Design and Analysis of Algorithms
Part I: Divide and Conquer
Lecture 6: Counting Inversion Problem and Polynomial Multiplication Problem

Ke Xu and Yongxin Tong

（许可 与 童咏昕）

School of CSE, Beihang University

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Review to Divide-and-Conquer Paradigm

- Divide-and-conquer (D&C) is an important algorithm design paradigm.
  - **Divide**
    Dividing a given problem into two or more subproblems (ideally of approximately equal size)
  - **Conquer**
    Solving each subproblem (directly if small enough or recursively)
  - **Combine**
    Combining the solutions of the subproblems into a global solution

Review to Divide-and-Conquer Paradigm

- In Part I, we will illustrate Divide-and-Conquer using several examples:
  - Maximum Contiguous Subarray (最大子数组)
  - Counting Inversions (逆序计数)
  - Polynomial Multiplication (多项式乘法)
  - QuickSort and Partition (快速排序与划分)
  - Randomized Selection (随机化选择)
  - Lower Bound for Sorting (基于比较的排序下界)

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Counting inversions

Music site tries to match your song preferences with others.

- You rank *n* songs.
- Music site consults database to find people with similar tastes.

Counting inversions

Music site tries to match your song preferences with others.

- You rank *n* songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Counting inversions

Music site tries to match your song preferences with others.

- You rank *n* songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Counting inversions

Music site tries to match your song preferences with others.

- You rank *n* songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: a_1, a_2, …, a_n.

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Counting inversions

Music site tries to match your song preferences with others.

- You rank *n* songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j are inverted if $i < j$, but $a_i > a_j$.

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

Formal Definition

- Input: An array of reals $A[1\ldots n]$
- Output: The total number of inversions, namely

$$\sum_{1 \le i < j \le n} X_{i,j}$$

$$X_{i,j} = \begin{cases} 1, & A[i] > A[j] \\ 0, & A[i] \le A[j] \end{cases}$$

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

A Brute Force Algorithm

List each pair i < j and count the inversions.

Input: L
Output: r
$r \leftarrow 0;$

A Brute Force Algorithm

List each pair i < j and count the inversions.

Input: L
Output: r
$r \leftarrow 0;$
for $i \leftarrow 1$ to $L.length$ **do**

A Brute Force Algorithm

List each pair i < j and count the inversions.

Input: L
Output: r
$r \leftarrow 0;$
for $i \leftarrow 1$ *to* $L.length$ **do**
    for $j \leftarrow i + 1$ *to* $L.length$ **do**
        if $L[i] > L[j]$ **then**

A Brute Force Algorithm

List each pair i < j and count the inversions.

Input: L
Output: r
$r \leftarrow 0$;
for $i \leftarrow 1$ to $L.length$ **do**
    for $j \leftarrow i + 1$ to $L.length$ **do**
        if $L[i] > L[j]$ **then**
            $r \leftarrow r + 1$;

A Brute Force Algorithm

List each pair i < j and count the inversions.

Input: L
Output: r
$r \leftarrow 0$;
for $i \leftarrow 1$ *to* $L.length$ **do**
  for $j \leftarrow i + 1$ *to* $L.length$ **do**
    if $L[i] > L[j]$ **then**
      $r \leftarrow r + 1$;
    end
  end
end
return r;

A Brute Force Algorithm

List each pair i < j and count the inversions.

Input: L
Output: r
$r \leftarrow 0$;
for $i \leftarrow 1$ *to* $L.length$ **do**
  for $j \leftarrow i+1$ *to* $L.length$ **do**
    if $L[i] > L[j]$ **then**
      $r \leftarrow r+1$;
    end
  end
end
return r;

$O(n^2)$ comparisons and additions.

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Review to Merge Sort

Mergesort(A, left, right)

if *left* $<$ *right* **then**
  center $\leftarrow \lfloor$(left $+$ right)$/2\rfloor$;
  Mergesort(A, left, center);
  Mergesort(A, center$+$1, right);
  "Merge" the two sorted arrays;
end

- To sort the entire array $A[1 \dots n]$, we make the initial call Mergesort(A, 1, n).
- Key subroutine: "Merge"

Counting inversions: divide-and-conquer

Input

14	**7**	**18**	**3**	**10**	**19**	**11**	**23**	**2**	**25**	**16**	**17**

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.

Input

14	7	18	3	10	19	11	23	2	25	16	17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

11	23	2	25	16	17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

11	23	2	25	16	17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

11	23	2	25	16	17

Count inversions in left half A

14-7,14-3,14-10,7-3,18-3,18-10

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

Count inversions in left half A

14-7,14-3,14-10,7-3,18-3,18-10

11	23	2	25	16	17

Count inversions in right half B

11-2,23-2,23-16,23-17,25-16,25-17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a $\in$ A and b $\in$ B.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

Count inversions in left half A

14-7,14-3,14-10,7-3,18-3,18-10

11	23	2	25	16	17

Count inversions in right half B

11-2,23-2,23-16,23-17,25-16,25-17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a∈A and b∈B.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

Count inversions in left half A

14-7,14-3,14-10,7-3,18-3,18-10

11	23	2	25	16	17

Count inversions in right half B

11-2,23-2,23-16,23-17,25-16,25-17

Count inversions (a,b) with a∈A and b∈B

14-11,14-2,7-2,18-11,18-2,18-16,18-17,3-2,10-2,19-11,19-2,19-16,19-17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a∈A and b∈B.
- Return sum of three counts.

Input

14	7	18	3	10	19	11	23	2	25	16	17

14	7	18	3	10	19

Count inversions in left half A

14-7,14-3,14-10,7-3,18-3,18-10

11	23	2	25	16	17

Count inversions in right half B

11-2,23-2,23-16,23-17,25-16,25-17

Count inversions (a,b) with a∈A and b∈B

14-11,14-2,7-2,18-11,18-2,18-16,18-17,3-2,10-2,19-11,19-2,19-16,19-17

Counting inversions: divide-and-conquer

- Divide: separate list into two halves A and B.
- Conquer: recursively count inversions in each list.
- Combine: count inversions (a, b) with a∈A and b∈B.
- Return sum of three counts.

Input

14	**7**	**18**	**3**	**10**	**19**	**11**	**23**	**2**	**25**	**16**	**17**

14	**7**	**18**	**3**	**10**	**19**

11	**23**	**2**	**25**	**16**	**17**

Count inversions in left half A

14-7,14-3,14-10,7-3,18-3,18-10

Count inversions in right half B

11-2,23-2,23-16,23-17,25-16,25-17

Count inversions (a,b) with a∈A and b∈B

14-11,14-2,7-2,18-11,18-2,18-16,18-17,3-2,10-2,19-11,19-2,19-16,19-17

Output

6+6+13 =25

How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

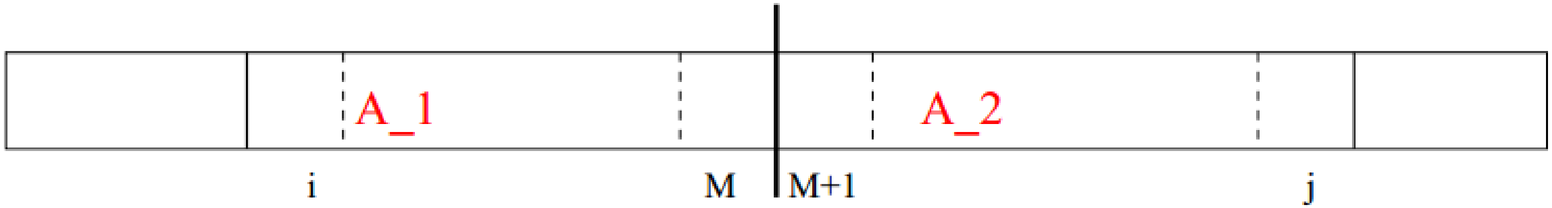
List A

14	7	18	3	10	19

List B

11	23	2	25	16	17

Review to the Conquer Step of MCS Problem



A_1 is in the form $A[i \ldots m]$, $V(i, m) = V(i+1, m) + A[i]$

MAX $\leftarrow A[m]$;
SUM $\leftarrow A[m]$;
for $i \leftarrow m-1$ **downto** *1* **do**
    SUM $\leftarrow$ SUM $+A[i]$;
    if *SUM* $>$ *MAX* **then**
        MAX $\leftarrow$ SUM;
    end
end
A_1=MAX;

How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

List A

14	**7**	**18**	**3**	**10**	**19**

List B

11	**23**	**2**	**25**	**16**	**17**

How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.

List A

14	**7**	**18**	**3**	**10**	**19**

List B

11	**23**	**2**	**25**	**16**	**17**

How to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.

Sort A

3	**7**	**10**	**14**	**18**	**19**

List B

11	**23**	**2**	**25**	**16**	**17**

How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

How to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element $b \in B$,
  - binary search in A to find how many elements in A are greater than b.

Sort A

3	**7**	**10**	**14**	**18**	**19**

Sort B

2	**11**	**16**	**17**	**23**	**25**

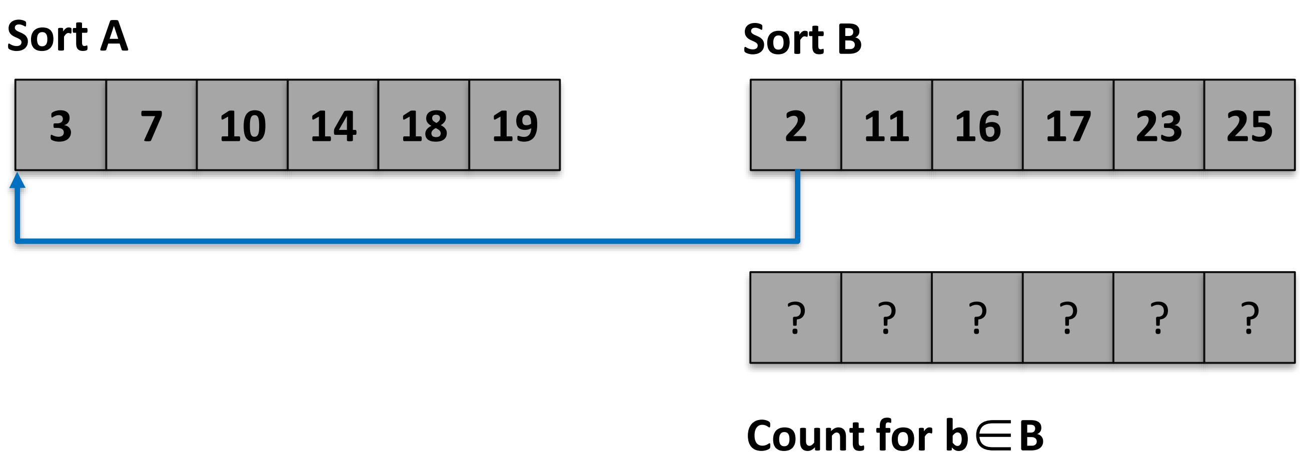
How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.



