Merge Sort

Lecture 3

Why Study Merge Sort?

- Good introduction to divide & conquer
  - Improves over Selection, Insertion, Bubble sorts
- Calibrate your preparation
- Motivates guiding principles for algorithm analysis (worst--case and asymptotic analysis)

The Sorting Problem

Input : array of n numbers, unsorted.

5	4	1	8	7	2	6	3

Output : Same numbers, sorted in increasing order

1	2	3	4	5	6	7	8

Merge sort

- Invented by John von Neumann (1903-1957)
- Follows divide and conquer paradigm.
- Developed merge sort for EDVAC in 1945

Divide and Conquer Approach

1. Divide the problem into sub problems
2. Solve the sub problems
3. Combine the solution of sub problems

Merge Sort

1. **Divide:** Divide the unsorted list into two sub lists of about half the size.
2. **Conquer:** Sort each of the two sub lists recursively until we have list sizes of length 1,in which case the list itself is returned.
3. **Combine:** Merge the two-sorted sub lists back into one sorted list

Merge Sort Pseudocode

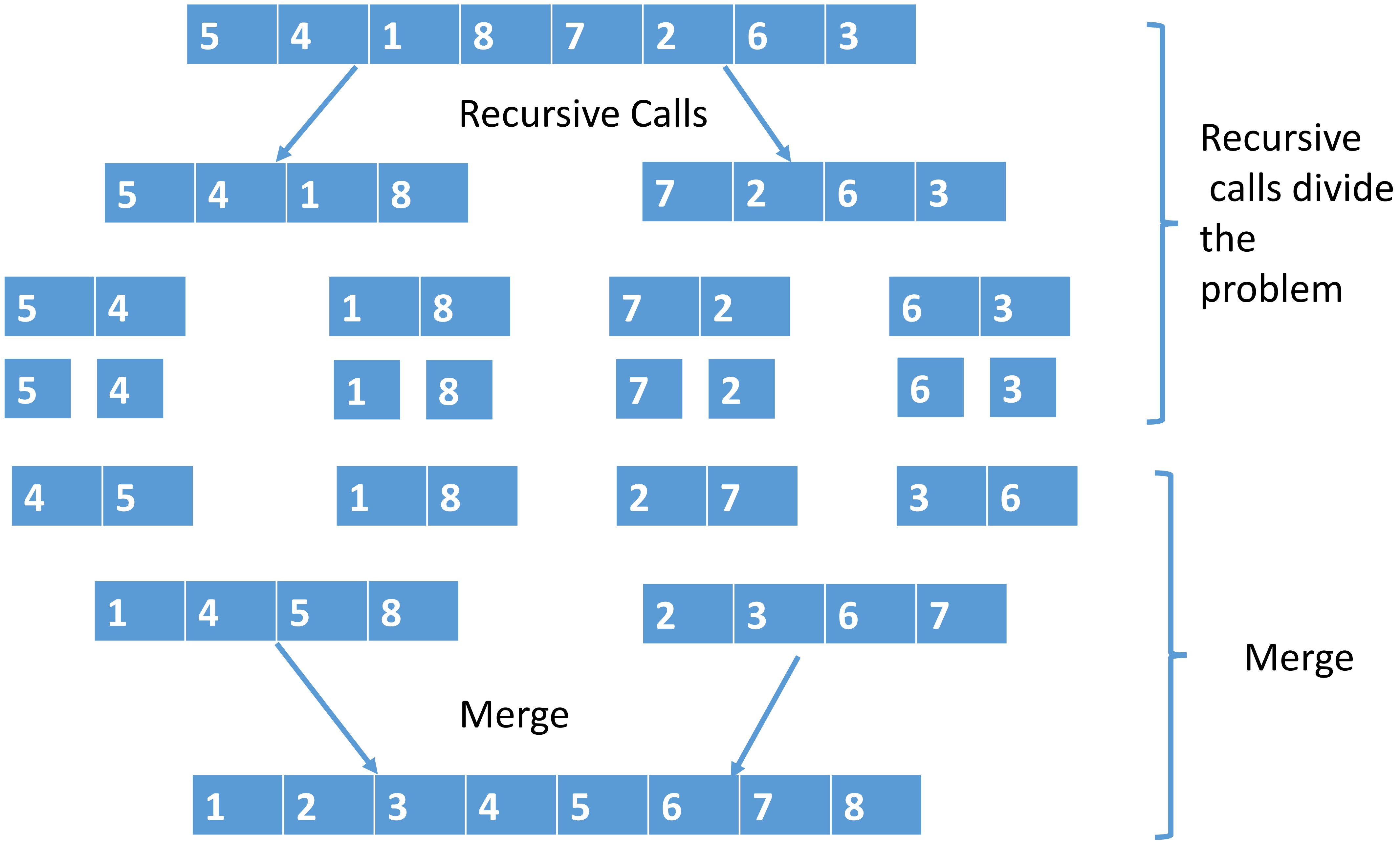
- Merge Sort: Pseudocode
  ---- recursively sort 1st half of the input array
  ---- recursively sort 2nd half of the input array
  ---- merge two sorted sublists into one
  [ignores base cases]

Pseudo code for mergesort

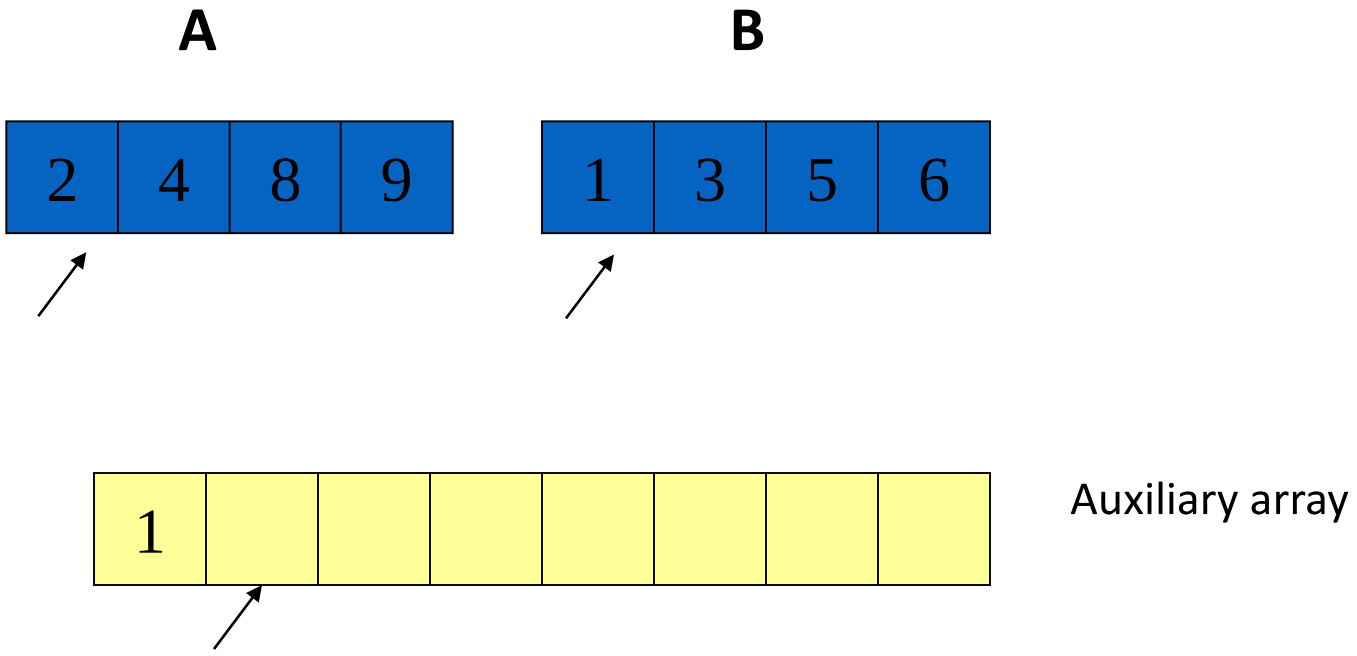
- Mergesort(array, p, q)

```
{
      if (q – p = 1)
            return array
      m = (p+q)/2
      A = Mergesort(array, p, m)
      B = Mergesort(array, m+1, q)
      C = merge(A, B)
      return C
}
```

