Merge Sort

Joy Mukherjee

Sorting

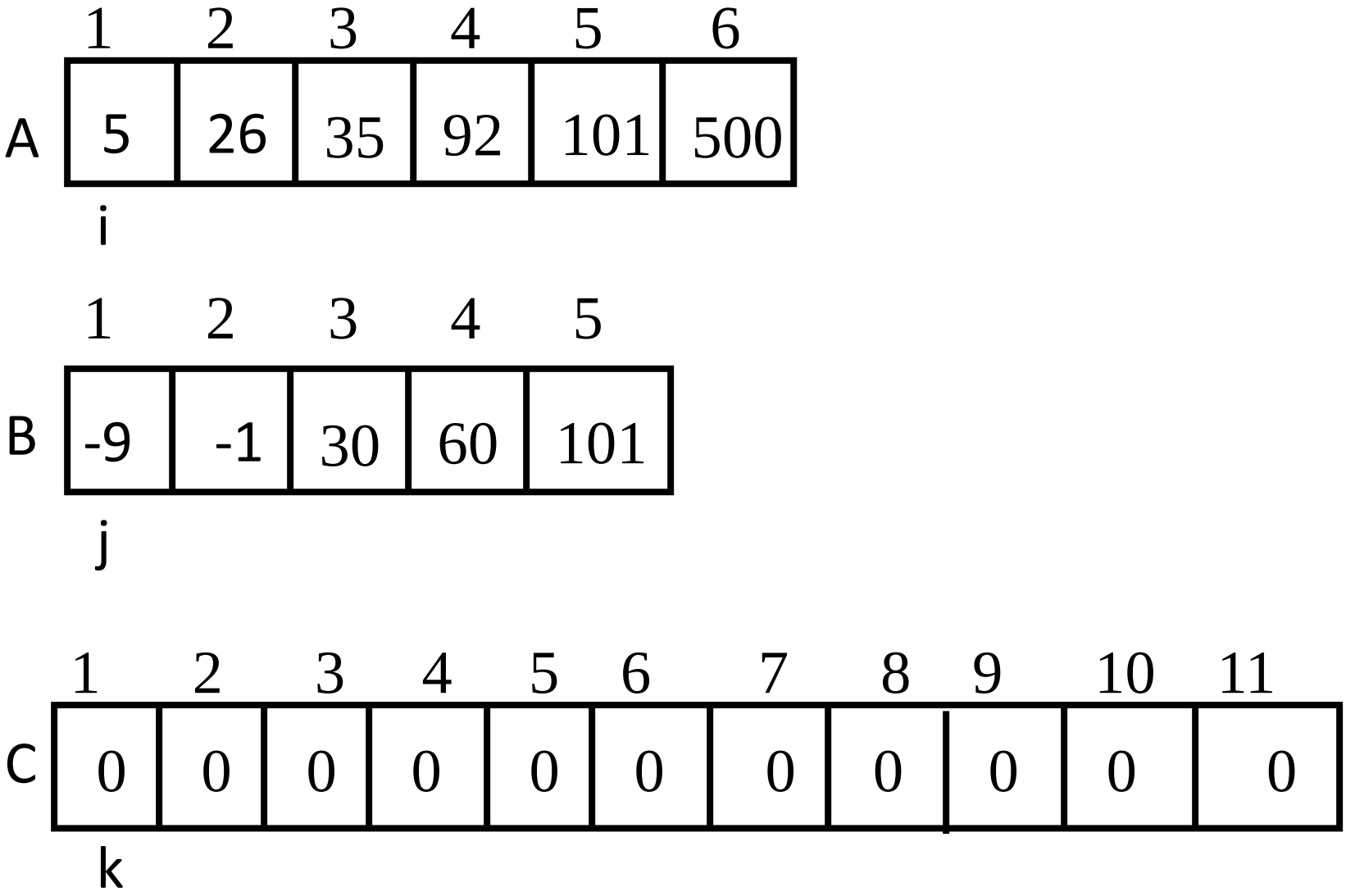
- Given an array A[] of n integers, our objective is to arrange them in non-decreasing order/non-increasing order.
- Bubble Sort/Selection Sort/Insertion Sort
- Merge Sort/Quick Sort/Heap Sort
- Counting Sort/Radix Sort
- Bucket Sort

Merge Sort

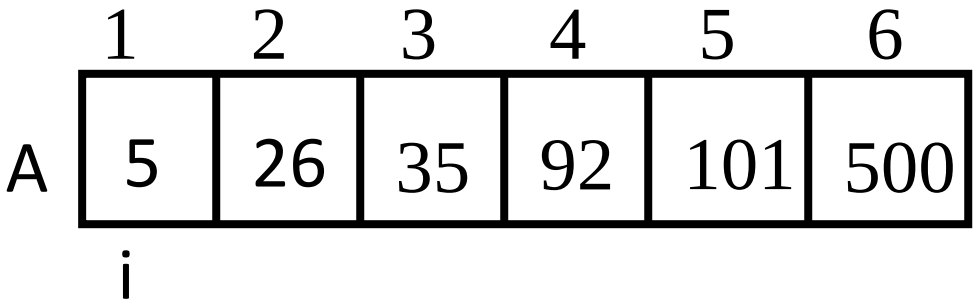
- Given two sorted arrays of length m and n, how do we merge them into a single sorted array of size m+n?

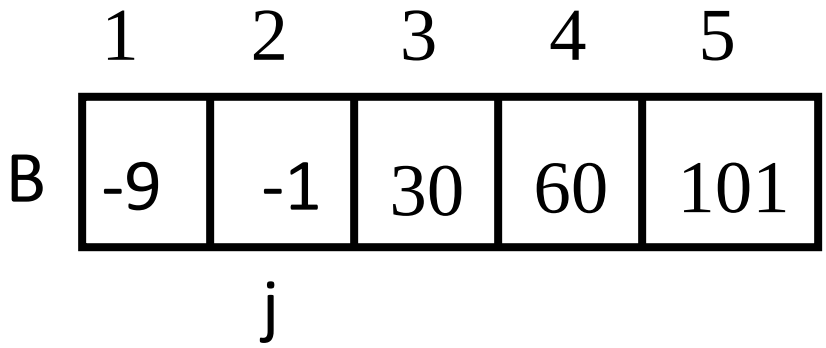
- We use an extra array of size m+n to merge the two sorted arrays.

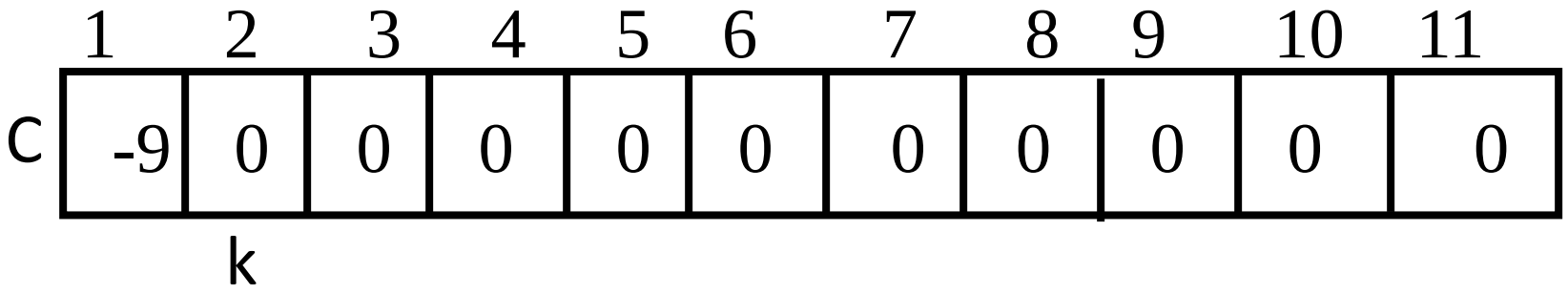
How to merge two sorted array?



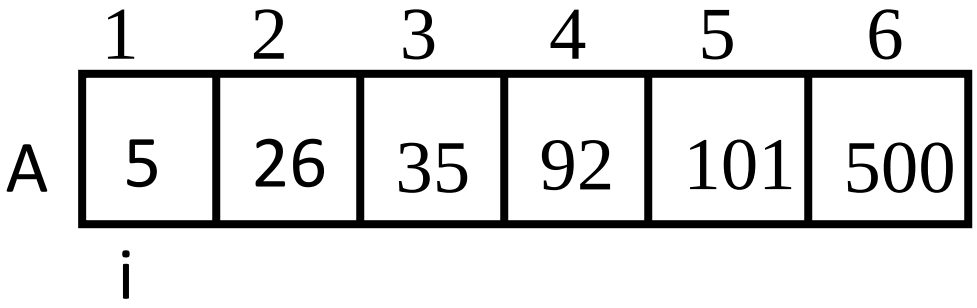
How to merge two sorted array?

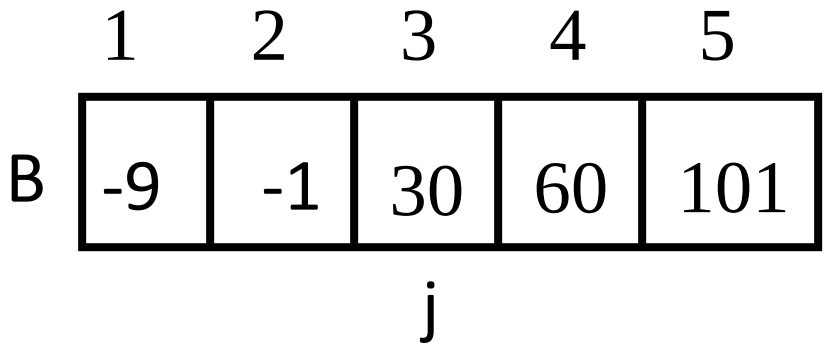


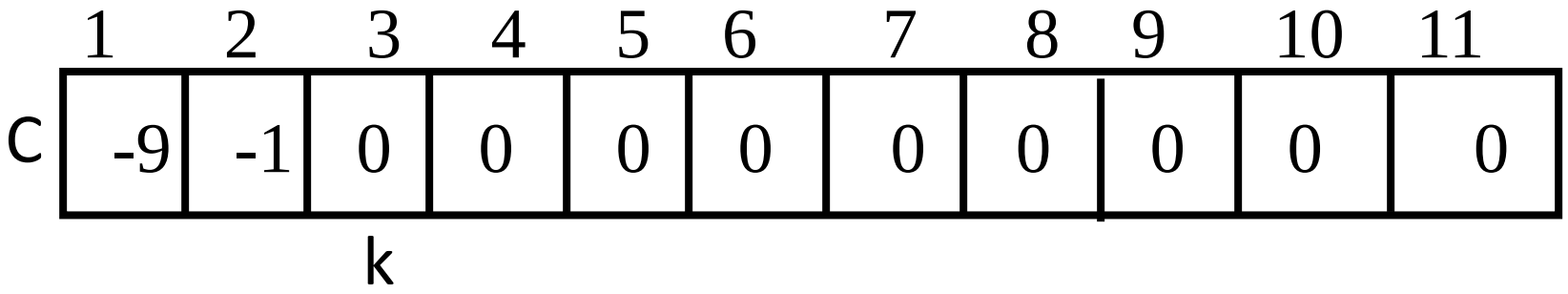




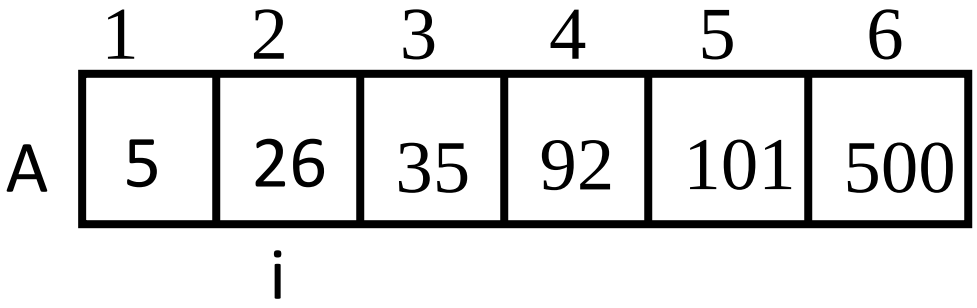
How to merge two sorted array?

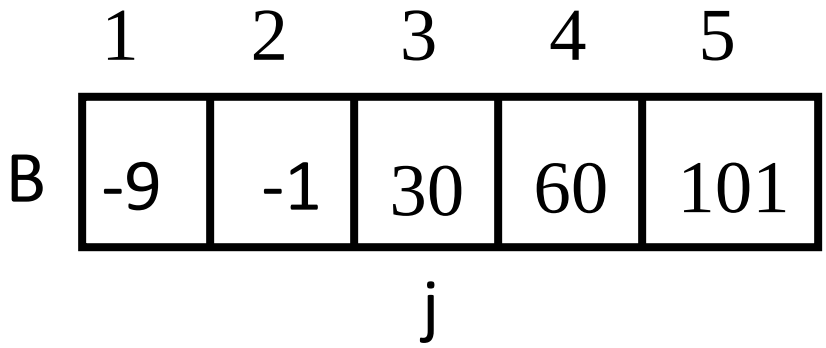


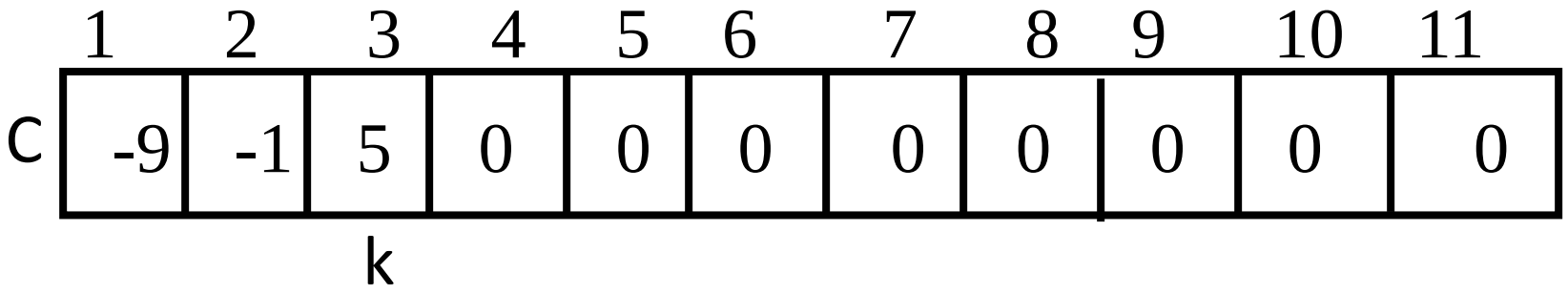




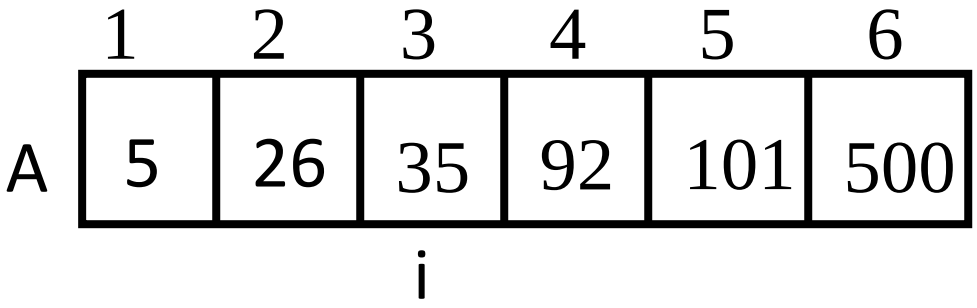
How to merge two sorted array?

