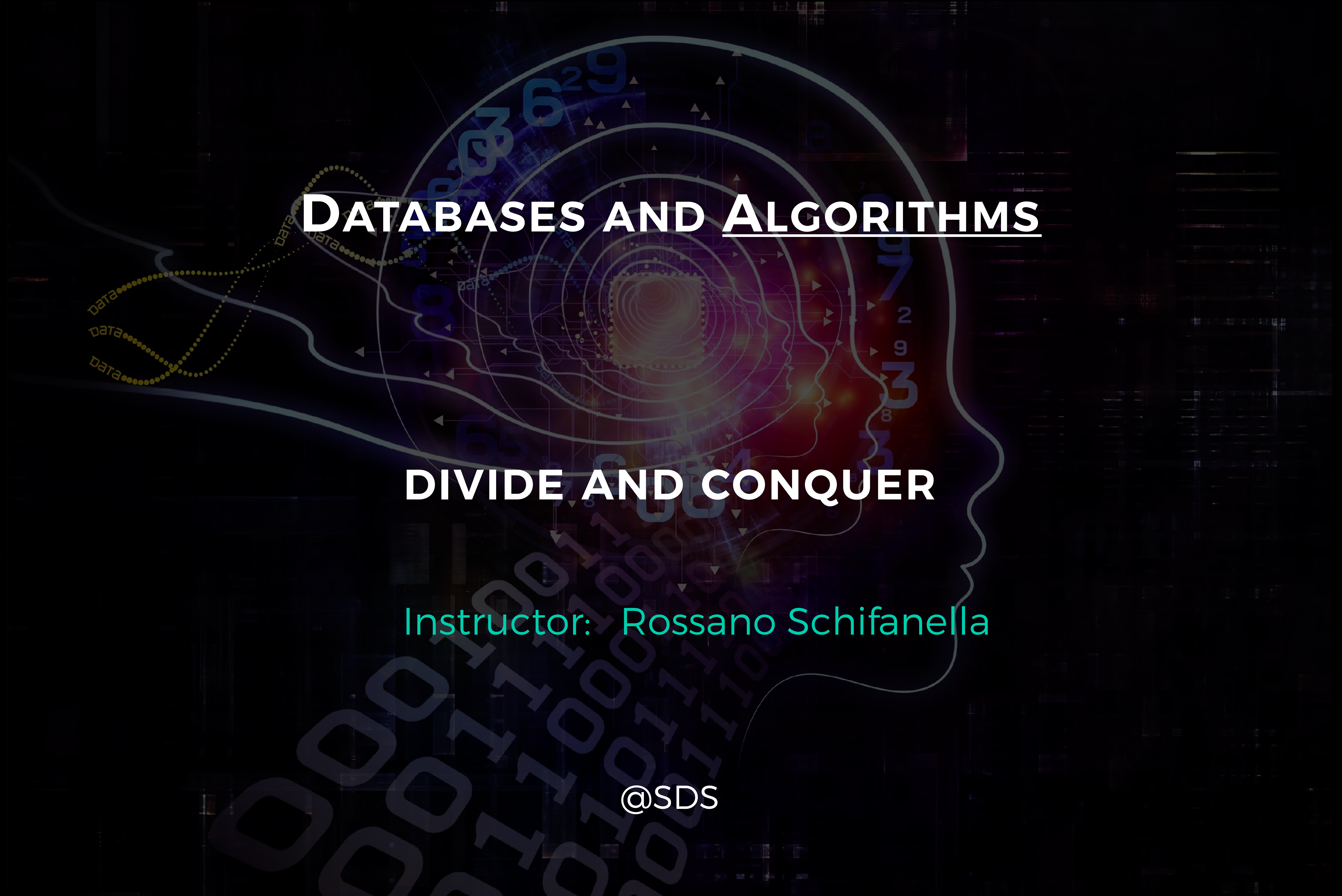


Databases and Algorithms

DIVIDE AND CONQUER

Instructor: Rossano Schifanella

@SDS

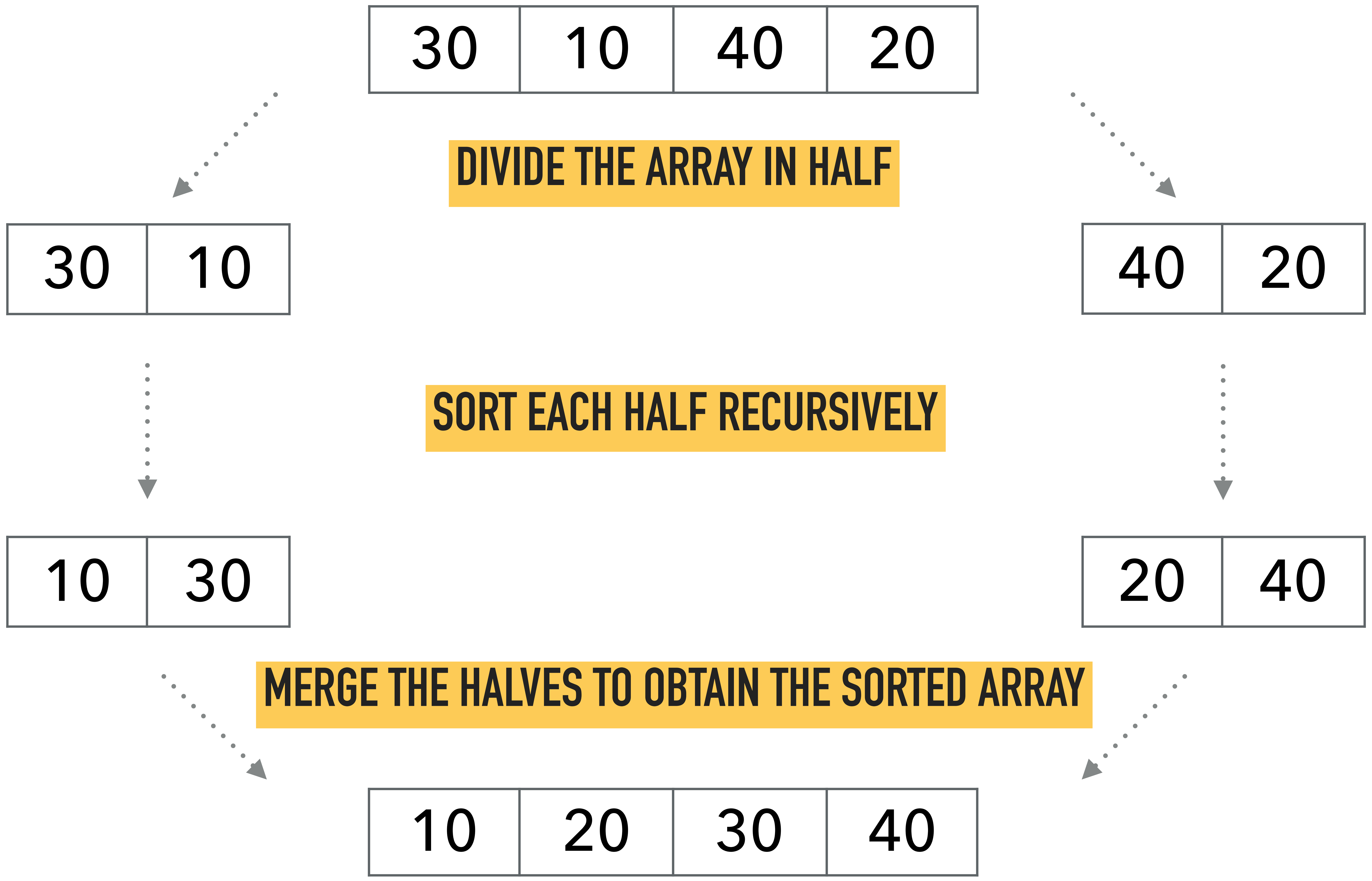
DIVIDE-and-conquer paradigm

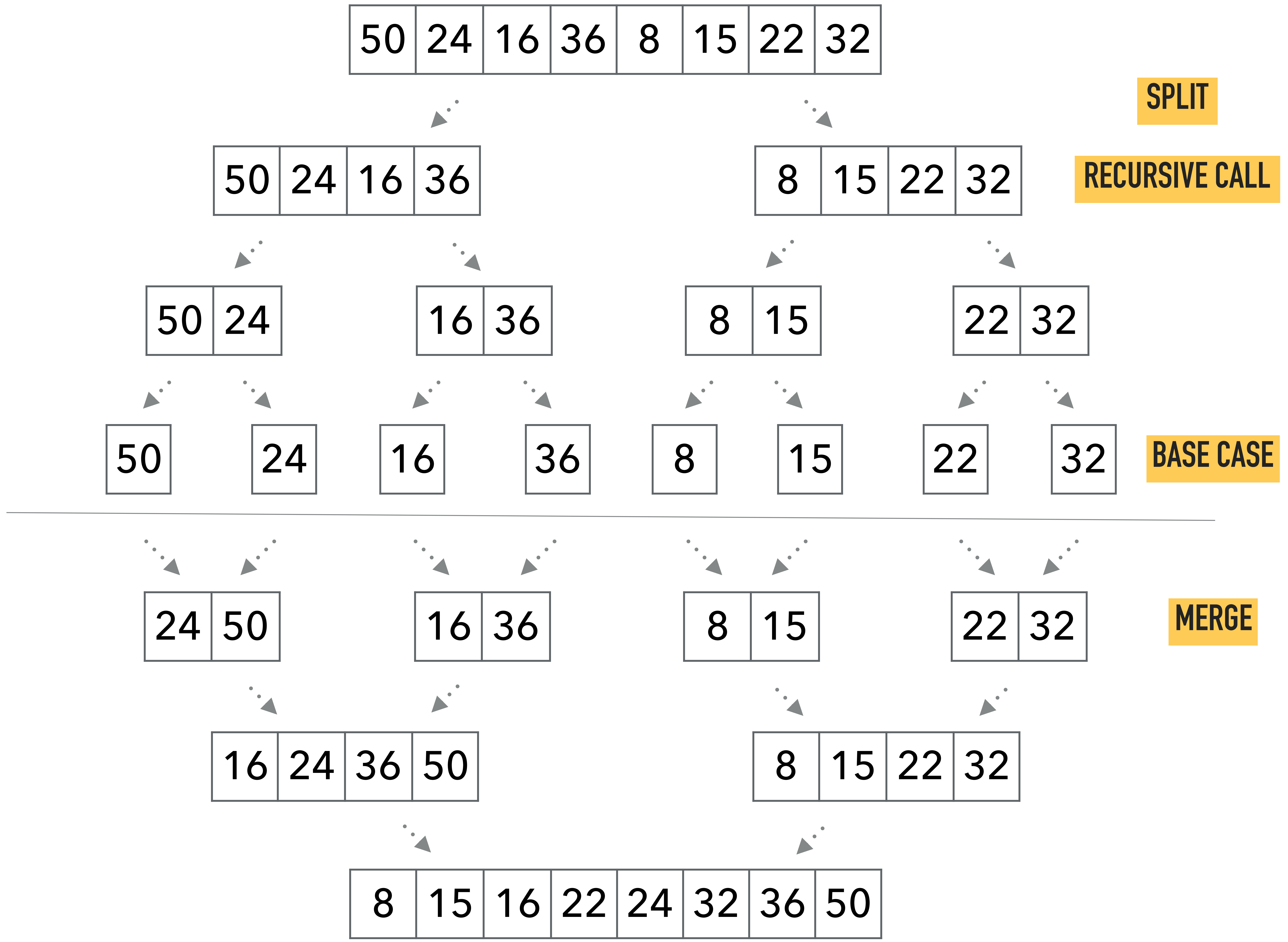
- Many useful algorithms are <u>**recursive**</u> in structure
  - A recursive algorithm is an algorithm which calls itself with "smaller (or simpler)" input values, and which obtains the result for the current input by applying simple operations to the returned value for the smaller (or simpler) input.
  - Example: factorial
    - fact(n) = 1 if $n \leq 1$
    - fact(n) = n * fact(n-1) if $n > 1$

divide-and-conquer paradigm

- The divide-and-conquer paradigm involves **three steps at each level of the recursion**:
  - **DIVIDE** the problem into a number of subproblems that are smaller instances of the same problem.
  - **CONQUER** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
  - **COMBINE** the solutions to the subproblems into the solution for the original problem.

MERGE-SORT


30 10 40 20
DIVIDE THE ARRAY IN HALF
30 10
40 20
SORT EACH HALF RECURSIVELY
10 30
20 40
MERGE THE HALVES TO OBTAIN THE SORTED ARRAY
10 20 30 40


50 24 16 36 8 15 22 32
SPLIT
RECURSIVE CALL
50 24 16 36
8 15 22 32
50 24
16 36
8 15
22 32
50
24
16
36
8
15
22
32
BASE CASE
24 50
16 36
8 15
22 32
MERGE
16 24 36 50
8 15 22 32
8 15 16 22 24 32 36 50

```
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)
```

INPUT: Two subarrays A[p..q] and A[q+1..r] are in sorted order ($p \le q < r$).

