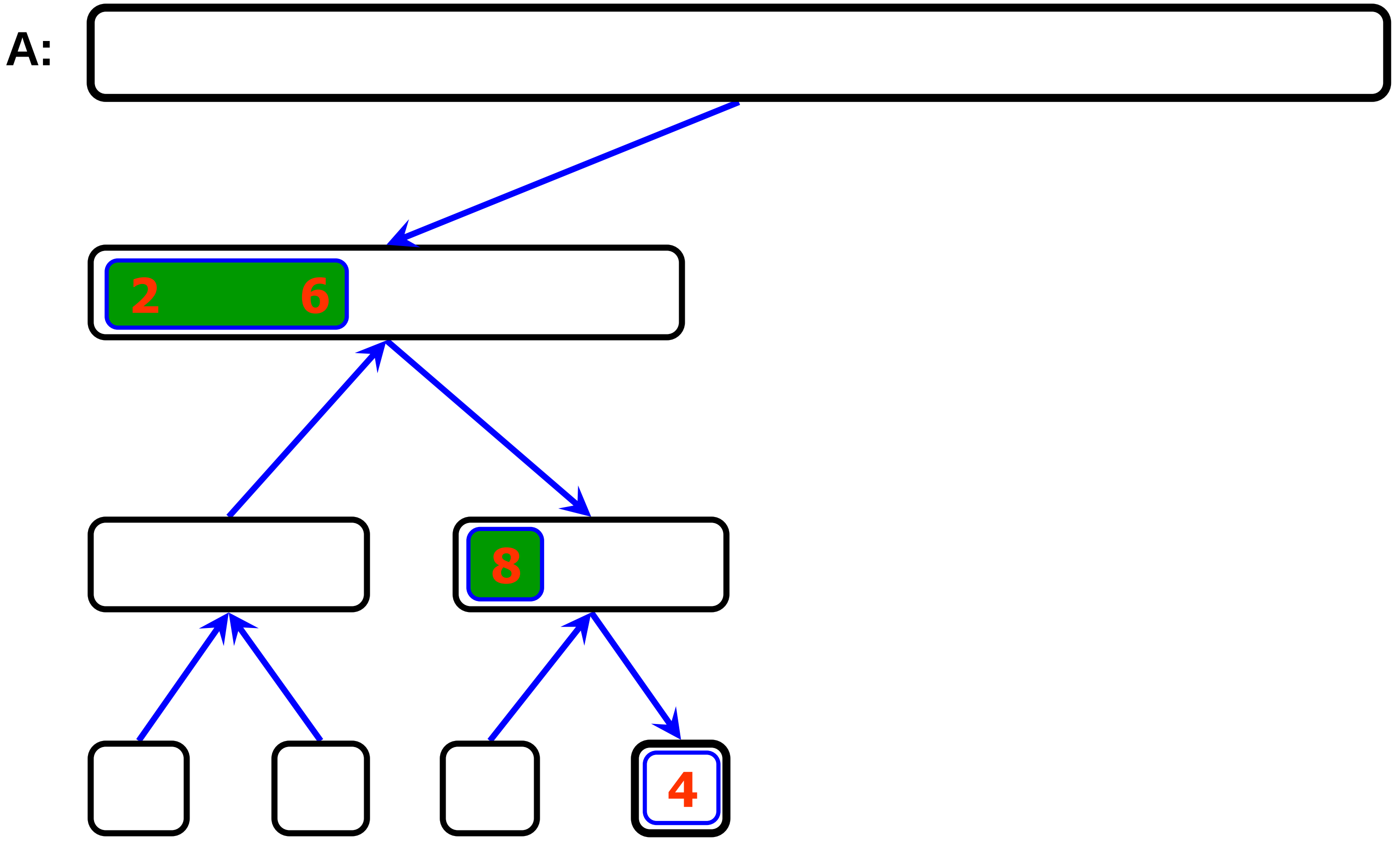
Merge-Sort(A, 0, 7)

