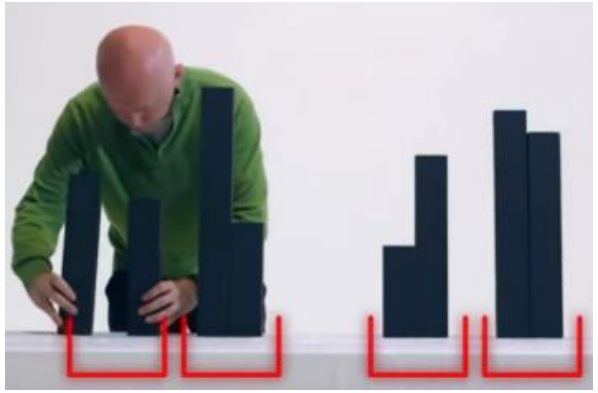
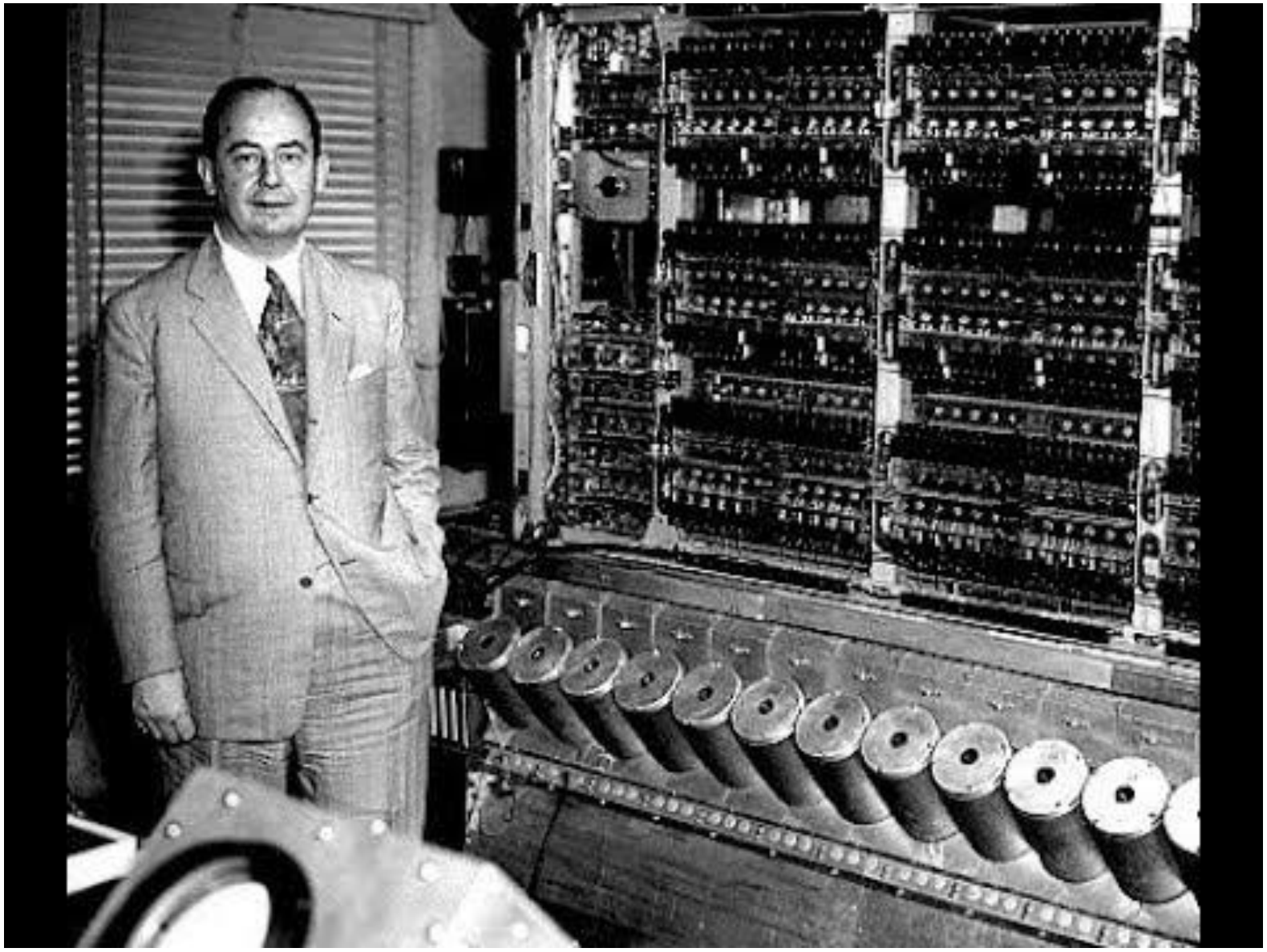
Mergesort

Recursive and O(n log n)

