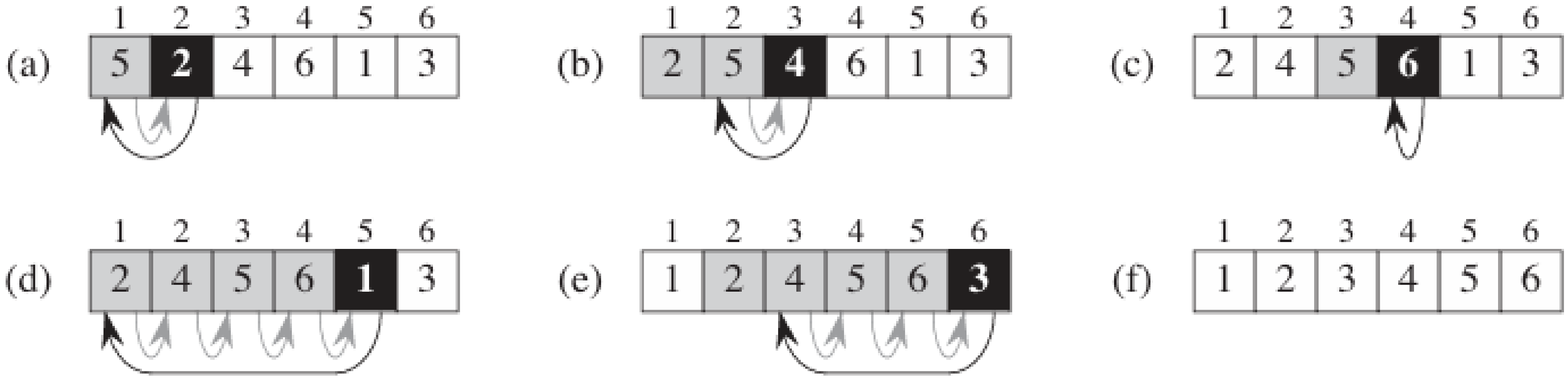
# Divide and Conquer Approach

Merge sort, Integer multiplication, Solving recurrence relation

# Insertion Sort



# Insertion Sort: Algorithm

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

# Bubble Sort

```
BUBBLESORT(A)
for i = 1 to A.length - 1
    for j = A.length downto i + 1
        if A[j] < A[j - 1]
            exchange A[j] with A[j - 1]
```

# General Idea

- In this approach, the problem is solved recursively, i.e., a procedure calls itself several times in order to solve the problem
- Idea is to divide the original problem into several sub-problems that are either same or closely related to the original problem but are smaller in size
- The results of sub problems is combined to get the solution to original problem
- The combination procedure may be slightly unrelated to the original problem and might be treated as an overhead

# Steps:

- The divide and conquer paradigm has three steps:
  1. **Divide**: the problem into a number of subproblems that are similar instances of the original problem
  2. **Conquer:** the subproblems by solving them recursively, if the subproblems are small however, solve them in a straightforward manner
  3. **Combine:** the solutions to the subproblems into the solution of the original problem

# Merge Sort

- The merge sort procedure closely follows the divide and conquer approach of problem solving.
- Steps of merge sort:
    1. Divide: the n-element array into two arrays of size n/2 each
    2. Conquer: sort the two sub-arrays recursively
    3. Combine: Merge the two sorted sub-arrays to get the final sorted array
- The procedure is said to “bottom out” when there are no more elements left to divide
- At this point, the merge procedure is invoked

# Merge Process

*Time complexity*: $\theta(n)$

```
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
```

# Analyzing the Merge Process

- Steps 1 to 3 take constant time (say c0)
- Steps 4 to 5 take – n1*c1 time
- Steps 6 to 7 take – n2*c1 time
- Steps 8 to 11 take constant time c3
- Steps 12 to 17 take (r-p+1)*c4 time
- Total time – c0 + n1*c1 + n2*c2 + c3 + (r-p+1)*c4
- => c0 + (r-p+1)c1 => n*c1
- => $\theta(n)$ where n = r-p+1