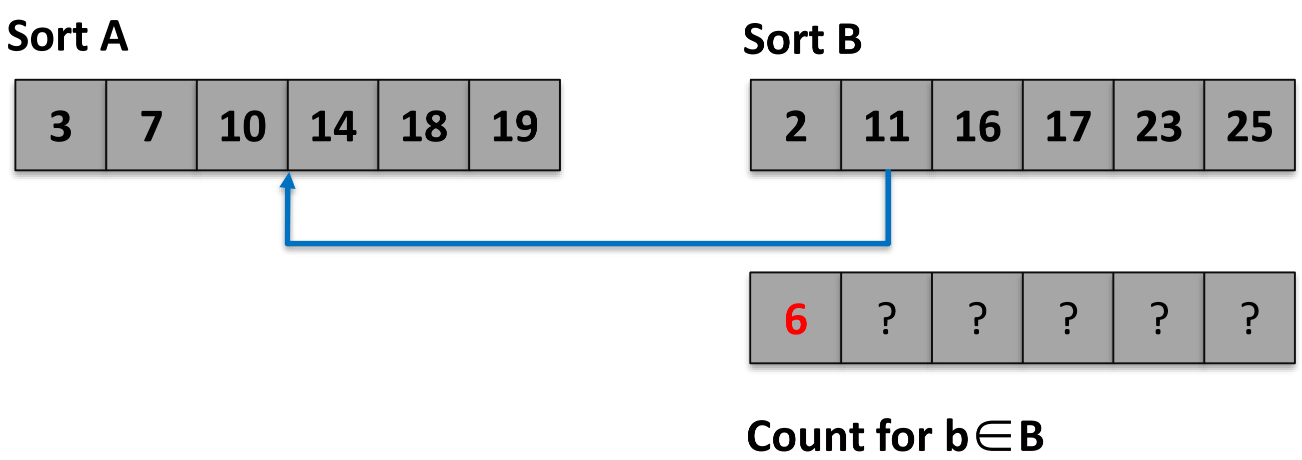
How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.



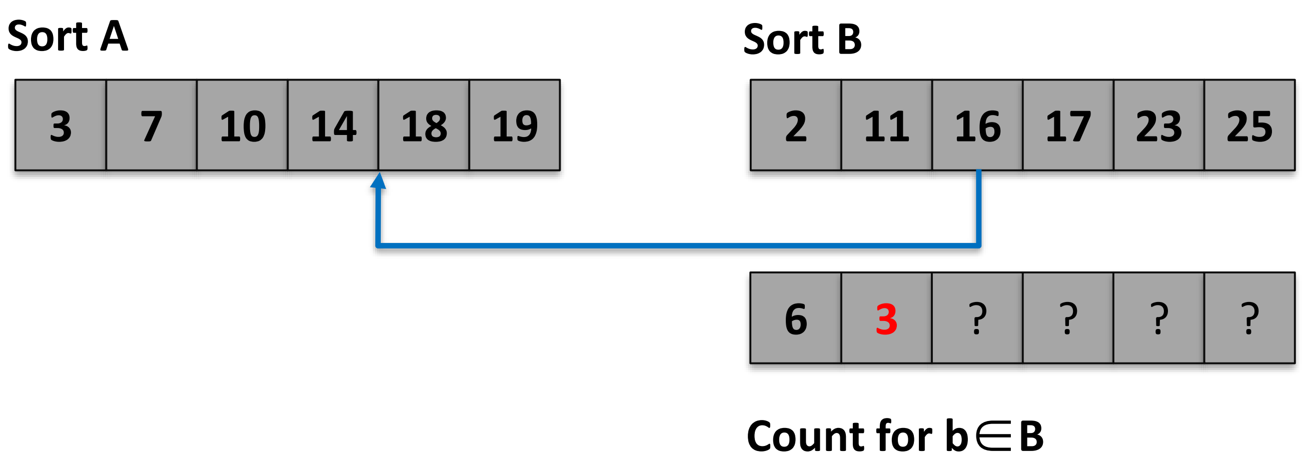
How to combine two subproblems?

Q. How to count inversions (a, b) with $a \in A$ and $b \in B$?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element $b \in B$,
  - binary search in A to find how many elements in A are greater than b.



How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

6	3	2	?	?	?

Count for b∈B

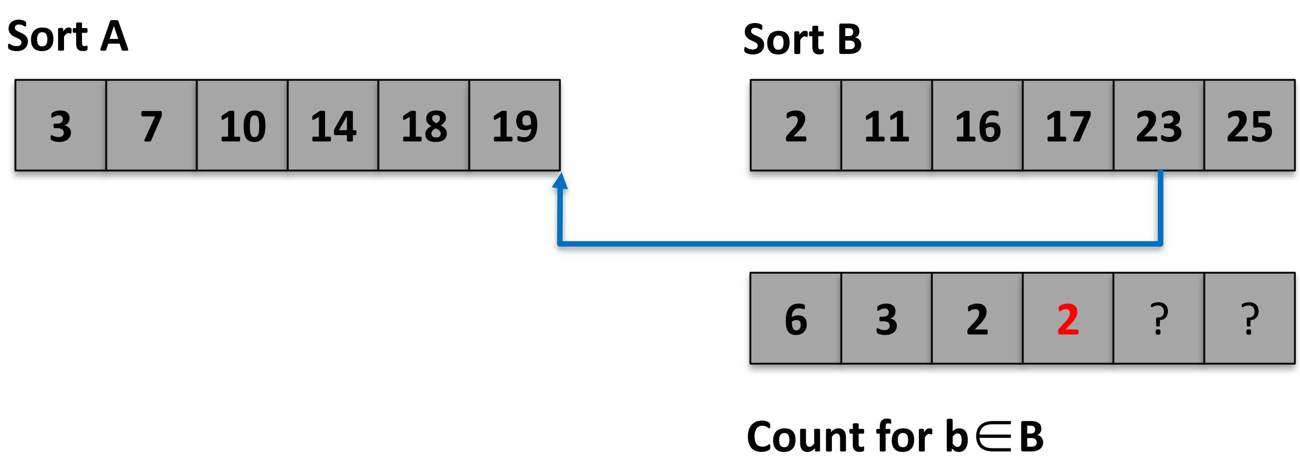
How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.



How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

6	3	2	2	0	?

Count for b∈B

How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

6	3	2	2	0	0

Count for b∈B

How to combine two subproblems?

Q. How to count inversions (a, b) with a ∈ A and b ∈ B?

A. Easy if A and B are sorted!

Warmup algorithm.

- Sort A and B.
- For each element b ∈ B,
  - binary search in A to find how many elements in A are greater than b.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

6	3	2	2	0	0

Count for b∈B

Inversions between A and B: 6+3+2+2=13

Combine two subproblems: Improvement

Count inversions (a, b) with a $\in$ A and b $\in$ B, assuming A and B are sorted.

Combine two subproblems: Improvement

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.

Combine two subproblems: Improvement

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.

Combine two subproblems: Improvement

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

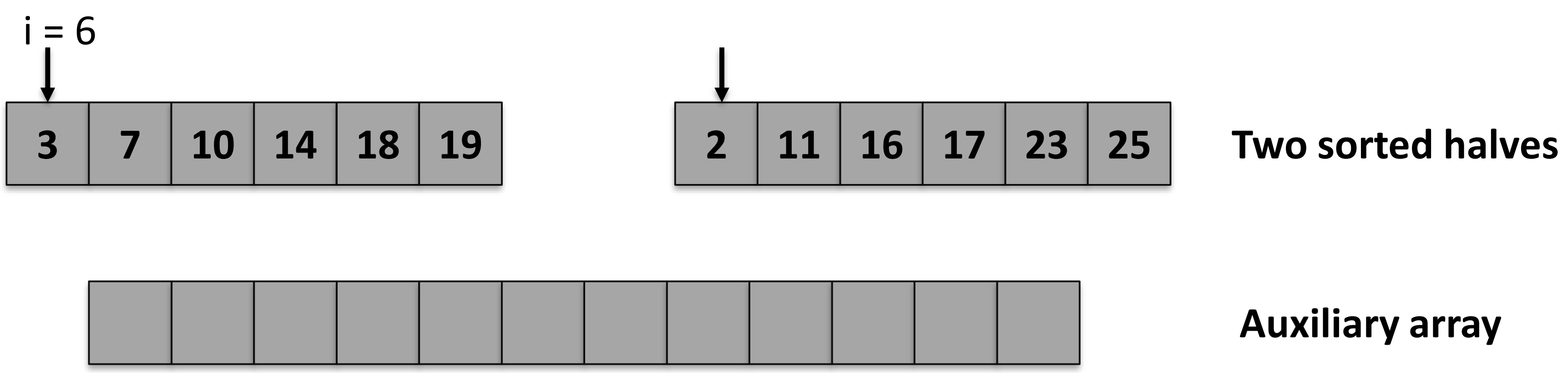
3	7	10	14	18	19	2	11	16	17	23	25		**Two sorted halves**

												Auxiliary array

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

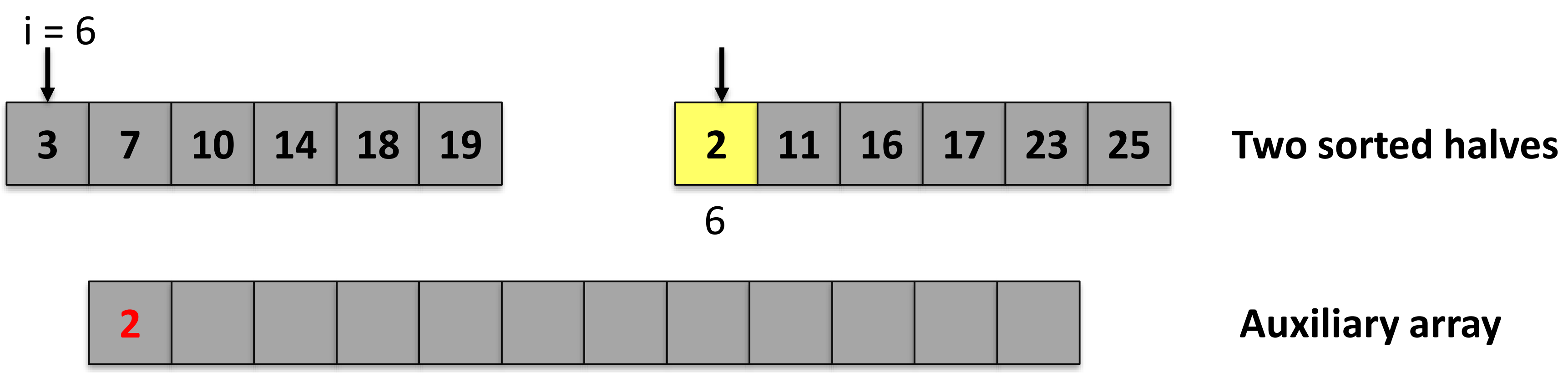


Total:

