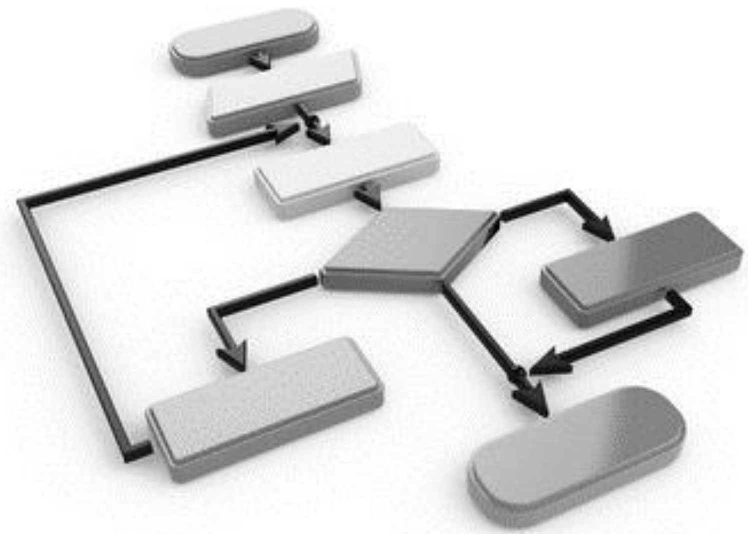


Lecture 02

Divide and Conquer

CSE373: Design and Analysis of Algorithms

Divide and Conquer

Recursive in structure

- ***Divide*** the problem into independent sub-problems that are similar to the original but smaller in size
- ***Conquer*** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
- ***Combine*** the solutions to create a solution to the original problem

Merge Sort

6	18	56	62	1	9	15	43

Merge Sort

Unsorted

6	18	56	62	1	9	15	43

Sorted **Sorted**

Merge Sort

6	18	56	62	1	9	15	43

Merging

Merge Sort

6	18	56	62	1	9	15	43

Merging

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1							

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1							

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6						

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6						

- Left half
- Right half
- Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9					

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9					

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15				

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15				

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18			

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18			

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18	43		

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18	43		

Left half

Right half

Minimum between first elements in both halves

Merge Sort

6	18	56	62	1	9	15	43

Merging

1	6	9	15	18	43	56	62

Left half

Right half

Minimum between first elements in both halves

Merge Sort

1	6	9	15	18	43	56	62

Merging

1	6	9	15	18	43	56	62

Left half

Right half

Minimum between first elements in both halves

Merge Sort

1	6	9	15	18	43	56	62

Merge Sort

```
Merge(A, p, q, r)
n1 ← q – p + 1
n2 ← r – q
    for i ← 1 to n1
        do L[i] ← A[p + i – 1]
    for j ← 1 to n2
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ←p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                i ← i + 1
            else A[k] ← R[j]
                j ← j + 1
```

Input: Array containing sorted subarrays A[p..q] and A[q+1..r].

Output: Merged sorted subarray in A[p..r].

***Sentinels**, to avoid having to check if either subarray is fully copied at each step.*

Merge Sort

Divide: Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each

Conquer: Sort the two subsequences recursively using merge sort.

Combine: Merge the two sorted subsequences to produce the sorted answer.

Merge Sort

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
    if p < r
        then q ← ⌊(p+r)/2⌋
            MergeSort (A, p, q)
            MergeSort (A, q+1, r)
            Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: *MergeSort(A, 1, n)*

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

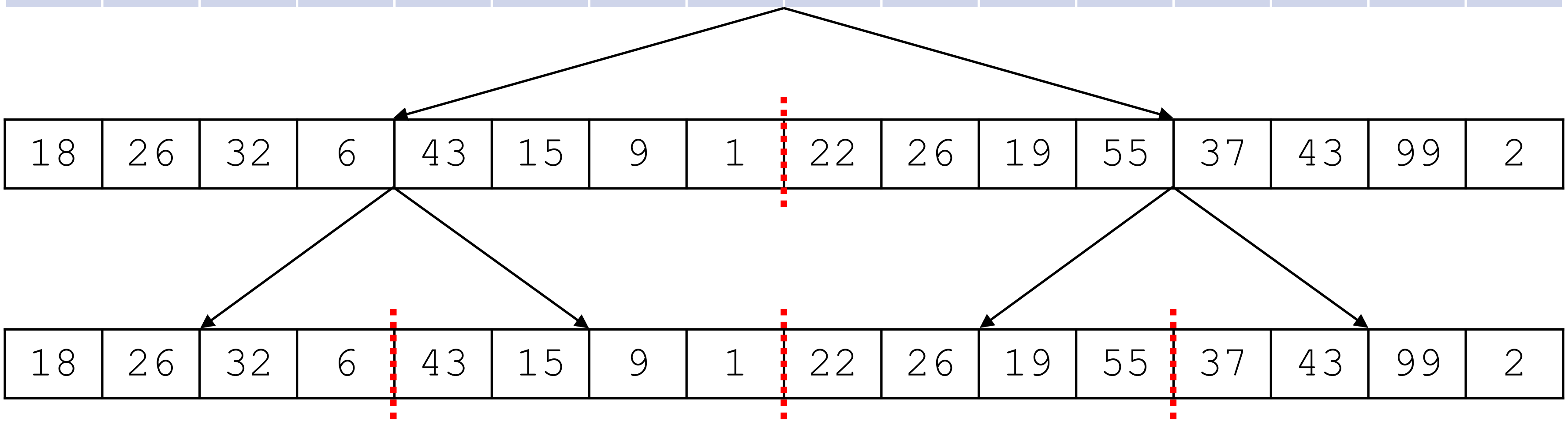
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