Merge Sort Example

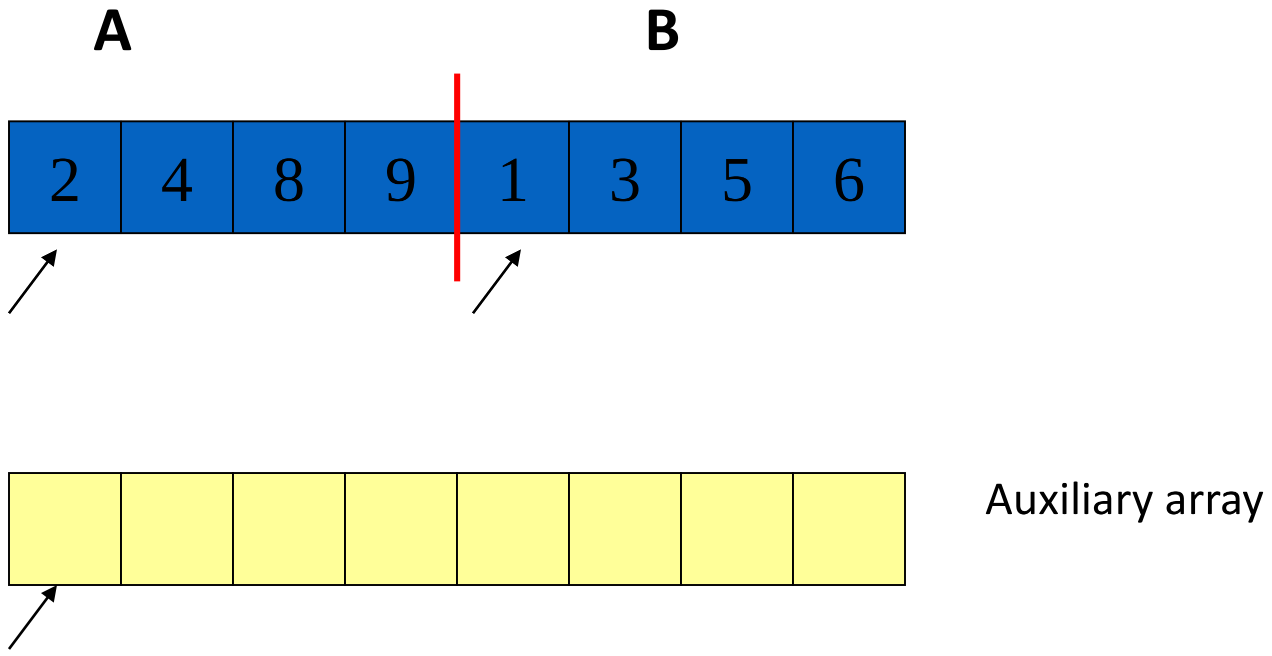


How to merge two sorted lists to get sorted list?



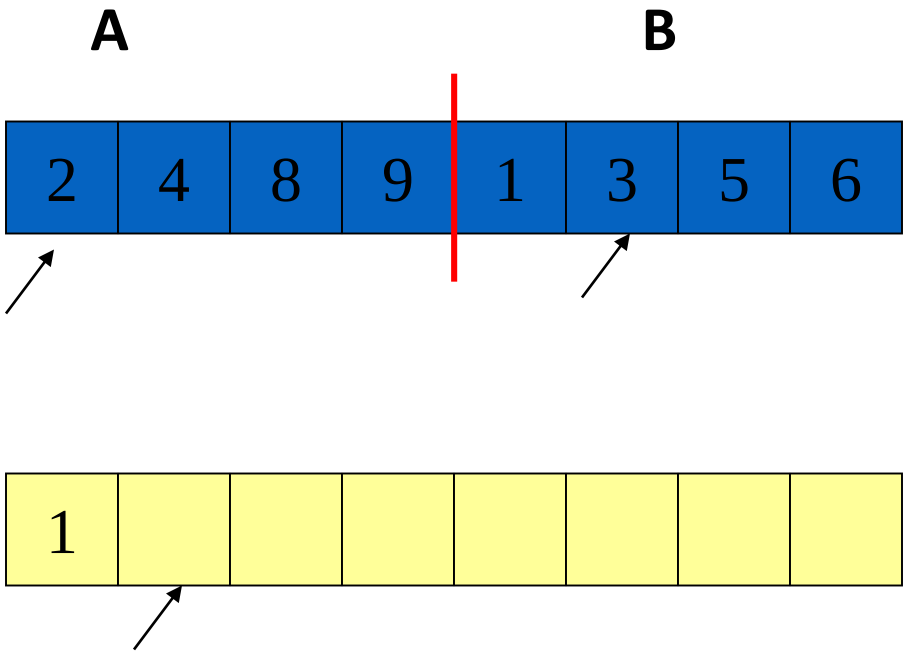
Merge Operation

- The merging requires an auxiliary array.



Merge Operation

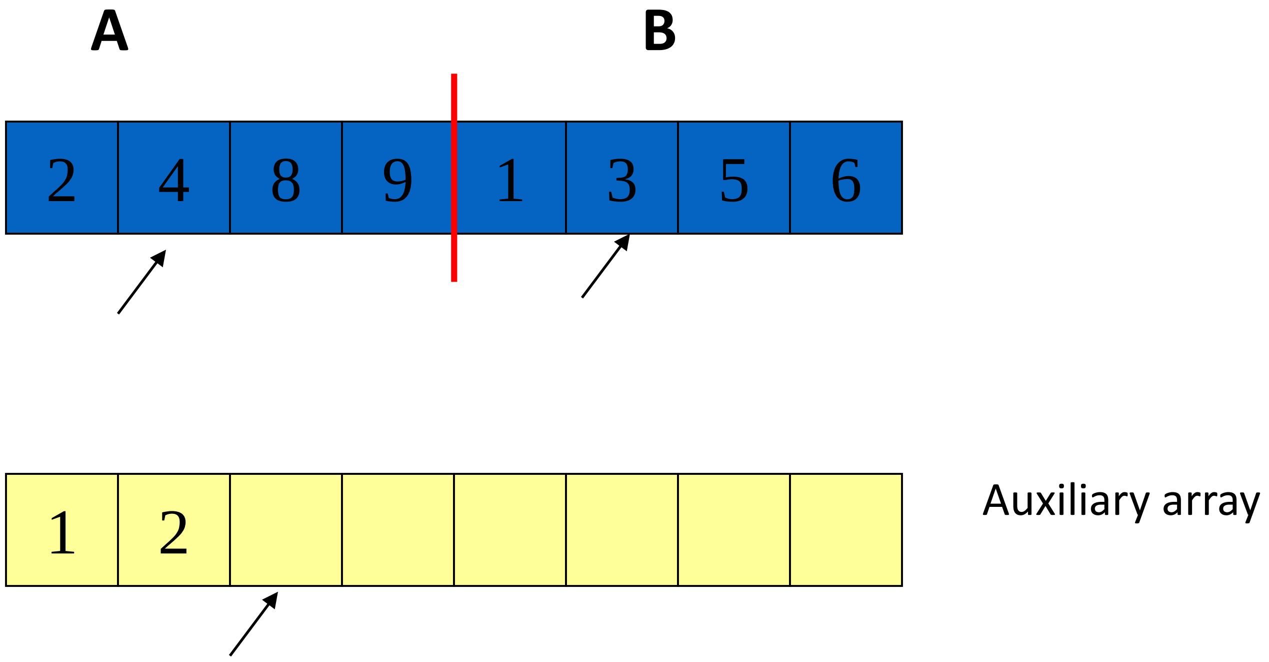
- The merging requires an auxiliary array.