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

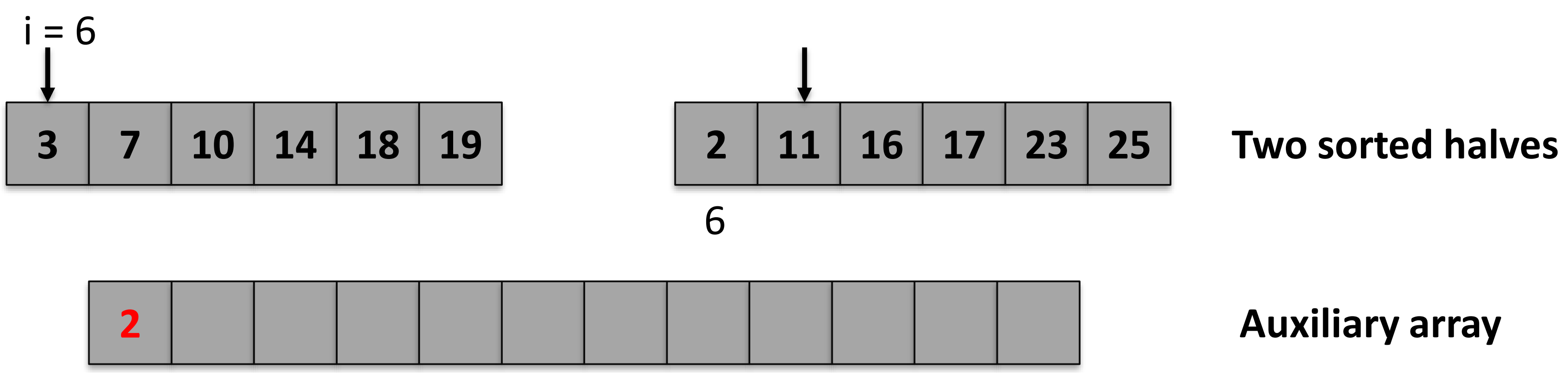


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

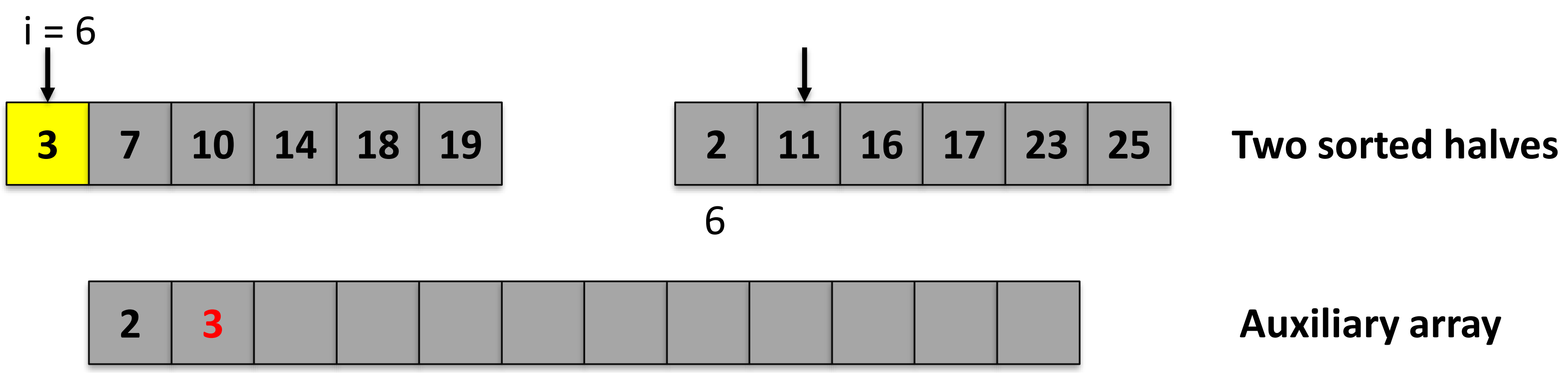


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

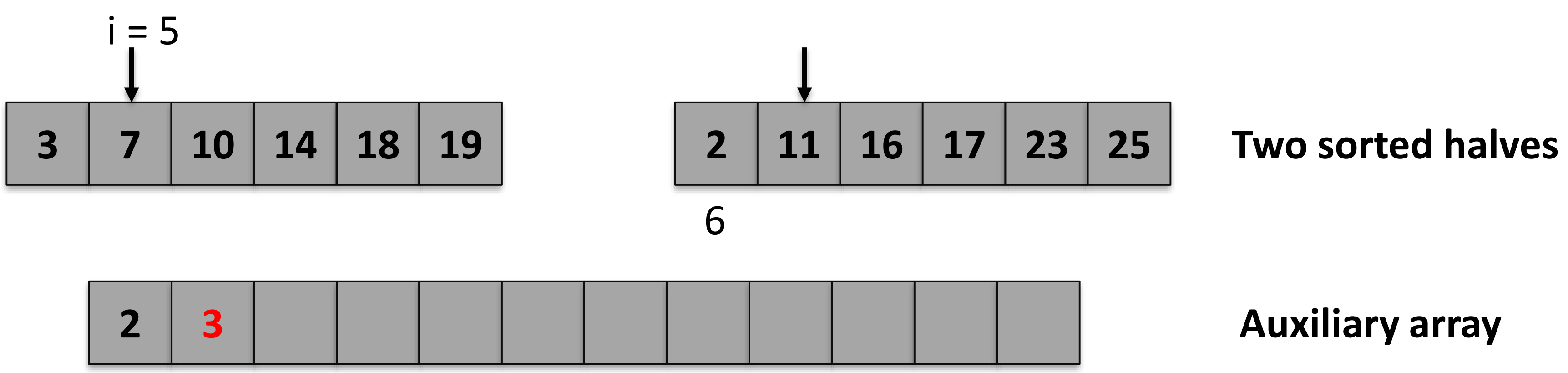


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

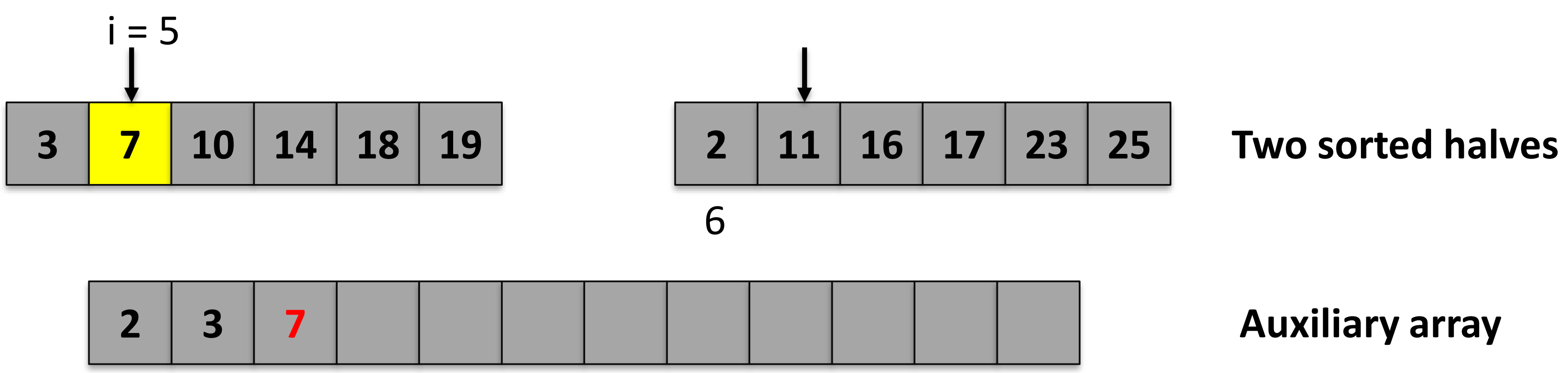


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

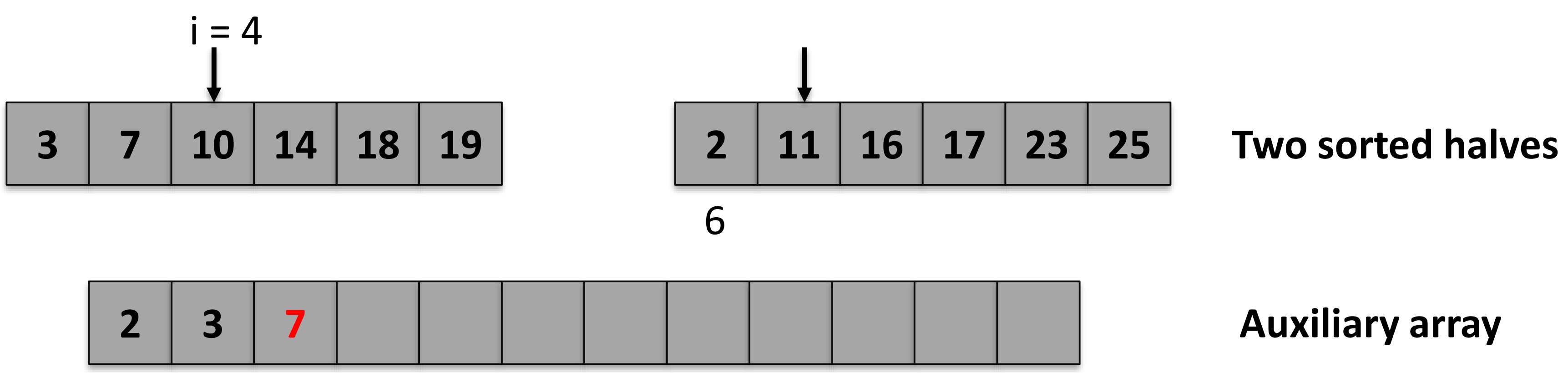


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

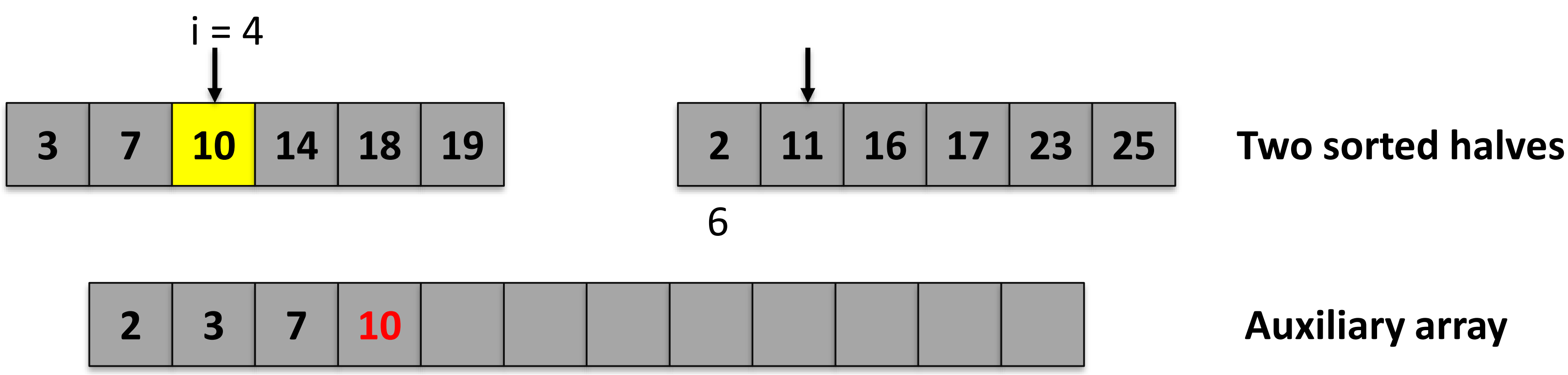


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

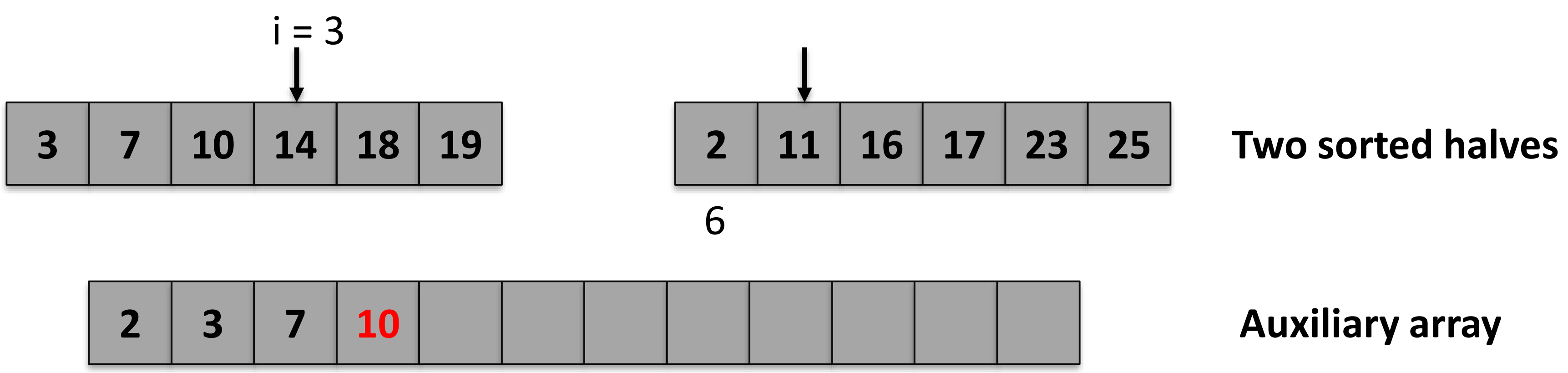


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

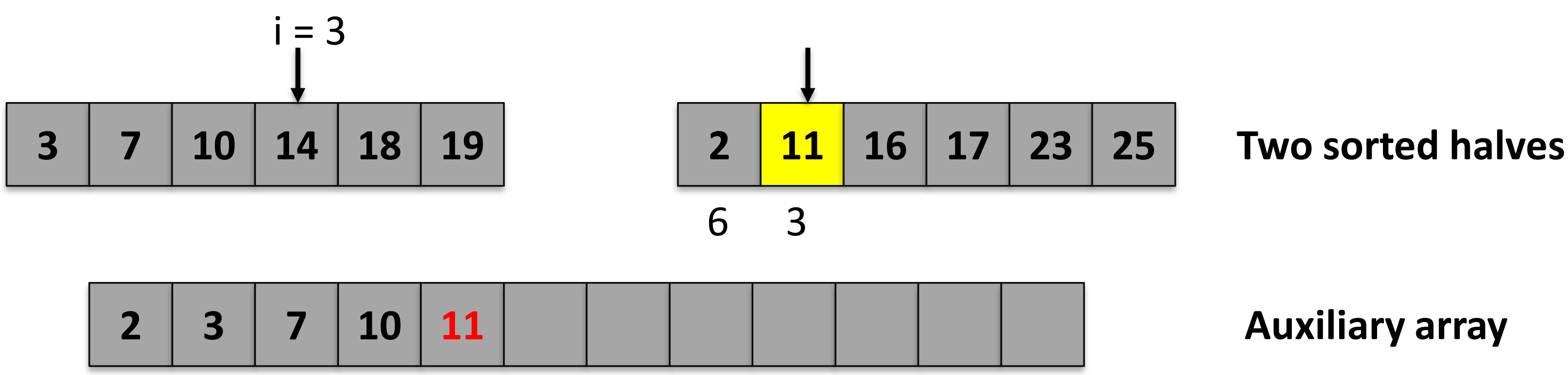


Total: 6

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

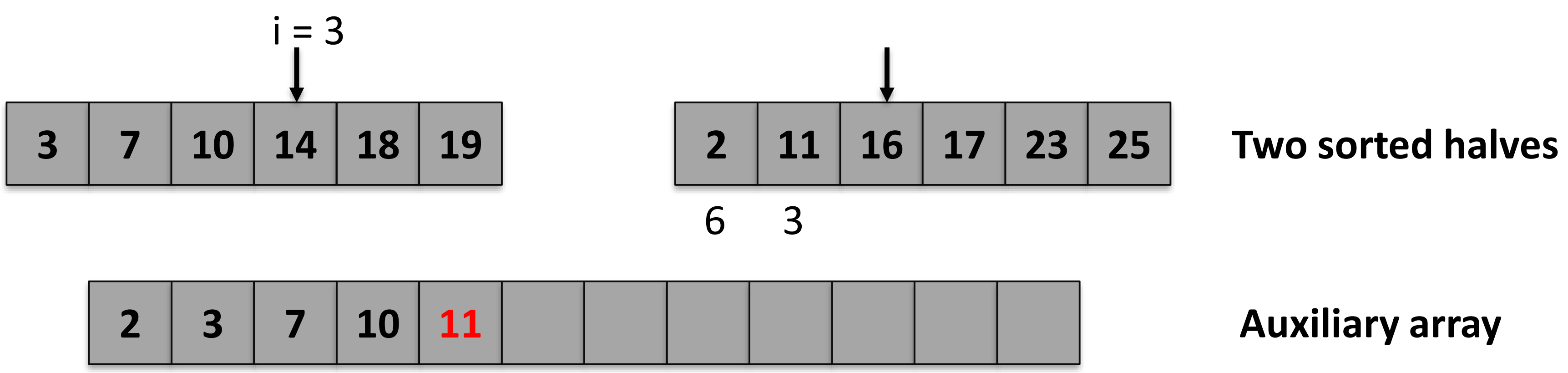


Total: 6+3

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

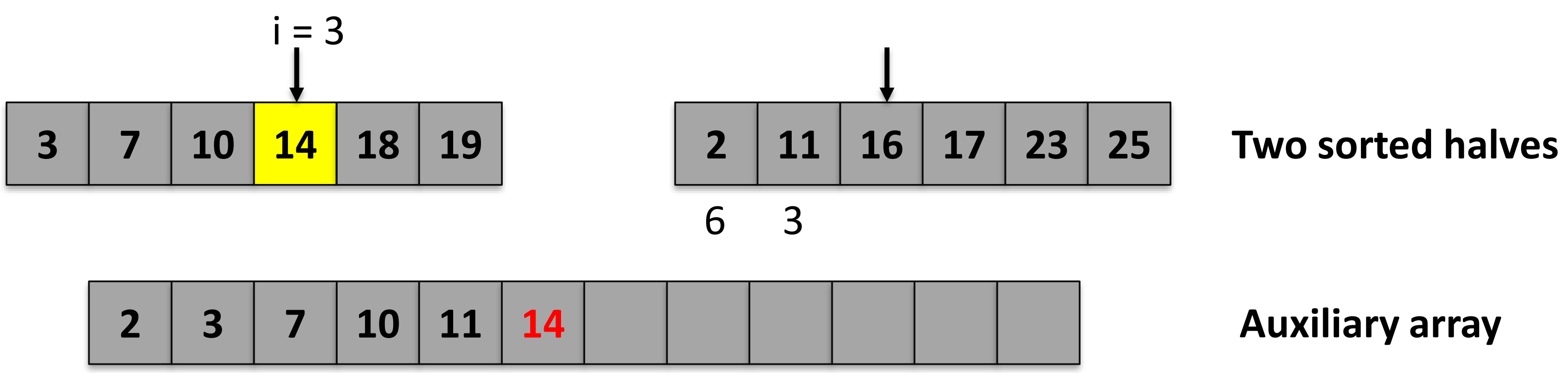


Total: 6+3

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



Total: 6+3

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

i = 2

3	**7**	**10**	**14**	**18**	**19**

2	**11**	**16**	**17**	**23**	**25**
6	3				

Two sorted halves