Merge-Sort(A, 2, 2), return



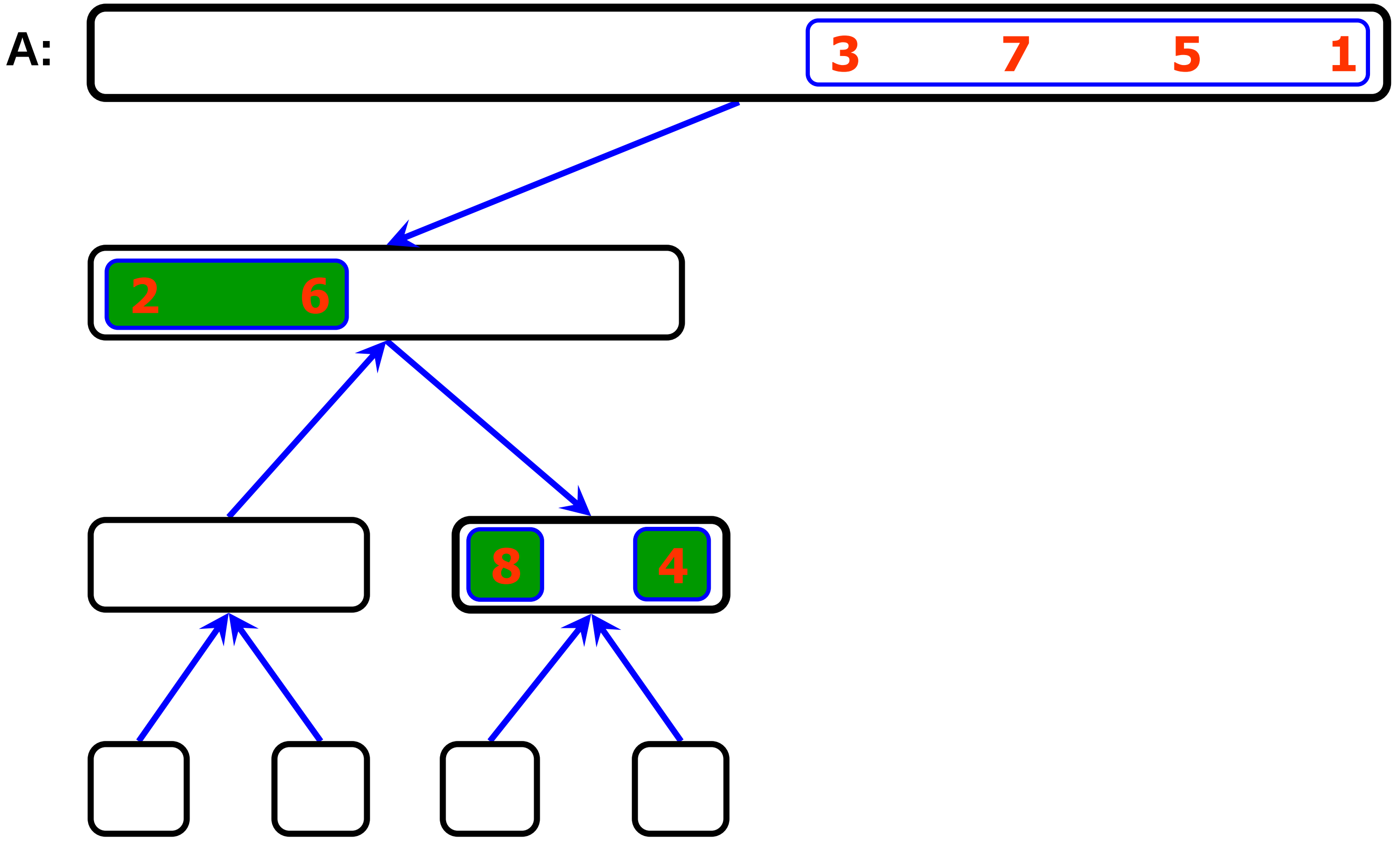
Merge-Sort(A, 0, 7)

Merge-Sort(A, 3, 3), base case



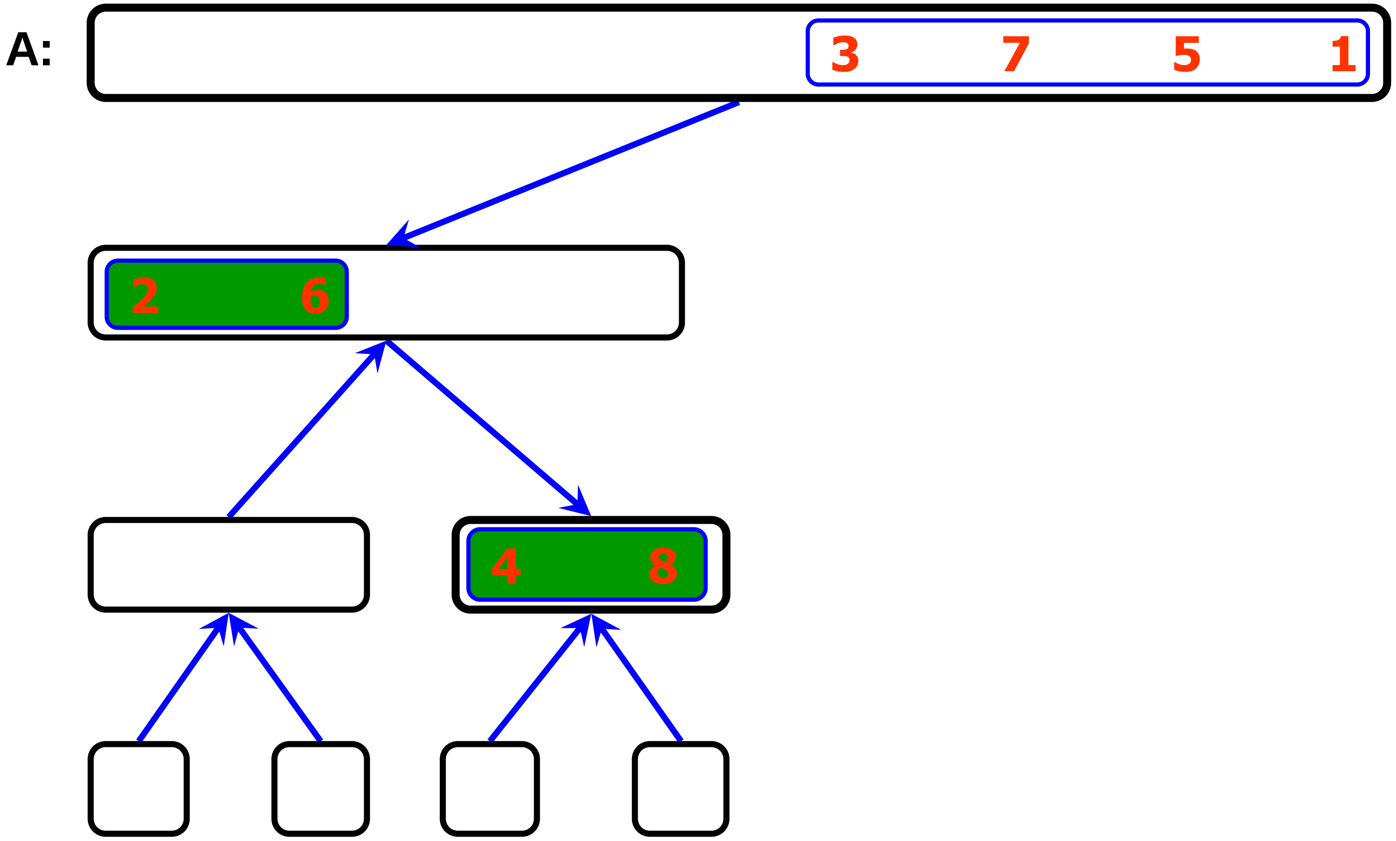
Merge-Sort(A, 0, 7)