Auxiliary array

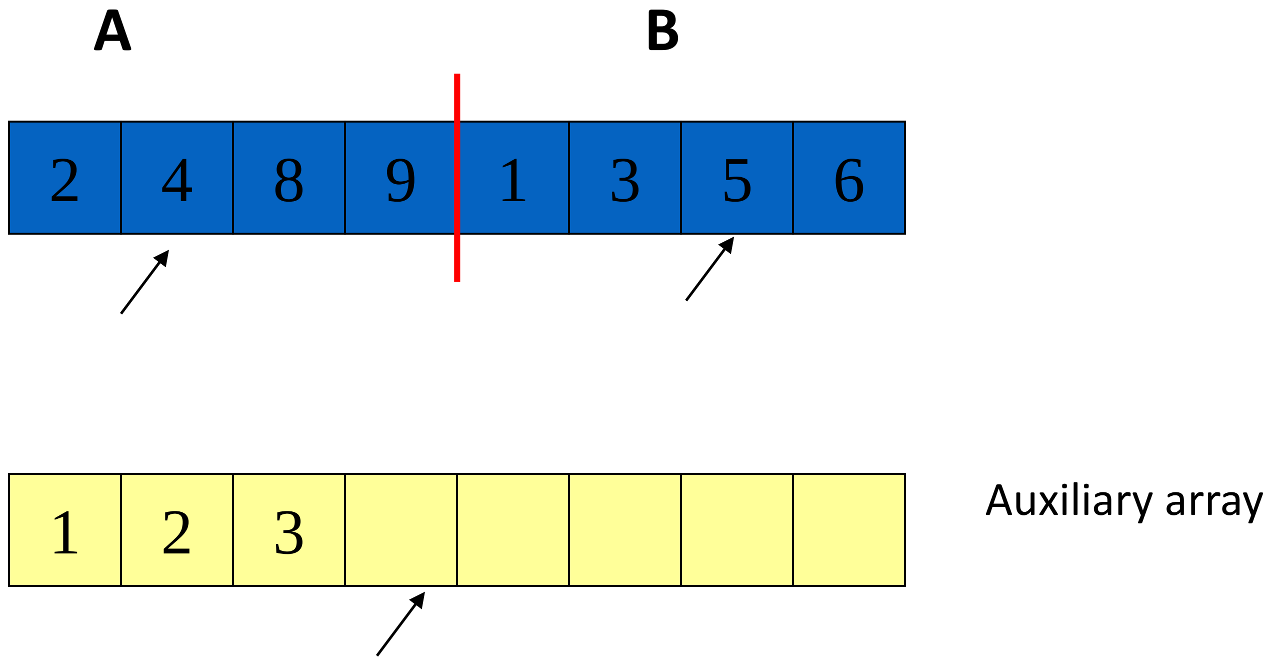
Merge Operation

- The merging requires an auxiliary array.



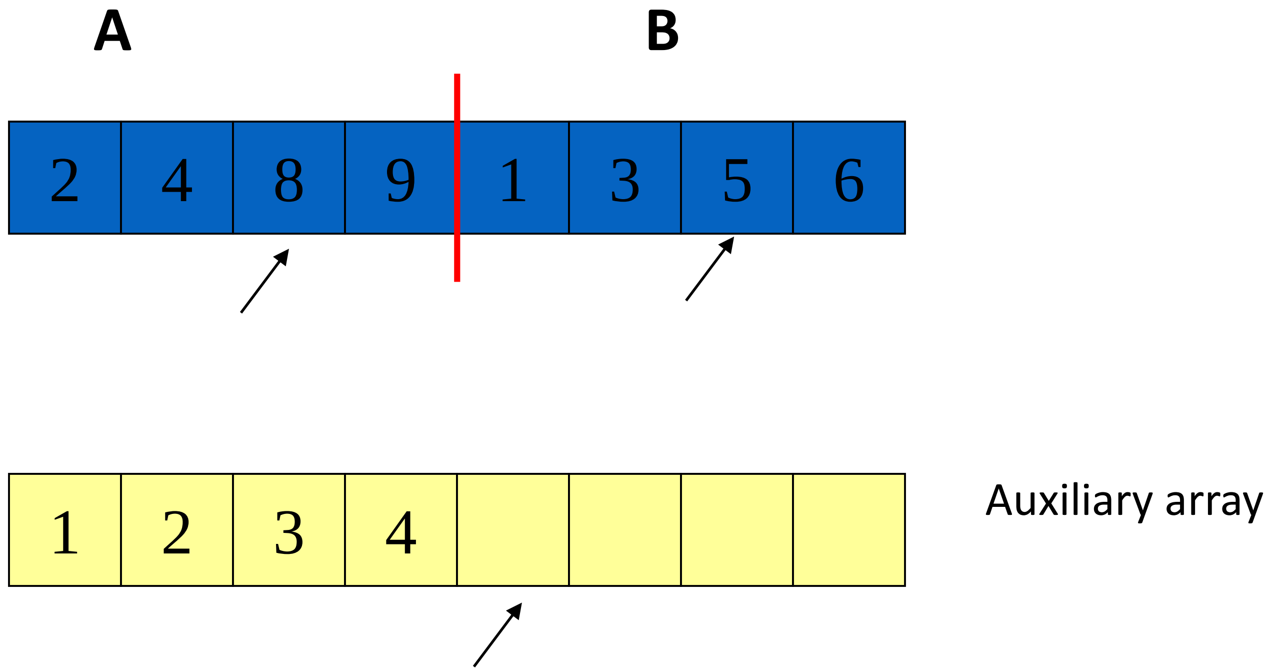
Merge Operation

- The merging requires an auxiliary array.