2	**3**	**7**	**10**	**11**	**14**						

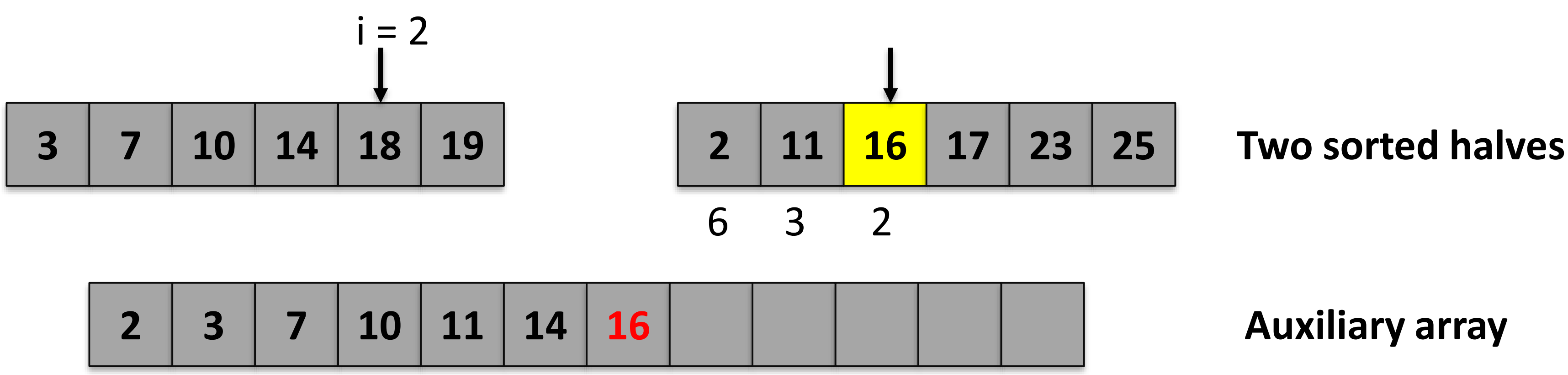
Auxiliary array

Total: 6+3

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

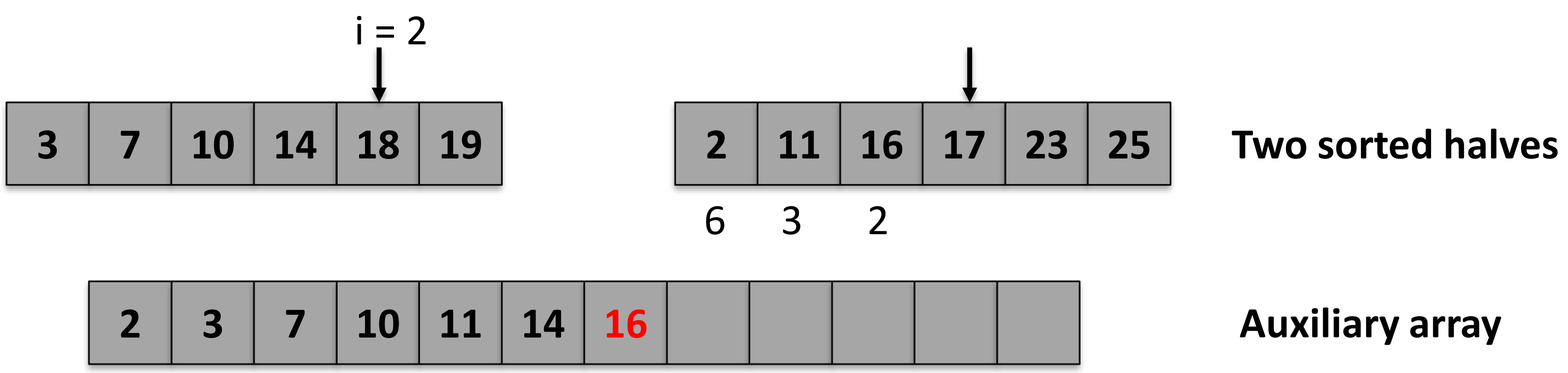


Total: 6+3+2

Combine two subproblems: Improvement

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



Total: 6+3+2

Combine two subproblems: Improvement

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

i = 2

3	7	10	14	18	19		2	11	16	**17**	23	25
							6	3	2	2		

Two sorted halves

2	3	7	10	11	14	16	**17**				

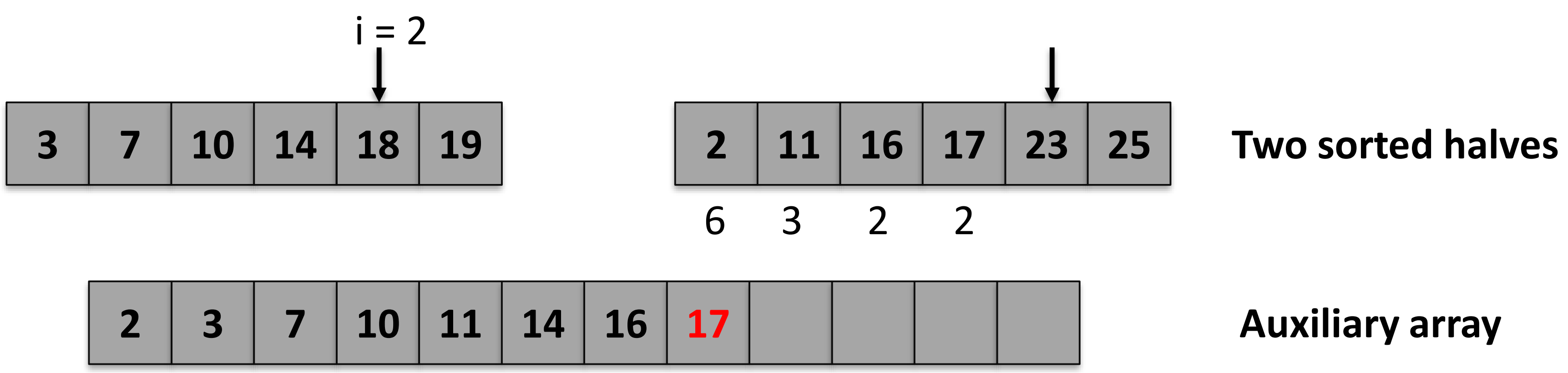
Auxiliary array

Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a $\in$ A and b $\in$ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

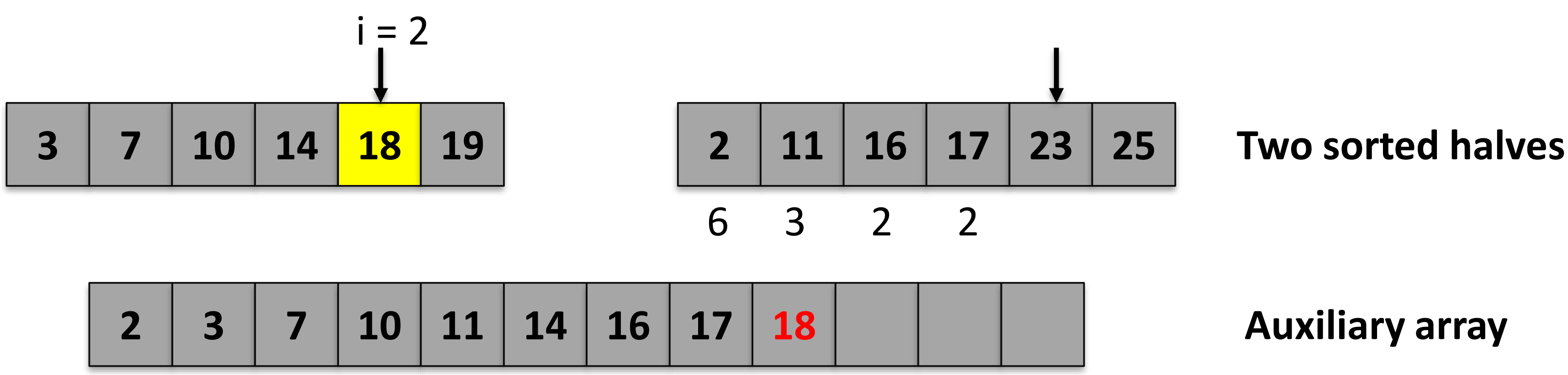


Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

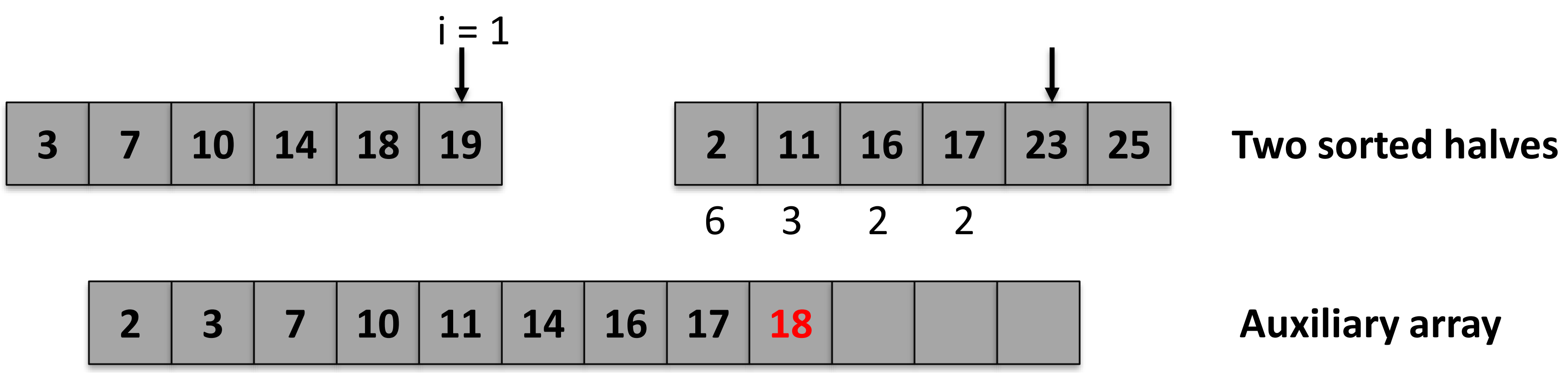


Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

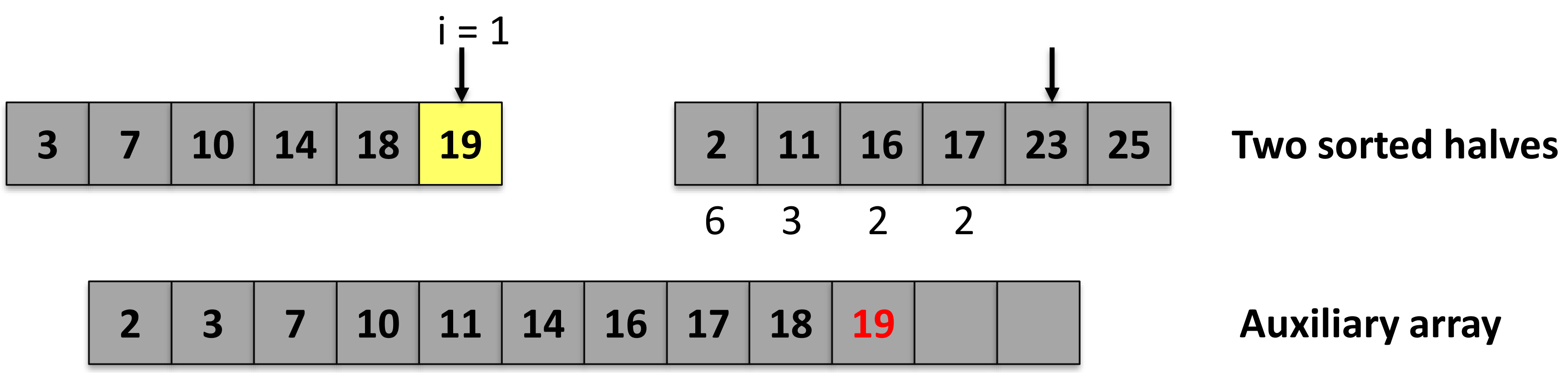


Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a $\in$ A and b $\in$ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

First half exhausted i = 0

Two sorted halves

3	**7**	**10**	**14**	**18**	**19**

2	**11**	**16**	**17**	**23**	**25**
6	3	2	2		

Auxiliary array