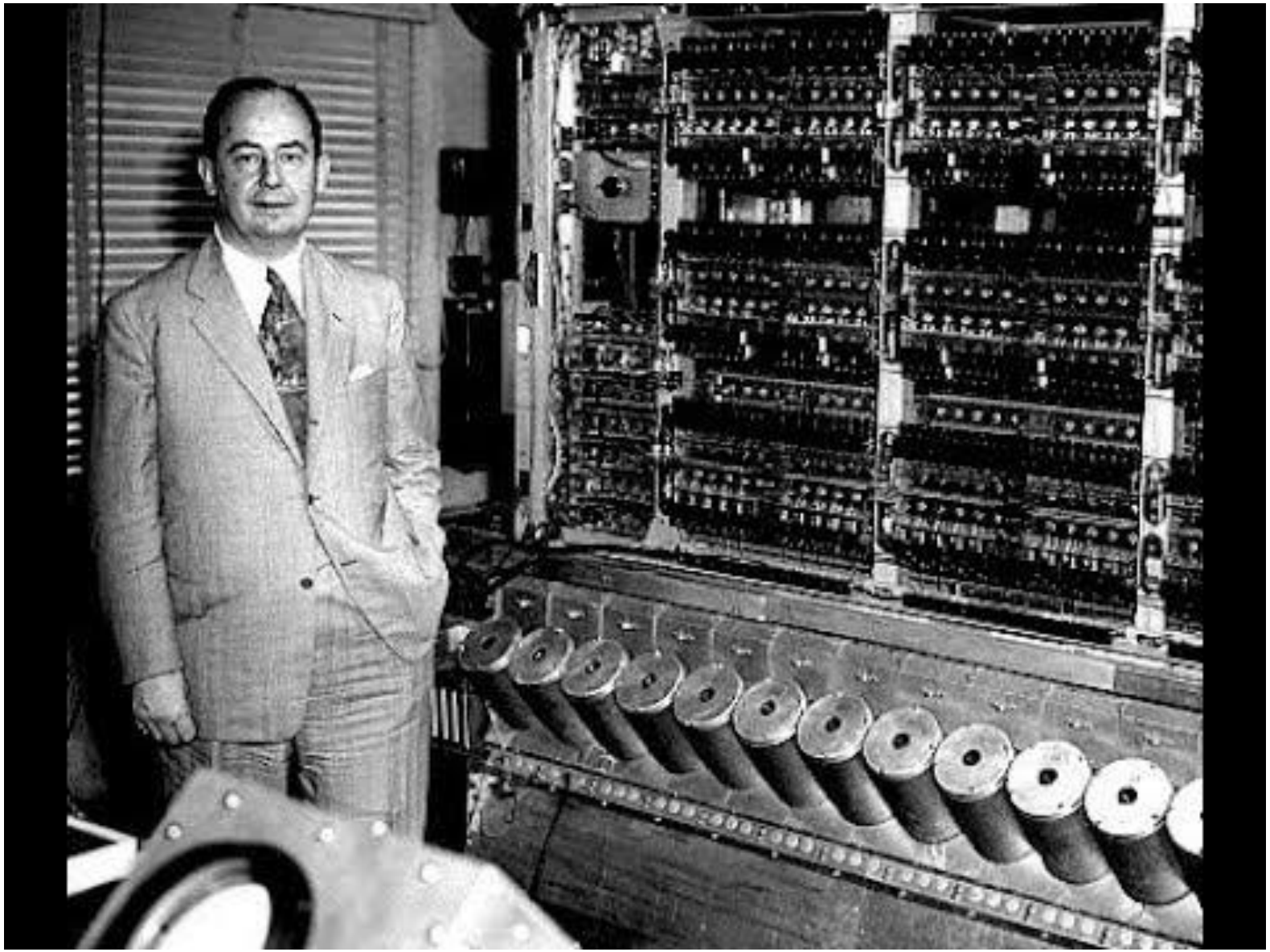


John Von Neumann



John Von Neumann

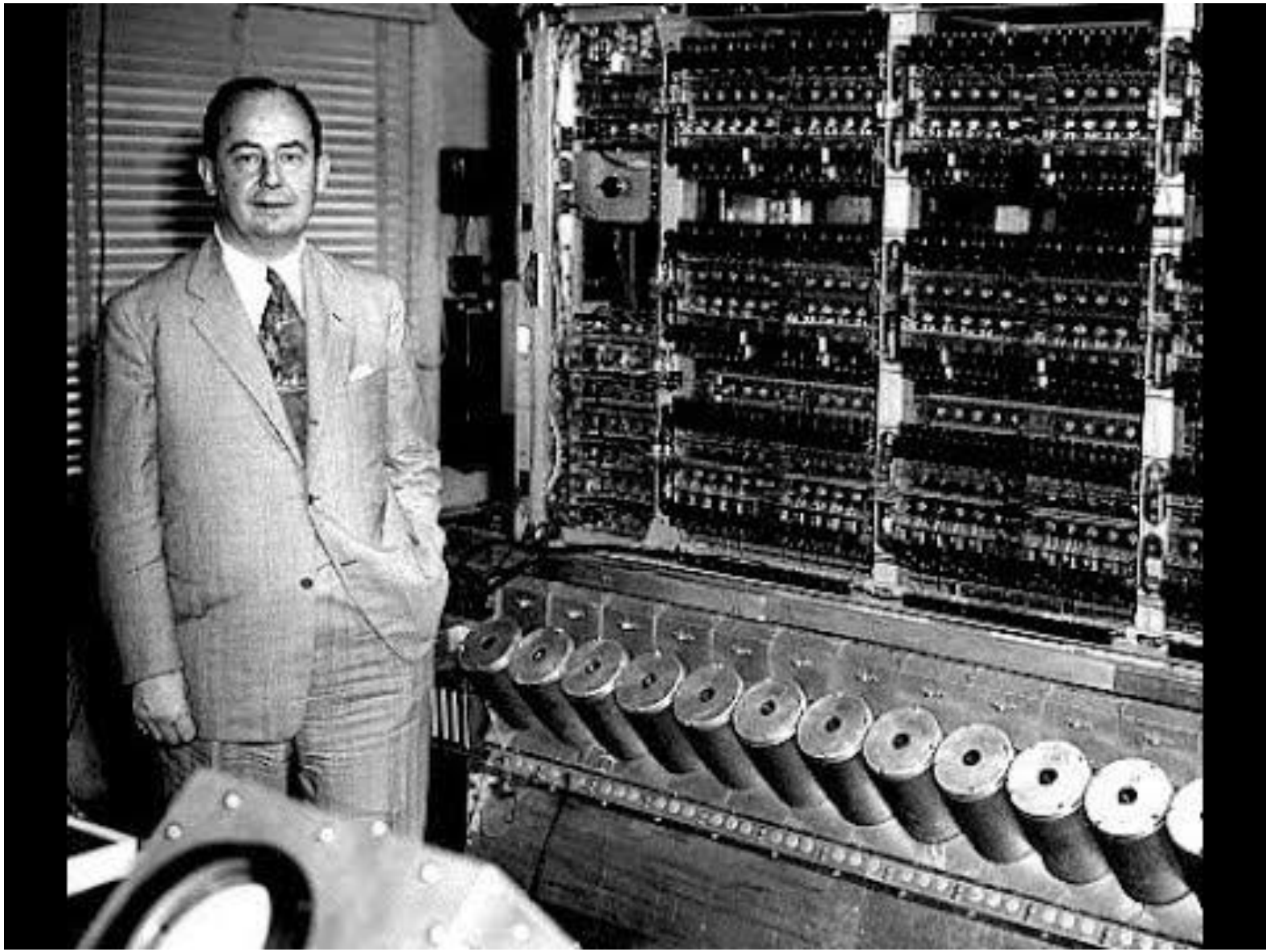
Foundations of Mathematics



John Von Neumann

Game theory

Foundations of Mathematics

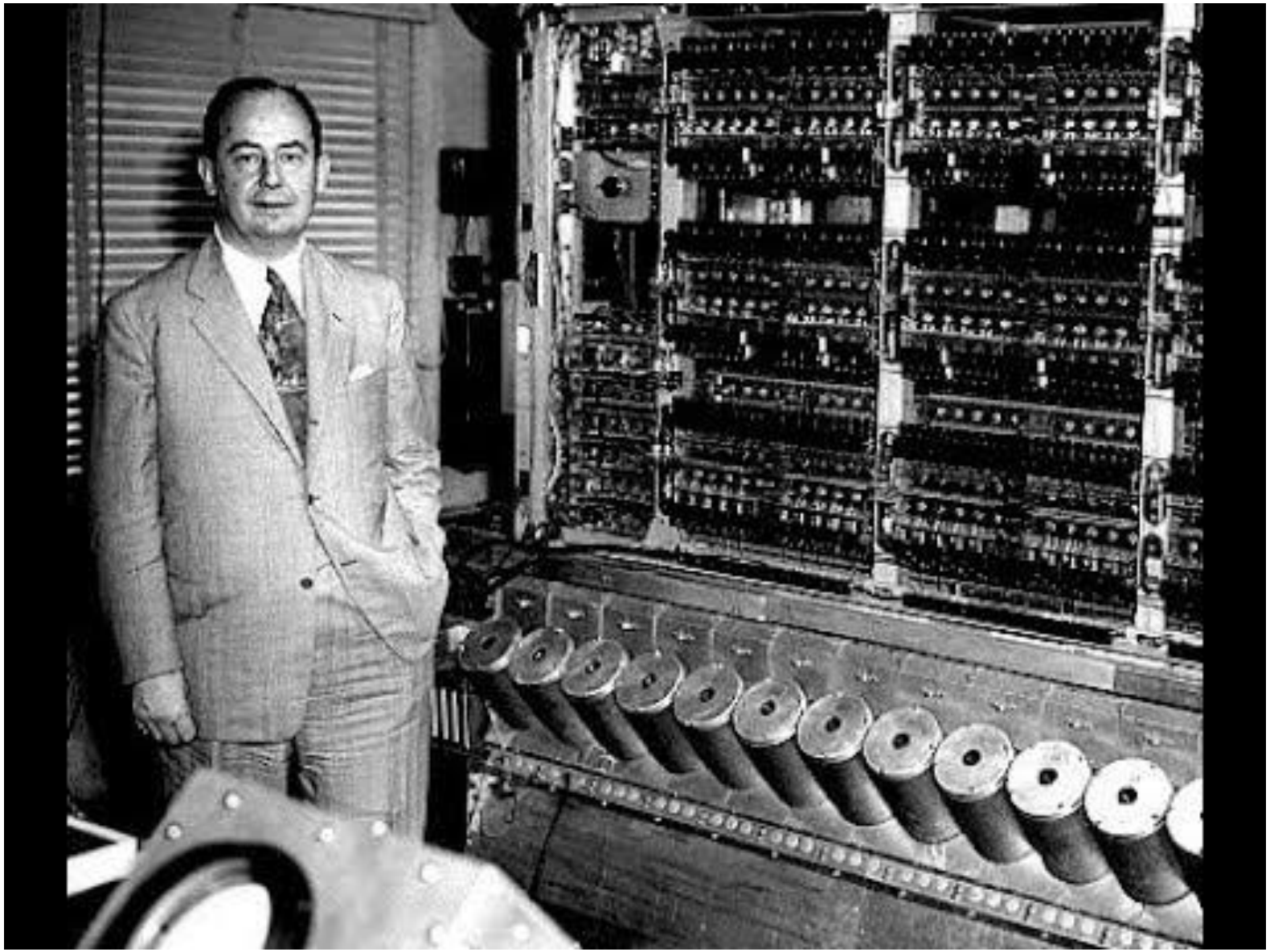


John Von Neumann

Game theory

Foundations of Mathematics

Digital Computer



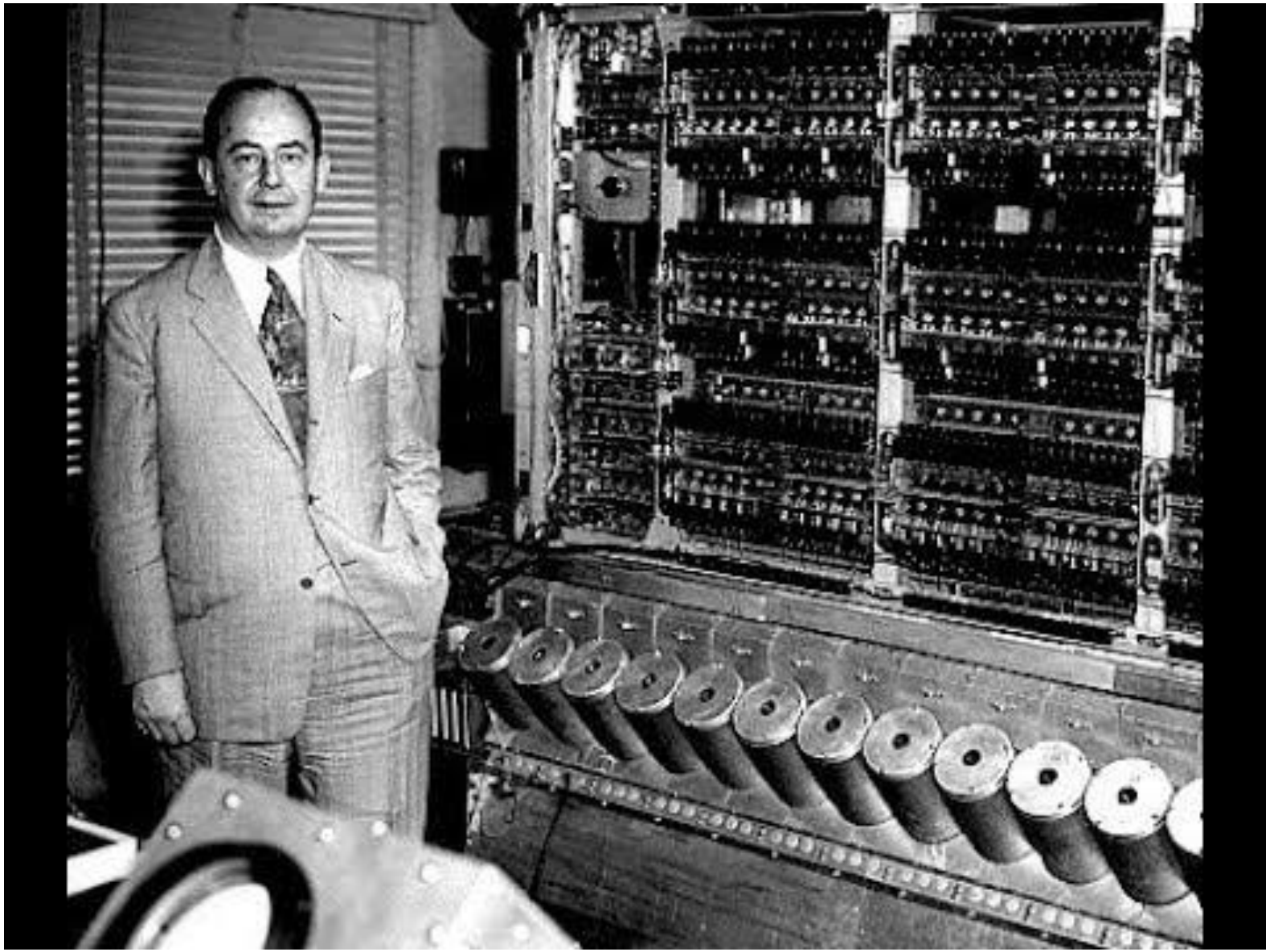
John Von Neumann

Game theory

Foundations of Mathematics

Digital Computer

ENIAC



John Von Neumann

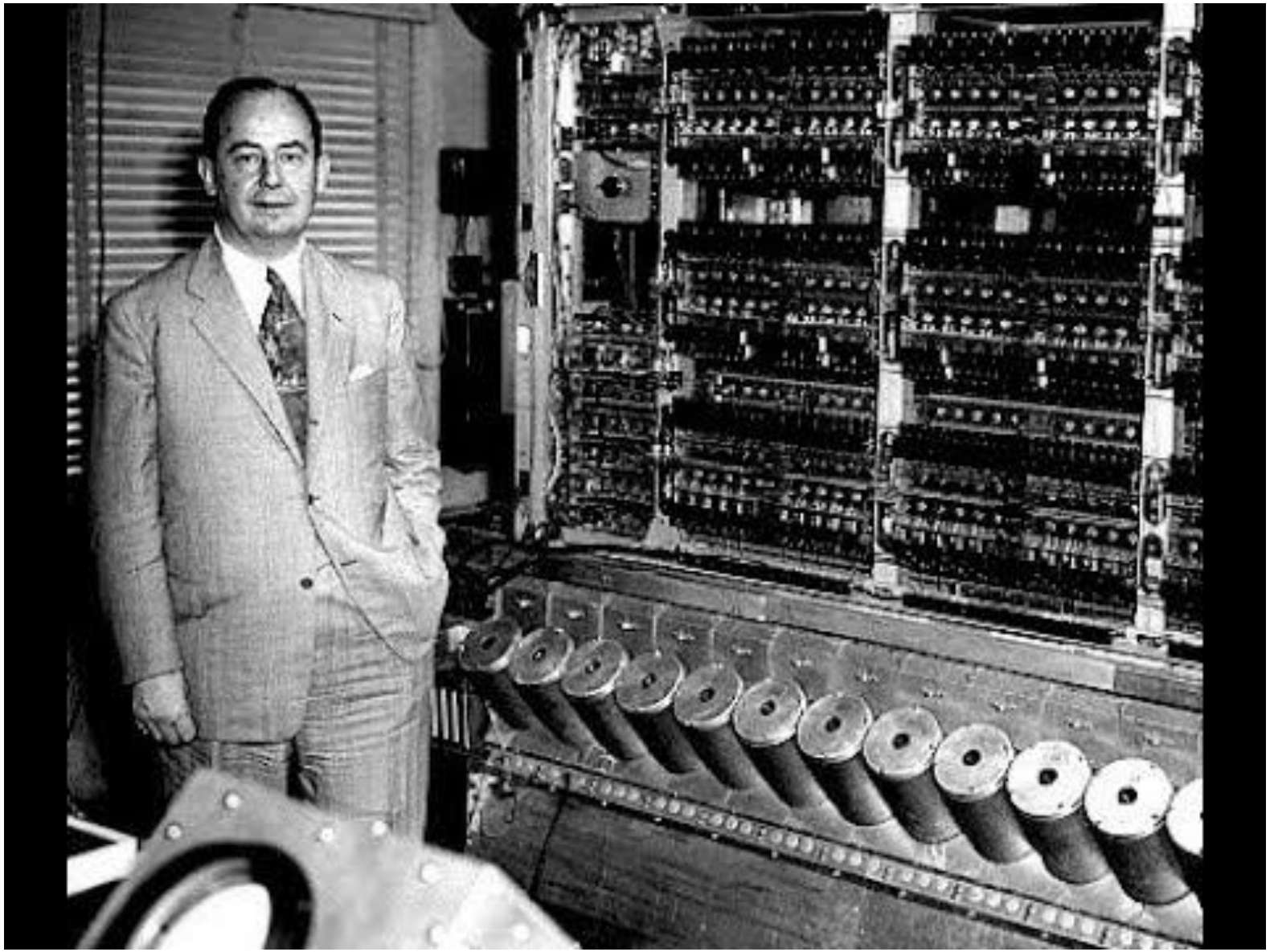
Game theory

Foundations of Mathematics

Digital Computer

ENIAC

First Climate Modelling Software



John Von Neumann

Game theory

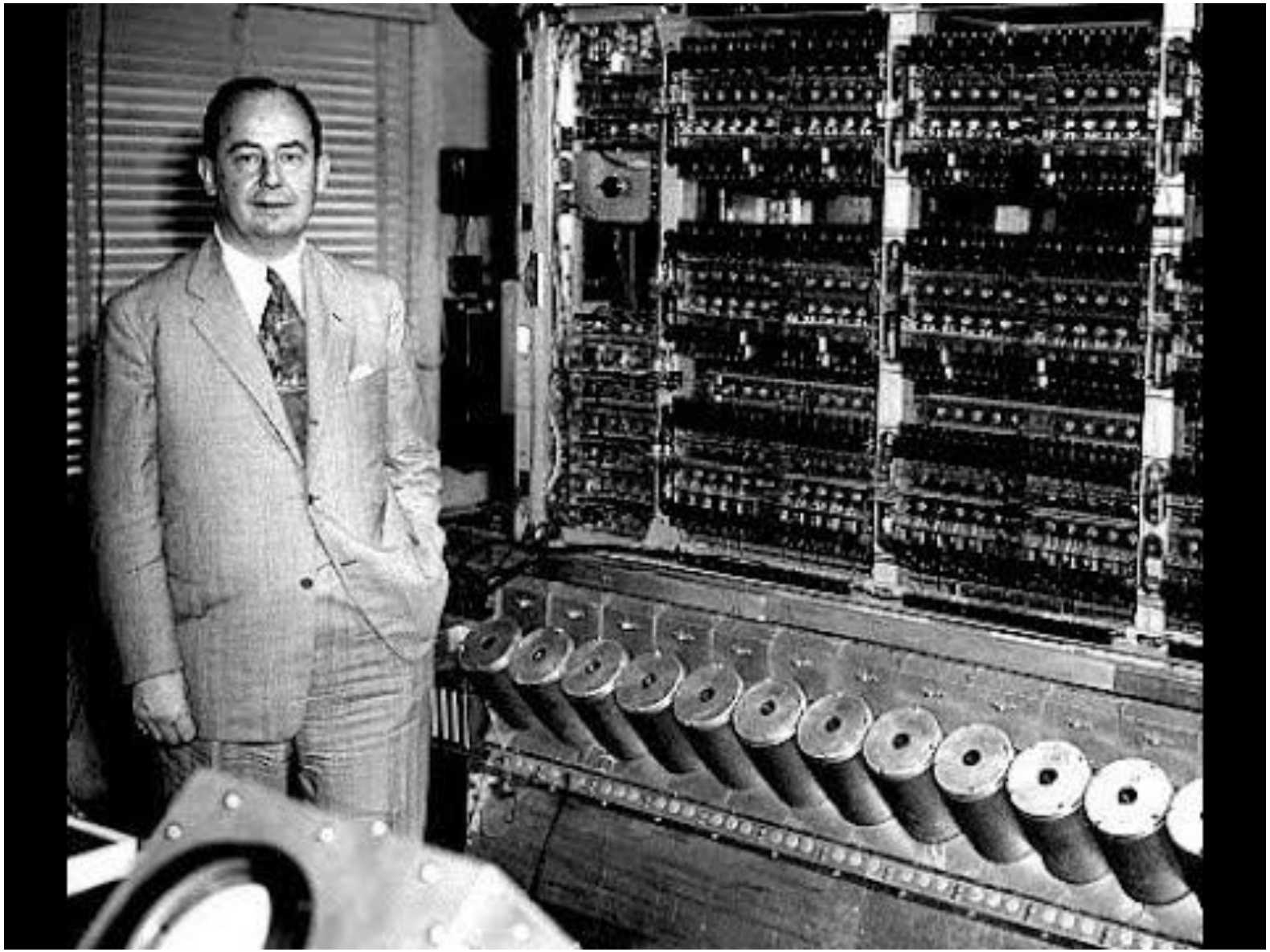
Foundations of Mathematics

Digital Computer

ENIAC

First Climate Modelling Software

Quantum Mechanics



John Von Neumann

Game theory

Manhattan Project

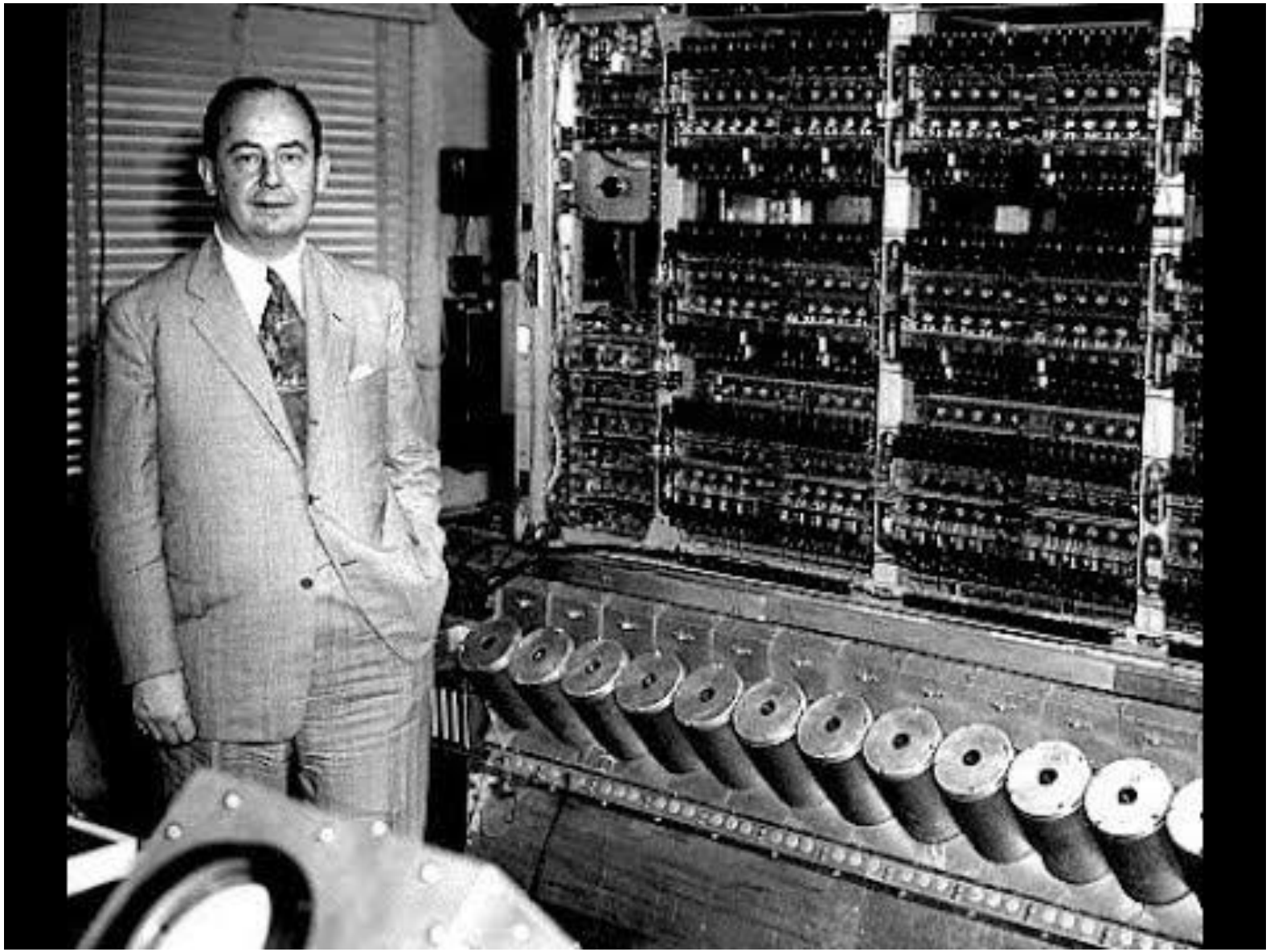
Foundations of Mathematics

Digital Computer

ENIAC

First Climate Modelling Software

Quantum Mechanics

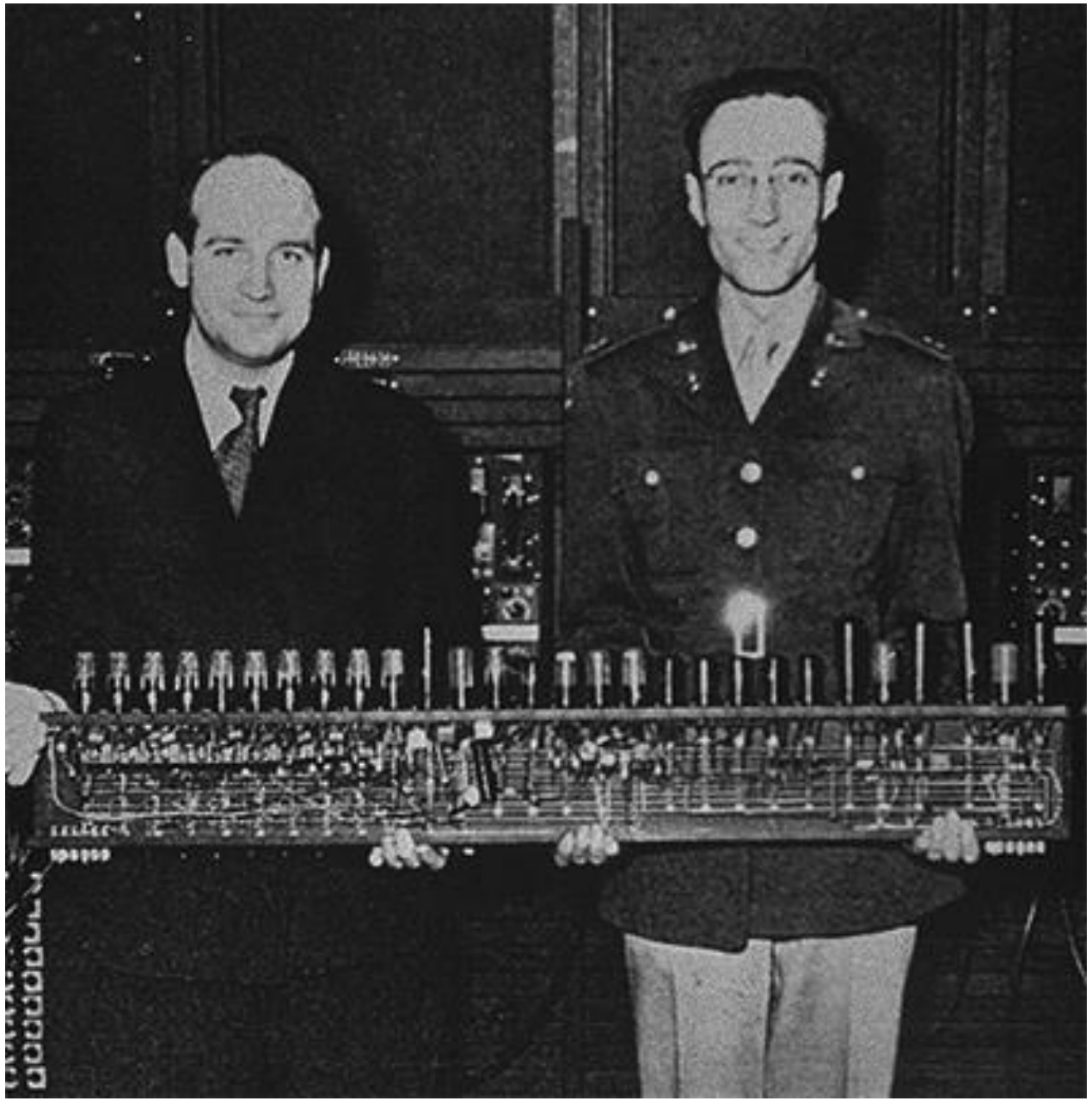


“Keeping up with him was ... impossible. The feeling was you were on a tricycle chasing a racing car.”

