Advanced Algorithms & Data Structures

Department of CSE

ADVANCED ALGORITHMS AND DATA STRUCTURES
23CS03HF

Topic:

Divide and Conquer, Merge Sort

AIM OF THE SESSION

To familiarize students with the concept of Divide and Conquer and Merge sort

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1.Demonstrate :- Divide and conquer and merge sort.

Describe :- Recurrence relation and time complexity of merge sort

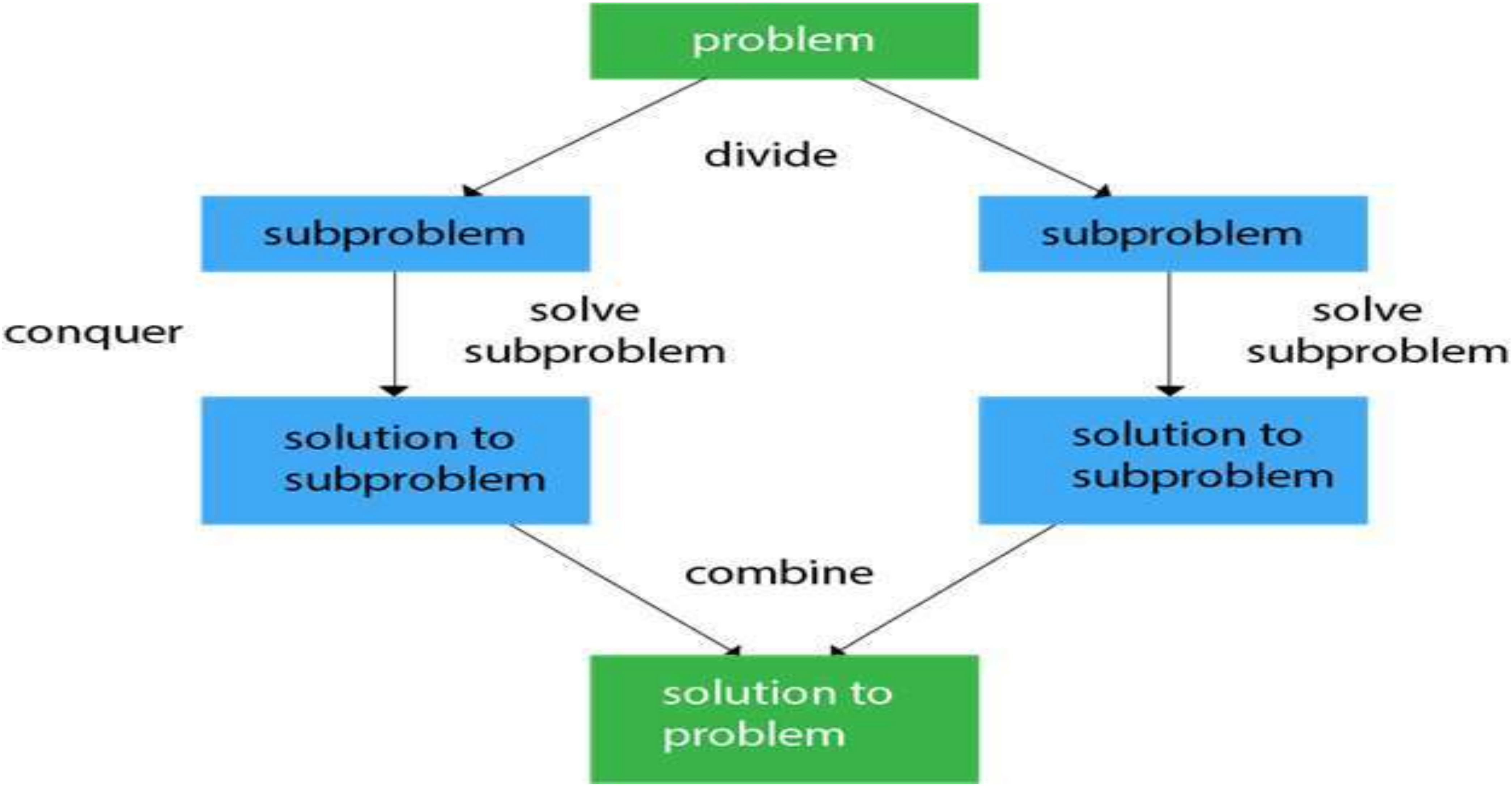
LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define :- Merge sort .
2. Describe :- Recurrence relation and time complexity of merge sort
3. Summarize:- It gives the description about the merge sort and time complexity of merge sort

Divide and Conquer

- It is a top-down approach. The algorithms which follow the divide & conquer techniques involve three steps:
- i. Divide the original problem into a set of subproblems.
- ii. Solve every subproblem individually, recursively.
- iii.Combine the solution of the subproblems (top level) into a solution of the whole original problem.


problem
divide
subproblem
subproblem
conquer
solve subproblem
solve subproblem
solution to subproblem
solution to subproblem
combine
solution to problem

- Broadly, we can understand approach in a three-step process.

  Divide/ Break

  This step involves breaking the problem into smaller sub-problems. Sub-problems should represent a part of the original problem.
- This step generally takes a recursive approach to divide the problem until no sub-problem is further divisible.

  Conquer/Solve

  This step receives a lot of smaller sub-problems to be solved..

  Merge/Combine

  When the smaller sub-problems are solved, this stage recursively combines them until they formulate a solution of the original problem.
- This algorithmic approach works recursively and conquer & merge steps works so close that they appear as one

DIVIDE AND CONQUER ALGORITHM

Algorithm D and C(P)
{
if small(P)
then return S(P)
else
{ divide P into smaller instances P1 ,P2 .....Pk
Apply D and C to each sub problem
Return combine (D and C(P1)+ D and C(P2)+.......+D and C(Pk))}
}

Examples

1) Binary Search
2) Quicksort
3) Merge Sort
4) Strassen’s Algorithm
8) Calculating power
9) Factorial problem
10) Towers of Hanoi
11) Tree traversals :
12) Fibonacci number
13) Finding min,max etc

Merge Sort

- Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list.
- Idea:
  - Divide the unsorted list into N sub lists, each containing 1 element.
  - Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. will

  now convert into N/2 lists of size 2.
  - Repeat the process till a single sorted list of obtained.

Suppose we had to sort an array A. A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

Divide

If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays :A[p..q] and A[q+1, r]

Conquer

In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them

Combine

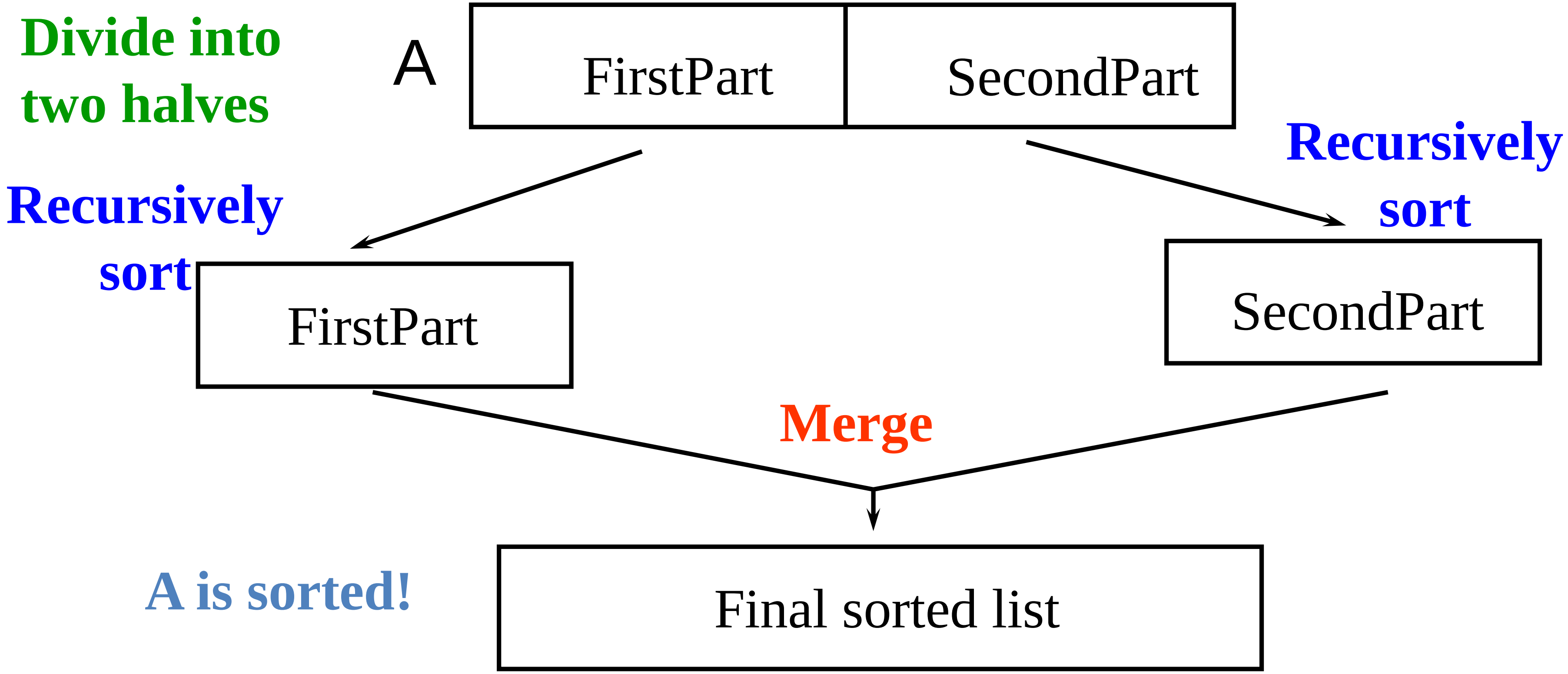
When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays
A[p..q] and A[q+1, r]

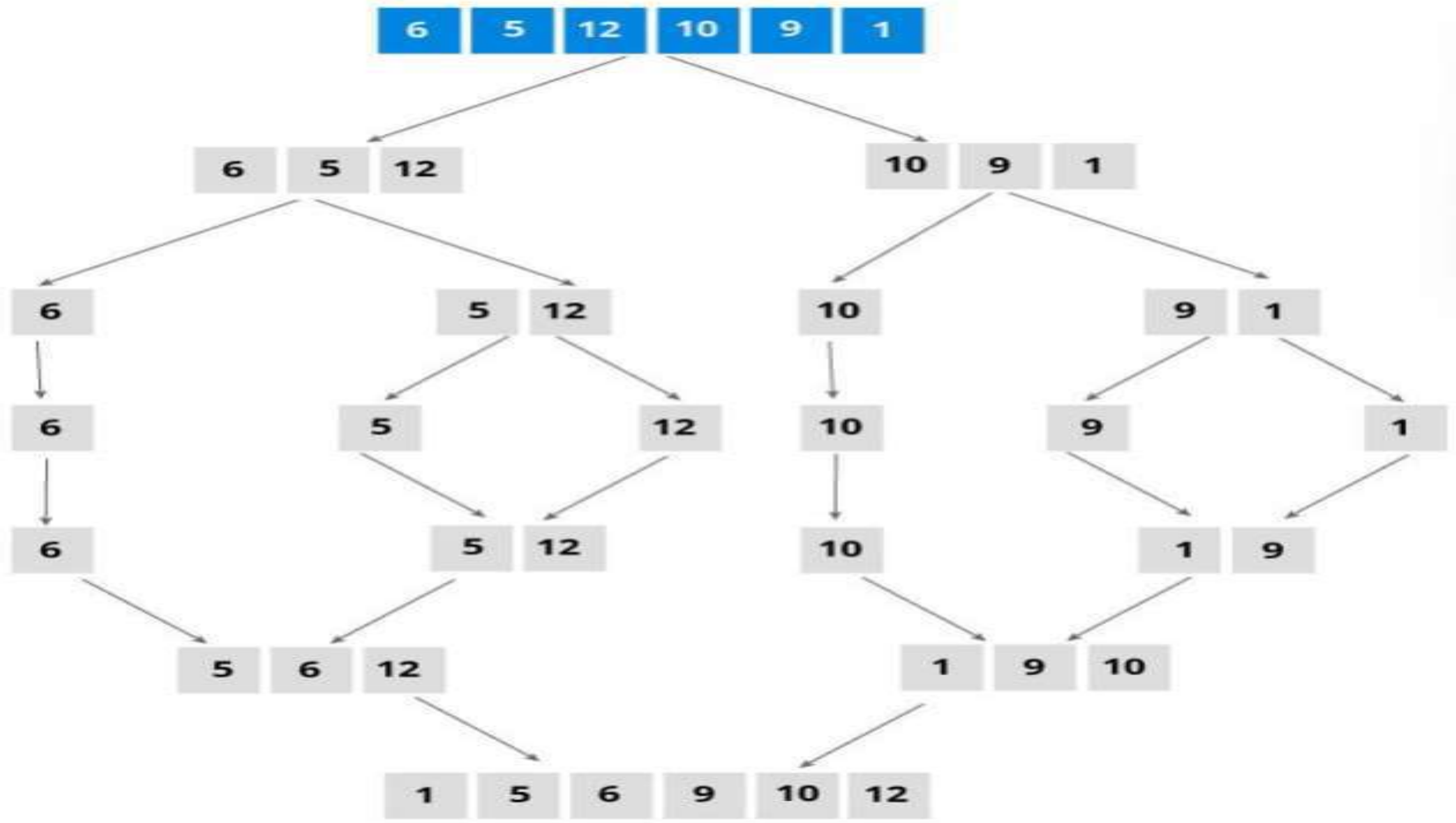
While comparing two sublists for merging, the first element of both lists is taken into consideration.

While sorting in ascending order, the element that is of a lesser value becomes a new element of the sorted list. This procedure is repeated until both the smaller sublists are empty and the new combined sublist comprises all the elements of both the sublists

MERGE SORT:

IDEA




6 5 12 10 9 1
6 5 12
10 9 1
6
5 12
10
9 1
6
5
12
10
9
1
6
5 12
10
1 9
5 6 12
1 9 10
1 5 6 9 10 12

Merge Sort: Algorithm

Algorithm MergeSort (low, high)

// a[low: high] is a global array to be sorted.

// Small(P) is true if there is only one element to sort. In this case the list is already

sorted.

{   if   (low<high) then // if there are more than one element

Recursive Calls

{

// Divide P into sub problems.