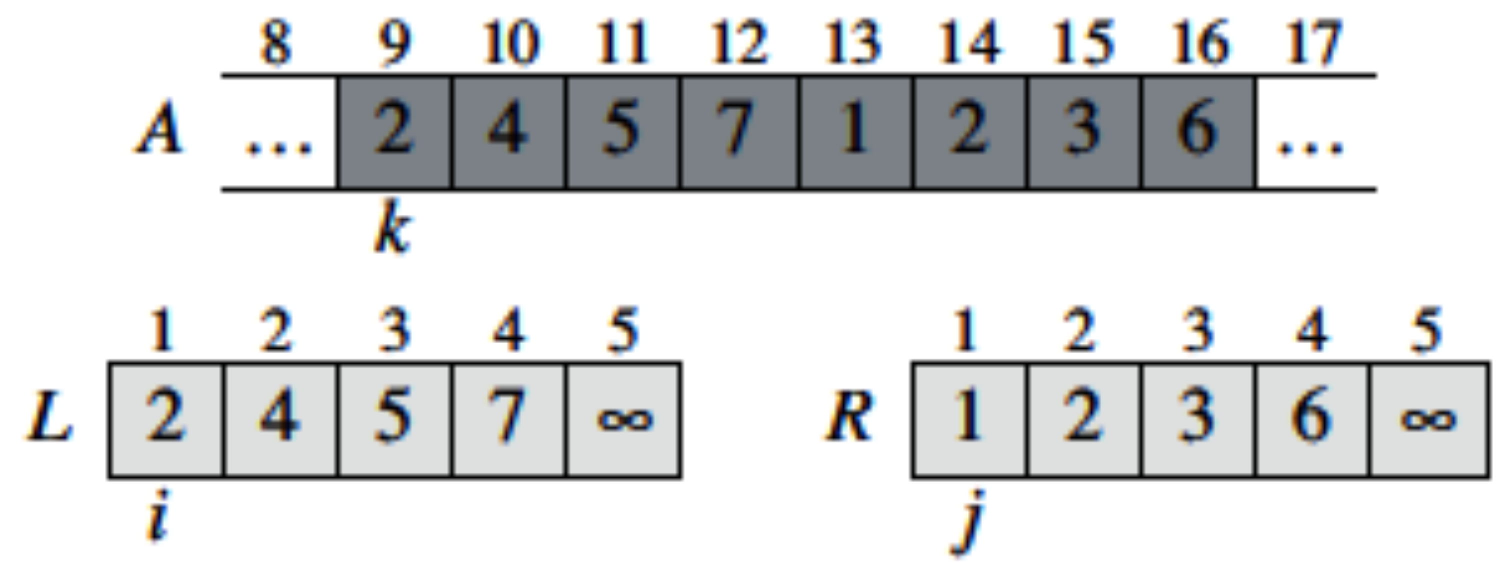
OUTPUT: A single sorted subarray A[p..r] by merging the input subarrays.

```
MERGE(A, p, q, r)
    $n_1 = q-p+1$
    $n_2 = r-q$
    let $L[1..n_1]$ and $R[1..n_2]$ be new arrays containing copy of A[p..q] and A[q+1, r]
    i = 1
    j = 1
    for k = p to r
        if $L[i] \le R[j]$
            A[k] = L[i]
            i = i + 1
        else
            A[k] = R[j]
            j = j + 1
```

MERGE TWO SORTED SUBARRAYS

Merge

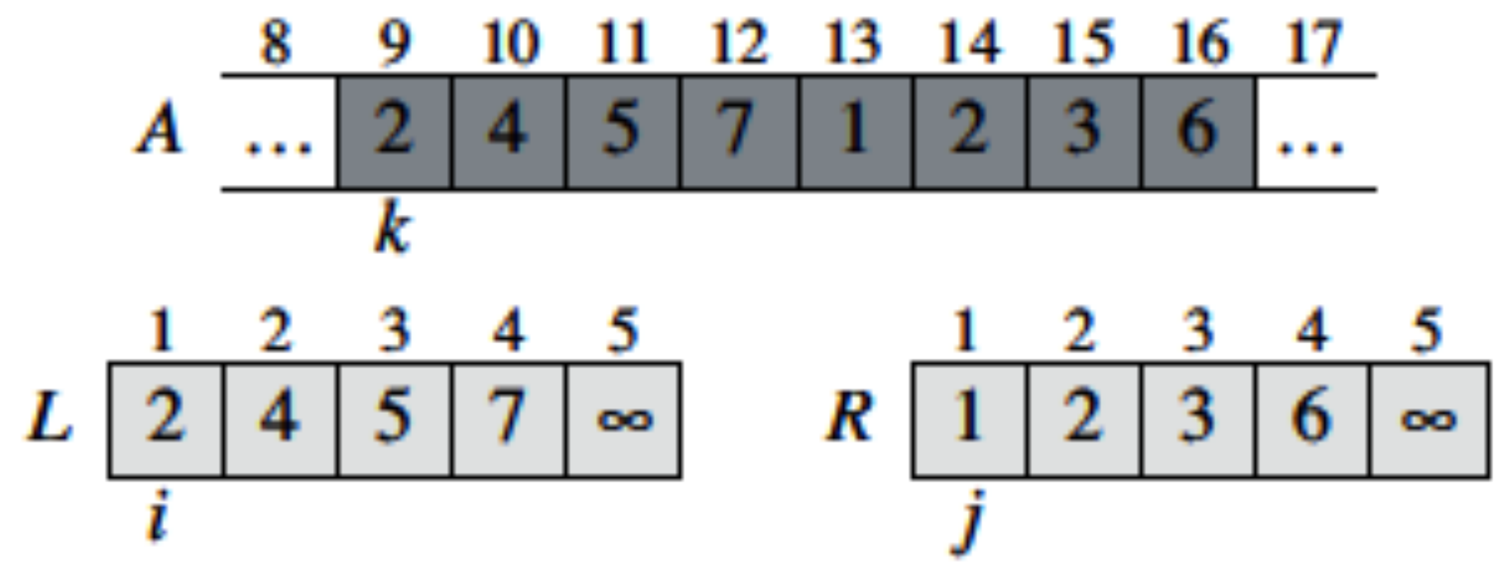
IN THIS EXAMPLE, p=9 AND r =16, THEN q=(9+16)/2=12



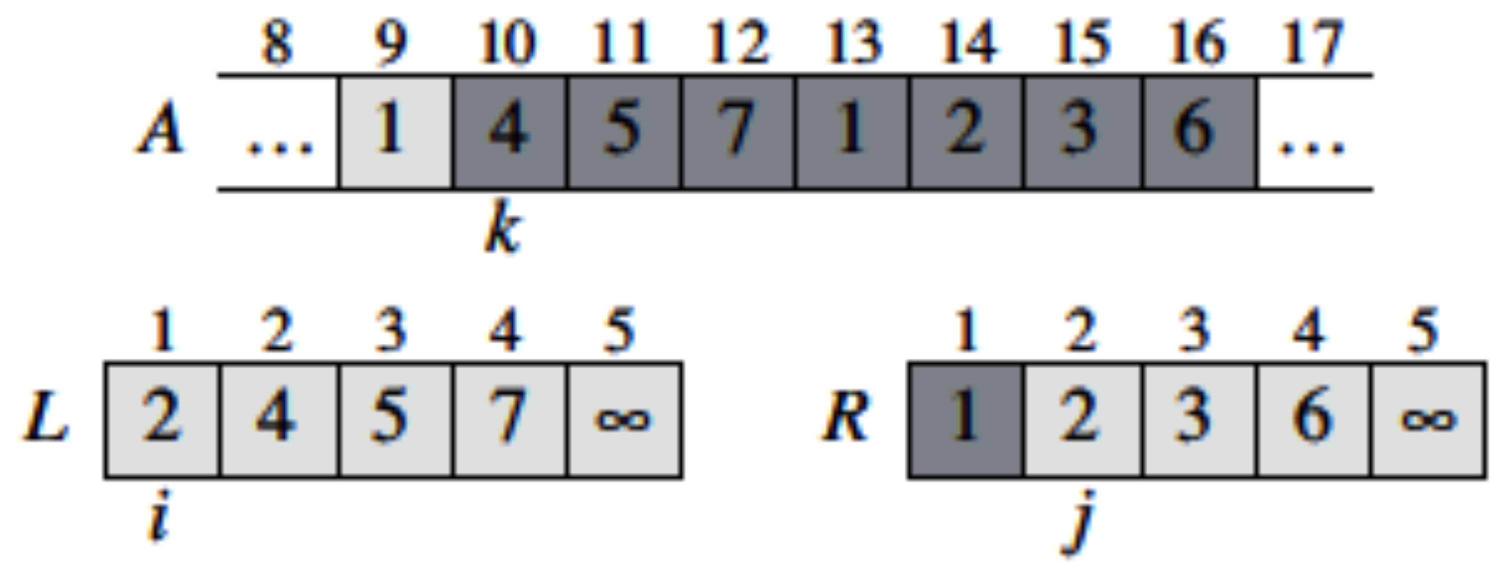
(a)

Merge

IN THIS EXAMPLE, p=9 AND r =16, THEN q=(9+16)/2=12

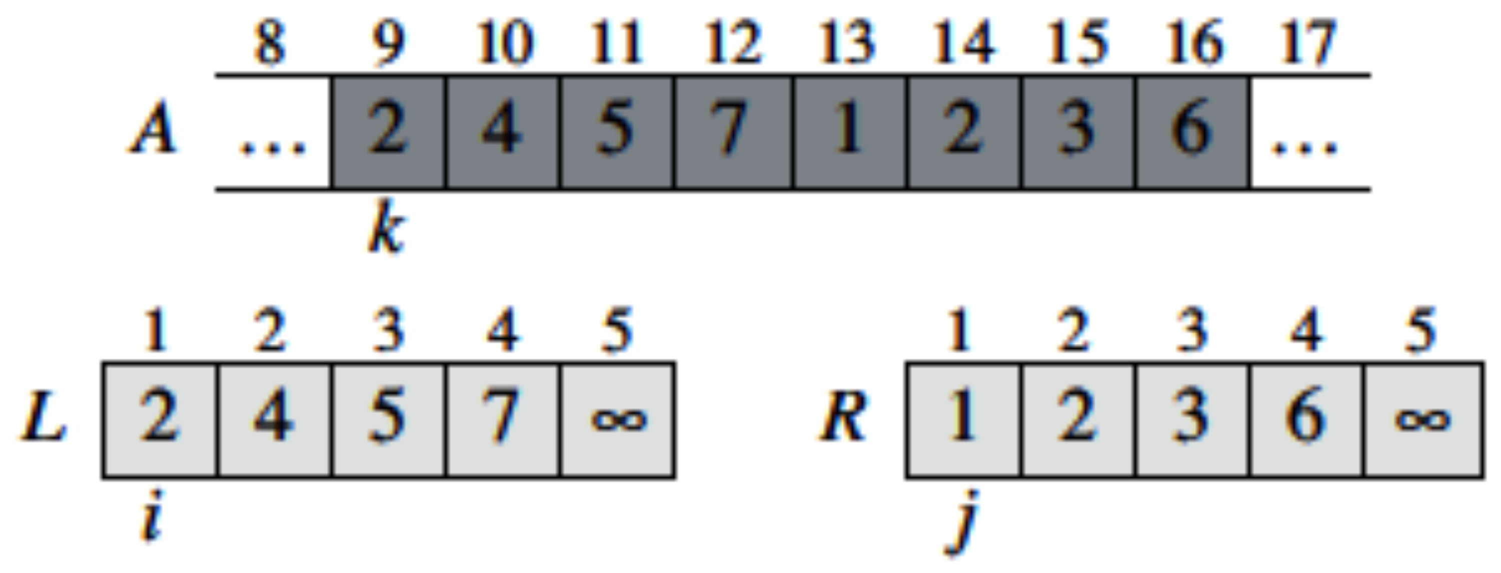


(a)

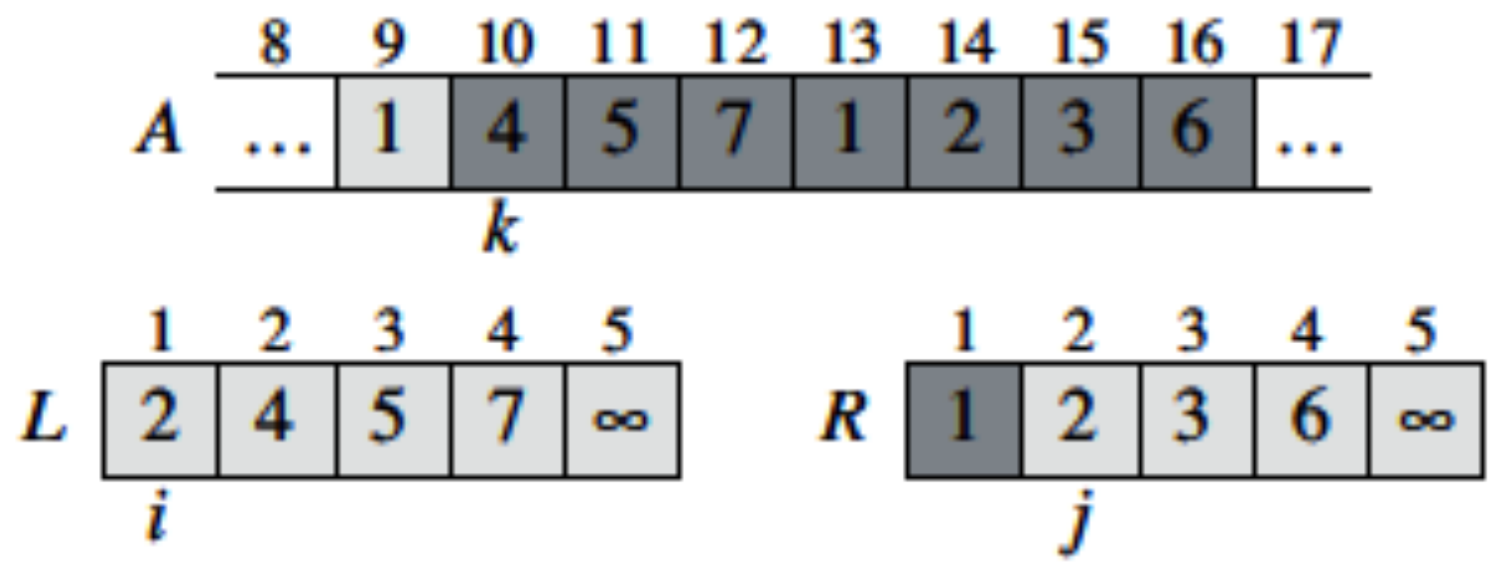


(b)