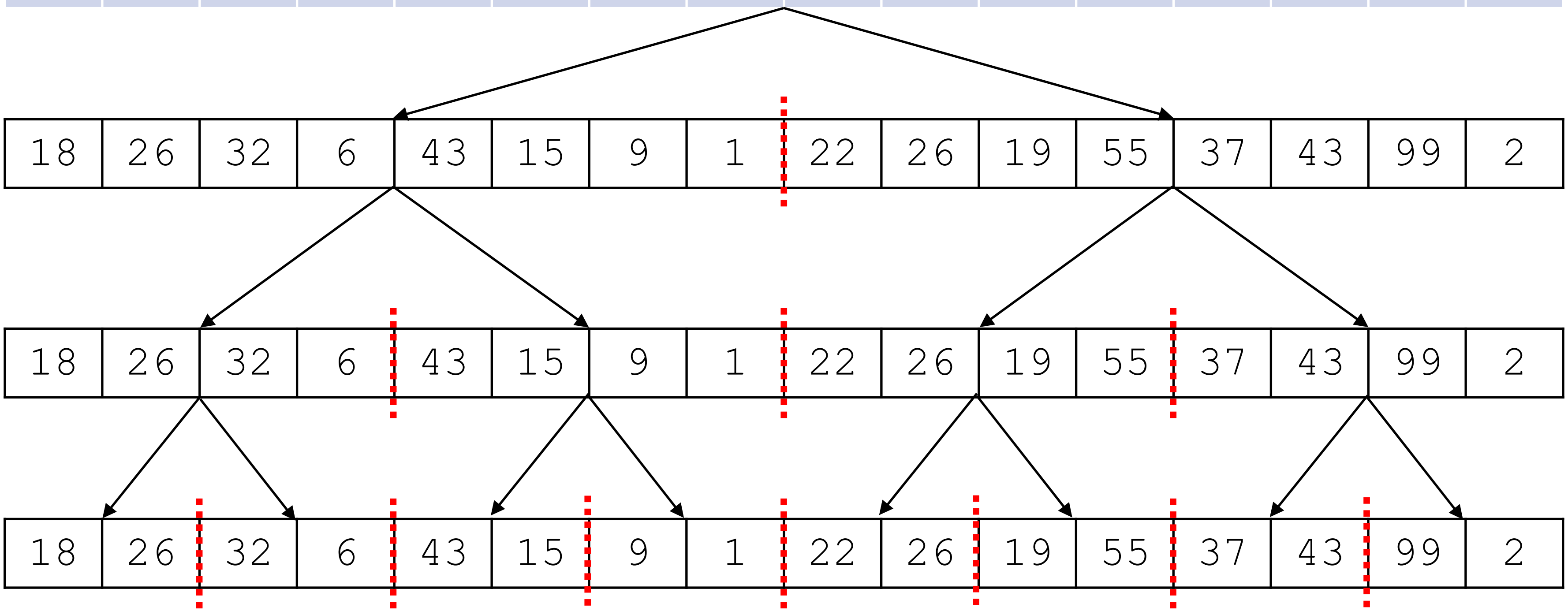
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2



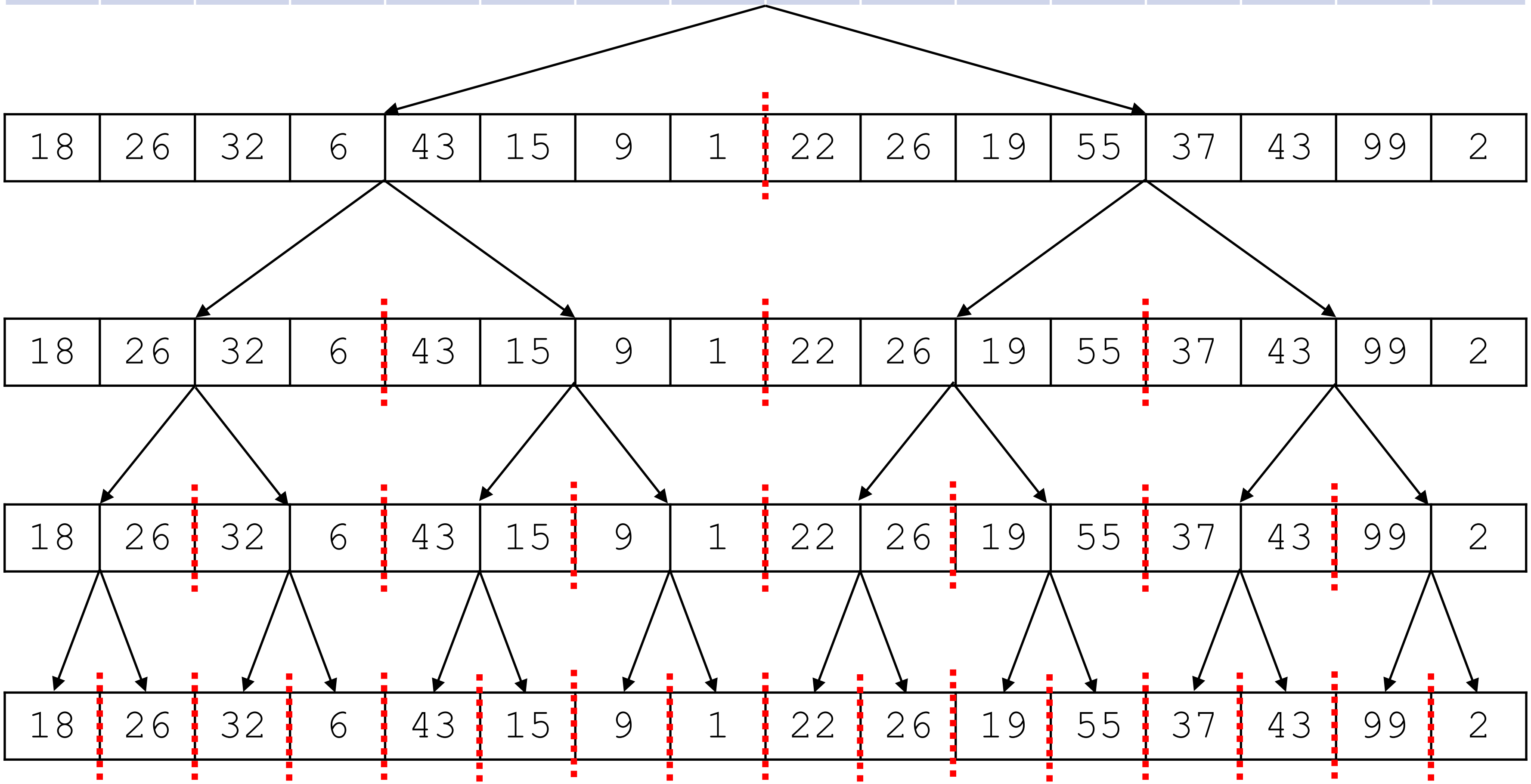
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2