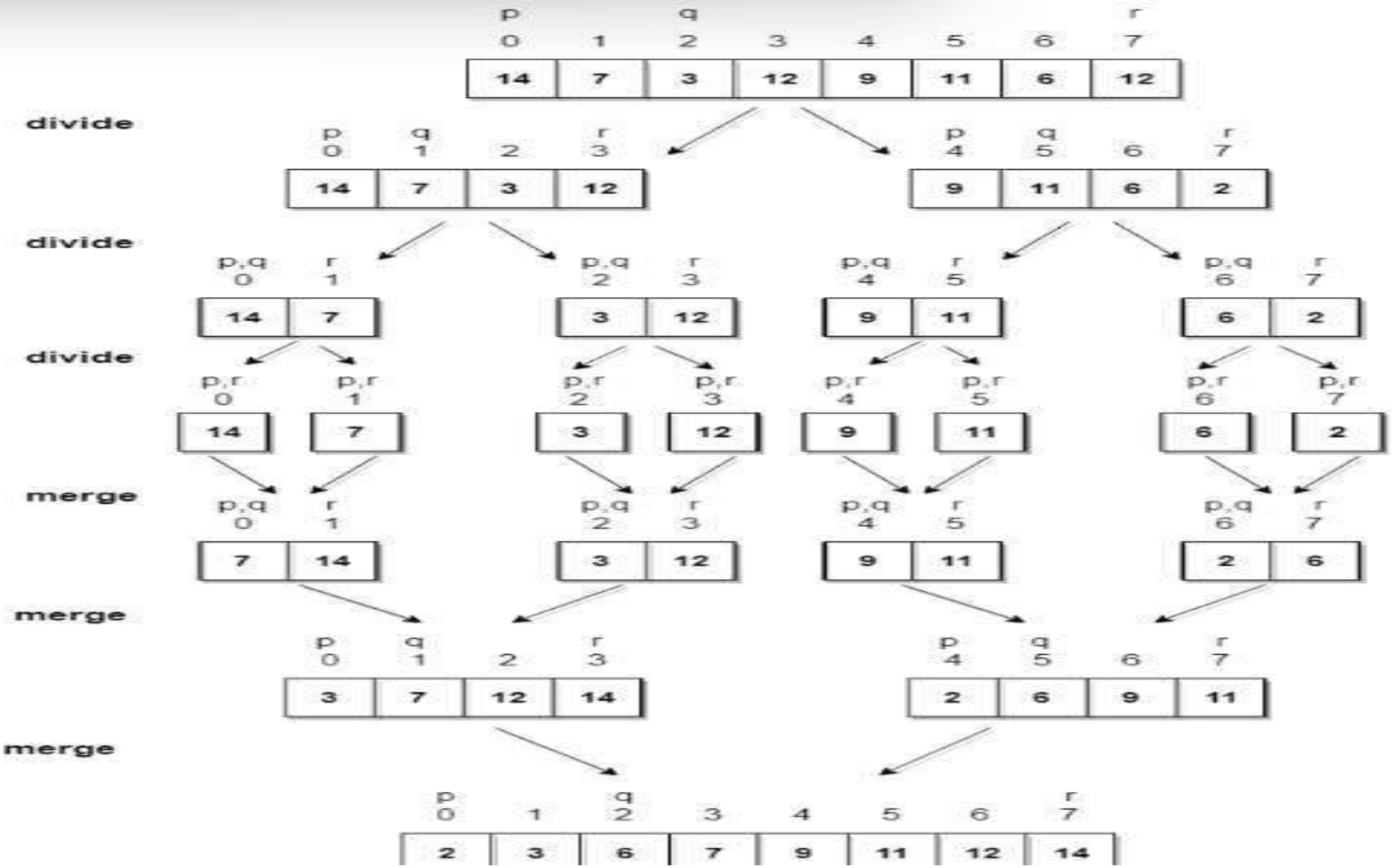
// Find where to split the set.

mid := [(low+high)/2];

```
//solve the sub problems.
MergeSort(low,mid);
MergeSort(mid+1, high);
// Combine the solutions.
Merge(low, mid, high);
}
}
```

- The merge function works as follows:

  1. Create copies of the subarrays L ← A[p..q] and R ← A[q+1..r].
  2. Create three pointers i,j and k
     1. i maintains current index of L, starting at 1
     2. j maintains current index of R, starting at 1
     3. k maintains current index of A[p..q], starting at p
  3. Until we reach the end of either L or R, pick the larger among the elements from L and R and place them in the correct position at A[p..q]
  4. When we run out of elements in either L or R, pick up the remaining elements and put in A[p..q]


p q r
0 1 2 3 4 5 6 7
14 7 3 12 9 11 6 12
divide
p q r
0 1 2 3
14 7 3 12
p q r
4 5 6 7
9 11 6 2
divide
p,q r
0 1
14 7
p,q r
2 3
3 12
p,q r
4 5
9 11
p,q r
6 7
6 2
divide
p,r 0
14
p,r 1
7
p,r 2
3
p,r 3
12
p,r 4
9
p,r 5
11
p,r 6
6
p,r 7
2
merge
p,q r
0 1
7 14
p,q r
2 3
3 12
p,q r
4 5
9 11
p,q r
6 7
2 6
merge
p q r
0 1 2 3
3 7 12 14
p q r
4 5 6 7
2 6 9 11
merge
p q r
0 1 2 3 4 5 6 7
2 3 6 7 9 11 12 14

Merge-Sort(A, 0, 7)

Divide

A: 6 2 8 4 3 3 7 7 5 5 1 1

KL ACCREDITED BY NAAC WITH A++ GRADE

nirf 2022 NATIONAL INSTITUTIONAL RANKING FRAMEWORK RANKED 27 AMONG ALL UNIVERSITIES

CATEGORY 1 UNIVERSITY BY MHRD, Govt. of India

43 YEARS OF EDUCATIONAL LEADERSHIP

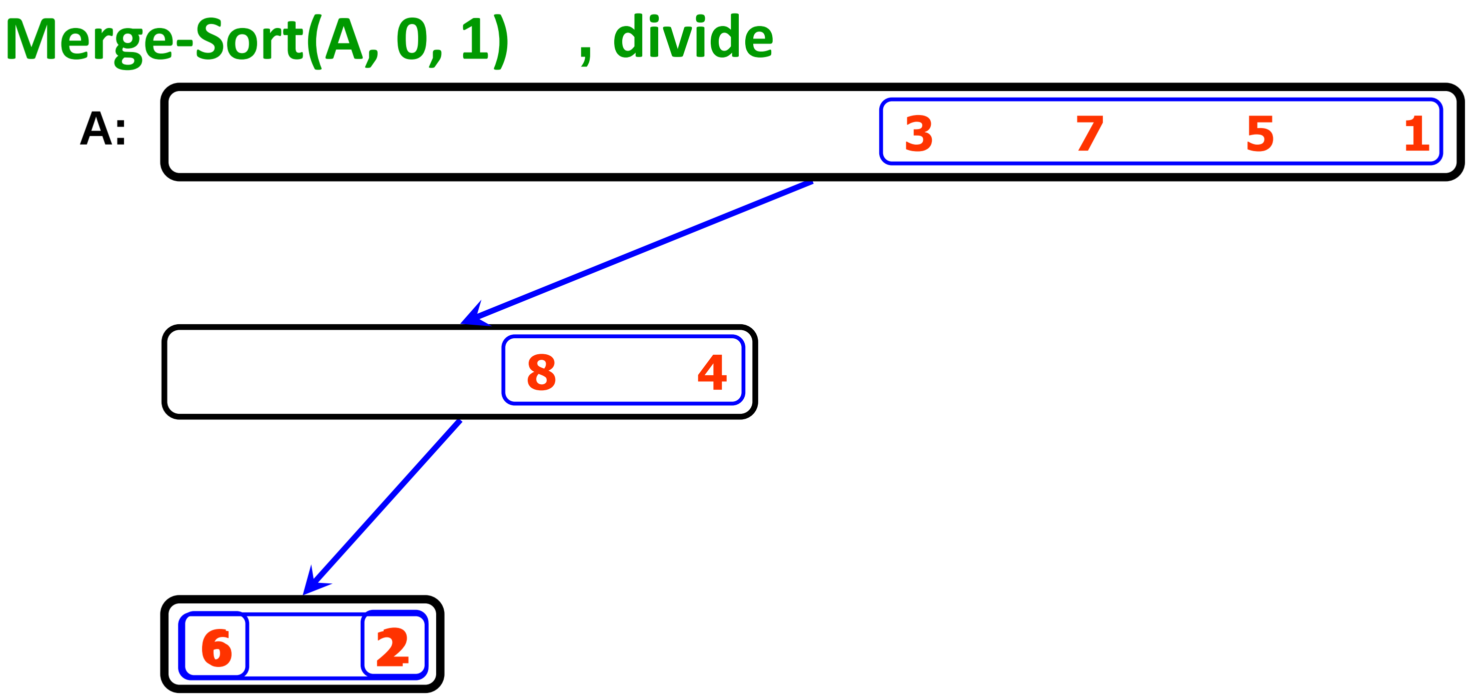
Merge-Sort(A, 0, 7)

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

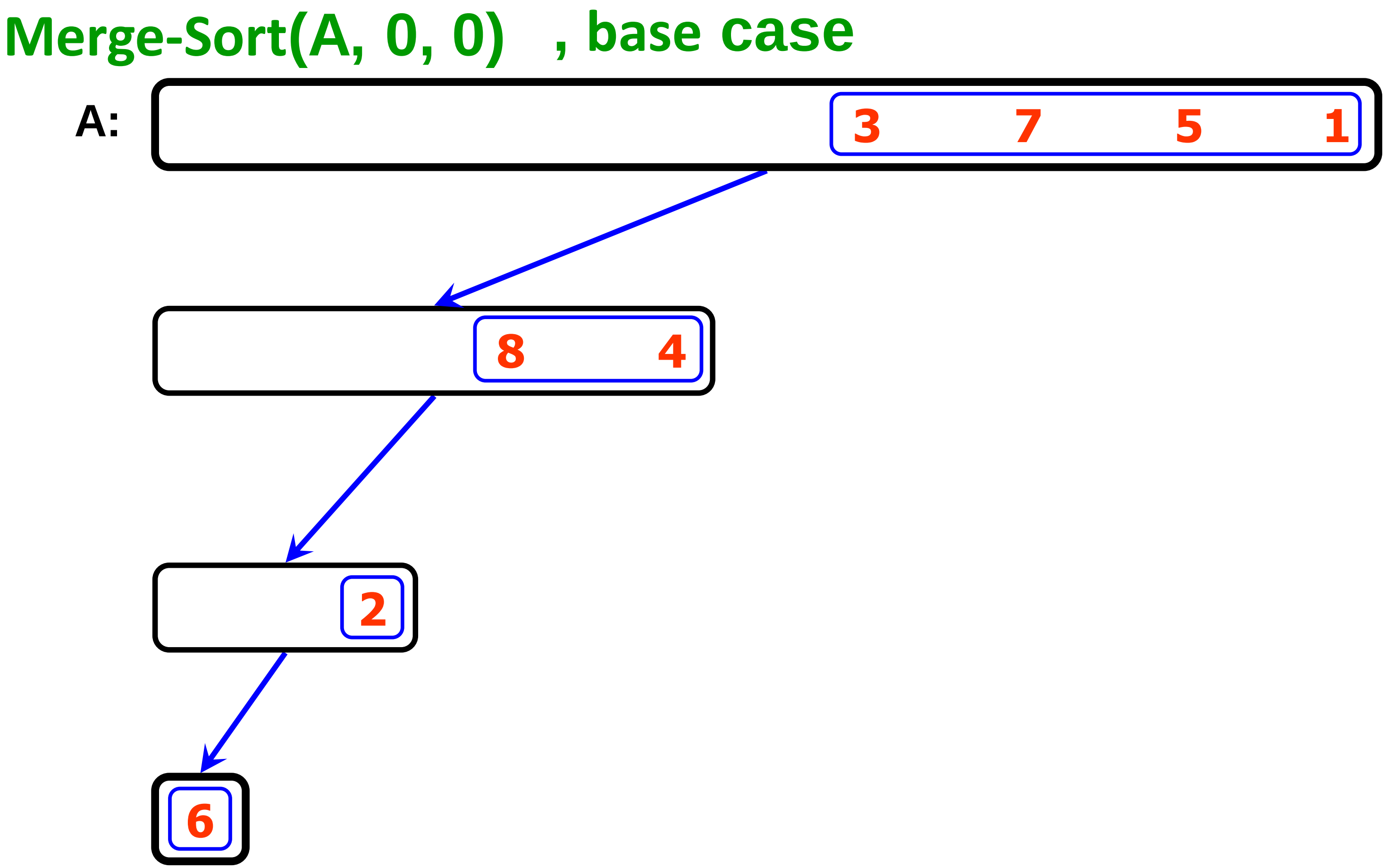
A: | | | | | 3 | 7 | 5 | 1

6 | 2 | 8 | 4

Merge-Sort(A, 0, 7)

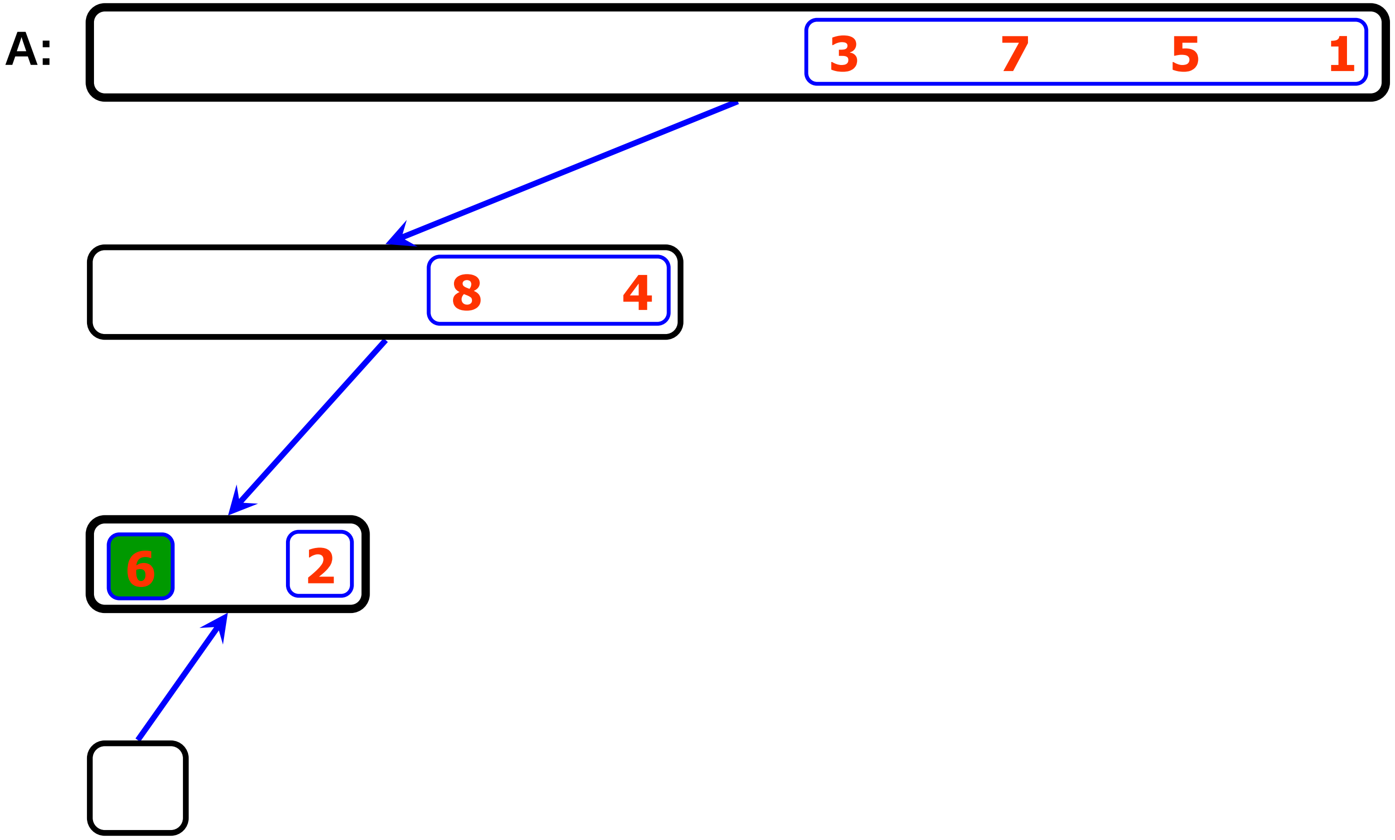


Merge-Sort(A, 0, 7)



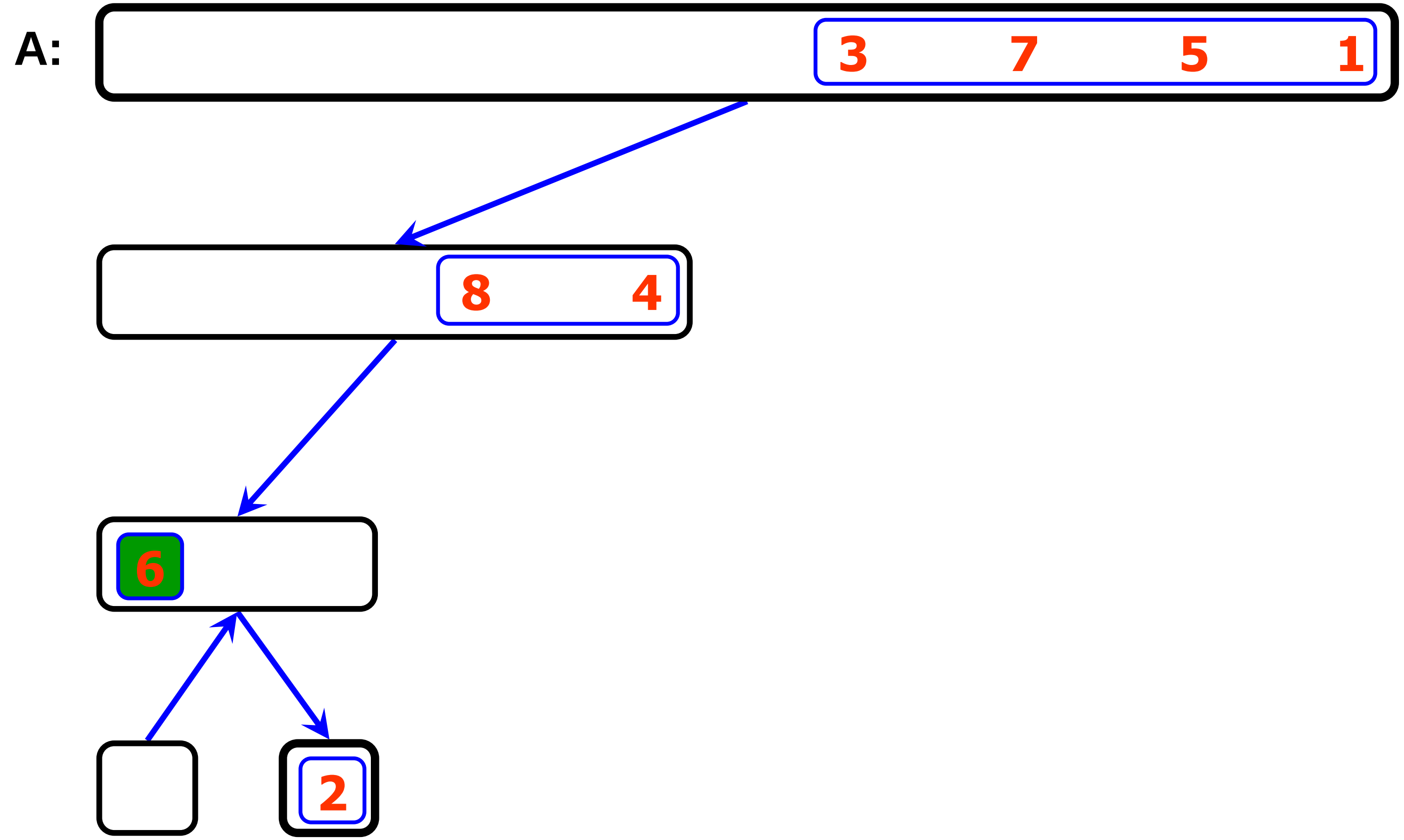
Merge-Sort(A, 0, 7)

