Divide and Conquer

Divide and Conquer

- A problem solving approach / technique / strategy / paradigm.
- Recursive in nature,
- involves three steps:

- ***Divide*** the problem into a number of *subproblems*.
- ***Conquer*** the *subproblems* by solving them ***recursively***.

  Base case:

  If the subproblem sizes are small enough,

  just solve them.
- ***Combine*** the *solutions to the subproblem* into the solution for the original problem.

Running Time: Divide and conquer

- Suppose that our division of the problem yields a subproblems, each of which is $1/b$ the size of the original.
  - We shall see many divide-and-conquer algorithms in which $a \neq b$.
- Let $D(n)$ denote time to divide the problem into subproblems.
- Let $C(n)$ denote time to combine the solutions to the subproblems into the solution to the original problem.
- We get the recurrence

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

Merge Sort

- A sorting algorithm based on divide and conquer.

- **Divide** by splitting into two subarrays

  $A[p .. q]$ and $A[q+1 .. r]$,

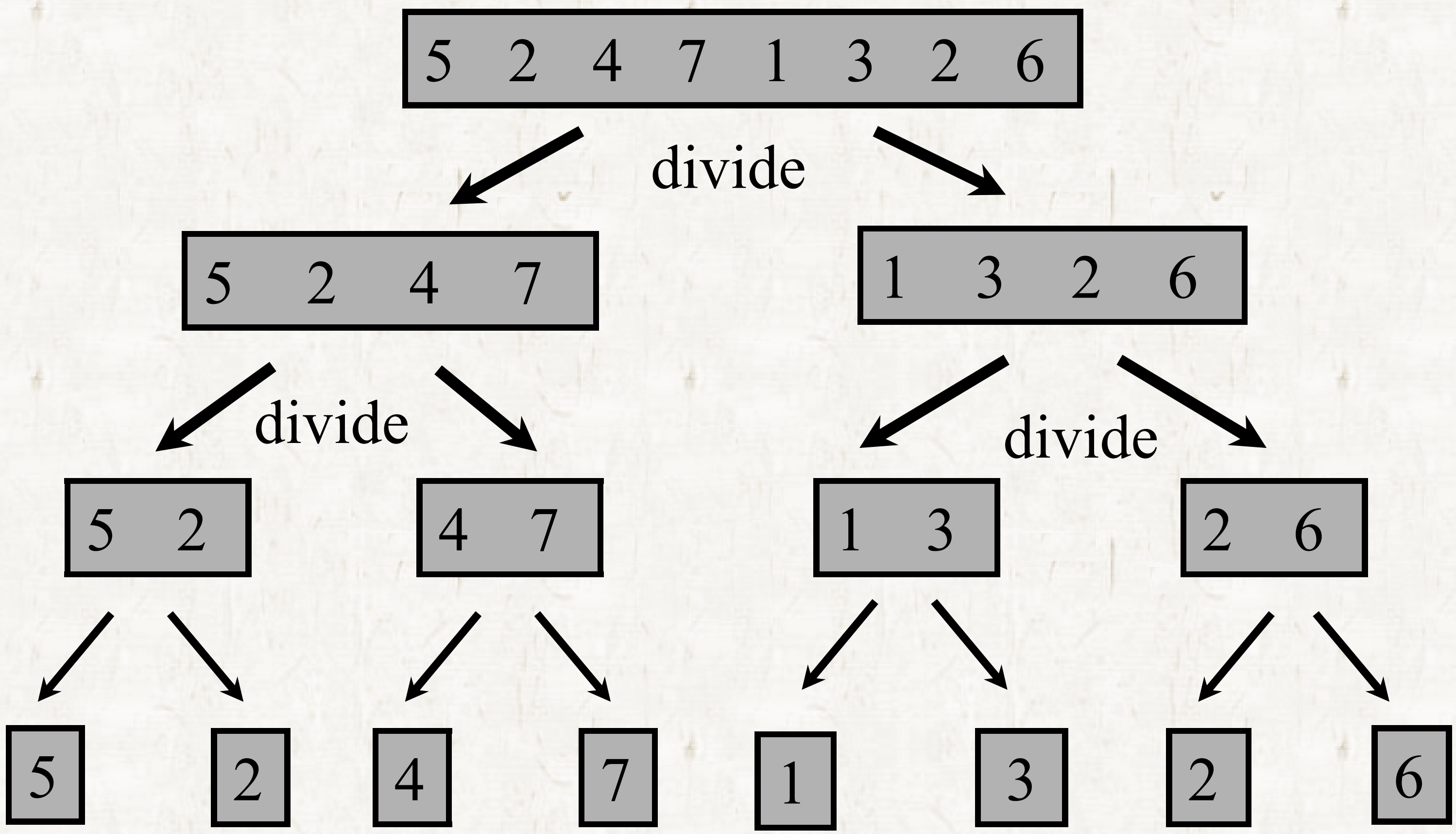
  where q is the halfway point of $A[p .. r]$.
- **Conquer** by recursively sorting the two subarrays