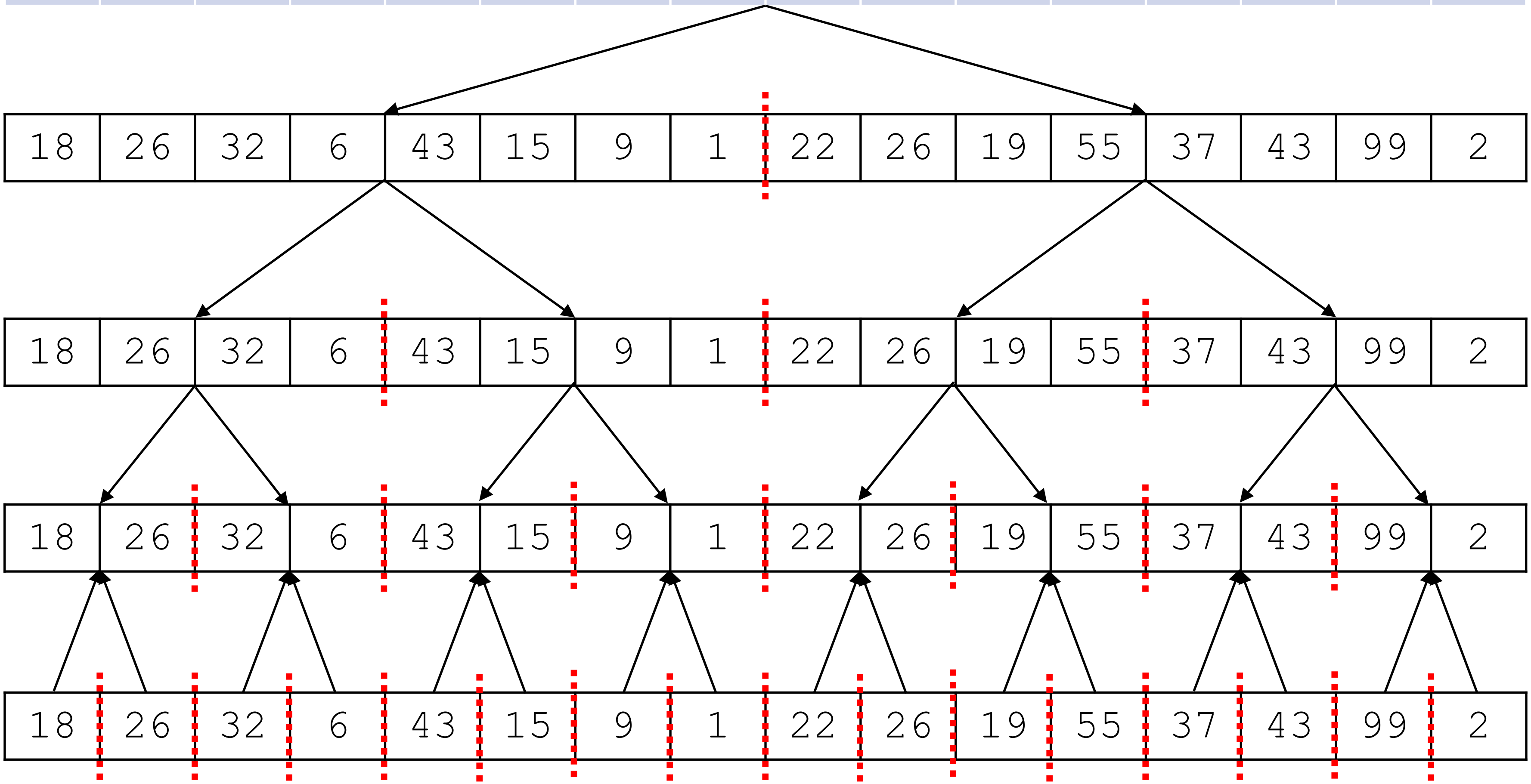
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2