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

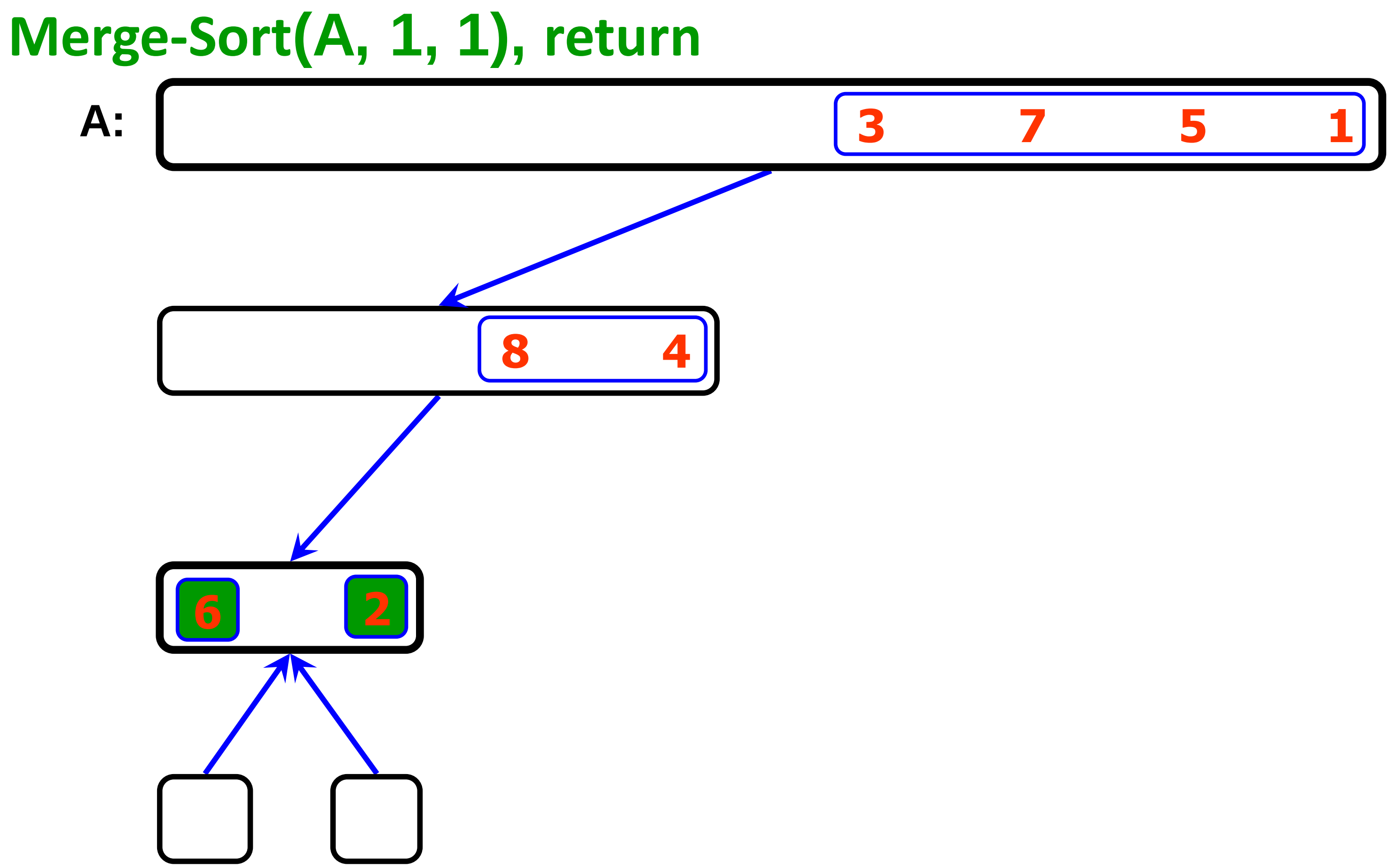


Merge-Sort(A, 0, 7)

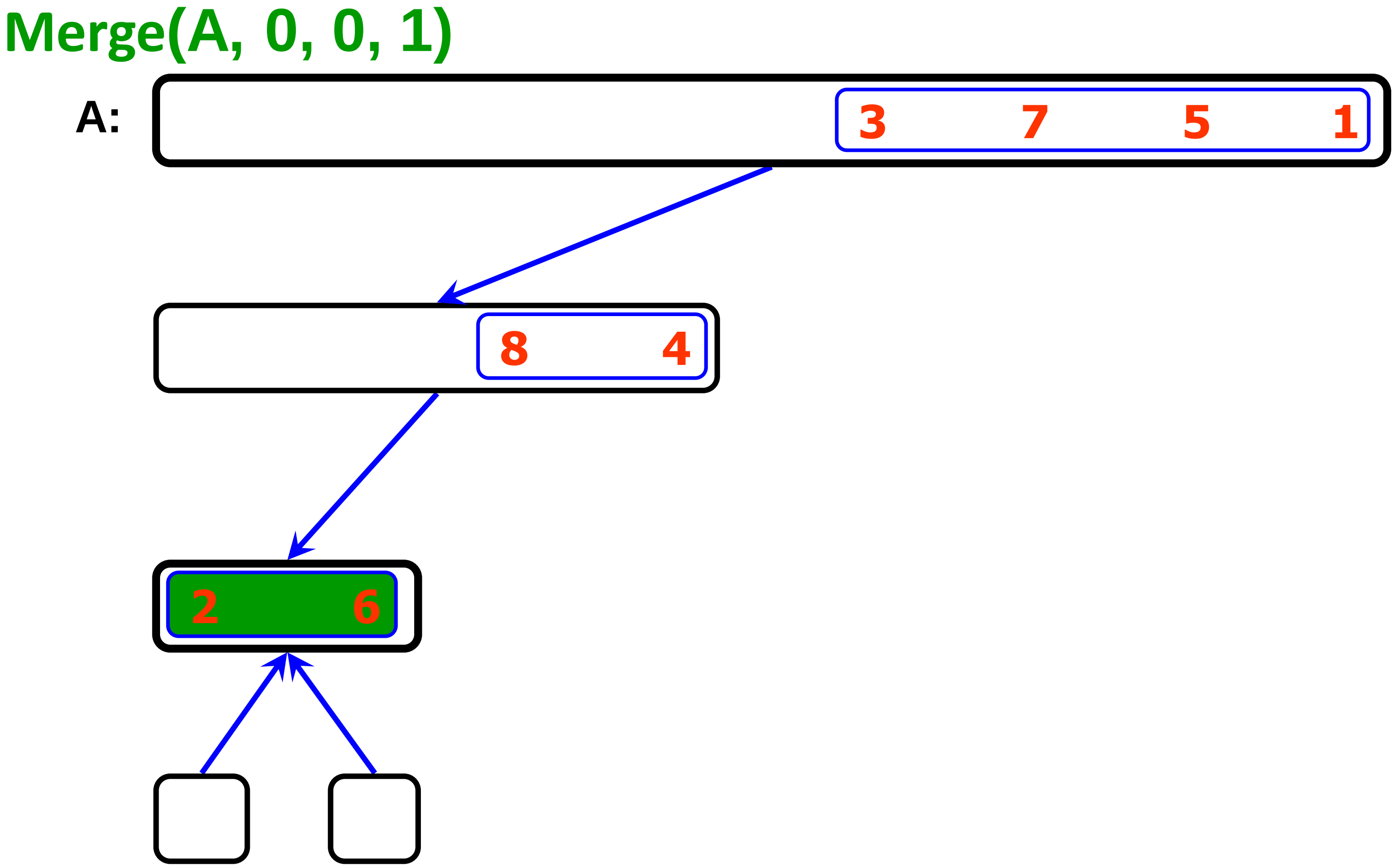
Merge-Sort(A, 1, 1) , base case



Merge-Sort(A, 0, 7)

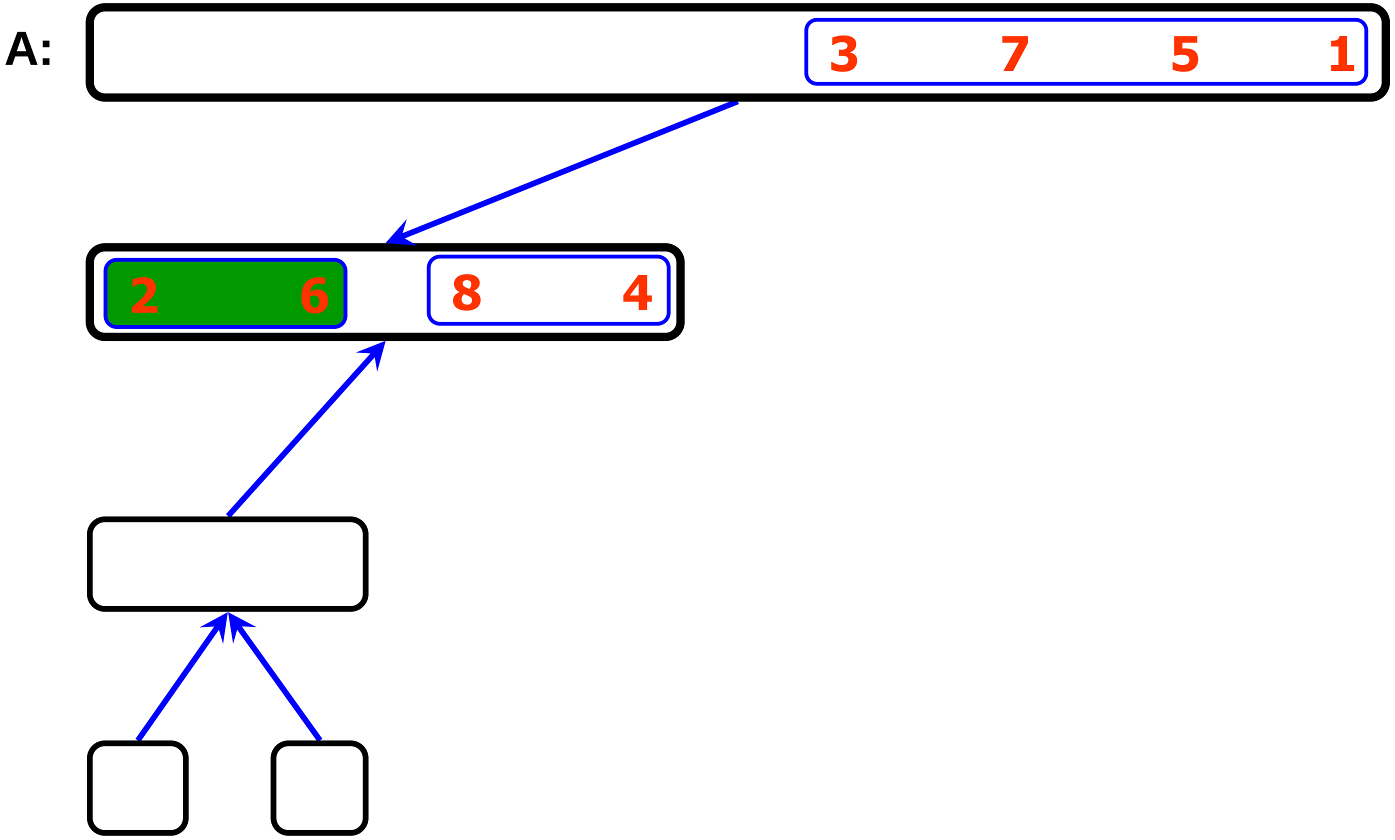


Merge-Sort(A, 0, 7)

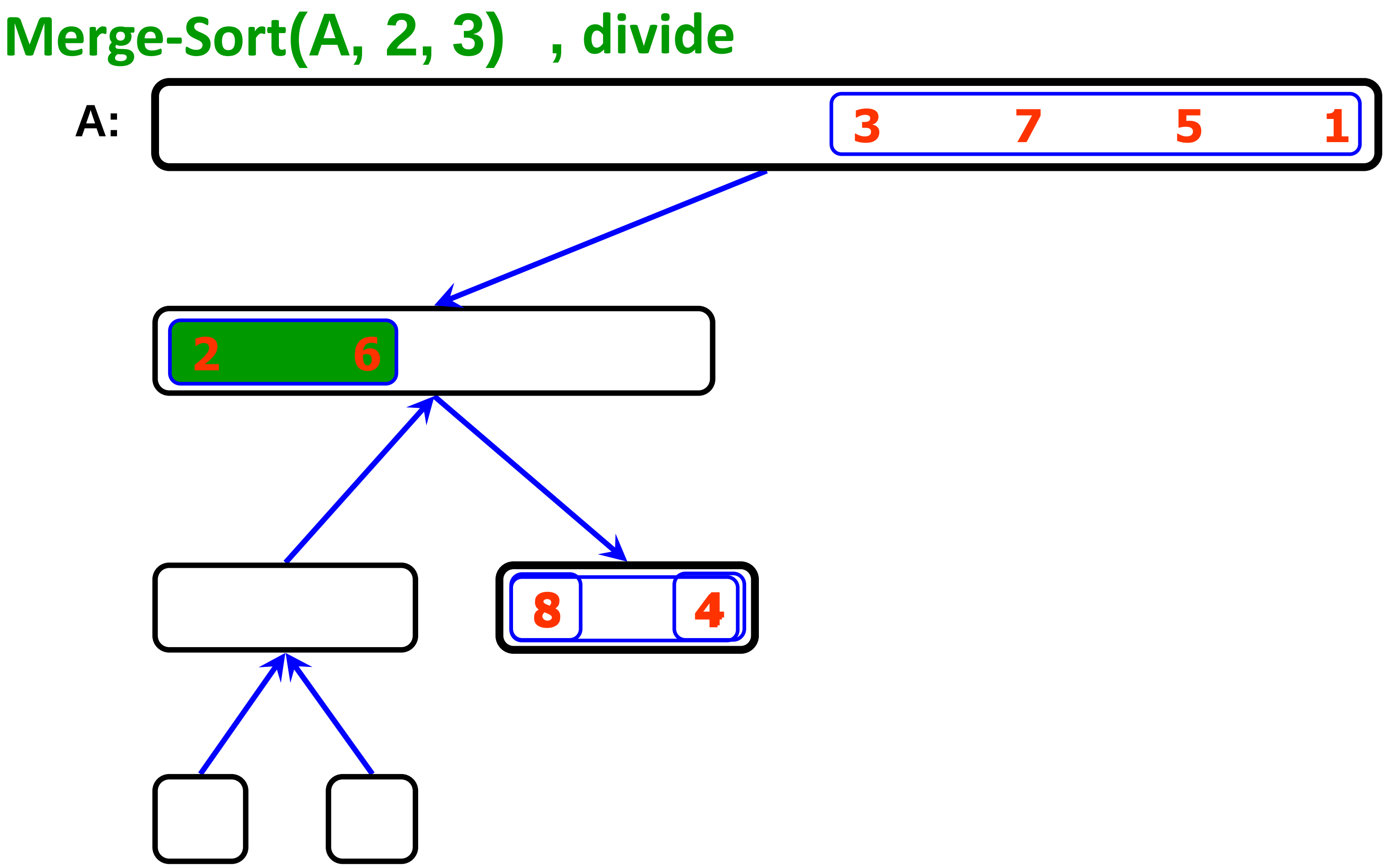


Merge-Sort(A, 0, 7)