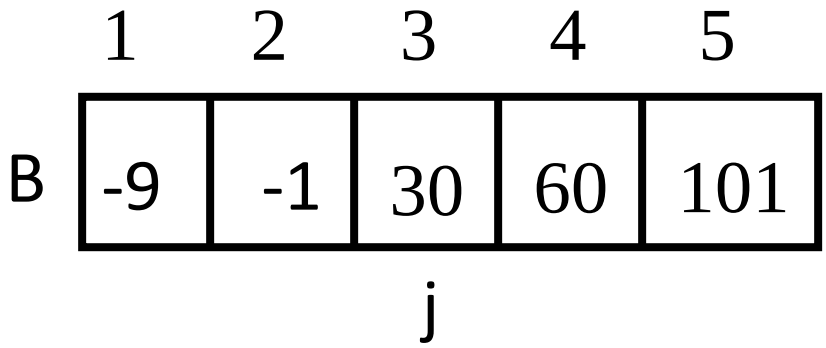


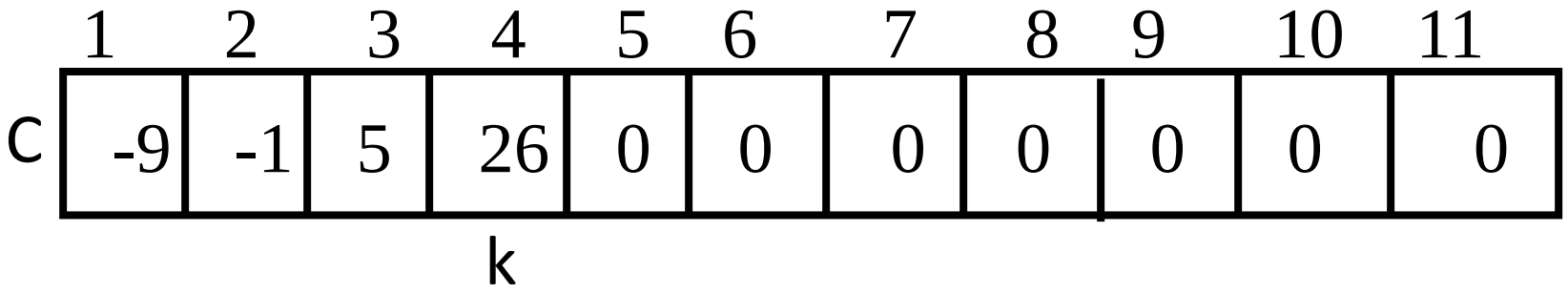




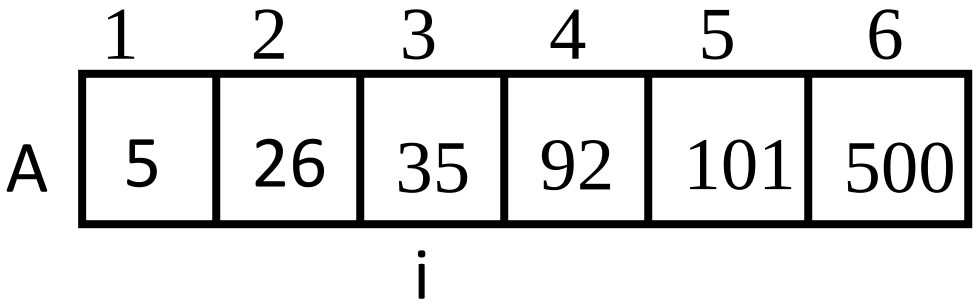
How to merge two sorted array?

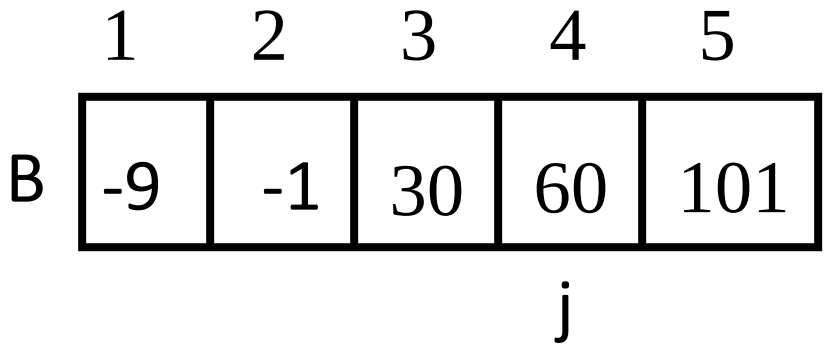


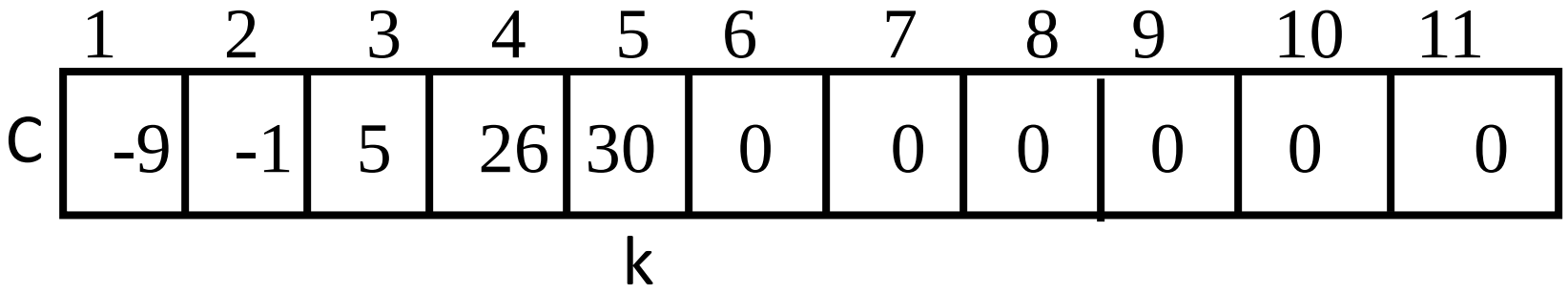




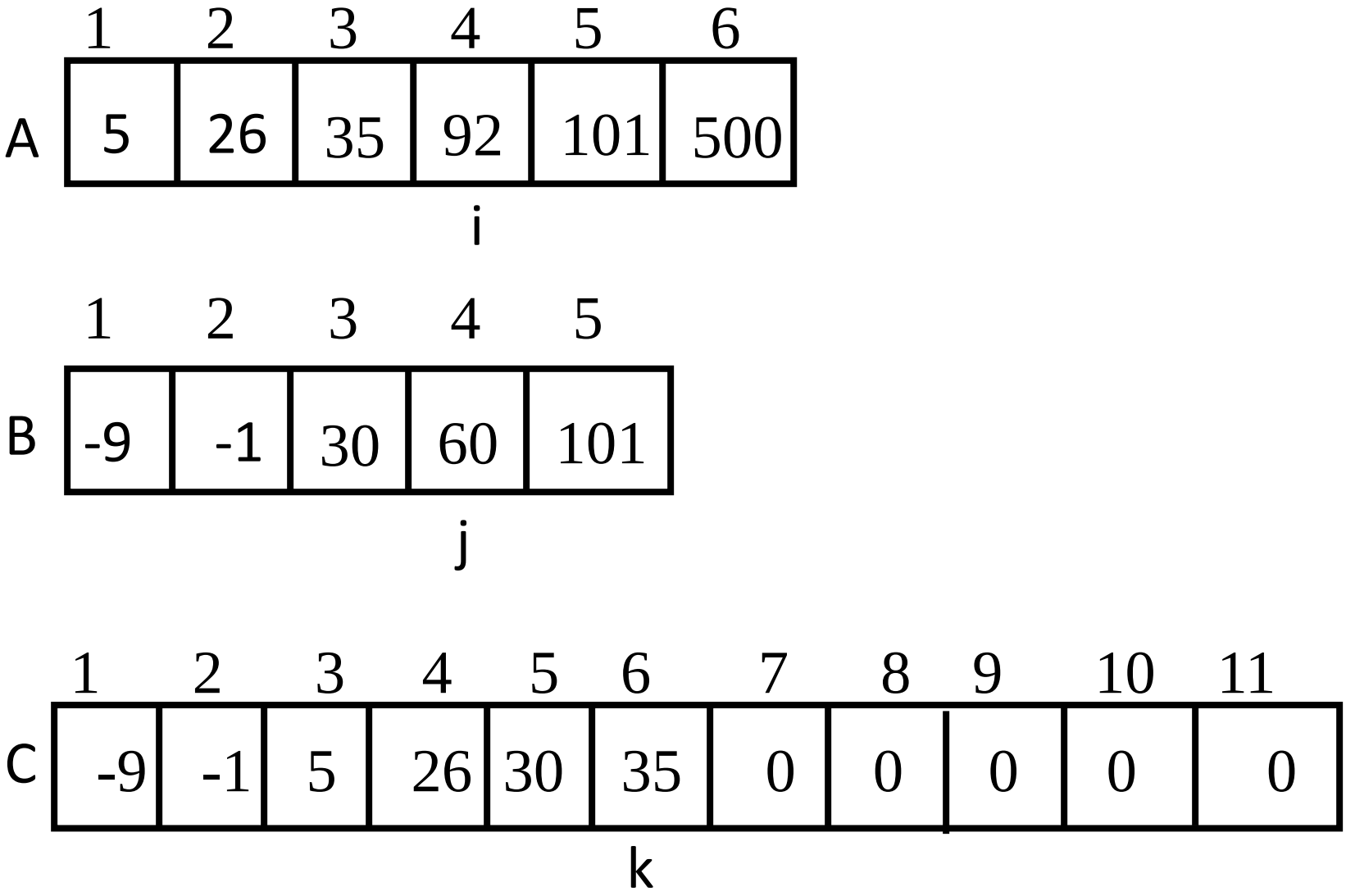
How to merge two sorted array?



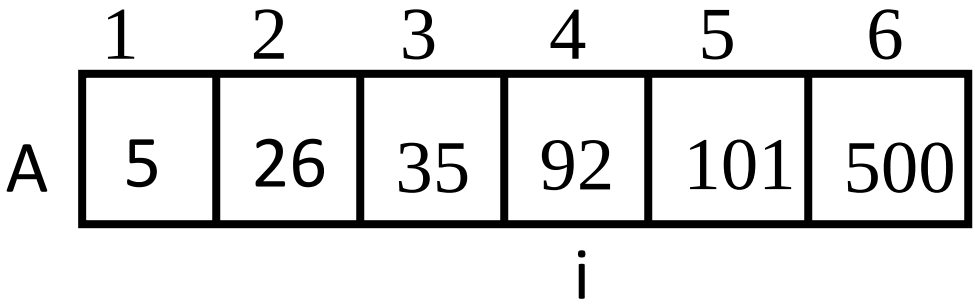


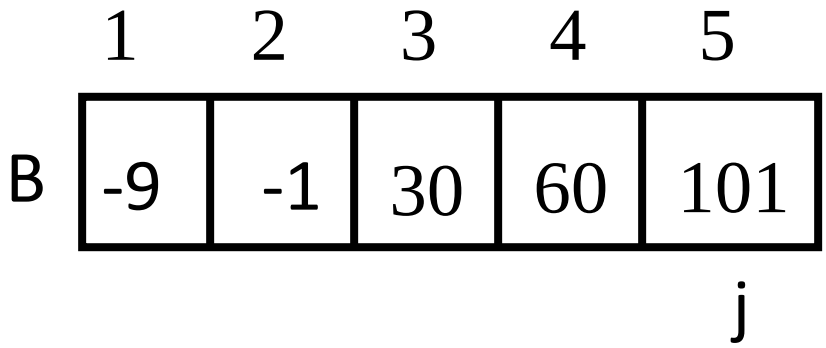


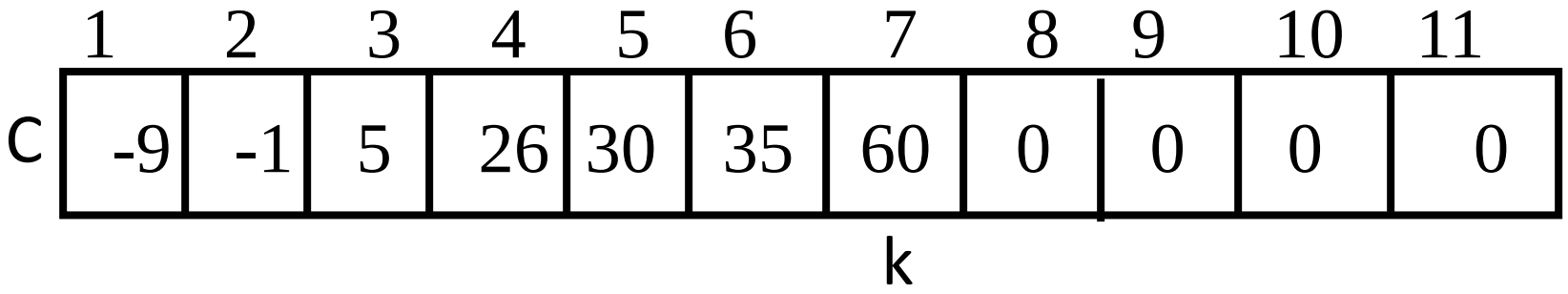
How to merge two sorted array?



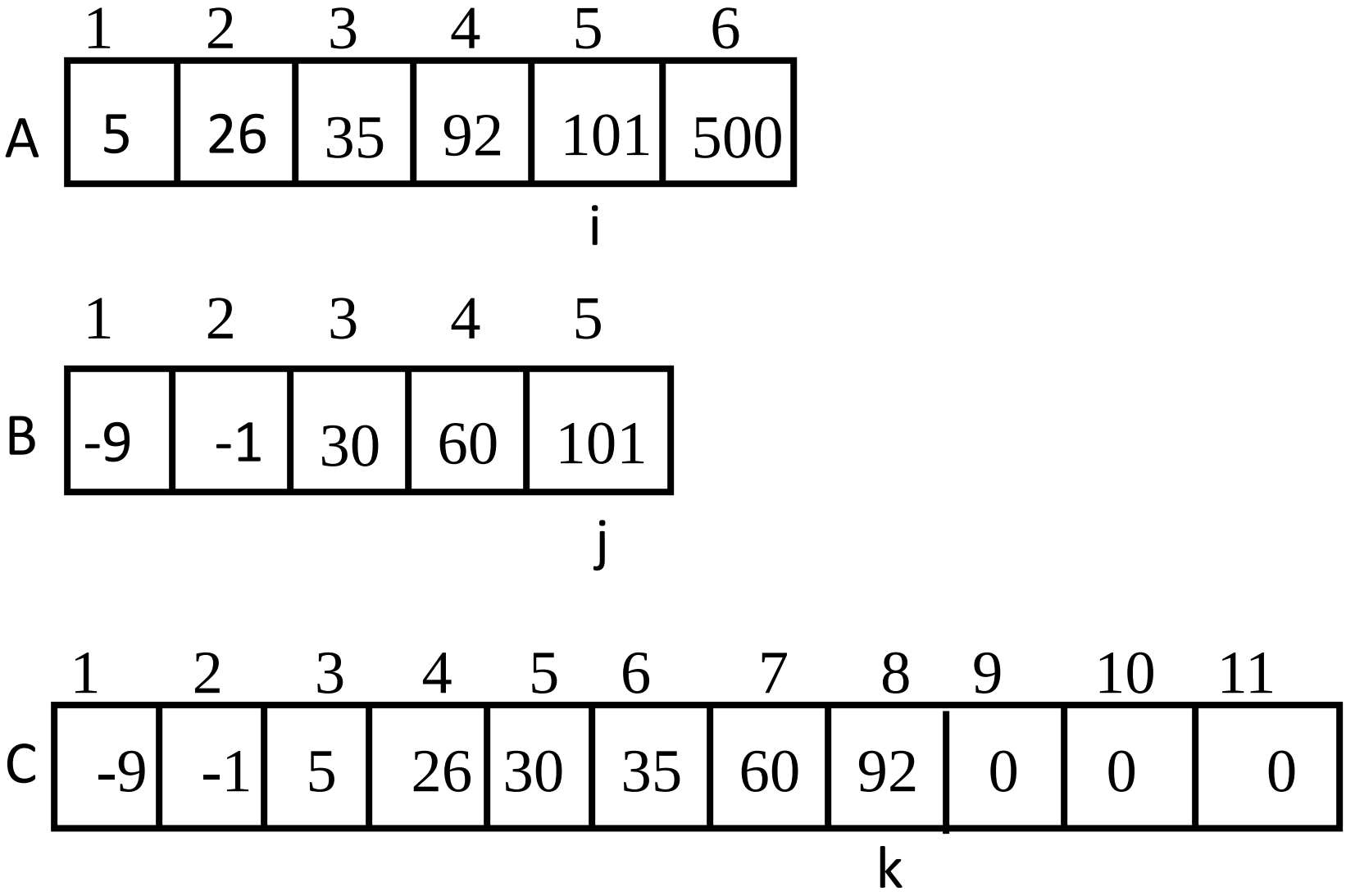
How to merge two sorted array?