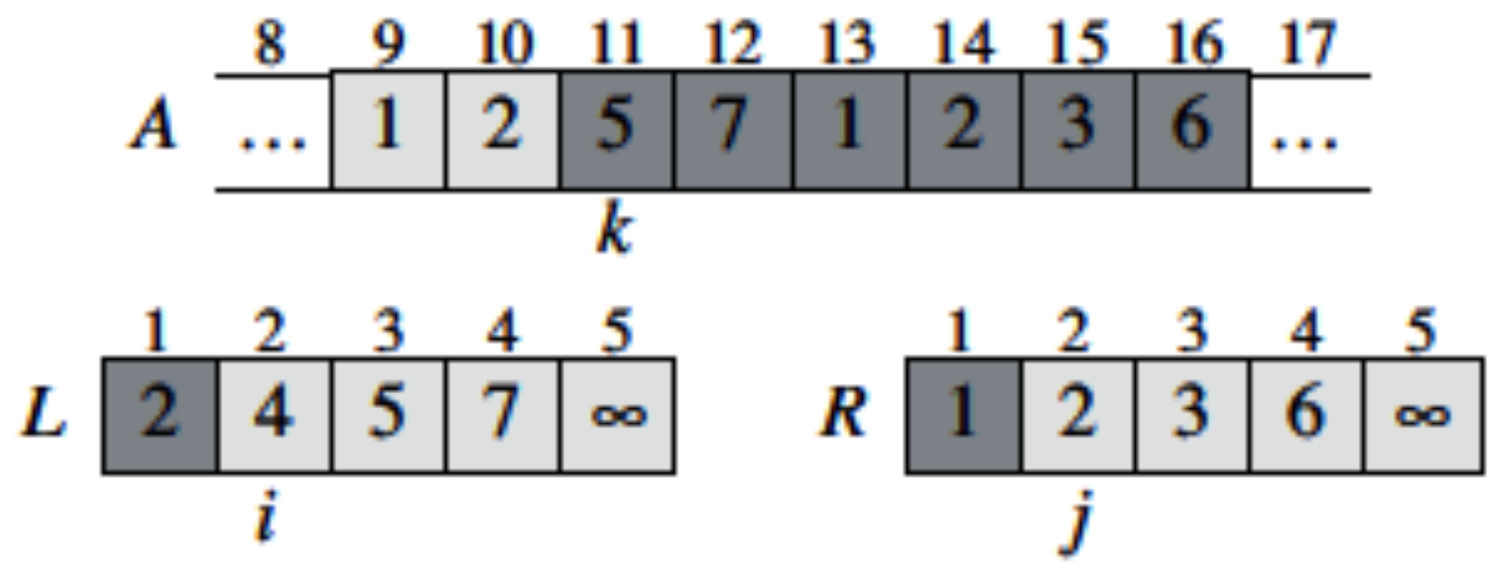
Merge


8 9 10 11 12 13 14 15 16 17
A … 2 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(a)

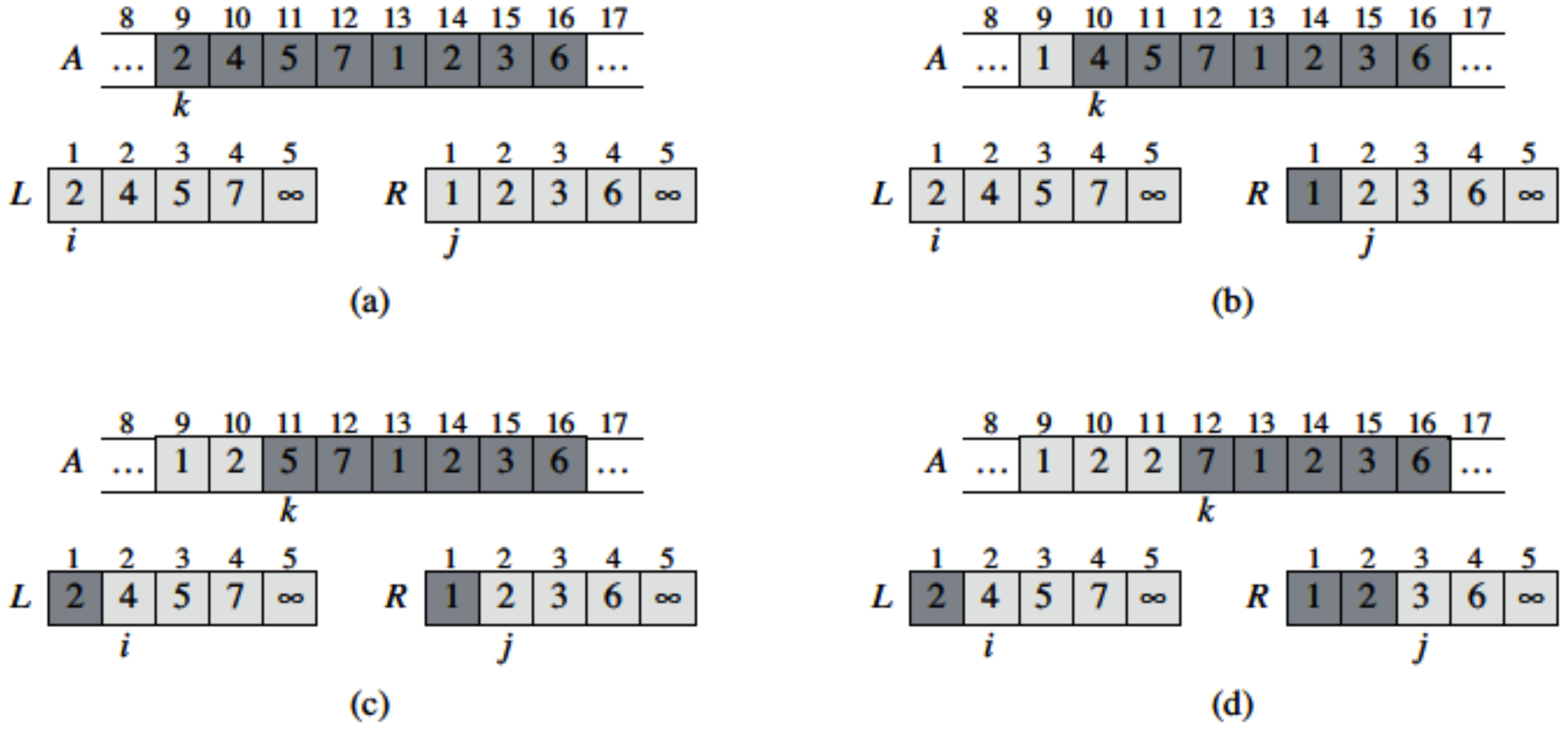

8 9 10 11 12 13 14 15 16 17
A … 1 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(b)

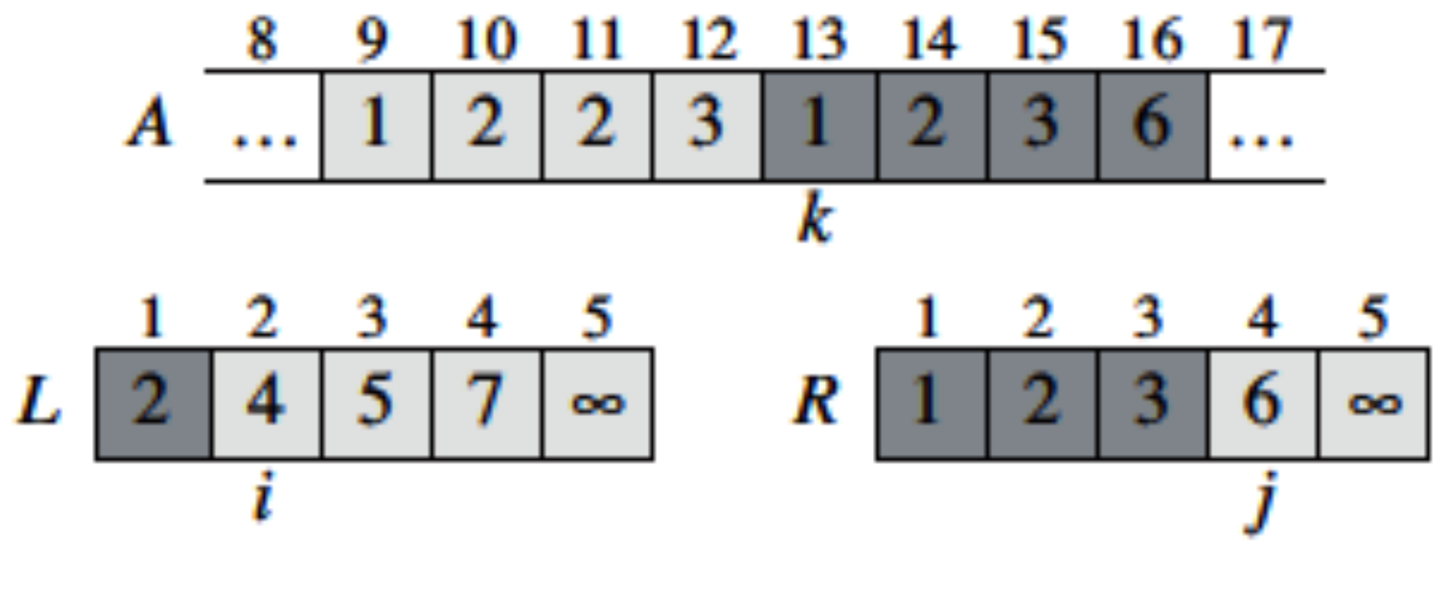

8 9 10 11 12 13 14 15 16 17
A … 1 2 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(c)