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

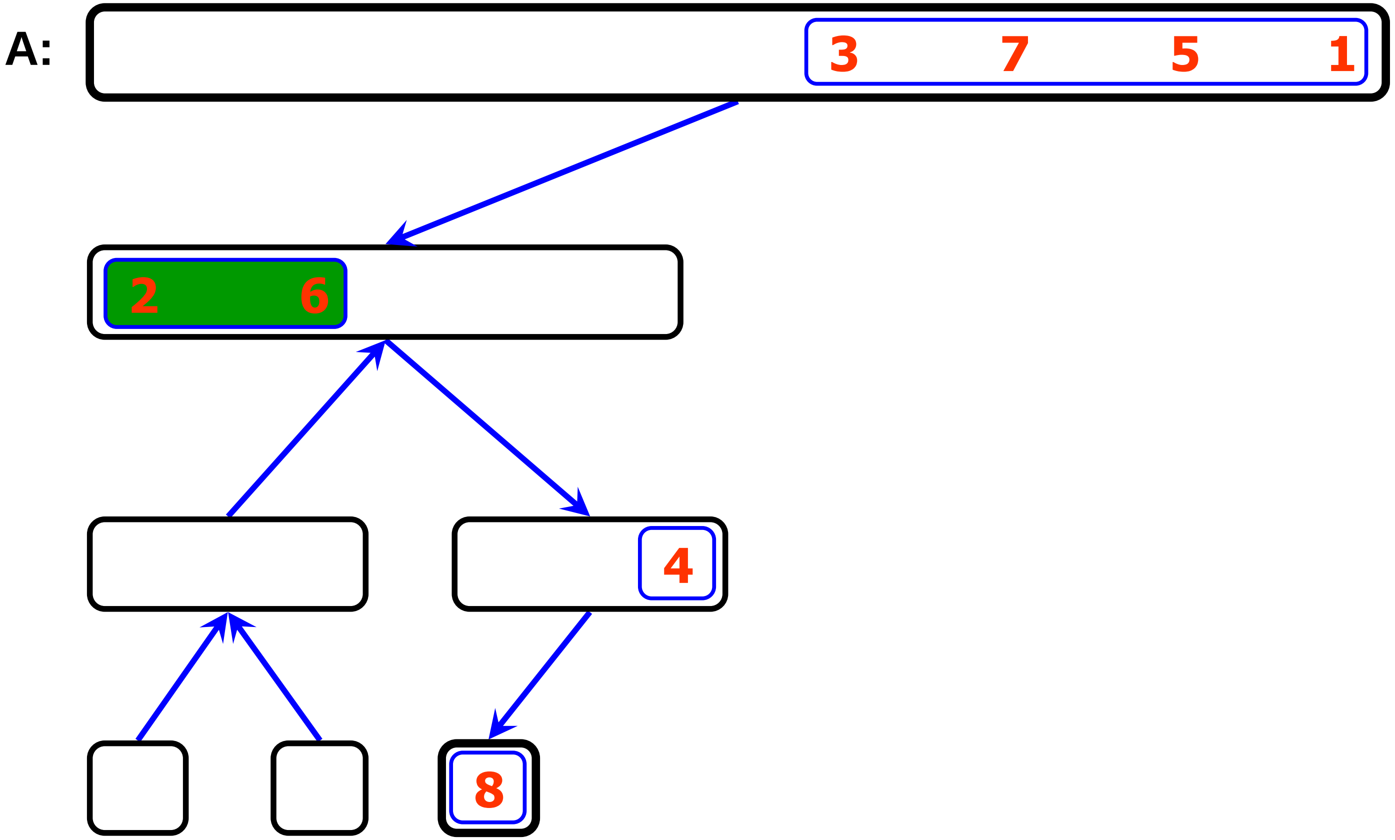


Merge-Sort(A, 0, 7)



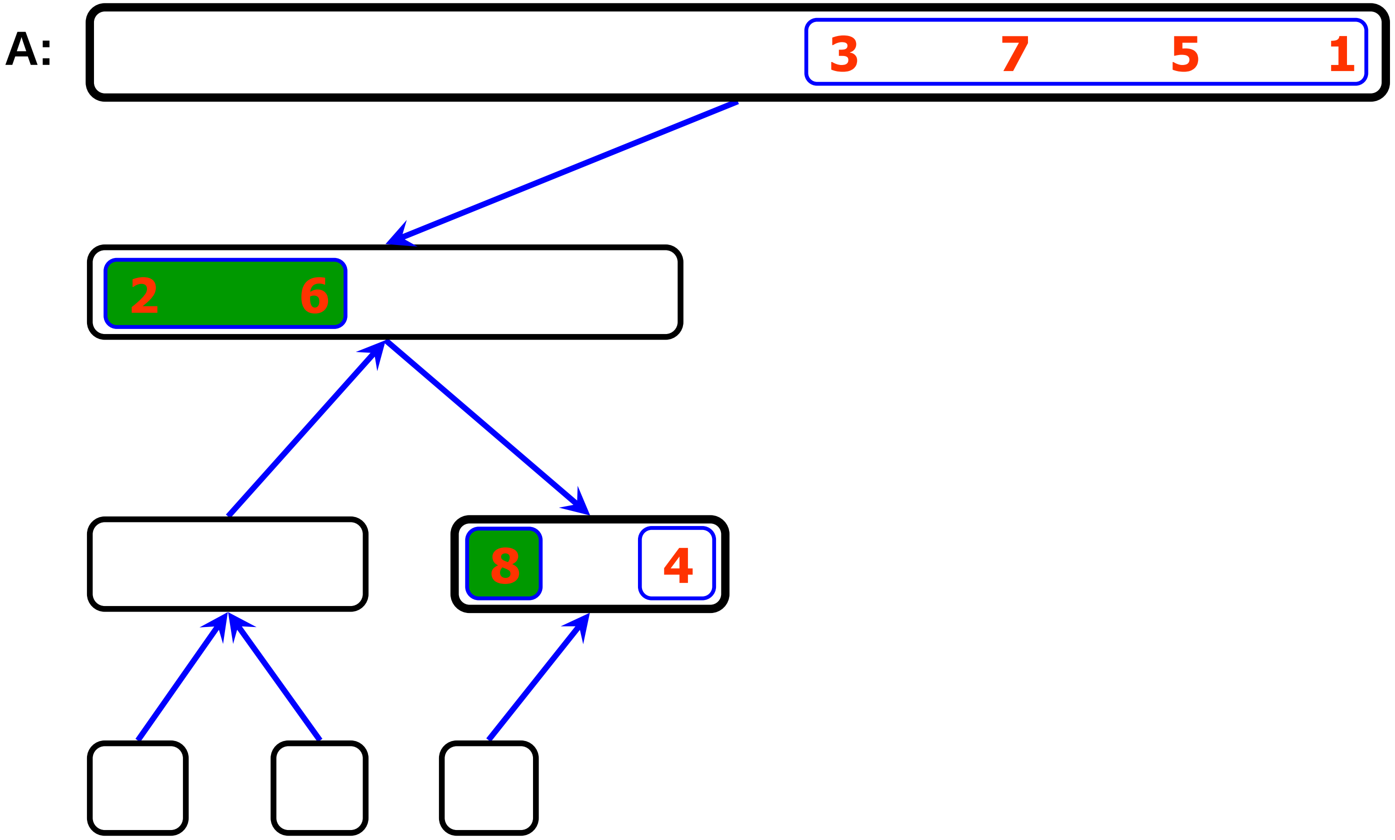
Merge-Sort(A, 0, 7)

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



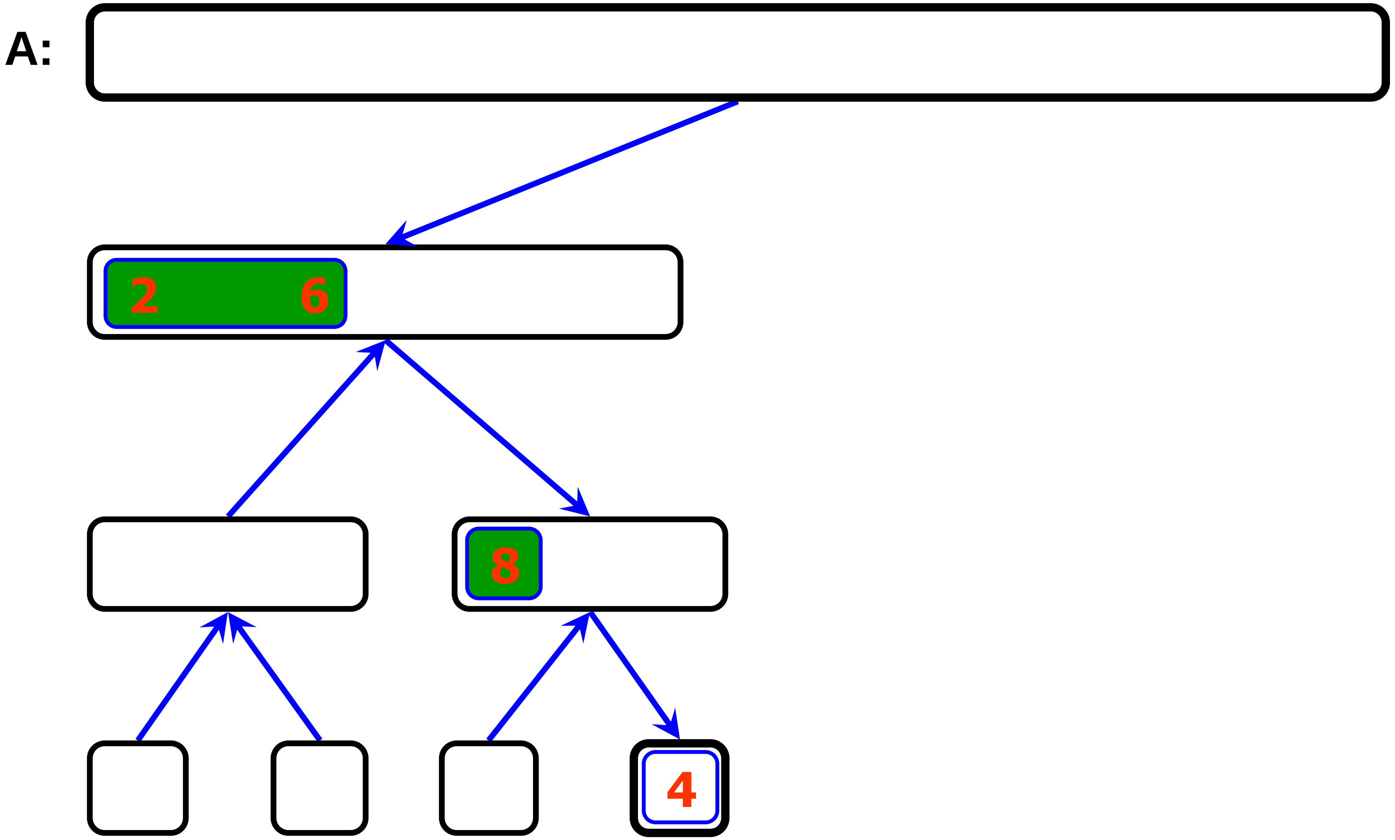
Merge-Sort(A, 0, 7)

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



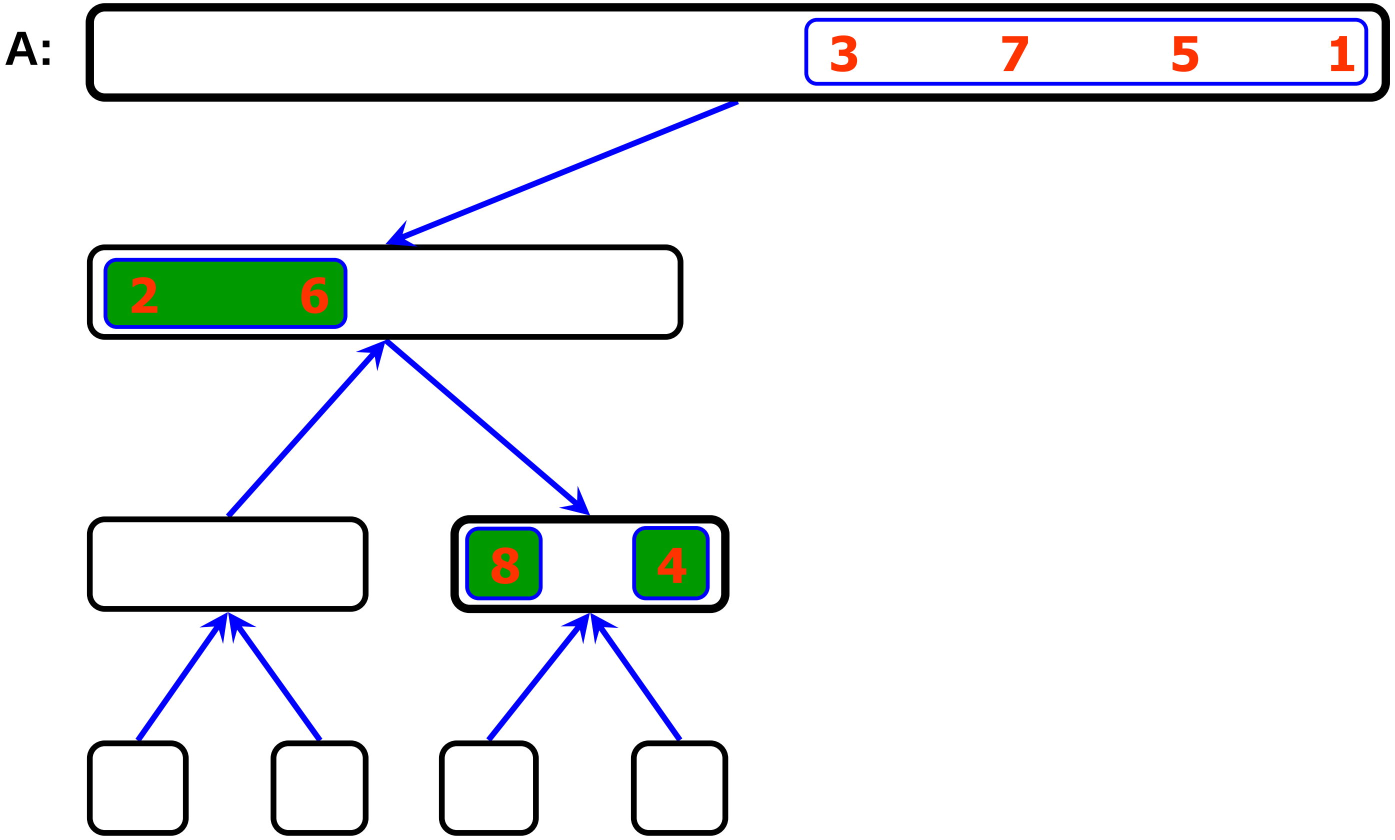
Merge-Sort(A, 0, 7)

