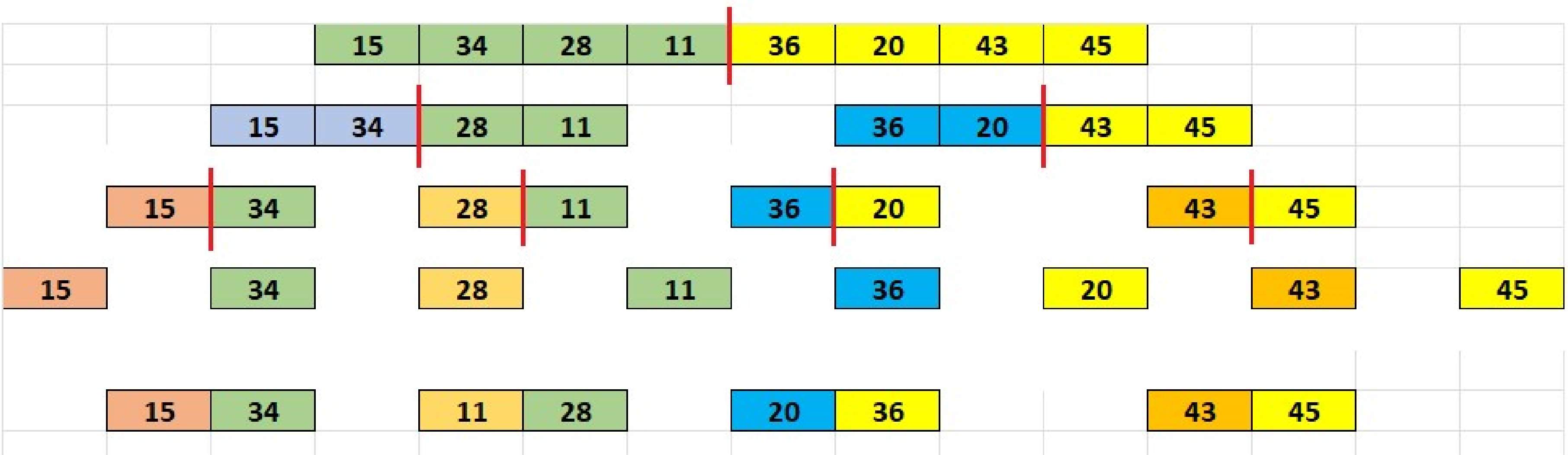
# Lecture 12

Merge Sort: Overview, Worst, Best & Average Case Analysis;

# Example – Merge_Sort

› It is based on Divide and Conquer
› It divides the array into equal halves and then combine them in a sorted manner



# Algorithm - Steps

**MERGE_SORT**(Low, High)

```
{ IF Low<High THEN
    {
    Mid=(Low + High)/2
    MERGE_SORT(Low, Mid)
    MERGE_SORT(Mid+1, High)
    MERGE_SORT(Low, Mid, High)
    }
}
```

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

**MERGE**(A, B, m, n)

```
    { i=j=k=1
    WHILE (i<=m && j<=n)
        {
        IF (A[i]<B[j] THEN
            C[k++]=A[i++]
        ELSE
            C[k++]=B[j++]
        }
        FOR (;i<=m; i++)
            C[k++]=A[i]
        FOR (;j<=n; j++)
            C[k++]=B[j]
    }
```

# Few Steps of Execution of Algorithm

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

Using Master Theorem
a=2, b=2

$f(n)=n^{\log_2 2}=n$

$T(n)=\Theta(n \log n)$ – **Best, Average & Worst-Case Complexity**

**Space Complexity: O(1)**
**No extra variable be required.**

| i | A | j | B | k | C | condition | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 5 | | 2 | A[i}<B[j] | TRUE | i++ | k++ |
| | 8 | | 9 | | | | | | |
| | 15 | | 12 | | | | | | |
| | 18 | | 17 | | | | | | |

| i | A | j | B | k | C | condition | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | | 5 | 1 | 2 | A[i}<B[j] | TRUE | i++ | k++ |
| 2 | 8 | 2 | 9 | 2 | 5 | A[i]<B[j] | FALSE | j++ | K++ |
| | 15 | | 12 | | | | | | |
| | 18 | | 17 | | | | | | |

| i | A | j | B | k | C | condition | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | | 5 | 1 | 2 | A[i}<B[j] | TRUE | i++ | k++ |
| | 8 | 2 | 9 | 2 | 5 | A[i]<B[j] | FALSE | j++ | K++ |
| 3 | 15 | | 12 | 3 | 8 | A[i]<B[j] | TRUE | i++ | K++ |
| | 18 | | 17 | | | | | | |