# Merge Sort Procedure

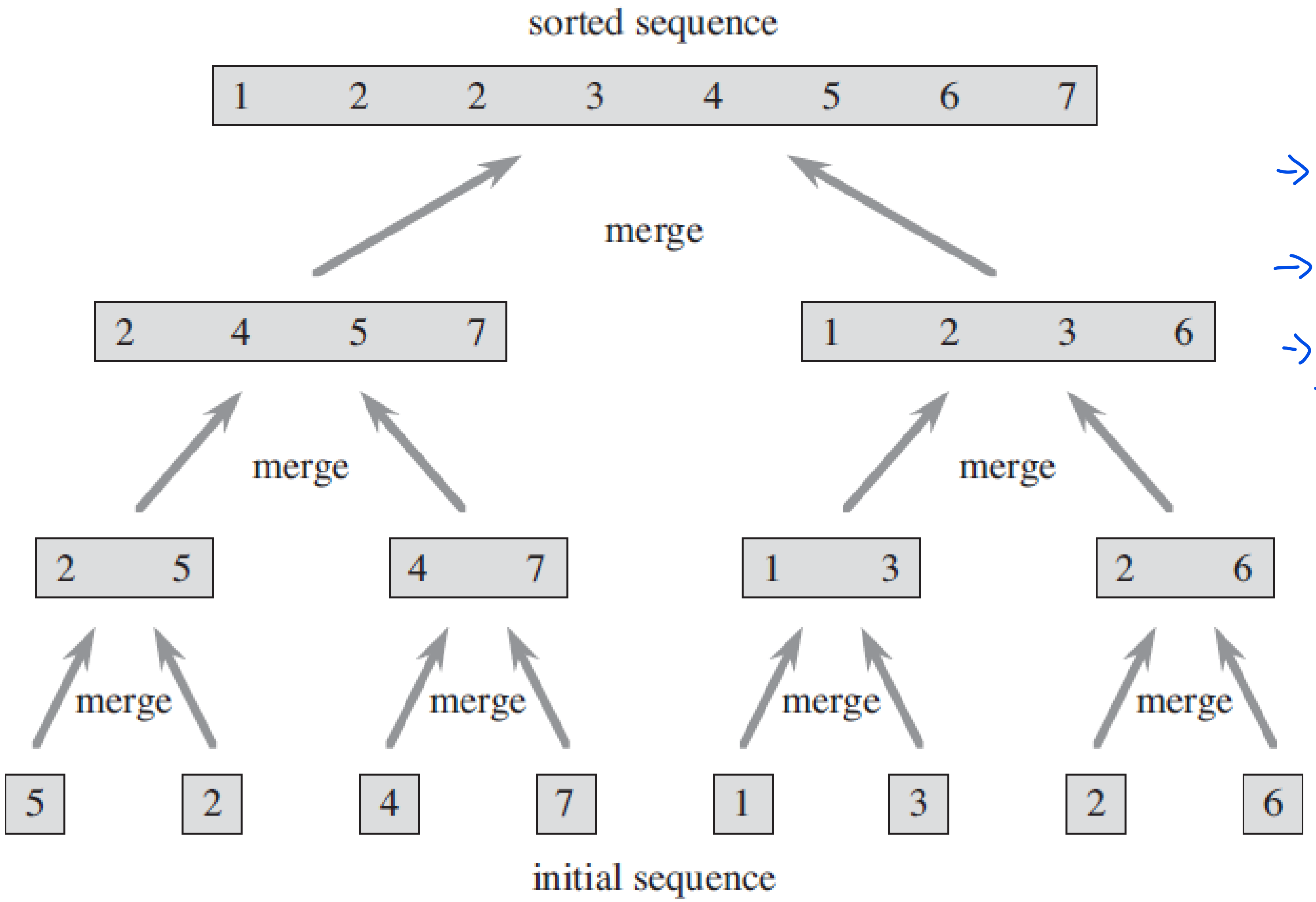
```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

T(n) = $\theta(1) \quad for\ n = 1$

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$$

# Analysis of Divide and Conquer Algorithms

- The running time can be described using recurrence equation or simply recurrence
- It describes the overall running time of the algorithm on a problem size of ‘n’ in terms of running time on smaller inputs
- Mathematical tools can be used to solve the recurrence and obtain the bounds on running time




$$\frac{n}{2^i} = 1$$