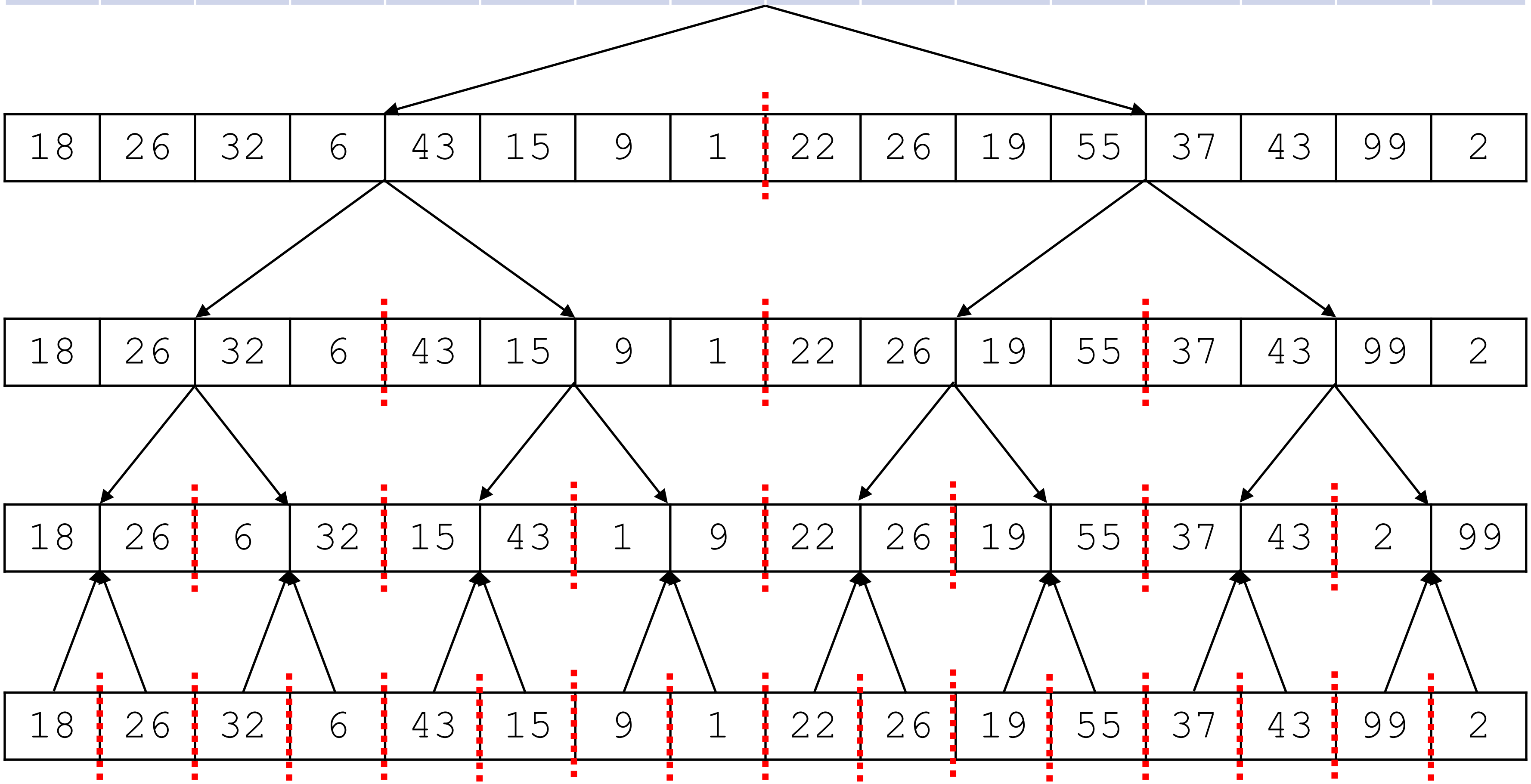
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2



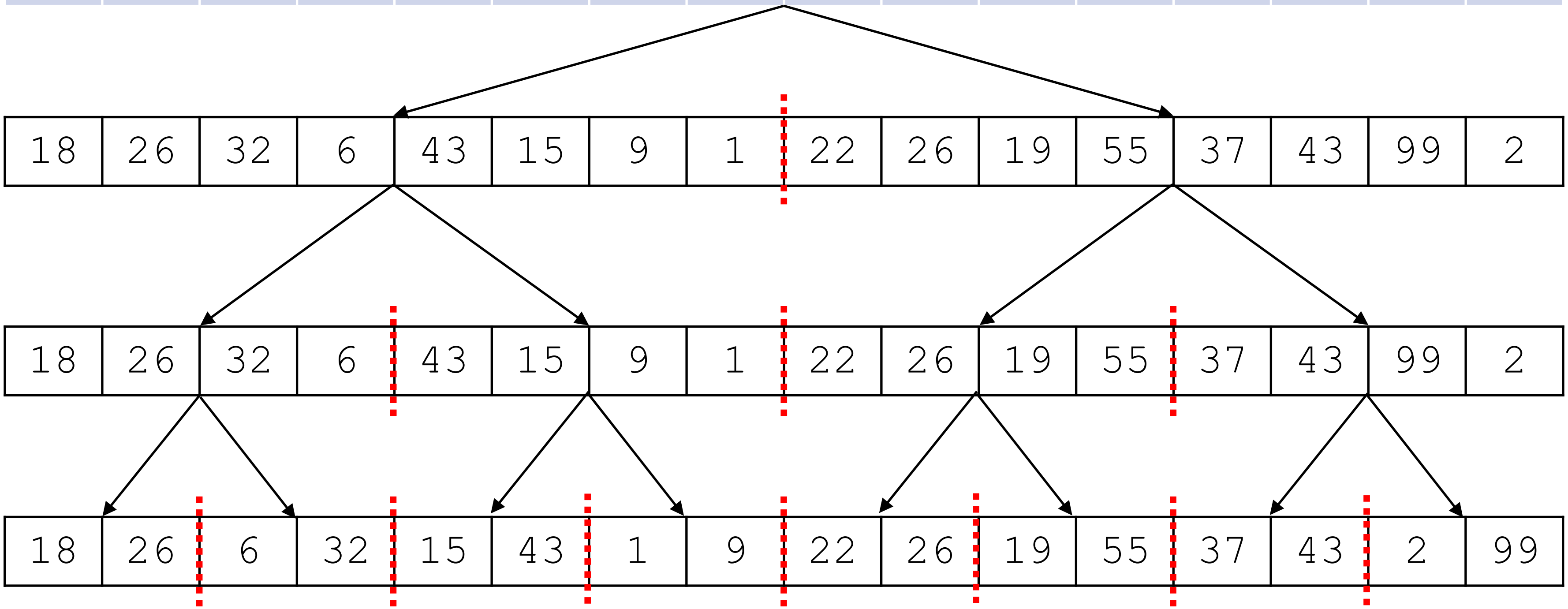
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2