# *Divide-and-Conquer*

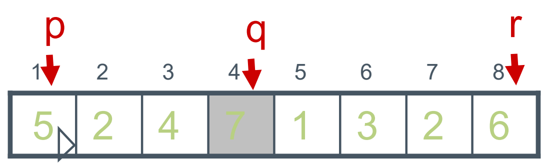
- ***Divide*** *the problem into a number of sub-problems*
  - *Similar sub-problems of smaller size*
- ***Conquer*** *the sub-problems*
  - *Solve the sub-problems recursively*
  - *Sub-problem size small enough $\Rightarrow$ solve the problems in straightforward manner*
- ***Combine*** *the solutions of the sub-problems*
  - *Obtain the solution for the original problem*

# *Merge Sort Approach*

- *To sort an array A[p . . r]:*
- ***Divide***
  - *Divide the n-element sequence to be sorted into two subsequences of n/2 elements each*
- ***Conquer***
  - *Sort the subsequences recursively using merge sort*
  - *When the size of the sequences is 1 there is nothing more to do*
- ***Combine***
  - *Merge the two sorted subsequences*

# Merge Sort

*Alg.:* *MERGE-SORT(A, p, r)*




**if** $p < r$ ▷ *Check for base case*

**then** $q \leftarrow \lfloor (p + r)/2 \rfloor$ ▷ *Divide*

*MERGE-SORT(A, p, q)* ▷ *Conquer*

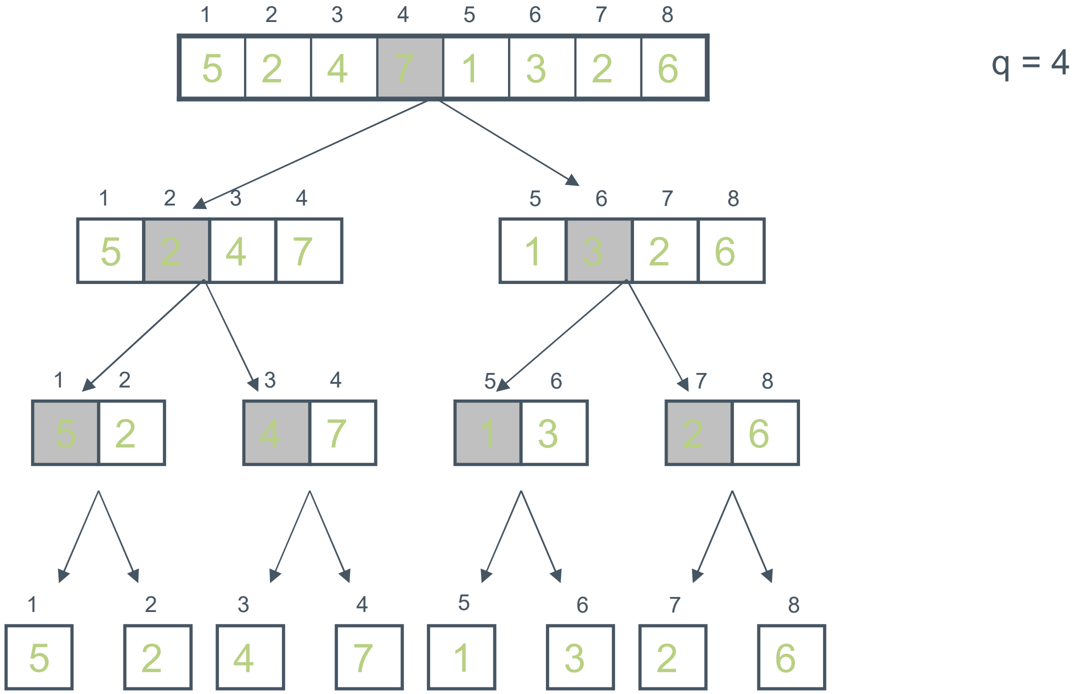
*MERGE-SORT(A, q + 1, r)* ▷ *Conquer*