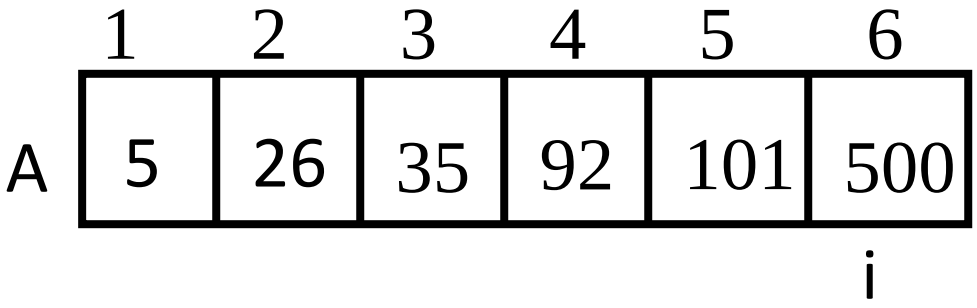


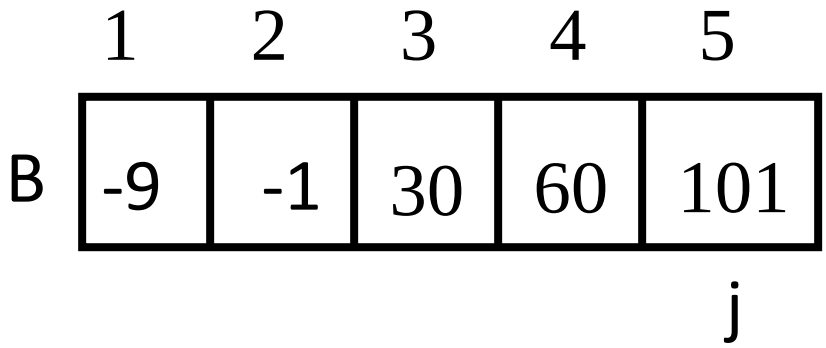


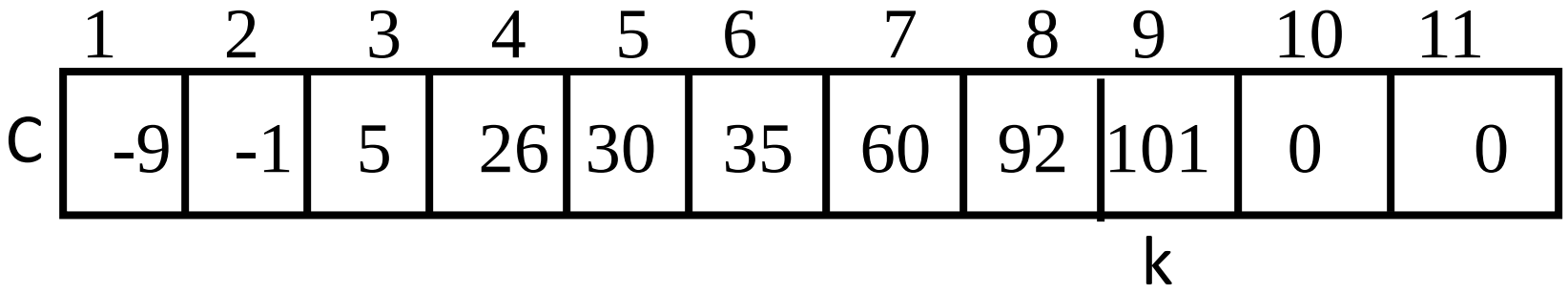
How to merge two sorted array?



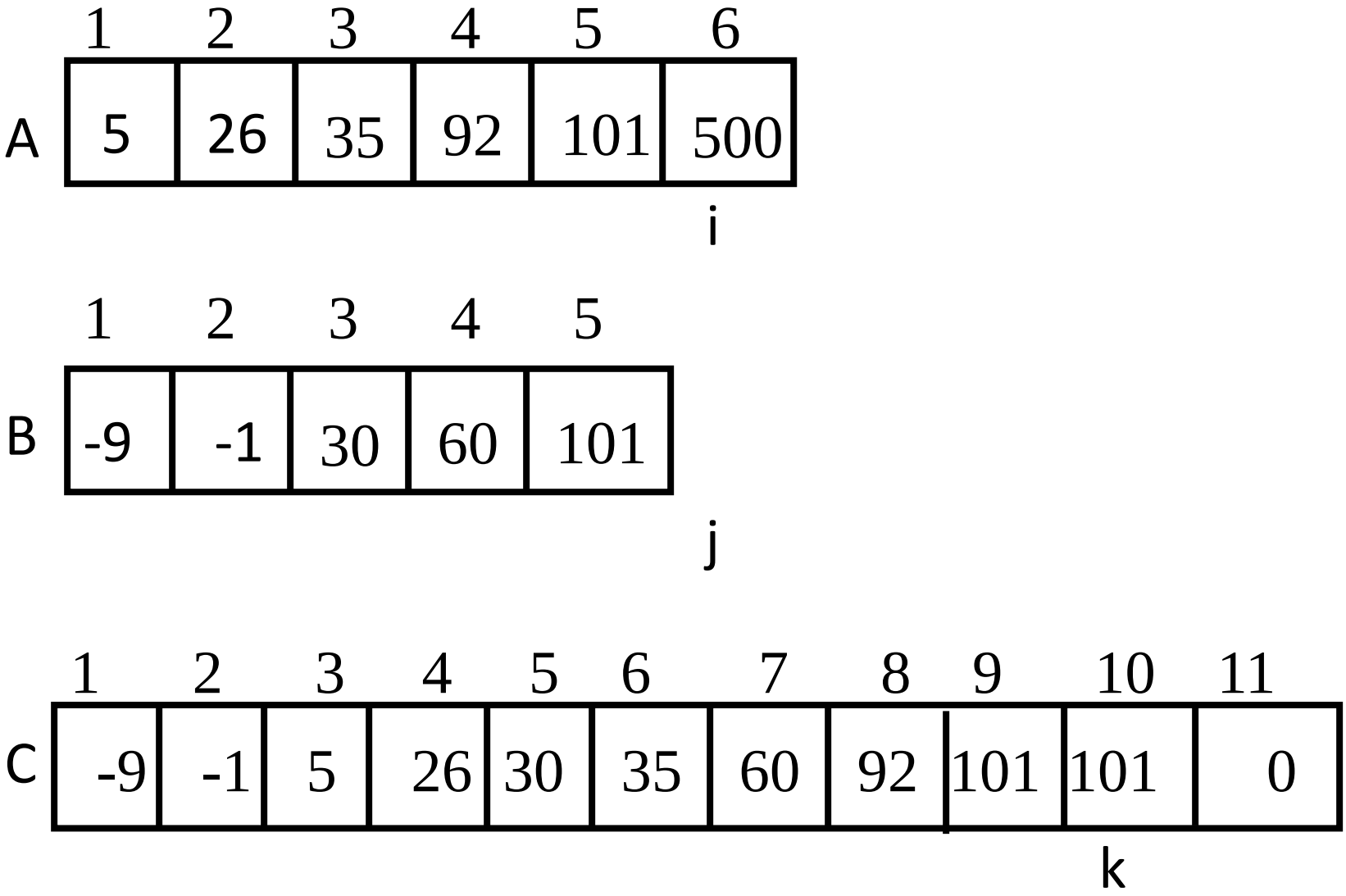
How to merge two sorted array?



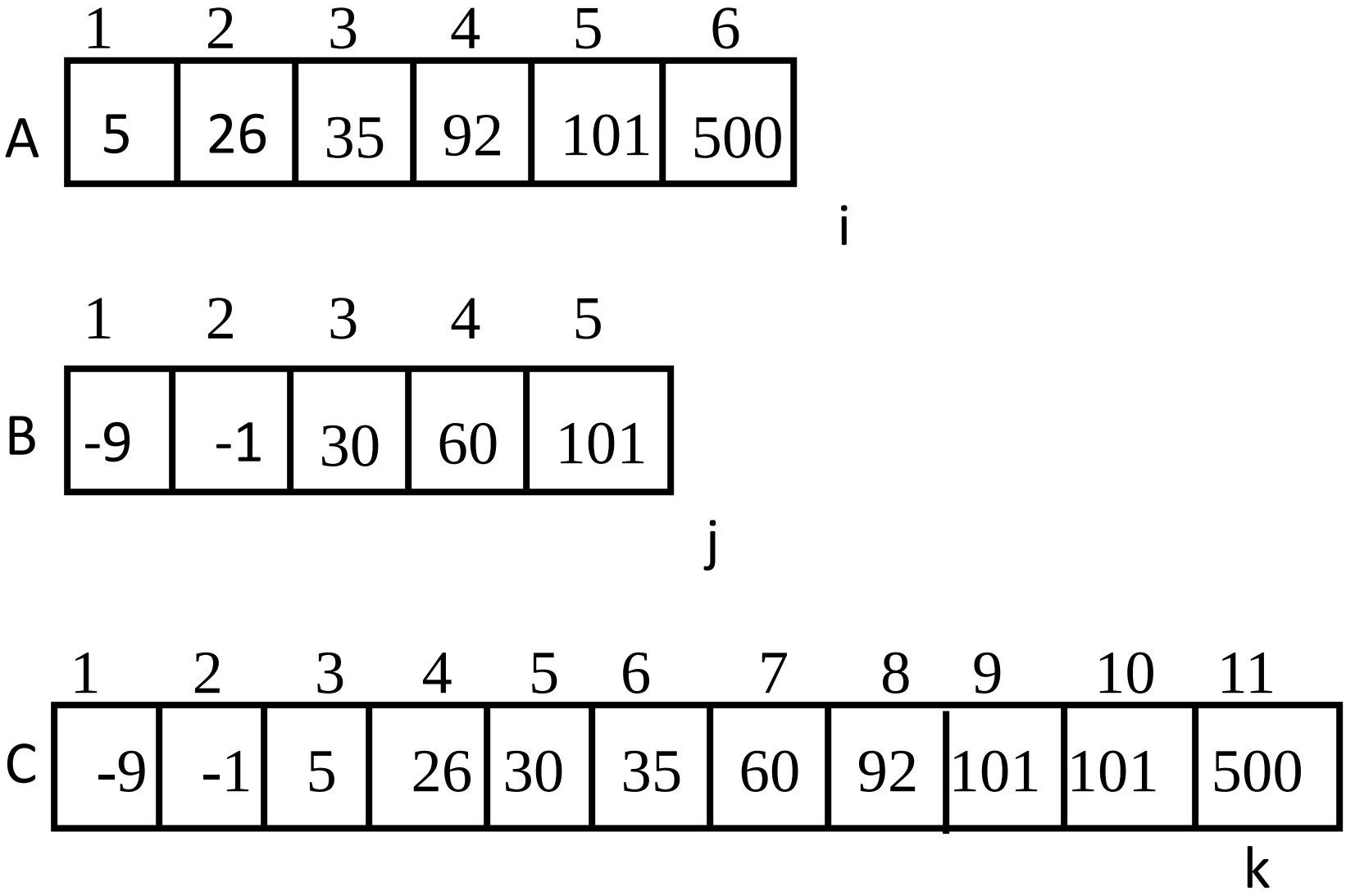




How to merge two sorted array?

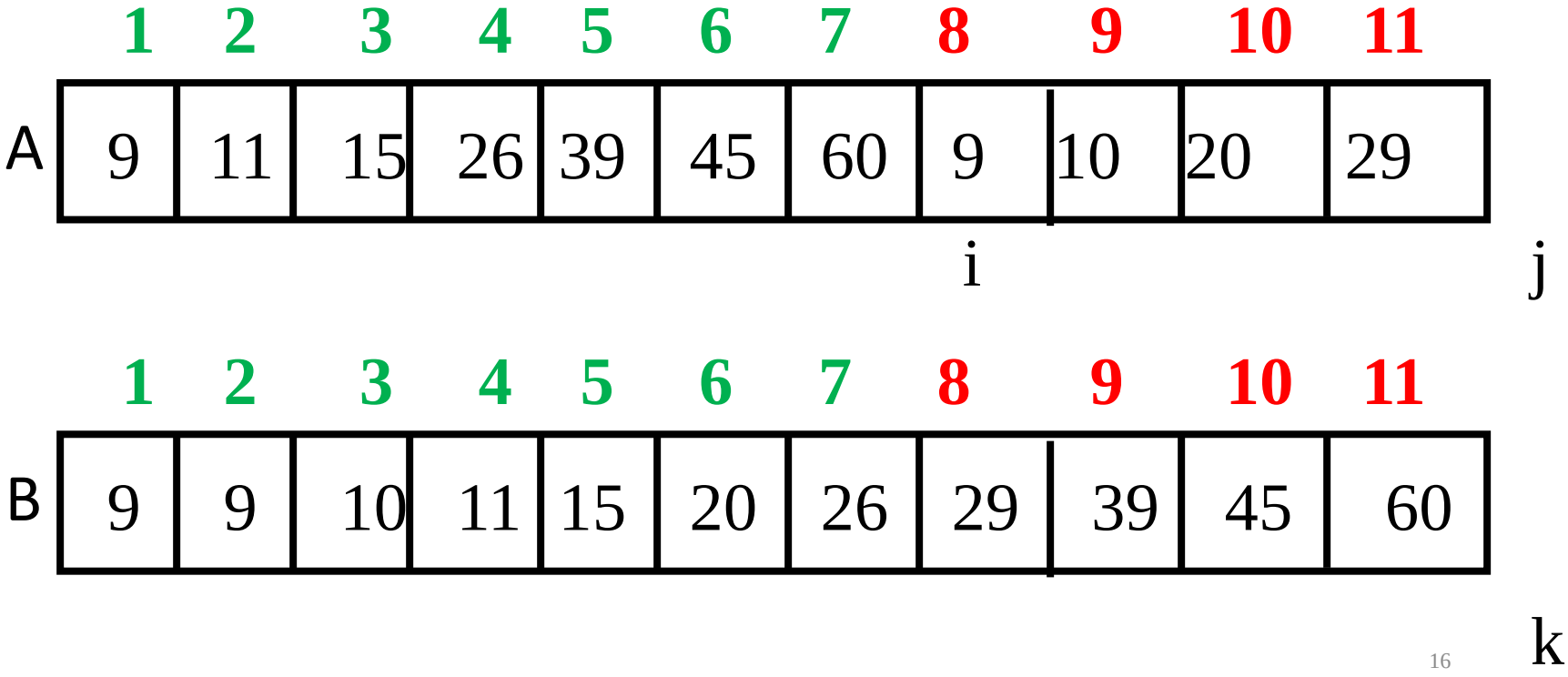


How to merge two sorted array?