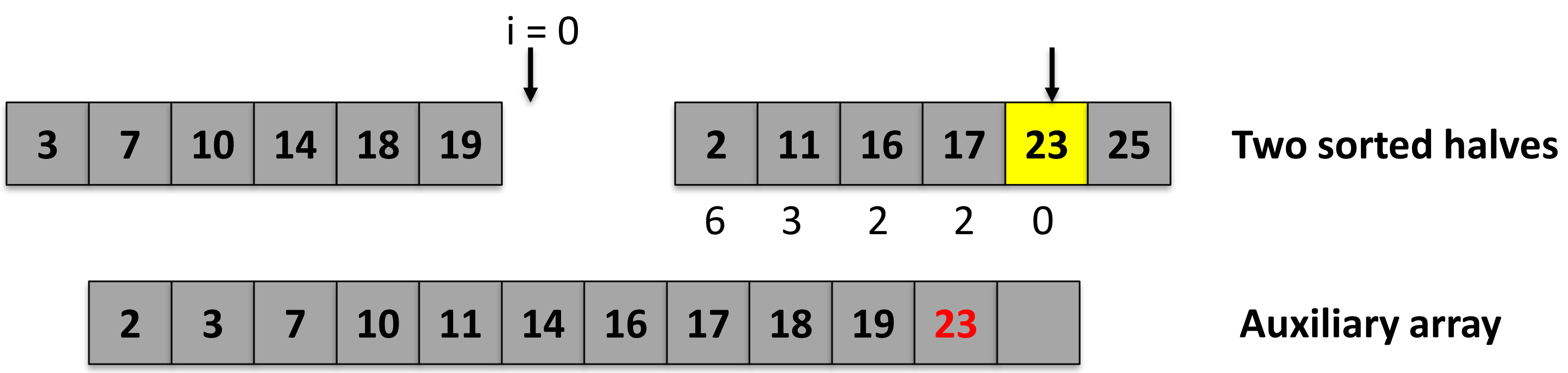
2	**3**	**7**	**10**	**11**	**14**	**16**	**17**	**18**	**19**		

Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



Total: 6+3+2+2+0

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

i = 0

3	7	10	14	18	19		2	11	16	17	23	25	**Two sorted halves**
							6	3	2	2	0	

2	3	7	10	11	14	16	17	18	19	23	25	**Auxiliary array**

Total: 6+3+2+2+0+0

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ A and b ∈ B, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

i = 0

Two sorted halves

3	**7**	**10**	**14**	**18**	**19**

2	**11**	**16**	**17**	**23**	**25**
6	3	2	2	0	

Auxiliary array

2	**3**	**7**	**10**	**11**	**14**	**16**	**17**	**18**	**19**	**23**	**25**

Total: 6+3+2+2+0+0 = 13

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0, L \leftarrow \emptyset$;

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0$,$L \leftarrow \emptyset$;
while *both* A *and* B *are not empty* **do**
| // Let a and b represent the first element of A and B, repectively

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0$,$L \leftarrow \emptyset$;
while *both* A *and* B *are not empty* **do**
  // Let a and b represent the first element of A and B, repectively
  if $a < b$ **then**