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

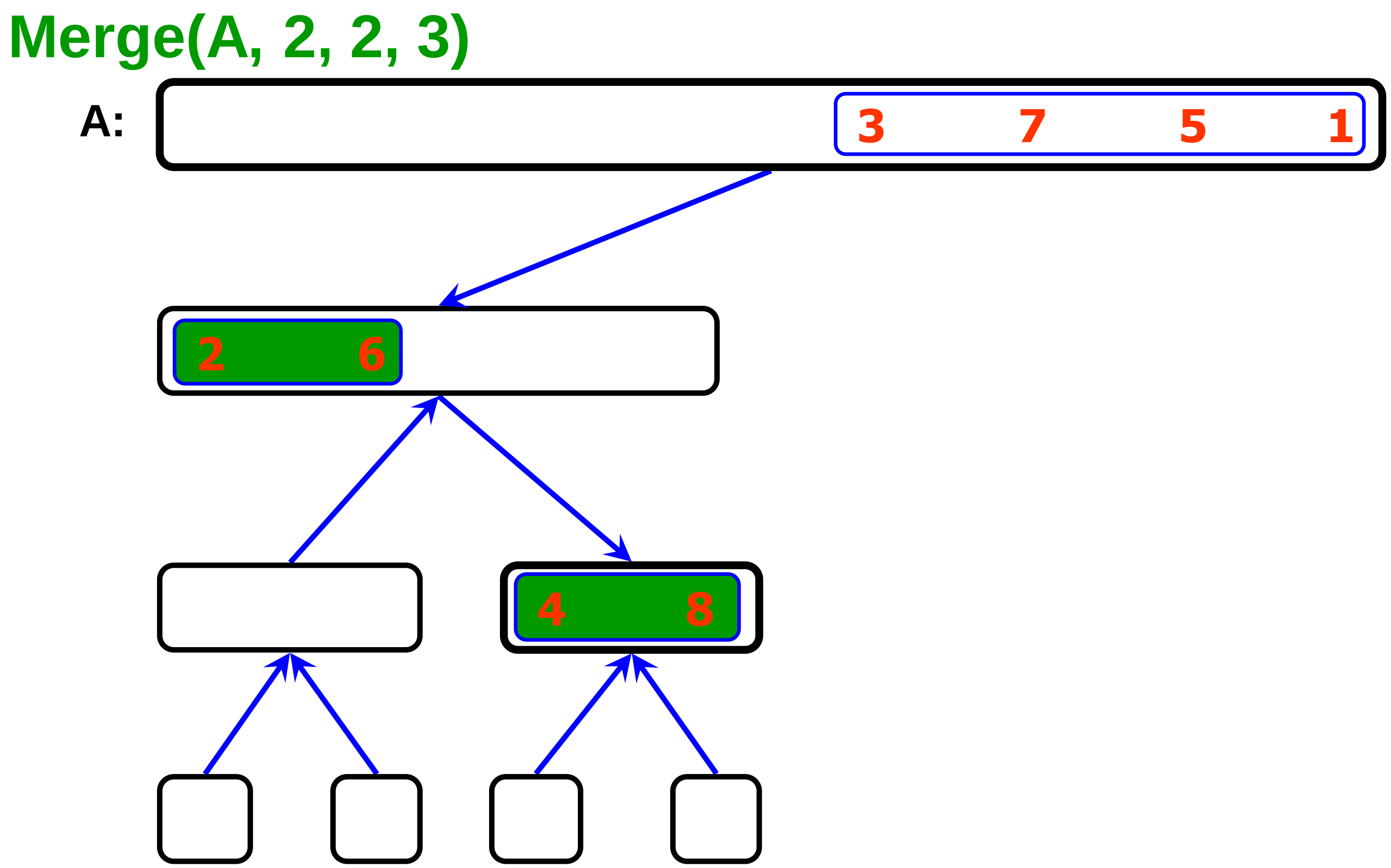


Merge-Sort(A, 0, 7)

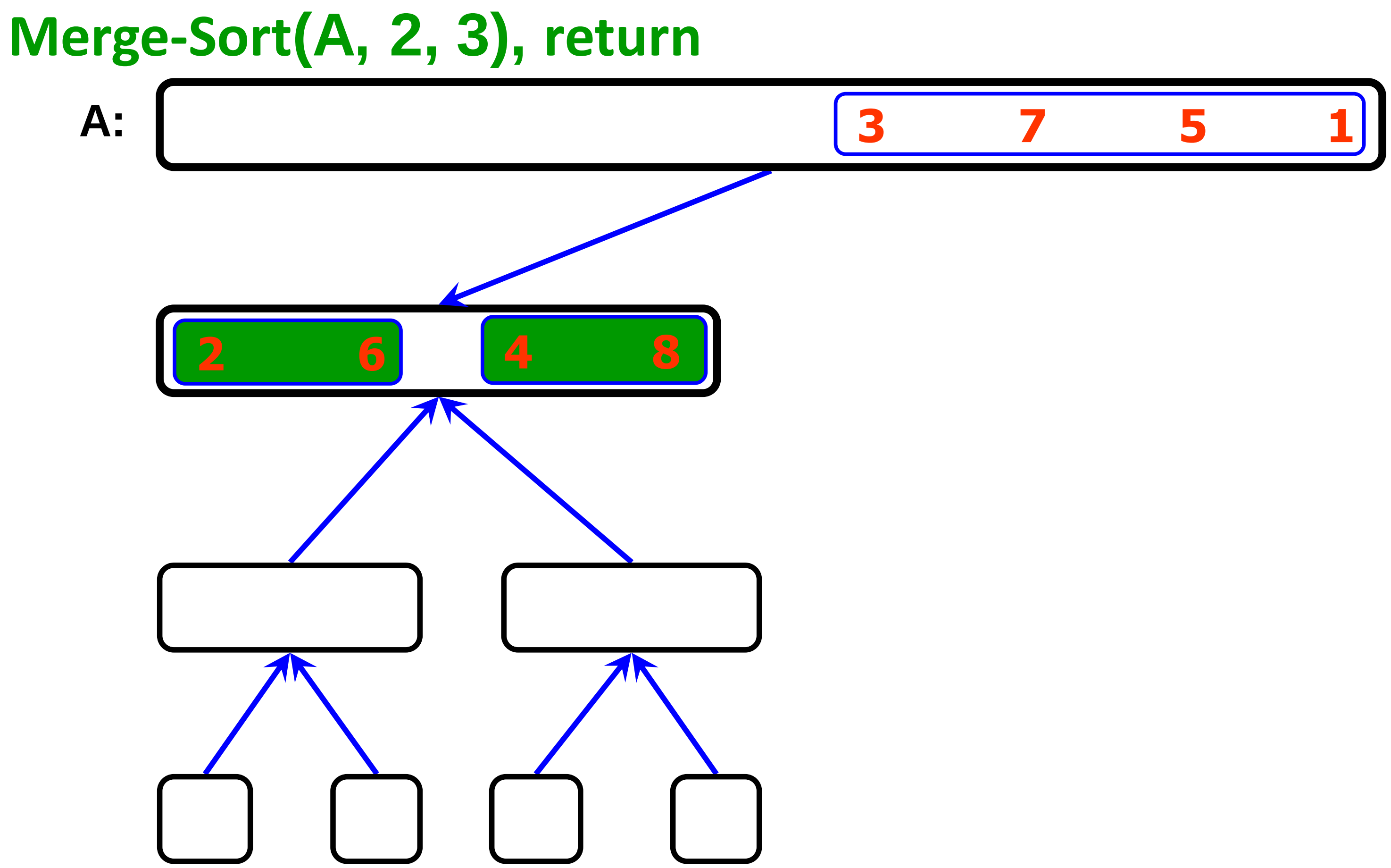
Merge-Sort(A, 3, 3), return



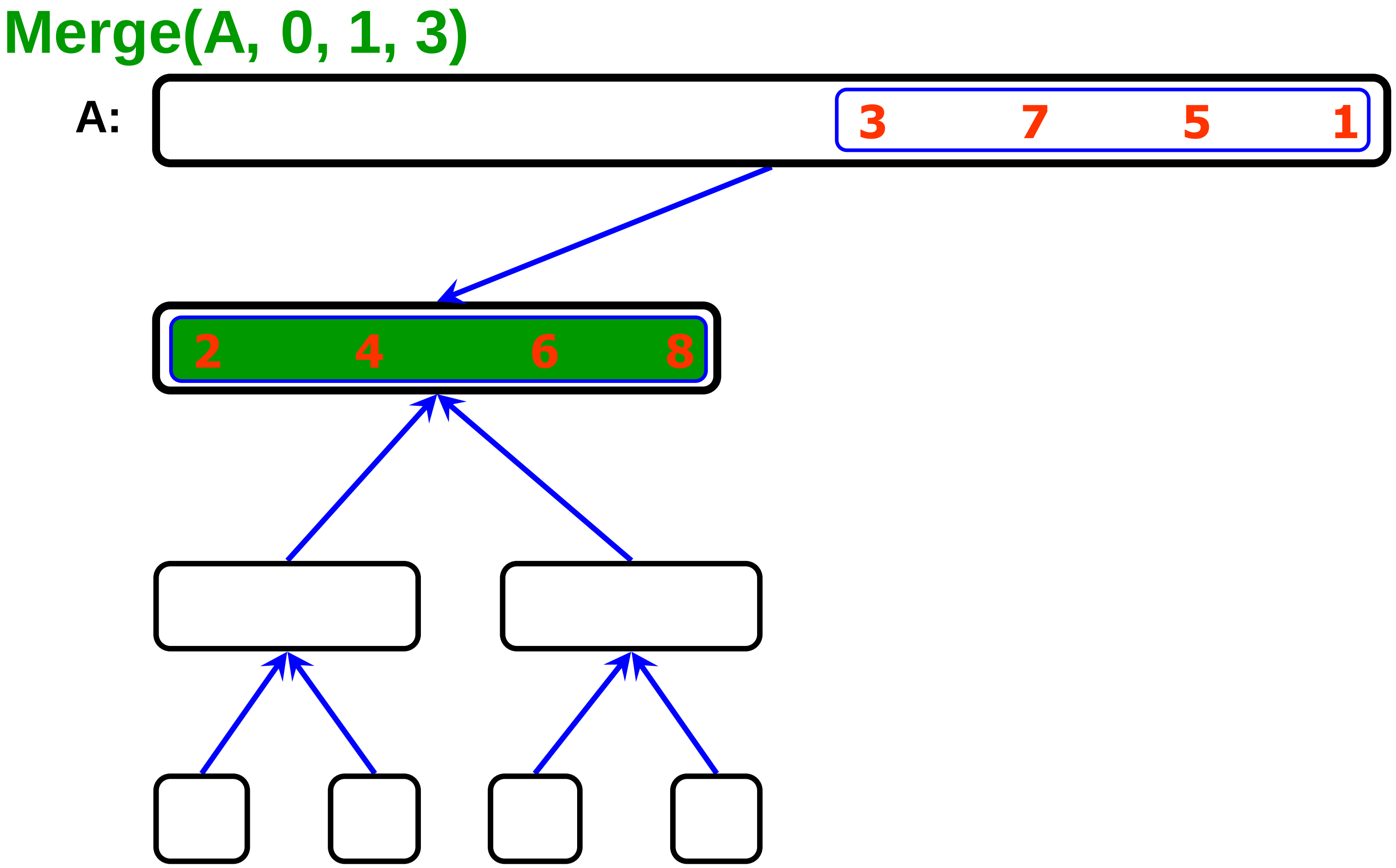
Merge-Sort(A, 0, 7)



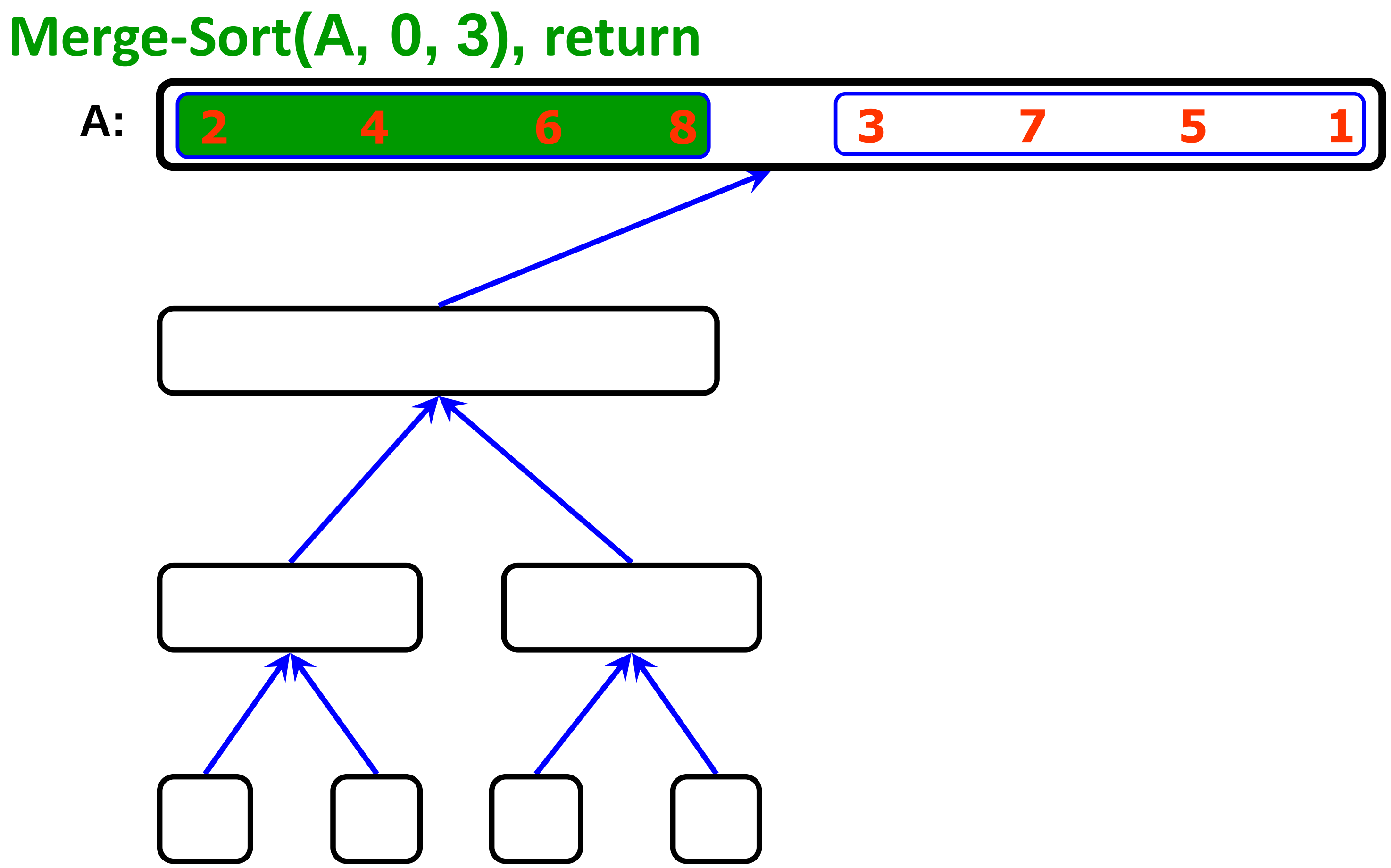
Merge-Sort(A, 0, 7)



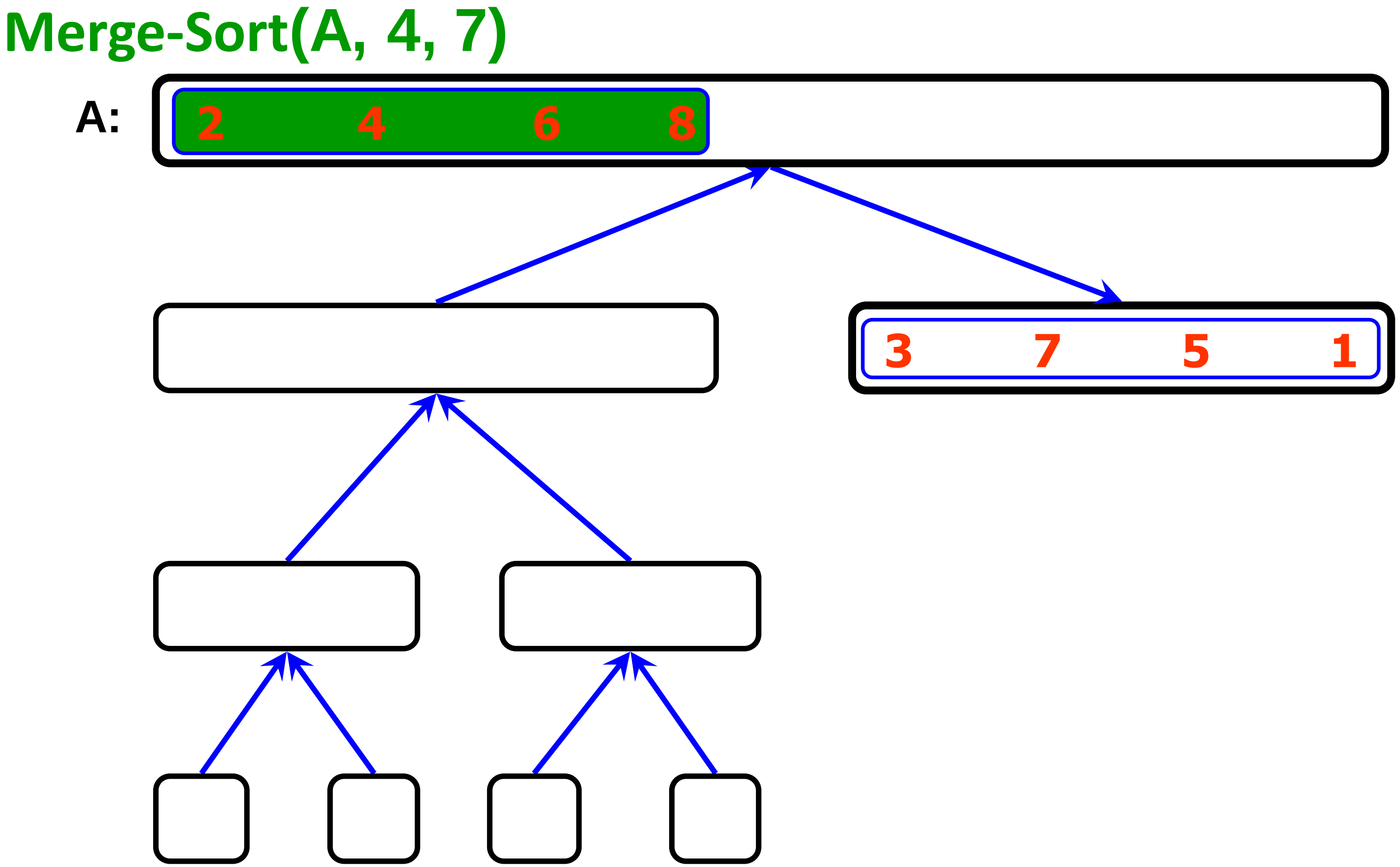
Merge-Sort(A, 0, 7)