Merge

- A [low..high]
- A[low..mid] and A[mid+1..high]



Merge

```
void merge(int a[], int low, int mid,
  int high)
{
      int i = low, j = mid+1, k = 0;
      int b[high-low+1];
      while(i <= mid && j <= high) { // Both
      array are not exhausted
        if(a[i] <= a[j])
                {b[k] = a[i]; i++; k++;}
        else
                {b[k] = a[j]; j++; k++;}
      }
```

a[low] to a[mid] = A
a[mid+1] to a[high] = B
b[0] to b[high-low] = C

Merge

```
    while(j != high+1) // a[mid+1] to a[high] is not exhausted
        b[k++] = a[j++];
    while(i != mid+1) // a[low] to a[mid] is not exhausted
        b[k++] = a[i++];

    for(i = low, j = 0; i <= high; i++, j++)
        a[i] = b[j];  // Copy sorted b[0 .. High-low] to a[low to high]
}
```

Merge Sort

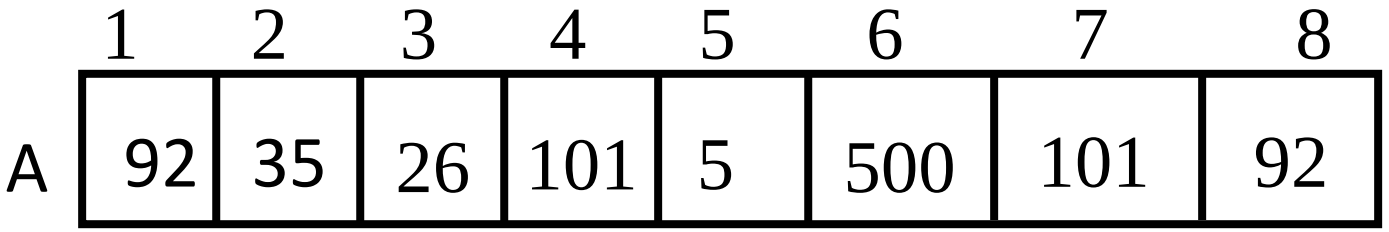
- It is a recursive function.
  - Divide the array into almost equal two halves
  - Recursively sort left half **A[low..mid]**
  - Recursively sort right half **A[mid+1..high]**
  - Merge the two halves to get the array sorted **A[low..high]**

Merge Sort

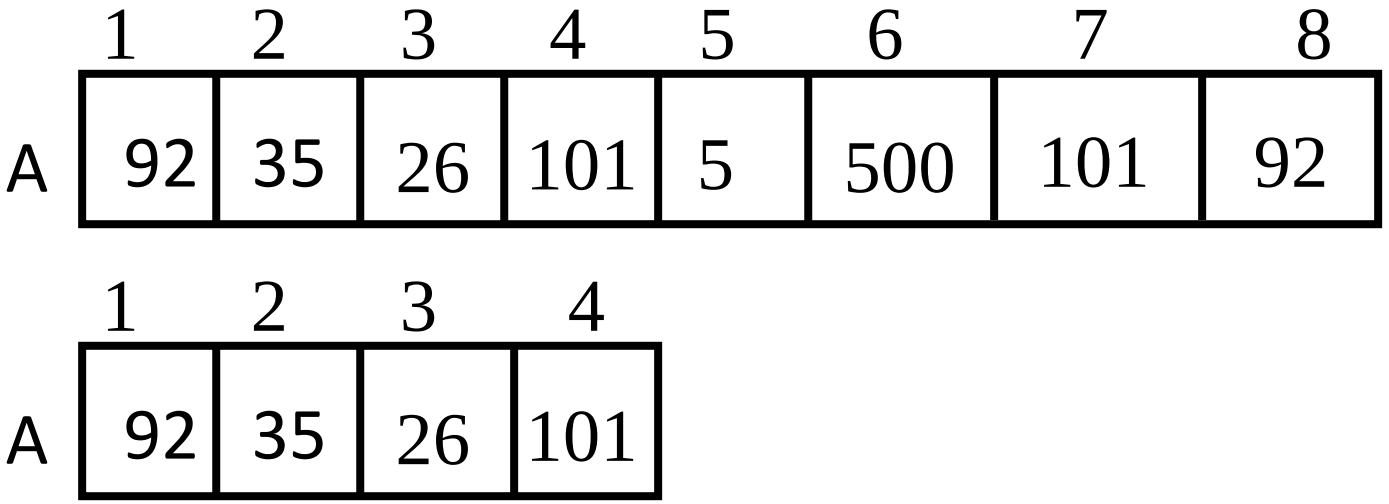
```
void mergesort(int a[], int low, int high)
{
  if(low < high) {
        int mid = (high+low)/2;
        mergesort(a, low, mid); // Sort the left half
        mergesort(a, mid+1, high); // Sort the right half
        merge(a, low, mid, high); // Merge the sorted halves
  }
}
```

From main(): mergesort(a, 0, n-1);

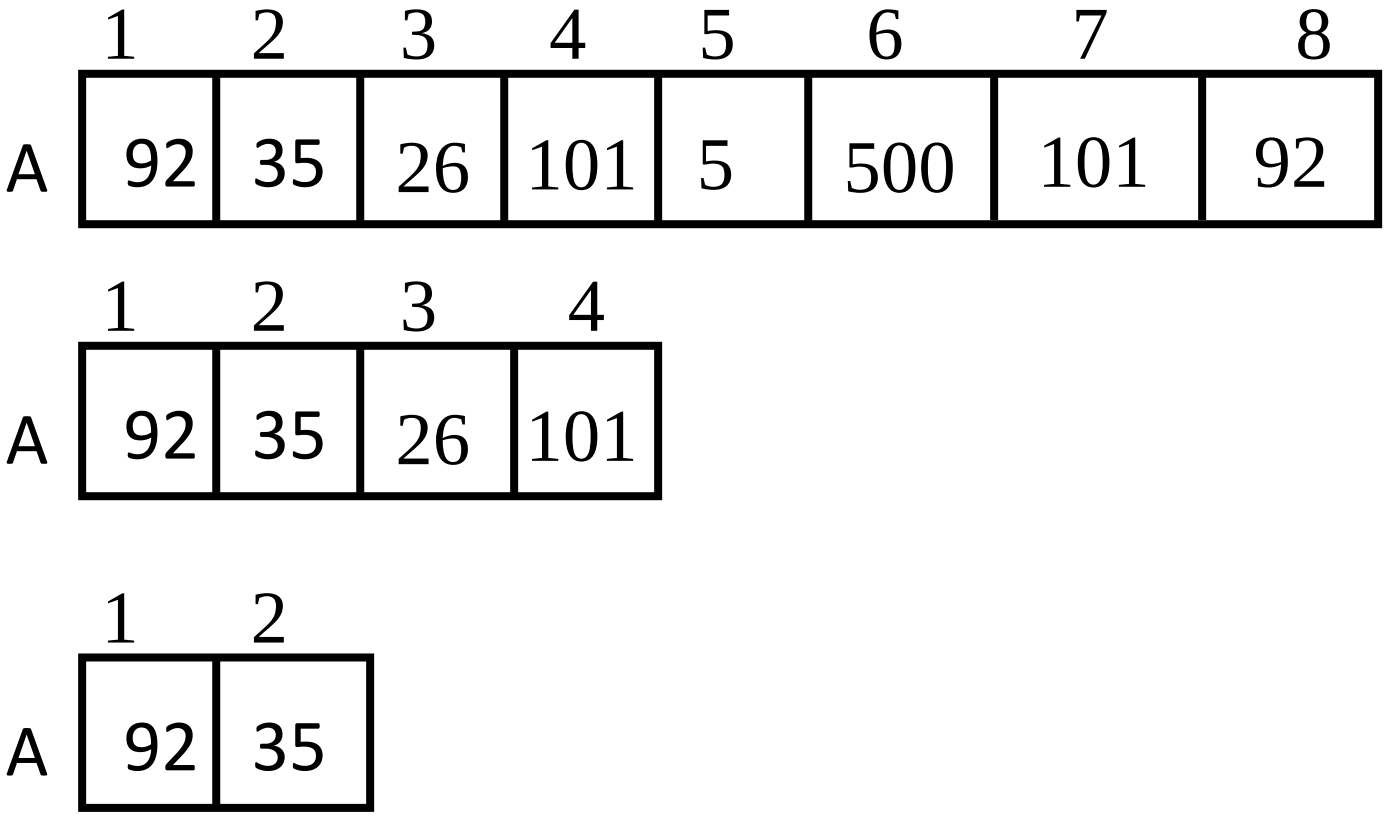
Merge Sort



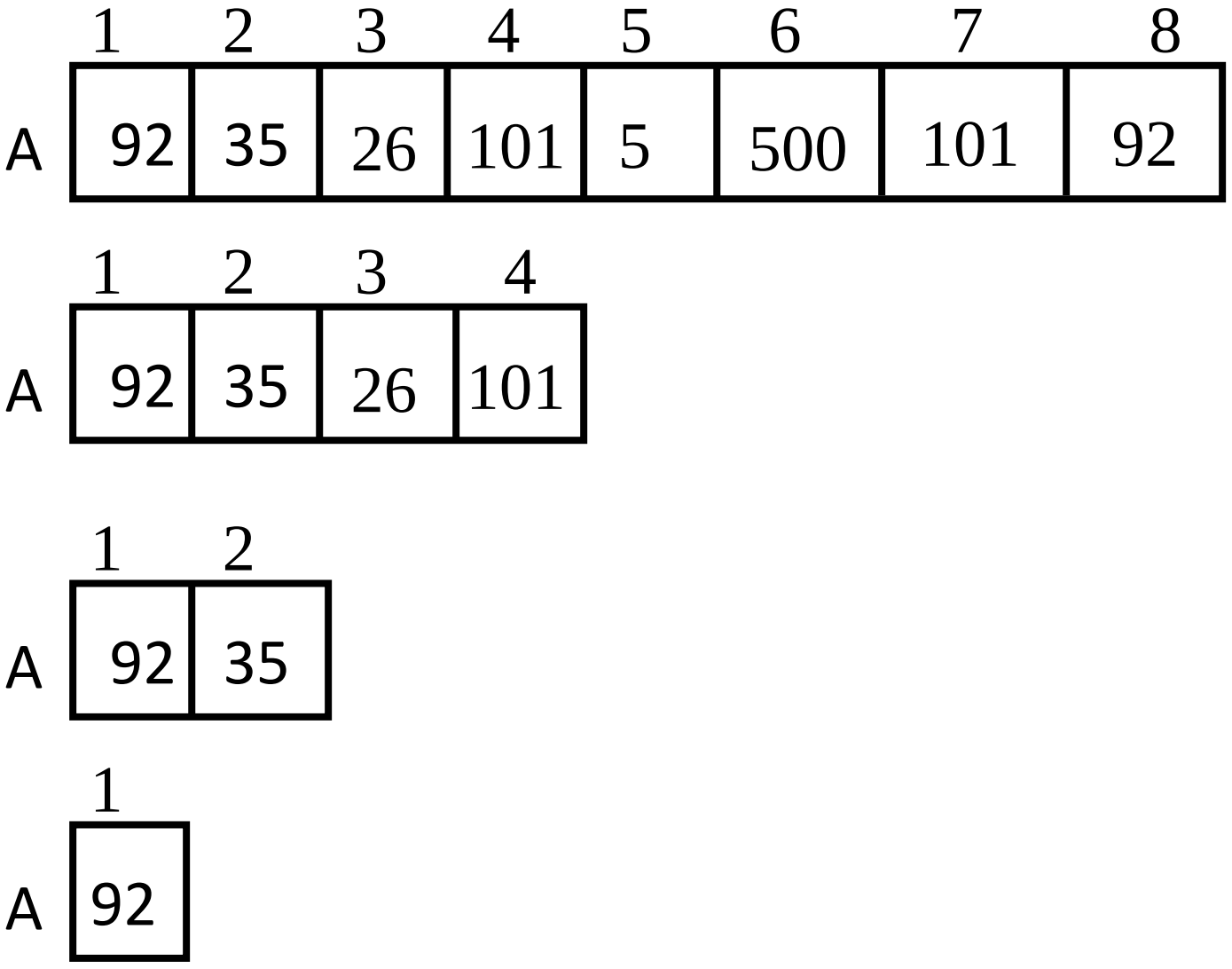
Merge Sort : Divide (Level 1)



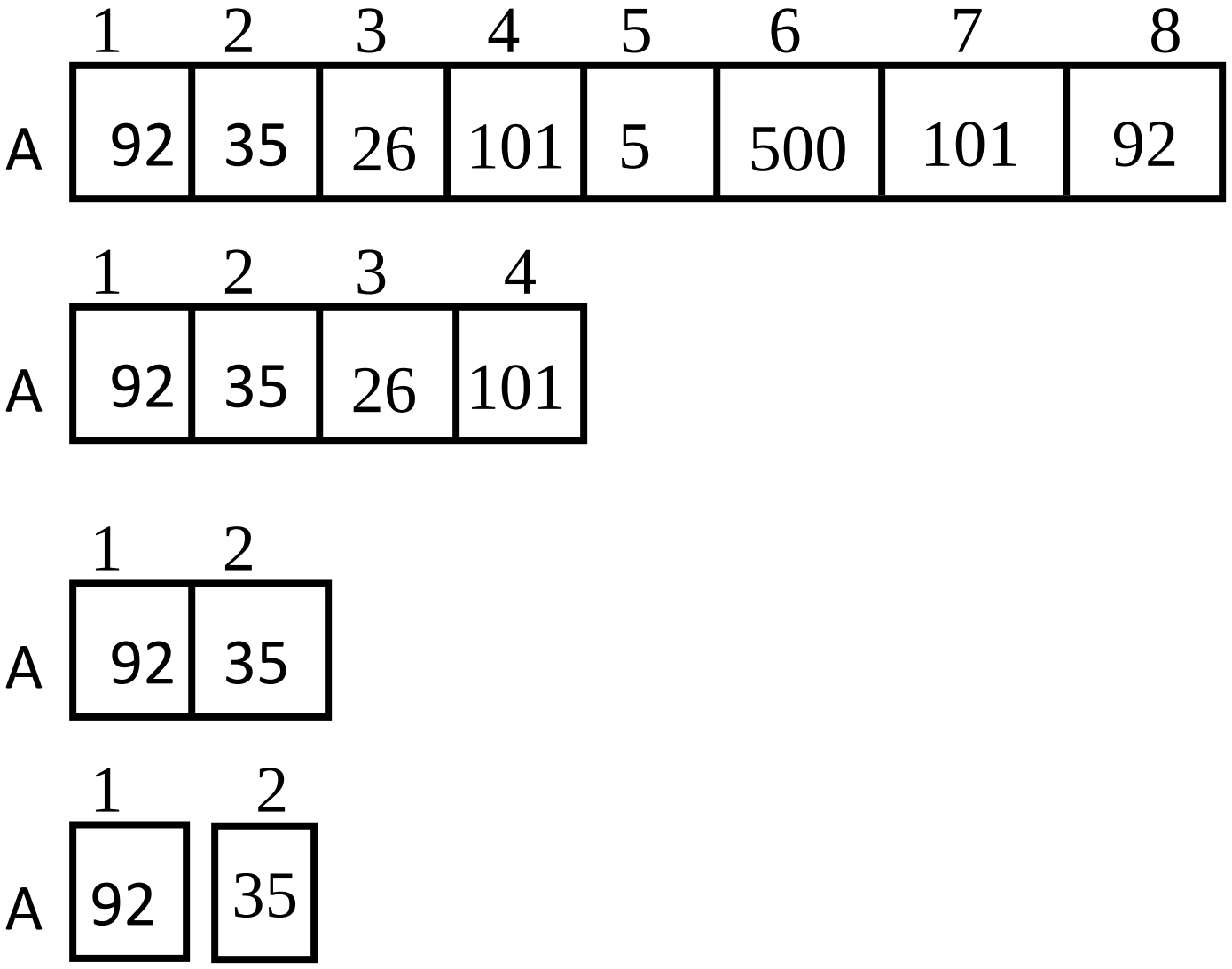
Merge Sort : Divide (Level 2)



