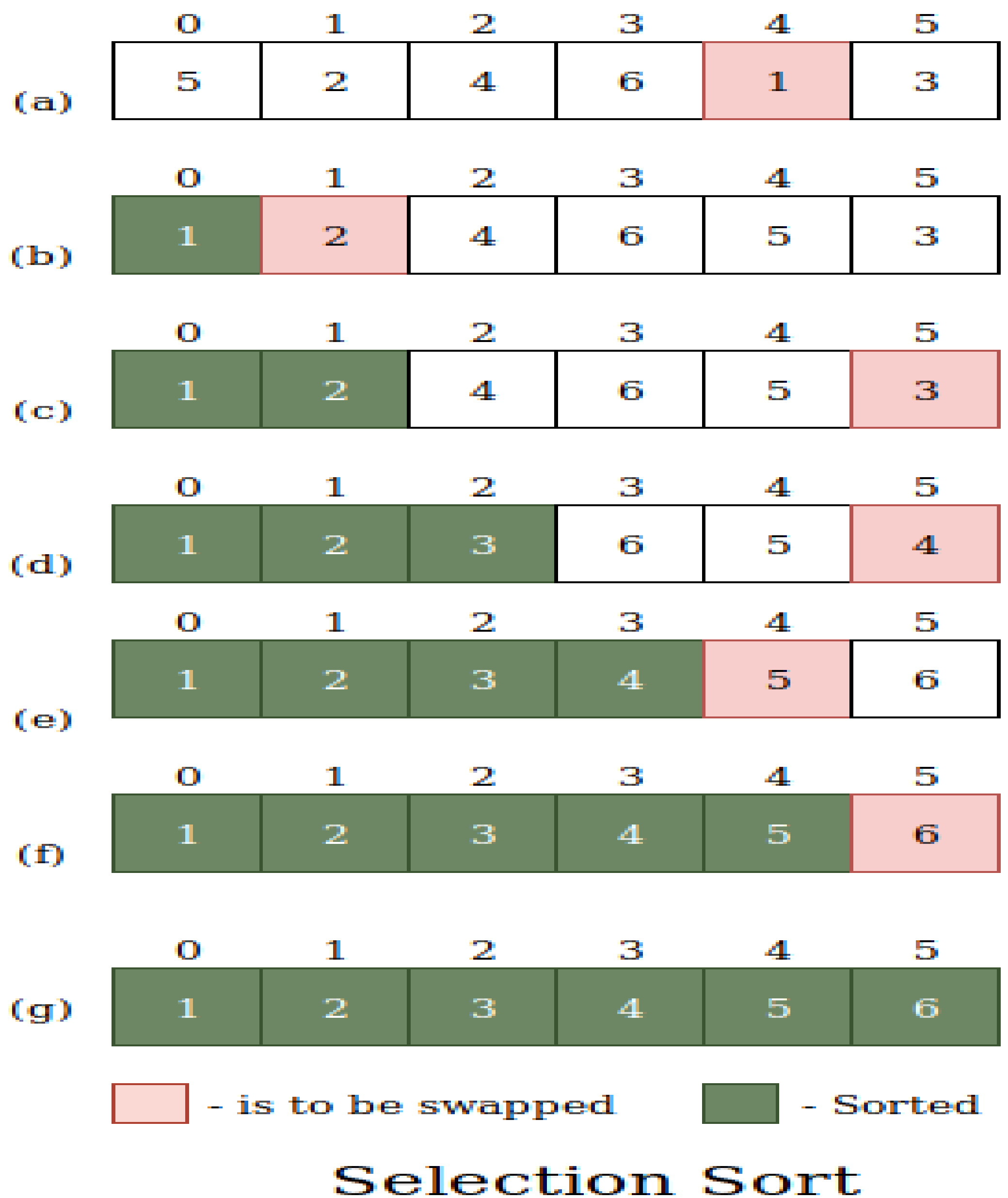
# Design and Analysis of Algorithms

## Lecture-2

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

# Some other sorting algorithms

- **Selection Sort**