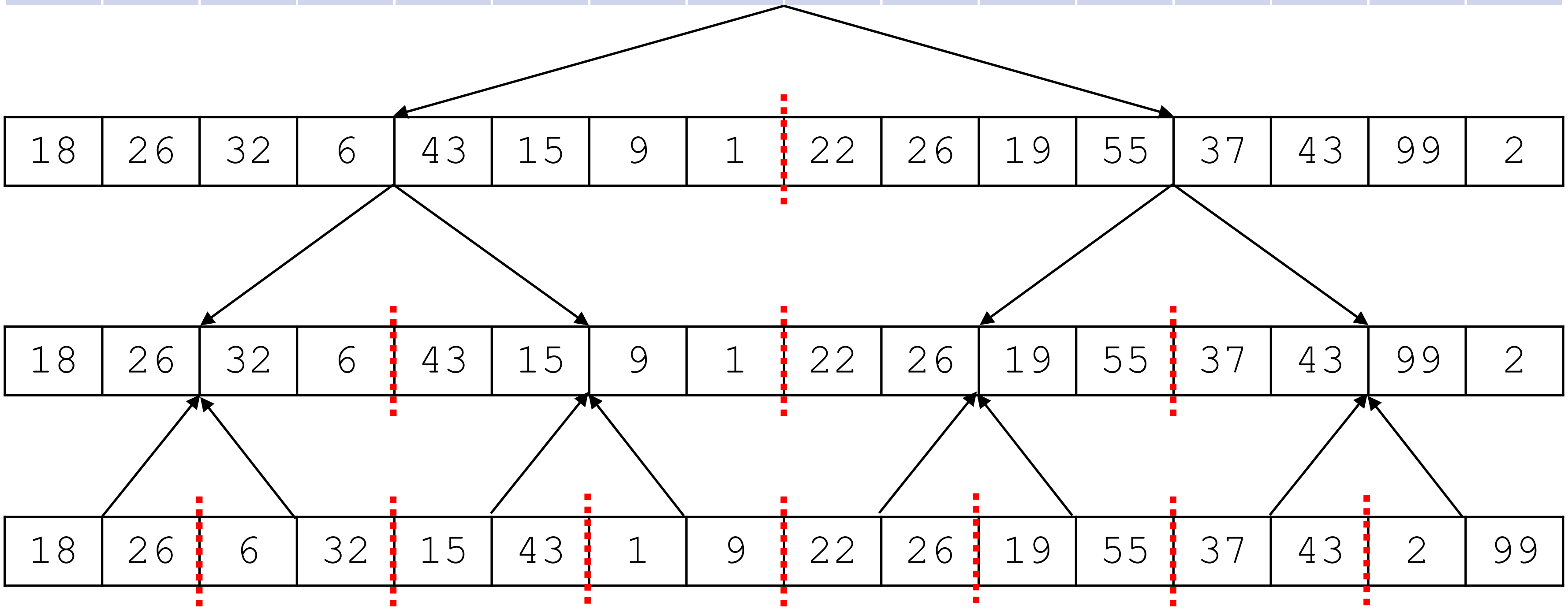
Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