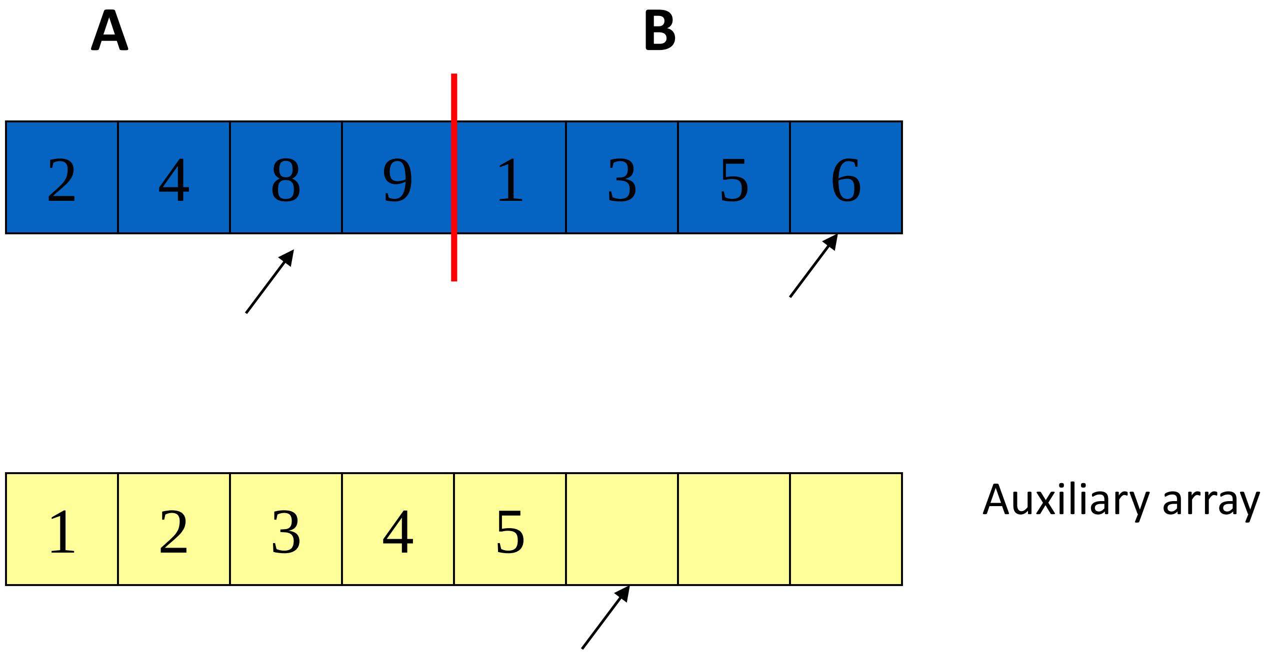
Merge Operation

- The merging requires an auxiliary array.



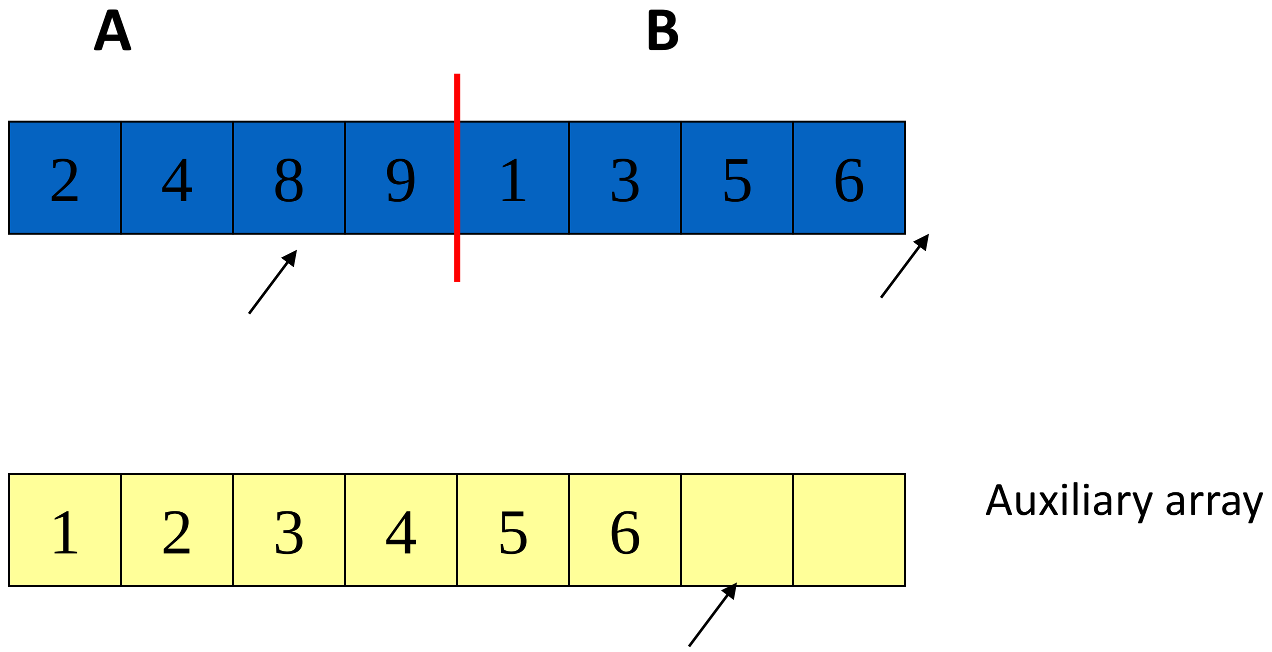
Merge Operation

- The merging requires an auxiliary array.



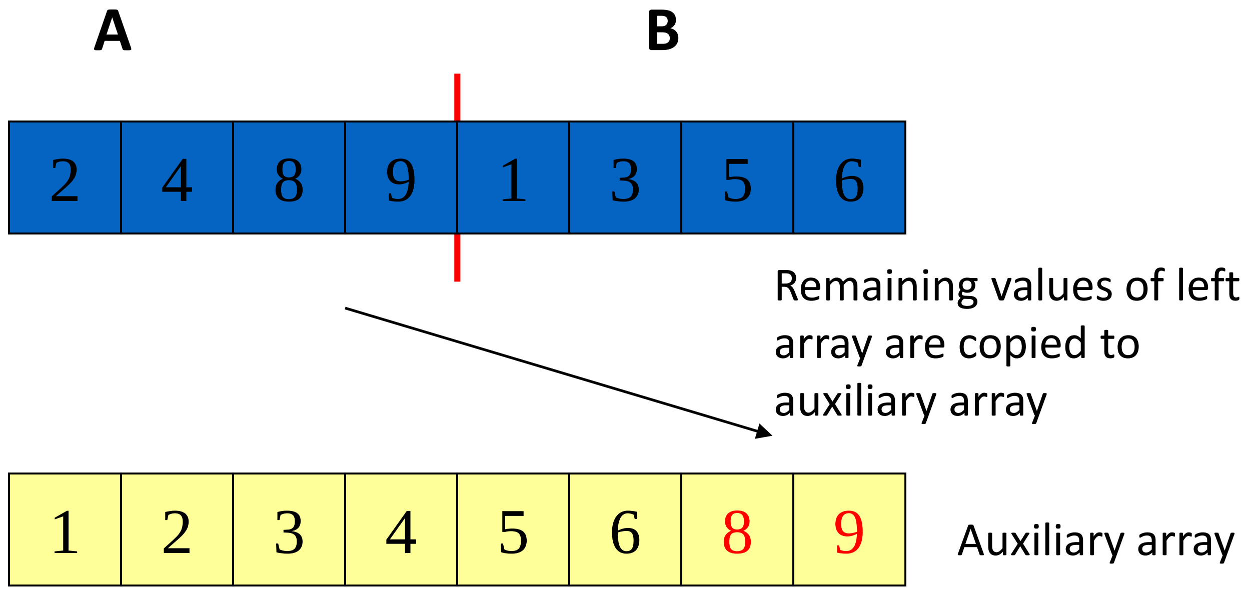
Merge Operation

- The merging requires an auxiliary array.