To gain a measure of von Neumann's achievements, consider that had he lived a normal span of years, he would certainly have been a recipient of a Nobel Prize in economics. And if there were Nobel Prizes in computer science and mathematics, he would have been honored by these, too. [In addition, von Neumann should be] thought of as a triple Nobel laureate or, possibly, a 4-fold winner, for his work in physics, in particular, quantum mechanics.

John von Neumann planned out this algorithm in 1945.

Programming didn’t yet exist.



Merge Means?

Merge Means?

To join two things together

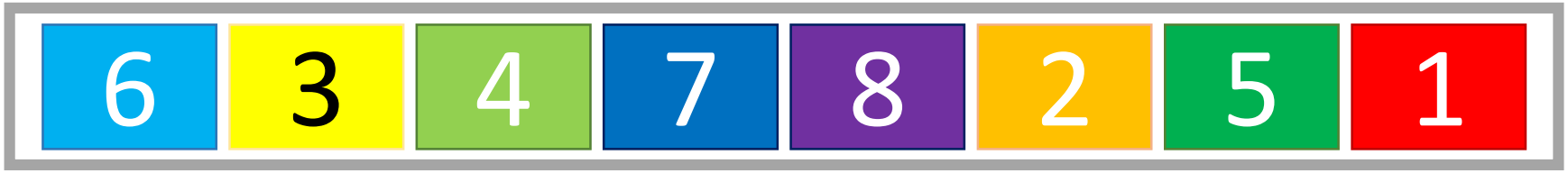
Merge Means?

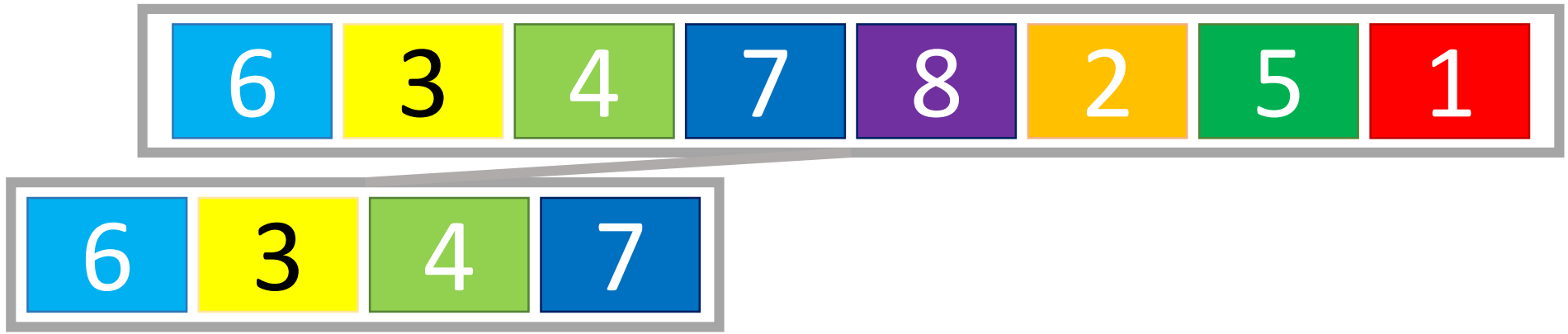
To join two things together

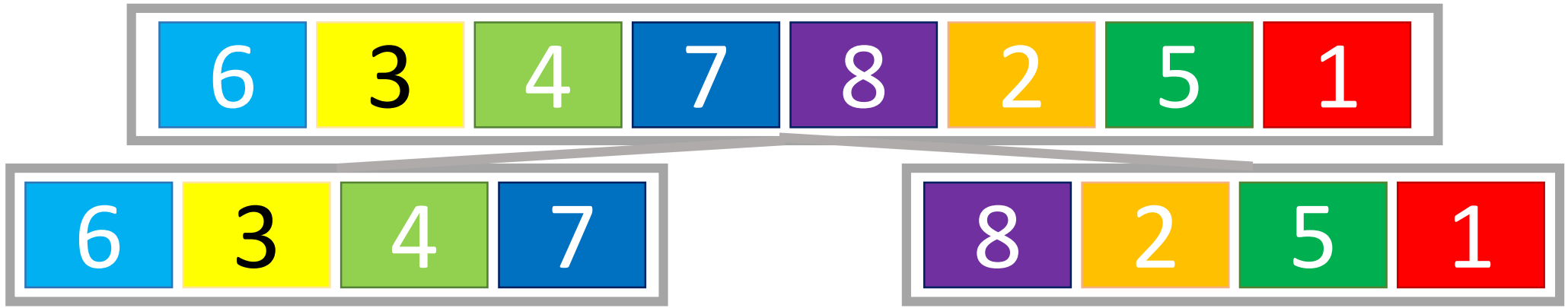
In an orderly fashion

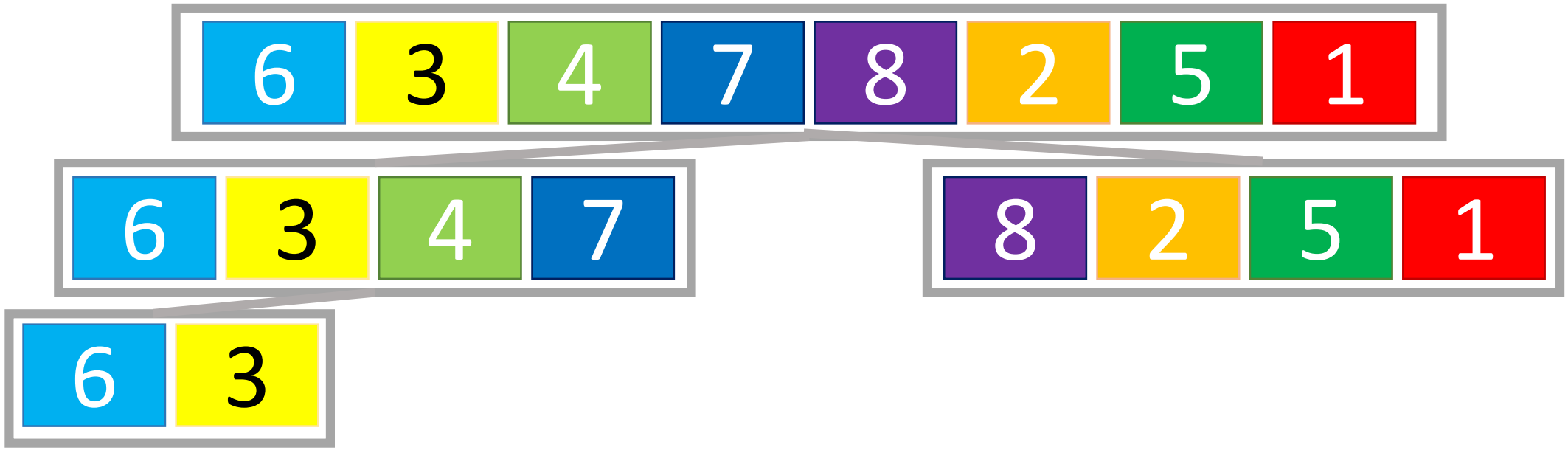
Conceptually, merge sort works as follows:

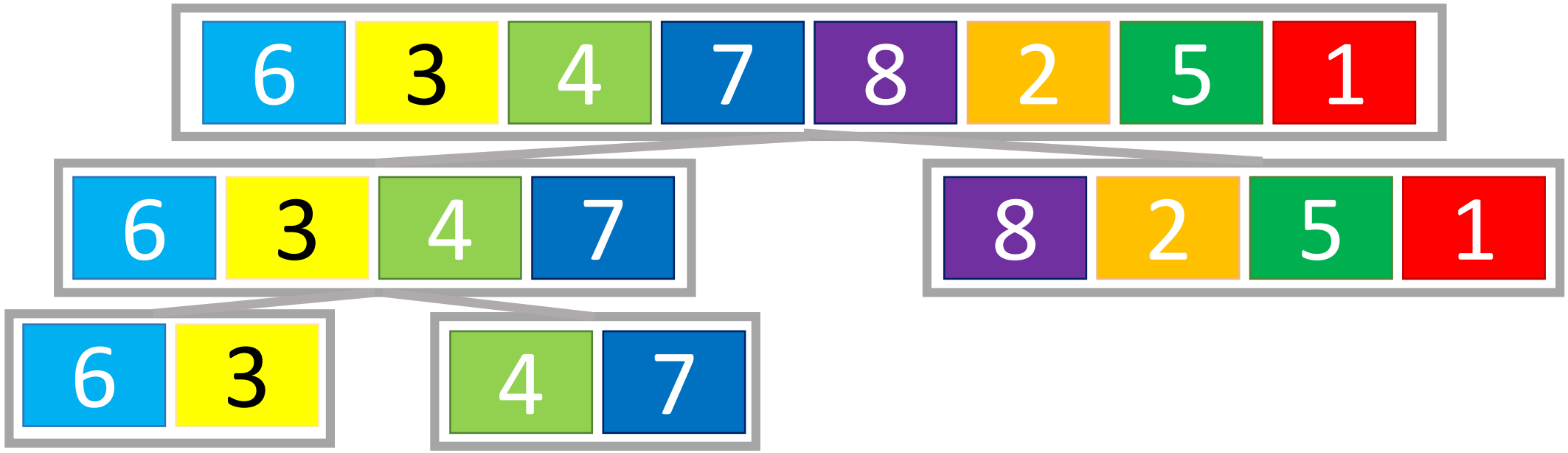
- Divide the unsorted list into n sublists, each containing 1 element (a list of 1 element is considered sorted).
- Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