Merge

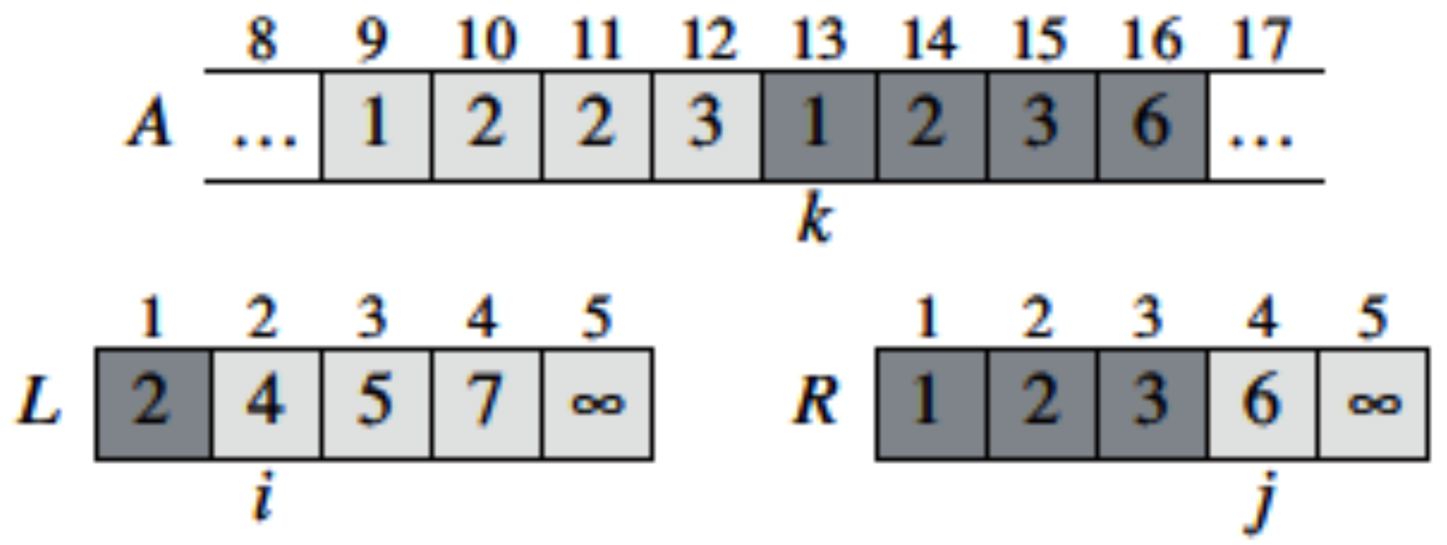


Merge


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

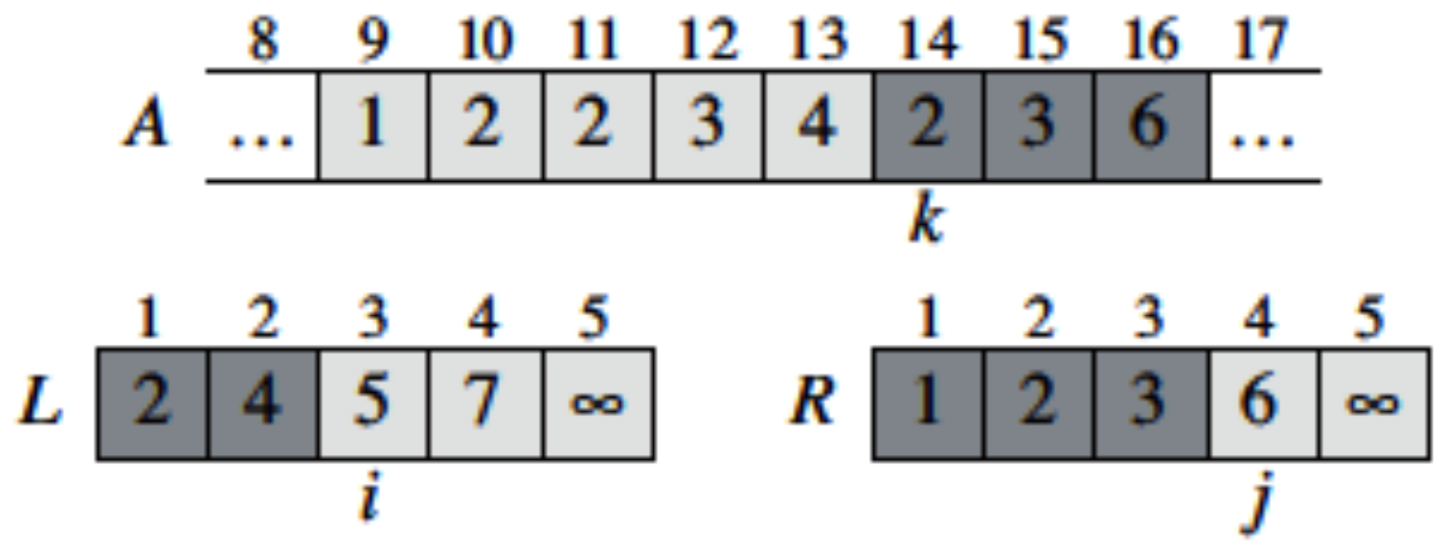
(e)

Merge



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

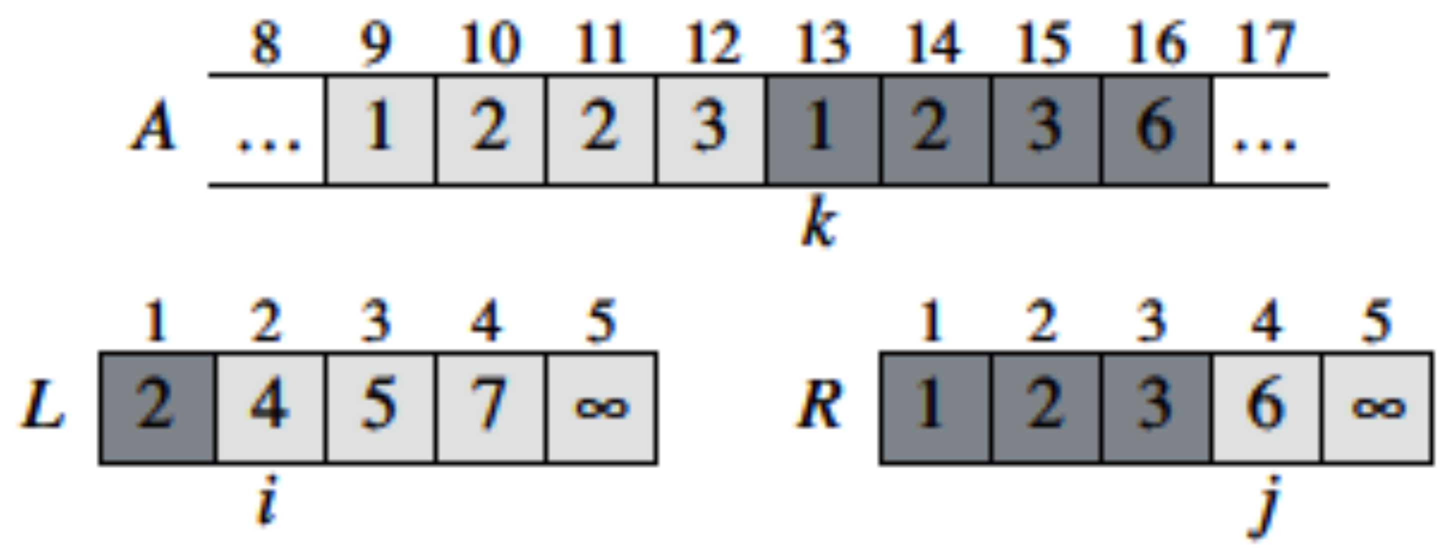
(e)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

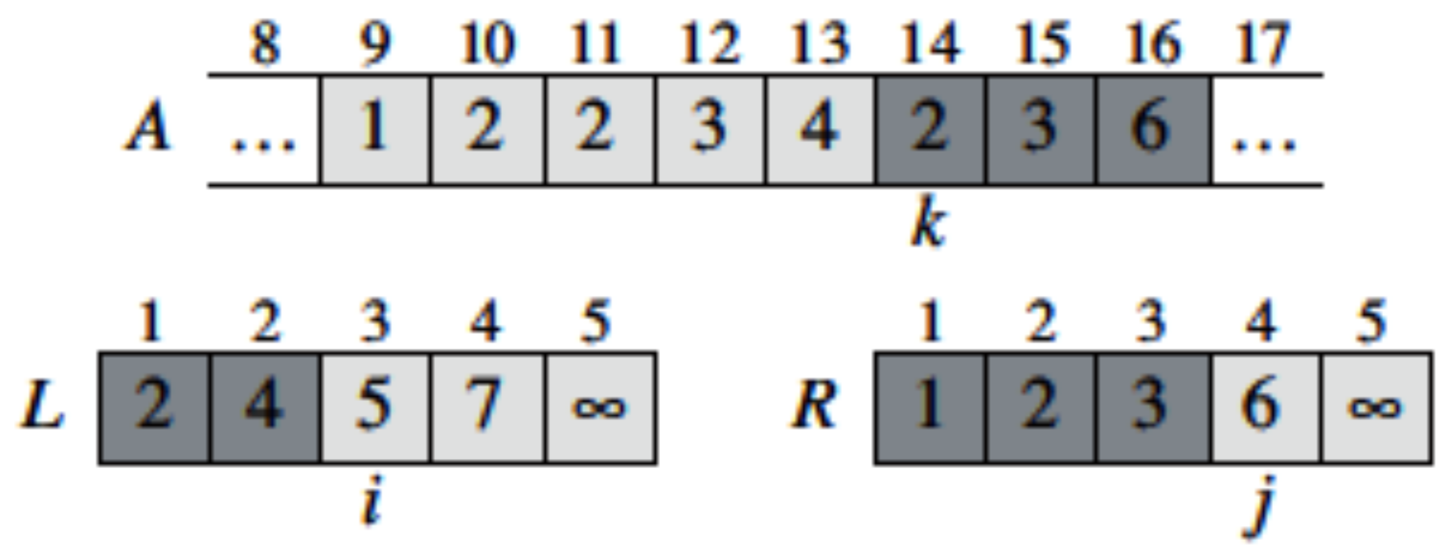
(f)

Merge



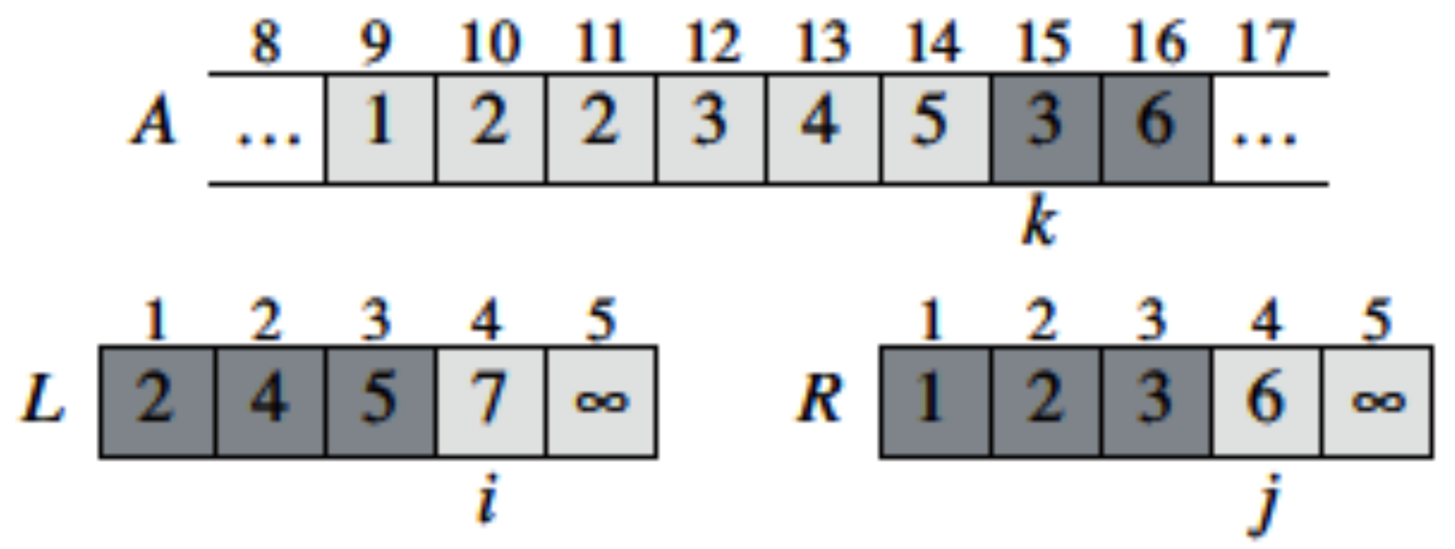
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(e)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

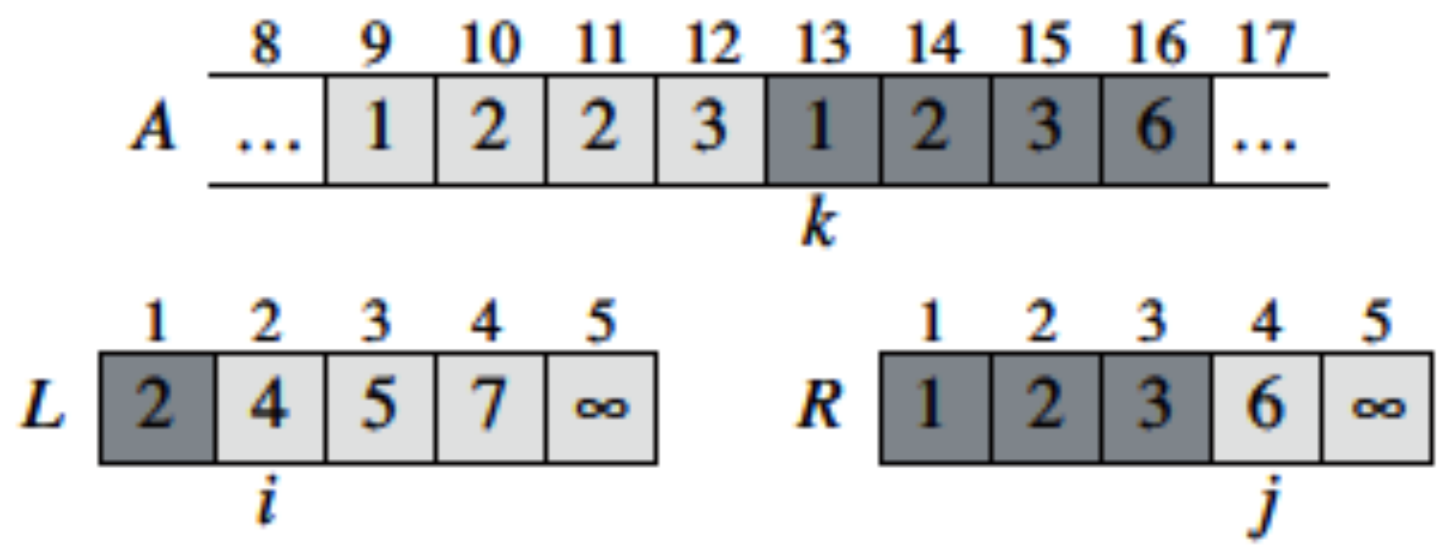
(f)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

