ADVANCED DATA STRUCTURES AND ALGORITHMS

Associate Professor Dr. Raed Ibraheem Hamed

University of Human Development,
College of Science and Technology
Computer Science Department

2015 – 2016

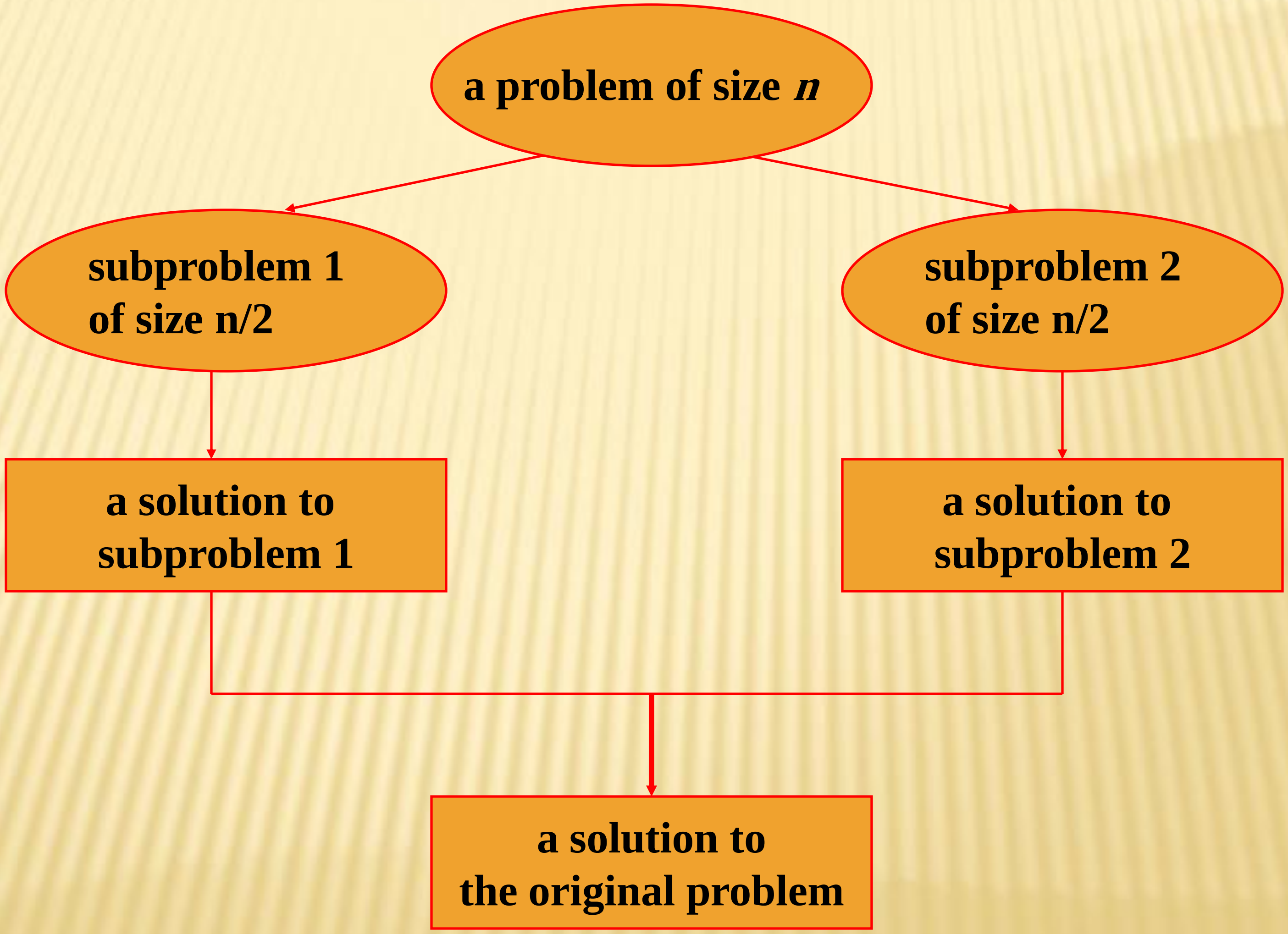
Divide and Conquer

- **Recursive in structure**
  - ***Divide*** the problem into sub-problems that are similar to the original but smaller in size.
  - ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - ***Combine*** the solutions to create a solution to the original problem

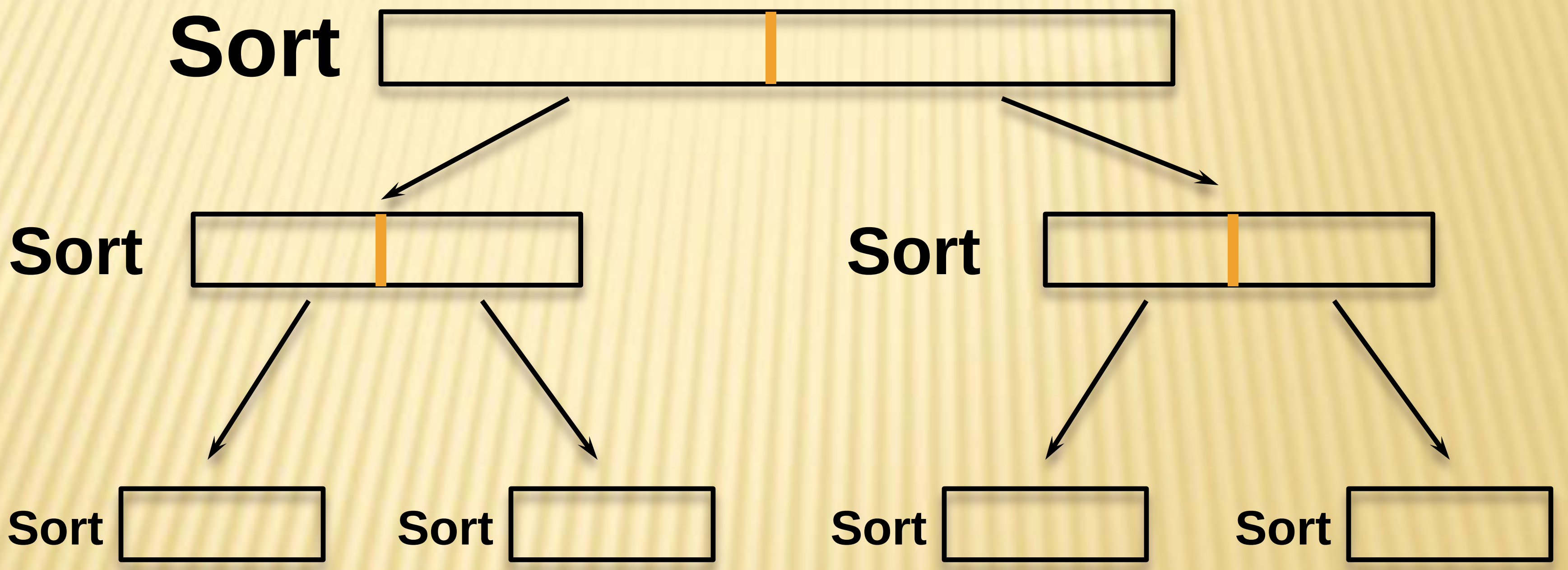
"Divide and Conquer"

- Very important strategy in computer science:
  - Divide problem into smaller parts
  - Independently solve the parts
  - Combine these solutions to get overall solution
- **Idea 1**: Divide array into two halves, *recursively* sort left and right halves, then *merge* two halves → **Mergesort**
- **Idea 2 :** Partition array into items that are "small" and items that are "large", then recursively sort the two sets → **Quicksort**