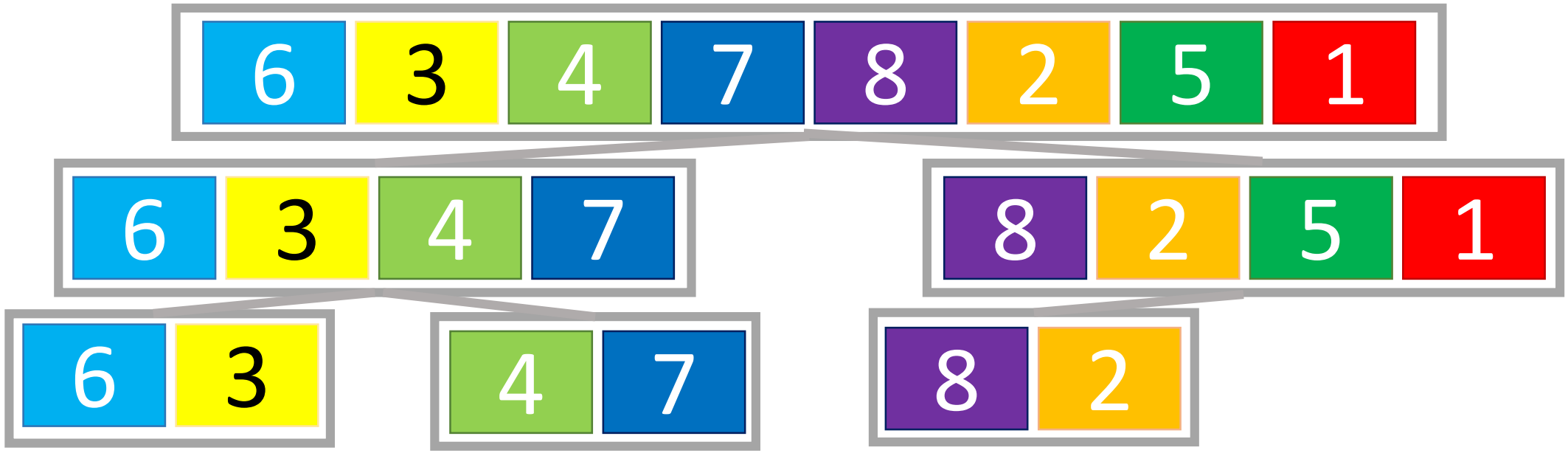

6
3
4
7
8
2
5
1

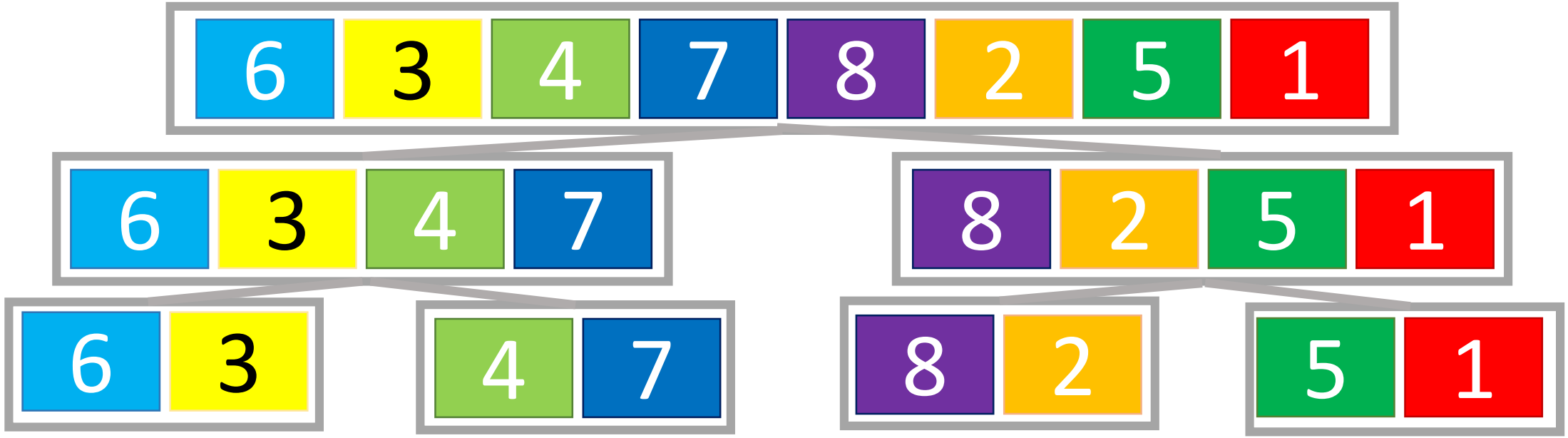

6
3
4
7
8
2
5
1
6
3
4
7

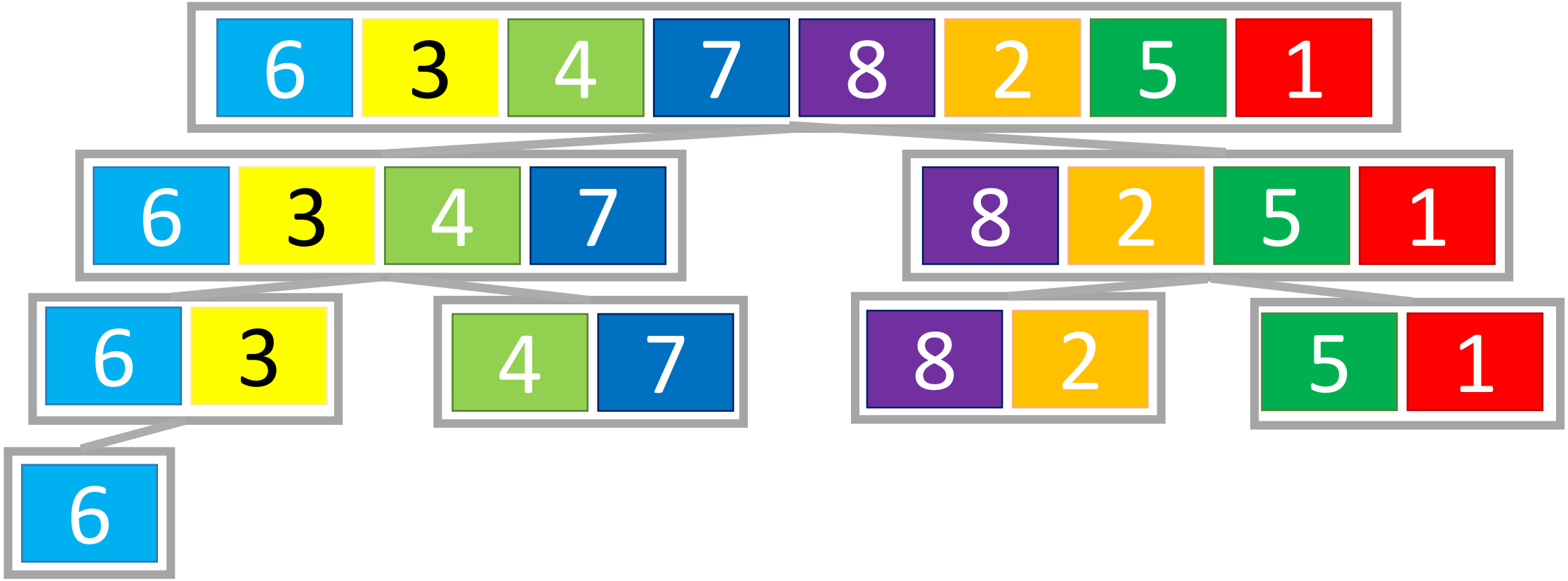

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1

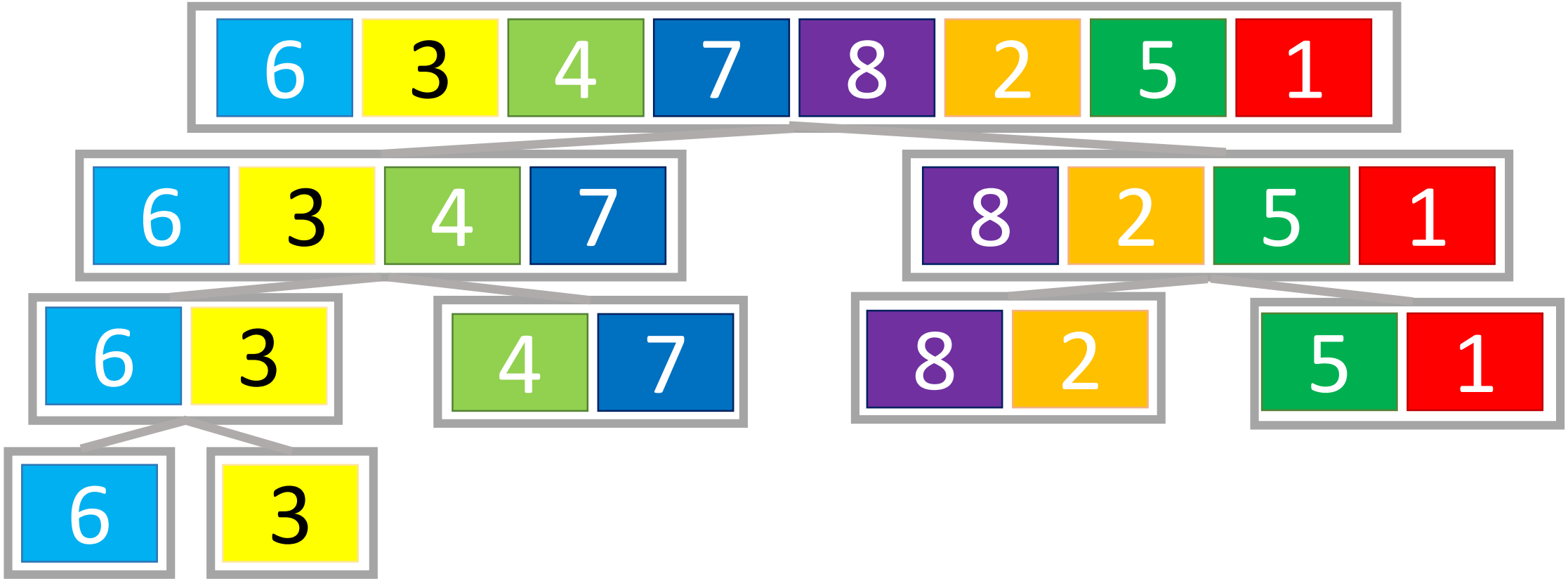

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3

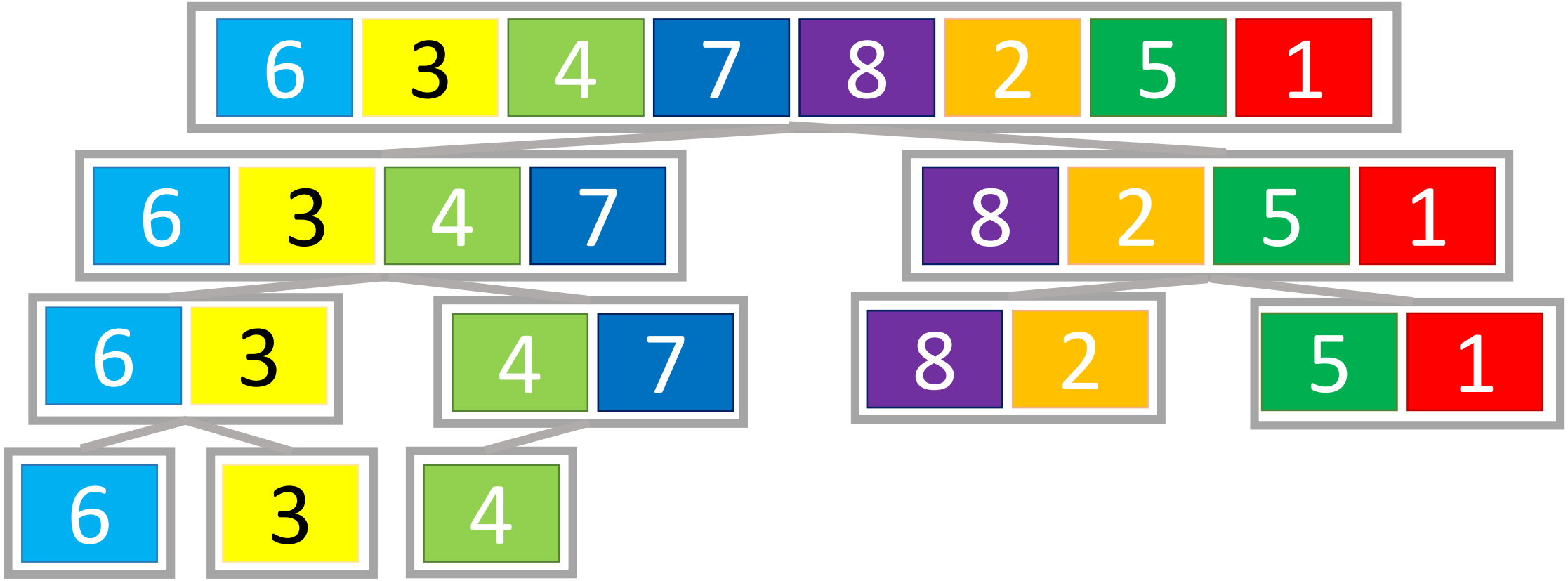

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7

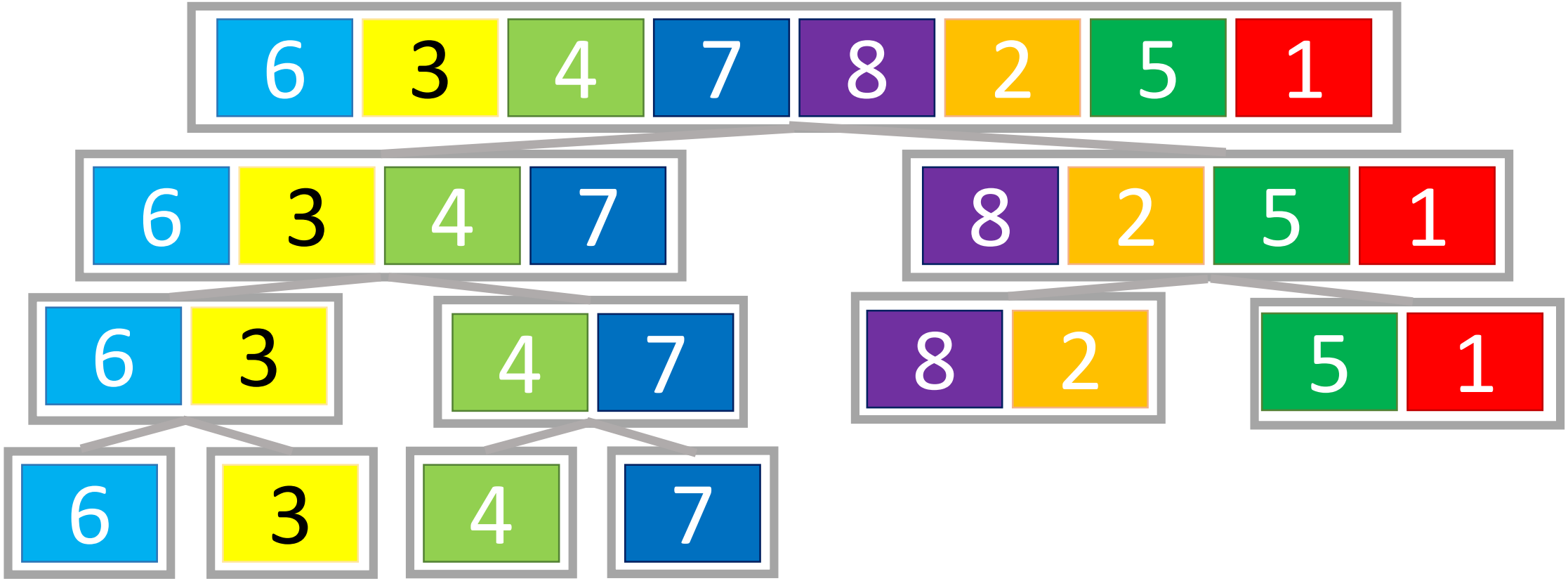

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2

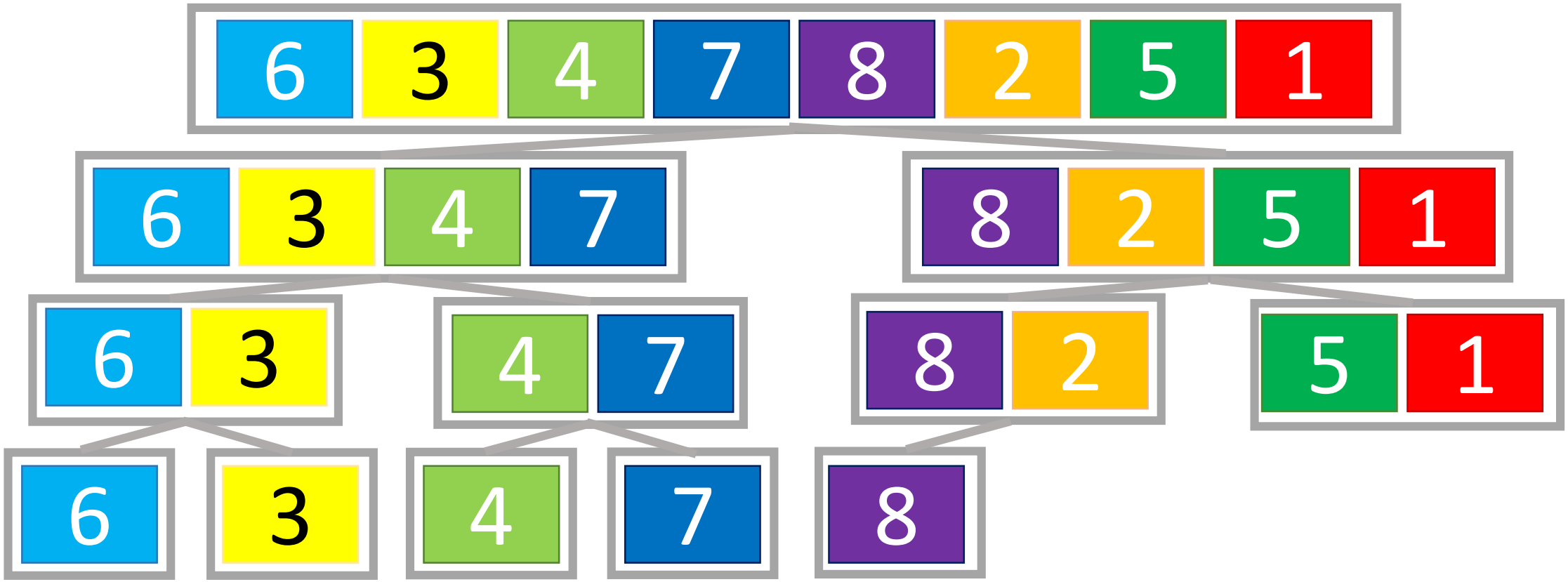

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1

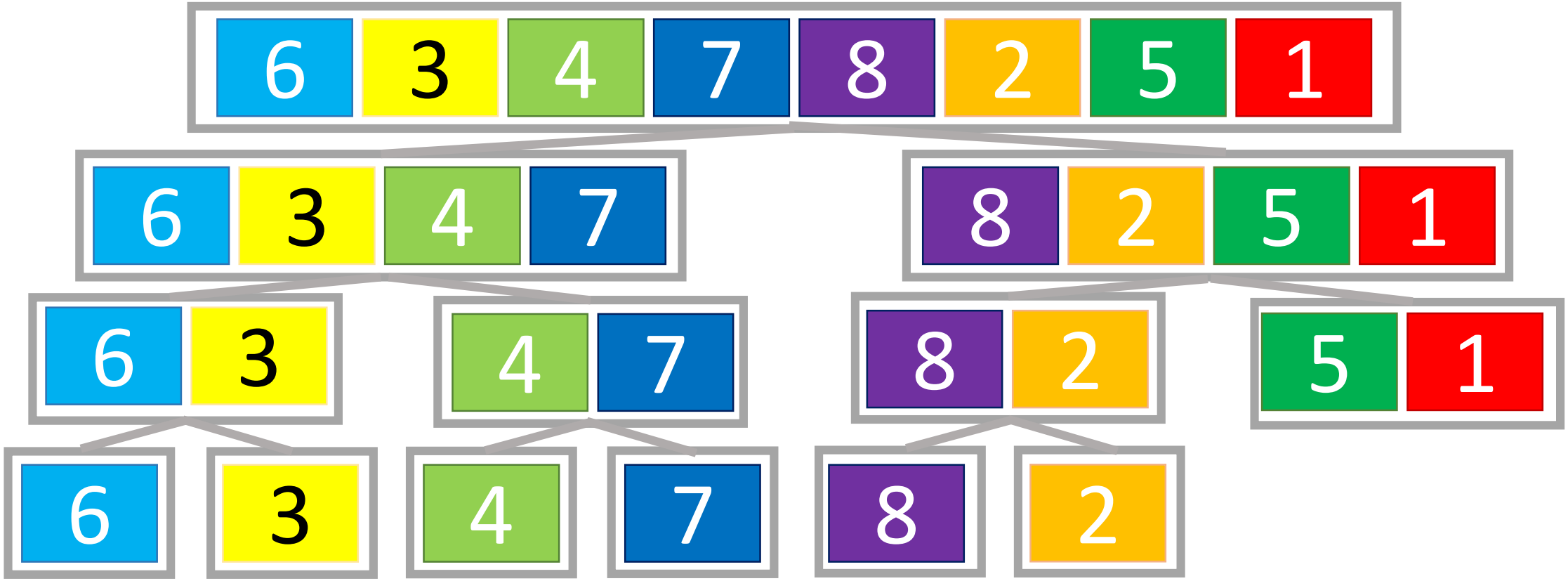

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6