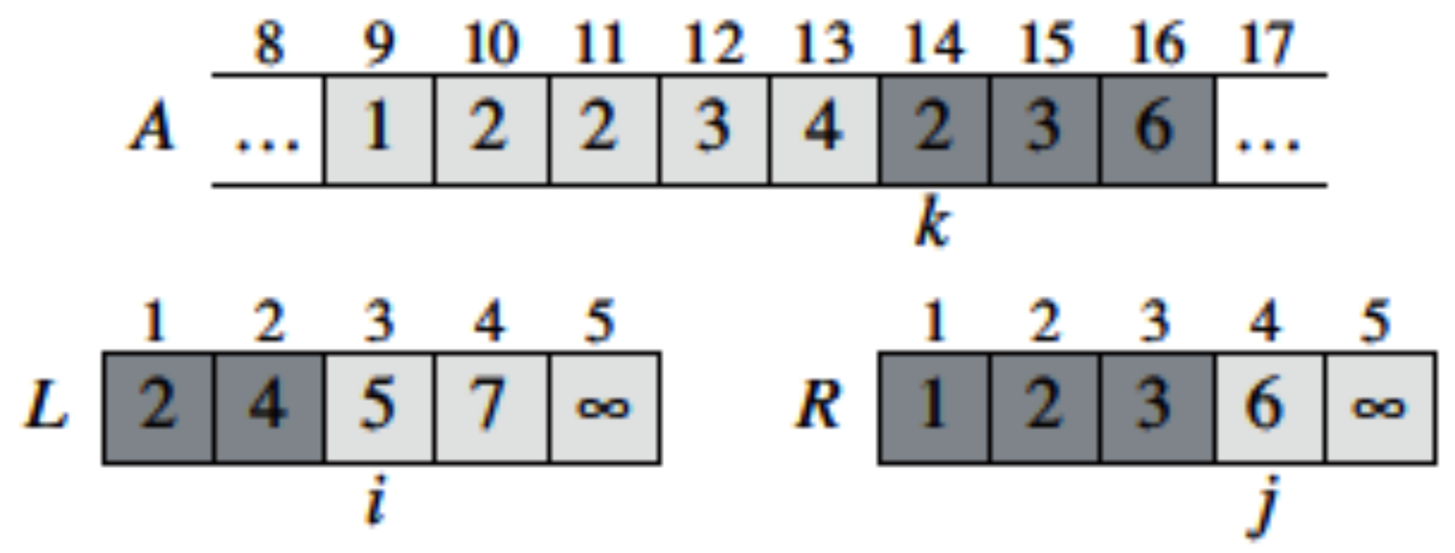
(g)

Merge



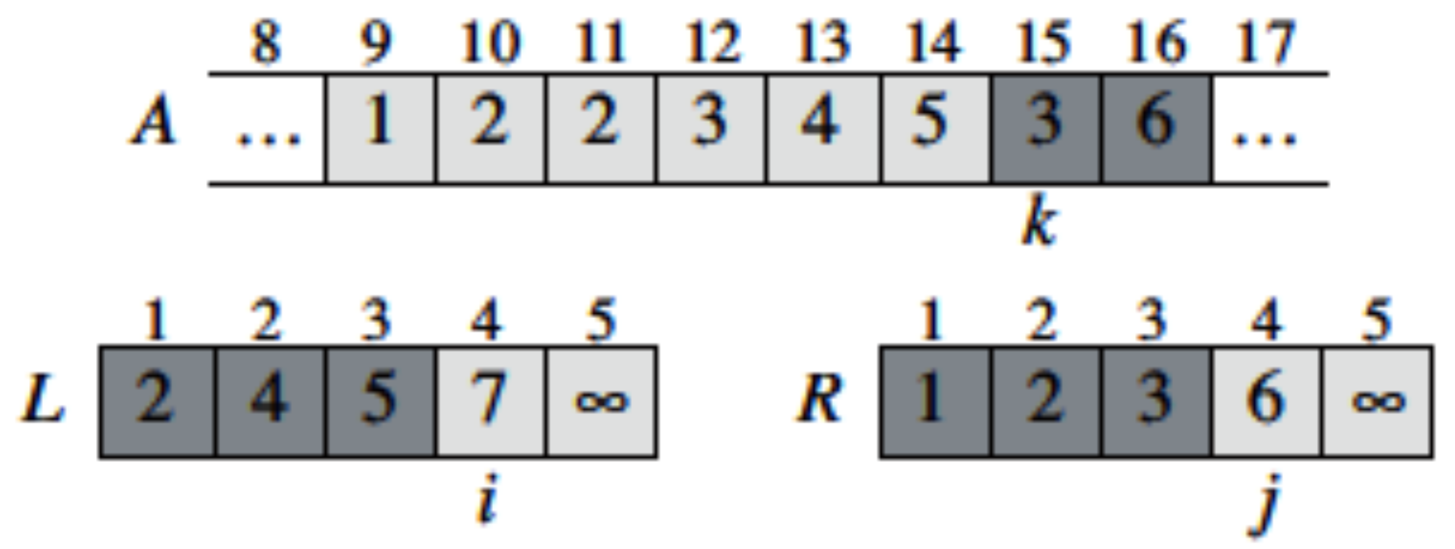
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(e)



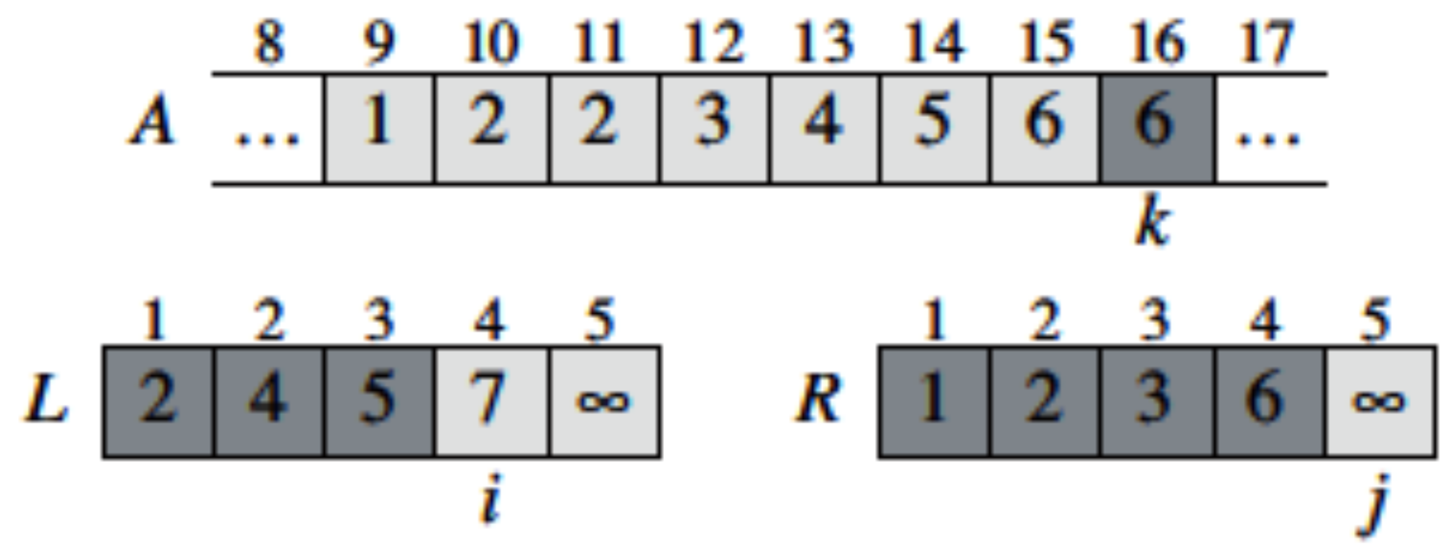
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(f)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

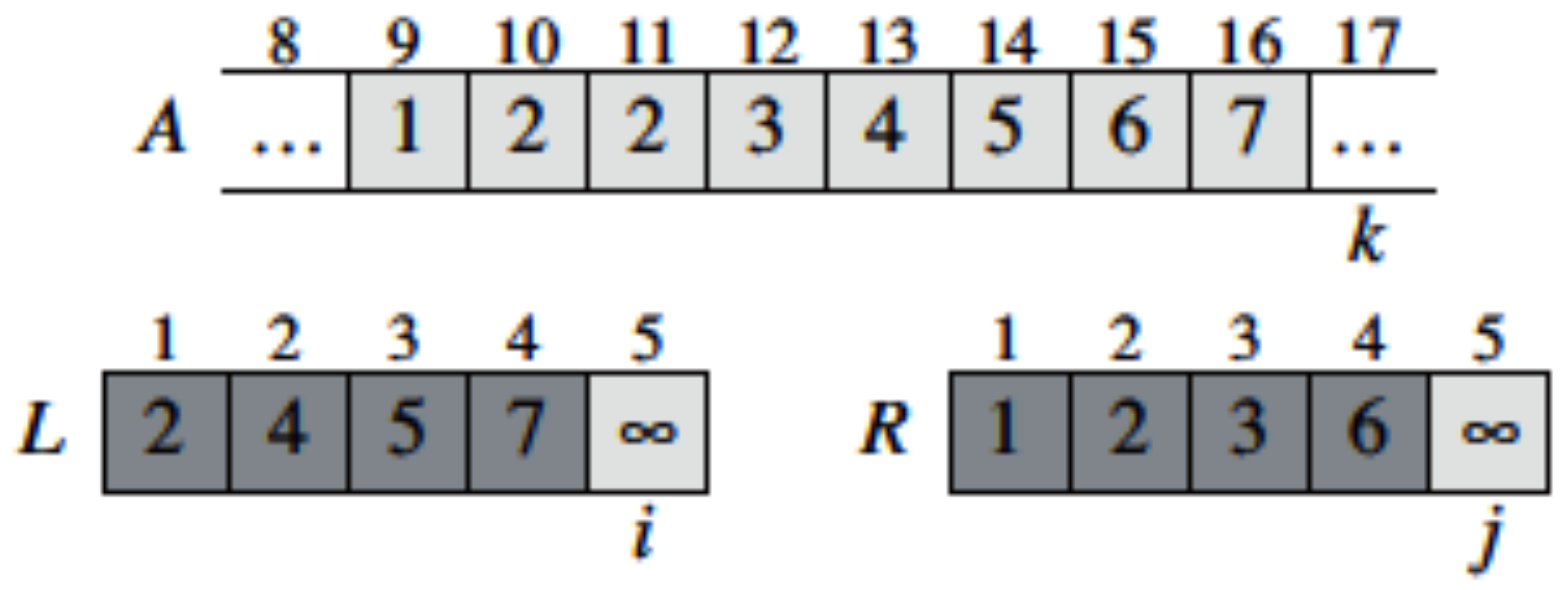
(g)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(h)

Merge


8
9
10
11
12
13
14
15
16
17
A
…
1
2
2
3
4
5
6
7
…
k
1
2
3
4
5
L
2
4
5
7
∞
i
1
2
3
4
5
R
1
2
3
6
∞
j

(i)

Allocation stack

merge-sort(A,1,1)	p==r

merge-sort(A,1,2)	MERGE-SORT(A, 1, 1)
	MERGE-SORT(A, 2, 2)
	MERGE(A, 1, 1, 2)

merge-sort(A,1,4)	MERGE-SORT(A, 1, 2)
	MERGE-SORT(A, 3, 4)
	MERGE(A, 1, 2, 4)

Allocation stack

merge-sort(A,2,2)	p==r

merge-sort(A,1,2)	MERGE-SORT(A, 1, 1)
	MERGE-SORT(A, 2, 2)
	MERGE(A, 1, 1, 2)

merge-sort(A,1,4)	MERGE-SORT(A, 1, 2)
	MERGE-SORT(A, 3, 4)
	MERGE(A, 1, 2, 4)

Allocation stack

merge(A,1,1,2)	output: A[1,2] is sorted

merge-sort(A,1,2)	MERGE-SORT(A, 1, 1)
	MERGE-SORT(A, 2, 2)
	MERGE(A, 1, 1, 2)

merge-sort(A,1,4)	MERGE-SORT(A, 1, 2)
	MERGE-SORT(A, 3, 4)
	MERGE(A, 1, 2, 4)

Allocation stack

merge-sort(A,1,2)

```
MERGE-SORT(A, 1, 1)
MERGE-SORT(A, 2, 2)
MERGE(A, 1, 1, 2)
```

done!
A[1,2] is sorted

merge-sort(A,1,4)

```
MERGE-SORT(A, 1, 2)
MERGE-SORT(A, 3, 4)
MERGE(A, 1, 2, 4)
```

Allocation stack

merge-sort(A,3,3)	p==r

merge-sort(A,3,4)	MERGE-SORT(A, 3, 3)
	MERGE-SORT(A, 4, 4)
	MERGE(A, 3, 3, 4)

merge-sort(A,1,4)	MERGE-SORT(A, 1, 2)
	MERGE-SORT(A, 3, 4)
	MERGE(A, 1, 2, 4)

Allocation stack

merge-sort(A,4,4)	p==r

merge-sort(A,3,4)	MERGE-SORT(A, 3, 3)
	MERGE-SORT(A, 4, 4)
	MERGE(A, 3, 3, 4)

merge-sort(A,1,4)	MERGE-SORT(A, 1, 2)
	MERGE-SORT(A, 3, 4)
	MERGE(A, 1, 2, 4)

Allocation stack

merge(A,3,3,4)	output: A[3,4] is sorted

merge-sort(A,3,4)	MERGE-SORT(A, 3, 3)
	MERGE-SORT(A, 4, 4)
	MERGE(A, 3, 3, 4)

merge-sort(A,1,4)	MERGE-SORT(A, 1, 2)
	MERGE-SORT(A, 3, 4)
	MERGE(A, 1, 2, 4)