  $A[p .. q]$ and $A[q+1 .. r]$.
- **Combine** by merging the two sorted subarrays

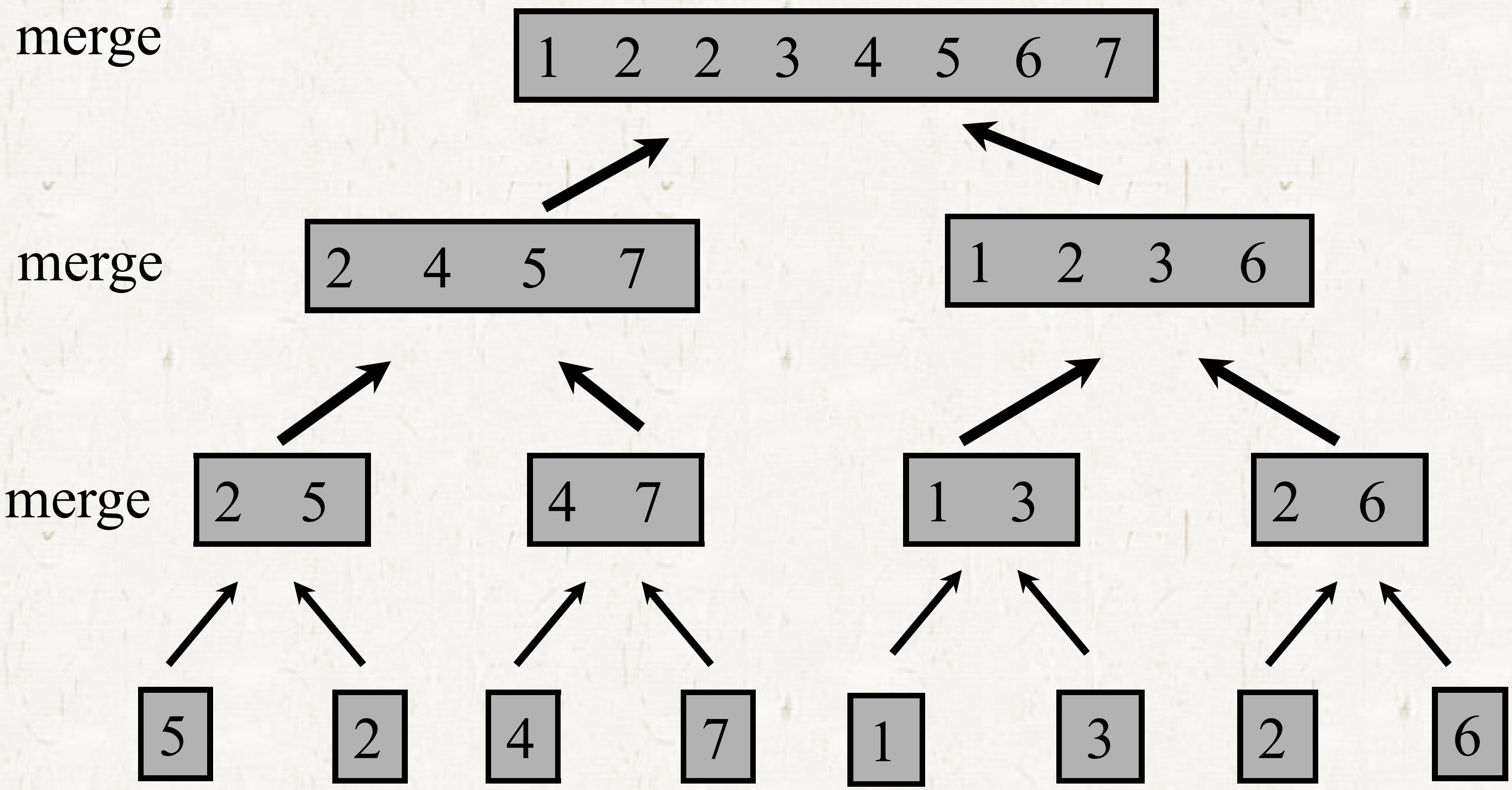
  $A[p .. q]$ and $A[q+1 .. r]$

  to produce a single sorted array $A[p .. r]$.