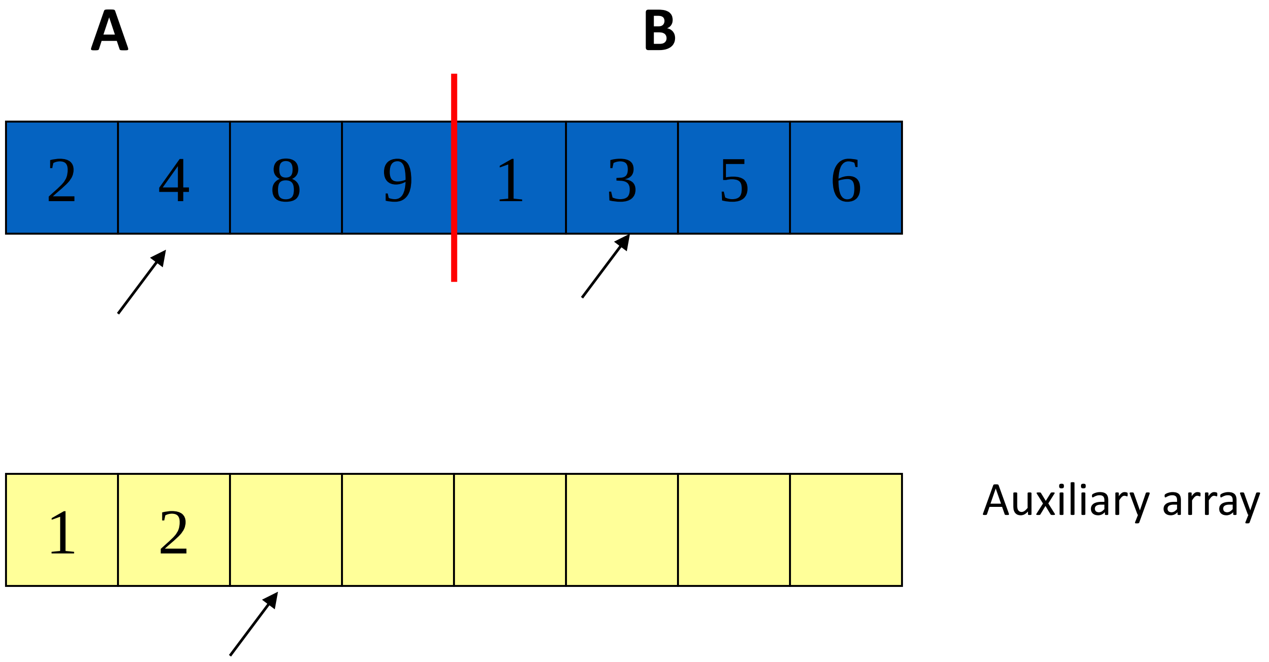


Merge Operation

- The merging requires an auxiliary array.



Merge Operation Running Time



Pseudocode for Merge

```
C = output [length = n]
A = 1st sorted array [n/2]
B = 2nd sorted array [n/2]
i = 1
j = 1
```

```
for k = 1 to n
        if A(i) ≤ B(j)
                C(k) = A(i)
                i++
        else
                C(k) = B(j)
                j++
end
```

Pseudocode for Merge (Running time)

C = output [length = n]
A = 1st sorted array [n/2]
B = 2nd sorted array [n/2]

$\left.\begin{array}{l} i = 1 \\ j = 1 \end{array}\right\}$ 2 operations

```
for k = 1 to n
    if A(i) ≤ B(j)
        C(k) = A(i)
        i++
    else
        C(k) = B(j)
        j++
end
```

Running Time of Merge

- running time of Merge on array of m numbers is

$$\leq 4m + 2$$

$$\leq 6m \qquad\qquad (since\, m\ > 1)$$

4n+2 is Θ(n)

$k_2 n \le 4n + 2 \le k_1 n$

$k_2 \le 4 + 2/n \le k_1$

$4 + 2/n \le k_1$
$k_1 = 6$

$4 + 2/n \ge k_2$
$k_2 = 4$

$n_0 = 1$

Merge Sort Running Time?

Key Question : running time of Merge Sort on array of n numbers ?

[running time = # of lines of code executed]

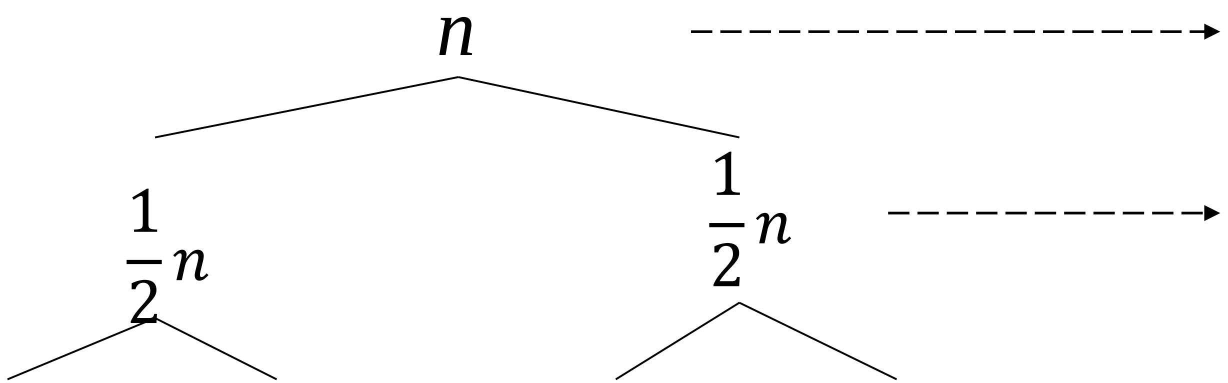
Merge Sort Running Time?