Allocation stack

merge-sort(A,3,4)

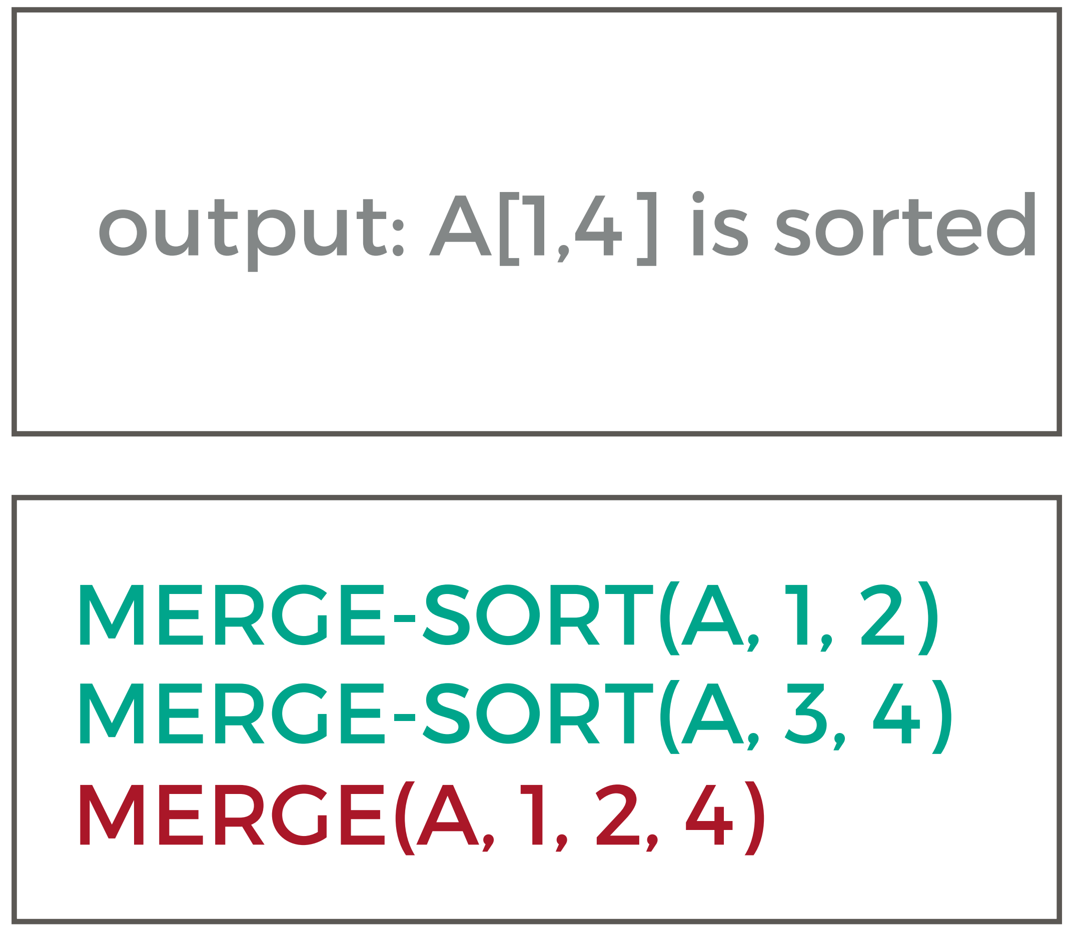
```
MERGE-SORT(A, 3, 3)
MERGE-SORT(A, 4, 4)
MERGE(A, 3, 3, 4)
```

done!
A[3,4] is sorted

merge-sort(A,1,4)

```
MERGE-SORT(A, 1, 2)
MERGE-SORT(A, 3, 4)
MERGE(A, 1, 2, 4)
```

Allocation stack



merge(A,1,2,4)

output: A[1,4] is sorted

merge-sort(A,1,4)

MERGE-SORT(A, 1, 2)
MERGE-SORT(A, 3, 4)
MERGE(A, 1, 2, 4)

Allocation stack

merge-sort(A,1,4)

MERGE-SORT(A, 1, 2)
MERGE-SORT(A, 3, 4)
MERGE(A, 1, 2, 4)

done!

A[1,4] is sorted

Analysis of MERGE

```
MERGE(A, p, q, r)
  $n_1$ = q-p+1 ........................................ $c_1$
  $n_2$ = r-q .......................................... $c_2$
  let L[1..$n_1$] and R[1..$n_2$] be new arrays containing copy of A[p..q] ..... $c_3 \cdot n_1$
  and A[q+1, r] ........................................ $c_4 \cdot n_2$
  i = 1 ................................................ $c_5$
  j = 1 ................................................ $c_6$
  for k = p to r ....................................... $c_7 \cdot n$
     if L[i]≤R[j]
        A[k] = L[i]
        i = i + 1
     else
        A[k] = R[j]
        j = j + 1
```

SUMMING UP $T(n)=\Theta(n)$ where $n=n_1+n_2$

Does merge-sort terminates?

- Size of the input subarray:
  - $n = r-p+1$
  - $n_1=q-p+1$
  - $n_2=r-(q+1)+1=r-q$
- We will show that $n_1<n$ and $n_2<n$
  - it means the size of the subproblems decrease by at least 1 in each recursive call, and so there cannot be more than n-1 levels of recursion.

Does merge-sort terminates?

$p < r$ implies:

$$p + r < 2r \Rightarrow \frac{p+r}{2} < r \Rightarrow \left\lfloor \frac{p+r}{2} \right\rfloor < r$$

$$\Rightarrow q < r \Rightarrow q - p + 1 < r - p + 1 \Rightarrow n_1 < n$$

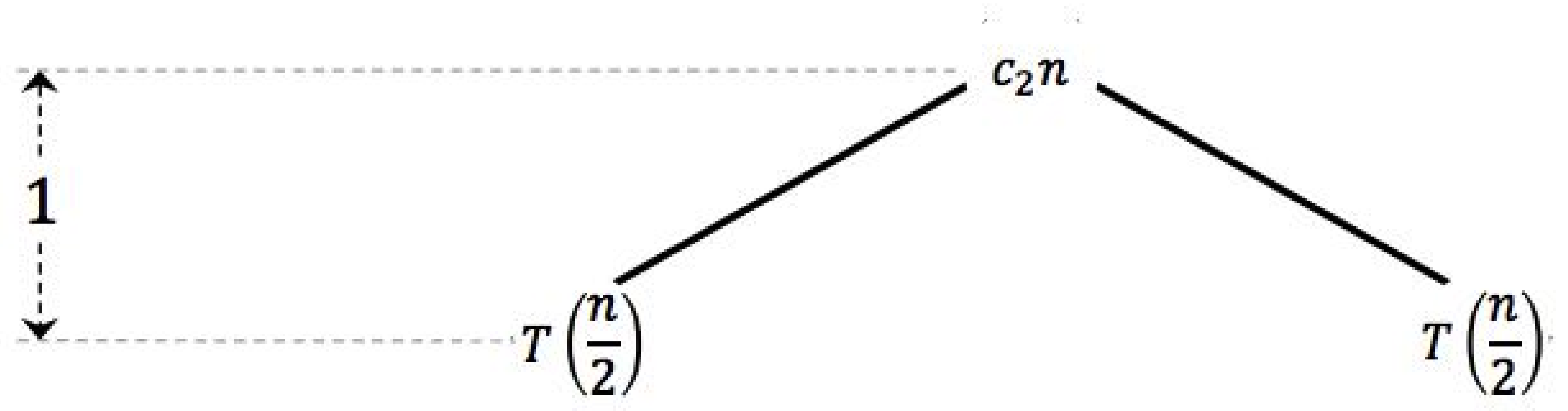
$p < r$ also implies:

$$2p < p + r \Rightarrow p < \frac{p+r}{2} \Rightarrow p \leq \left\lfloor \frac{p+r}{2} \right\rfloor \Rightarrow p \leq q$$

$$\Rightarrow -q \leq -p \Rightarrow r - q \leq r - p \Rightarrow r - q < r - p + 1 \Rightarrow n_2 < n$$

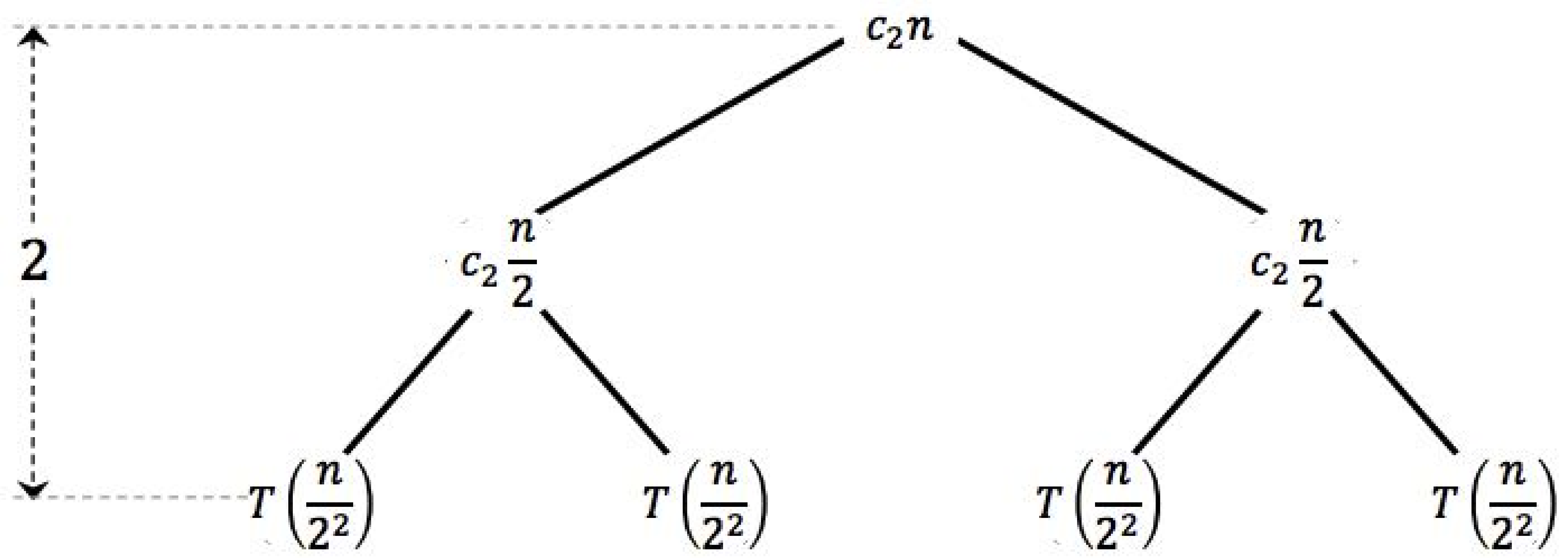
Analysis of MERGE-SORT

UNFOLDING THE RECURRENCE UP TO LEVEL 1



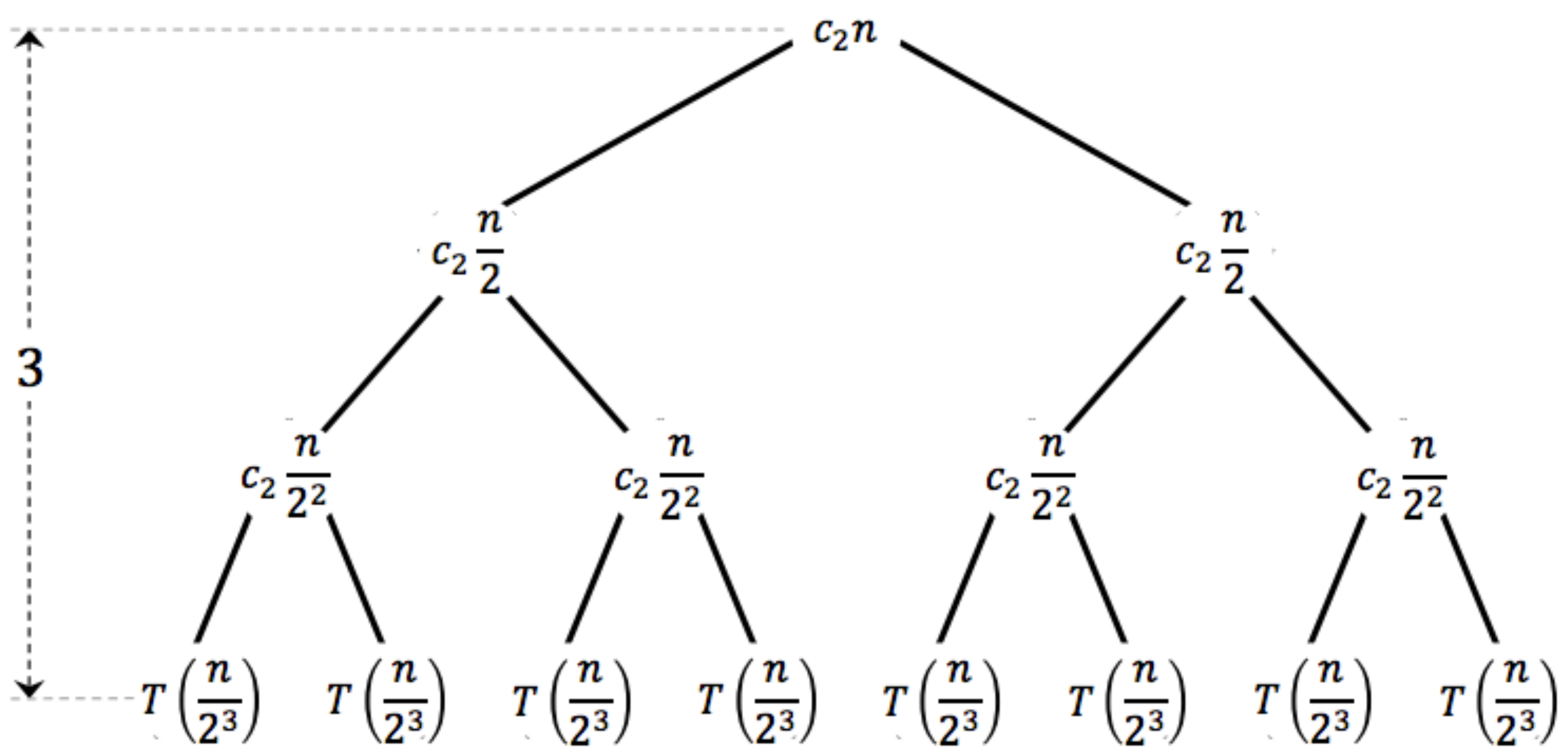
Analysis of MERGE-SORT

UNFOLDING THE RECURRENCE UP TO LEVEL 2


$c_2 n$
$c_2 \frac{n}{2}$
$c_2 \frac{n}{2}$
$T\left(\frac{n}{2^2}\right)$
$T\left(\frac{n}{2^2}\right)$
$T\left(\frac{n}{2^2}\right)$
$T\left(\frac{n}{2^2}\right)$
2