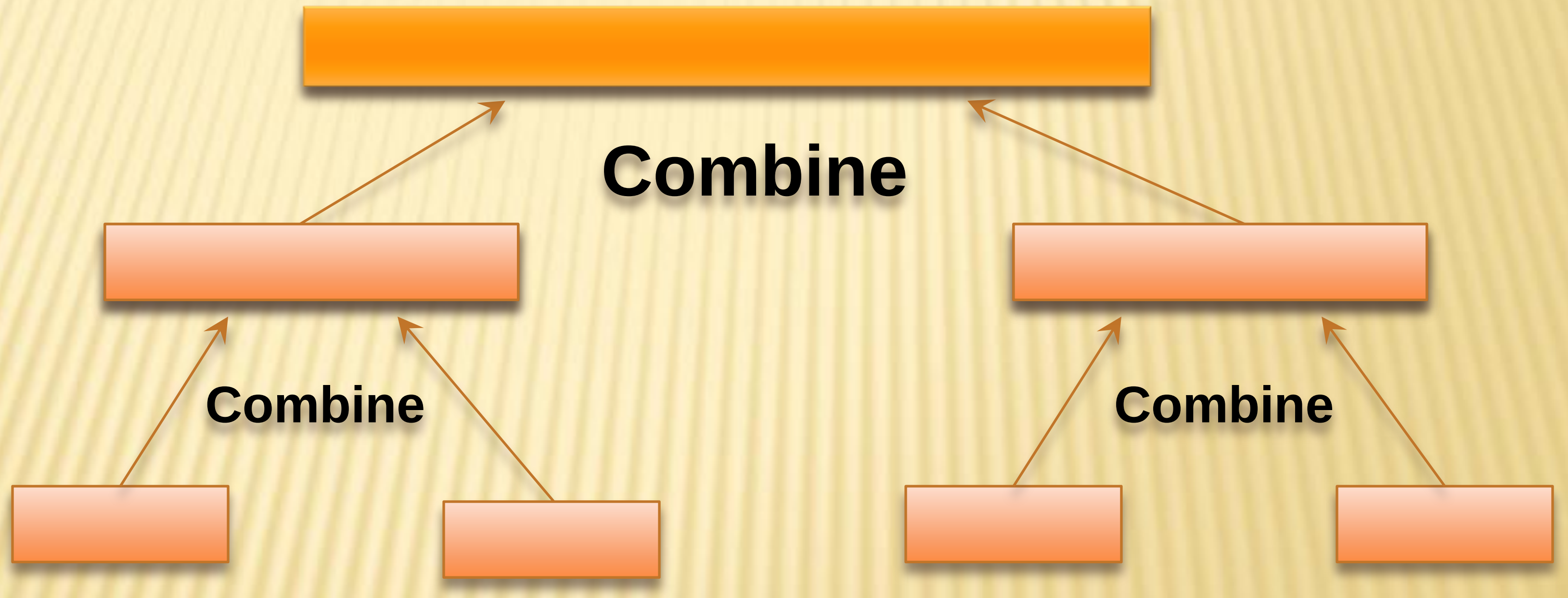
Divide-and-conquer Technique


a problem of size *n*
subproblem 1 of size n/2
subproblem 2 of size n/2
a solution to subproblem 1
a solution to subproblem 2
a solution to the original problem

Divide and Conquer



Divide and Conquer



Divide and Conquer

```
module sort(array)
{
   if (size of array > 1)
   {
      split(array, firstPart, secondPart)
      sort(firstPart)
      sort(secondPart)
      combine(firstPart, secondPart)
   }
}
```

Divide and Conquer

```
module sort(array)
{
   if (size of array > 1)
   {
      split(array, firstPart, secondPart)
      sort(firstPart)
      sort(secondPart)
      combine(firstPart, secondPart)
   }
}
```

Divide and Conquer

```
module sort(array)
{
   if (size of array > 1)
   {
      split(array, firstPart, secondPart)
      sort(firstPart)
      sort(secondPart)
      combine(firstPart, secondPart)
   }
}
```

Divide and Conquer

```
module sort(array)
{
   if (size of array > 1)
   {
      split(array, firstPart, secondPart)
      sort(firstPart)
      sort(secondPart)
      combine(firstPart, secondPart)
   }
}
```