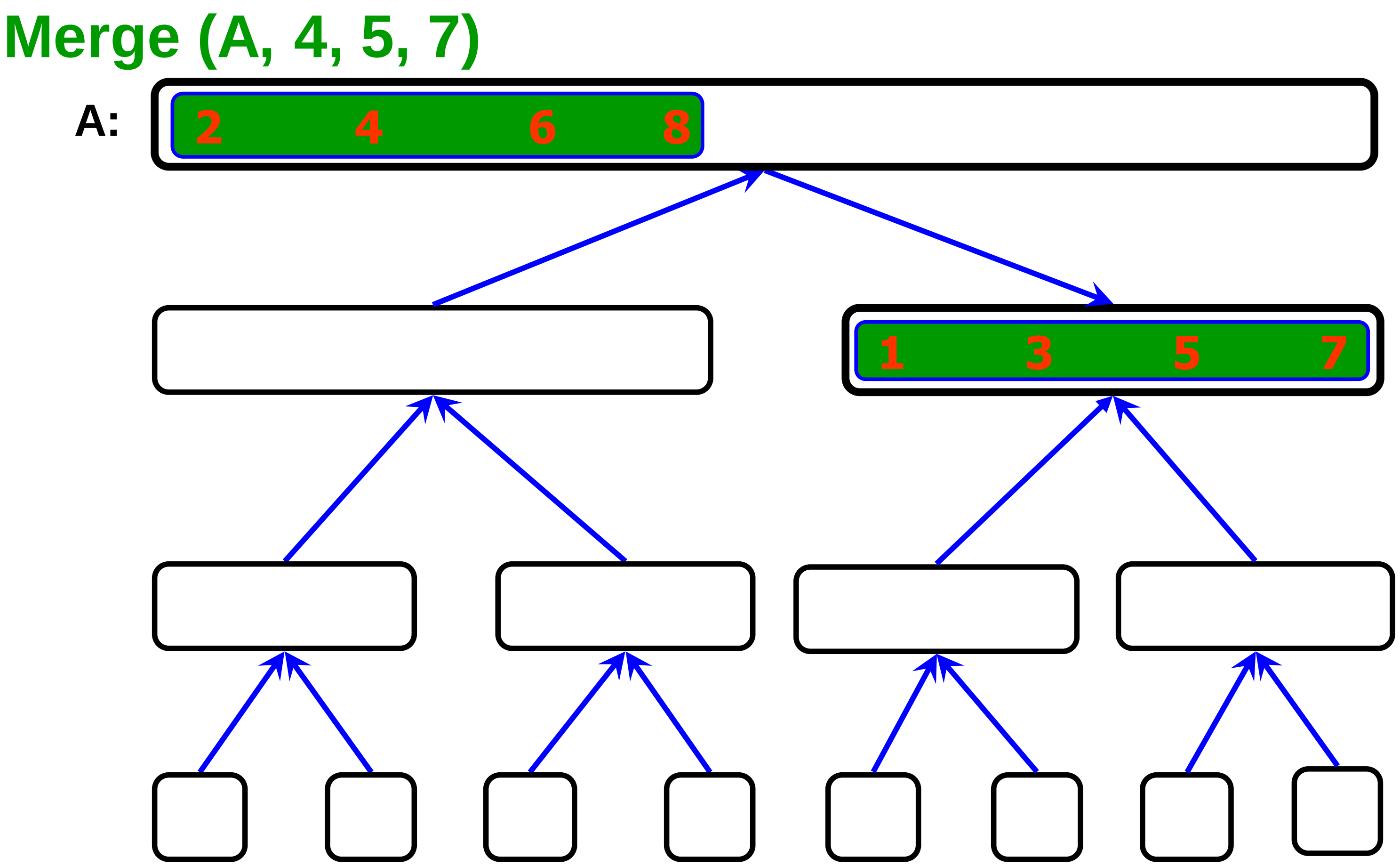
Merge-Sort(A, 0, 7)



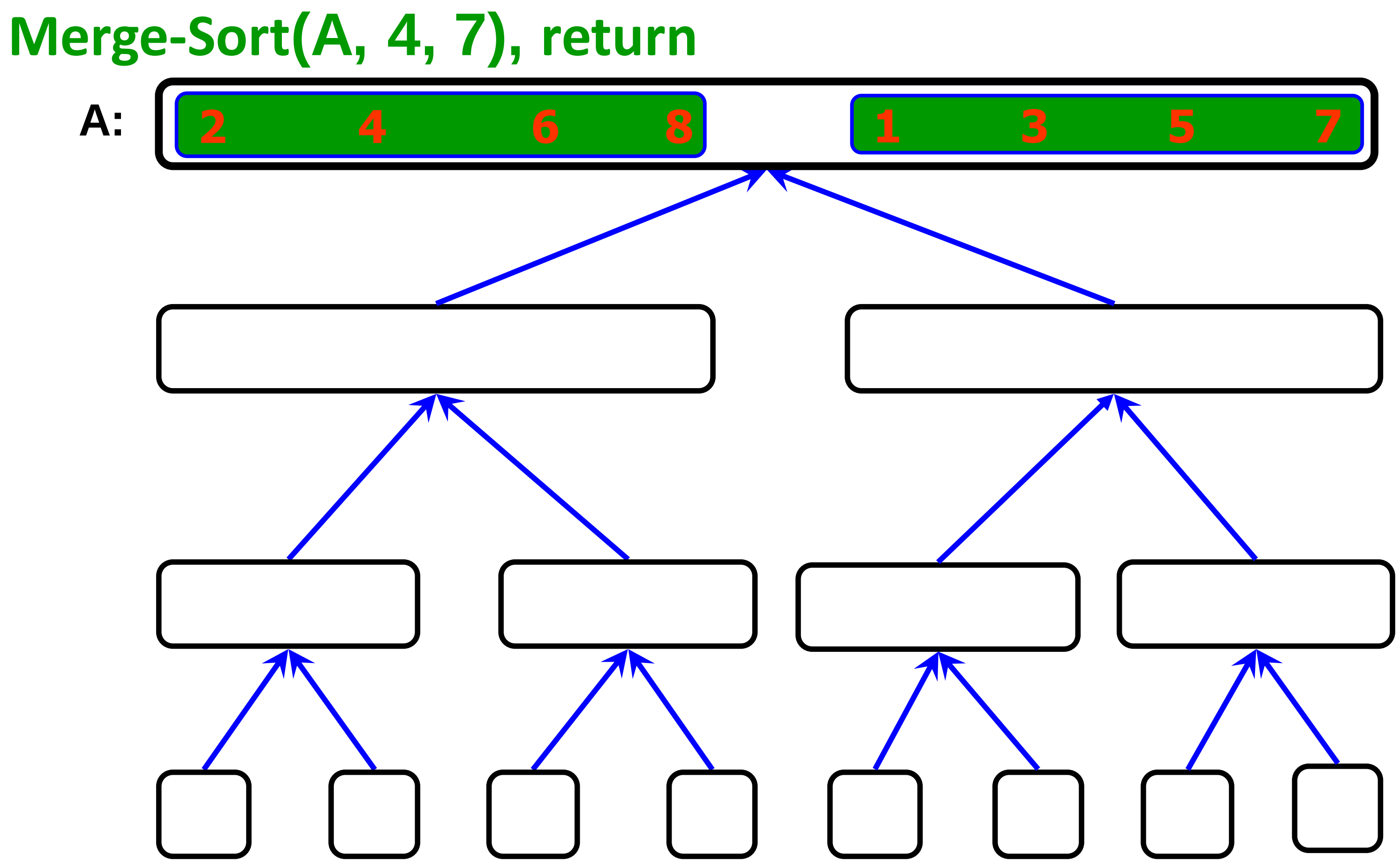
Merge-Sort(A, 0, 7)



Merge-Sort(A, 0, 7)

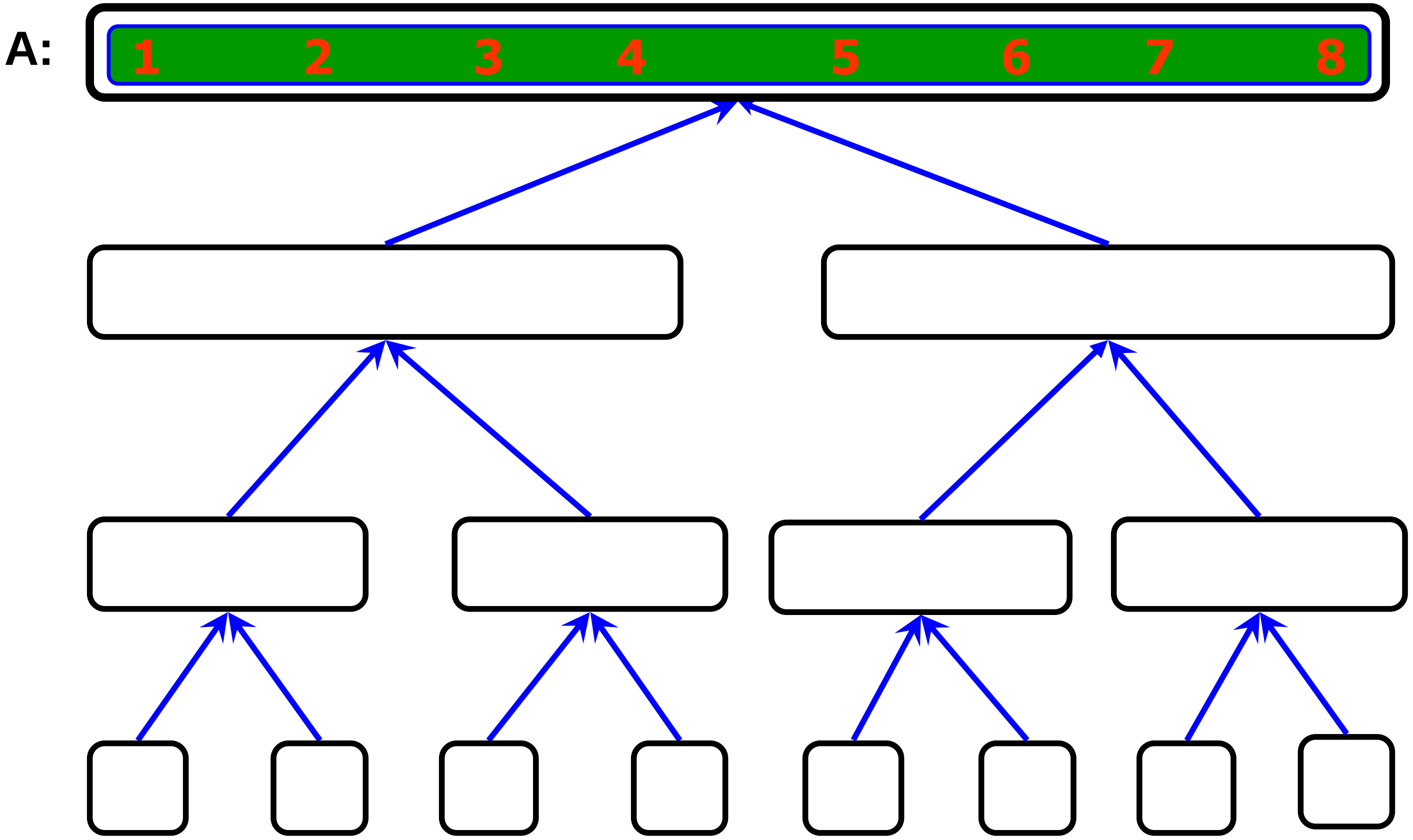


Merge-Sort(A, 0, 7)



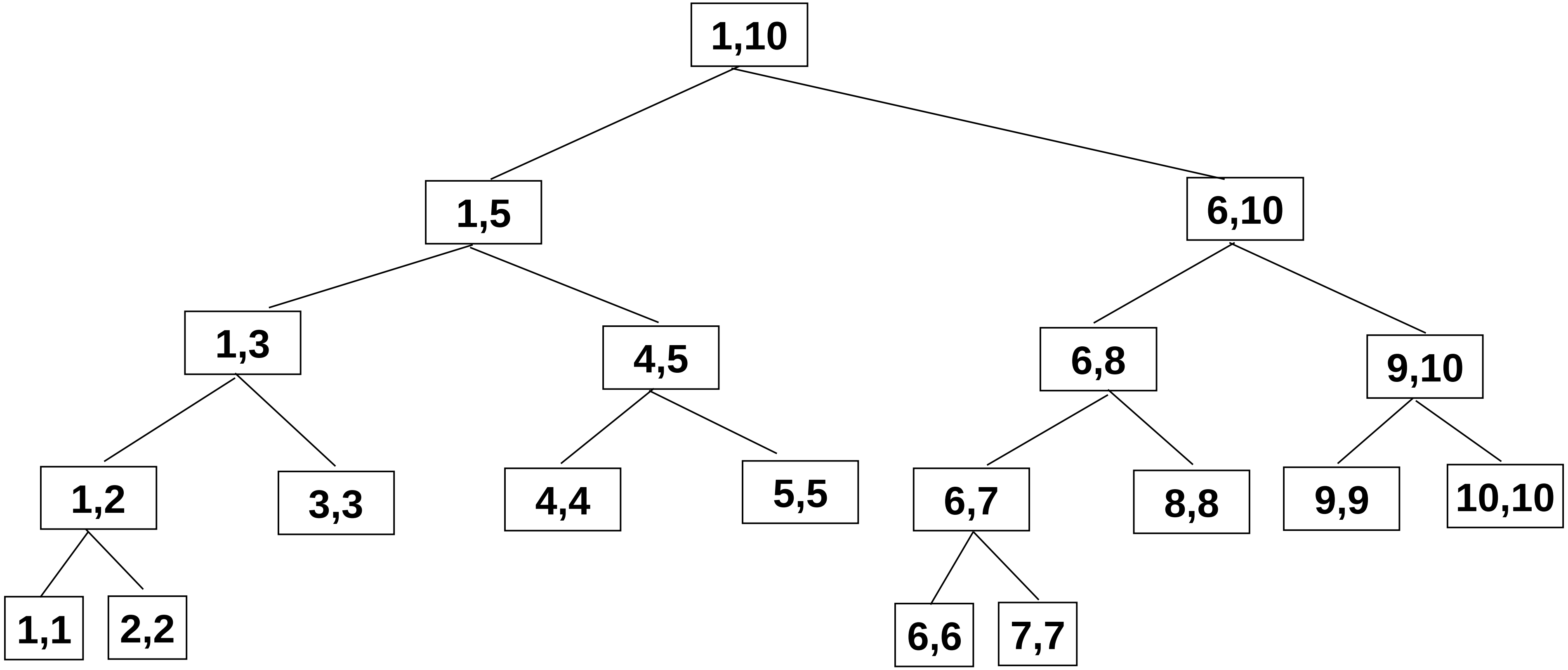
Merge-Sort(A, 0, 7)

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



Ex:- [179, 254, 285, 310, 351, 423, 450, 520, 652,861]