# Some other sorting algorithms

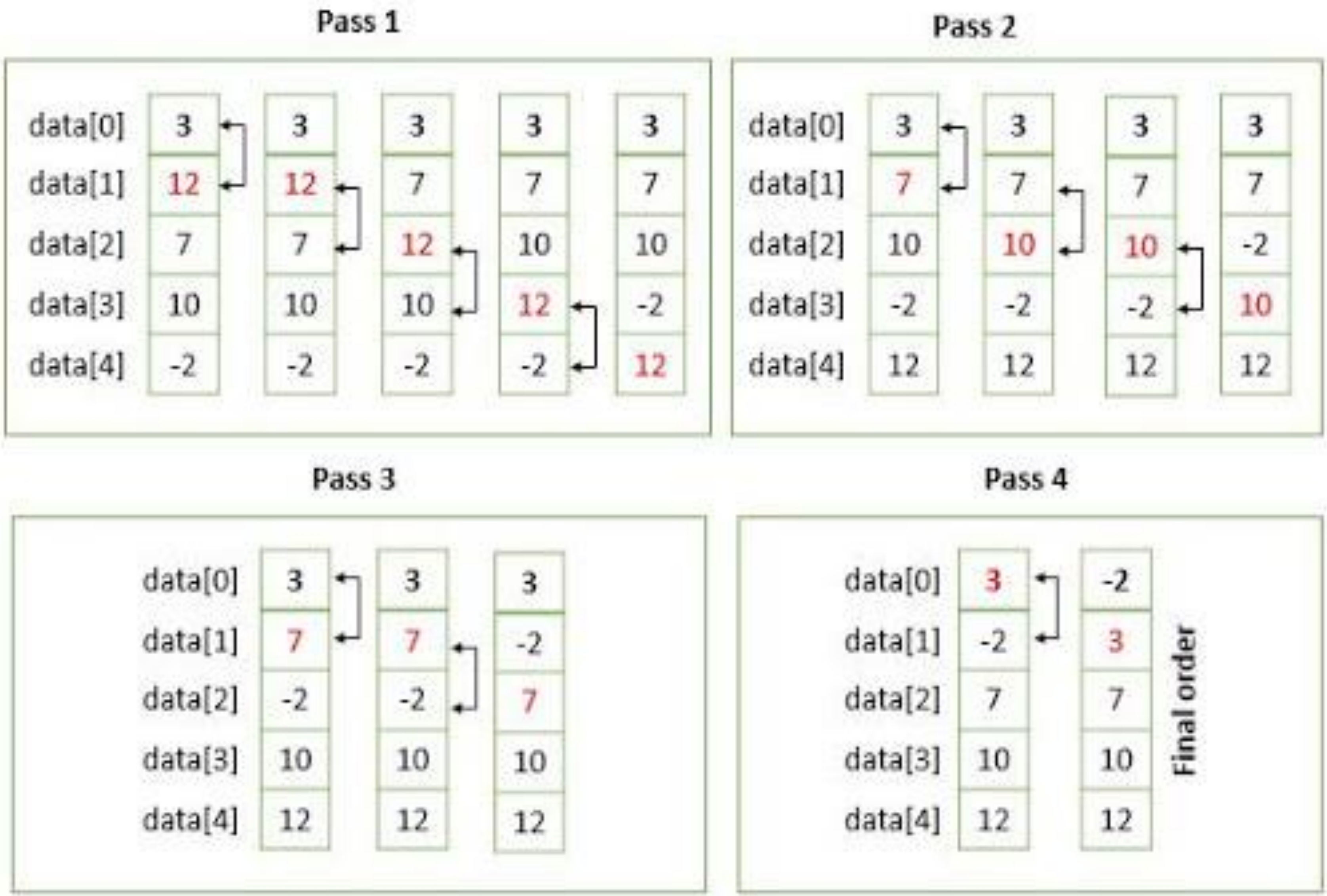
**Selection_sort(A)**

```
n ←length[A]
for i←1 to n-1
        min ← i
        for j←i+1 to n
                if(A[j] < A[min])
                        min ←j
        Interchange  A[i]  ↔ A[min]
```

