# Analysis of Algorithms
# Merge Sort

Dr. Ashish Sharma

# Sorting

- Insertion sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Best case: $\Theta(n)$
  - Worst case: $\Theta(n^2)$

- Bubble Sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Running time: $\Theta(n^2)$

# Sorting

- Selection sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Running time: $\Theta(n^2)$
- Merge Sort
  - Design approach: divide and conquer
  - Sorts in place: No
  - Running time: *Let's see*!!

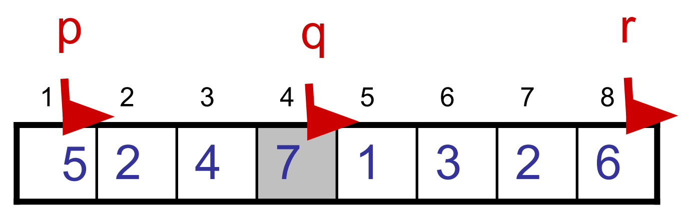
# Divide-and-Conquer

- **Divide** the problem into a number of subproblems
  - Similar sub-problems of smaller size
- **Conquer** the sub-problems
  - Solve the sub-problems recursively
  - Sub-problem size small enough ⇒ solve the problems in straightforward manner
- **Combine** the solutions to 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

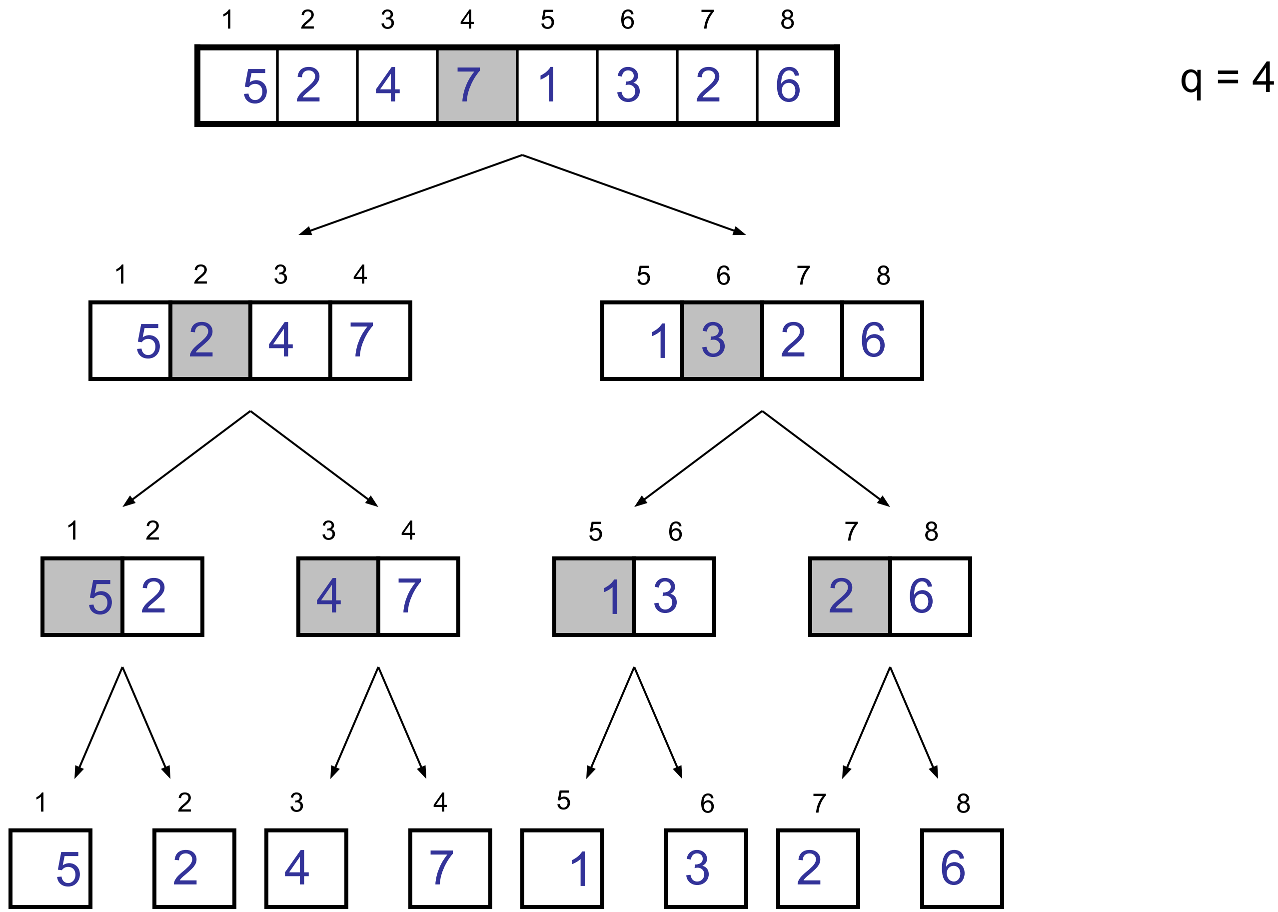
MERGE-SORT($A$, q + 1, r) ▷ Conquer

MERGE($A$, p, q, r) ▷ Combine

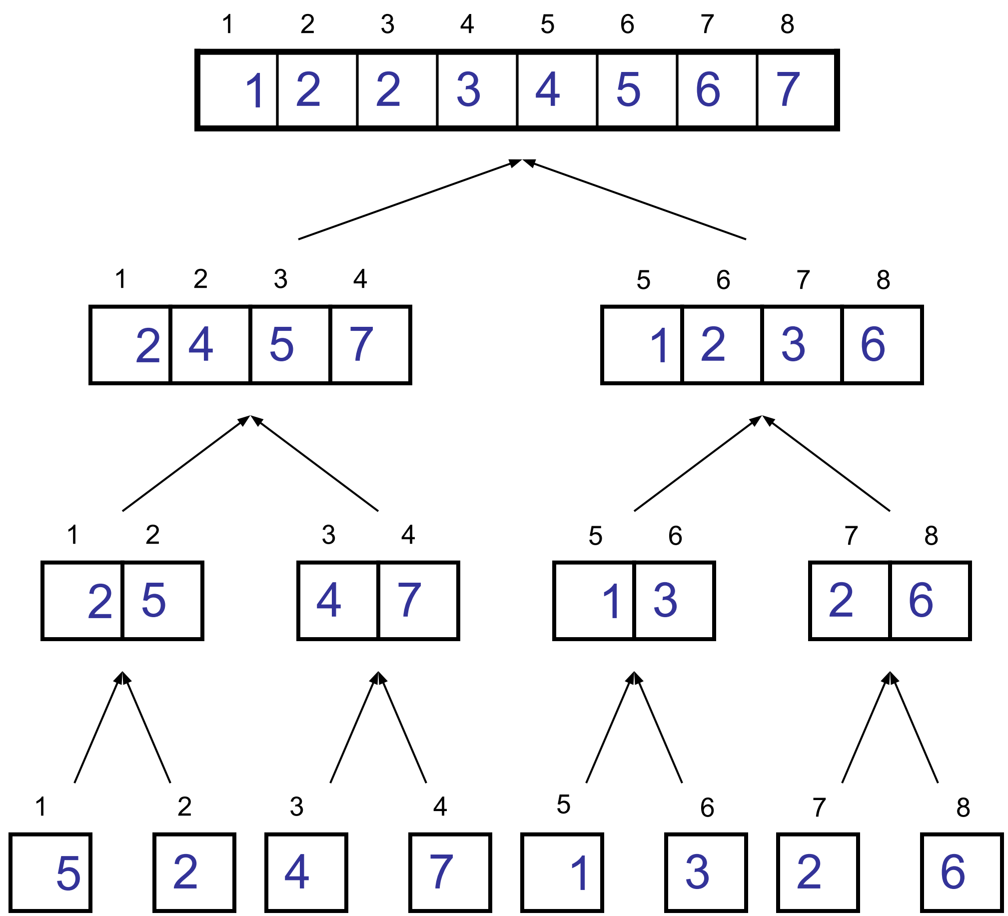
- Initial call: MERGE-SORT($A$, 1, $n$)