*MERGE(A, p, q, r)* ▷ *Combine*

- *Initial call: MERGE-SORT(A, 1, n)*

# Example – n Power of 2

*Divide*

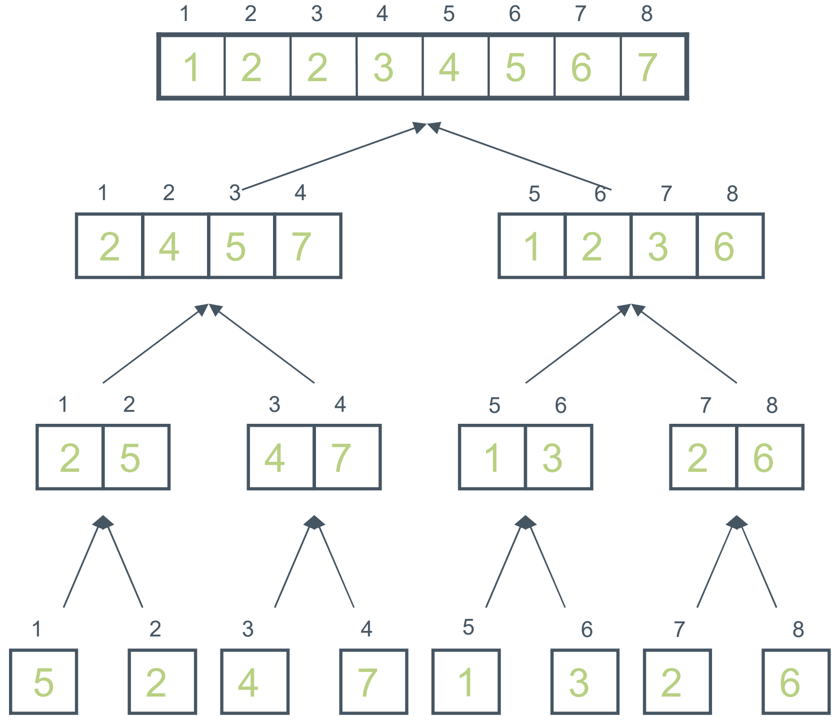


# *Example – n Power of 2*

*Conquer*

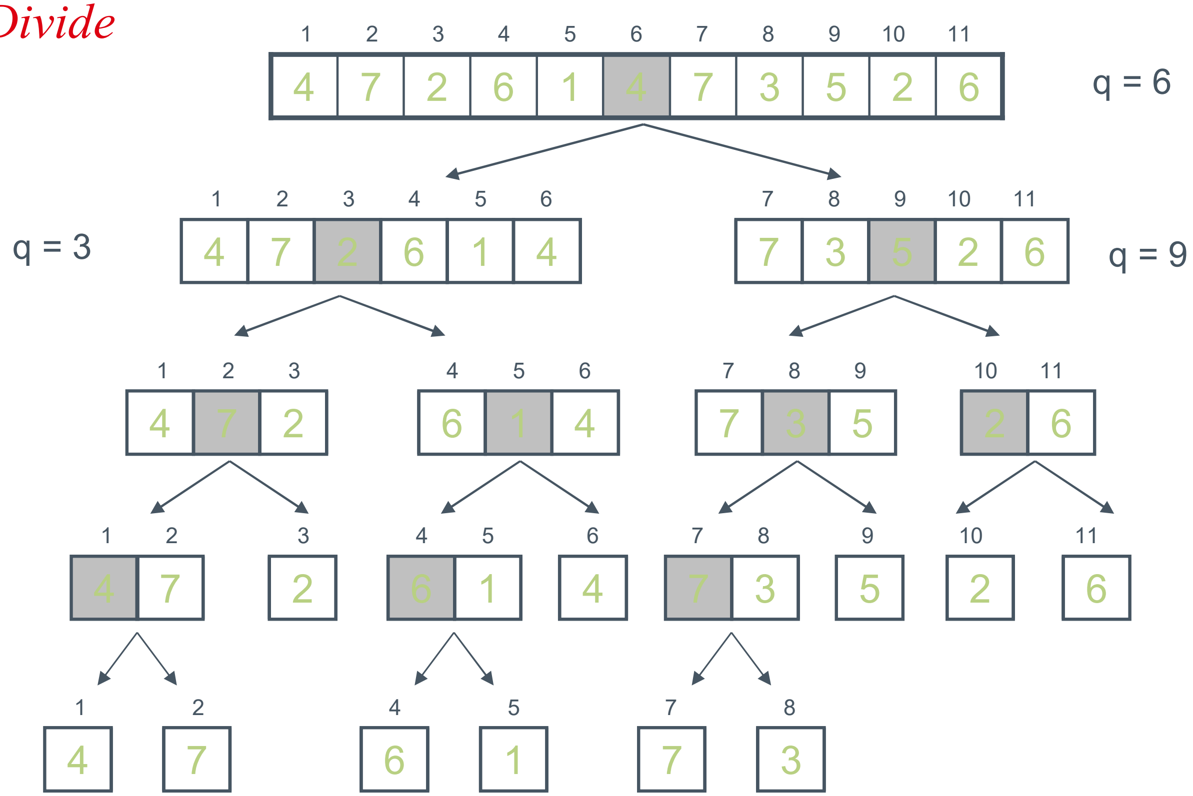
*and*

*Merge*



# *Example – n Not a Power of 2*

*Divide*

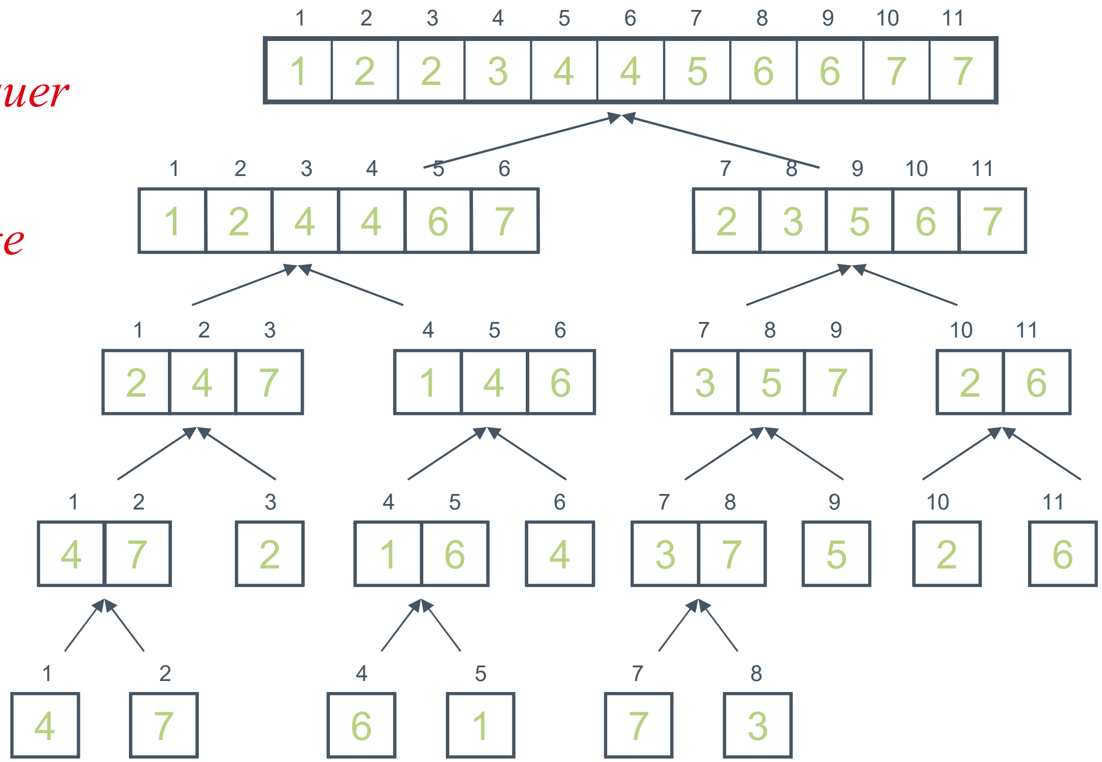


# *Example – n Not a Power of 2*

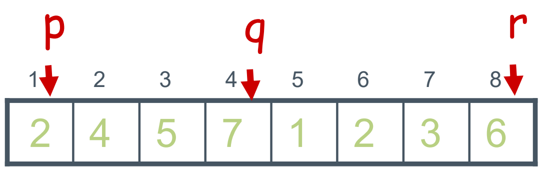
*Conquer*

*and*

*Merge*



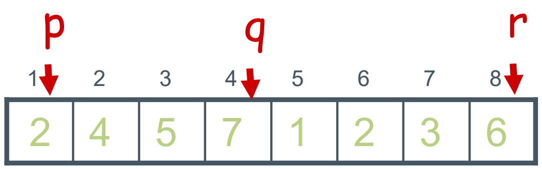
# *Merging*

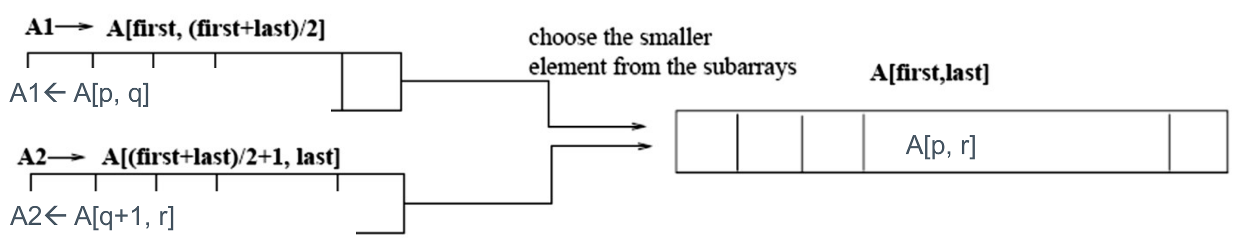