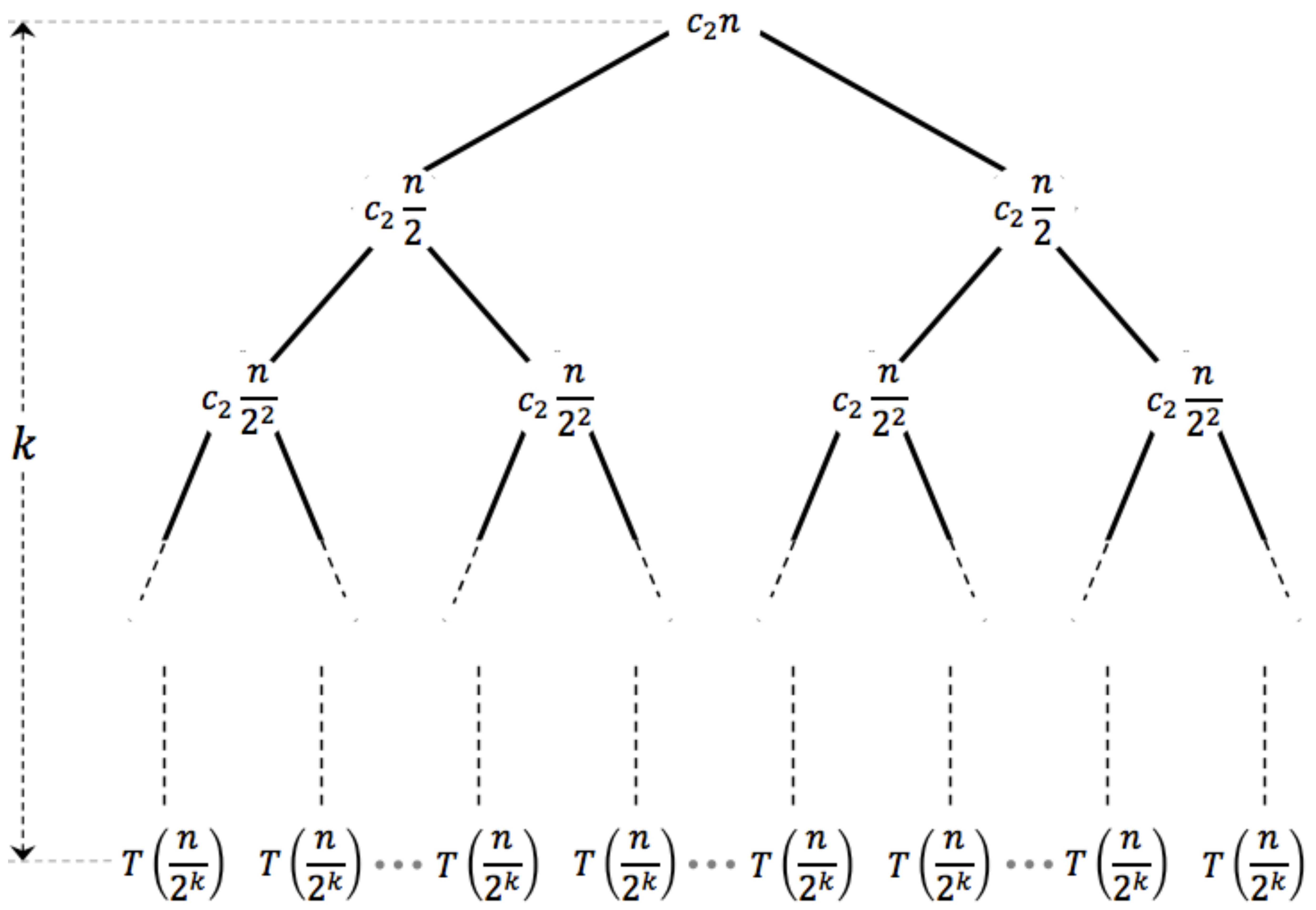
Analysis of MERGE-SORT

UNFOLDING THE RECURRENCE UP TO LEVEL 3


$c_2 n$
$c_2 \frac{n}{2}$
$c_2 \frac{n}{2}$
$c_2 \frac{n}{2^2}$
$c_2 \frac{n}{2^2}$
$c_2 \frac{n}{2^2}$
$c_2 \frac{n}{2^2}$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
$T\left(\frac{n}{2^3}\right)$
3

Analysis of MERGE-SORT

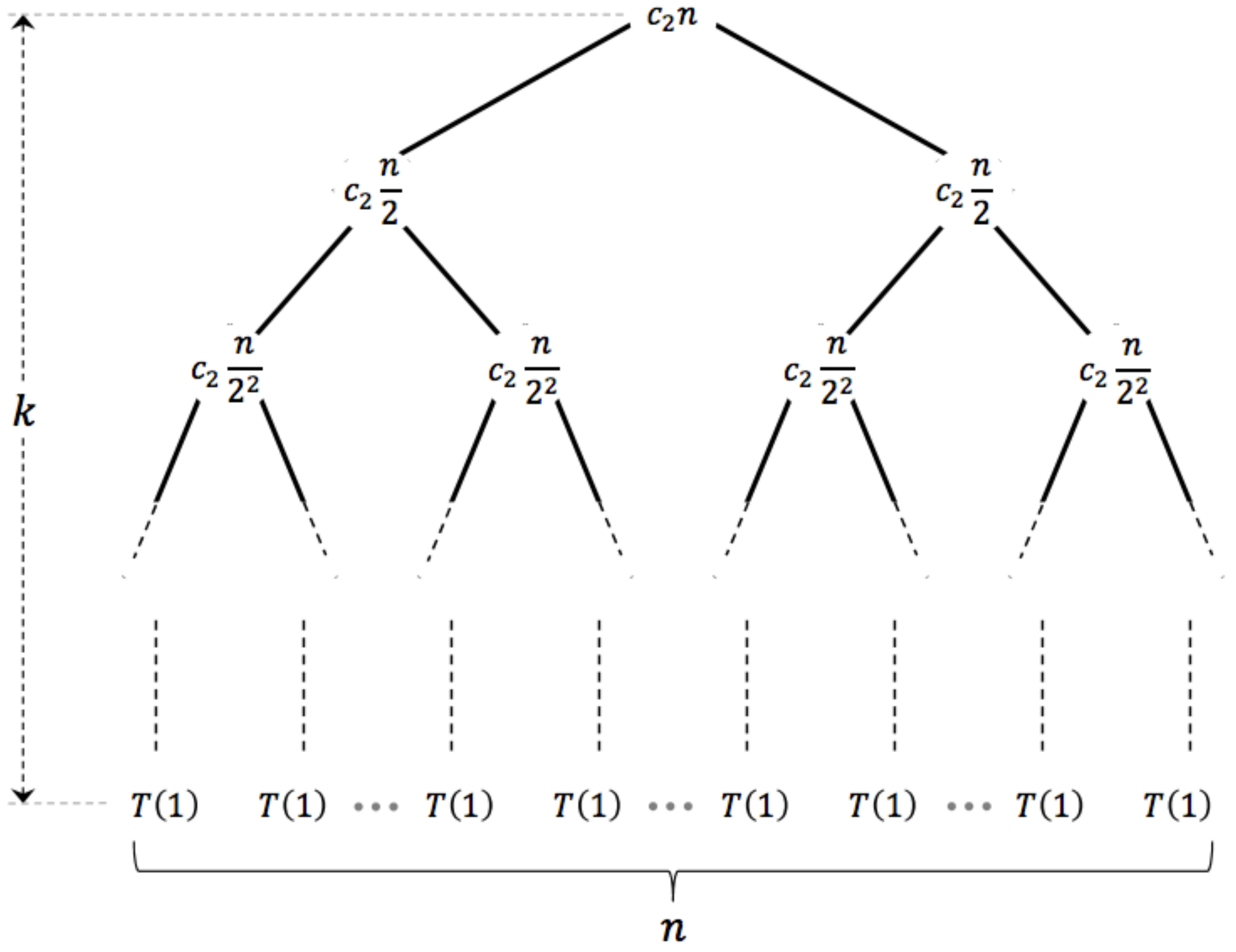
UNFOLDING THE RECURRENCE UP TO LEVEL K



$c_2 n$
$c_2 \frac{n}{2}$
$c_2 \frac{n}{2}$
$c_2 \frac{n}{2^2}$
$c_2 \frac{n}{2^2}$
$c_2 \frac{n}{2^2}$
$c_2 \frac{n}{2^2}$
k
$T\left(\frac{n}{2^k}\right)$ $T\left(\frac{n}{2^k}\right)$ $\cdots$ $T\left(\frac{n}{2^k}\right)$ $T\left(\frac{n}{2^k}\right)$ $\cdots$ $T\left(\frac{n}{2^k}\right)$ $T\left(\frac{n}{2^k}\right)$ $\cdots$ $T\left(\frac{n}{2^k}\right)$ $T\left(\frac{n}{2^k}\right)$