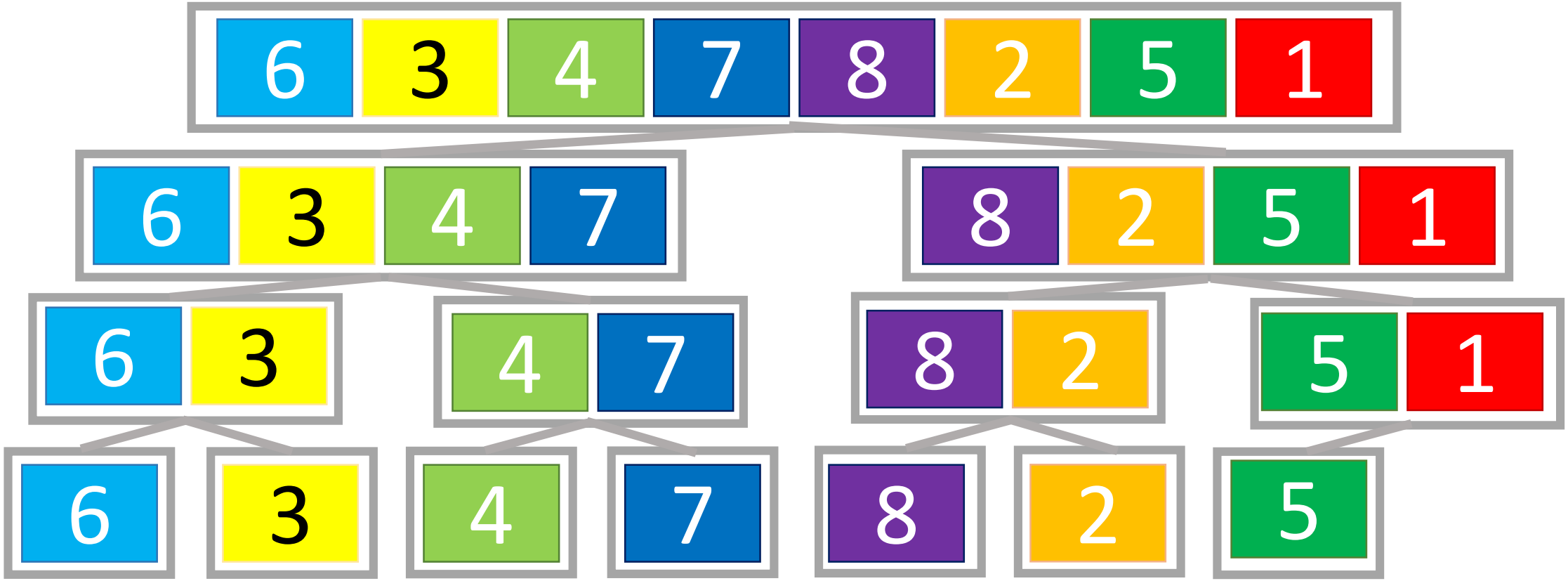

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3

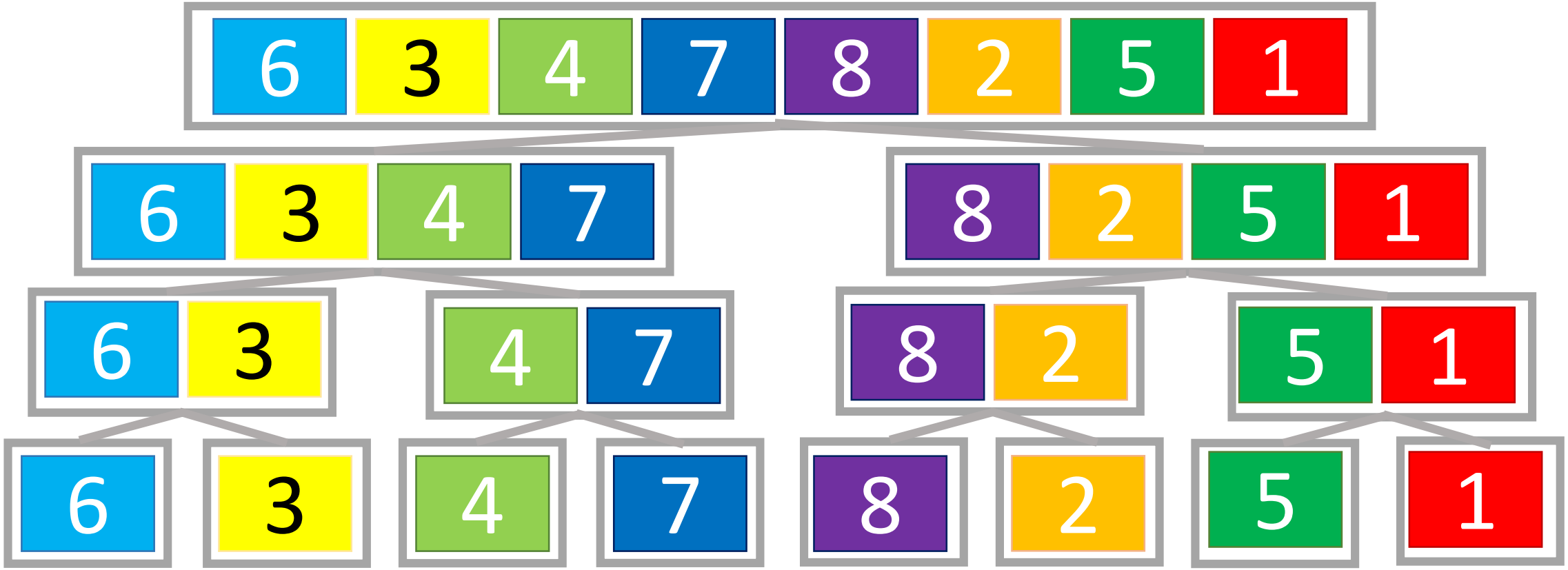

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4

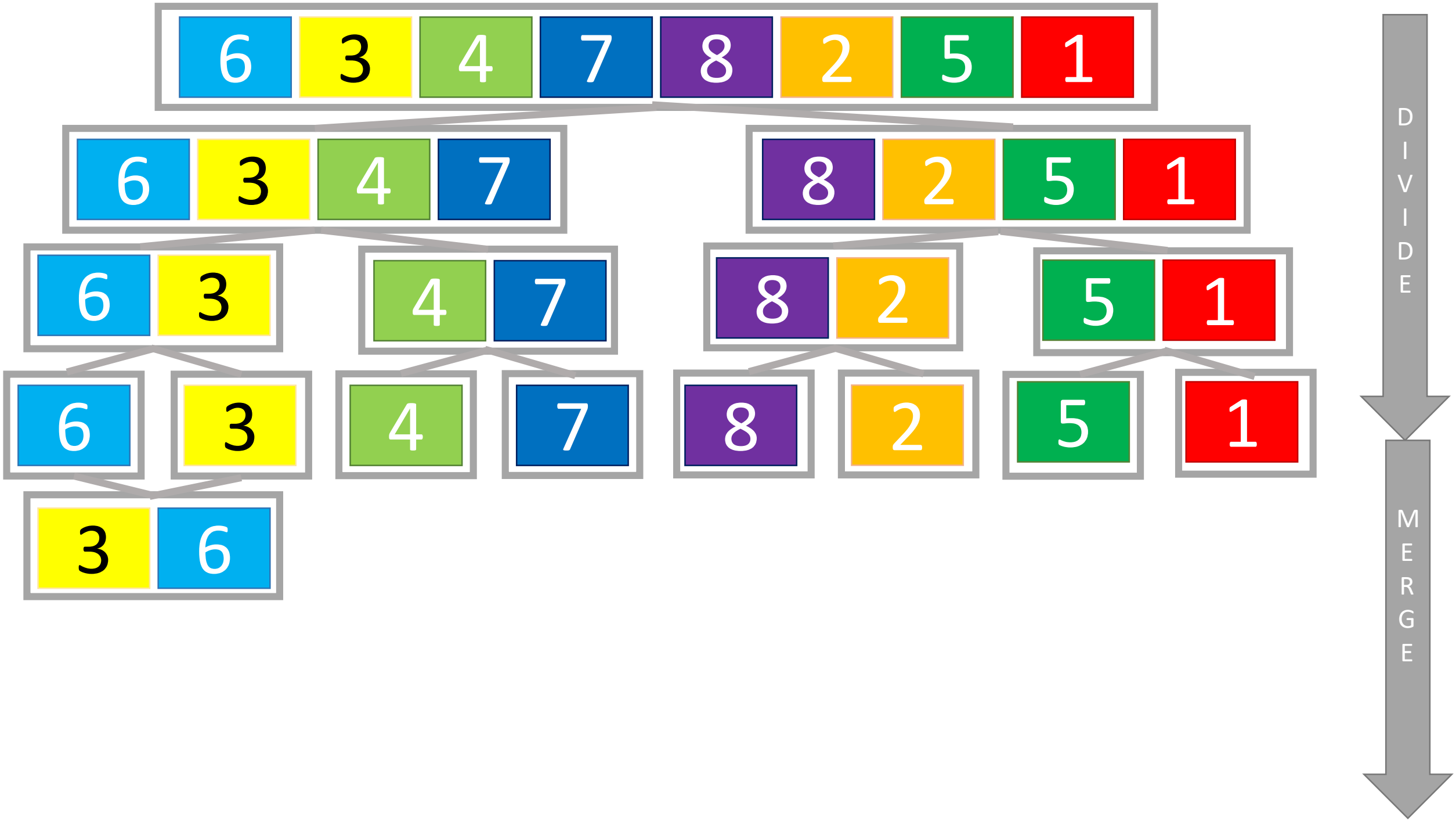

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7

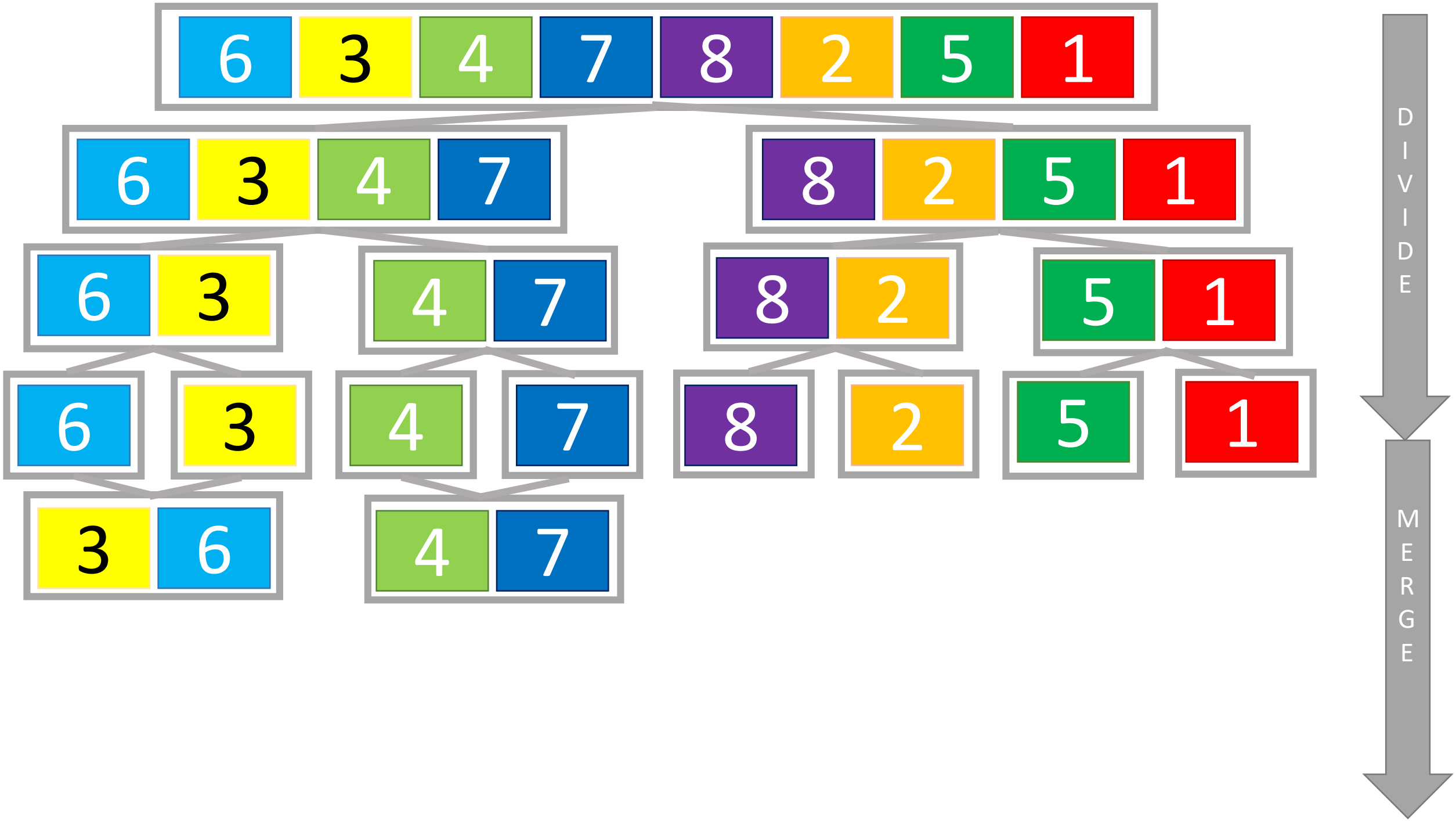

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8

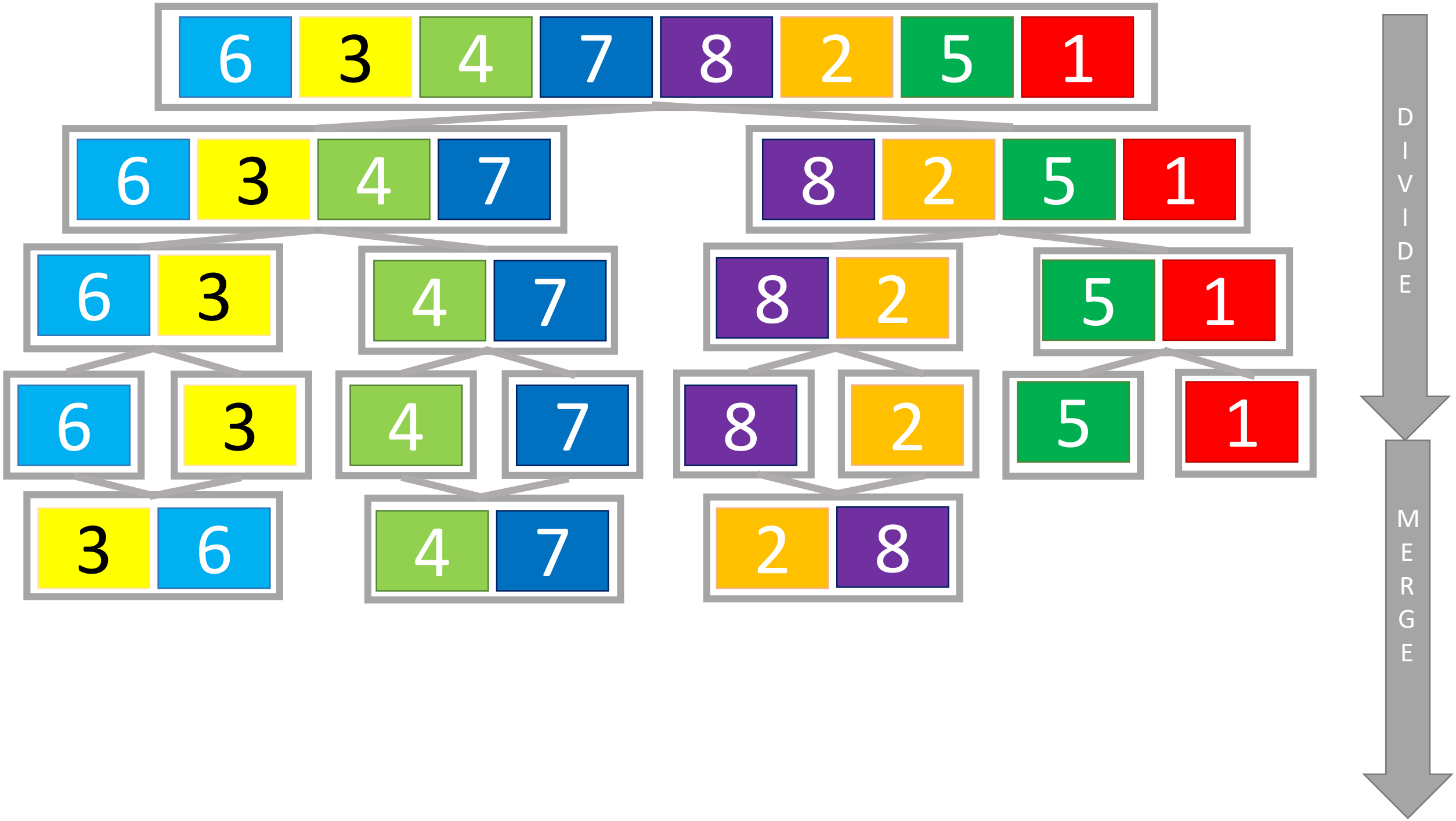

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2

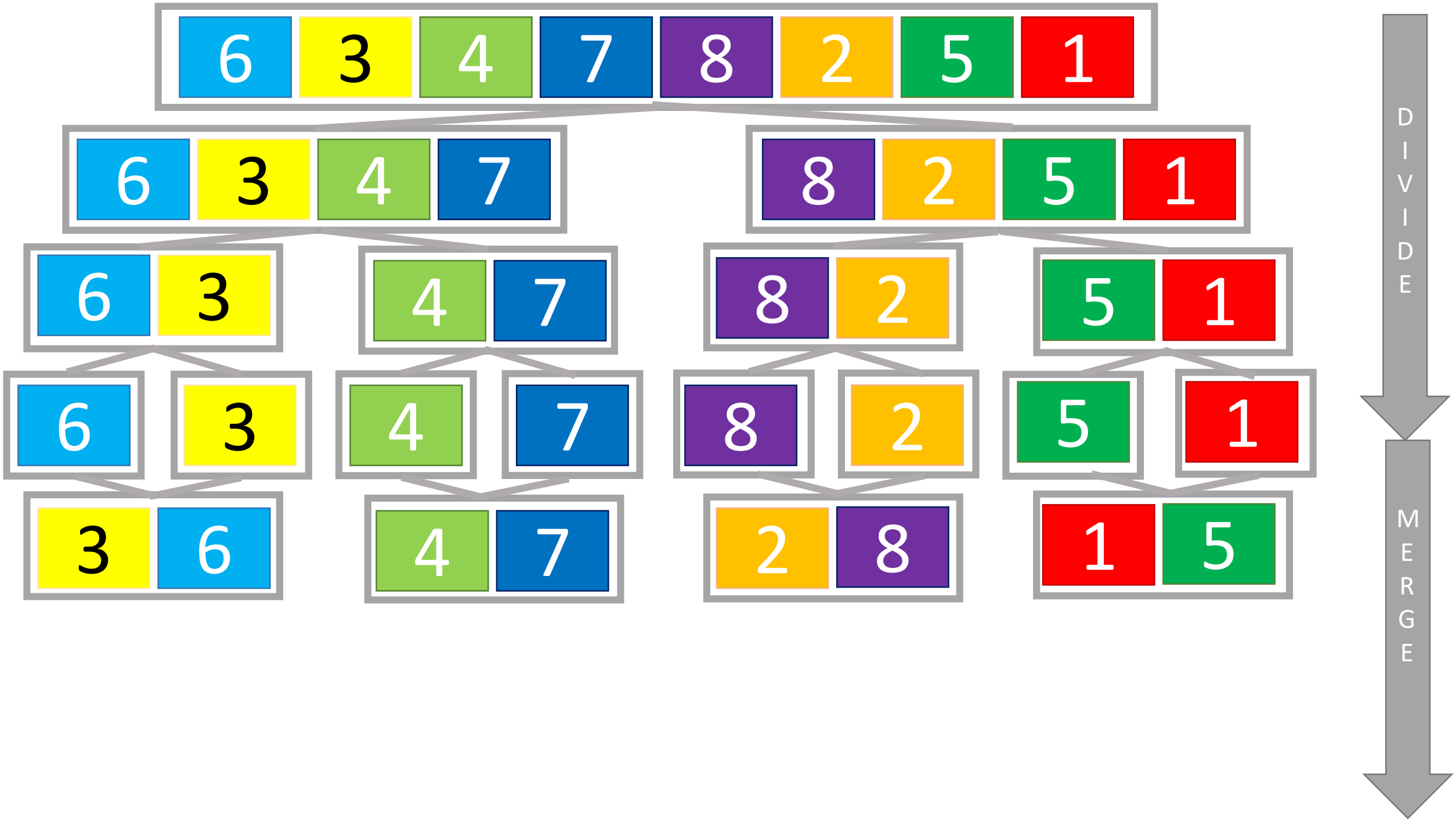

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5

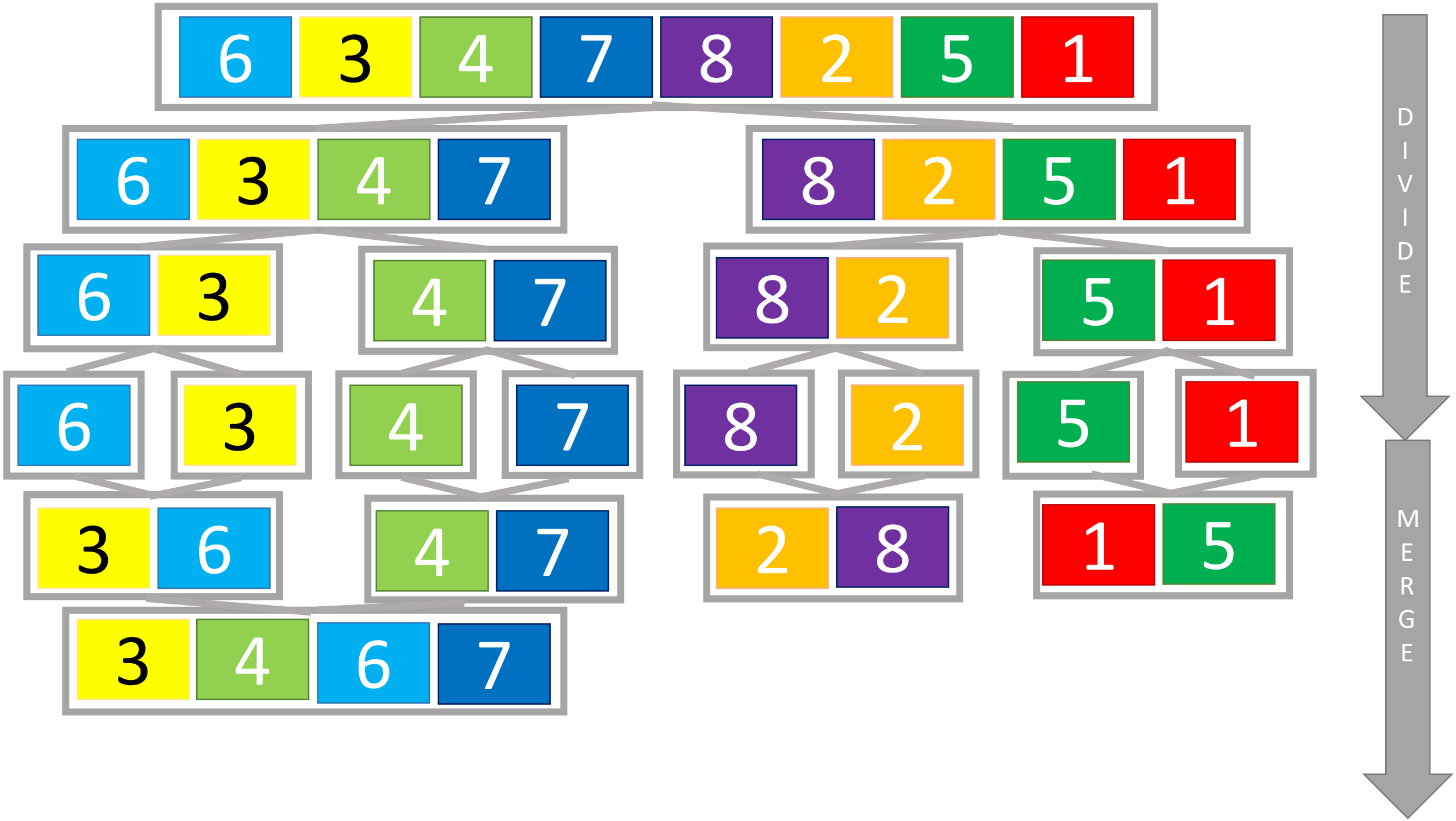

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1

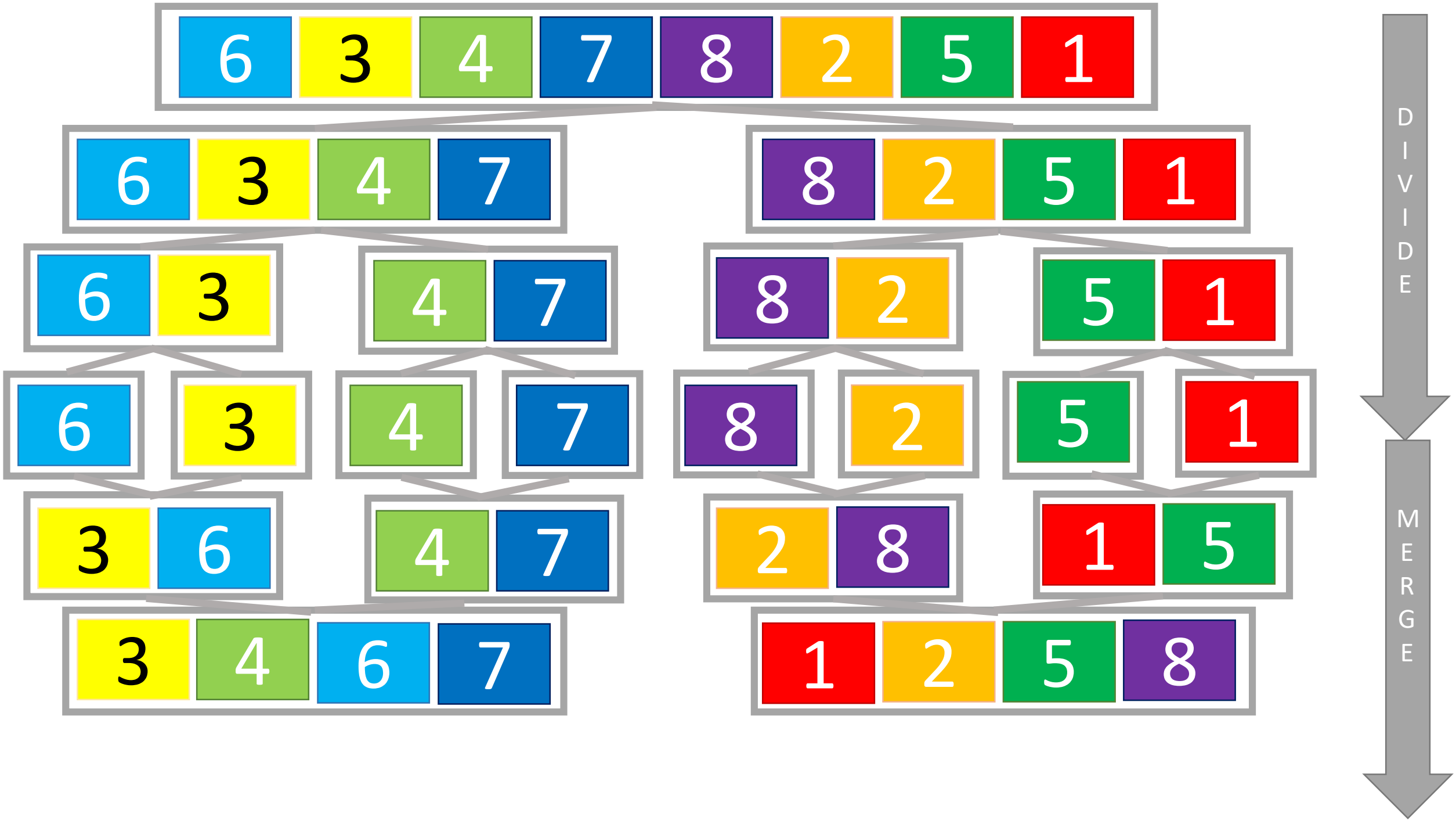