Divide and Conquer

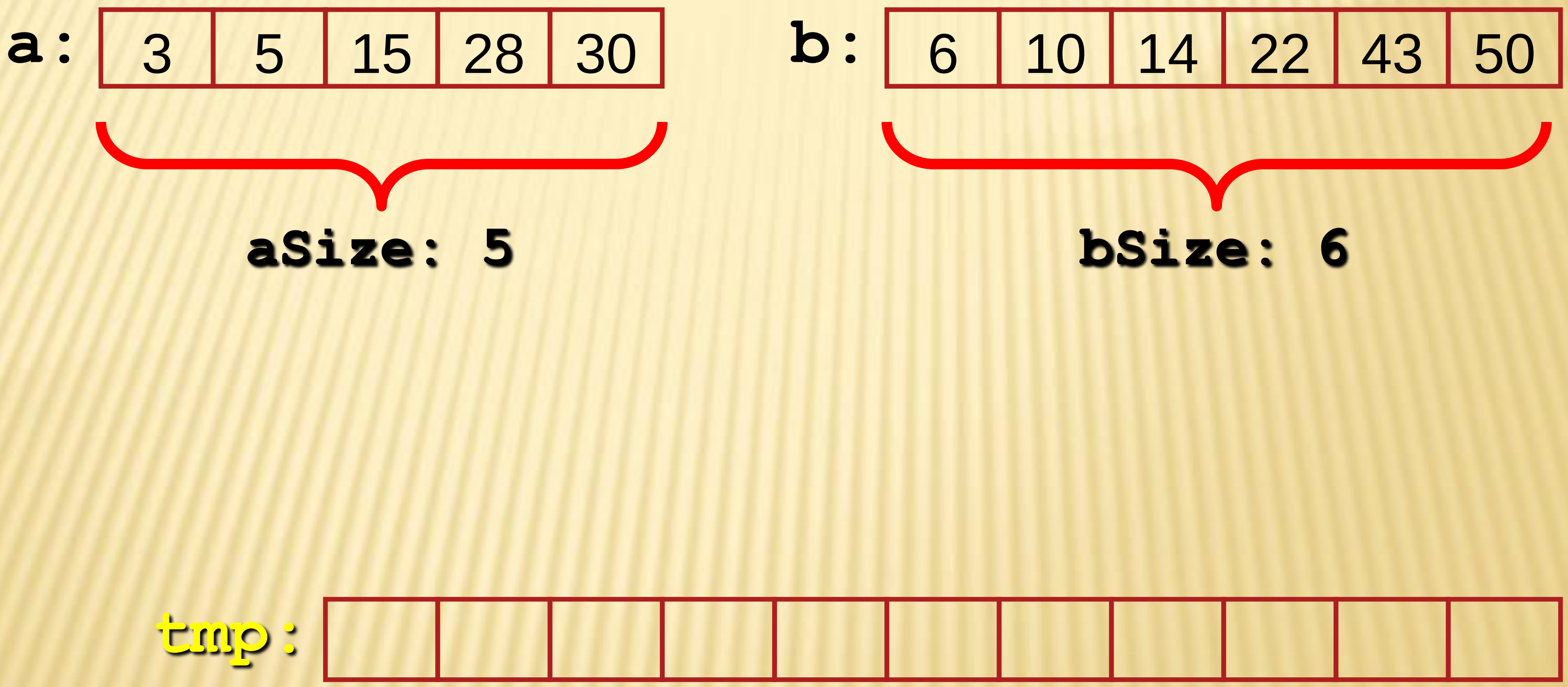
```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

Algorithm to merge sorted arrays

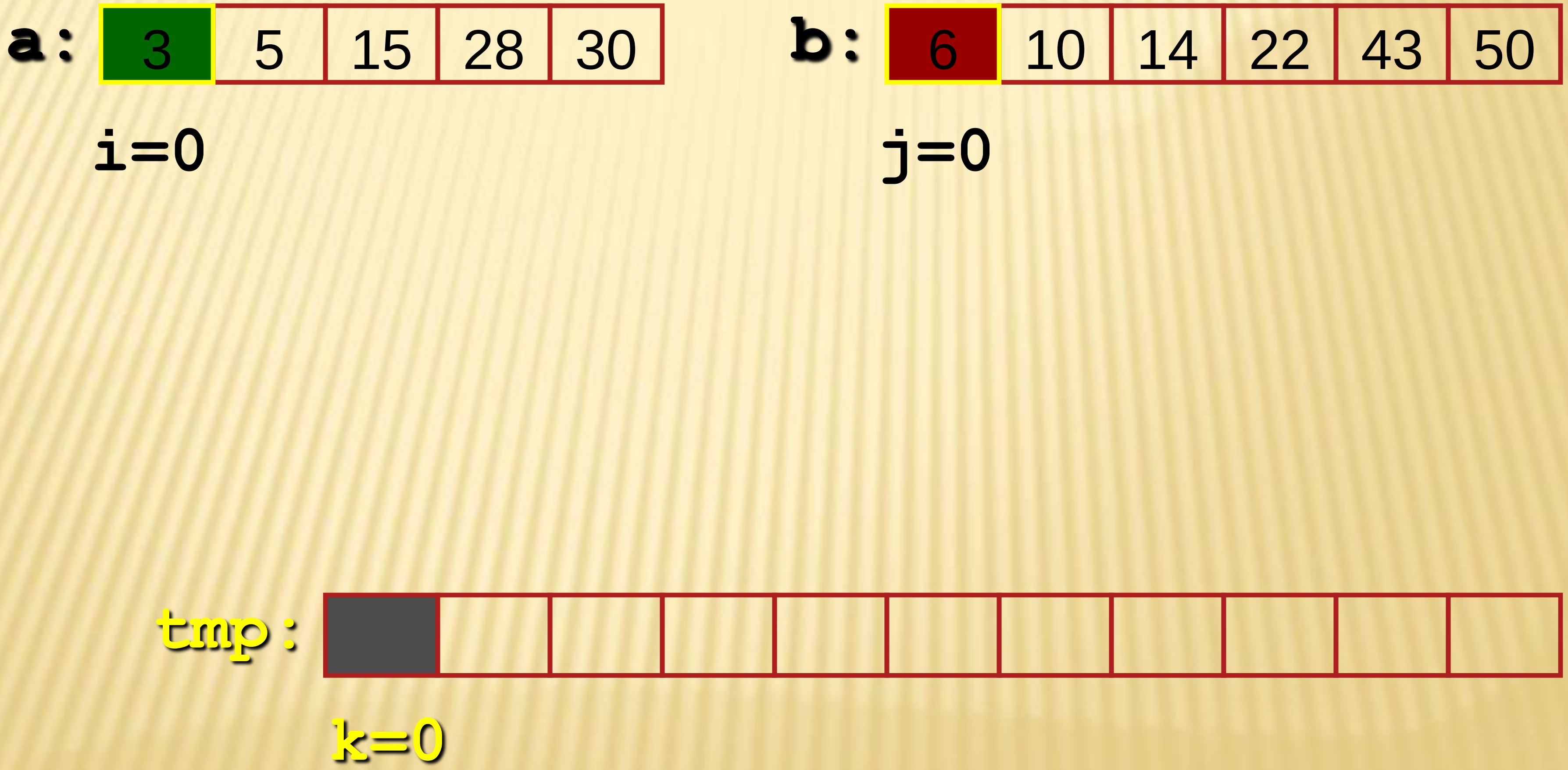
Merge algorithm
Assume, that both arrays are sorted in ascending order and we want resulting array to maintain the same order. Algorithm to merge two arrays A[0..m-1] and B[0..n-1] into an array C[0..m+n-1] is as following:

1) Introduce read-indices **i**, **j** to traverse arrays A and B, accordingly. Introduce write-index **k** to store position of the first free cell in the resulting array. By default **i** = **j** = **k** = 0.
2) At each step: if both indices are in range (**i** < m and **j** < n), choose minimum of (A[**i**], B[**j**]) and write it to C[**k**]. Otherwise go to step 4.
3) Increase **k** and index of the array, algorithm located minimal value at, by one. Repeat step 2.
4) Copy the rest values from the array, which index is still in range, to the resulting array.

Example: Mergearrays



Example: Mergearrays



Example: Mergearrays

a:

3	5	15	28	30

`i=0`

b:

6	10	14	22	43	50

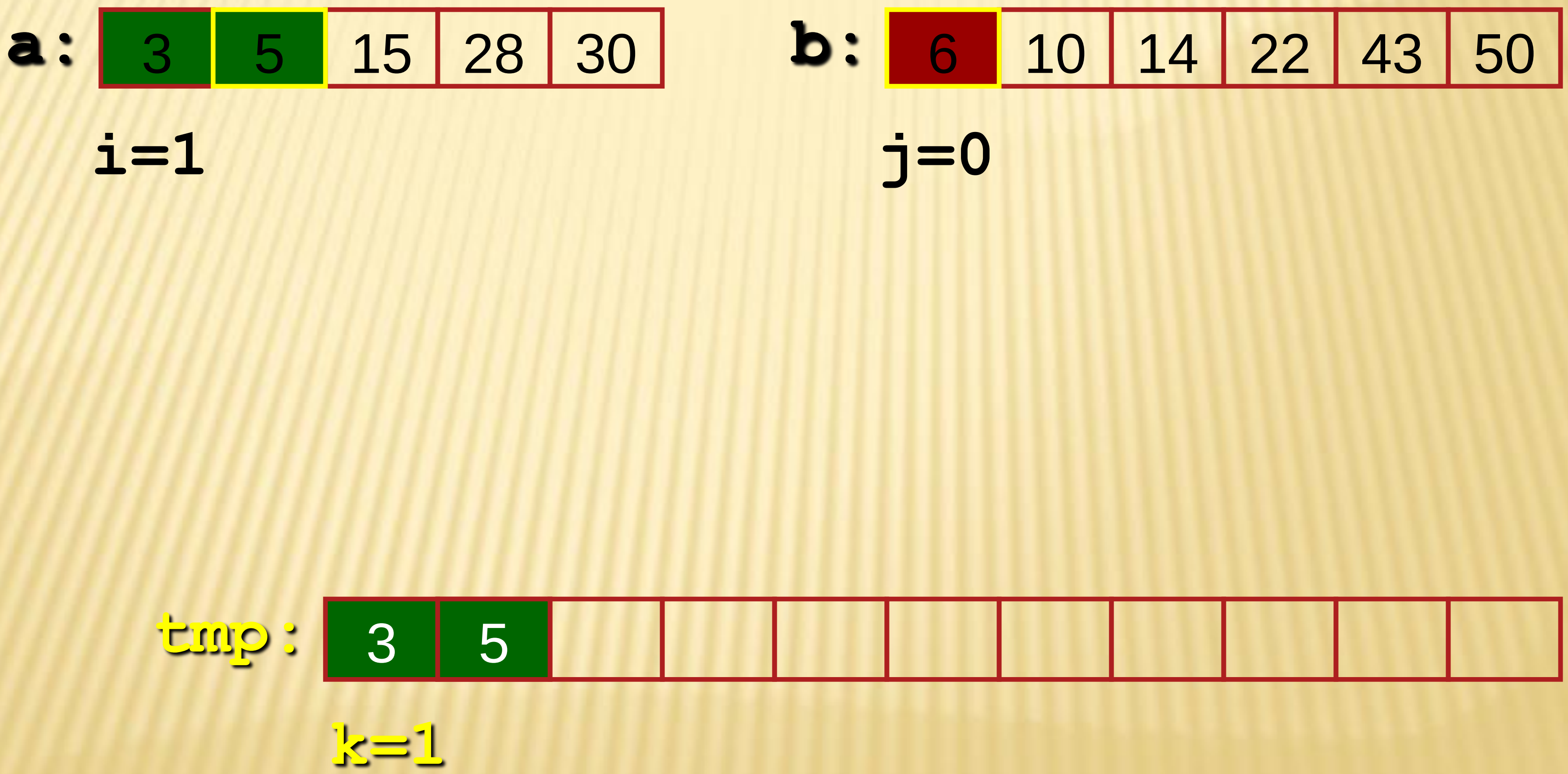
`j=0`

tmp:

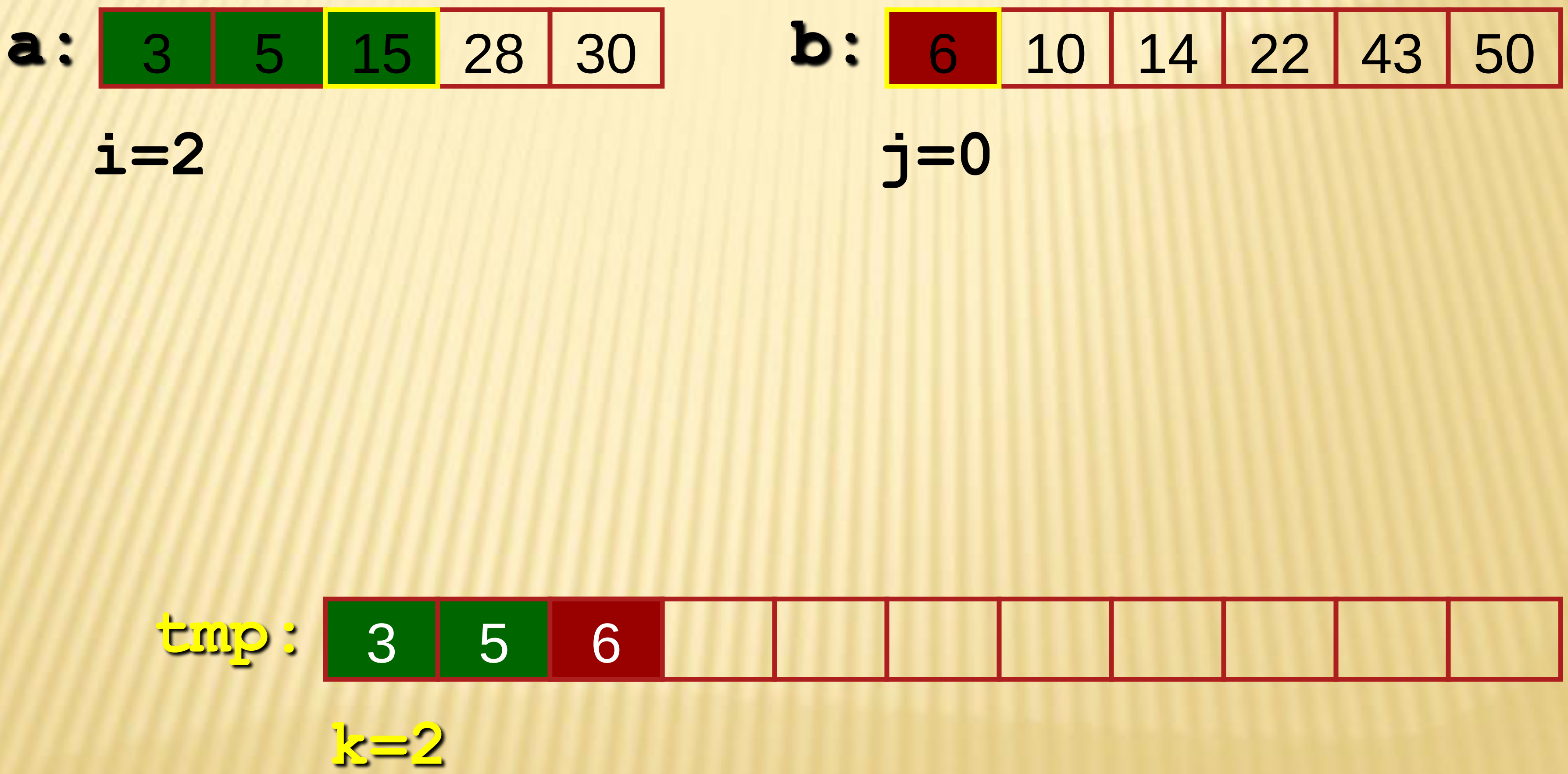
3										

`k=0`

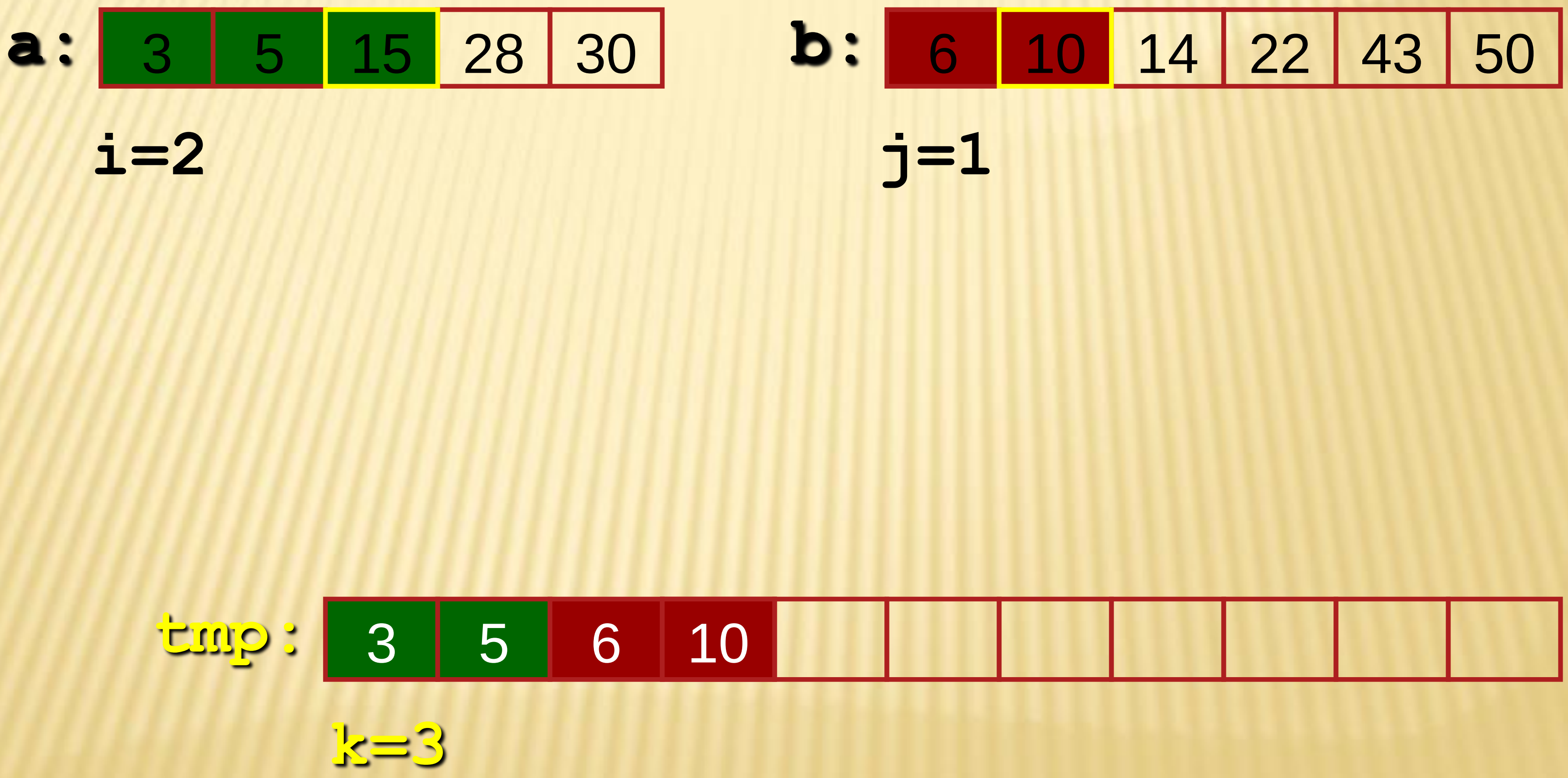
Example: Mergearrays



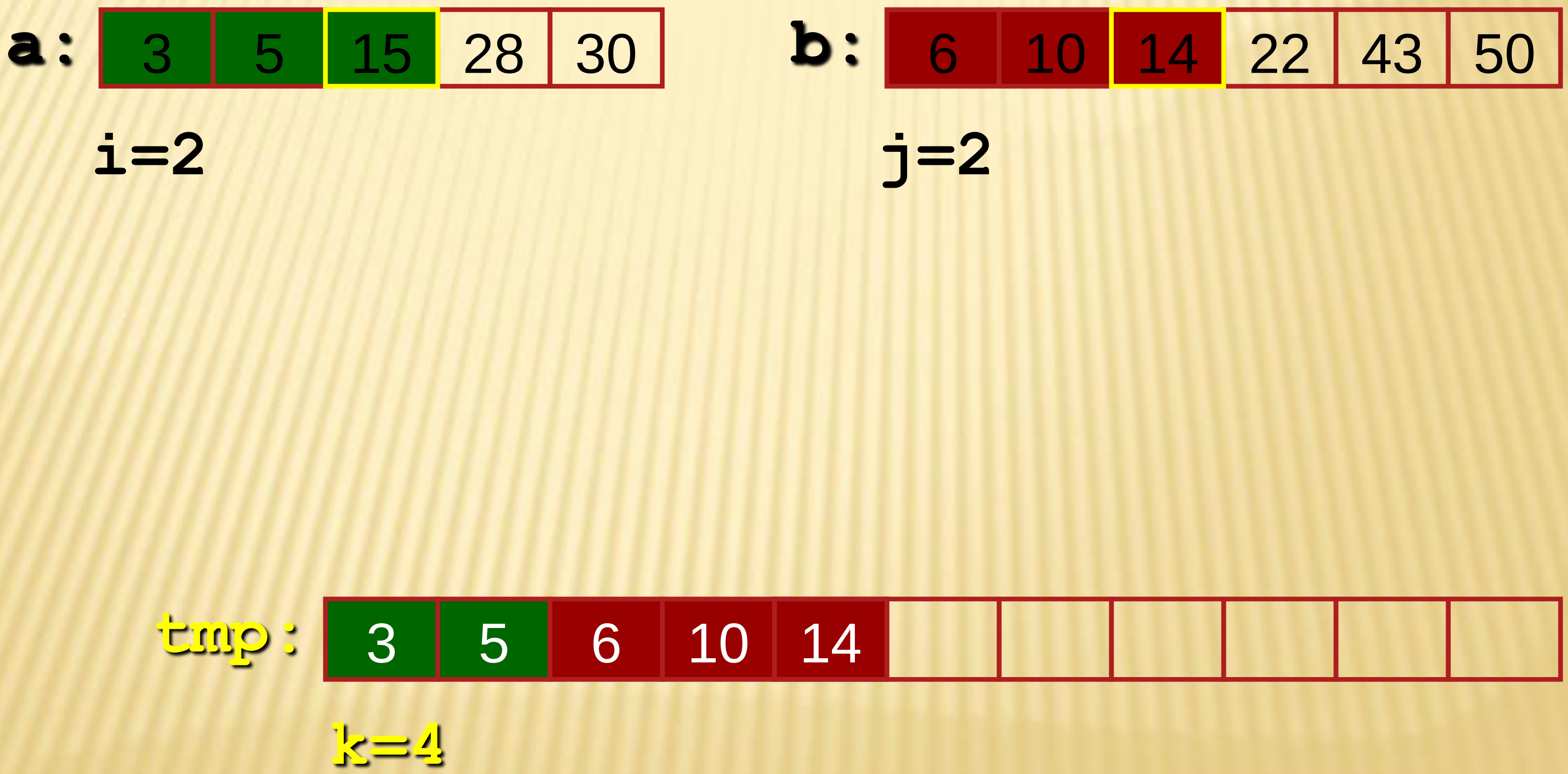
Example: Mergearrays



Example: Mergearrays



Example: Mergearrays



Example: Mergearrays