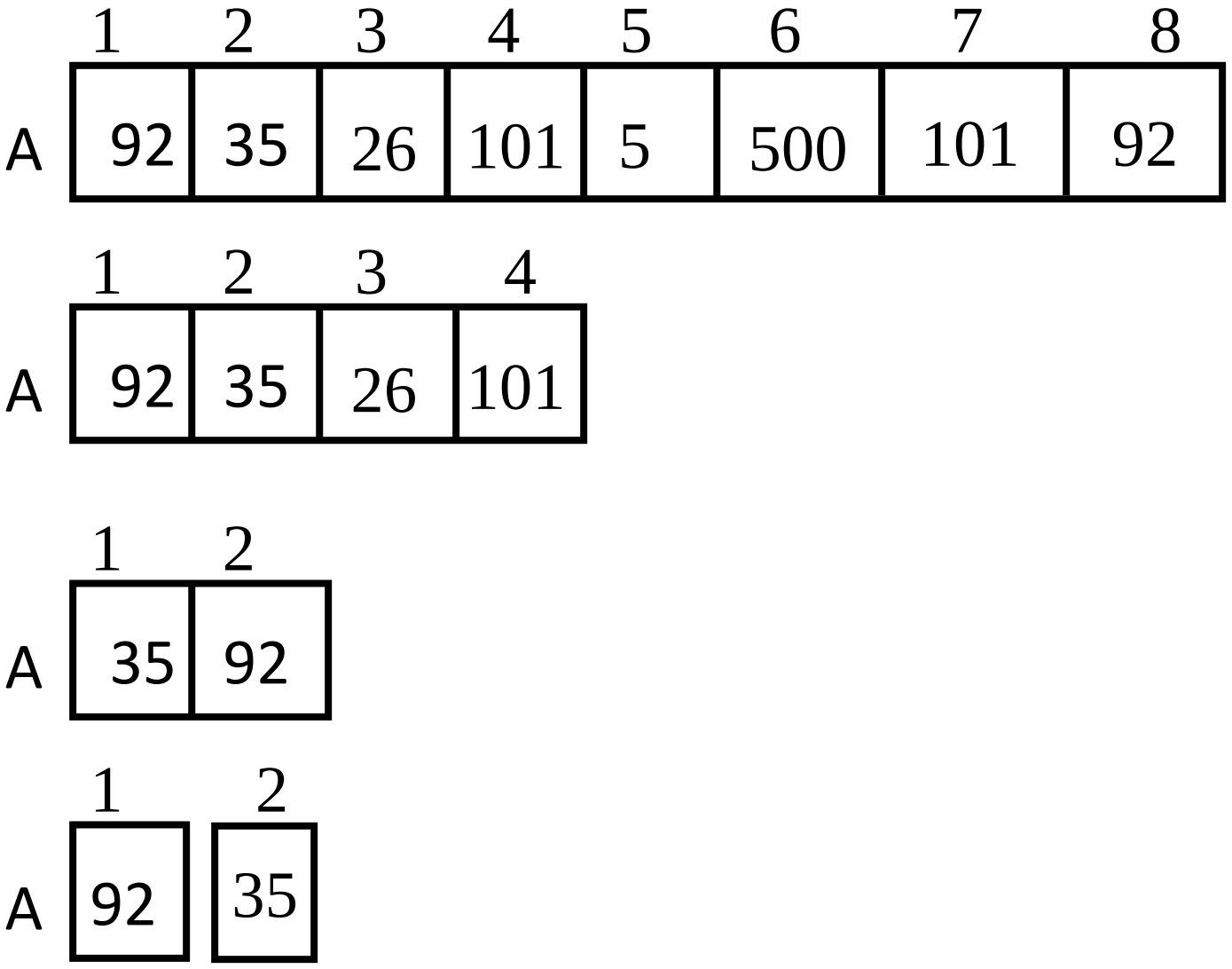
Merge Sort : Divide (Level 3) & Conquer (Level 4)



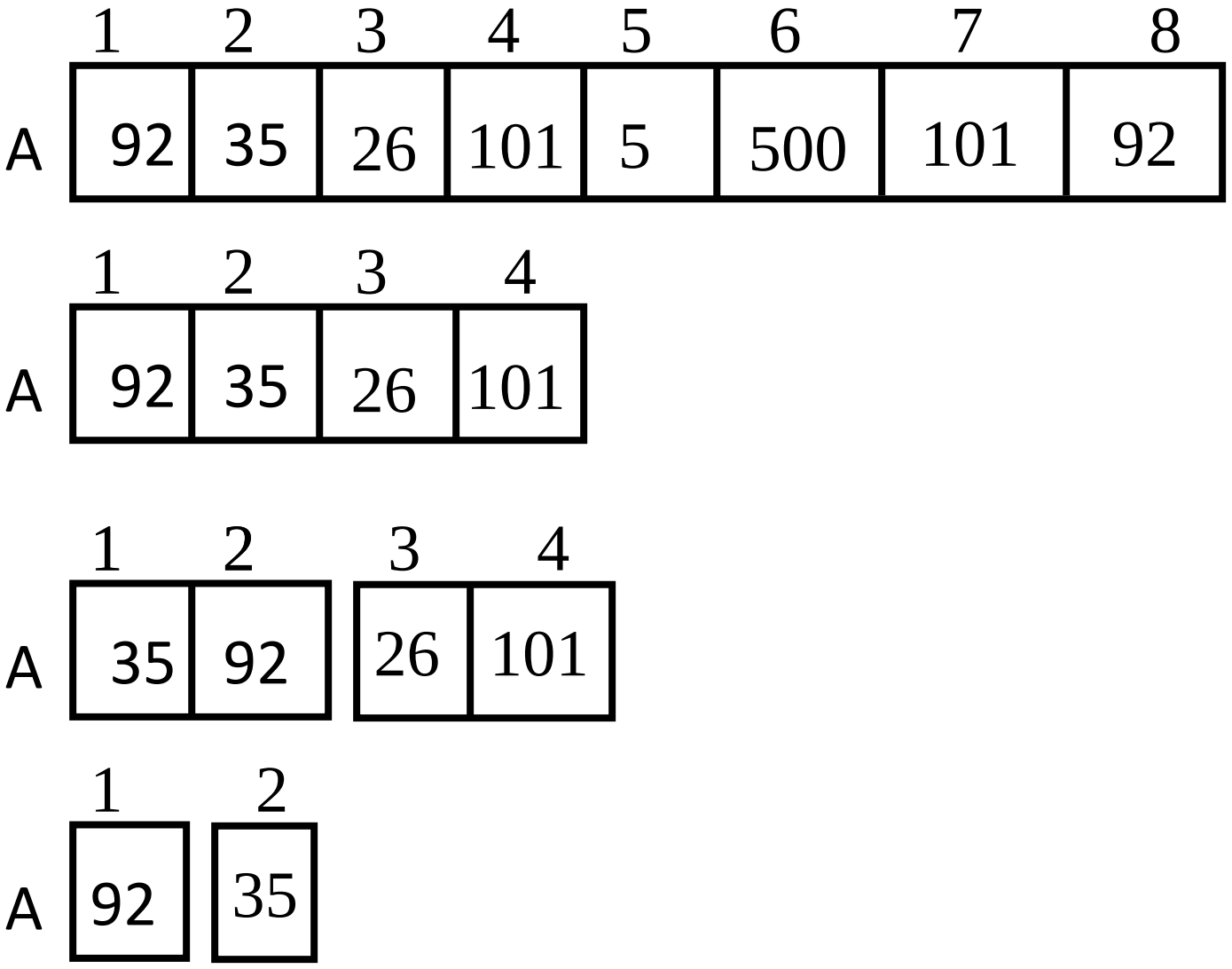
Merge Sort : Conquer (Level 4)



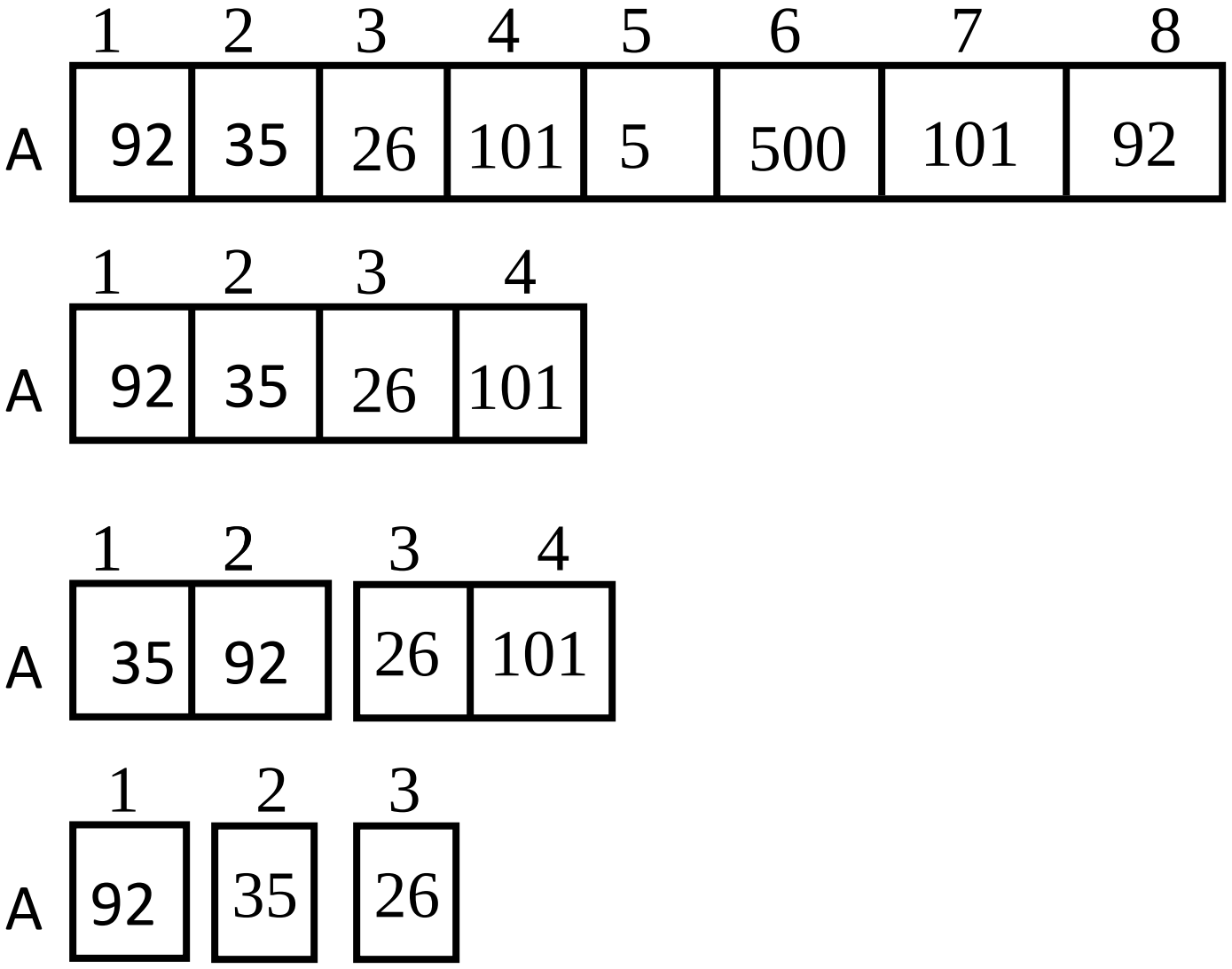
Merge Sort : Conquer (Level 3)



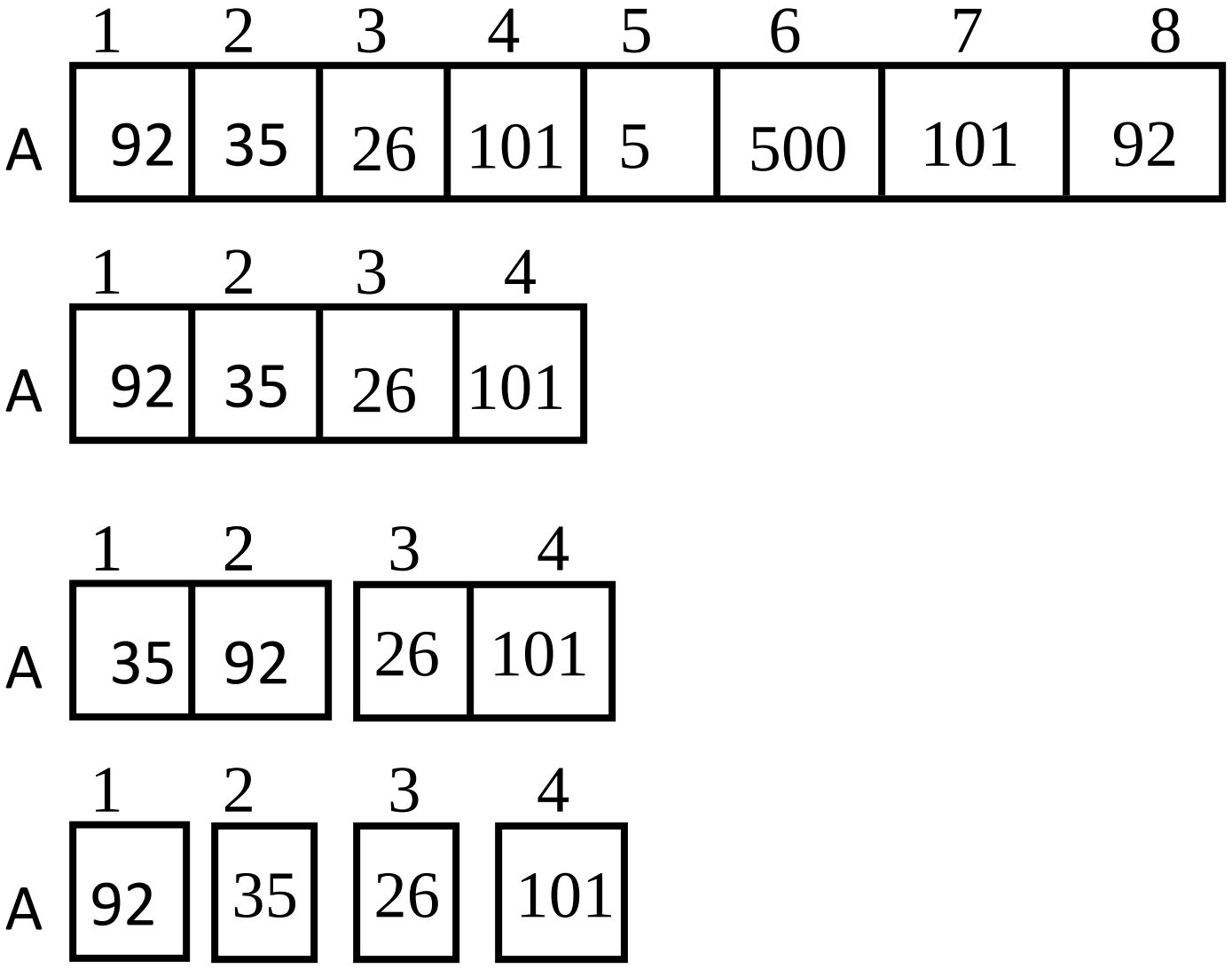
Merge Sort : Divide (Level 2)



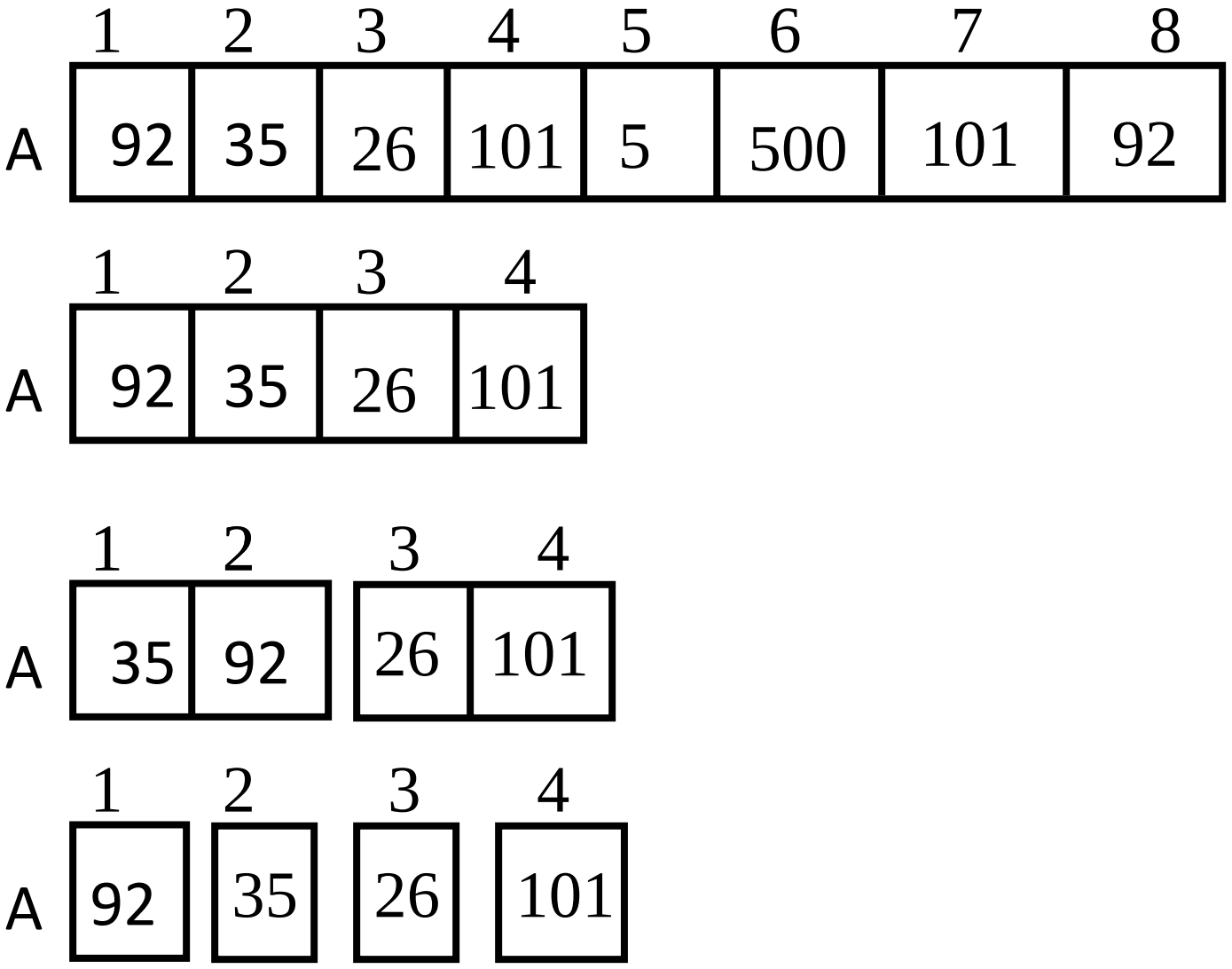
Merge Sort : Divide (Level 3) & Conquer (Level 4)