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
DIVIDE
MERGE

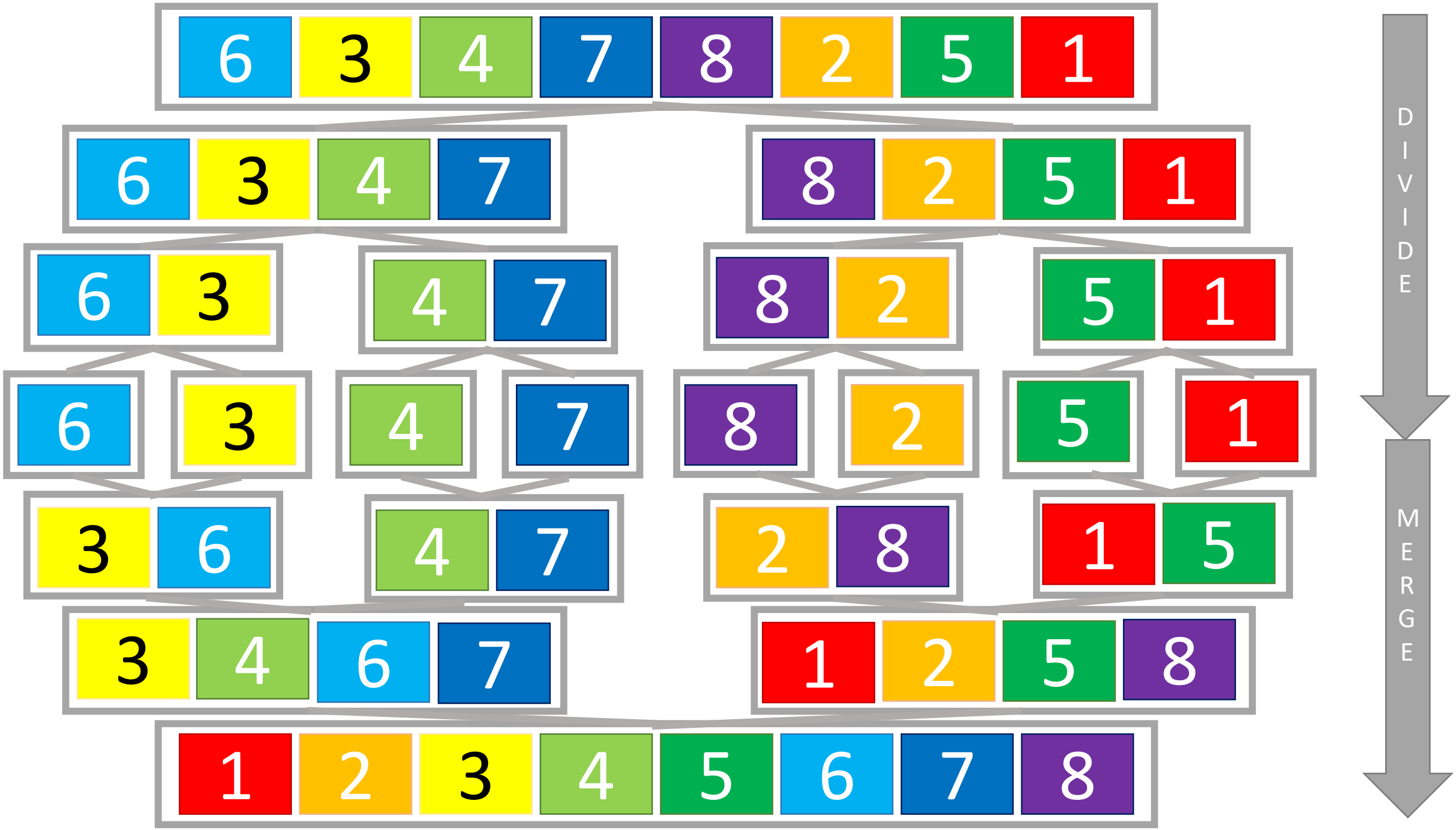

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
4 7
DIVIDE
MERGE


6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
4 7
2 8
DIVIDE
MERGE


6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
4 7
2 8
1 5
DIVIDE
MERGE

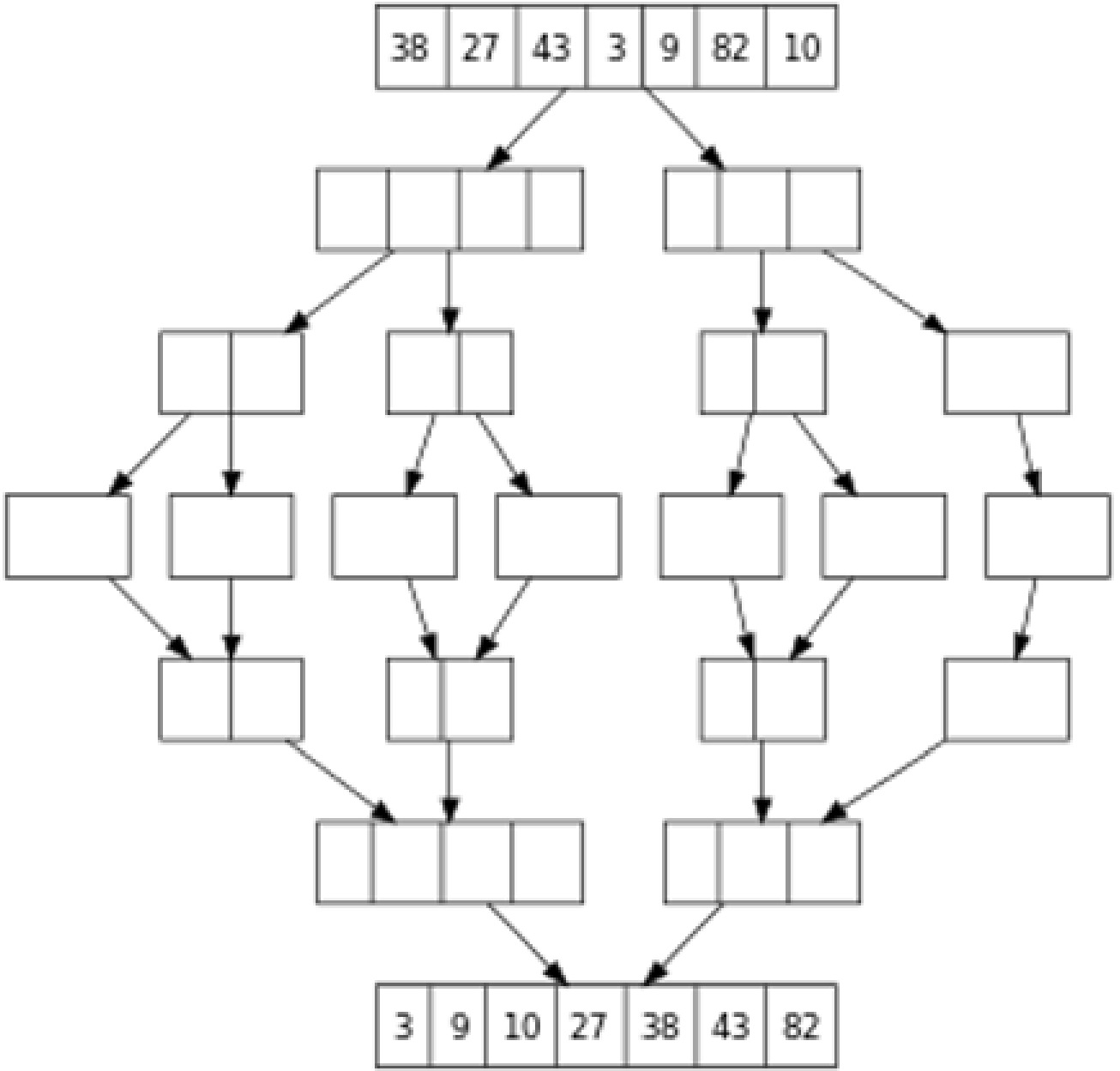

6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
4 7
2 8
1 5
3 4 6 7
DIVIDE
MERGE


6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
4 7
2 8
1 5
3 4 6 7
1 2 5 8
DIVIDE
MERGE


6 3 4 7 8 2 5 1
6 3 4 7
8 2 5 1
6 3
4 7
8 2
5 1
6
3
4
7
8
2
5
1
3 6
4 7
2 8
1 5
3 4 6 7
1 2 5 8
1 2 3 4 5 6 7 8
DIVIDE
MERGE

How do you handle uneven array sizes?