$\rightarrow \log n = \log 2^i$

$\rightarrow \log n = i \log 2$

$\rightarrow \dfrac{\log n}{\log 2} = i$

$\therefore \boxed{i = \log_2 n}$

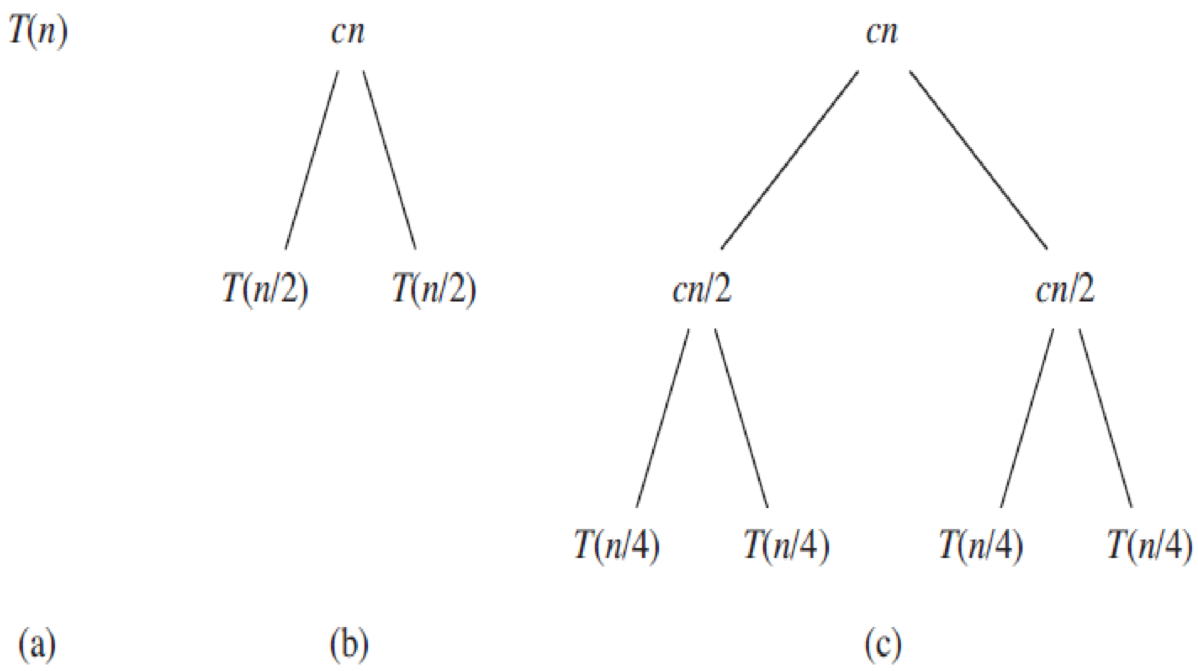
# Recurrence relation for Merge Sort

- Assuming that the list can always be divided into two equal halves
- Division takes constant time
- Merger takes constant time
- Each sub-problem takes half the time of original problem
- When the subproblem has just one element it takes constant time for sorting (no sorting required)
- Suppose, T(n) be the time for solving the original problem of size 'n' then, the recurrence relation can be written as:

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

Recurrence Equation!

# Solving Recurrence using Recursion Tree




- Recursion Tree can be used to solve simple recurrence equations
- Solving the above recurrence using recursion tree yields $\theta(nlogn)$ time complexity for merge sort

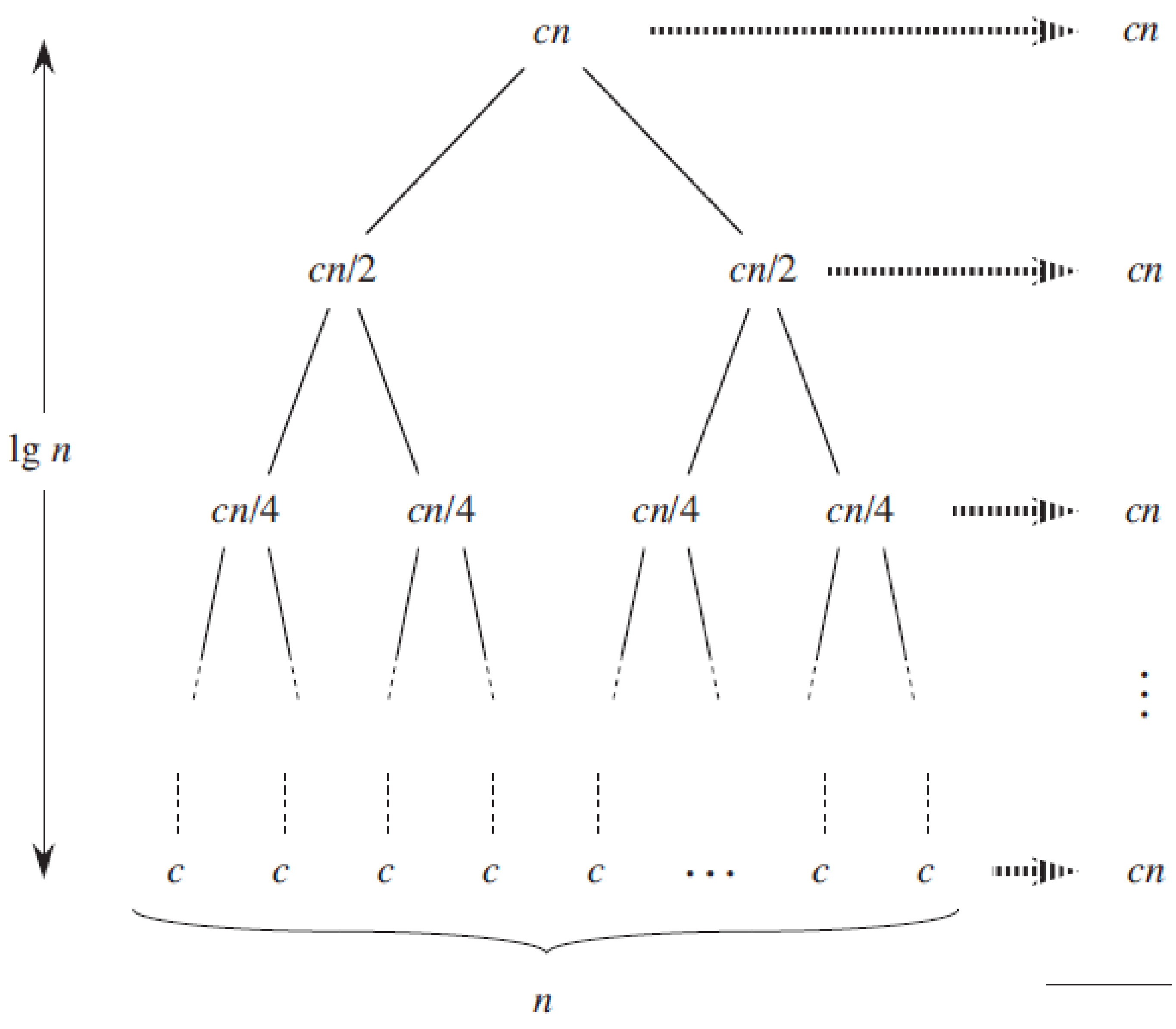
$$\frac{n}{2^i} = 1$$

$$\log n = i \log 2$$

$$\frac{\log n}{\log 2} = i$$

$$\boxed{\therefore i = \log_2 n}$$

Height




Total: $cn \lg n + cn$

(d)

(depth $*$ cost) + merge time

$cn \log_2 n + cn$

↳ merge time