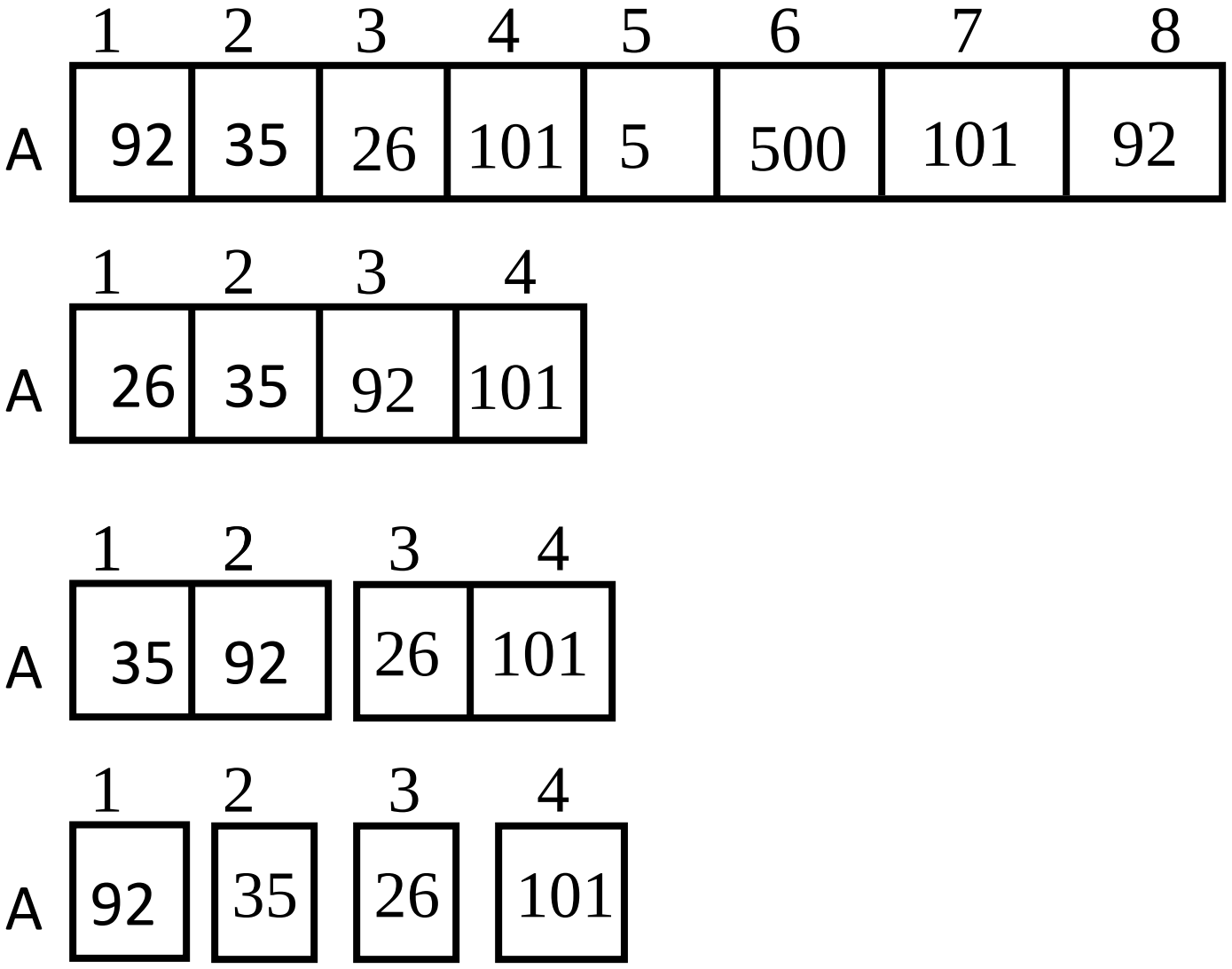
Merge Sort : Conquer (Level 4)



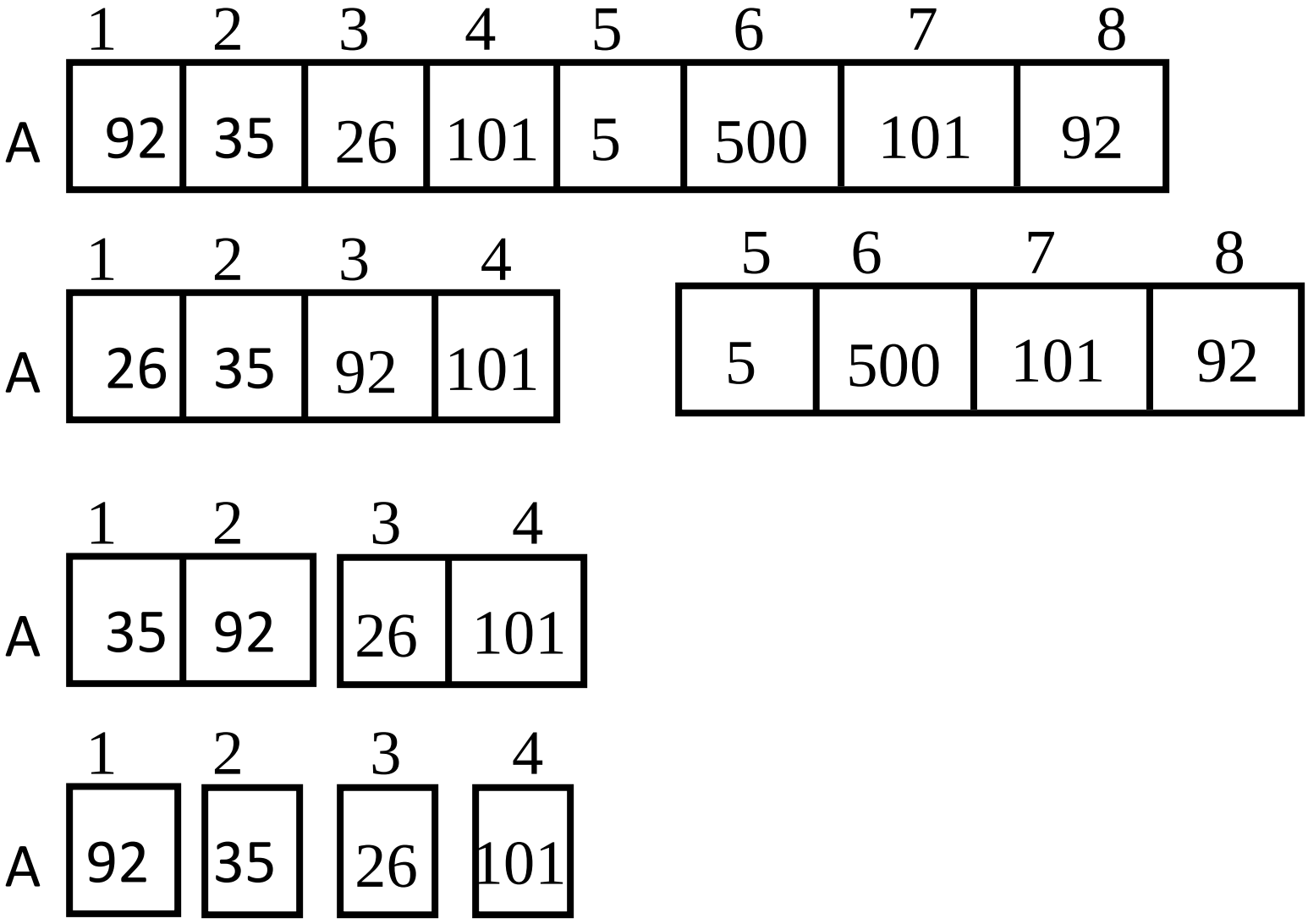
Merge Sort : Conquer (Level 3)



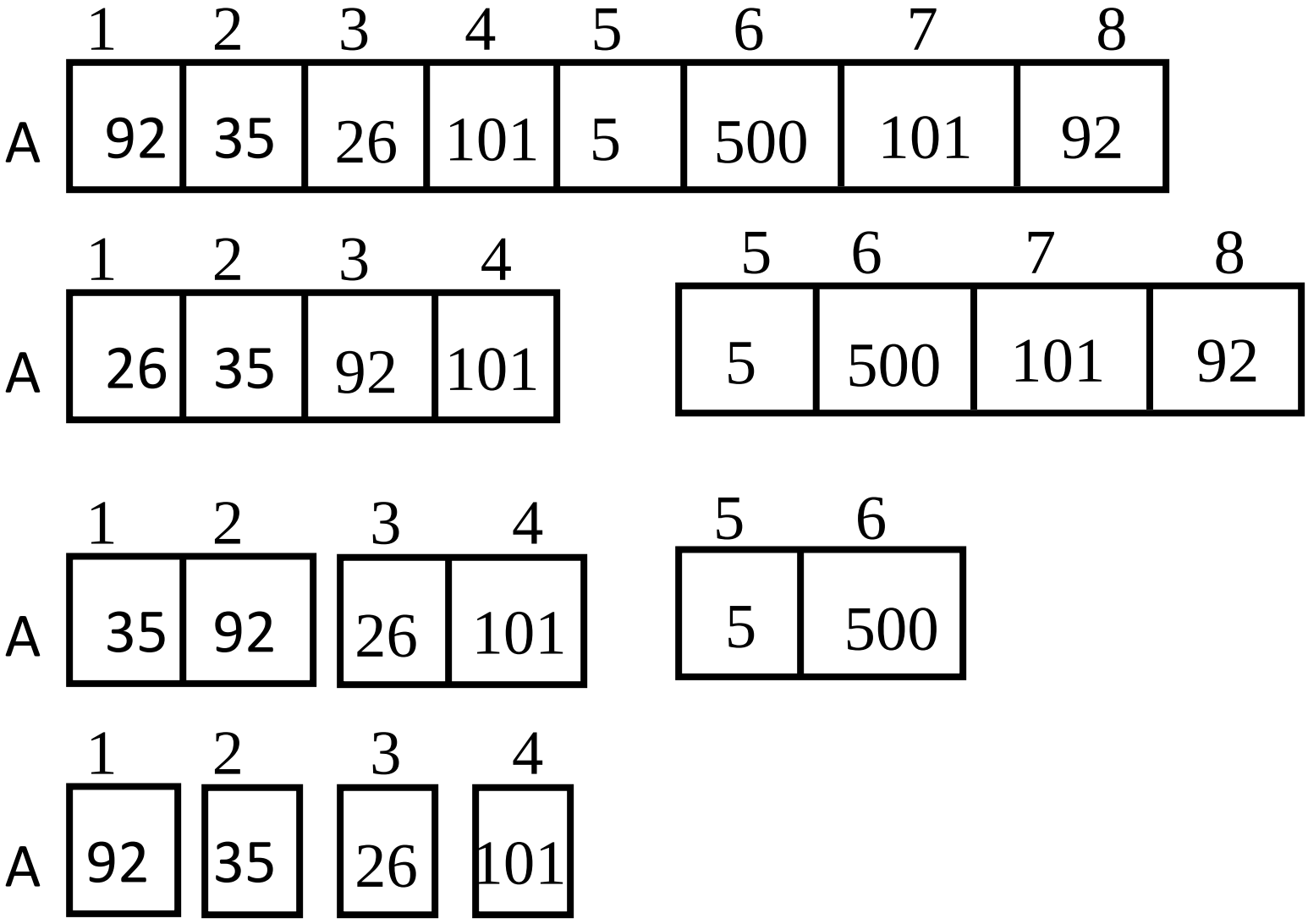
Merge Sort : Conquer (Level 2)



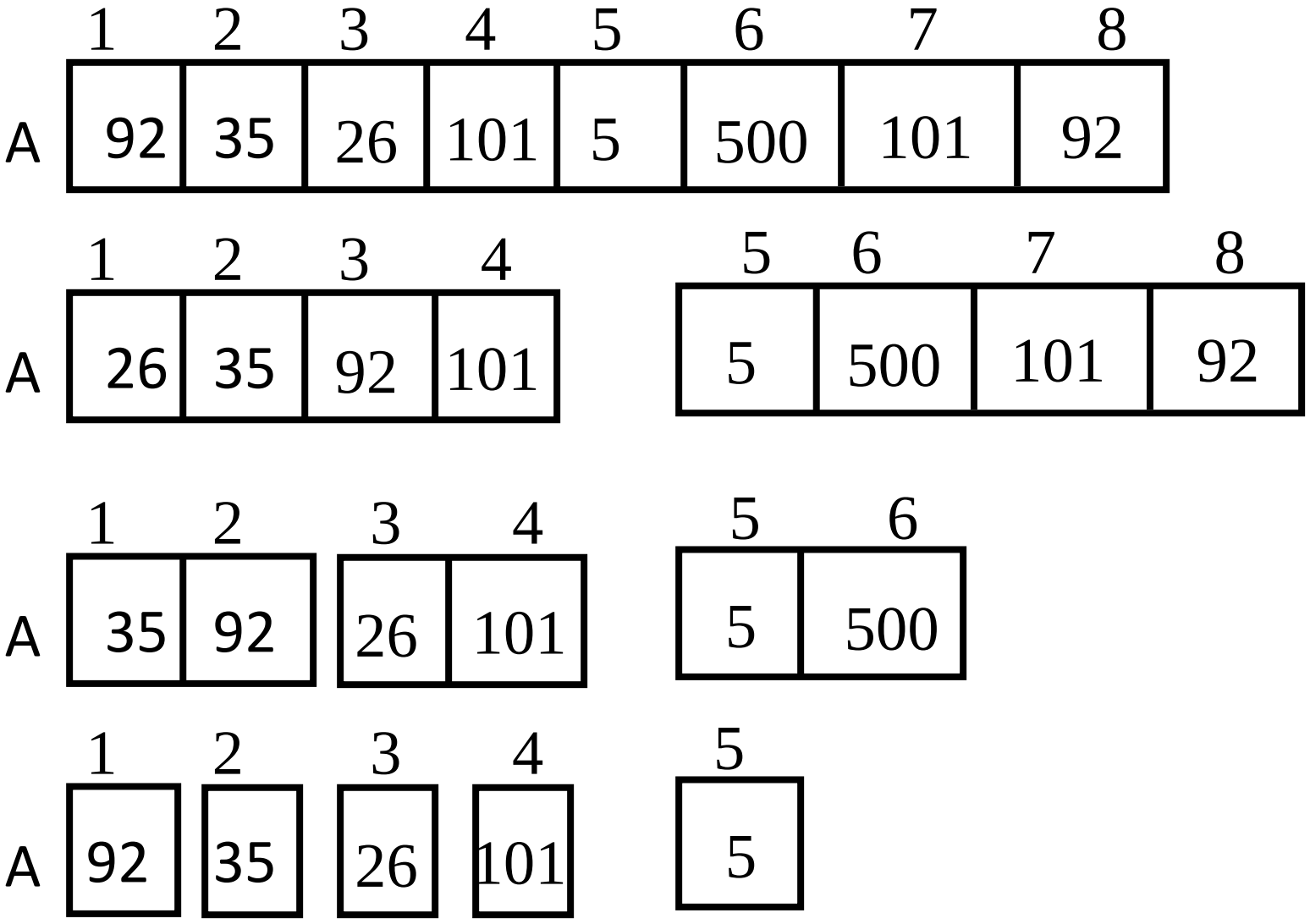
Merge Sort : Divide (Level 1)