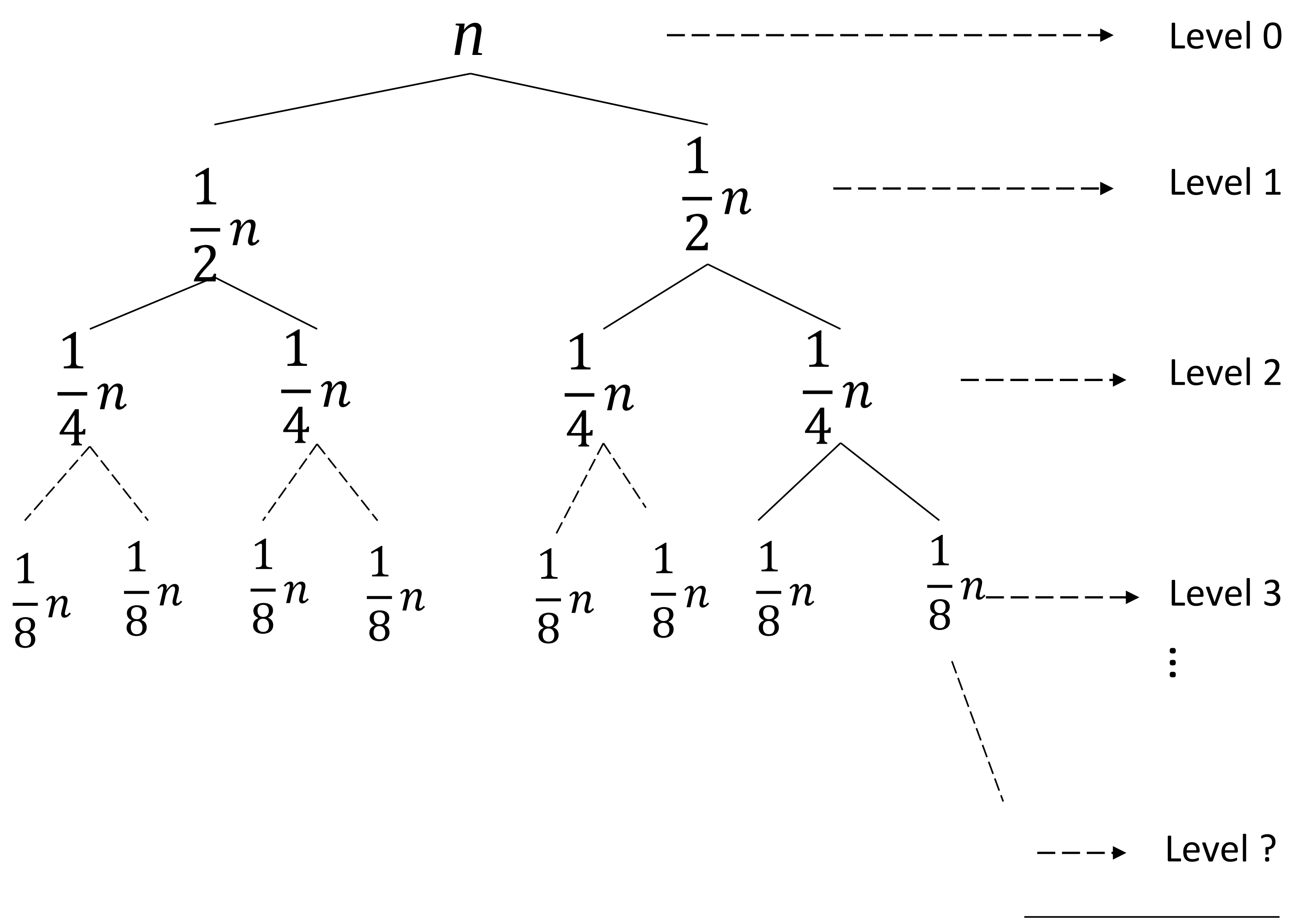
$$T(n) = T(n/2) + T(n/2) + \Theta(n)$$

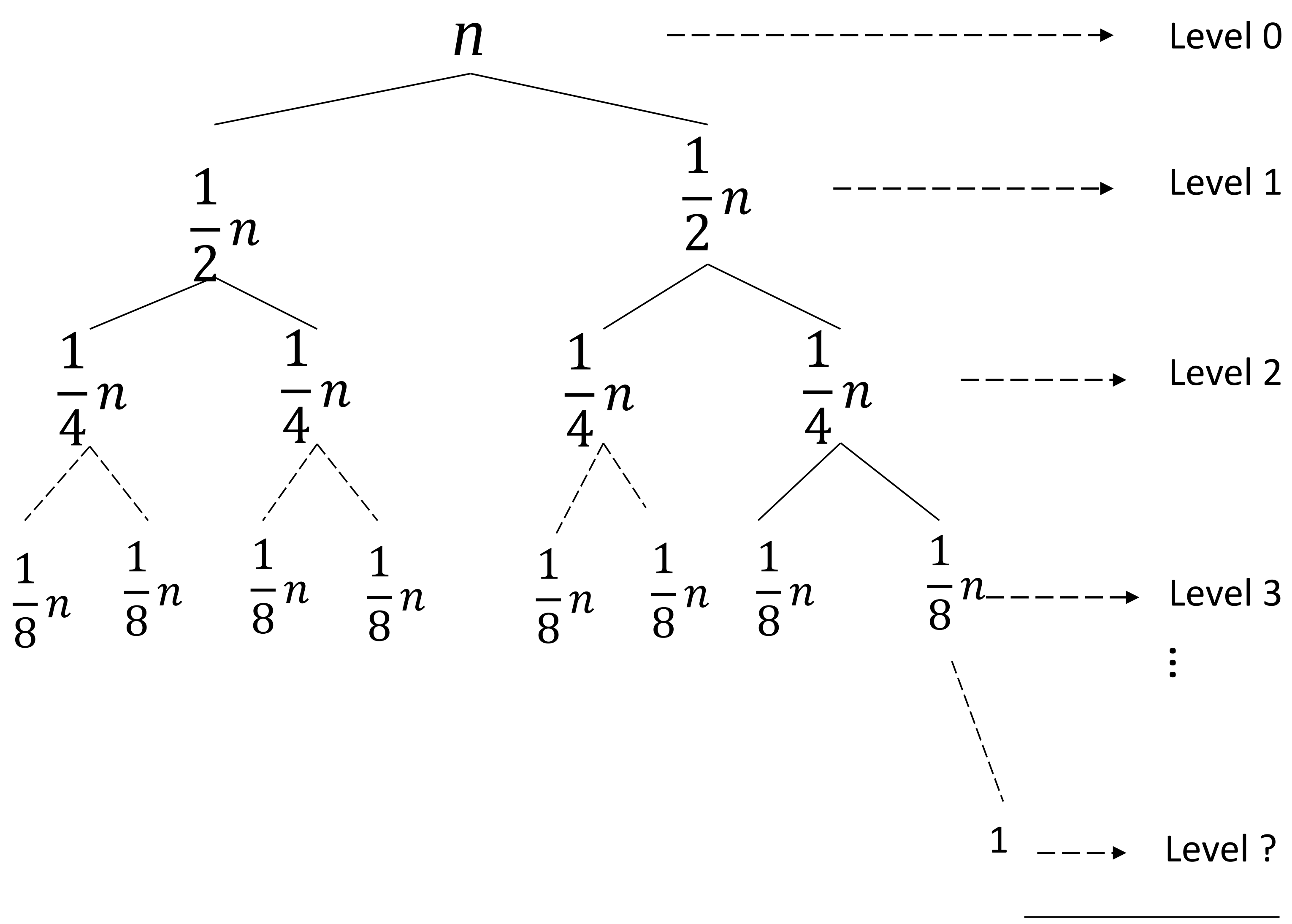
$$T(n) = 2T(n/2) + \Theta(n)$$

```
Mergesort(array,  p, q)
    {
        if (q – p = 1)
            return array
        m = (p+q)/2
        A = Mergesort(array, p, m)
        B = Mergesort(array, m+1, q)
        C = merge(A, B)
        return C
    }
```

T(n) = 2T(n/2) + Θ(n)




n
$\frac{1}{2}n$
$\frac{1}{2}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
Level 0
Level 1
Level 2
Level 3
⋮
Level ?



n
$\frac{1}{2}n$
$\frac{1}{2}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
1
Level 0
Level 1
Level 2
Level 3
⋮
Level ?

Question

Roughly how many levels does this recursion tree have (as a function of n, the length of the input array)?

a) A constant number (independent of n).