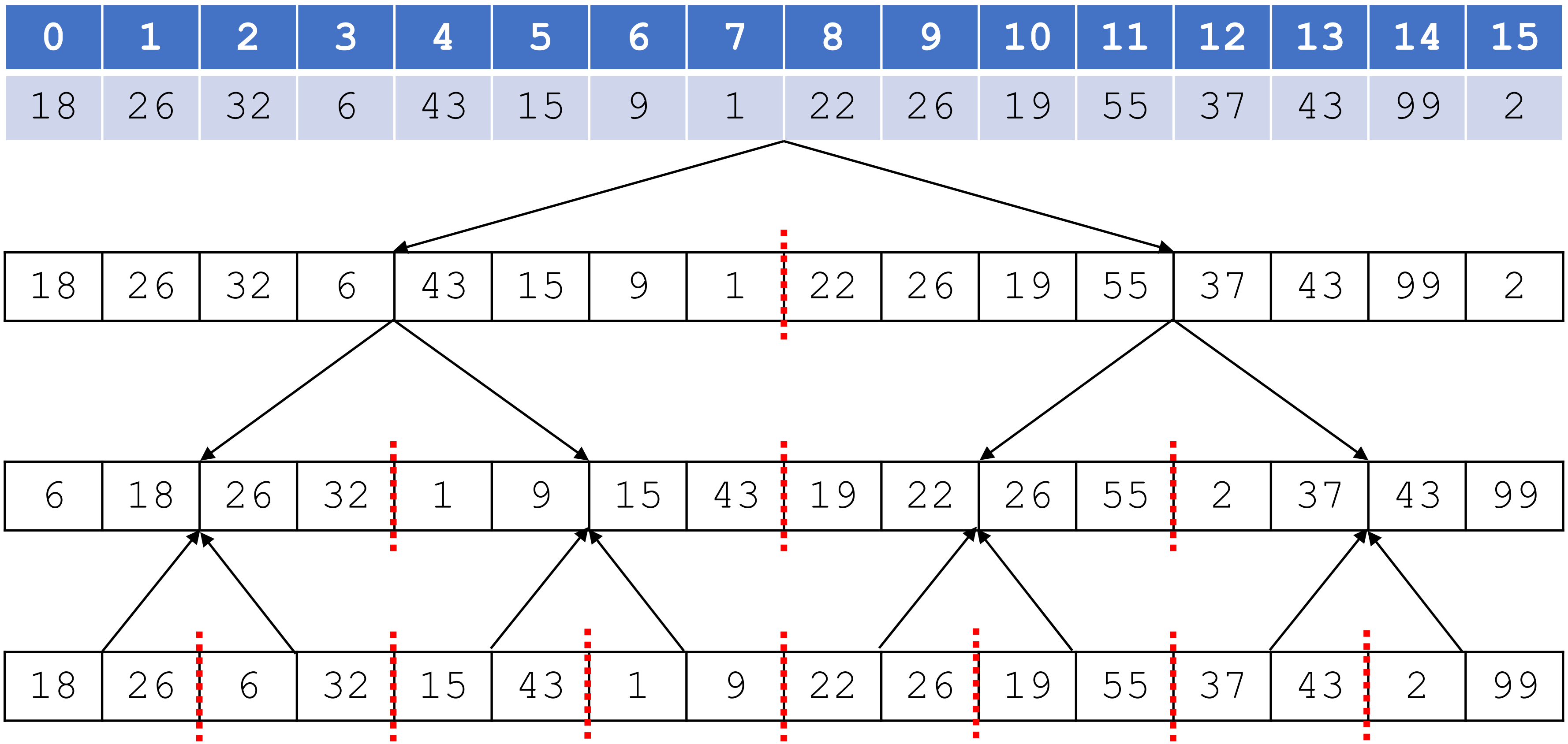


Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2



Merge Sort



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
18 26 32 6 43 15 9 1 22 26 19 55 37 43 99 2
18 26 32 6 43 15 9 1 22 26 19 55 37 43 99 2
6 18 26 32 1 9 15 43 19 22 26 55 2 37 43 99
18 26 6 32 15 43 1 9 22 26 19 55 37 43 2 99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

6	18	26	32	1	9	15	43	19	22	26	55	2	37	43	99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

6	18	26	32	1	9	15	43	19	22	26	55	2	37	43	99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

1	6	9	15	18	26	32	43	2	19	22	26	37	43	55	99

6	18	26	32	1	9	15	43	19	22	26	55	2	37	43	99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

1	6	9	15	18	26	32	43	2	19	22	26	37	43	55	99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
18	26	32	6	43	15	9	1	22	26	19	55	37	43	99	2

1	6	9	15	18	26	32	43	2	19	22	26	37	43	55	99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	6	9	15	18	19	22	26	26	32	37	43	43	55	99

1	6	9	15	18	26	32	43	2	19	22	26	37	43	55	99

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	6	9	15	18	19	22	26	26	32	37	43	43	55	99

Analysis of Merge Sort

Time complexity of divide and conquer approach: The original problem is divided into a sub-problems, each of which is $1/b$ the size of the original. The cost for dividing is $D(n)$ and the cost for combining is $C(n)$.

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

Merge Sort:

[Divide] $D(n) = \Theta(1)$

[Conquer] $2 * T(n/2)$

[Combine] $C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & if\ n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & if\ n > 1 \\ \qquad = \Theta(n \log_2 n) & \end{cases}$$

Recurrence Relations

Equation or an inequality that characterizes a function by its values on smaller inputs.

Recurrence relations arise when we analyze the running time of iterative or recursive algorithms.

Ex: Divide and Conquer.

$T(n) = \Theta(1)$ if $n \leq c$

$T(n) = a\,T(n/b) + D(n) + C(n)$ otherwise

Solution Methods

Substitution Method.

Recursion-tree Method.

Master Method.

Substitution Method

Guess the form of the solution, then
use mathematical induction to show it correct.

> Substitute guessed answer for the function when the inductive hypothesis is applied to smaller values – hence, the name.

Works well when the solution is easy to guess.

No general way to guess the correct solution.

Substitution Method

Recurrence: $T(n) = 1$ if $n = 1$

$T(n) = 2T(n/2) + n$ if $n > 1$

- ***Guess:*** $T(n) = n \lg n + n.$
- ***Induction:***
  - **Basis:** $n = 1 \Rightarrow n \lg n + n = 1 = T(n).$
  - **Hypothesis:** $T(k) = k \lg k + k$ for all $k < n.$
  - **Inductive Step:** $T(n) = 2\,T(n/2) + n$

$$
\begin{aligned}
&= 2\,((n/2)\lg(n/2) + (n/2)) + n \\
&= n\,(\lg(n/2)) + 2n \\
&= n \lg n - n + 2n \\
&= n \lg n + n
\end{aligned}
$$

Recursion-tree Method

Making a good guess is sometimes difficult with the substitution method.

Use **recursion trees** to devise good guesses.

Recursion Trees

- Show successive expansions of recurrences using trees.
- Keep track of the time spent on the subproblems of a divide and conquer algorithm.
- Help organize the algebraic bookkeeping necessary to solve a recurrence.

Recursion-tree – Example

Running time of Merge Sort:

$$T(n) = \Theta(1) \qquad \text{if } n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) \qquad \text{if } n > 1$$