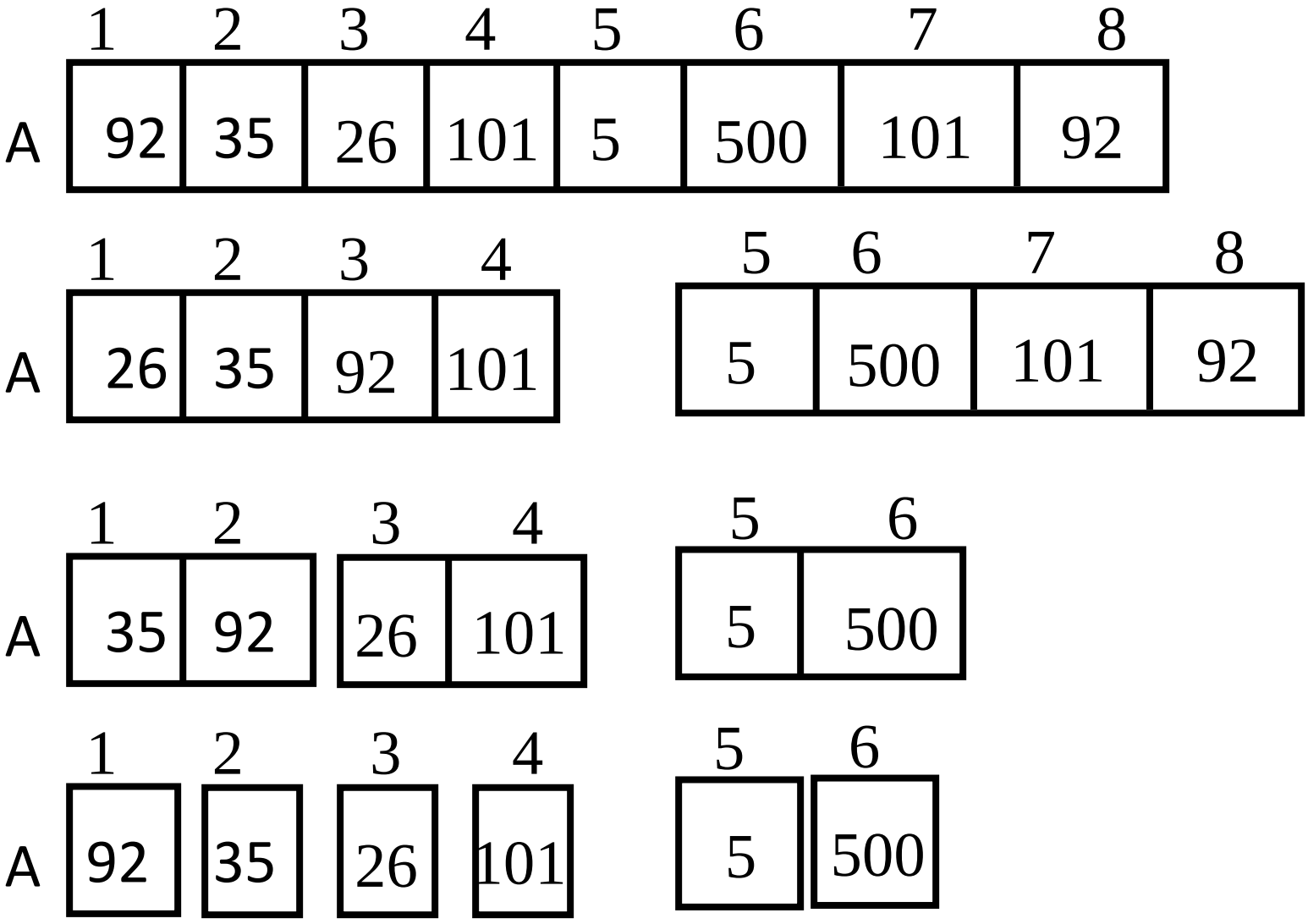
Merge Sort : Divide (Level 2)



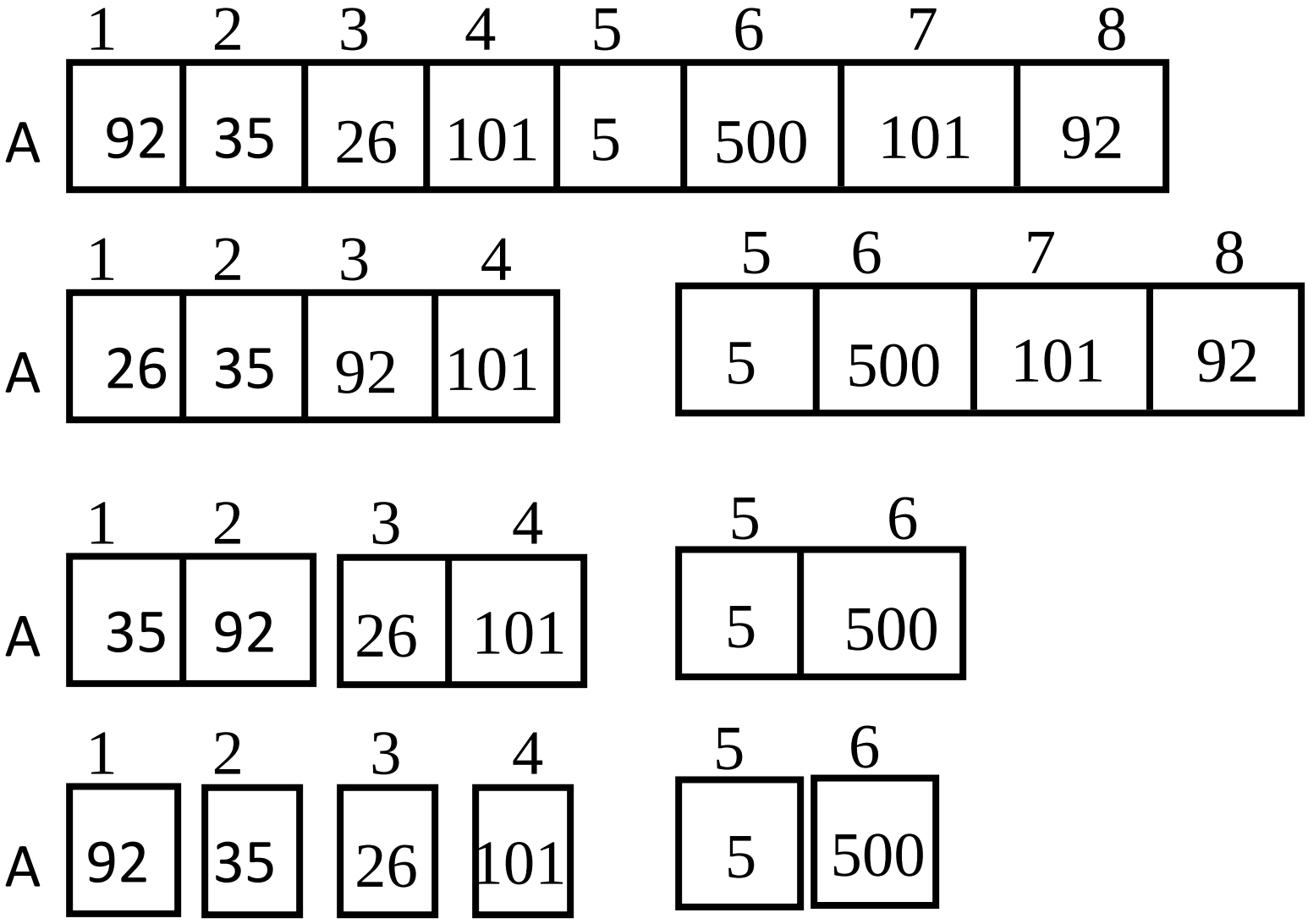
Merge Sort : Divide (Level 3) & Conquer (Level 4)



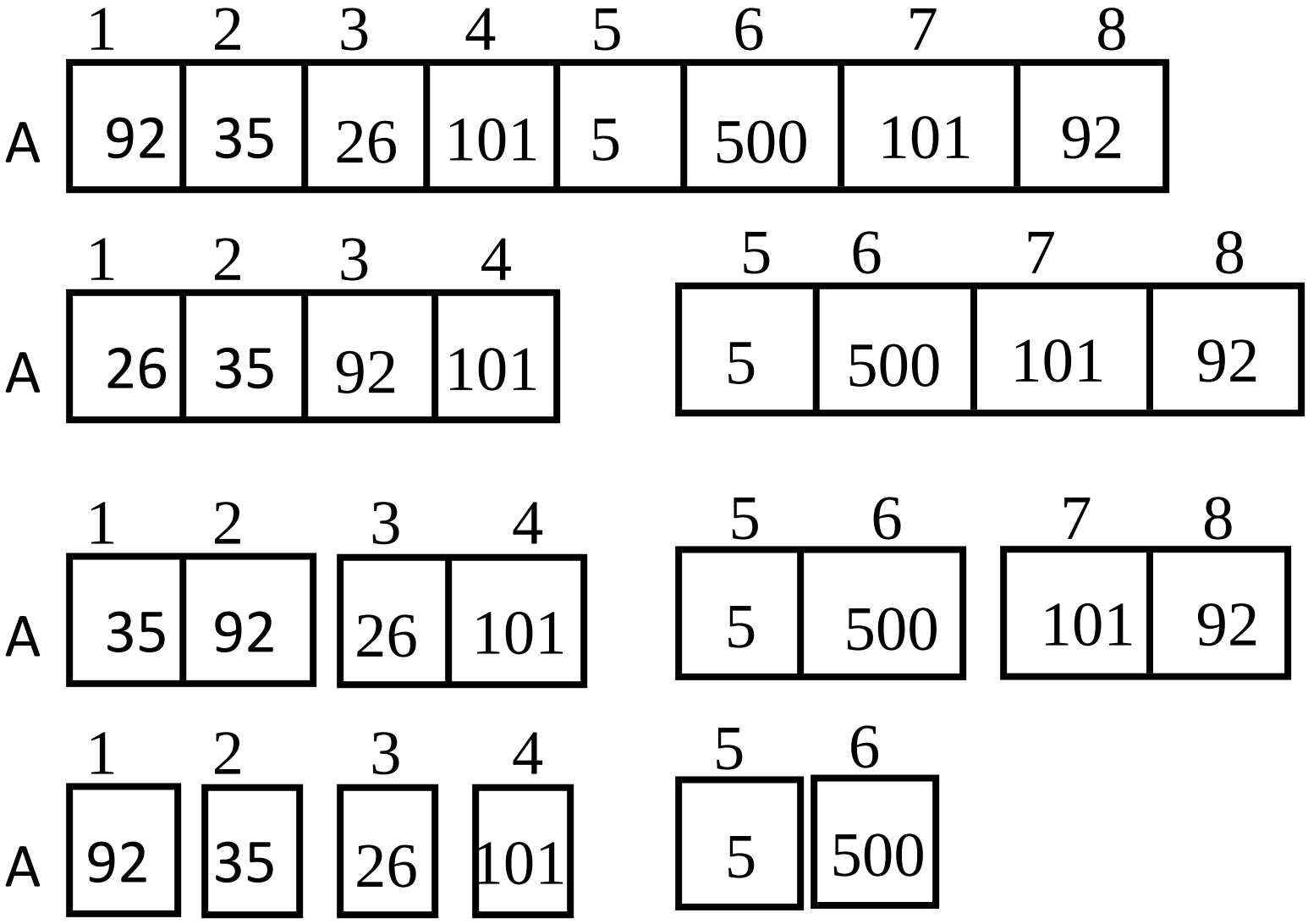
Merge Sort : Conquer (Level 4)



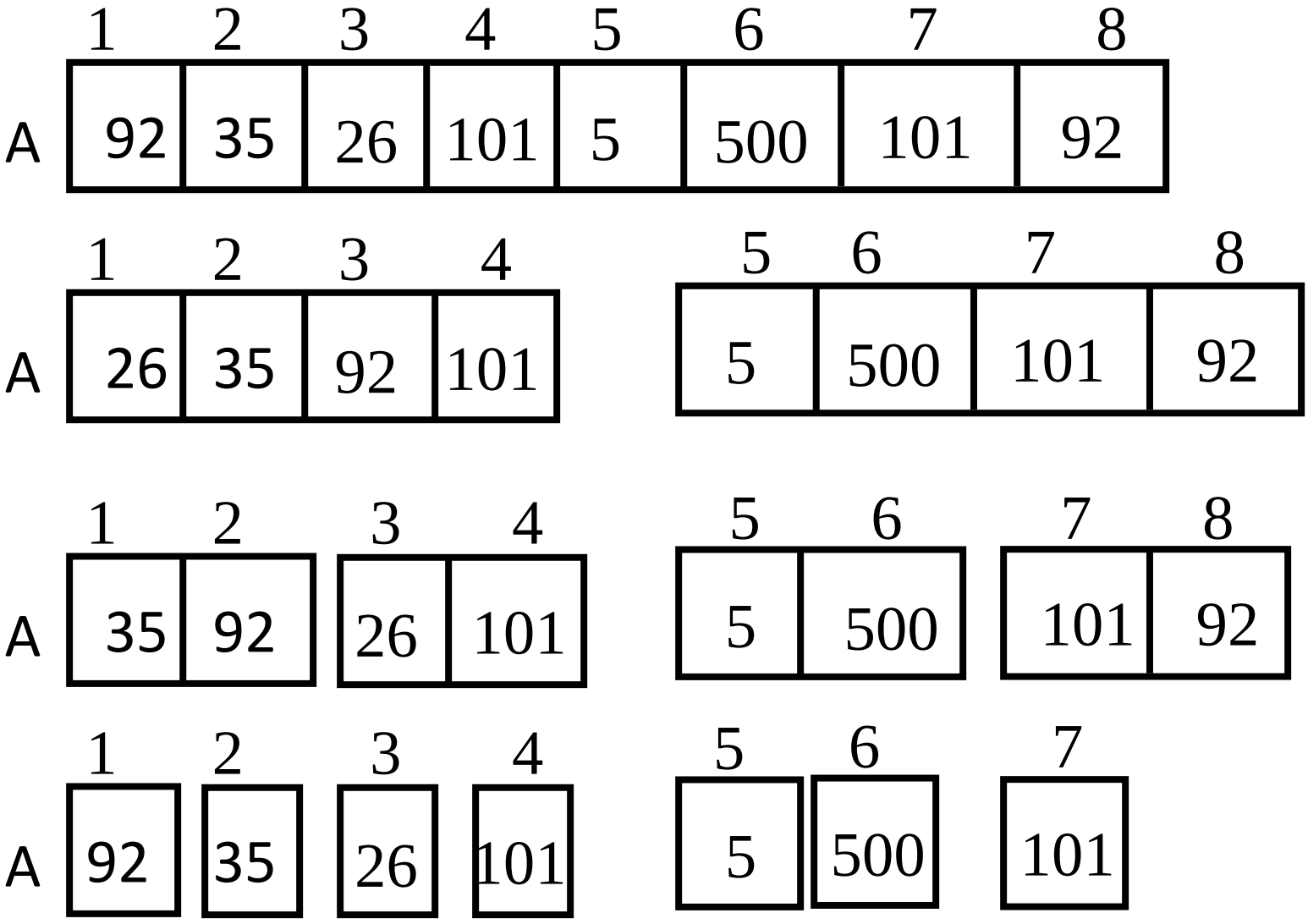
Merge Sort : Conquer (Level 3)