Tree of calls of merge sort



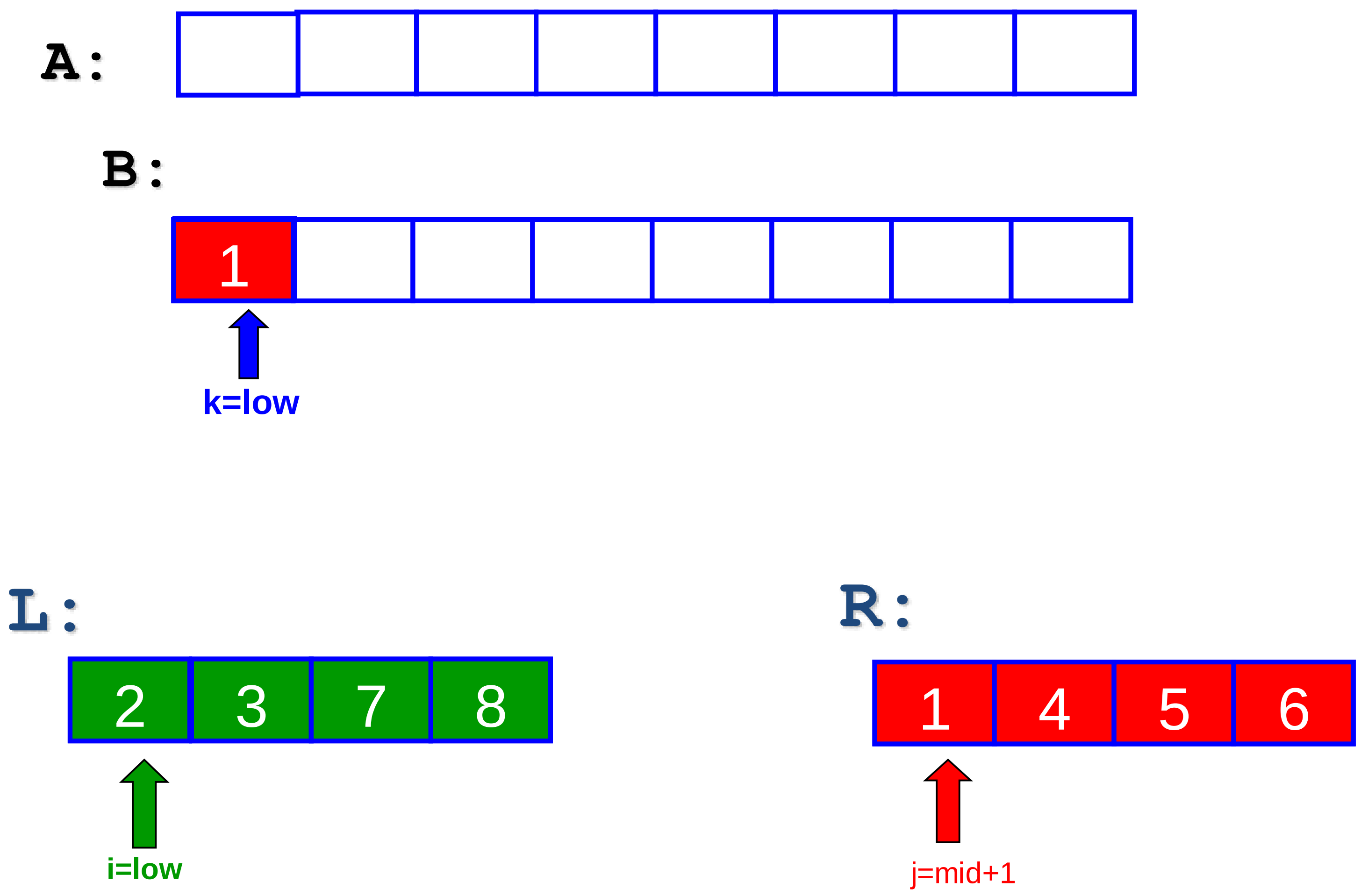
Merge-Sort: Merge Example

	low			mid				high
A:	2	3	7	8	1	4	5	6
B:								

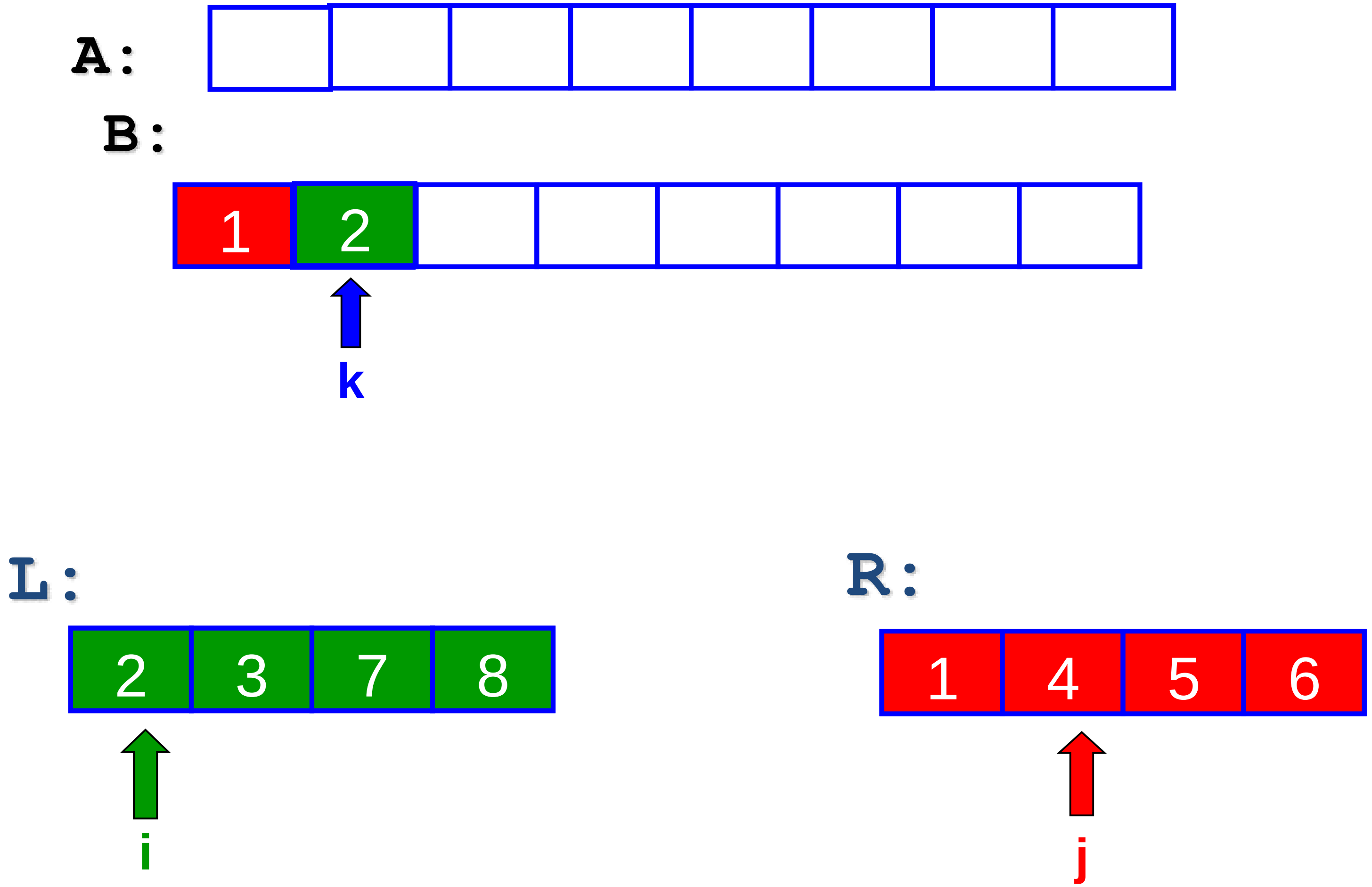
L:

R:

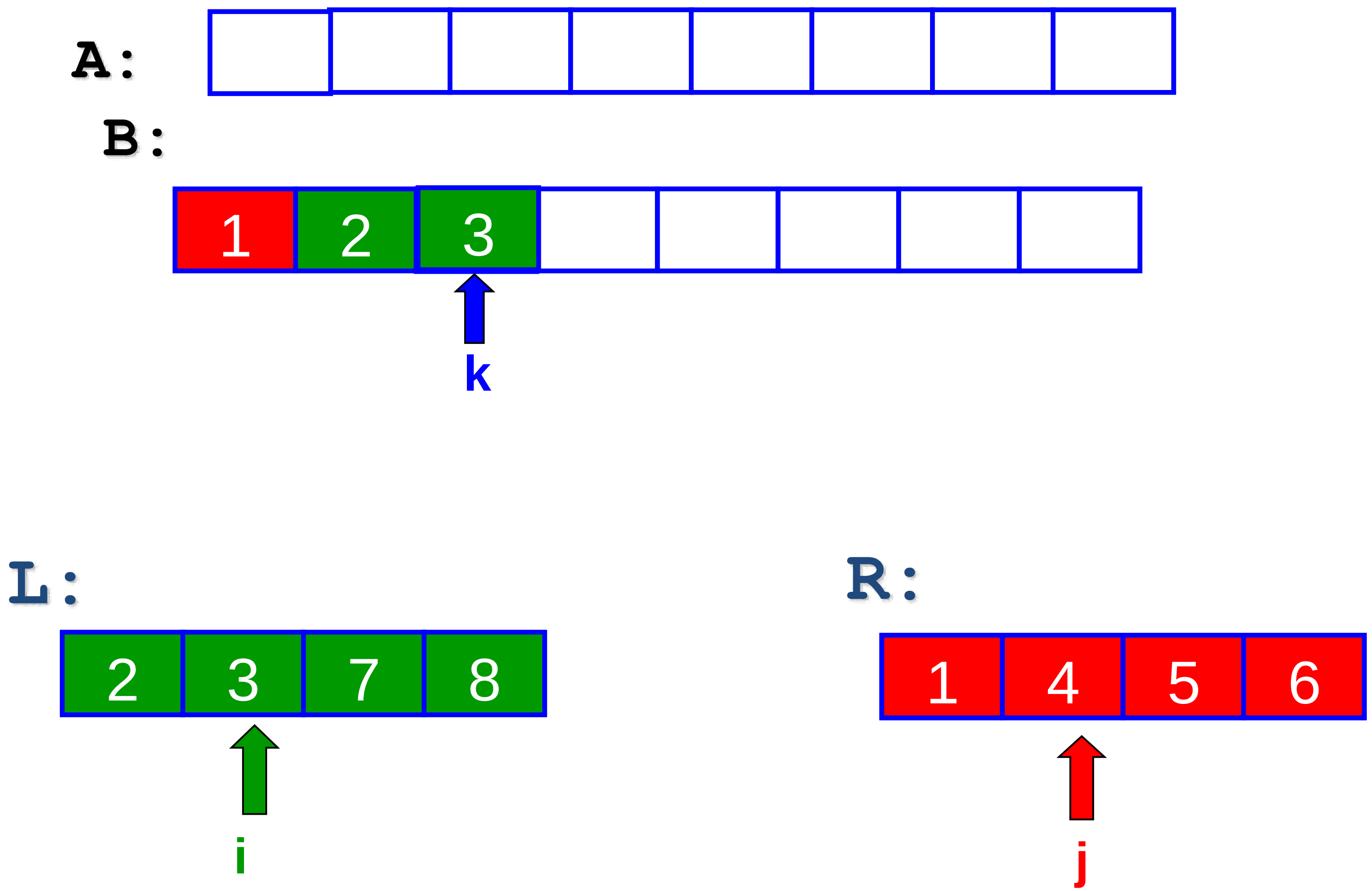
Merge-Sort: Merge Example



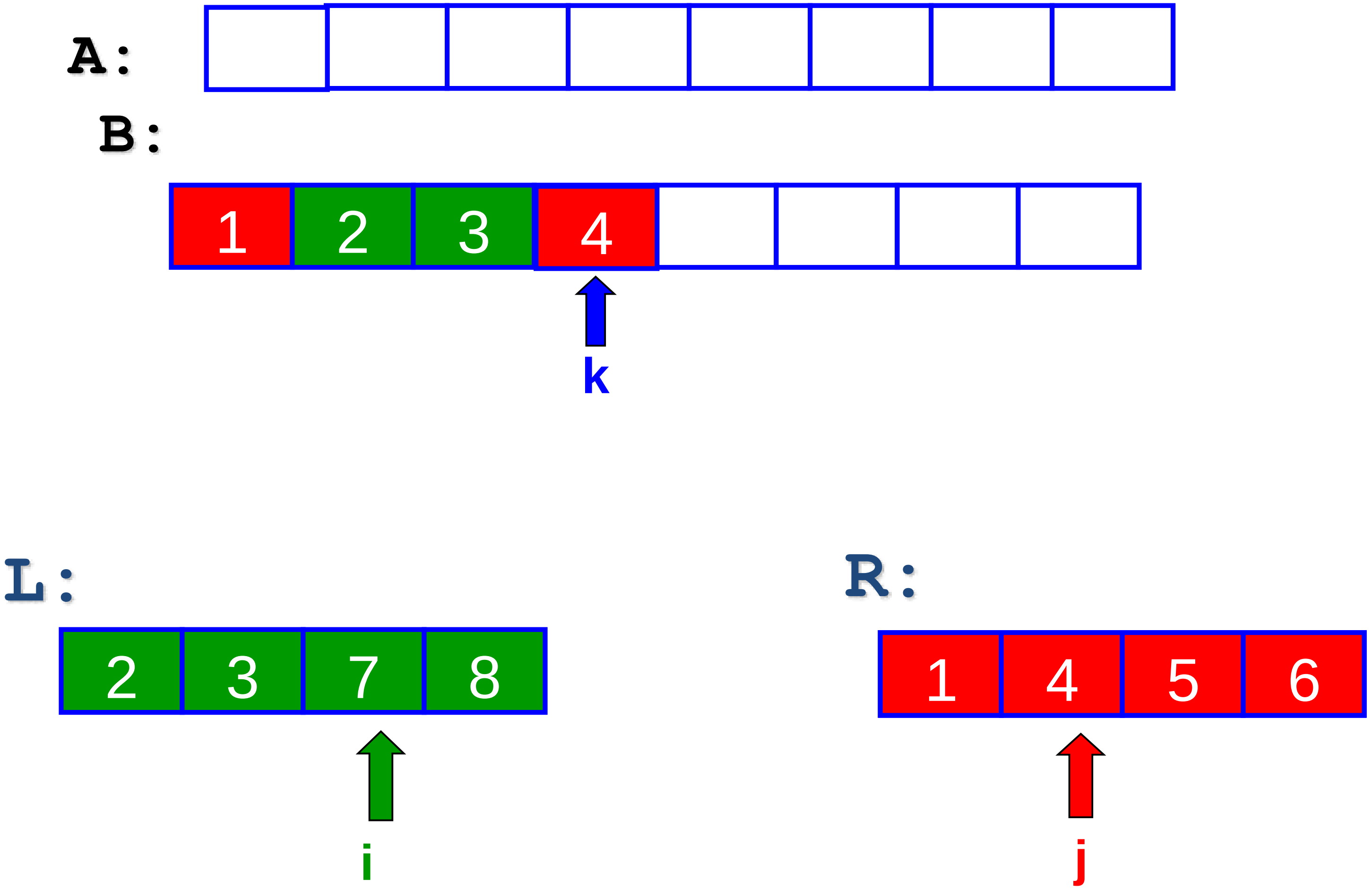
Merge-Sort: Merge Example



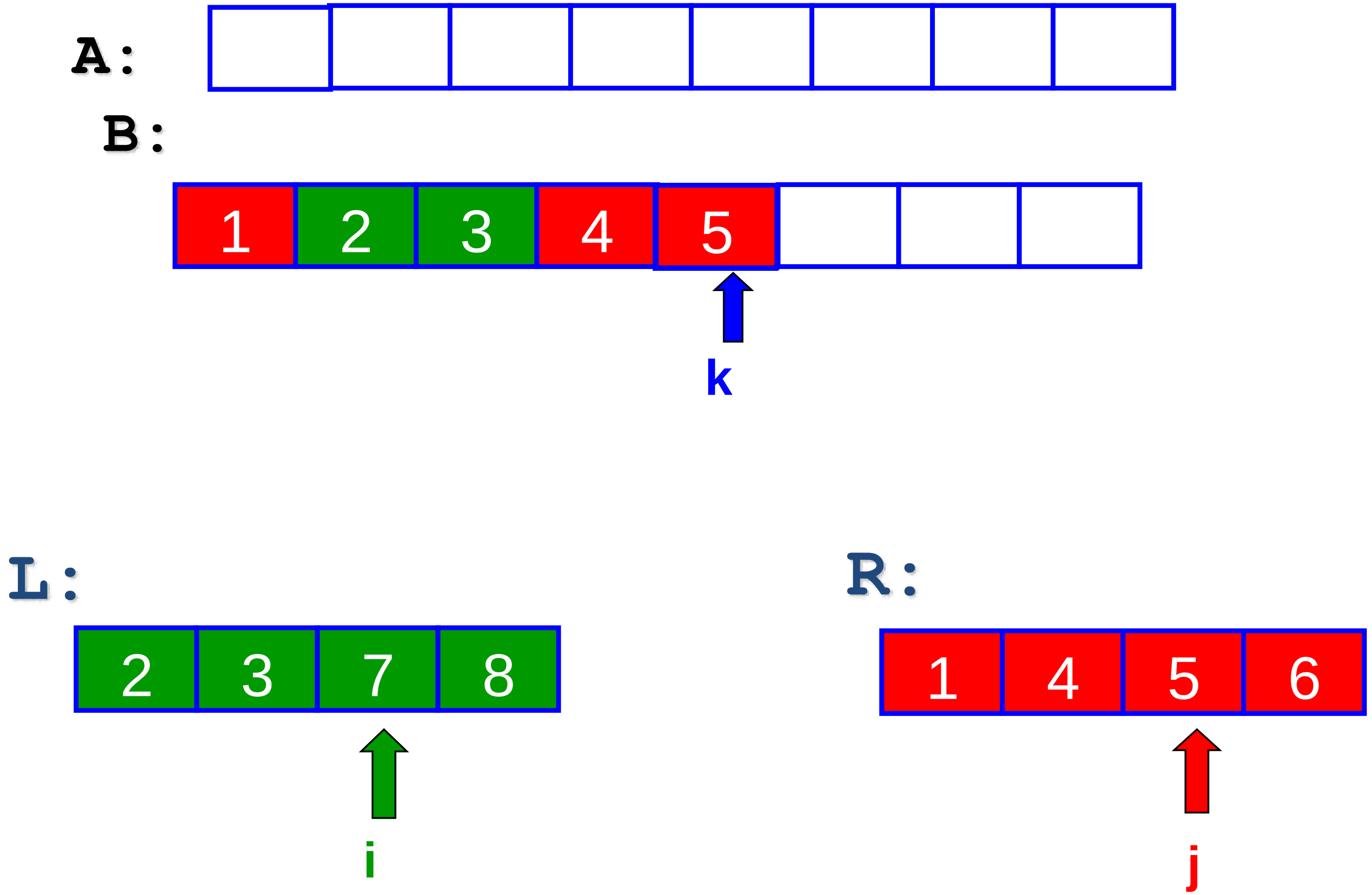
Merge-Sort: Merge Example



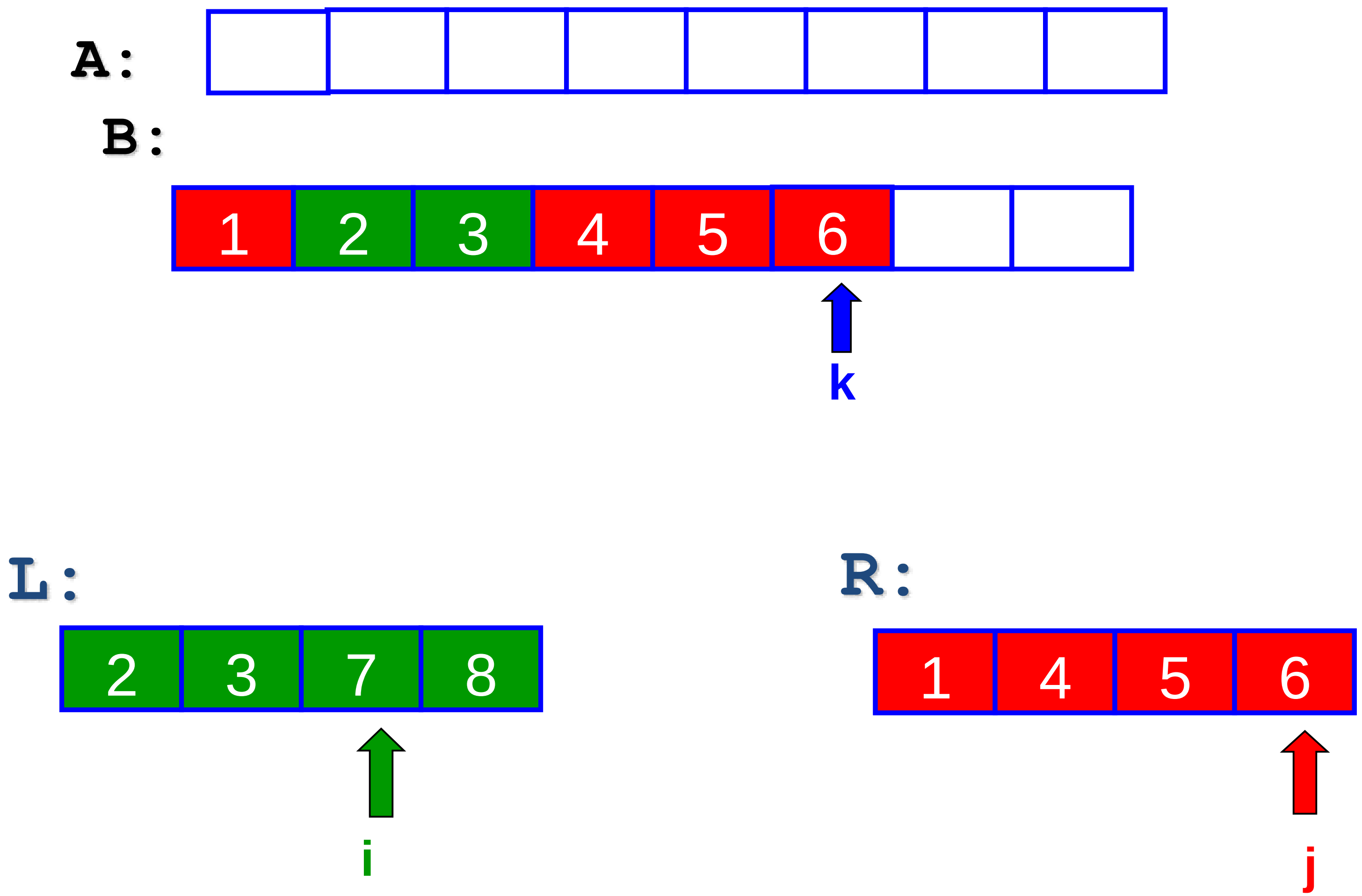
Merge-Sort: Merge Example



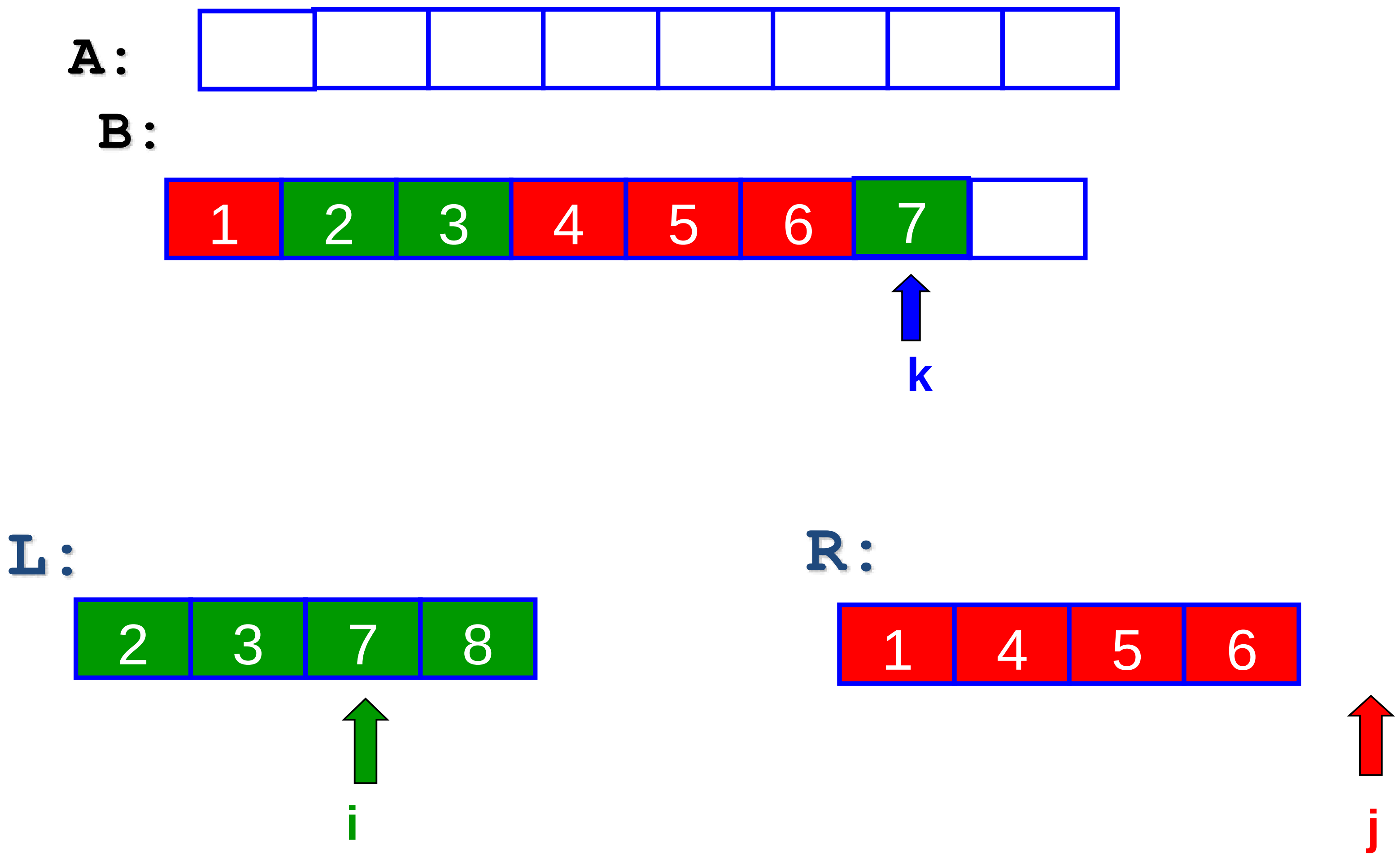
Merge-Sort: Merge Example



Merge-Sort: Merge Example



Merge-Sort: Merge Example



Merge-Sort: Merge Example

A:

B:

1	2	3	4	5	6	7	8

k

L:

2	3	7	8

i

R:

1	4	5	6

j

Merge-Sort: Merge Example

A:

B:

1	2	3	4	5	6	7	8

k

L:

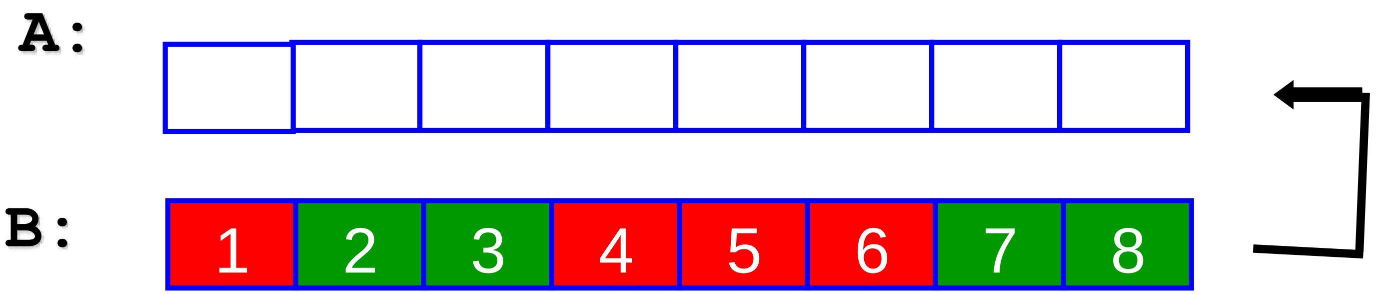
2	3	7	8

i

R:

1	4	5	6

j


A:
B:
1
2
3
4
5
6
7
8

Merge Sort Analysis

If the time for the merging operations is proportional to ‘n’, then the computing time for merge sort is described by the recurrence relation.

$$T(n) = \begin{cases} a & n=1, \text{'}a\text{'} \text{ a constant} \\ 2T(n/2)+cn & n>1, \text{'}c\text{'} \text{ a constant.} \end{cases}$$

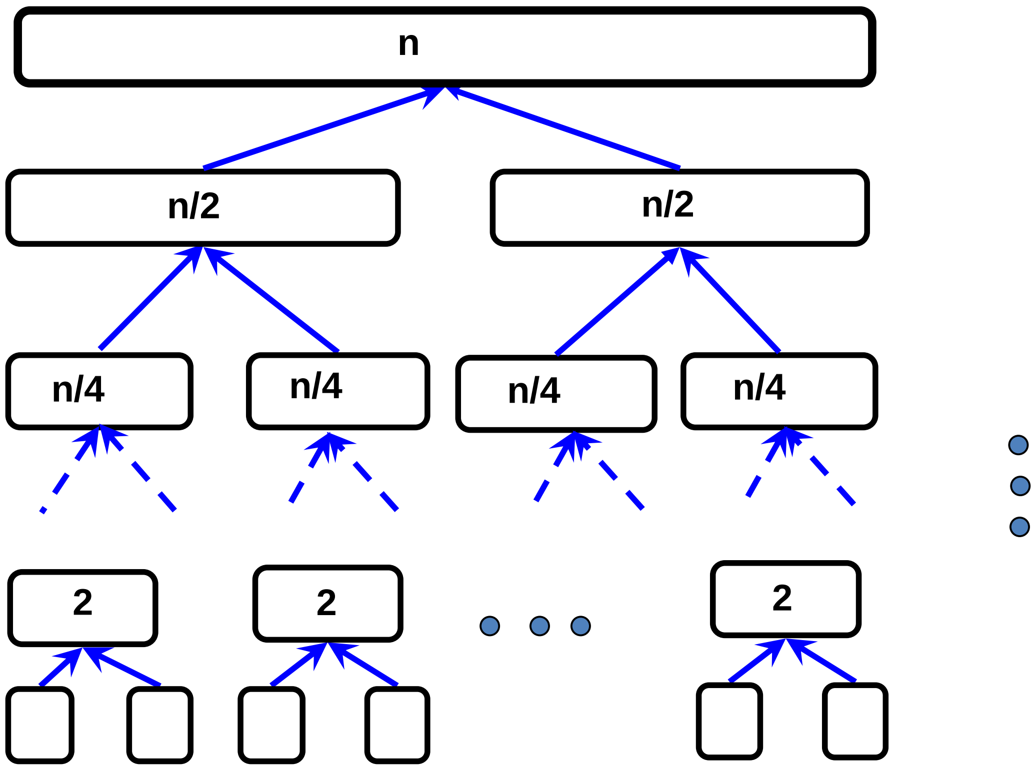
Mergesort Analysis

- Let T(N) be the running time for an array of N elements

- Mergesort divides array in half and calls itself on the two halves. After returning, it merges both halves using a temporary array

- Each recursive call takes T(N/2) and merging takes cn

Best/good Case



- Total time: O(nlogn)

Best Case Time Complexity

When n is a power of 2, $n = 2^k$, we can solve this equation by successive substitutions:

$$\begin{aligned}
T(n) &= 2(2T(n/4) + cn/2) + cn \\
&= 4T(n/4) + 2cn \\
&= 4(2T(n/8) + cn/4) + 2cn \\
&\vdots \\
&= 2^k T(1) + kcn \\
&= an + cn \log n
\end{aligned}$$

It is easy to see that if $2^k < n \leq 2^{k+1}$, then $T(n) \leq T(2^{k+1})$. Therefore

$$T(n) = O(n \log n)$$

SUMMARY

- **Divide and Conquer Strategy**: Breaks down a problem into smaller sub-problems, solves each sub-problem independently, and combines their solutions for the final result.

- **Recursive Process**: Continuously splits the input into smaller sections until the base case (often a single element) is reached.

- **Merge Sort Example**: Uses divide and conquer to split an array into two halves recursively.

- **Merging**: Combines the sorted sub-arrays into a single sorted array.

SELF-ASSESSMENT QUESTIONS

What is the average case time complexity of merge sort?

(a) O(n log n)

(b) $O(n^2)$

(c) $O(n^2 \log n)$

1. $O(n \log n^2)$

The divide and conquer approach is most suitable for problems that can be:

(a) Broken down into non-overlapping sub-problems.

(b) Solved in a linear fashion.

(c) Solved without recursion.

(d) Solved iteratively.

TERMINAL QUESTIONS

1. Explain how the divide and conquer strategy is used in merge sort. What are the main steps involved?
2. Explain how the divide and conquer approach can be applied to solving problems other than sorting. Provide at least two examples.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books :

1. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein., 3rd, 2009, The MIT Press.

2 Algorithm Design Manual, Steven S. Skiena., 2nd, 2008, Springer.

3 Data Structures and Algorithms in Python, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser., 2nd, 2013, Wiley.

4 The Art of Computer Programming, Donald E. Knuth, 3rd, 1997, Addison-Wesley Professiona.

MOOCS :

1. https://www.coursera.org/specializations/algorithms?=

2.https://www.coursera.org/learn/dynamic-programming-greedy-algorithms#modules

THANK YOU