Merge-Sort(A, 3, 3), return



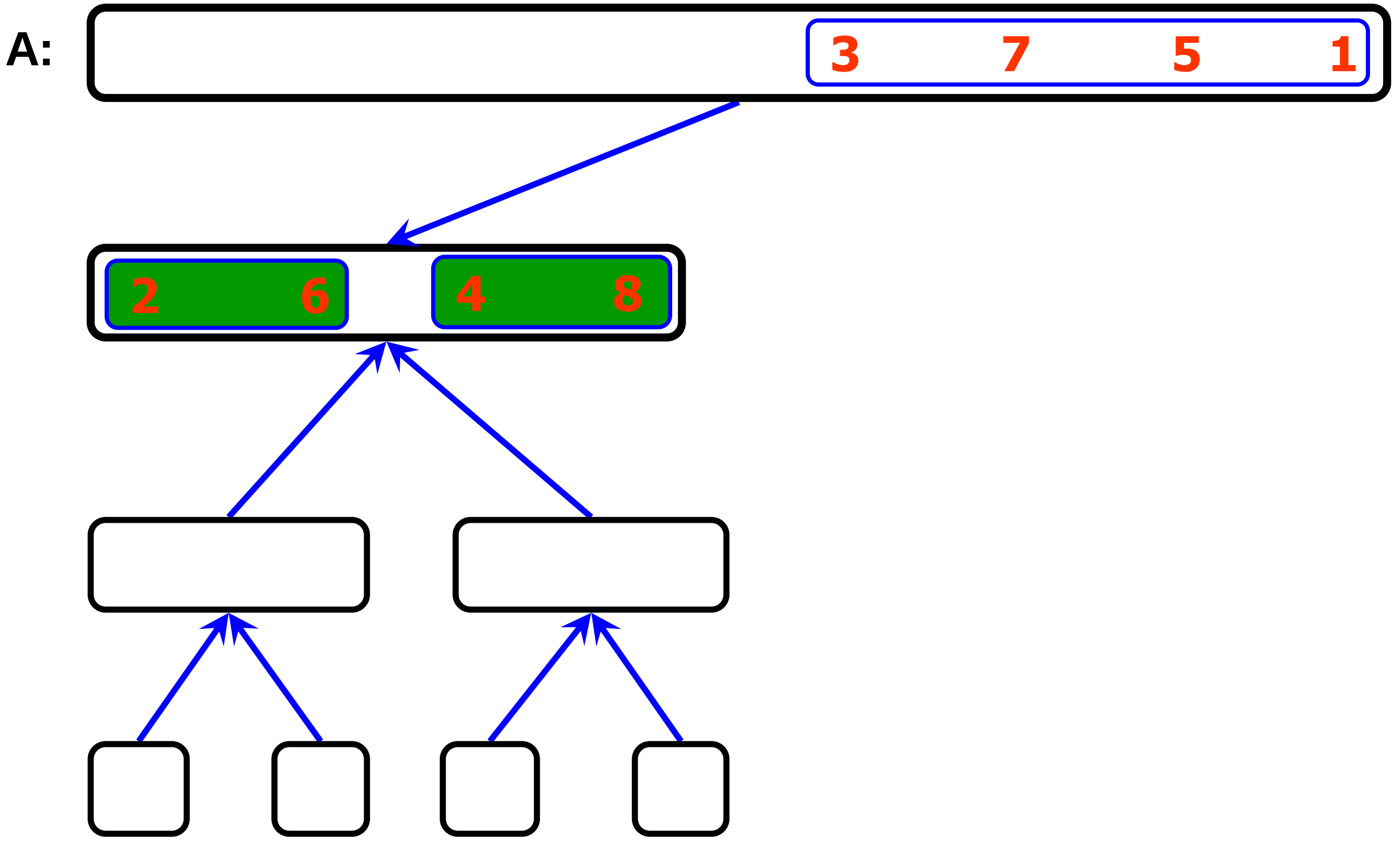
Merge-Sort(A, 0, 7)

Merge(A, 2, 2, 3)