Merge sort: Divide Scenario



Merge sort: Merge Scenario



Merge sort: Merge Operation

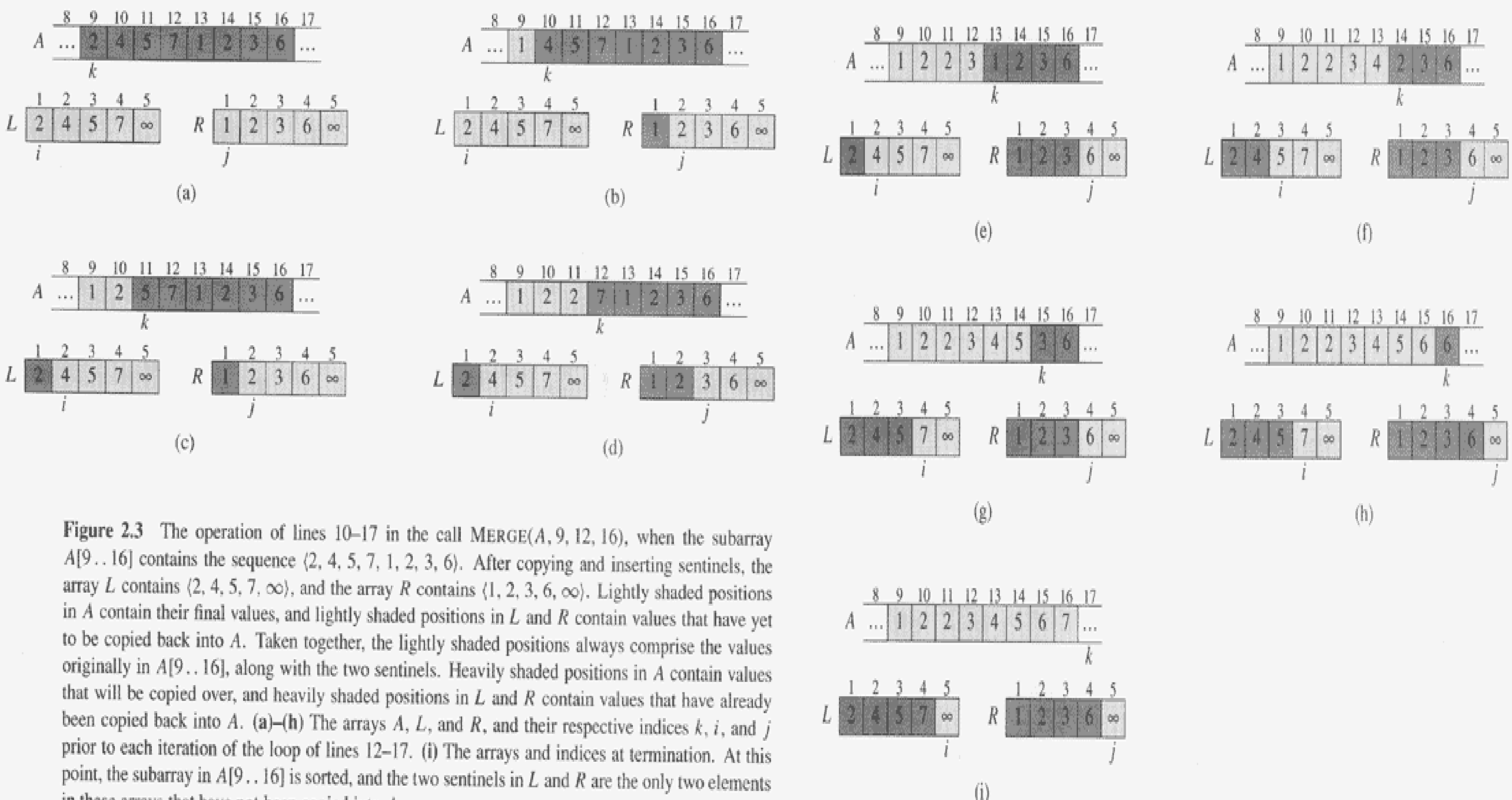


Figure 2.3 The operation of lines 10–17 in the call MERGE(A, 9, 12, 16), when the subarray $A[9..16]$ contains the sequence $\langle 2, 4, 5, 7, 1, 2, 3, 6 \rangle$. After copying and inserting sentinels, the array L contains $\langle 2, 4, 5, 7, \infty \rangle$, and the array R contains $\langle 1, 2, 3, 6, \infty \rangle$. Lightly shaded positions in A contain their final values, and lightly shaded positions in L and R contain values that have yet to be copied back into A. Taken together, the lightly shaded positions always comprise the values originally in $A[9..16]$, along with the two sentinels. Heavily shaded positions in A contain values that will be copied over, and heavily shaded positions in L and R contain values that have already been copied back into A. **(a)–(h)** The arrays A, L, and R, and their respective indices k, i, and j prior to each iteration of the loop of lines 12–17. **(i)** The arrays and indices at termination. At this point, the subarray in $A[9..16]$ is sorted, and the two sentinels in L and R are the only two elements in these arrays that have not been copied into A.

Pseudo code: Merge Procedure

```
MERGE(A, p, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i ← 1 to n1