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0, L \leftarrow \emptyset$;
while *both* A *and* B *are not empty* **do**
    // Let a and b represent the first element of A and B, repectively
    if $a < b$ **then**
        Move a to the back of L;//$A.length$ is decreased by 1;
    end
    else

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0$,$L \leftarrow \emptyset$;
while *both* A *and* B *are not empty* **do**
    // Let a and b represent the first element of A and B, repectively
    if $a < b$ **then**
        Move a to the back of L;//$A.length$ is decreased by 1;
    end
    else
        Increase r by $A.length$;
        Move b to the back of L;
    end
end

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0, L \leftarrow \emptyset$;
while *both* A *and* B *are not empty* **do**
    // Let a and b represent the first element of A and B, repectively
    if $a < b$ **then**
        Move a to the back of L;//$A.length$ is decreased by 1;
    end
    else
        Increase r by $A.length$;
        Move b to the back of L;
    end
end
if A *is not empty* **then**

Combine two subproblems: Improvement

Merge-and-Count(A, B)

Input: A, B
Output: r, L
$r \leftarrow 0, L \leftarrow \emptyset$;
while *both* A *and* B *are not empty* **do**
    // Let a and b represent the first element of A and B, repectively
    if $a < b$ **then**
        Move a to the back of L;//$A.length$ is decreased by 1;
    end
    else
        Increase r by $A.length$;
        Move b to the back of L;
    end
end
if A *is not empty* **then**
    Move A to the back of L;
end
else

Combine two subproblems: Improvement

Merge-and-Count(A, B)

```
Input: A, B
Output: r, L
r ← 0, L ← ∅;
while both A and B are not empty do
    // Let a and b represent the first element of A and B, repectively
    if a < b then
        Move a to the back of L;//A.length is decreased by 1;
    end
    else
        Increase r by A.length;
        Move b to the back of L;
    end
end
if A is not empty then
    Move A to the back of L;
end
else
    Move B to the back of L;
end
return L, r;
```

Combine two subproblems: Improvement

- For every element in A and B,
  - Only O(1) times operations are executed.

Combine two subproblems: Improvement

- For every element in A and B,
  - Only O(1) times operations are executed.

- Function *Sort-and-Count(A,B)* can be executed in O(n) time where n is the number of elements in A and B.

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if $L.length = 1$ **then**
    return $0, L$;
end
Divide L into two halves A and B;

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if $L.length = 1$ **then**
  return $0, L$;
end
Divide L into two halves A and B;
$(r_A, A) \leftarrow$Sort-and-Count(A);$//T(\lceil \frac{n}{2} \rceil)$

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if $L.length = 1$ **then**
  return $0, L$;
end
Divide L into two halves A and B;
$(r_A, A) \leftarrow$Sort-and-Count(A);$//T(\lceil \frac{n}{2} \rceil)$
$(r_B, B) \leftarrow$Sort-and-Count(B);$//T(\lfloor \frac{n}{2} \rfloor)$

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if $L.length = 1$ **then**
  return $0, L$;
end
Divide L into two halves A and B;
$(r_A, A) \leftarrow$Sort-and-Count(A);$//T(\lceil \frac{n}{2} \rceil)$
$(r_B, B) \leftarrow$Sort-and-Count(B);$//T(\lfloor \frac{n}{2} \rfloor)$
$(r_L, L) \leftarrow$Merge-and-Count(A, B);$//O(n)$

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if $L.length = 1$ **then**
    return $0, L$;
end
Divide L into two halves A and B;
$(r_A, A) \leftarrow$Sort-and-Count(A);$//T(\lceil \frac{n}{2} \rceil)$
$(r_B, B) \leftarrow$Sort-and-Count(B);$//T(\lfloor \frac{n}{2} \rfloor)$
$(r_L, L) \leftarrow$Merge-and-Count(A, B);$//O(n)$
return $r_A + r_B + r_L, L$;

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if $L.length = 1$ **then**
    return $0, L$;
end
Divide L into two halves A and B;
$(r_A, A) \leftarrow$Sort-and-Count(A);$//T(\lceil \frac{n}{2} \rceil)$
$(r_B, B) \leftarrow$Sort-and-Count(B);$//T(\lfloor \frac{n}{2} \rfloor)$
$(r_L, L) \leftarrow$Merge-and-Count(A, B);$//O(n)$
return $r_A + r_B + r_L, L$;

$$T(n) = \begin{cases} O(1), & if\ n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n) & otherwise \end{cases}$$