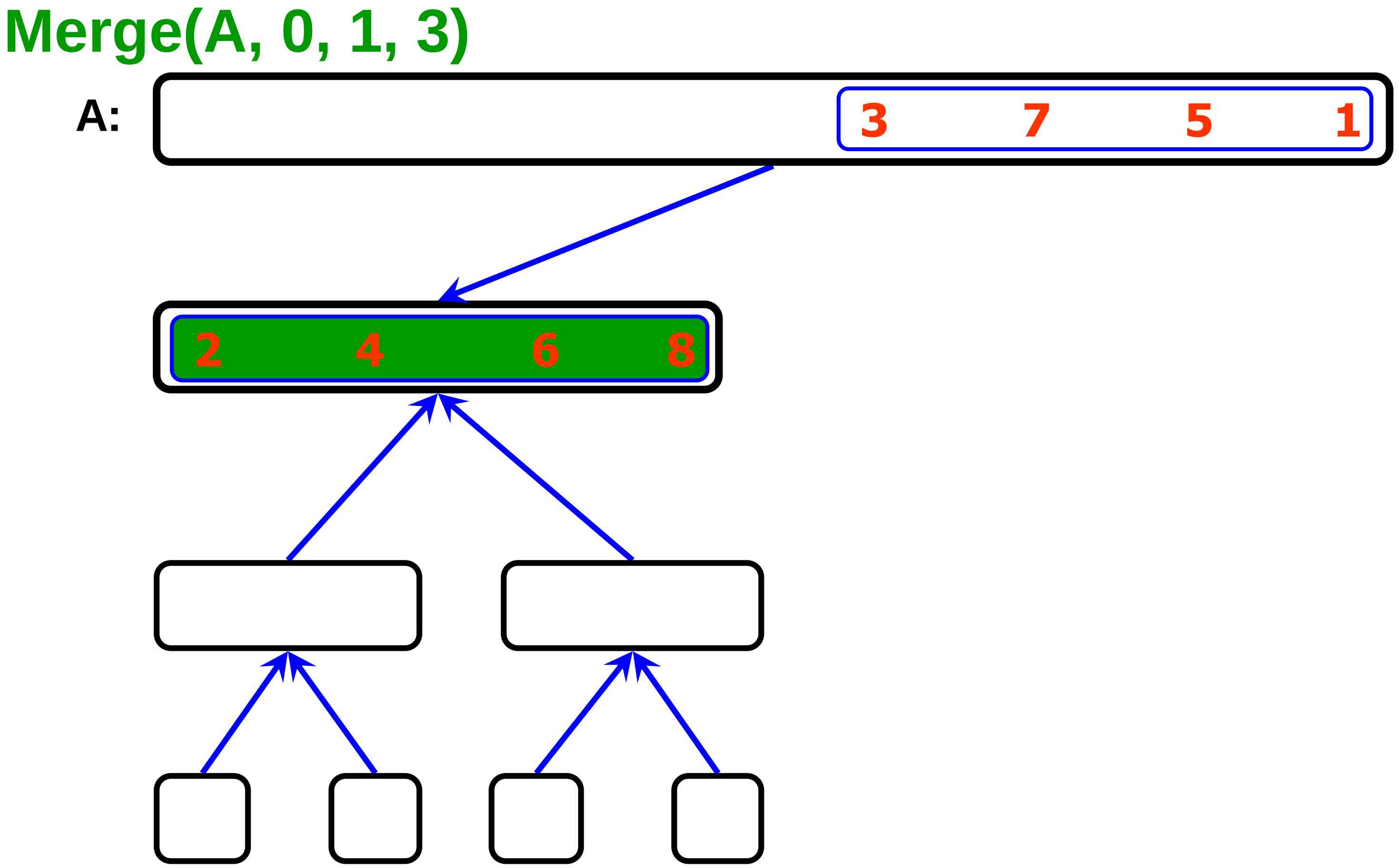


Merge-Sort(A, 0, 7)