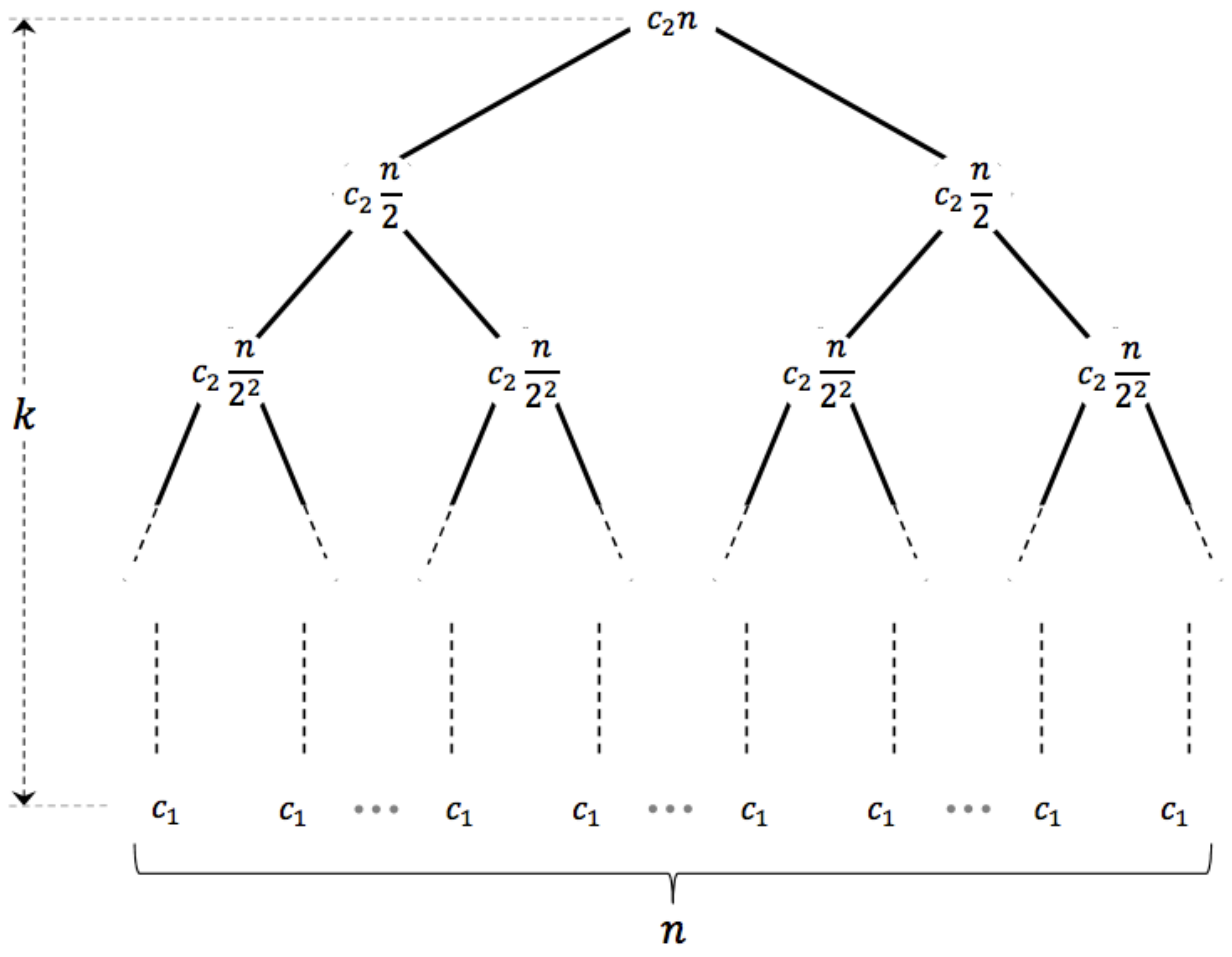
Analysis of MERGE-SORT

$n = 2^K \Rightarrow n/2^k = 1$

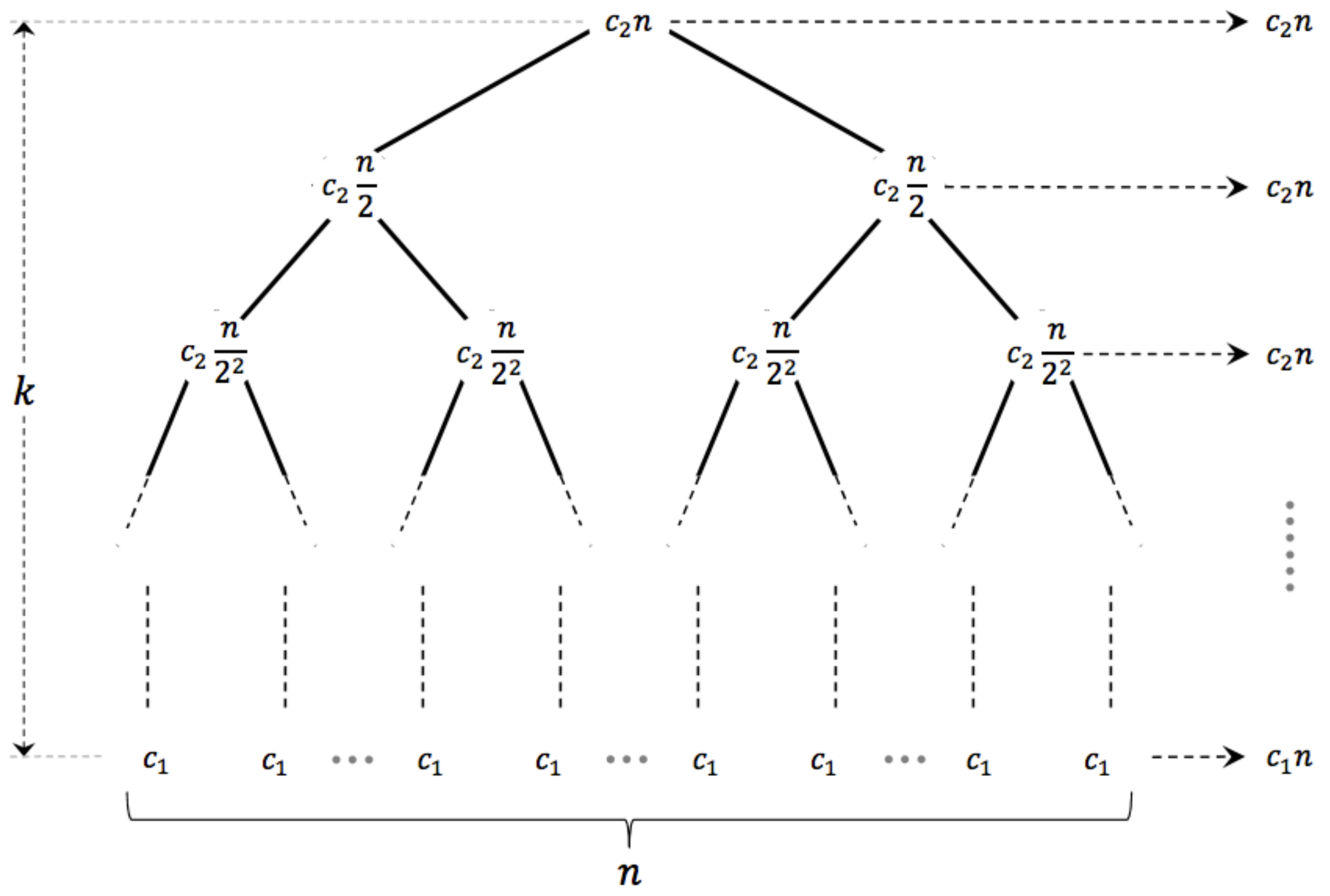


Analysis of MERGE-SORT

$T(n/2^k) = T(1) = c_1$

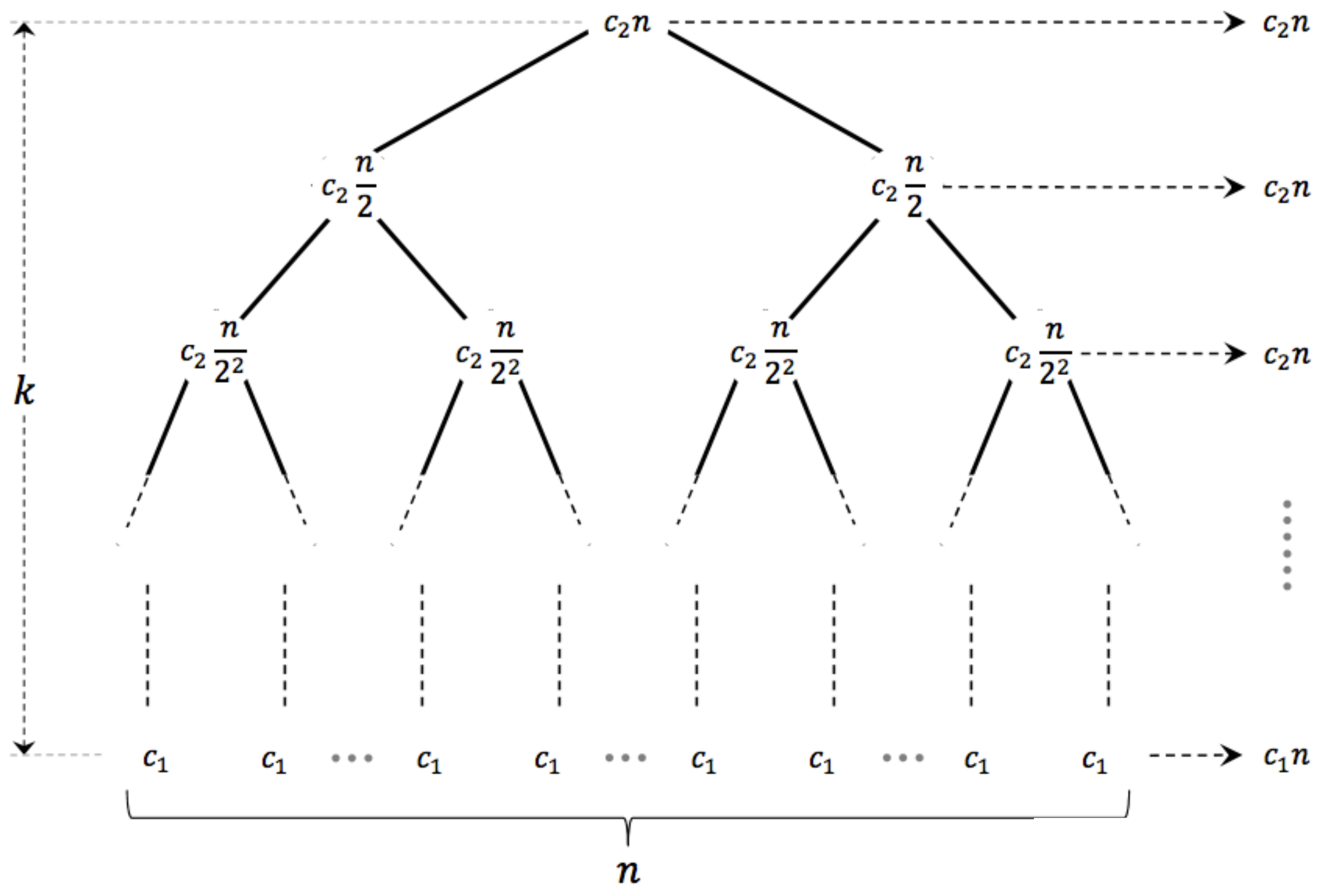


Analysis of MERGE-SORT



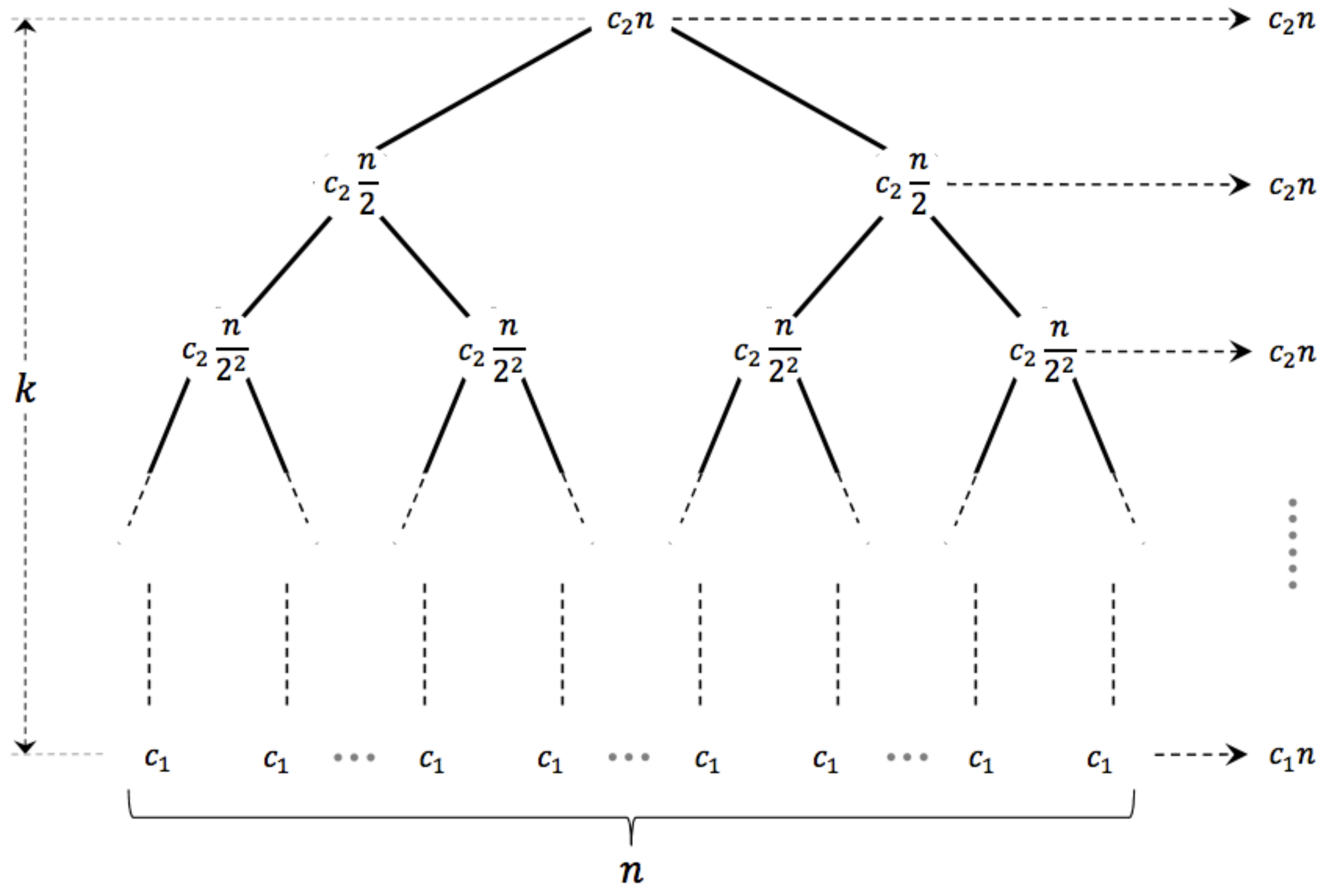
Analysis of MERGE-SORT

$T(n) = c_2 \cdot n \cdot K + c_1 \cdot n$



Analysis of MERGE-SORT

$n = 2^k \Rightarrow k = \log_2 n \Rightarrow c_2 \cdot n \cdot K + c_1 \cdot n = c_2 \cdot n \cdot \log_2 n + c_1 \cdot n = \Theta(n \log_2 n)$



Questions?

@rschifan

schifane@di.unito.it

http://www.di.unito.it/~schifane