a:

3	5	15	28	30

i=2

b:

6	10	14	22	43	50

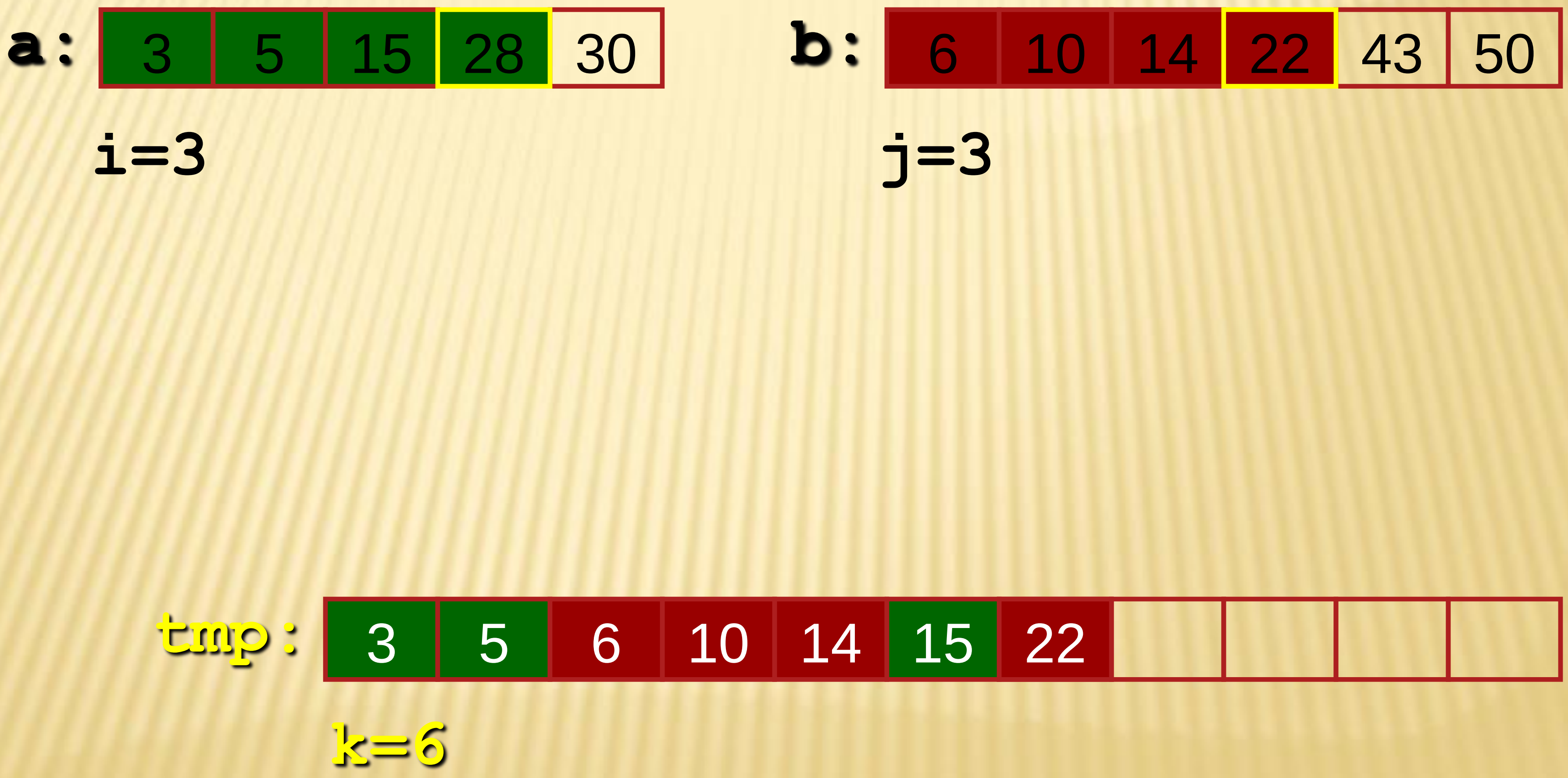
j=3

tmp:

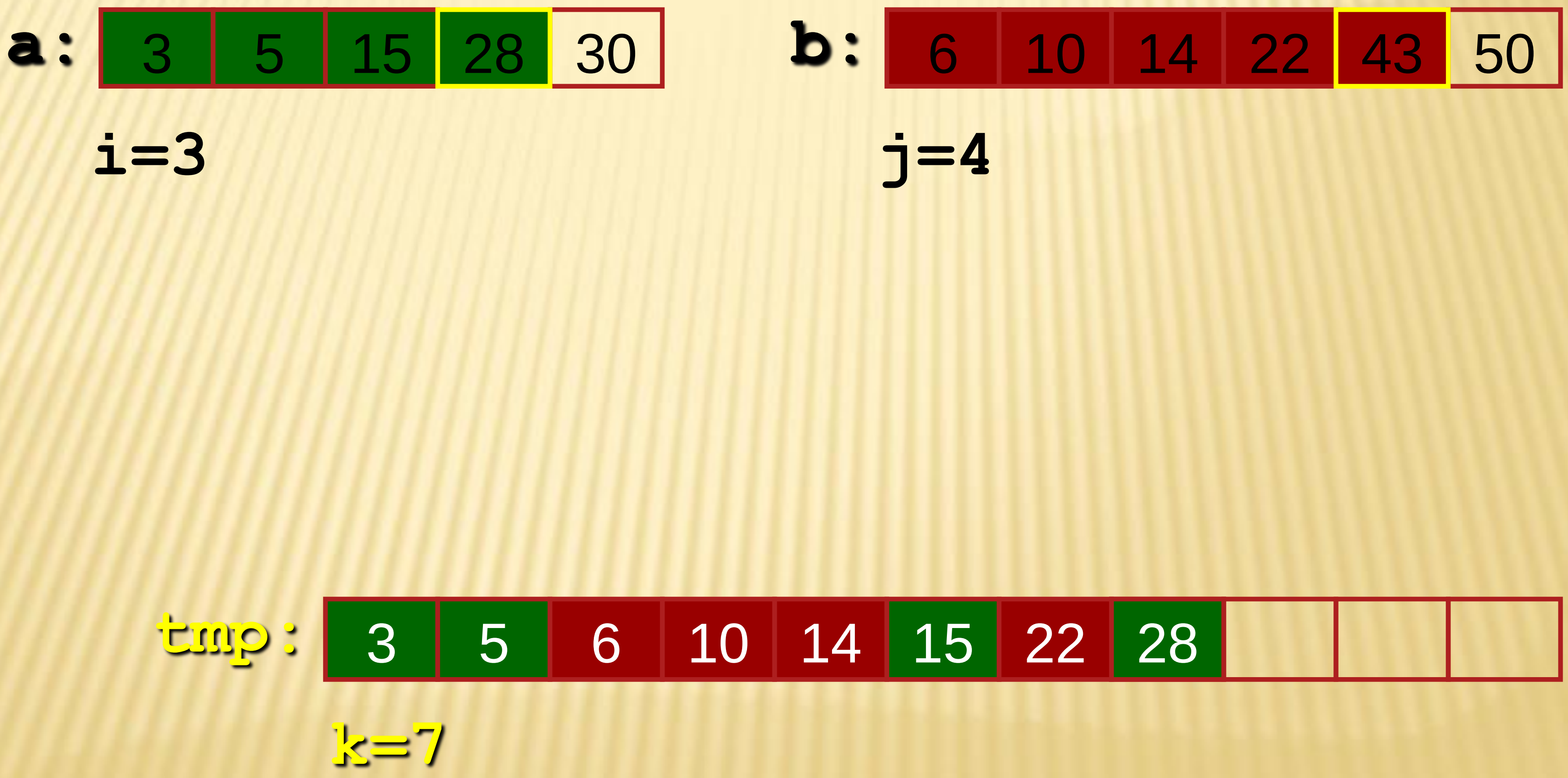
3	5	6	10	14	15					

k=5

Example: Mergearrays



Example: Mergearrays



Example: Mergearrays

a:

3	5	15	28	30

i=4

b:

6	10	14	22	43	50

j=4

tmp:

3	5	6	10	14	15	22	28	30		

k=8

Example: Mergearrays

a:

3	5	15	28	30

i=5

b:

6	10	14	22	43	50

j=4

Done.

tmp:

3	5	6	10	14	15	22	28	30	43	50

k=9

Algorithm of Mergearrays