$b)$ $\log_2 n$

$c)$ n

$d)$ $\sqrt{n}$

Question

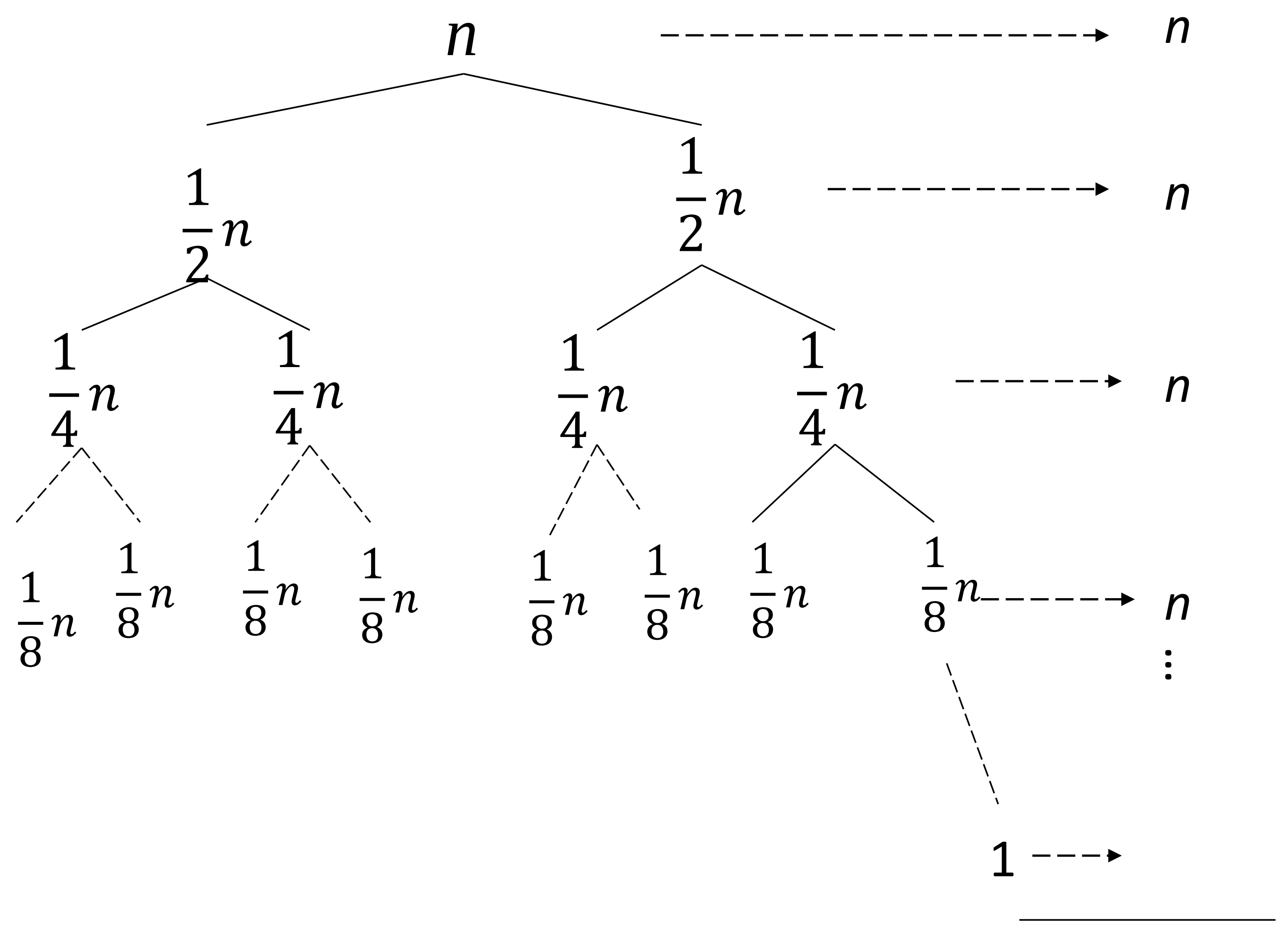
Roughly how many levels does this recursion tree have (as a function of n, the length of the input array)?

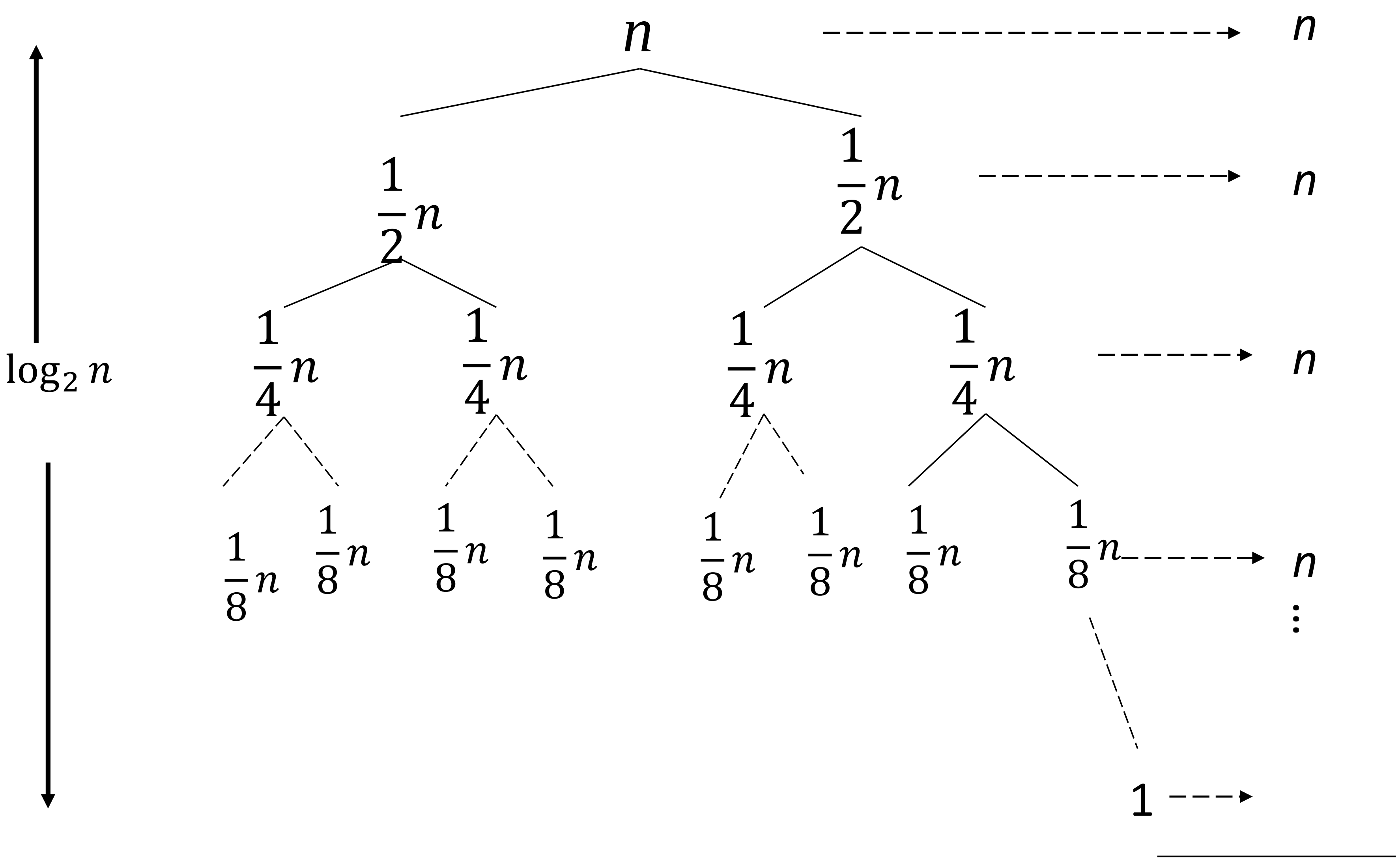
a) A constant number (independent of n).