Rewrite the recurrence as

$$T(n) = \boldsymbol{c} \qquad \text{if } n = 1$$

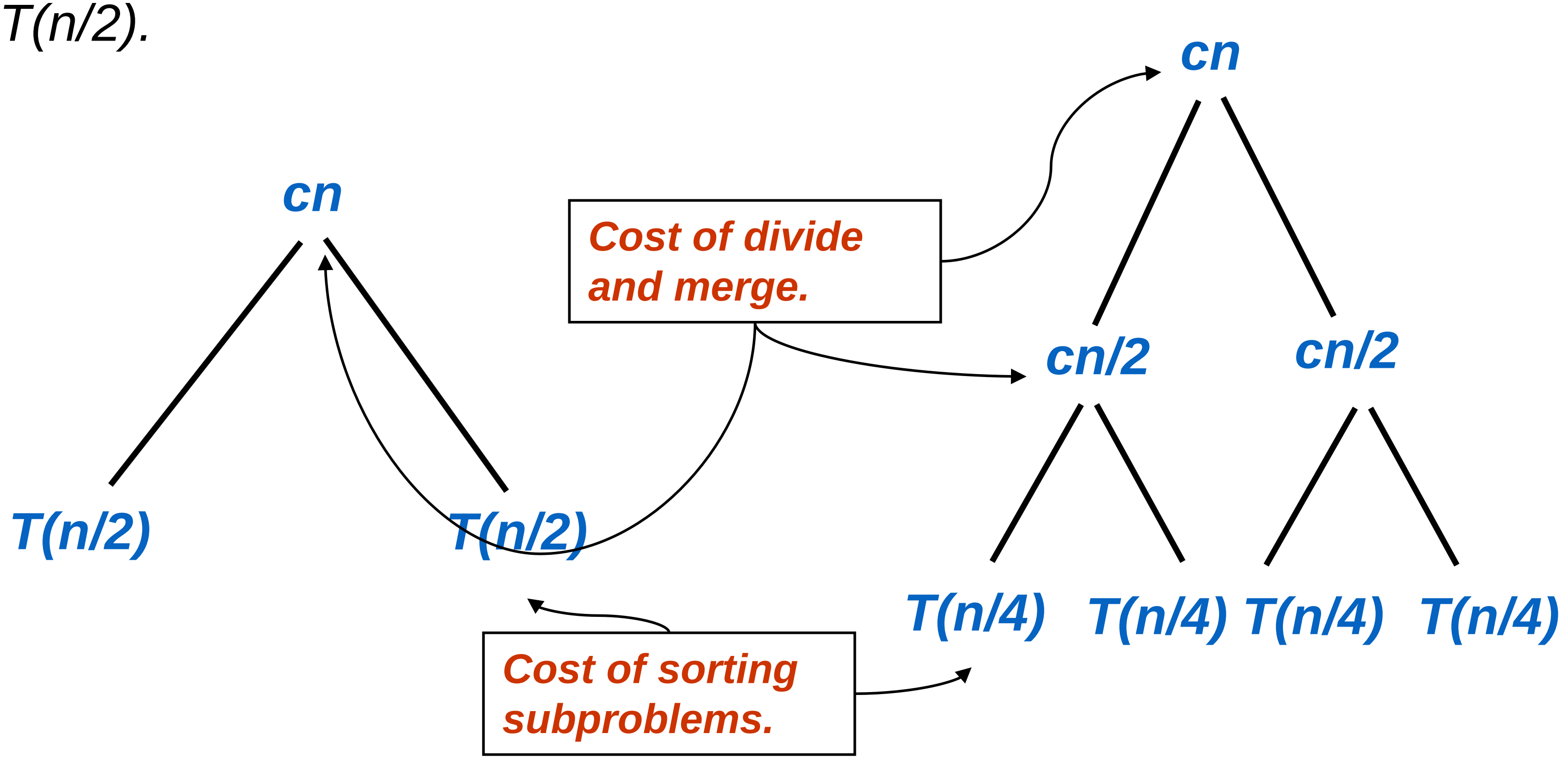
$$T(n) = 2T(n/2) + \boldsymbol{cn} \qquad \text{if } n > 1$$

$\boldsymbol{c > 0}$: Running time for the base case and time per array element for the divide and combine steps.

Recursion Tree for Merge Sort

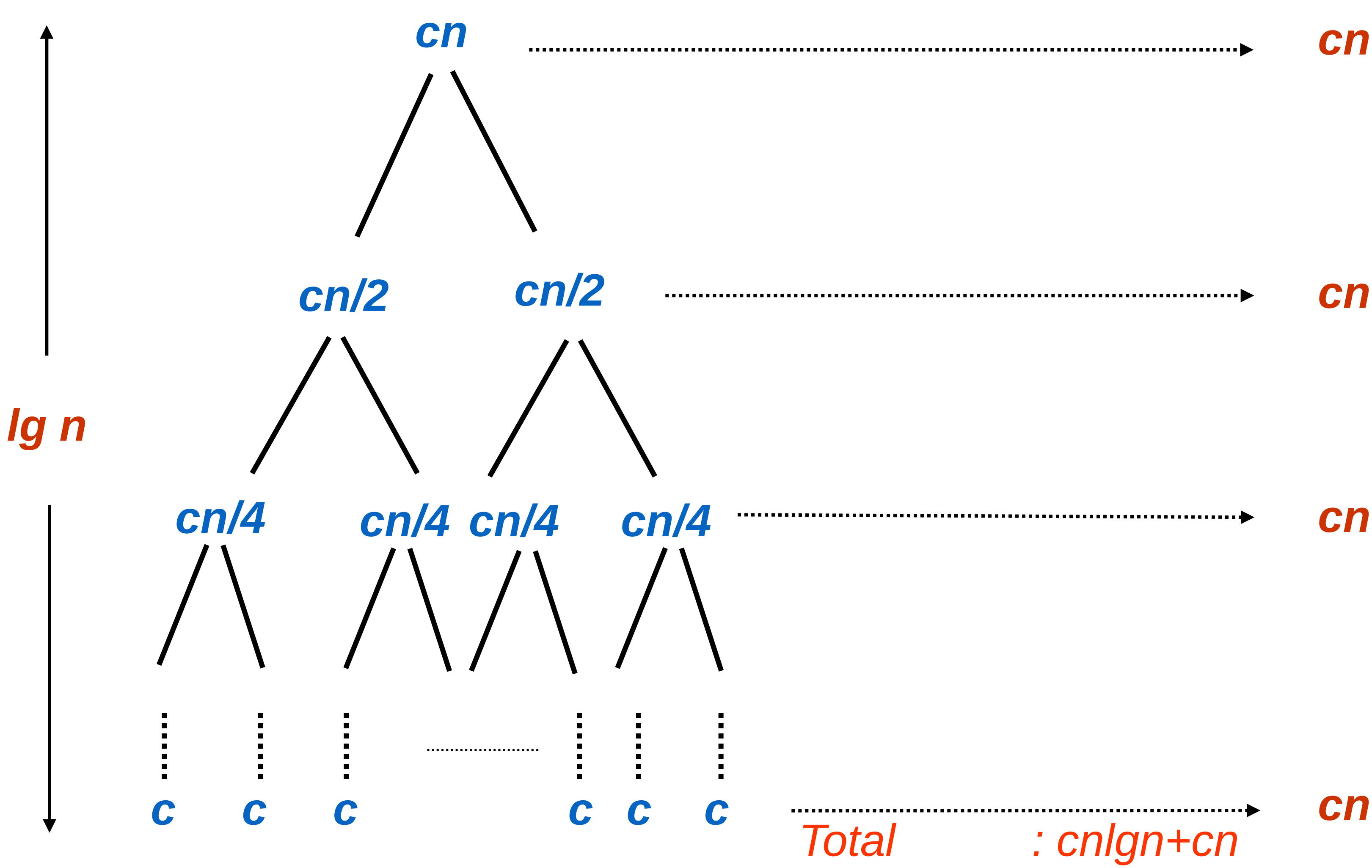
For the original problem, we have a cost of cn, plus two subproblems each of size (n/2) and running time T(n/2).