Merge-Sort(A, 2, 3), return

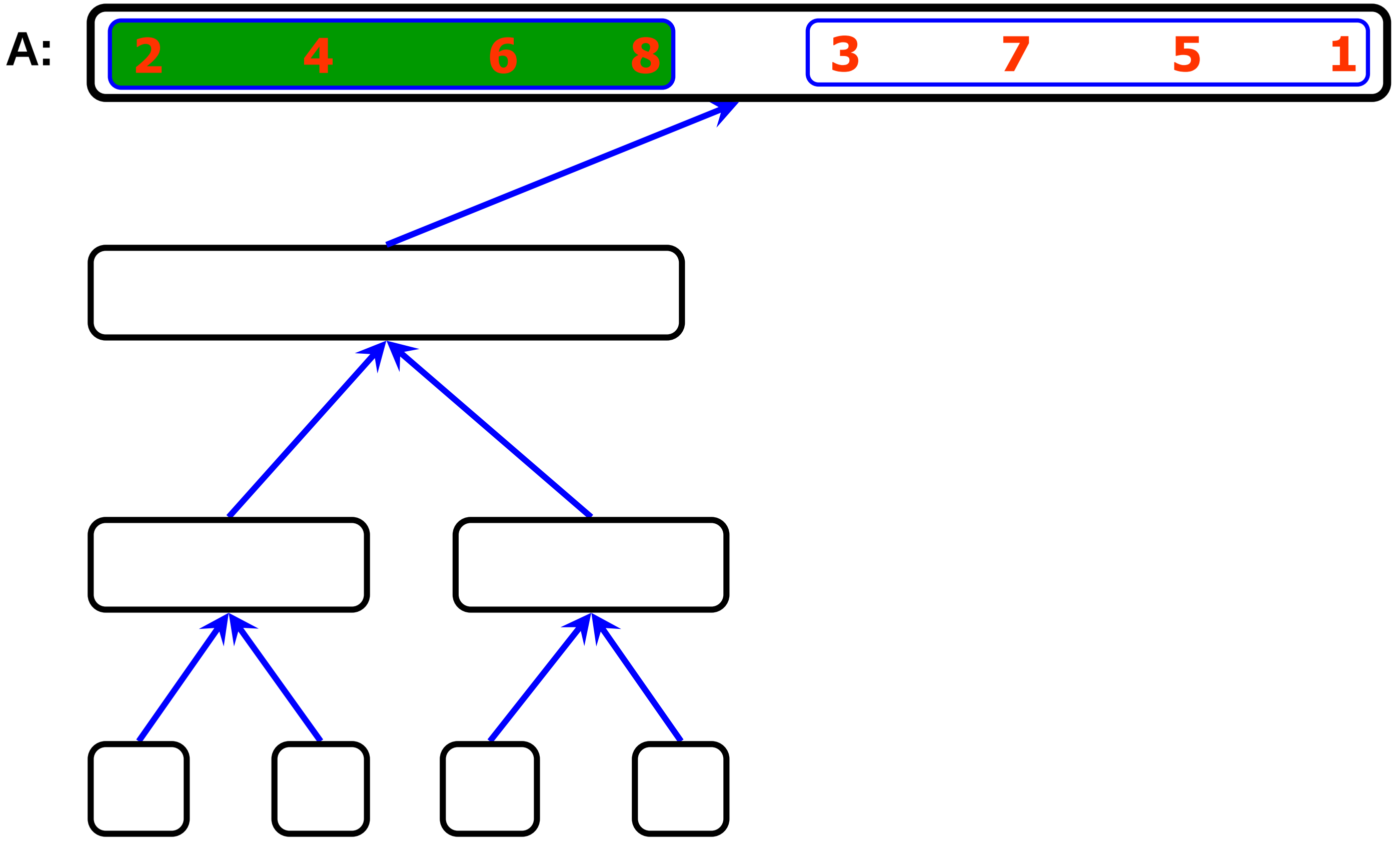


Merge-Sort(A, 0, 7)



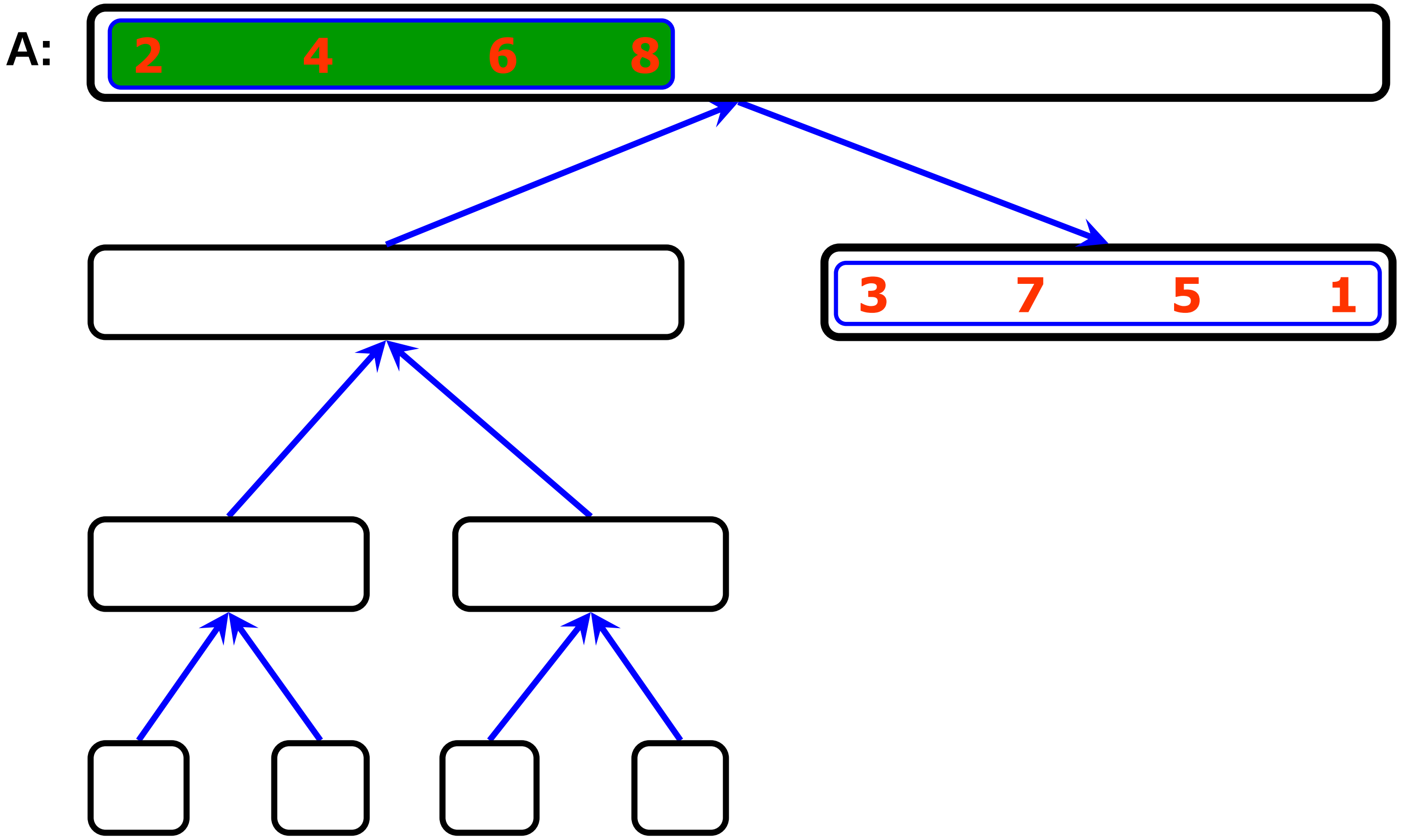
Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 3), return