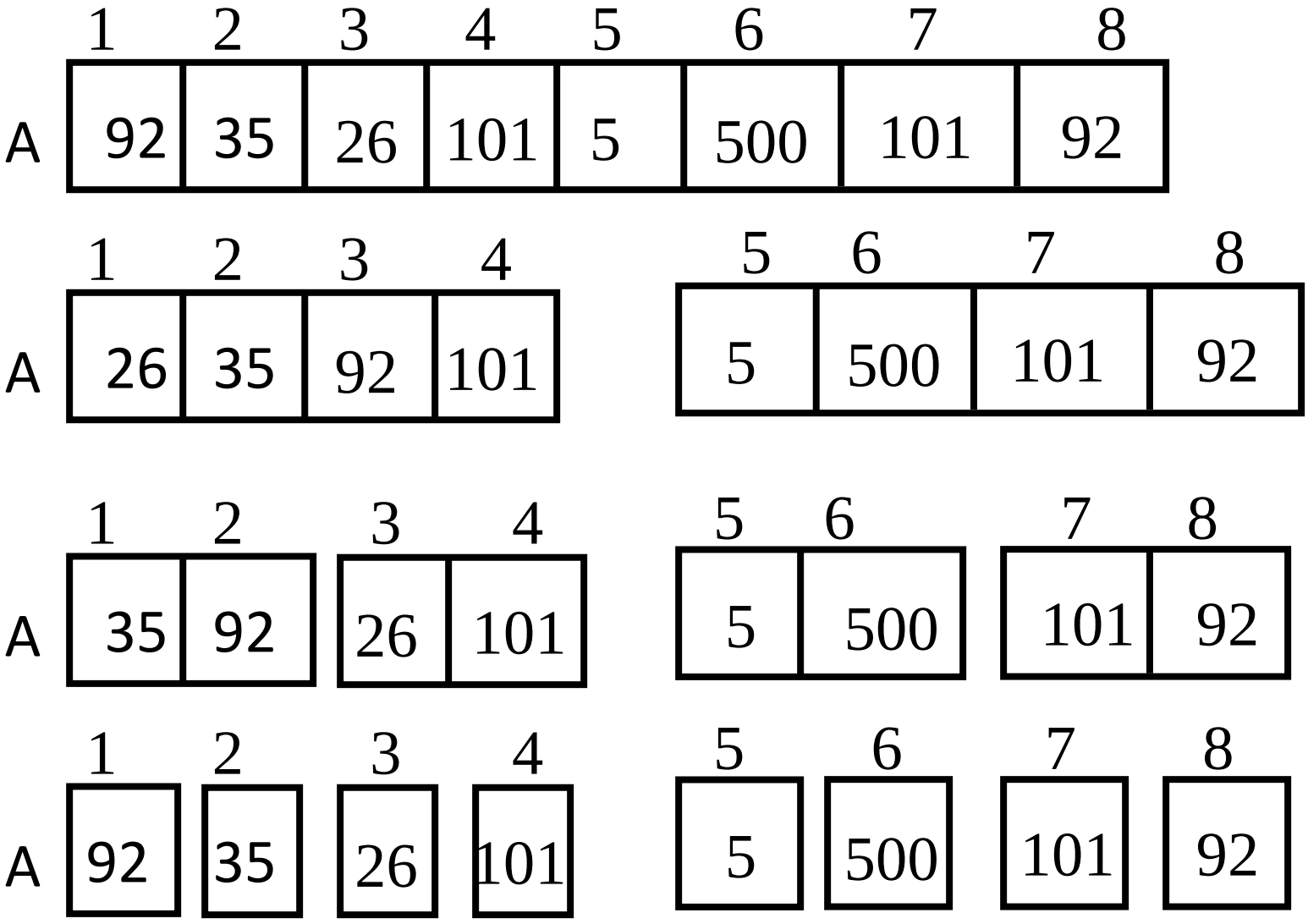
Merge Sort : Divide (Level 2)



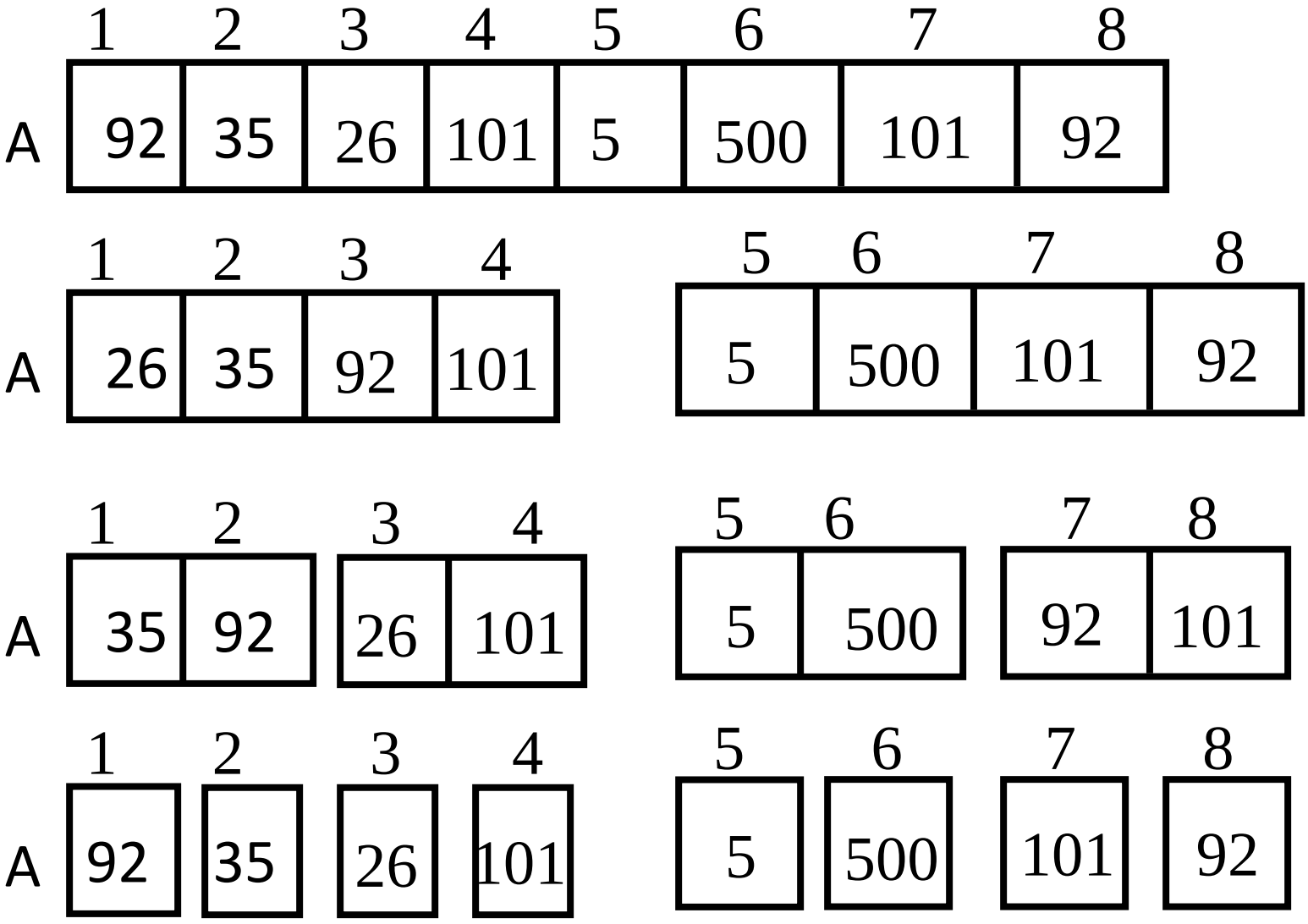
Merge Sort : Divide (Level 3) & Conquer (Level 4)



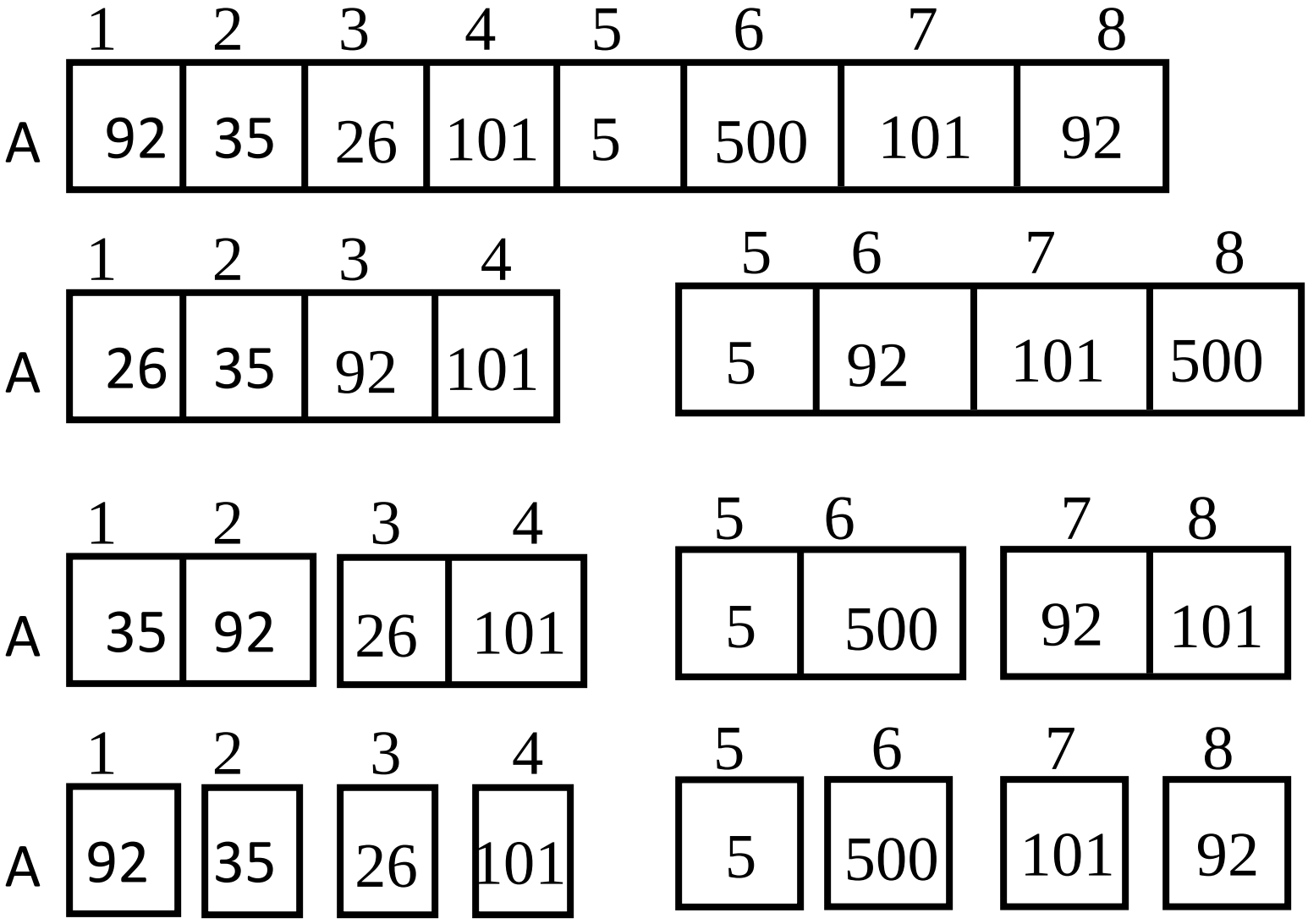
Merge Sort : Conquer (Level 4)



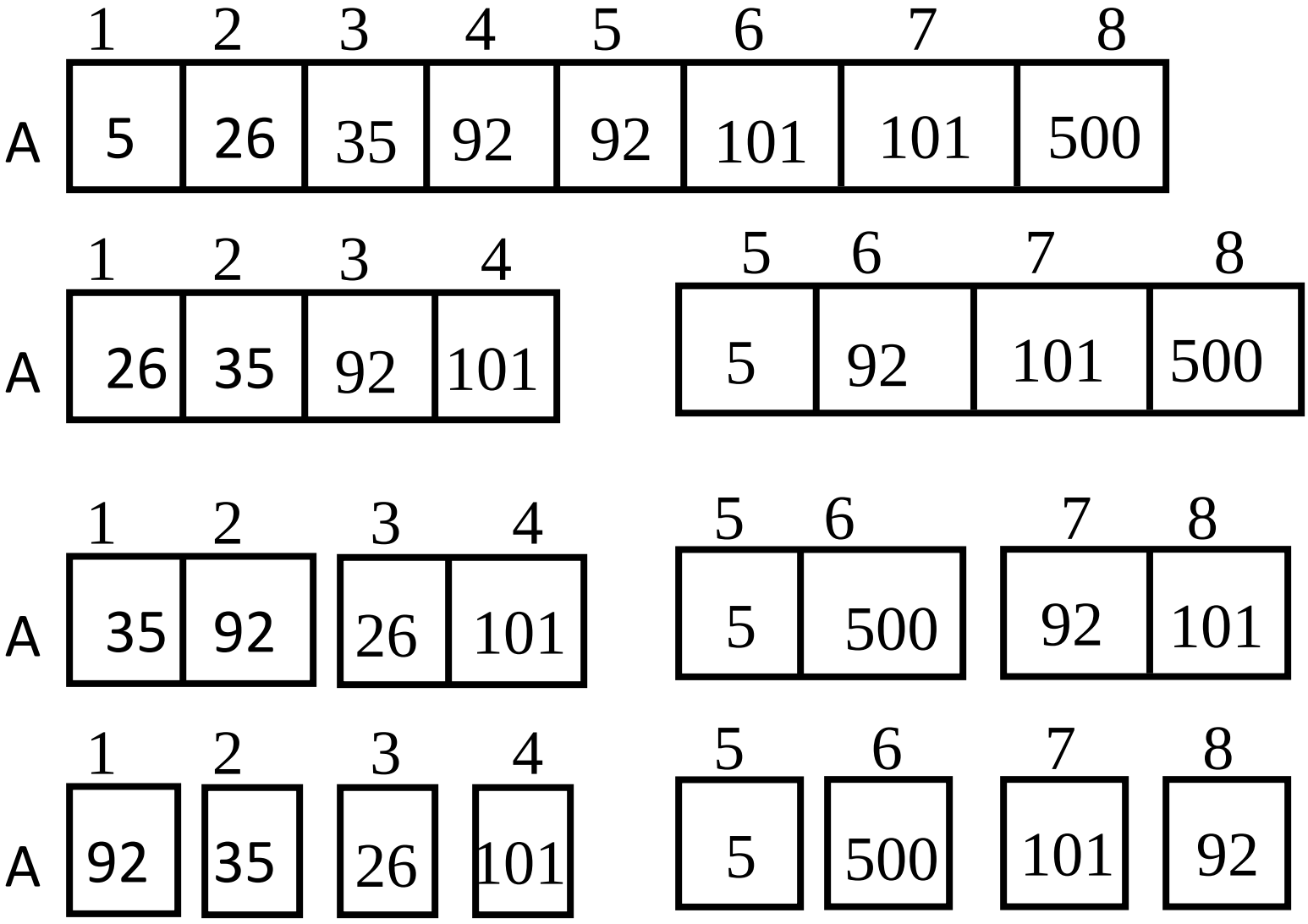
Merge Sort : Conquer (Level 3)



Merge Sort : Conquer (Level 2)



Merge Sort : Conquer (Level 1)



Merge Sort

```
void mergesort(int a[], int low, int high)
{
  if(low < high) {
        int mid = (high+low)/2;
        mergesort(a, low, mid); // Sort the left half
        mergesort(a, mid+1, high); // Sort the right half
        merge(a, low, mid, high); // Merge the sorted halves
  }
}
```

Time Complexity: Merge Sort

- T(n) = Time to merge sort array of size n = 2^k integers
- T(1) = 1

$T(n) = 2T(n/2) + cn$

$2T(n/2) = 2^2T(n/2^2) + cn$

$2^2T(n/2^2) = 2^3T(n/2^3) + cn$

.....

$2^{k-1}T(n/2^{k-1}) = 2^kT(n/2^k) + cn$

$T(n) = 2^kT(n/2^k) + cn.\ k$ (Summation of k equations)

$T(n) = nT(1) + c.n.\log_2 n = n + c.n.\log_2 n = O(n\log_2 n)$

Space Complexity: Merge Sort

- Space needed to solve merge sort
- Do not consider the space for input array
- Consider the extra space needed in terms of input size
- At the bottom level, an array of size 2 is required
- Immediately higher level, an array of size 4 is required
- In the next higher level, an array of size 8 is required
- …
- In the top level, an array of size $2^k = n$ is required
- Important to note: when array in one level is used/created, the arrays in the bottom levels have already been destroyed.
- Therefore, the space complexity = $O(\max(2, 4, 8, \ldots, 2^k = n)) = O(n)$