$b)$ $\log_2 n$ (correct answer)

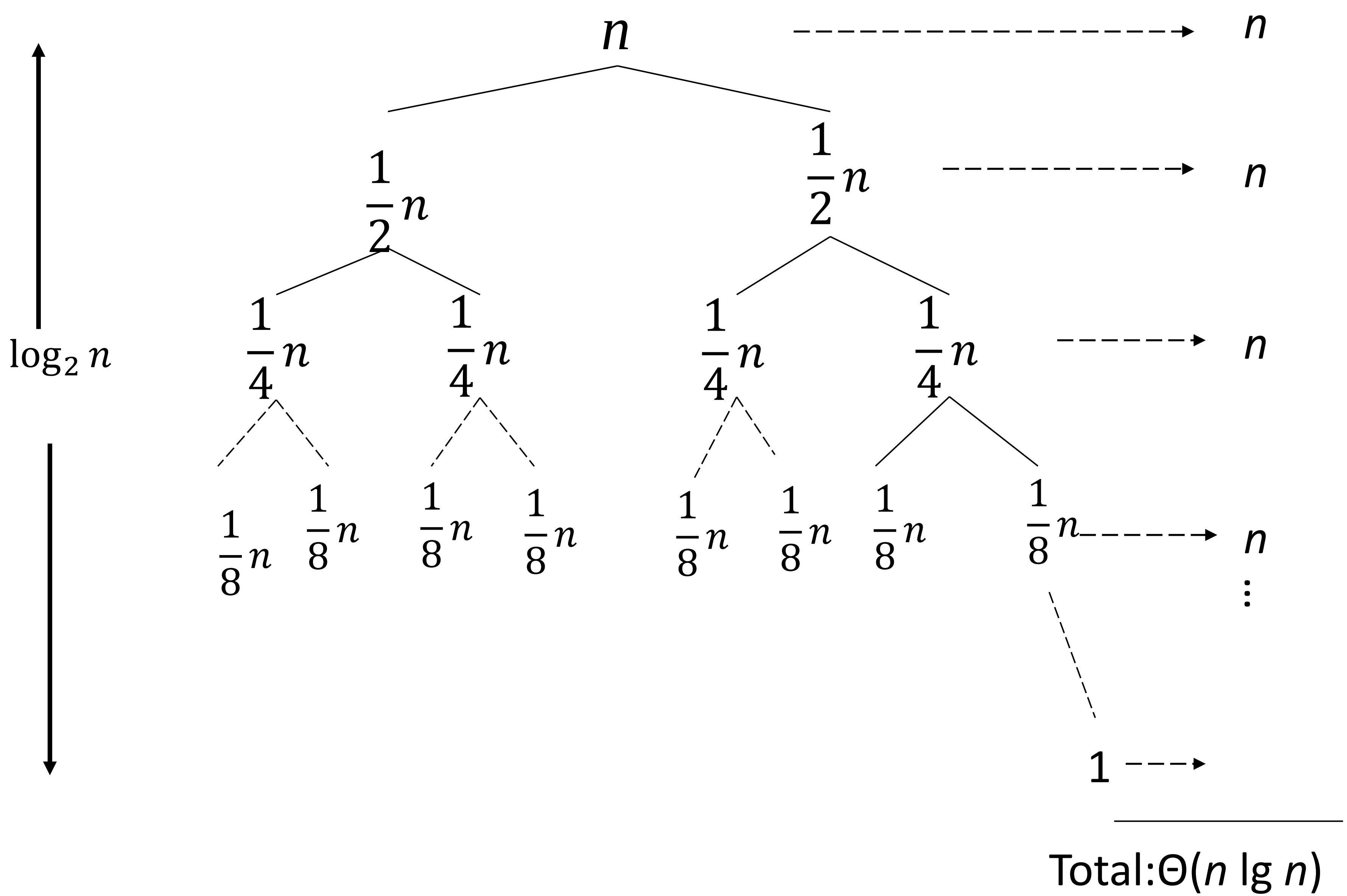
$c)$ n

$d)$ $\sqrt{n}$


n
$\frac{1}{2}n$
$\frac{1}{2}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
1
n
n
n
n
⋮



n
$\frac{1}{2}n$
$\frac{1}{2}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{4}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
$\frac{1}{8}n$
1
$\log_2 n$
n
n
n
n
⋮


n
$\frac{1}{2}n$ $\frac{1}{2}n$
$\frac{1}{4}n$ $\frac{1}{4}n$ $\frac{1}{4}n$ $\frac{1}{4}n$
$\frac{1}{8}n$ $\frac{1}{8}n$ $\frac{1}{8}n$ $\frac{1}{8}n$ $\frac{1}{8}n$ $\frac{1}{8}n$ $\frac{1}{8}n$ $\frac{1}{8}n$
1
$\log_2 n$
n
n
n
n
⋮
Total:Θ(n lg n)

What is the pattern ? Fill in the blanks in the following statement: at each level j = 0,1,2,.., $\log_2 n$, there are <blank> subproblems, each of size <blank>.

$a)$ $2^j \quad and\ 2^j$, respectively

$b)$ $\frac{n}{2^j}\, and\, \frac{n}{2^j}$, respectively

$c)$ $2^j\ and\, \frac{n}{2^j}$, respectively

$d)$ $\frac{n}{2^j}\, and 2^j$, respectively

What is the pattern ? Fill in the blanks in the following statement: at each level j = 0,1,2,.., $\log_2 n$, there are <blank> subproblems, each of size <blank>.

$a)$ 2^j and 2^j, respectively

$b)$ $\frac{n}{2^j}$ and $\frac{n}{2^j}$, respectively

$c)$ 2^j and $\frac{n}{2^j}$, respectively

$d)$ $\frac{n}{2^j}$ $and 2^j$, respectively

Proof of claim (assuming n = power of 2) :

At each level j=0,1,2,.., $\log_2 n$,

Total # of operations at level j = 0,1,2,...,$\log_2 n$

$$\leq 2^j * 6\left(\frac{n}{2^j}\right) = 6n$$

# of level-j subproblems

Size of level-j subproblem

Work per level – j subproblem

Total

$$6n(\log_2 n + 1)$$

Work per level

# of levels

Running Time of Merge Sort

- For every input array of n numbers, Merge Sort produces a sorted output array and uses at most $6n \log_2 n + 6n$ operations.

- Prove $6n \log_2 n + 6n$ is Θ(n lg n).
- $k_2 n \log_2 n \leq 6n \log_2 n + 6n \leq k_1 n \log_2 n$

- Prove $6n \log_2 n \ + 6n$ is Θ(n lg n).
- $k_2 n \log_2 n \leq 6n \log_2 n \ + 6n \leq \quad k_1 \, n \log_2 n$

- $k_2 \ \leq 6 + \ 6/\log_2 n \ \leq \quad k_1$

- k_1 = 12
- k_2 = 6
- n_0 = 2

$T(n) = T(n/3) + T(2n/3) + \Theta(n)$