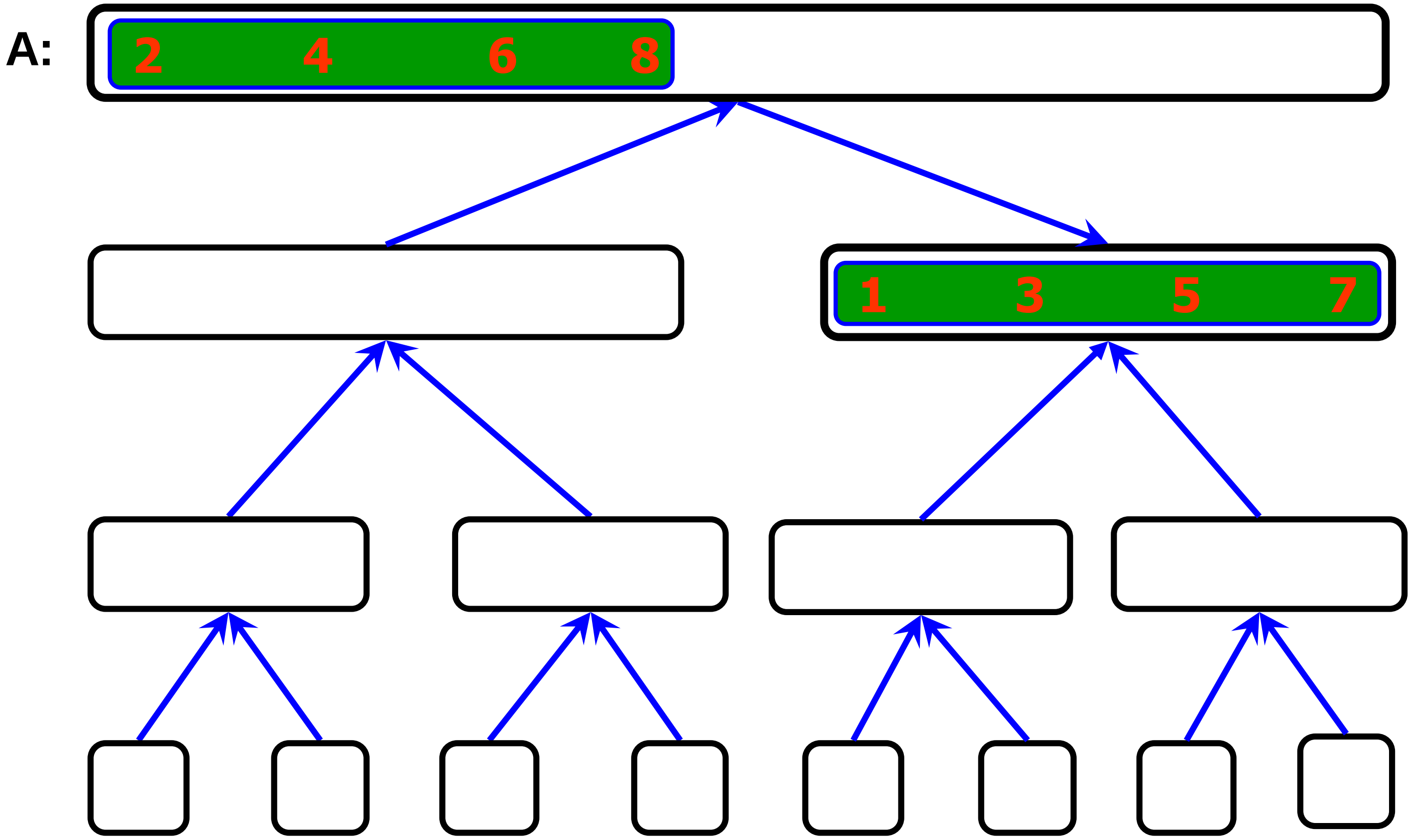
Merge-Sort(A, 0, 7)

Merge-Sort(A, 4, 7)