Each of the size n/2 problems has a cost of cn/2 plus two subproblems, each costing T(n/4).



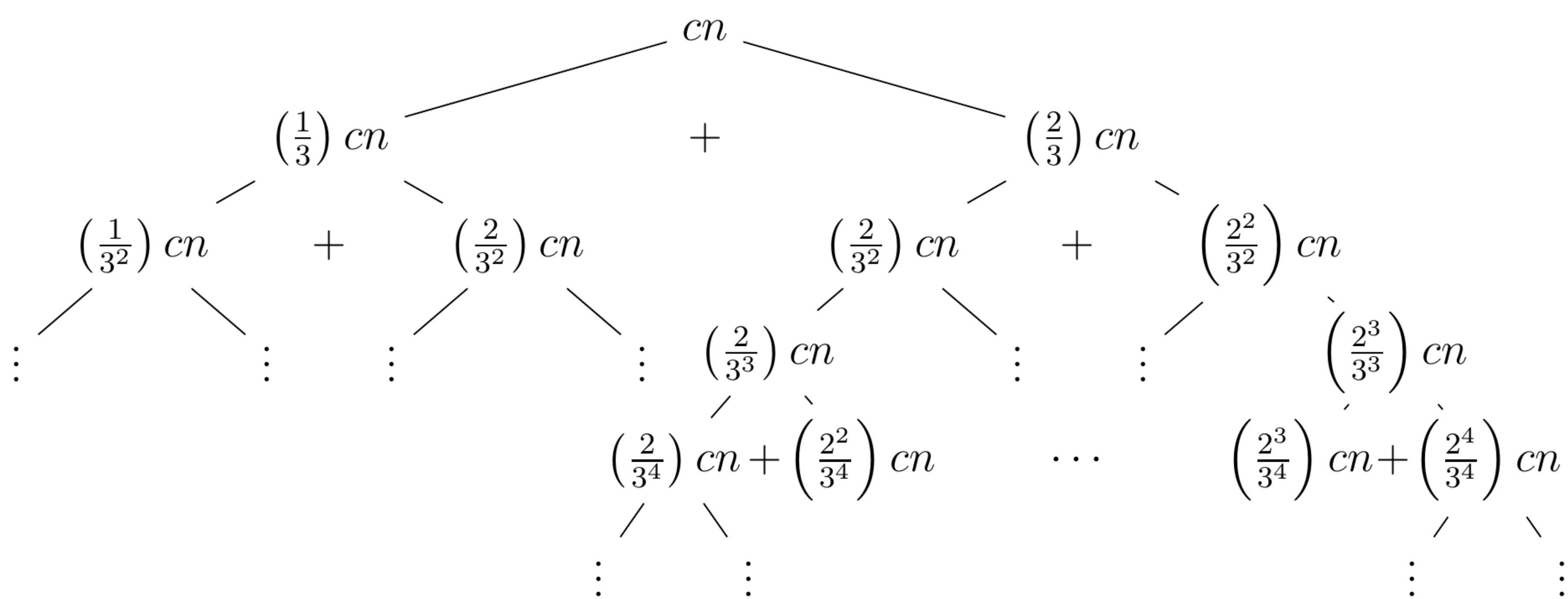
Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



Recursion Tree: Another Example

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



Level 0 : $cn = \left(\frac{1}{3} + \frac{2}{3}\right)^0 cn$

Level 1 : $\left(\frac{1}{3}\right)^1 \left(\frac{2}{3}\right)^0 cn + \left(\frac{1}{3}\right)^0 \left(\frac{2}{3}\right)^1 cn = \left(\frac{1}{3} + \frac{2}{3}\right)^1 cn$

Level 2 : $\sum_{i=0}^{2} \binom{2}{i} \left(\frac{1}{3}\right)^{2-i} \left(\frac{2}{3}\right)^i = \left(\frac{1}{3} + \frac{2}{3}\right)^2 cn$

$\vdots$

Level k : $\sum_{i=0}^{k} \binom{k}{i} \left(\frac{1}{3}\right)^{k-i} \left(\frac{2}{3}\right)^i = \left(\frac{1}{3} + \frac{2}{3}\right)^k cn$