 5       do L[i] ← A[p + i − 1]
 6  for j ← 1 to n2
 7       do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13       do if L[i] ≤ R[j]
14             then A[k] ← L[i]
15                  i ← i + 1
16             else A[k] ← R[j]
17                  j ← j + 1
```

Line 1-3 and 8-11 takes constant time and for loop takes *Θ(n1+n2) time*

Execute r-p+1 times

Pseudo code: Merge sort Procedure

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

Running time : Merge sort (1)

- **Divide:** $\boldsymbol{D(n)} = \Theta(1)$
  - The divide step just computes the middle of the subarray, which takes constant time.
- **Conquer:** $2T\,(n/2)$ $[a = b = 2]$.
  - We recursively solve two subproblems, each of size $n/2$.
- **Combine:** $\boldsymbol{C(n)} = \Theta(n)$
  - We have already showed that merging two sorted lists of size n/2 takes $\Theta(n)$ time.

Running time : Merge sort (2)

- ***D(n)+C(n)*** = $\Theta(1) + \Theta(n) = \Theta(n)$
- $T(n)$ can be represented as a recurrence.

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

Running time : Merge sort (3)

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

$$\Downarrow$$

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

- Where the constant c represents the time required to solve problems of size 1 as well as the time per array element of the divide and combine steps.