- ***Input:*** *Array A and indices p, q, r such that* $p \leq q < r$
  - *Subarrays A[p . . q] and A[q + 1 . . r] are sorted*
- ***Output:*** *One single sorted subarray A[p . . r]*

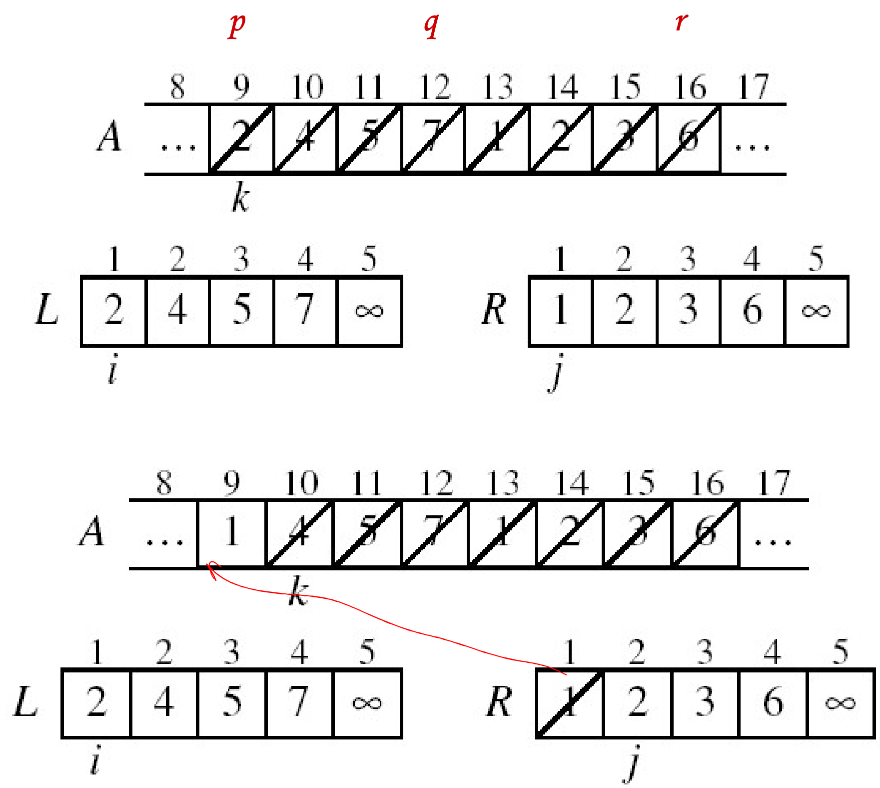
# Merging

- *Idea for merging:*
  - *Two piles of sorted cards*
    - *Choose the smaller of the two top cards*
    - *Remove it and place it in the output pile*
  - *Repeat the process until one pile is empty*
  - *Take the remaining input pile and place it face-down onto the output pile*