# Example – $n$ Power of 2

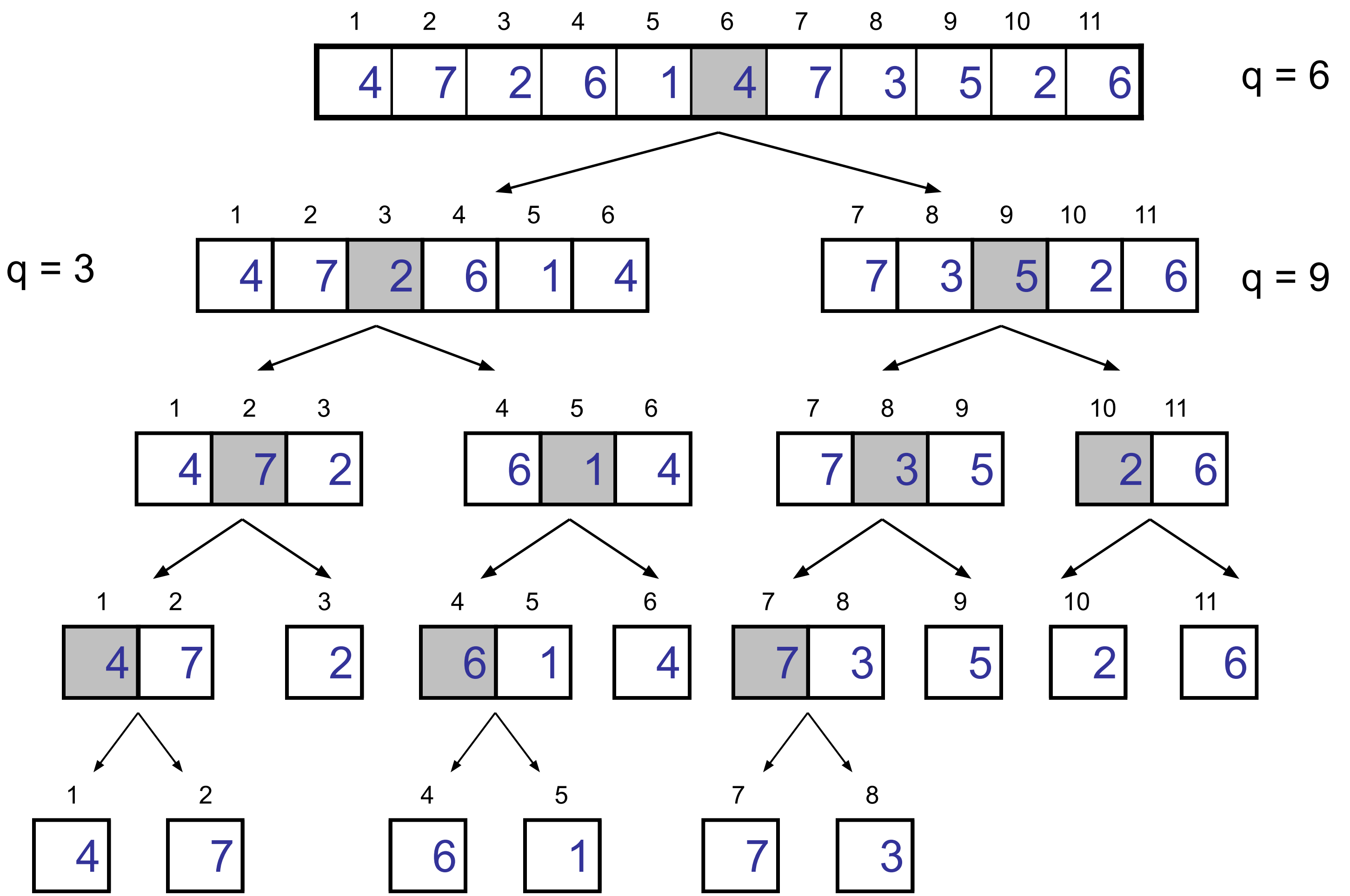
Example



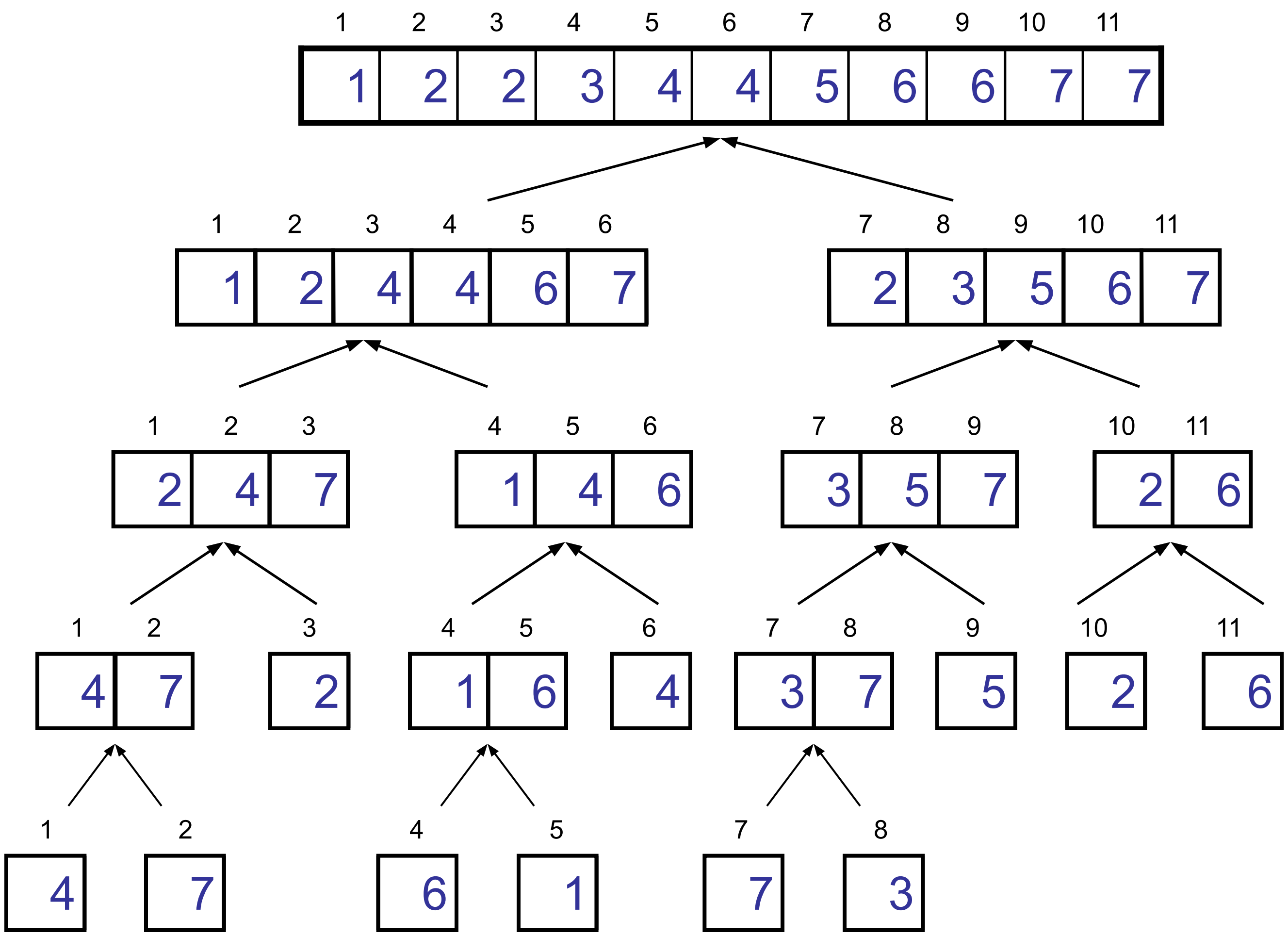
# Example – $n$ Power of 2



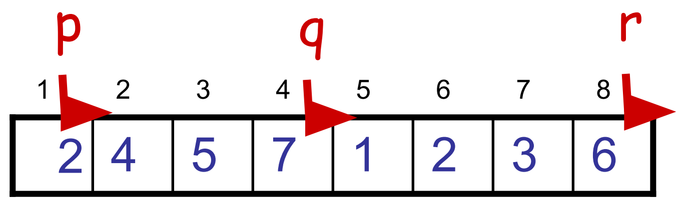
# Example – *n* Not a Power of 2



# Example – $n$ Not a Power of 2



# Merging




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

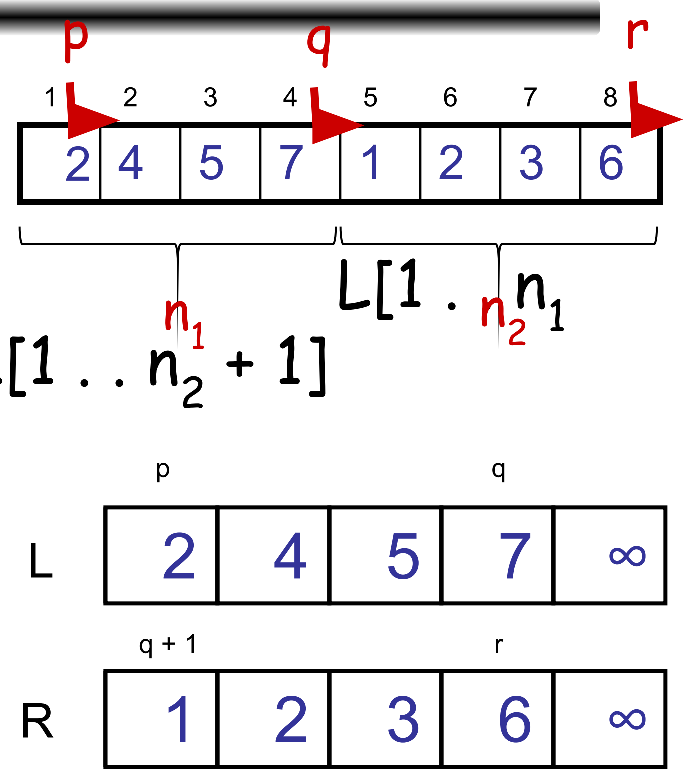
# Merging

- 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

# Merge - Pseudocode

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



# 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 $C(n)$

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

# 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 $n/2$
    $\Rightarrow 2T(n/2)$
- **Combine:**
  - MERGE on an $n$-element subarray takes $\Theta(n)$ time
    $\Rightarrow C(n) = \Theta(n)$

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

# 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: $T(n) = \Theta(n\lg n)$