```
mergeArrays(float a[],int aSize,float b[],int bSize,float tmp[])
{
    int   k, i = 0, j = 0;

    for (k = 0; k < aSize + bSize; k++)
    {
        if (i == aSize) {
           tmp[k] = b[j];
           j++;
        }
        else if (j == bSize) {
           tmp[k] = a[i];
           i++;
        }
        else if (a[i] <= b[j]) {
           tmp[k] = a[i];
           i++;
        }
        else {
           tmp[k] = b[j];
           j++;
        }
    }     }
```

Merge sort is based on the divide-and-conquer

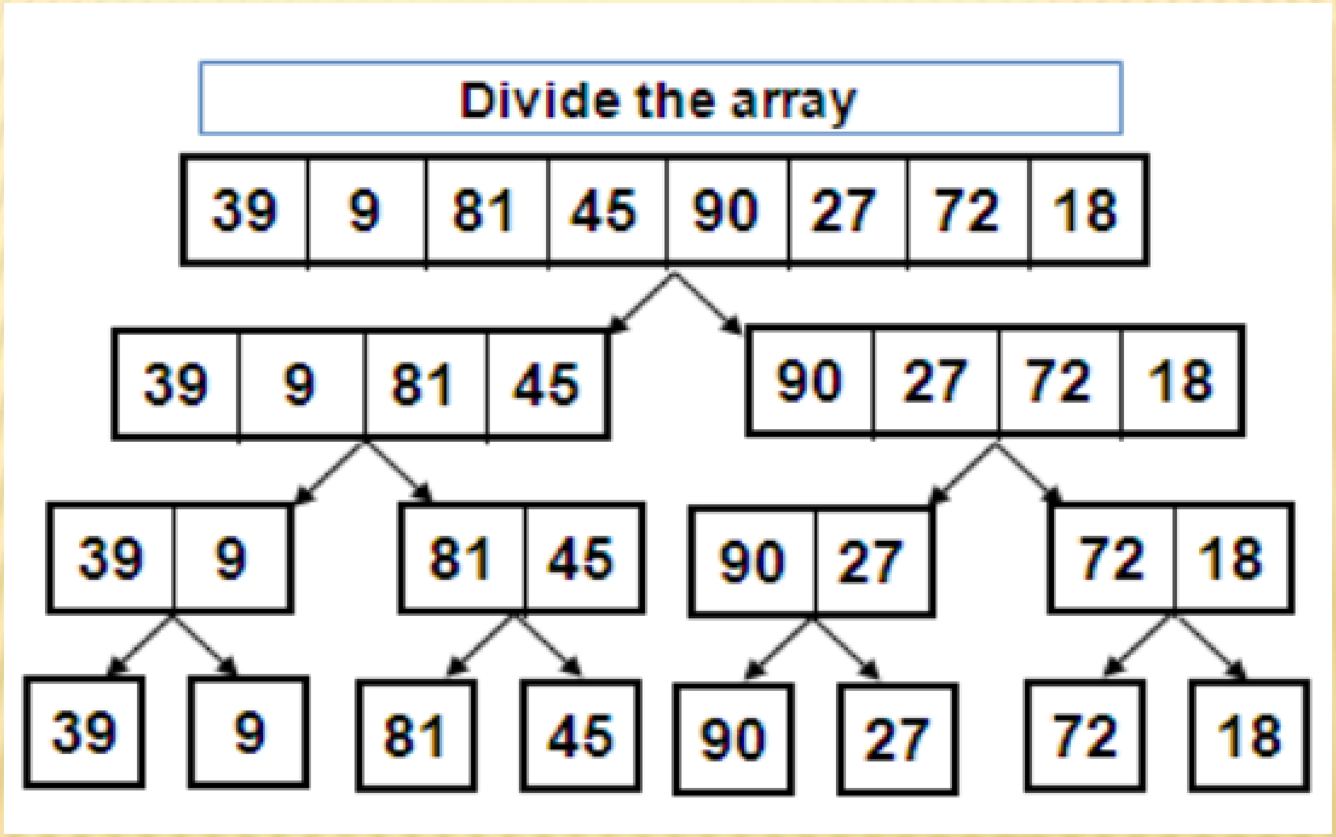
1. Divide Step If a given array A has zero or one element, simply return; **it is already sorted**. Otherwise, split A[p .. r] into two sub-arrays A[p .. q] and A[q + 1 .. r], each containing about half of the elements of A[p .. r]. That is, q is the halfway point of A[p .. r].

2. Conquer Step Conquer by recursively sorting the two sub-arrays A[p .. q] and A[q + 1 .. r].

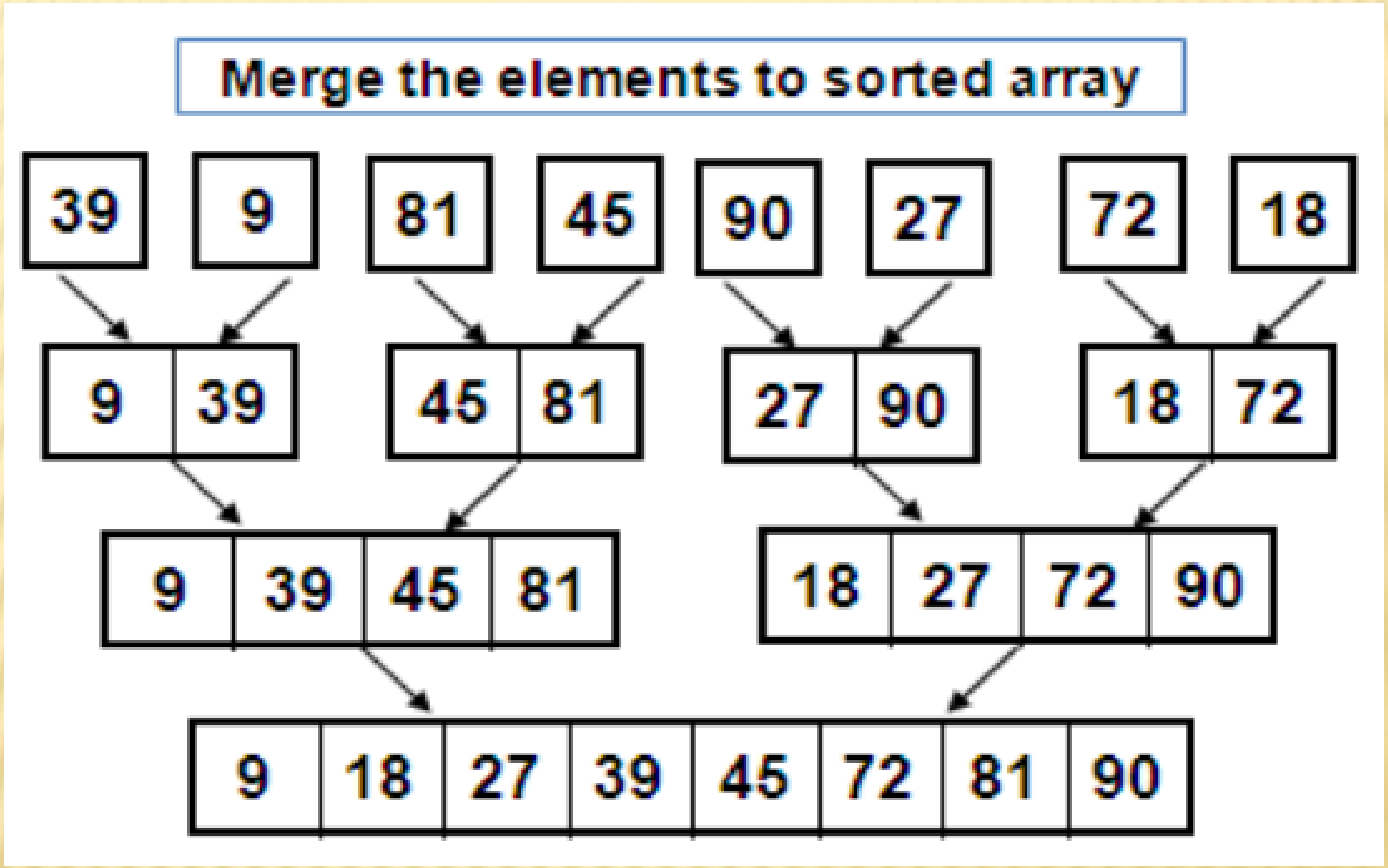