Example

Divide

14	7	18	3	10	19	11	23	2	25	16	17

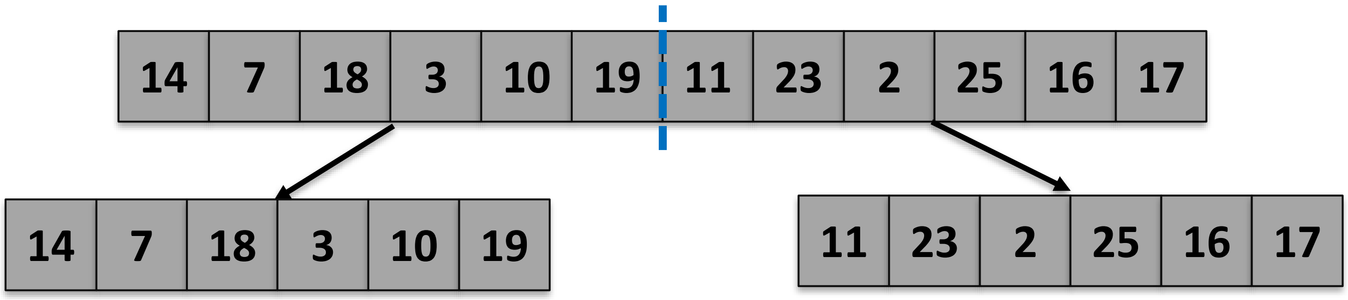
Example

Divide

14	7	18	3	10	19	11	23	2	25	16	17

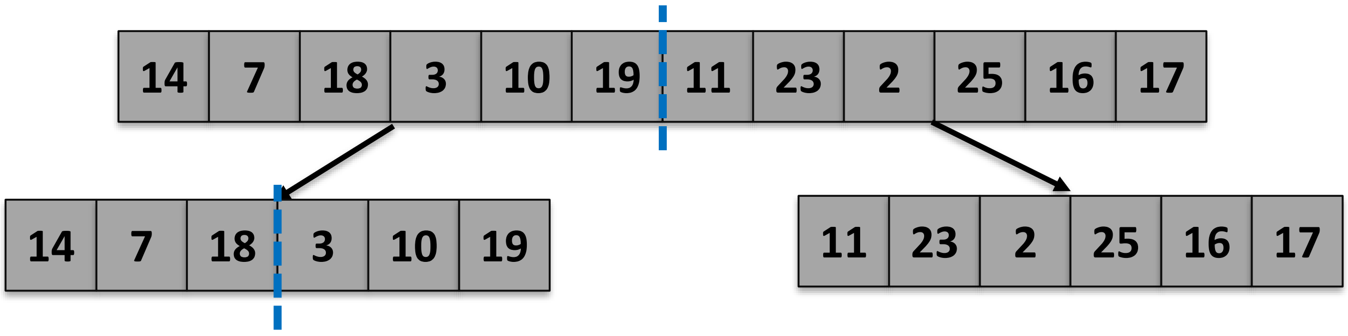
Example

Divide



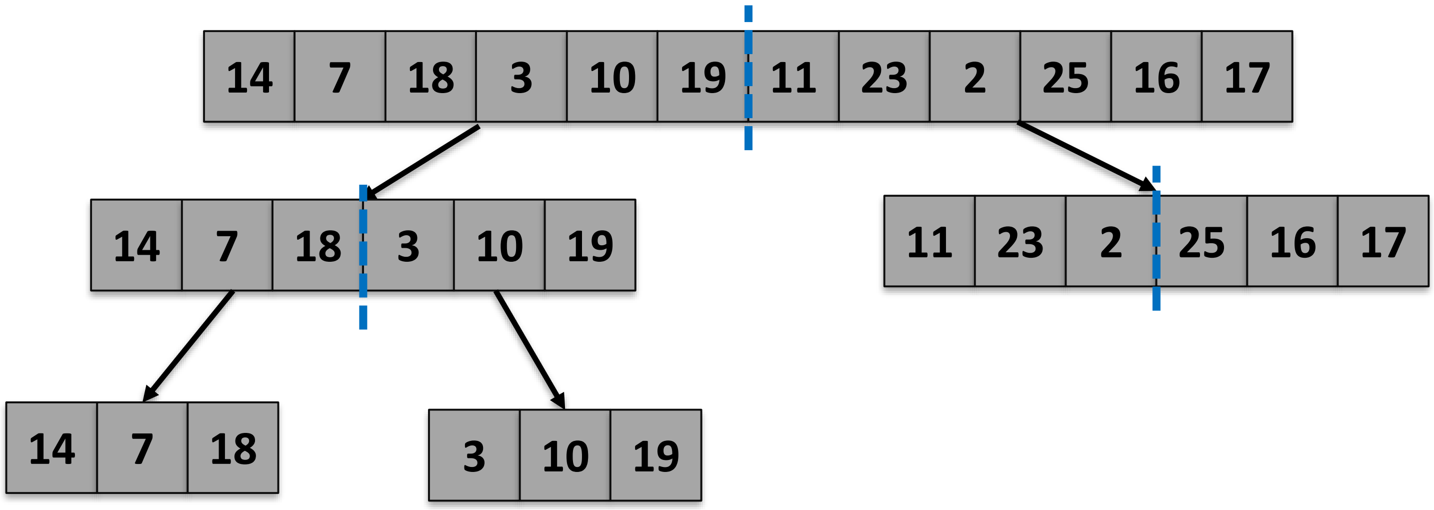
Example

Divide



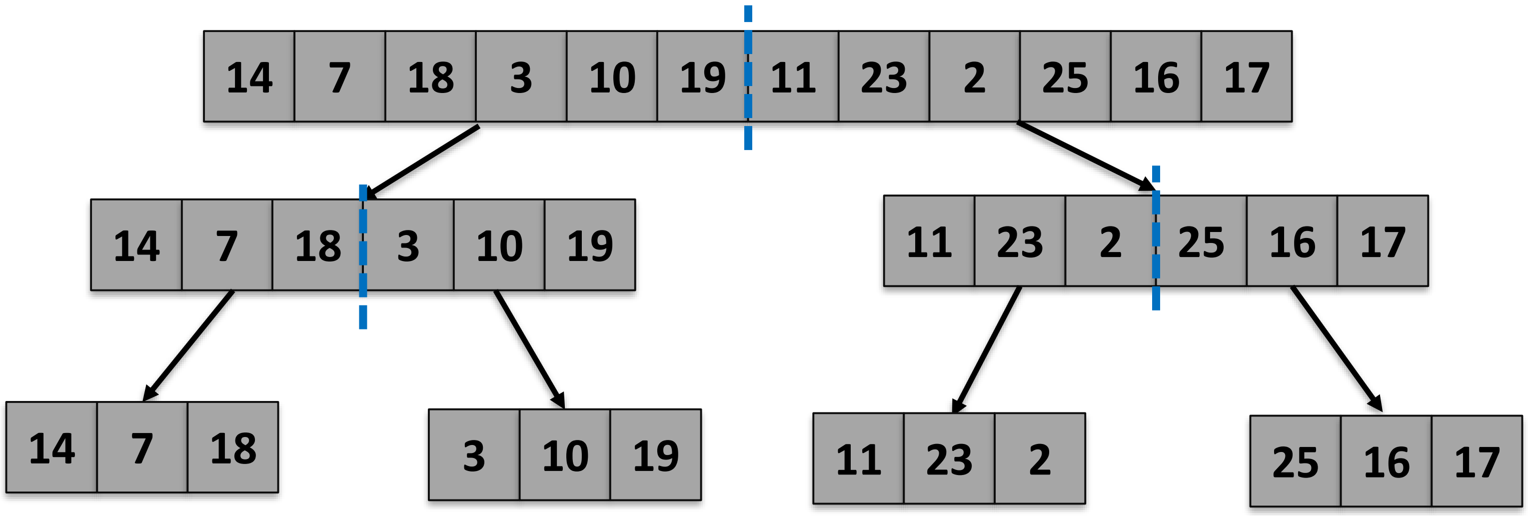
Example

Divide



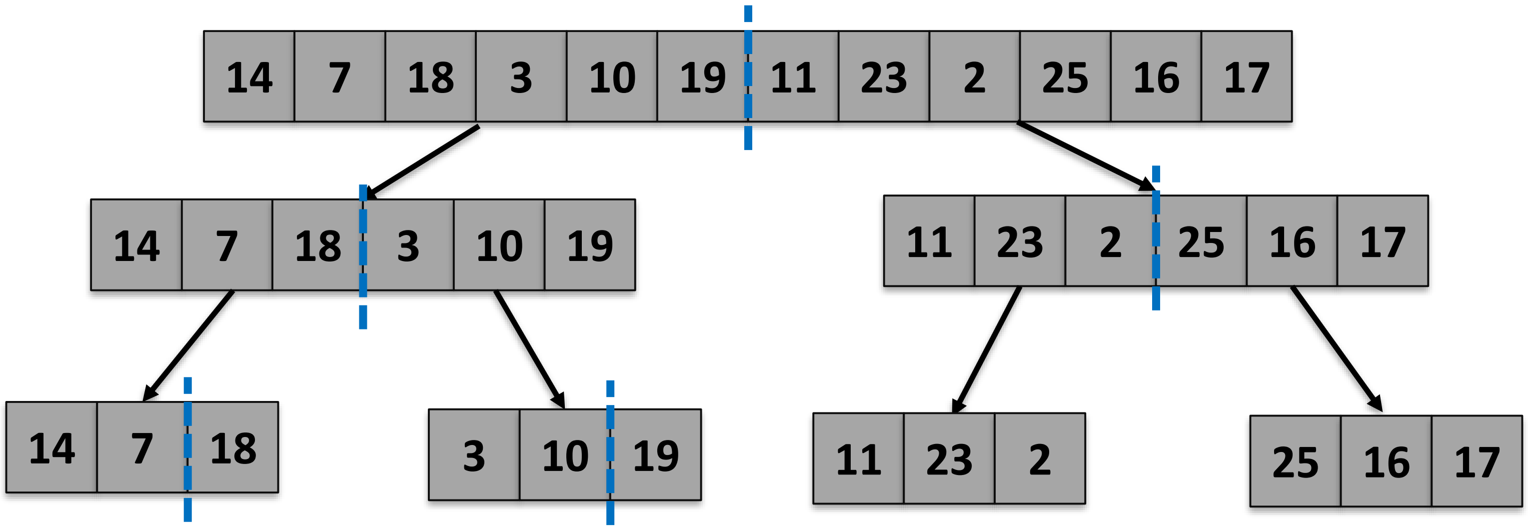
Example

Divide



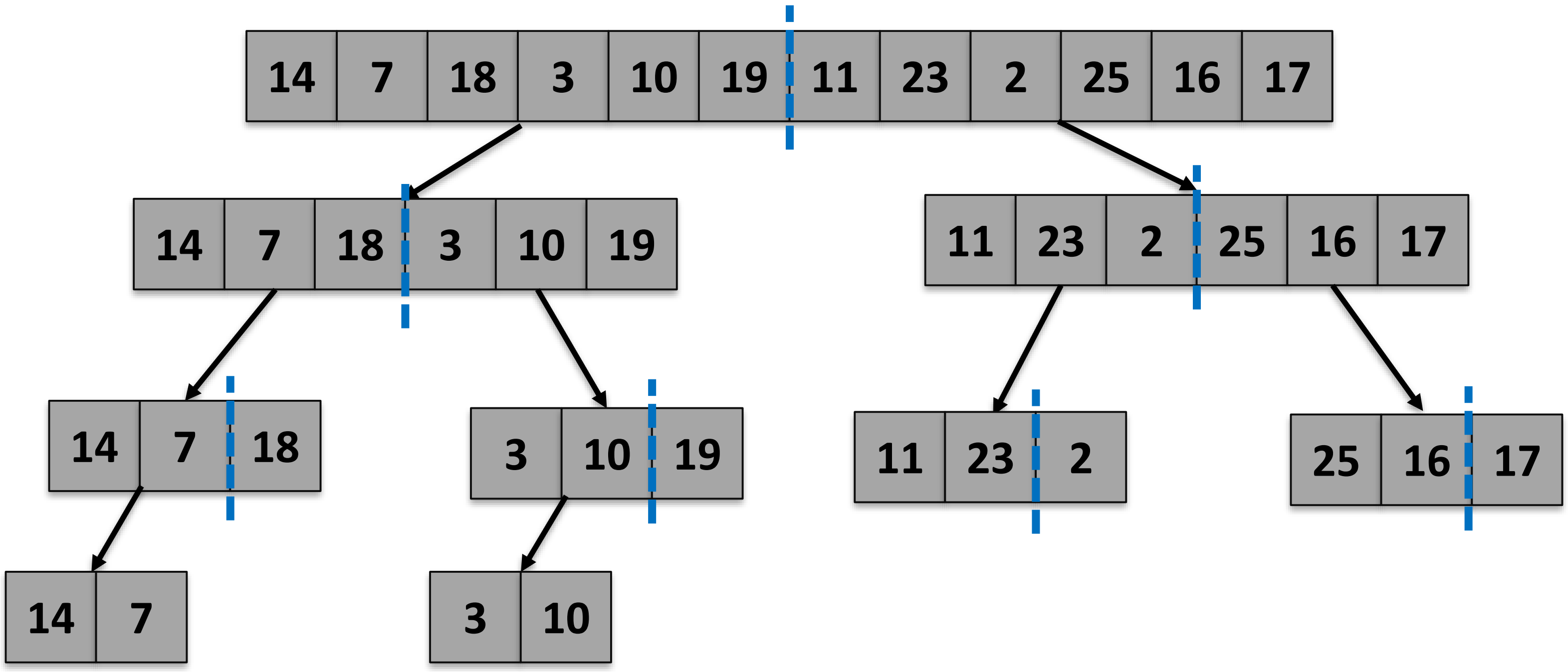
Example

Divide