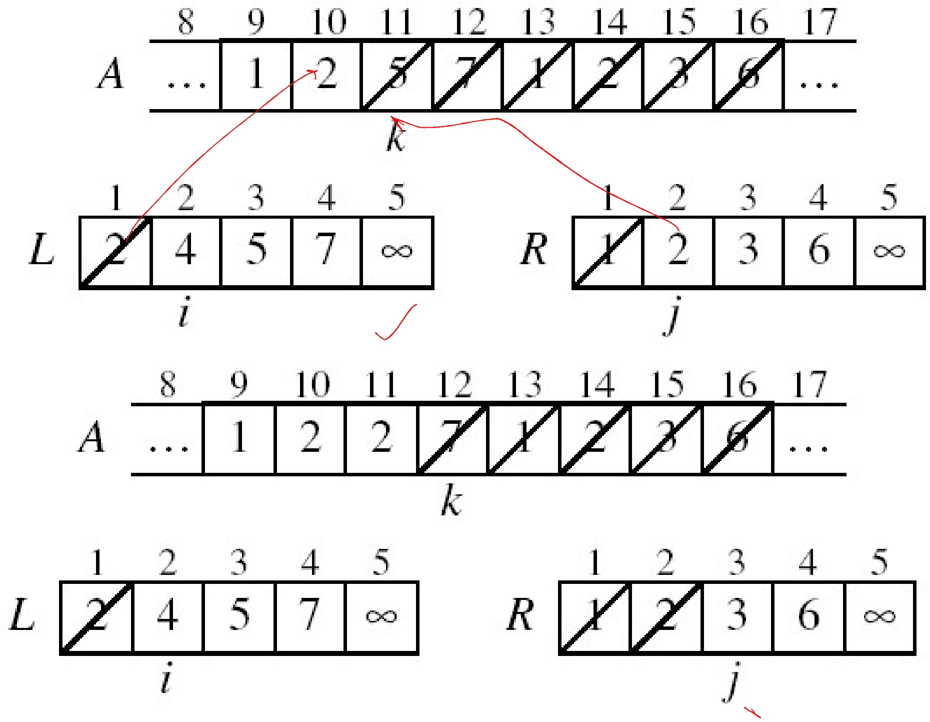


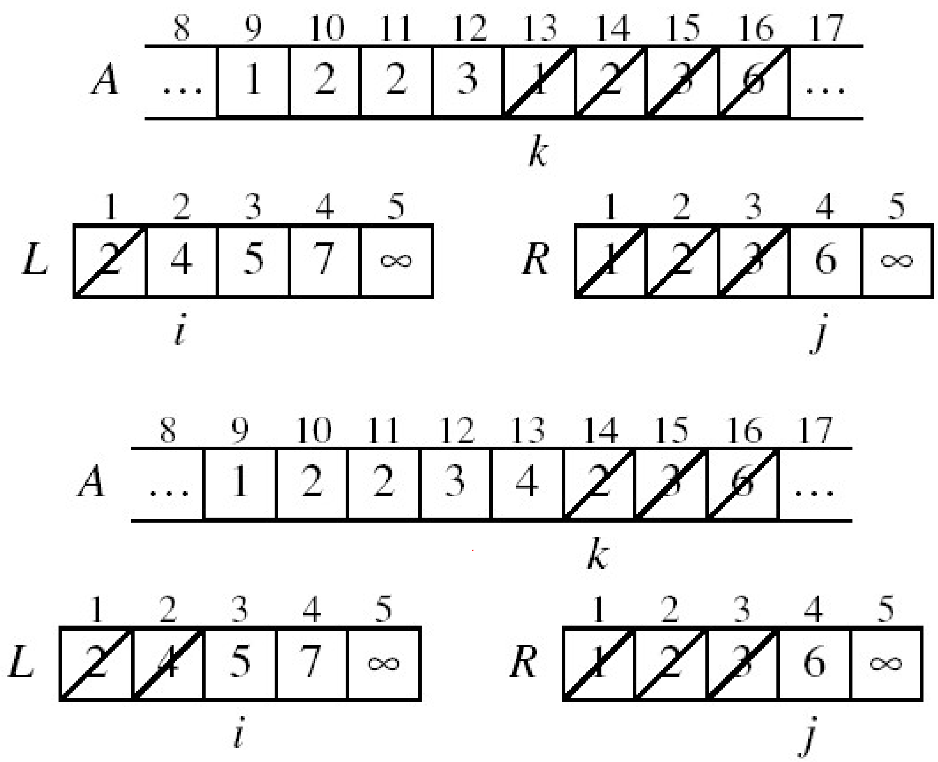
# *Example: MERGE(A, 9, 12, 16)*



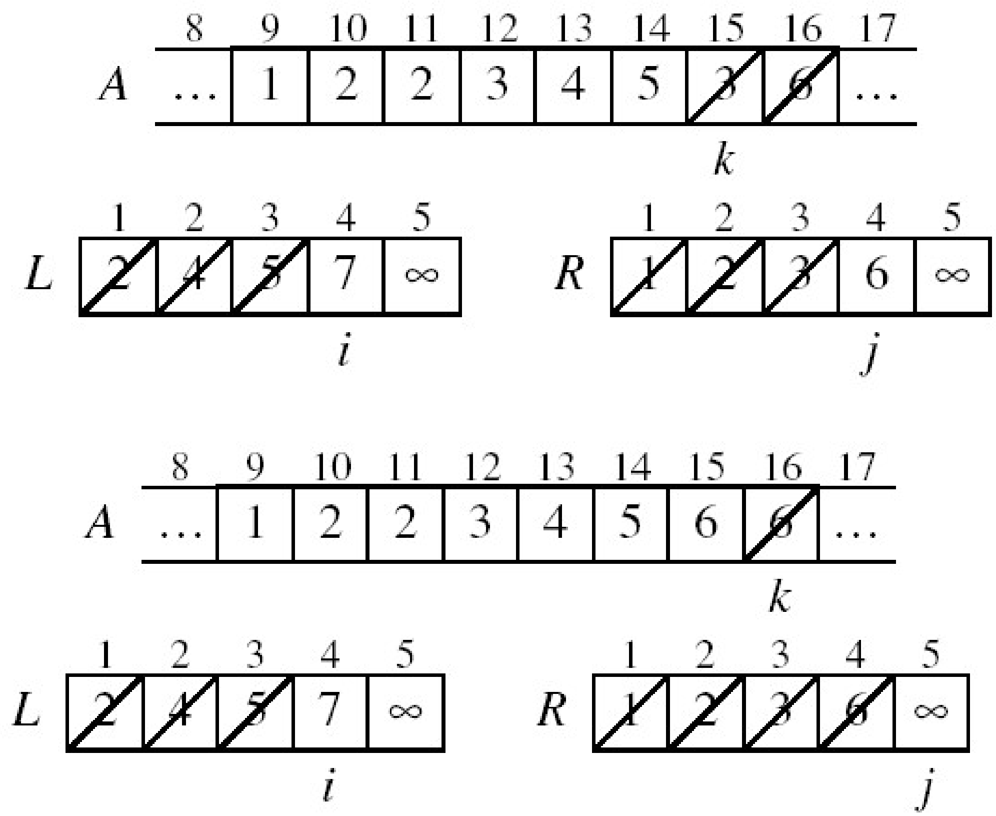
# *Example: MERGE(A, 9, 12, 16)*



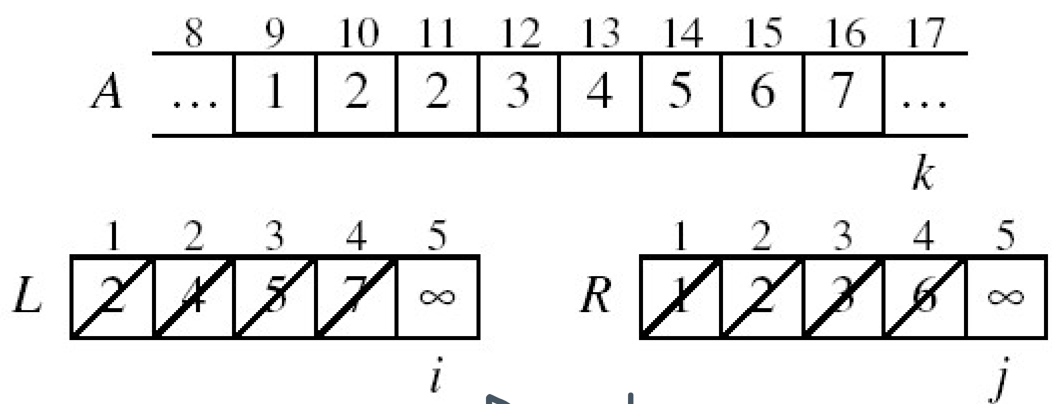
# *Example (Cont !!!)*



# *Example (Cont !!!)*



# *Example (Cont !!!)*

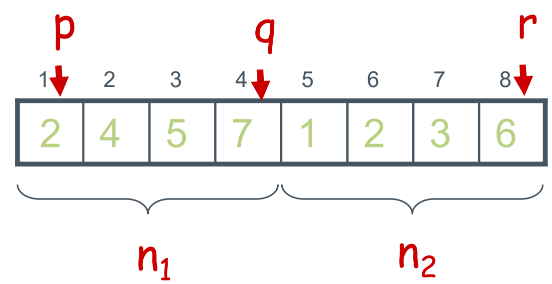



Done!

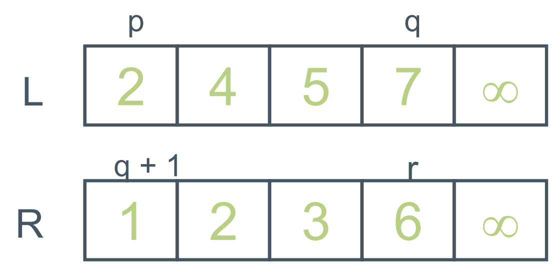
# *Merge - Pseudo code*

*Alg.: MERGE(A, p, q, r)*

1. Compute $n_1$ and $n_2$
2. Copy the first $n_1$ elements into $L[1 \,..\, n_1 + 1]$ and the next $n_2$ elements into $R[1 \,..\, n_2 + 1]$

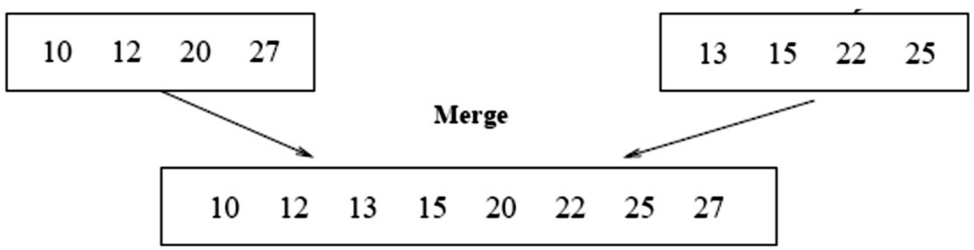



1. $L[n_1 + 1] \leftarrow \infty; \quad R[n_2 + 1] \leftarrow \infty$
2. $i \leftarrow 1; \quad j \leftarrow 1$
3. **for** $k \leftarrow p$ **to** $r$