Running time : Merge sort (4): Recursion tree

$T(n)$

cn

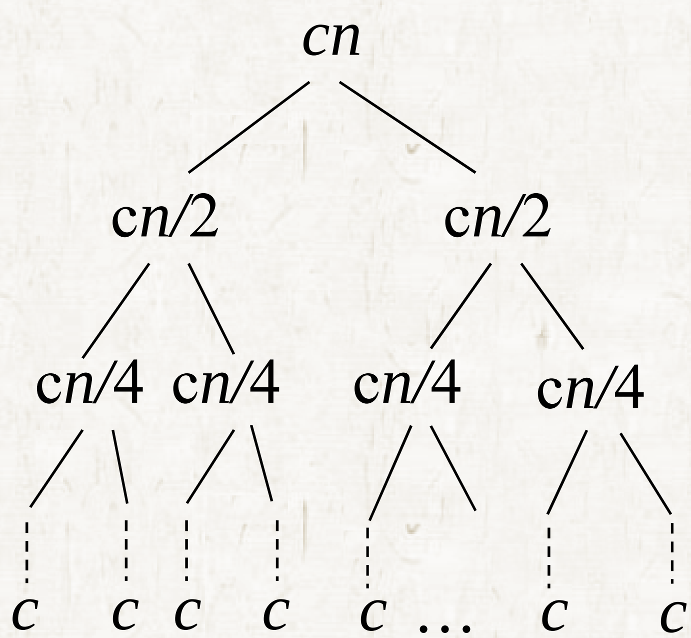
$T(n/2)$ $T(n/2)$

cn

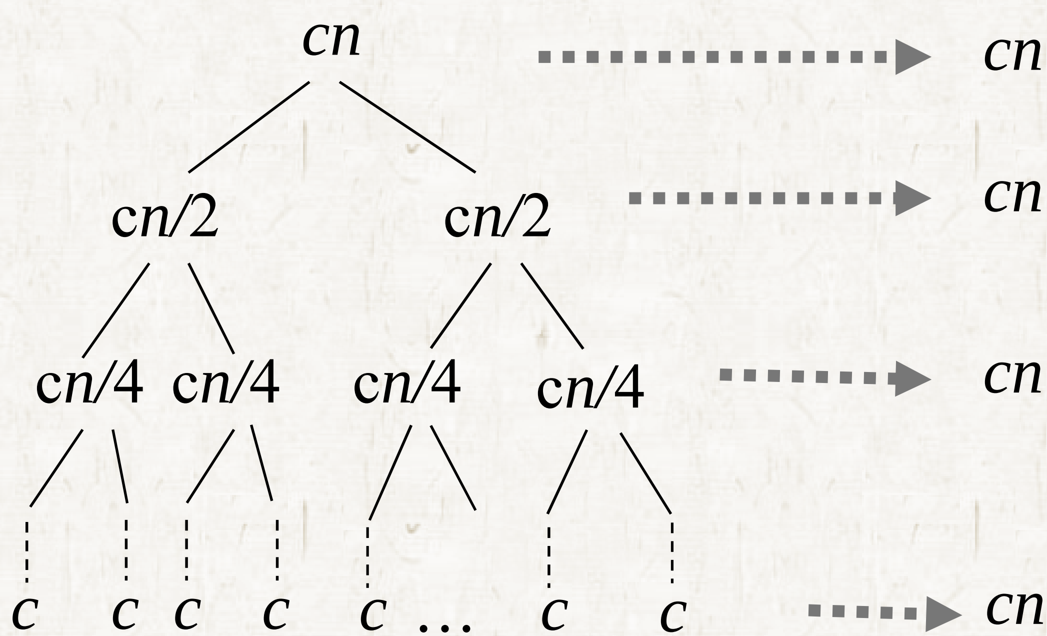
$cn/2$ $cn/2$

$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$

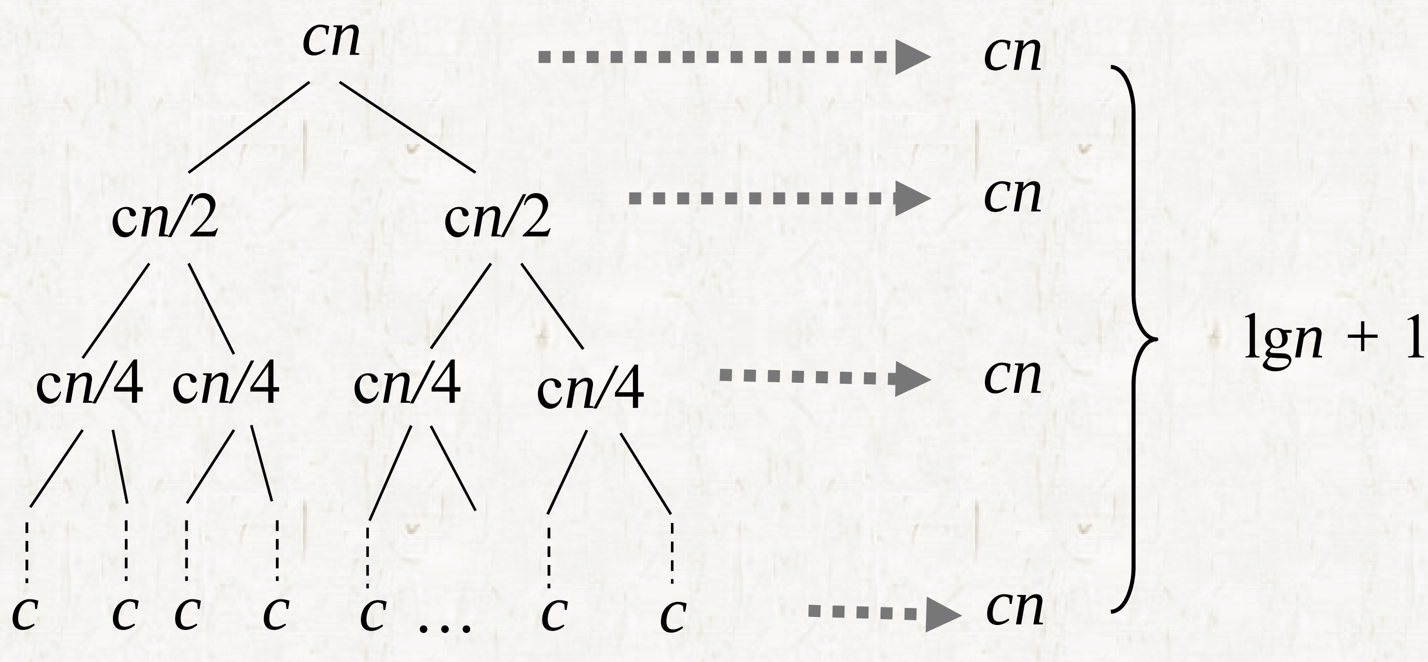
Running time : Merge sort (5): Recursion tree



Running time : Merge sort (6): Recursion tree



Running time : Merge sort (7): Recursion tree



Total : $cn\lg n + cn = \Theta(n \lg n)$

Running time : Merge sort (8): Substitution

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

Solution:

$$\begin{aligned}
T(n) &= 2T(n/2) + cn \\
&= 2\left\{2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right\} + cn \\
&= 2^2T\left(\frac{n}{2^2}\right) + cn + cn \\
&= 2^2T\left(\frac{n}{2^2}\right) + 2cn \\
&= 2^2\left\{2T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right\} + 2cn \\
&= 2^3T\left(\frac{n}{2^3}\right) + cn + 2cn \\
&= 2^3T\left(\frac{n}{2^3}\right) + 3cn \\
&\ \ \vdots \\
&= 2^kT\left(\frac{n}{2^k}\right) + kcn \\
&= nT(1) + cn\lg n \\
&= cn + cn\lg n \quad [T(1) = c] \\
&= \Theta(n\lg n).
\end{aligned}$$

$$\begin{aligned}
&\frac{n}{2^k} = 1 \\
&\Rightarrow n = 2^k \\
&\Rightarrow \lg n = \lg 2^k = k \\
&\therefore k = \lg n
\end{aligned}$$

Stay Safe