3. Combine Step Combine the elements back in A[p .. r] by merging the two sorted sub-arrays A[p .. q] and A[q + 1 .. r] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r)

Merge sort

Ex:- A list of unsorted elements are: 39 9 81 45 90 27 72 18

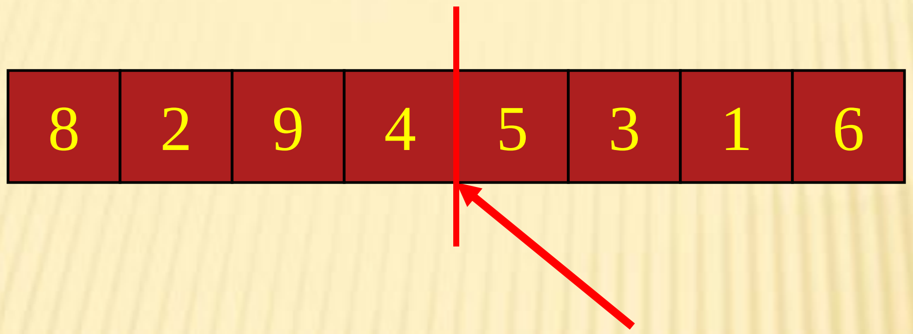


Merge sort



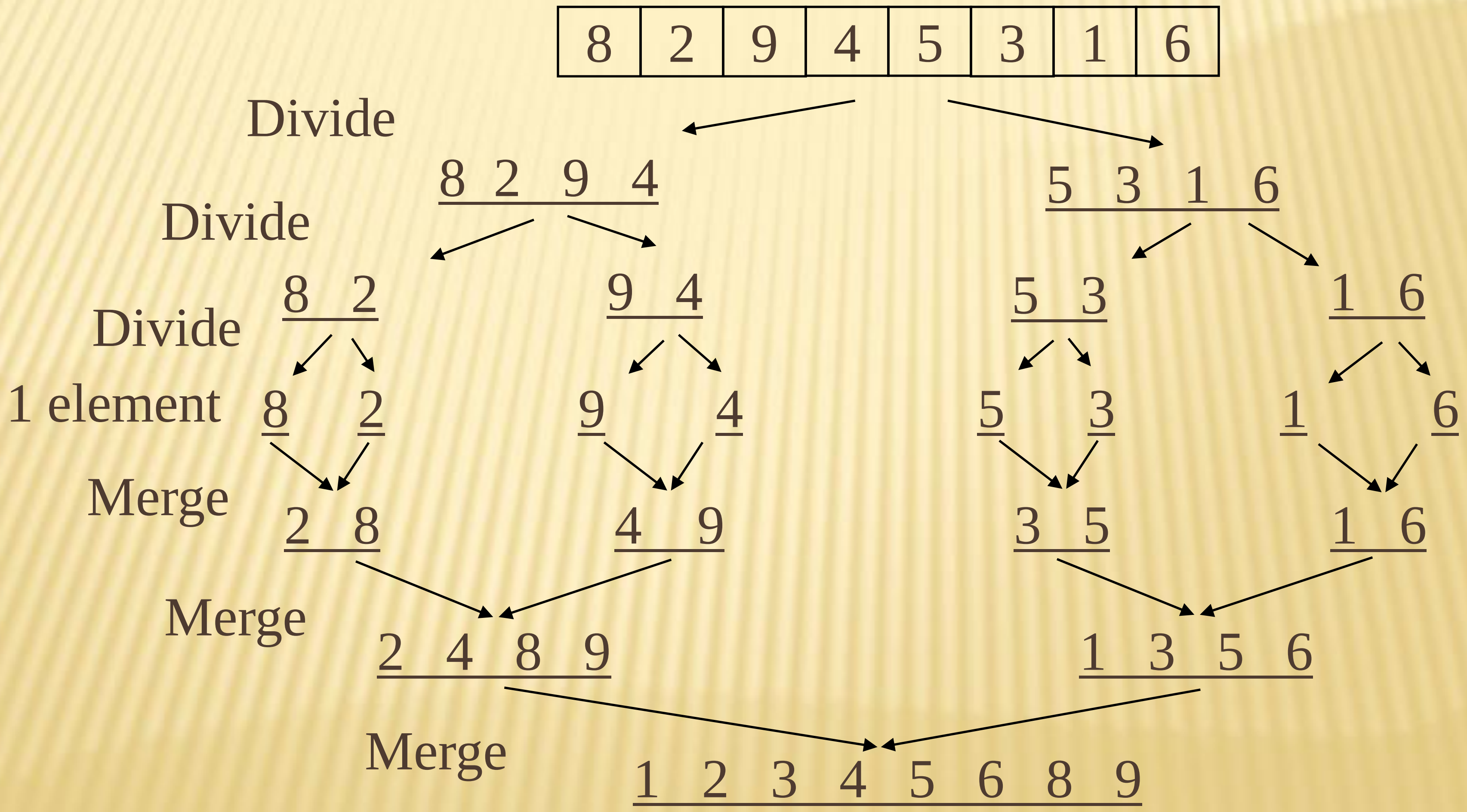
Sorted elements are: 9 18 27 39 45 72 81 90

Merge sort



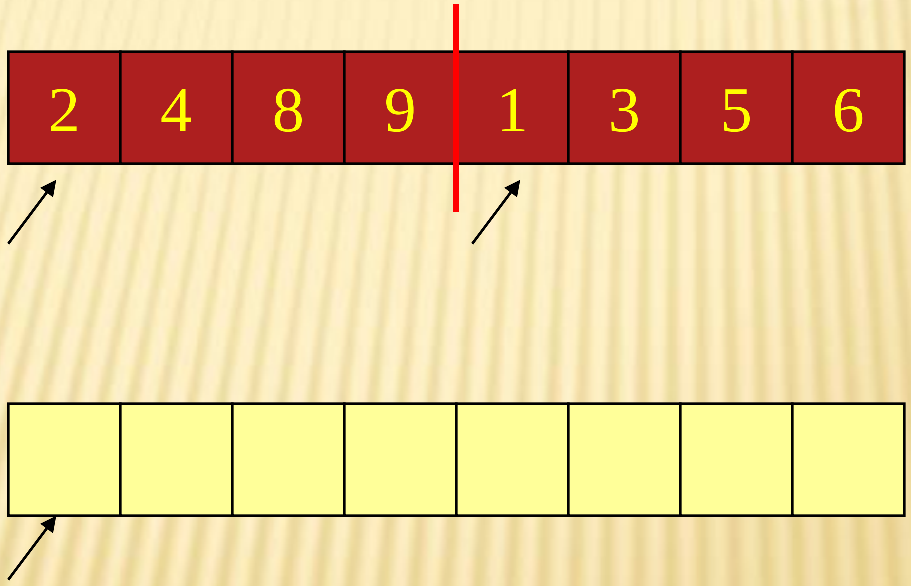
- Divide it in two at the midpoint
- Conquer each side in turn (by recursively sorting)
- Merge two halves together

Merge sort Example



Auxiliary Array

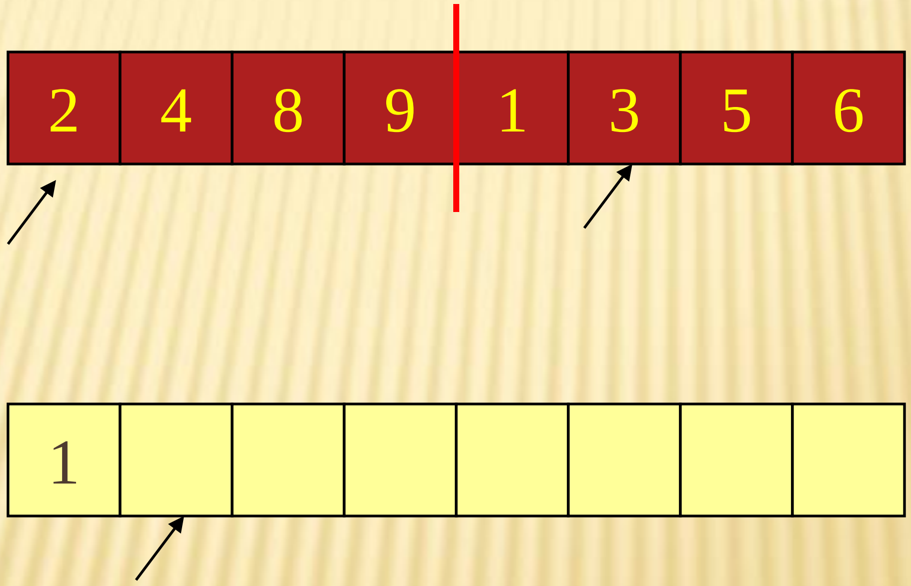
- The merging requires an auxiliary array.



Auxiliary array

Auxiliary Array

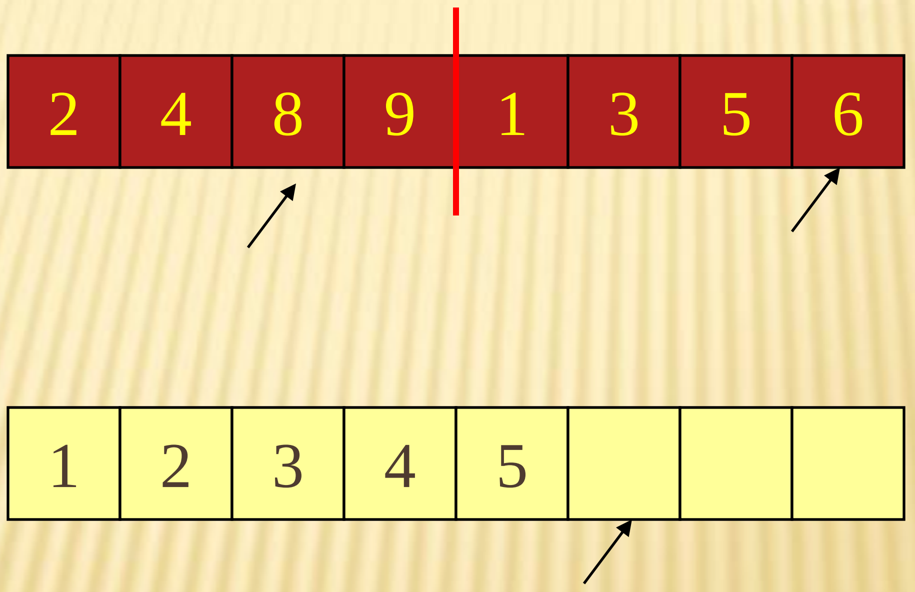
- The merging requires an auxiliary array.



Auxiliary array

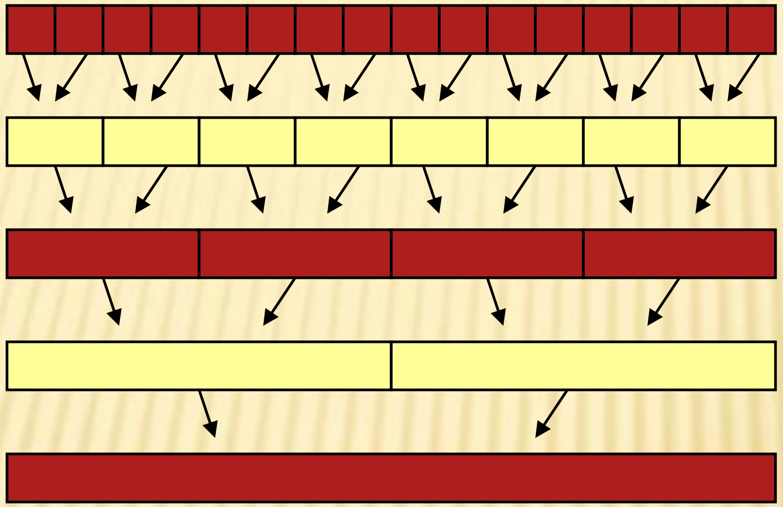
Auxiliary Array

- The merging requires an auxiliary array.



Auxiliary array

Iterative Merge sort



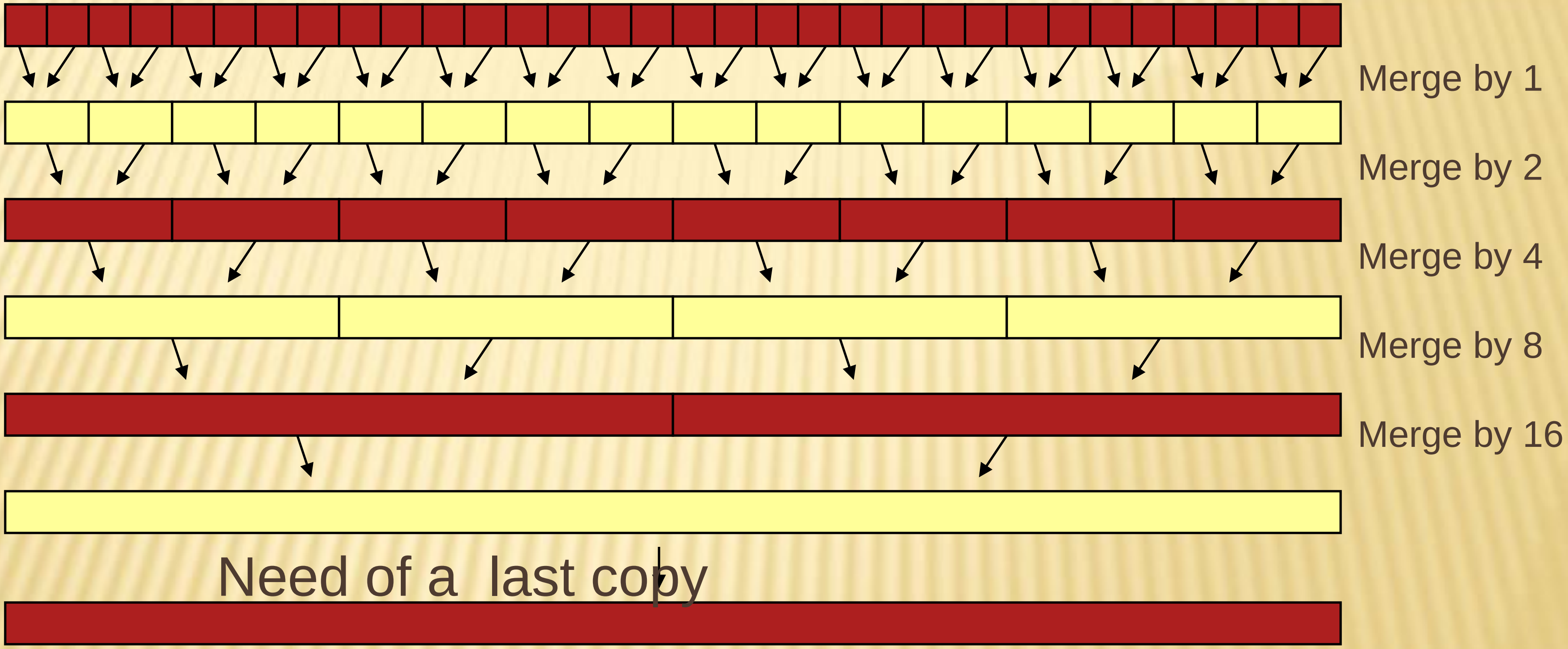
Merge by 1

Merge by 2

Merge by 4

Merge by 8

Iterative Merge sort



Thank You

???