4. **do if** $L[\,i\,] \leq R[\,j\,]$
5. **then** $A[k] \leftarrow L[\,i\,]$
6. $i \leftarrow i + 1$
7. **else** $A[k] \leftarrow R[\,j\,]$
8. $j \leftarrow j + 1$



# Running Time of Merge

- *Initialization (copying into temporary arrays):*
  - $\Theta(n_1 + n_2) = \Theta(n)$
- *Adding the elements to the final array:*
  - *- n iterations, each taking constant time* $\Rightarrow \Theta(n)$
- *Total time for Merge:*
  - $\Theta(n)$



# *Analyzing Divide-and Conquer Algorithms*

- *The recurrence is based on the three steps of the paradigm:*
  - *T(n) – running time on a problem of size n*
  - ***Divide** the problem into **a** subproblems, each of size **n/b**: takes D(n)*
  - ***Conquer** (solve) the subproblems aT(n/b)*
  - ***Combine** the solutions C(n)*

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

# *MERGE-SORT Running Time*

- ***Divide:***
  - *compute q as the average of p and r:* $D(n) = \Theta(1)$
- ***Conquer:***
  - *recursively solve 2 subproblems, each of size n/2* $\Longrightarrow 2T(n/2)$
- ***Combine:***
  - *MERGE on an n-element subarray takes* $\Theta(n)$ *time* $\Longrightarrow C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

# Solve the Recurrence

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

***Use Master's Theorem:***

*Compare n with f(n) = cn*

*Case 2: T(n) =* $\Theta$*(nlgn)*

## ***Merge Sort - Discussion***

- *Running time insensitive of the input*

- *Advantages:*
  - *Guaranteed to run in $\Theta(n \lg n)$*

- *Disadvantage*
  - *Requires extra space $\approx N$*

# Thank You!!!

Have a good day