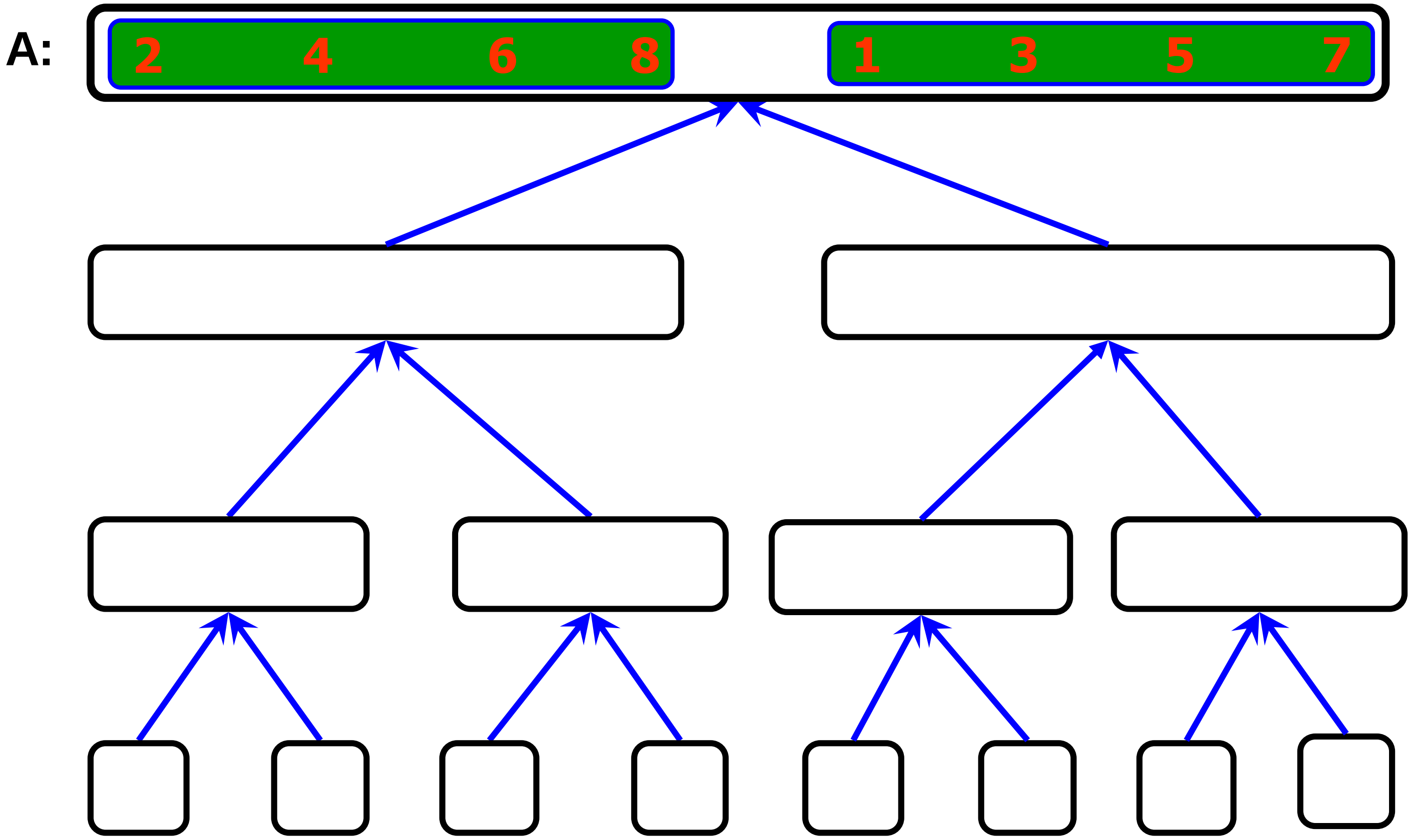
Merge-Sort(A, 0, 7)

Merge (A, 4, 5, 7)



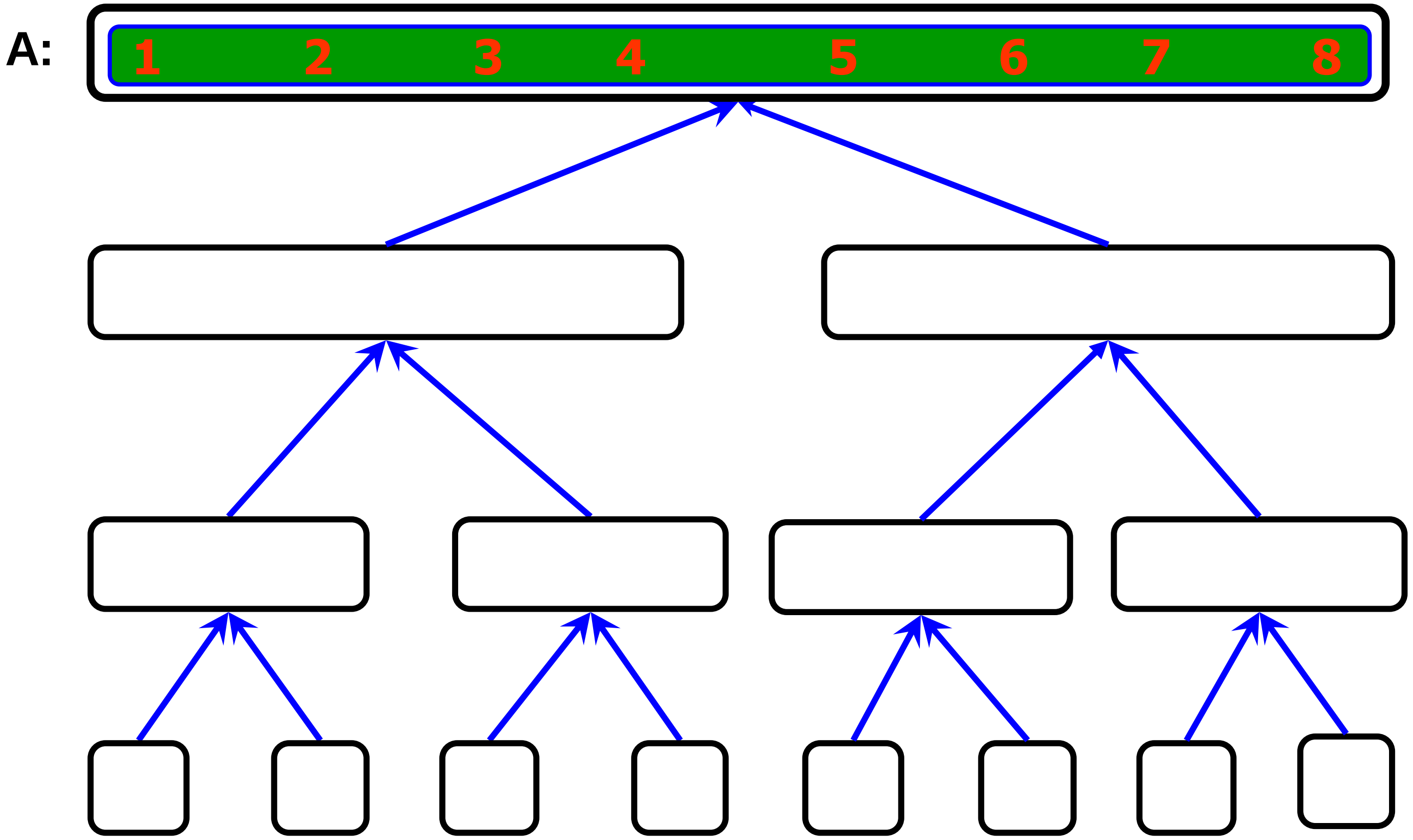
Merge-Sort(A, 0, 7)