**Time complexity, $T(n) = \theta\ (n^2)$**

# Some other sorting algorithms

## Bubble Sort



# Some other sorting algorithms

**Bubble _Sort(A)**

```
Procedure bubblesort (List array, number length_of_array)
        for i=1 to length_of_array - 1;
                for j=1 to length_of_array – I;
                        if array [ j ] > array [ j+1] then
                                temporary = array [ j+1]
                                array[ j+1] = array [j]
                                array[ j ] = temporary
                        end if
                end of j loop
        end of i loop
return array
End of procedure
```

# Divide and Conquer approach

The divide-and-conquer paradigm involves three steps at each level of the recursion:

**Divide** the problem into a number of sub-problems that are smaller instances of the same problem.

**Conquer** the sub-problems by solving them recursively. If the sub-problem sizes are small enough, however, just solve the sub problems in a straightforward manner.

**Combine** the solutions to the sub-problems into the solution for the original problem.

# Analysis of Divide and Conquer based algorithm

When an algorithm contains a recursive call to itself, we can often describe its running time by a ***recurrence equation*** or ***recurrence***, which describes the overall running time on a problem of size n in terms of the running time on smaller inputs. We can then use mathematical tools to solve the recurrence and provide bounds on the performance of the algorithm.

# Analysis of Divide and Conquer based algorithm

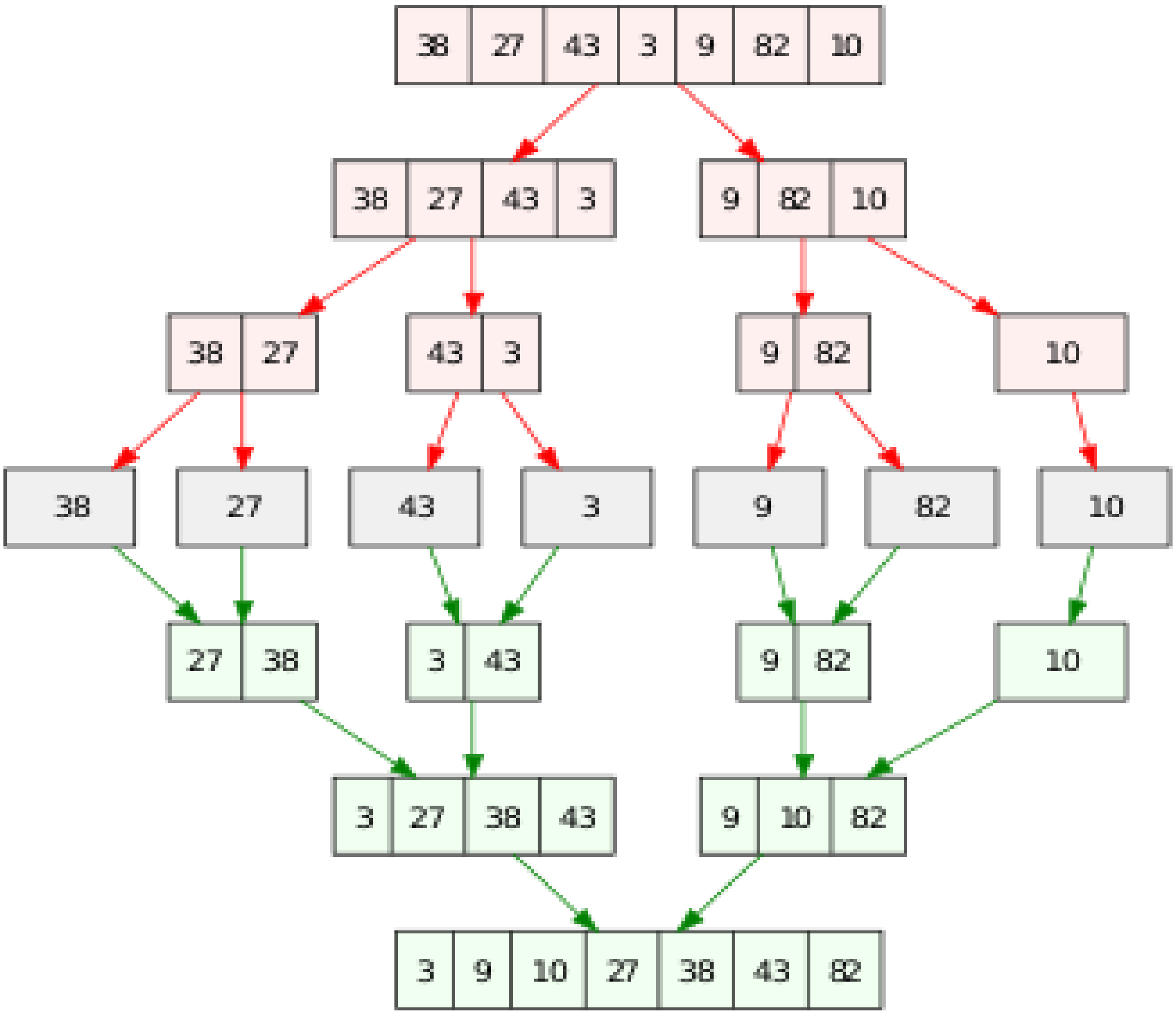
A recurrence equation for the running time of a divide-and-conquer algorithm uses the three steps of the basic paradigm.