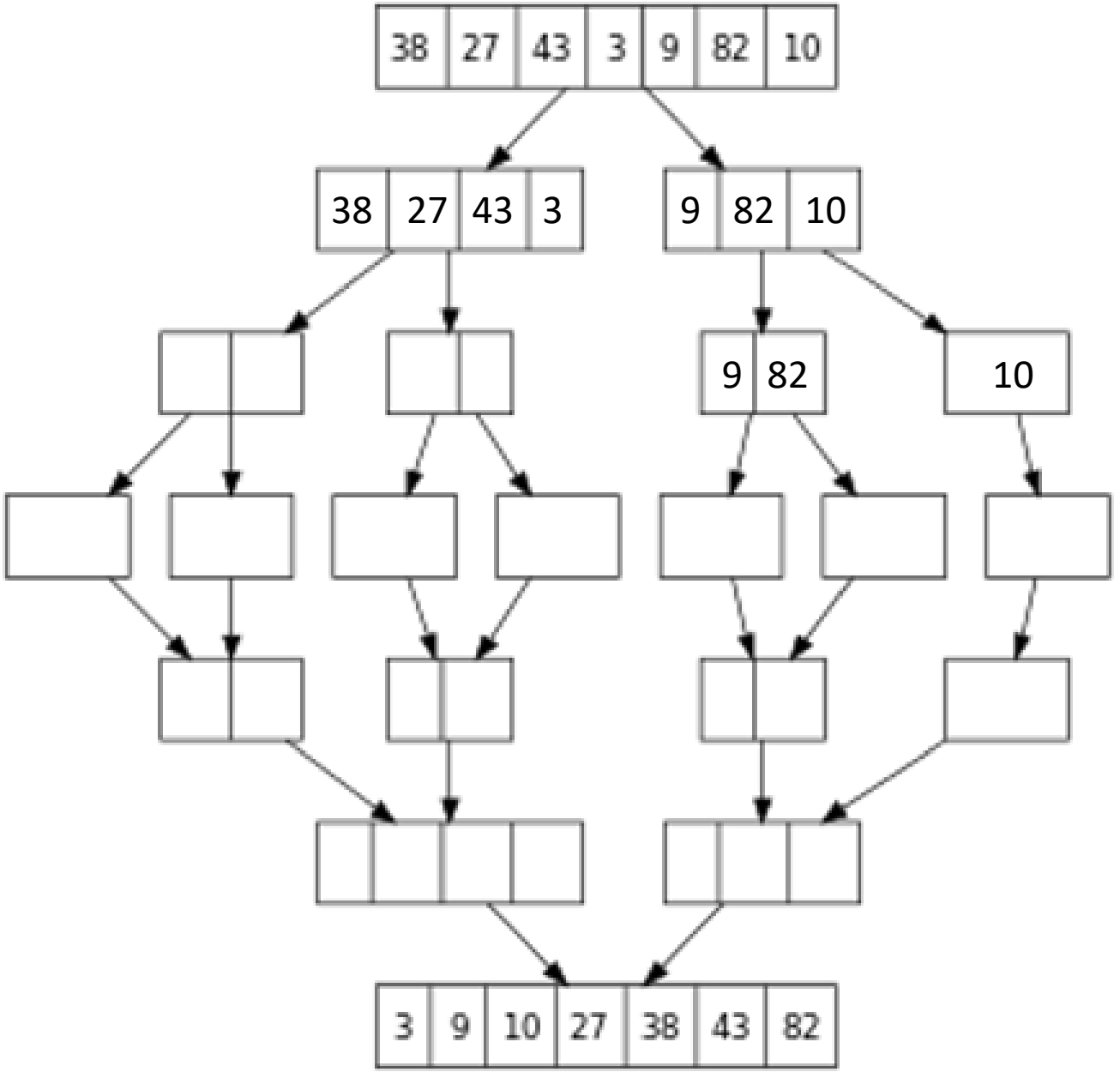
0. Fill in both tracings of mergesort:



How do you handle uneven array sizes?

Follow the path given.

0. Fill in both tracings of mergesort:



The code:

```
public int[] mergeSort(int[] array) {
 if (array.length <= 1)
    return array;
 else {
    int middle = array.length / 2;
    int firstHalf = mergeSort(array[0..middle - 1]);
    int secondHalf = mergeSort(array[middle..array.length - 1]);
    return merge(firstHalf, secondHalf);
  }
}
```

The code:

```
public int[] mergeSort(int[] array) {
 if (array.length <= 1)
    return array;
 else {
    int middle = array.length / 2;
    int firstHalf = mergeSort(array[0..middle - 1]);
    int secondHalf = mergeSort(array[middle..array.length - 1]);
    return merge(firstHalf, secondHalf);
  }
}
```

Recursive Call one

The code:

```
public int[] mergeSort(int[] array) {
 if (array.length <= 1)
    return array;
 else {
    int middle = array.length / 2;
    int firstHalf = mergeSort(array[0..middle - 1]);
    int secondHalf = mergeSort(array[middle..array.length - 1]);
    return merge(firstHalf, secondHalf);
  }
}
```

Recursive Call one

Recursive Call two

The code:

```
public int[] mergeSort(int[] array) {
 if (array.length <= 1)
    return array;
 else {
    int middle = array.length / 2;
    int firstHalf = mergeSort(array[0..middle - 1]);
    int secondHalf = mergeSort(array[middle..array.length - 1]);
    return merge(firstHalf, secondHalf);
  }
}
```

Recursive Call one

Recursive Call two

Merge the two sorted arrays

A reminder of a previous card:

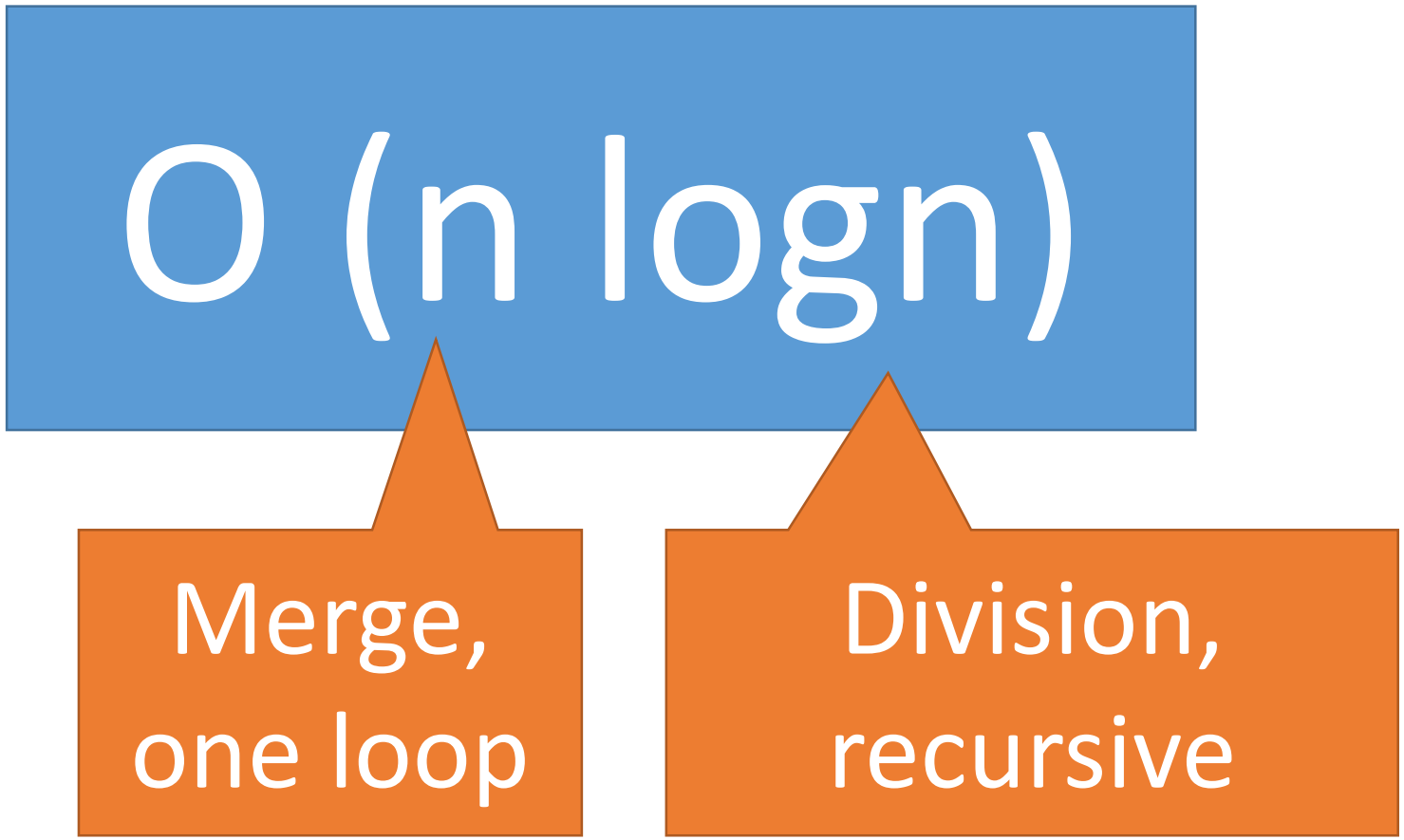
Algorithm Speeds

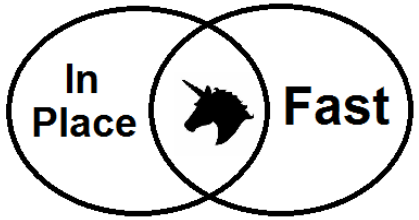
- O(1) – swap two values, size of a array
- O(log n) – binary search
- O(n) – max, min, average, print, linear search
- O(n log n) – merge sort, quick sort
- O(n^2) – bubble sort, selection sort
- O(n!) – Bogo sort

Merge sort’s speed

$O(n \log n)$

Merge sort’s speed

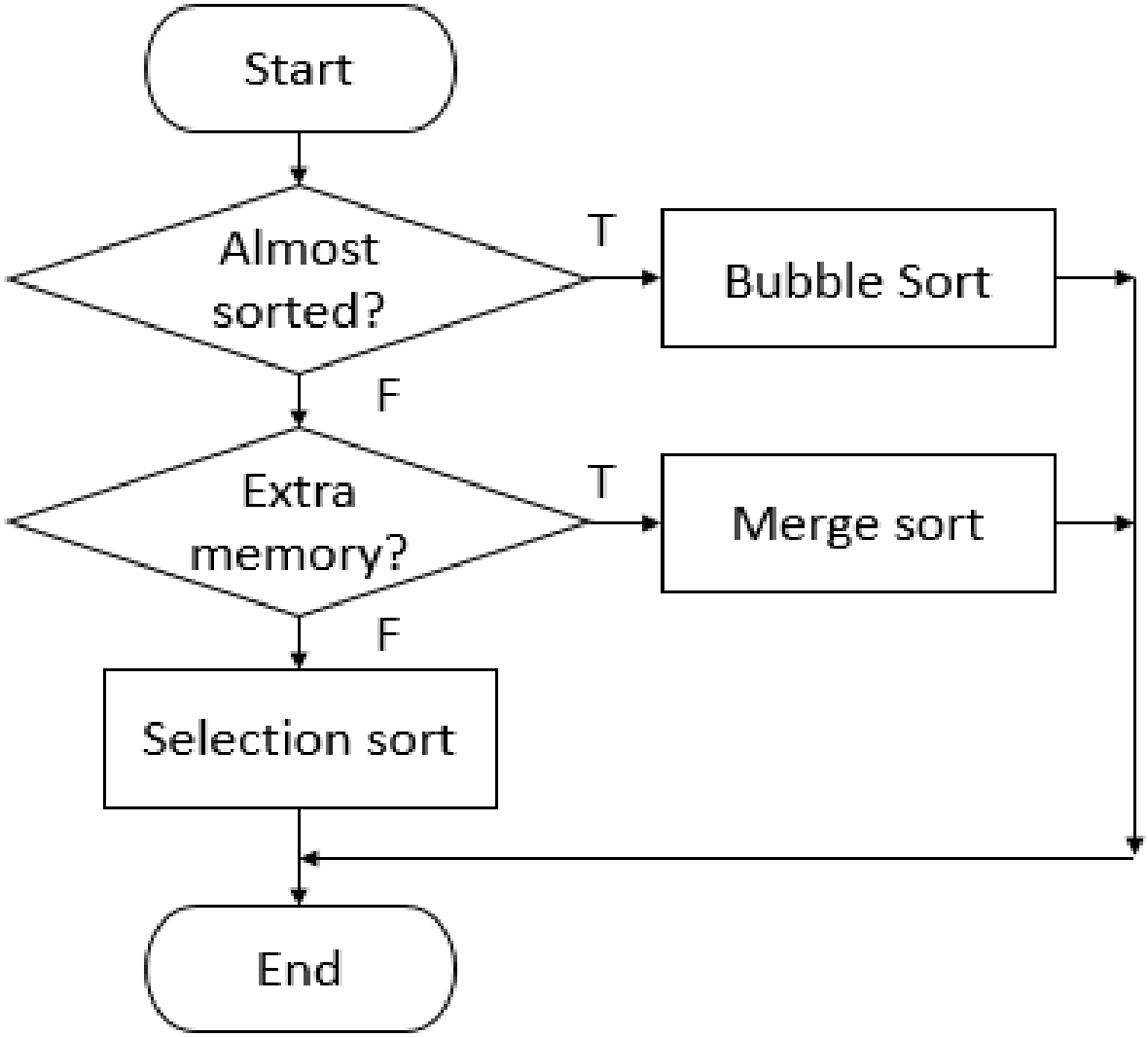




Mergesort Characteristics

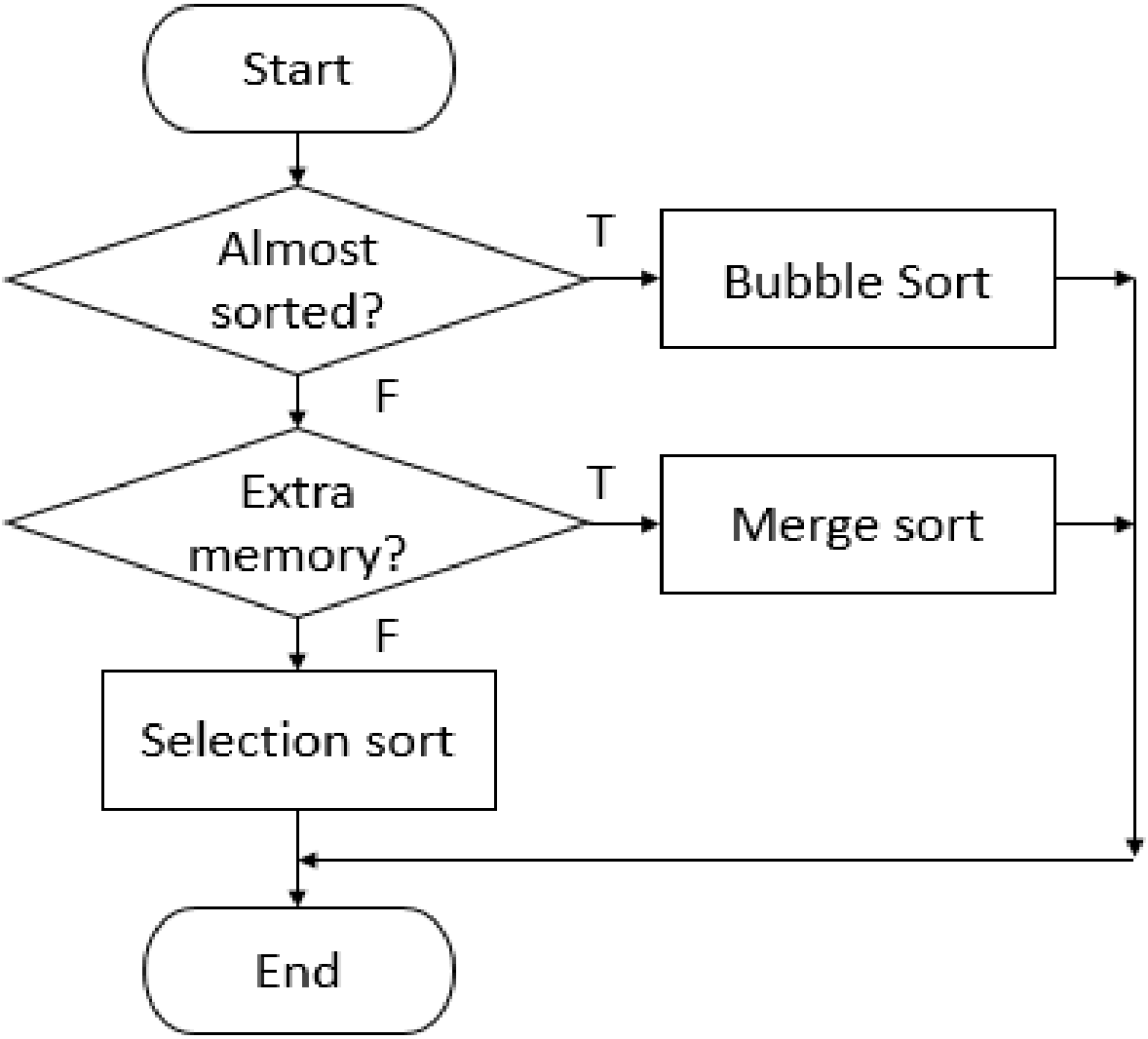
- Algorithm: First, divide the array recursively until you reach the base case: an array of one element. Second, repeatedly merge the array together.
- Speed: n log n. Slightly slower than Quicksort.
- Mergesort is not an in-place algorithm. It does not use swaps to move elements around. Additional memory is needed for the recursive calls.
- Trade-off: Mergesort gains its incredible speed by using extra memory.