Merge-Sort(A, 4, 7), return

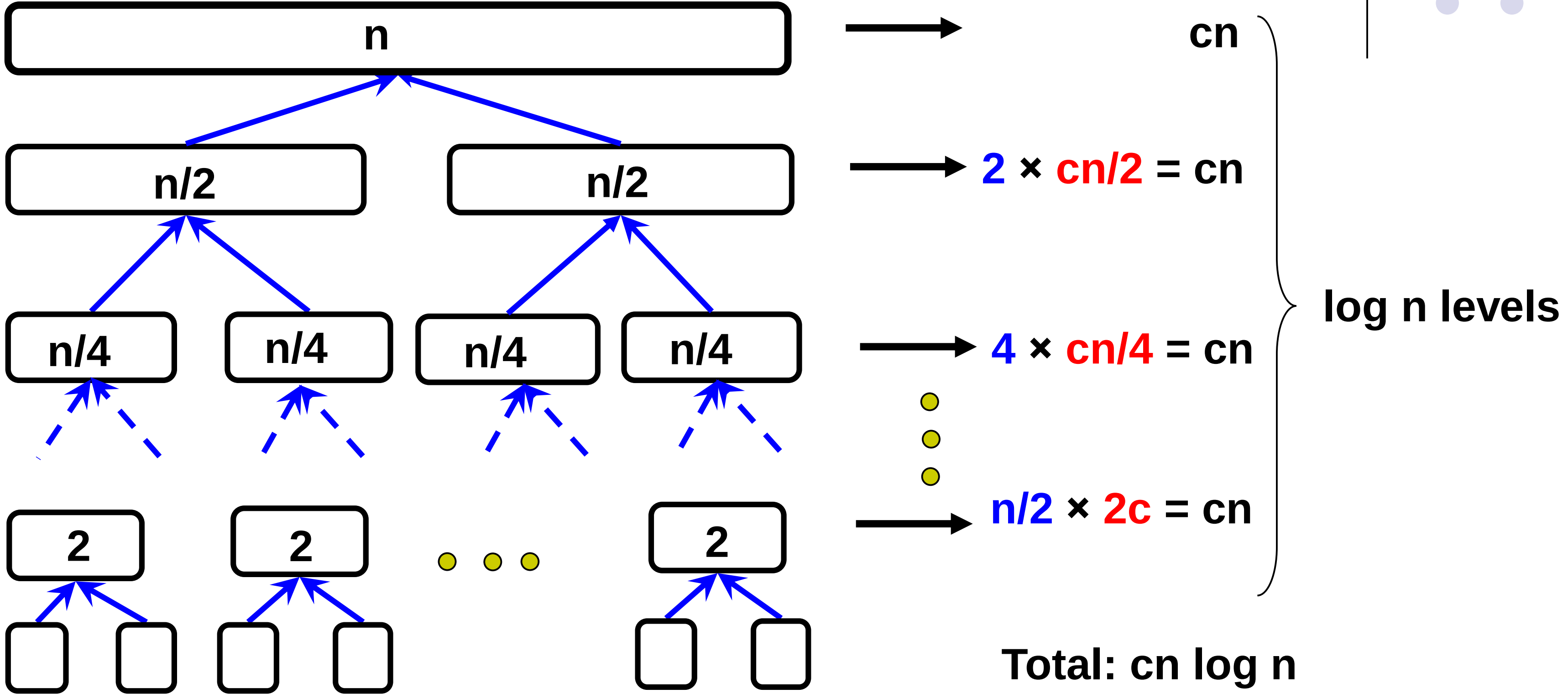


Merge-Sort(A, 0, 7)

Merge-Sort(A, 0, 7), done!



Merge-Sort Analysis



- Total running time: $\Theta(n\log n)$
- Total Space: $\Theta(n)$

Merge-Sort Summary

Approach: divide and conquer

Time

- Most of the work is in the merging
- Total time: Θ(n log n)

Space:

- Θ(n), more space than other sorts.