The recurrence equation for the running time of a divide-and-conquer algorithm is the following:-

$$
\begin{aligned}
T(n) &= aT(n/b) + D(n) + C(n), && \text{if } n > c \\
&= \theta(1) \quad , && \text{otherwise}
\end{aligned}
$$

Where, n is the size of original problem. a is the number of sub-problems in which the original problem divided at an instant. Each sub-problems has size n/b. c is a small integer.

# Merge Sort



38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

# Merge Sort Algorithm

MERGE-SORT$(A, p, r)$

**if** $p < r$

$\quad q = \lfloor (p + r)/2 \rfloor$

$\quad$ MERGE-SORT$(A, p, q)$

$\quad$ MERGE-SORT$(A, q + 1, r)$

$\quad$ MERGE$(A, p, q, r)$

# Merge Sort Algorithm

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
        L[i] = A[p + i - 1]
for j = 1 to n2
        R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else A[k] = R[j]
                j = j + 1
```

# Merge Sort Algorithm Analysis

The recurrence equation for the running time of merge sort algorithm will be

$$T(n) = 2T(n/2) + \theta(1) + \theta(n), \quad \text{if } n > 1$$
$$= \theta(1) \quad , \quad \text{otherwise}$$

It can be modified as:-

$$T(n) = 2T(n/2) + \theta(n) \quad , \quad \text{if } n > 1$$
$$= \theta(1) \quad , \quad \text{otherwise}$$

# Merge Sort Algorithm Analysis

When we solve recurrence equation, modify it as:-

$$T(n) = 2T(n/2) + \mathbf{cn}, \qquad \text{if } n > 1$$
$$= \mathbf{d}, \qquad \text{otherwise}$$

Here, c and d are some constants.

# Merge Sort Algorithm Analysis

## Iterative method:

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

$= 2(2T(n/4) + cn/2) + cn$

$= 2^2T(n/4) + 2cn$

$= 2^2(2T(n/8)+cn/4) + 2cn$

$= 2^3T(n/8)+ 3cn$

$= \ldots\ldots\ldots\ldots\ldots\ldots\ldots$

$= 2^kT(n/2^k) + kcn$

$= nT(1) + c\,n\log(n)$ **(Let $n = 2^k$ )**

$= dn + cn\log(n)$ **(since $T(1) = d$)**

$= \theta(n\log(n))$

Therefore, **$T(n) = \theta(n\log(n))$**