How do you choose which algorithm to use?

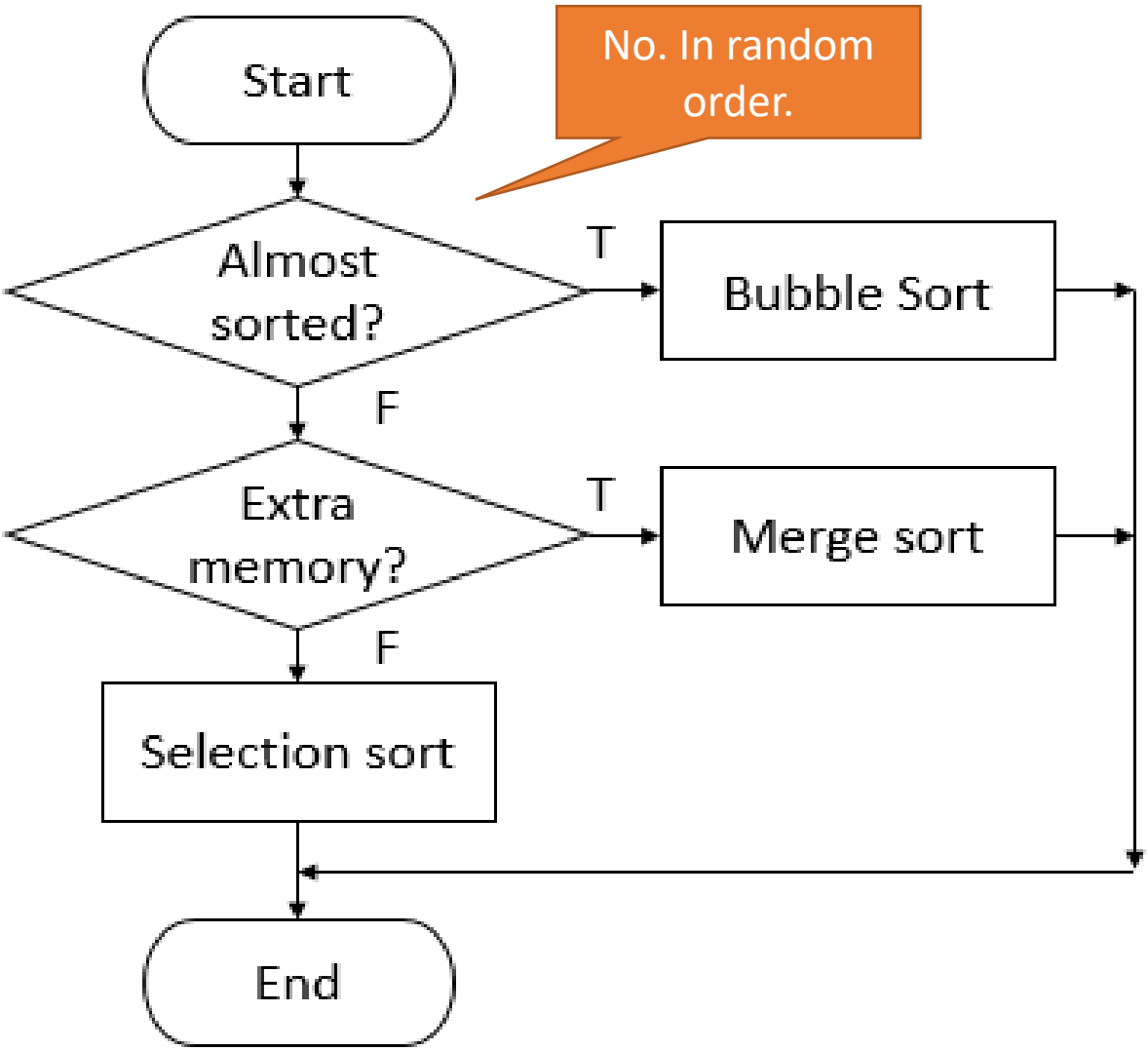


How do you choose which algorithm to use?

The array is 200 elements long (not very big), and it is in completely random order.



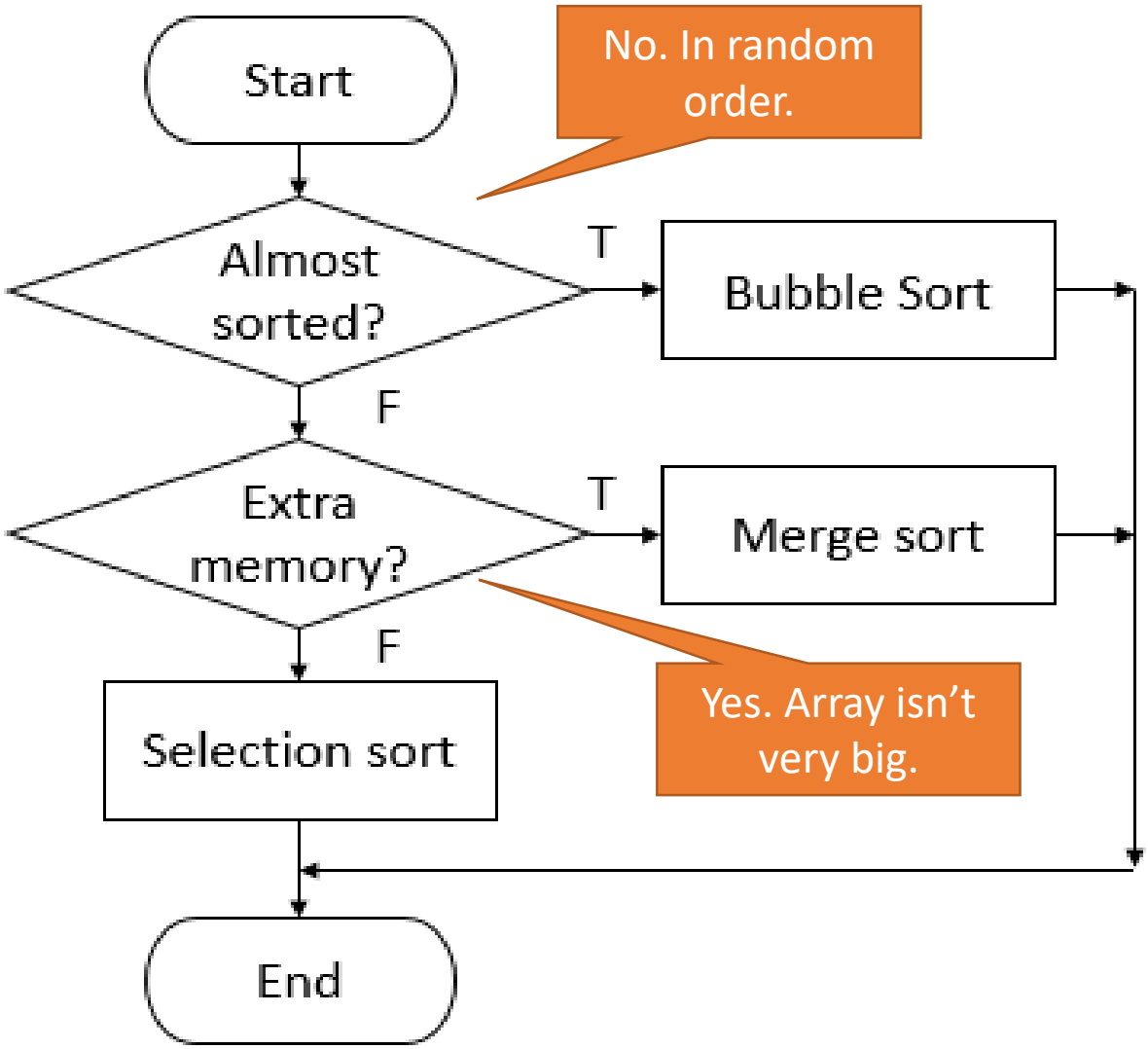
How do you choose which algorithm to use?



The array is 200 elements long (not very big), and it is in completely random order.

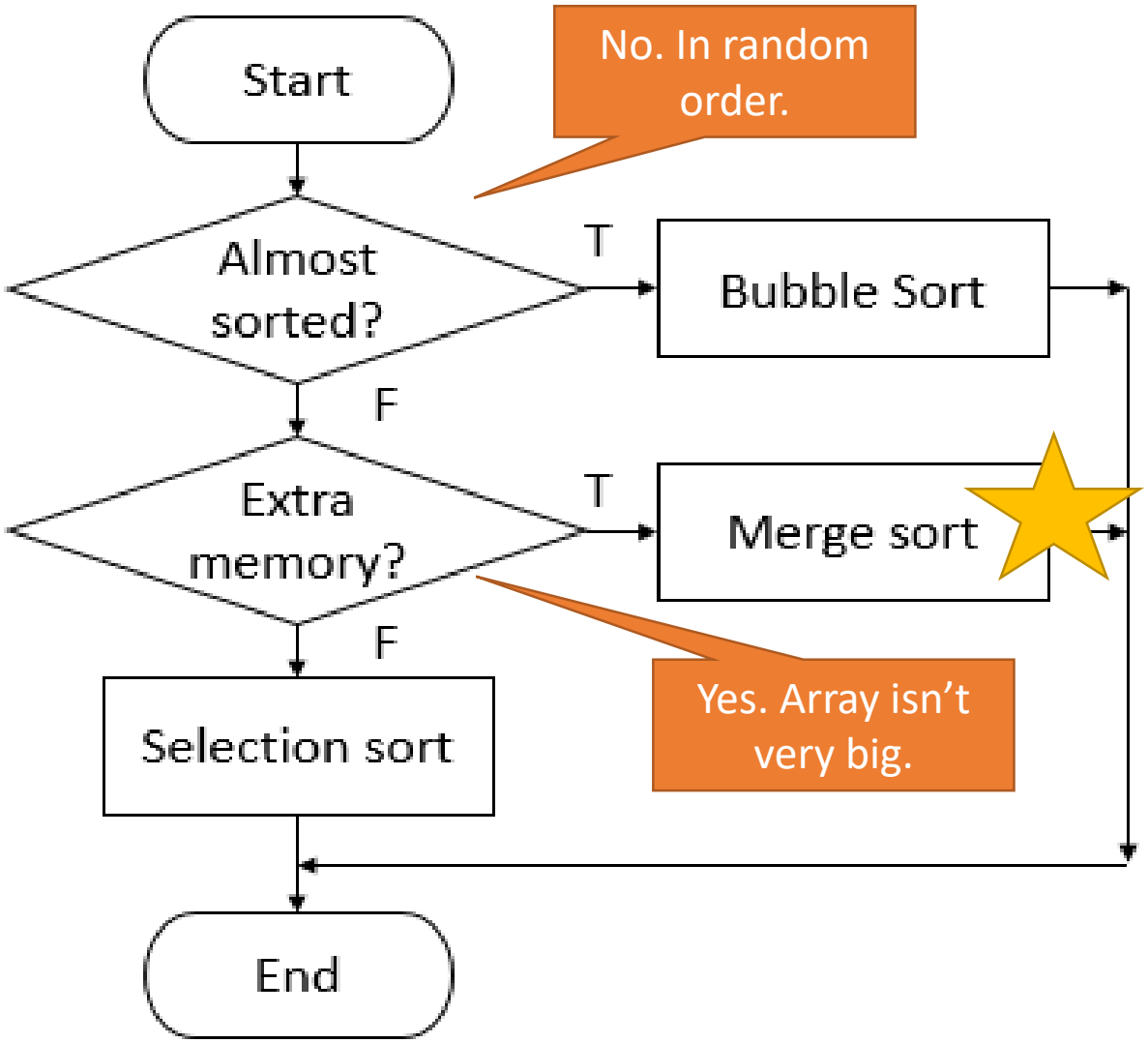
How do you choose which algorithm to use?

The array is 200 elements long (not very big), and it is in completely random order.



How do you choose which algorithm to use?

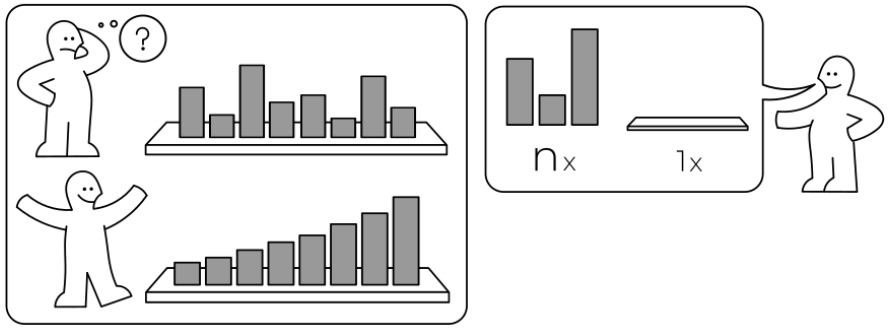
The array is 200 elements long (not very big), and it is in completely random order.



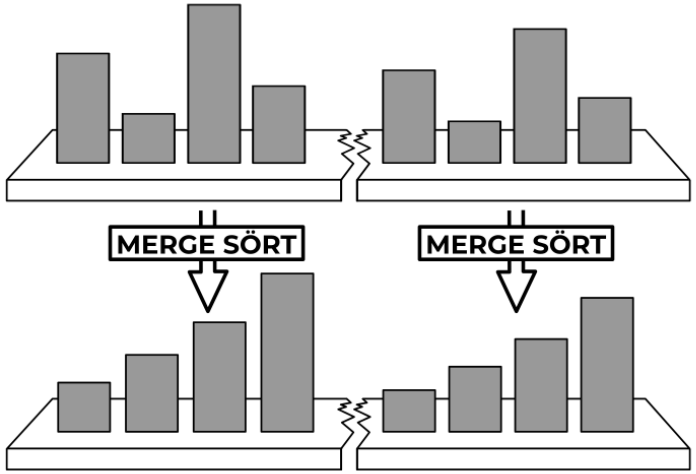
MERGE SÖRT

idea-instructions.com/merge-sort/
v1.2, CC by-nc-sa 4.0

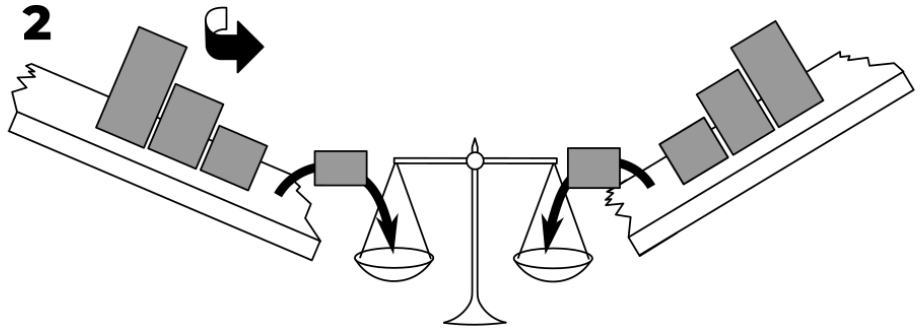
IDEA



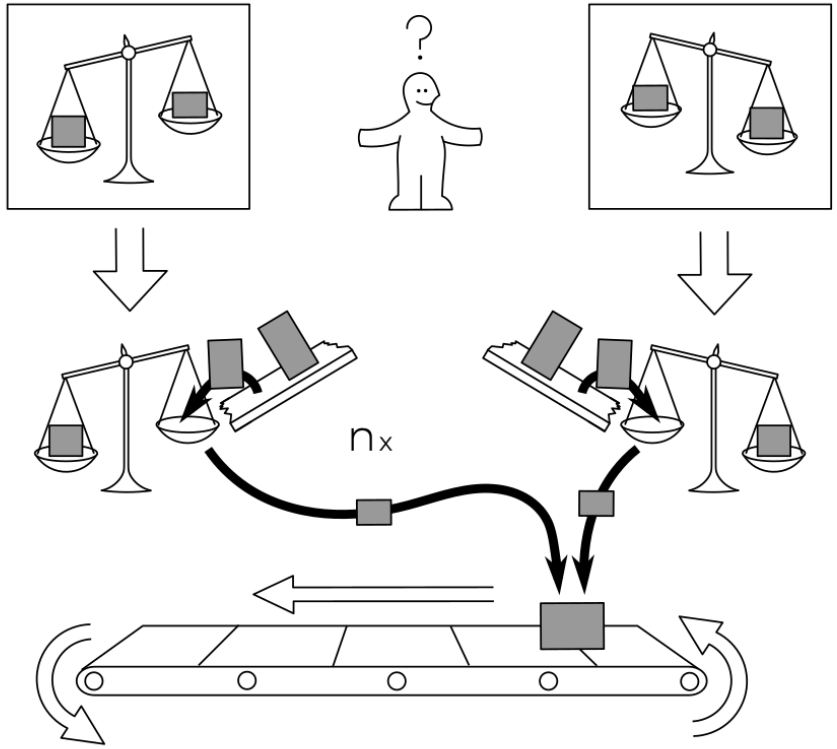
1



2



3



4