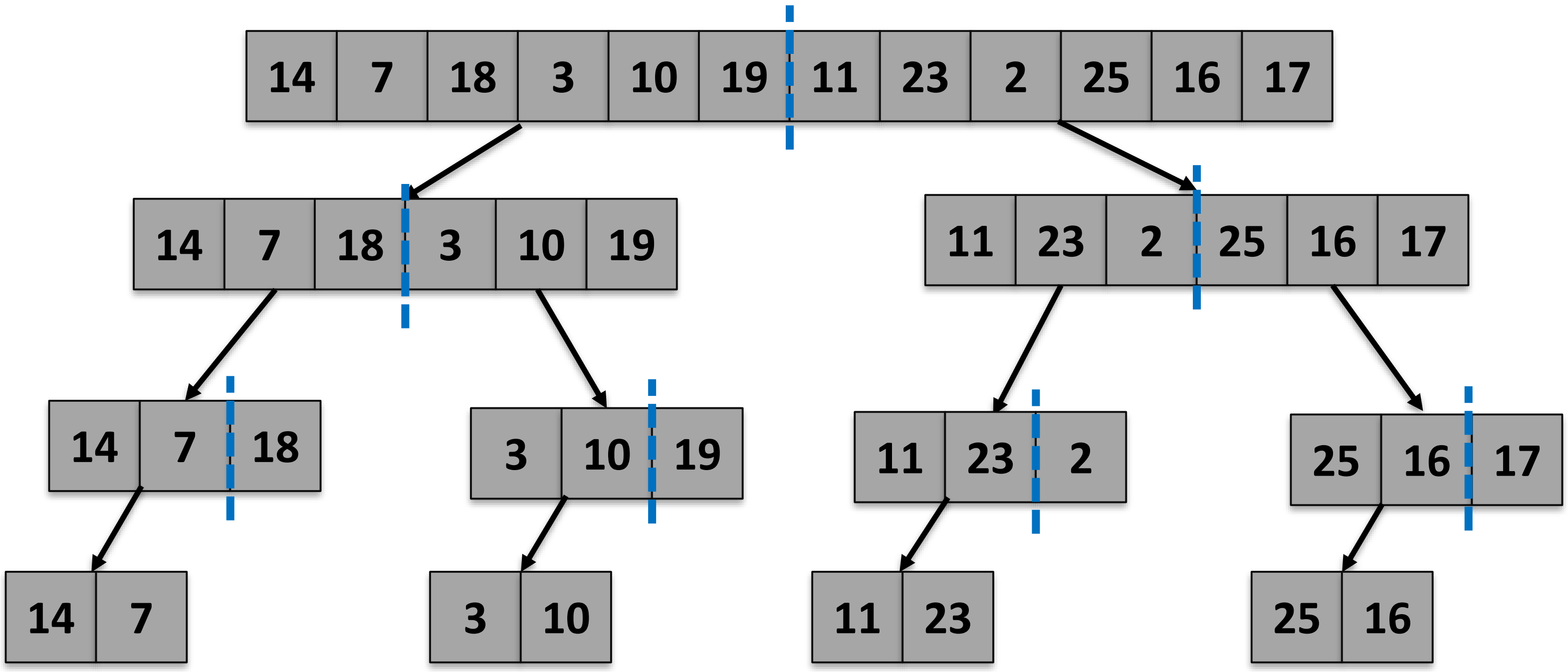
Example

Divide



Example

Divide



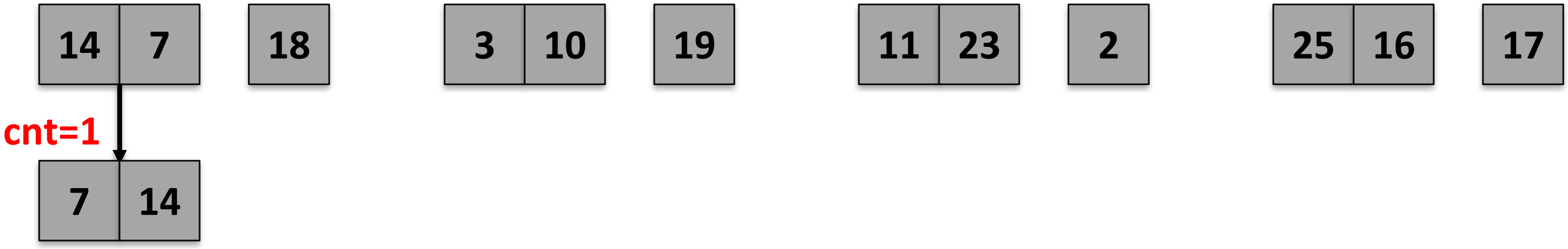
Example

Conquer

14	7	18	3	10	19	11	23	2	25	16	17

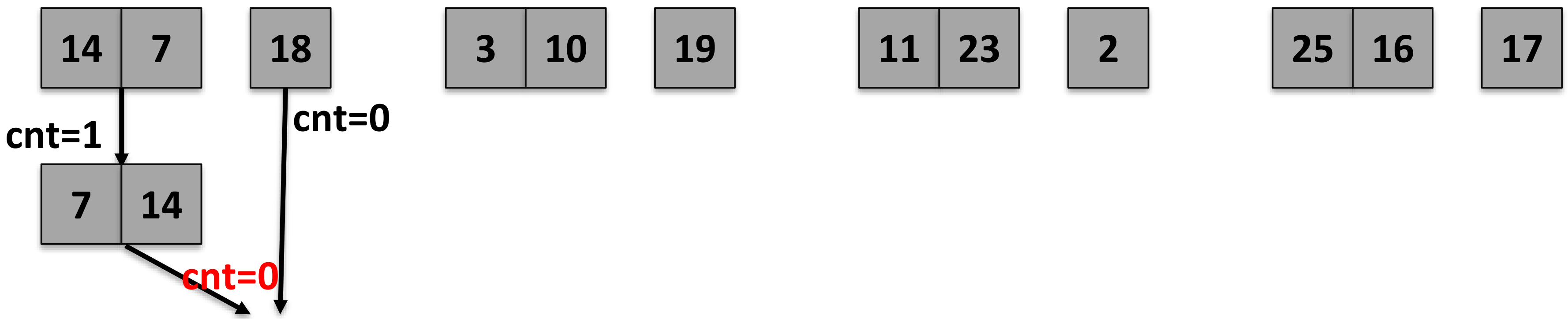
Example

Conquer



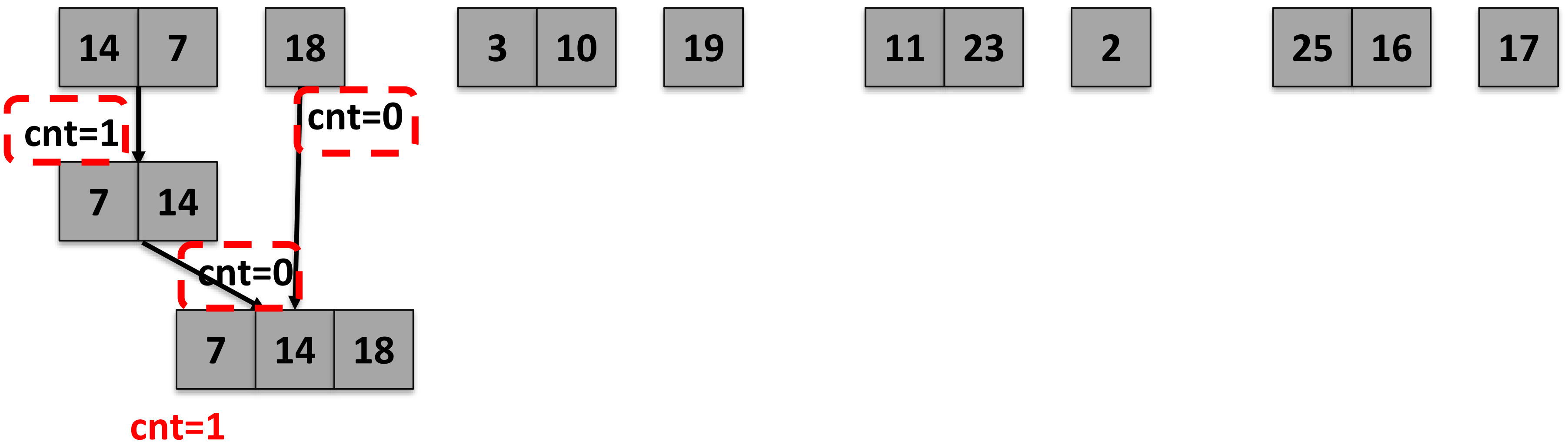
Example

Conquer



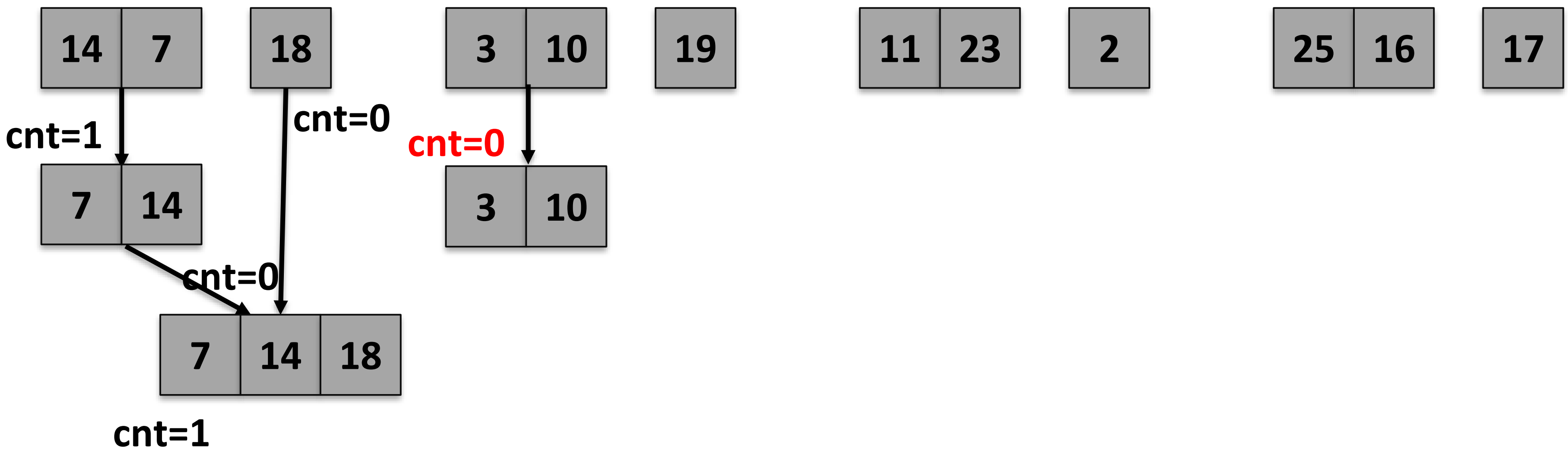
Example

Conquer



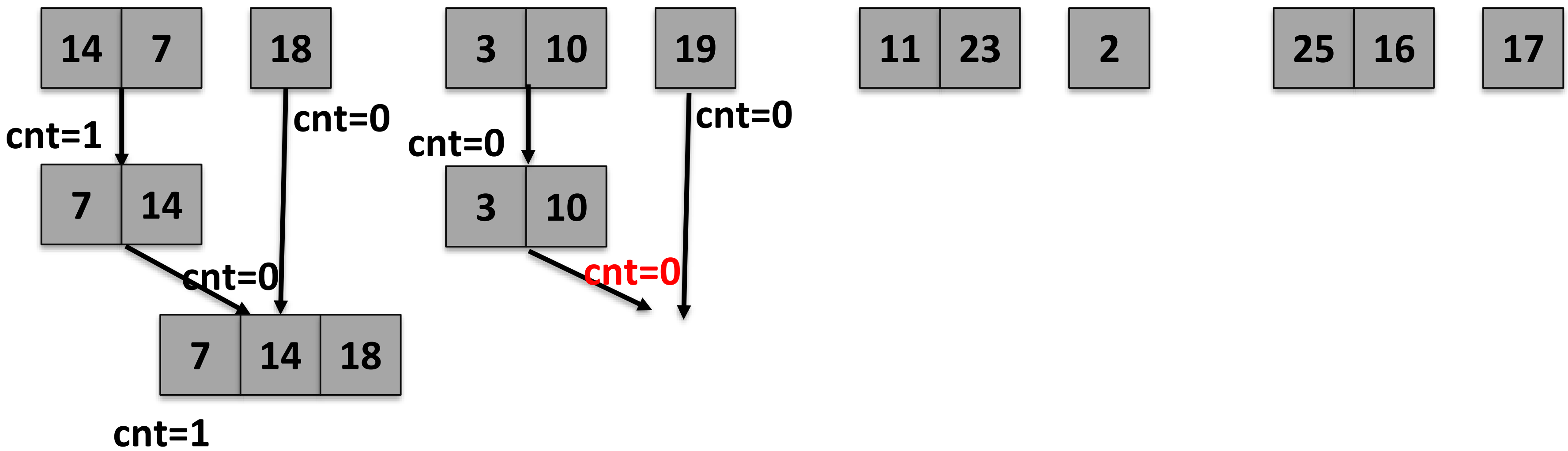
Example

Conquer



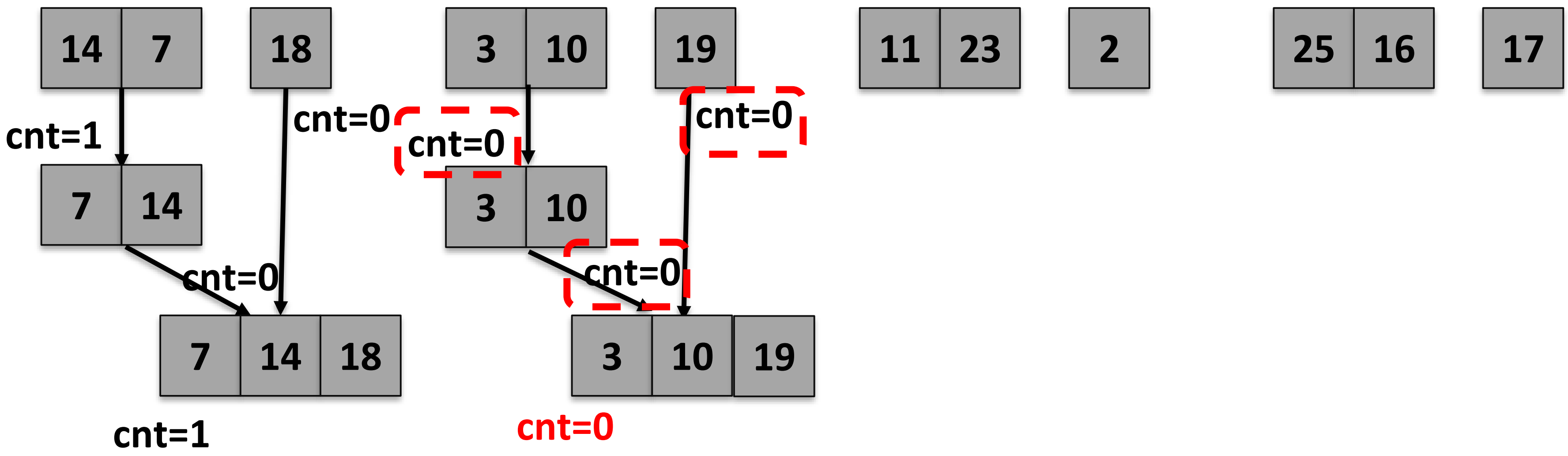
Example

Conquer



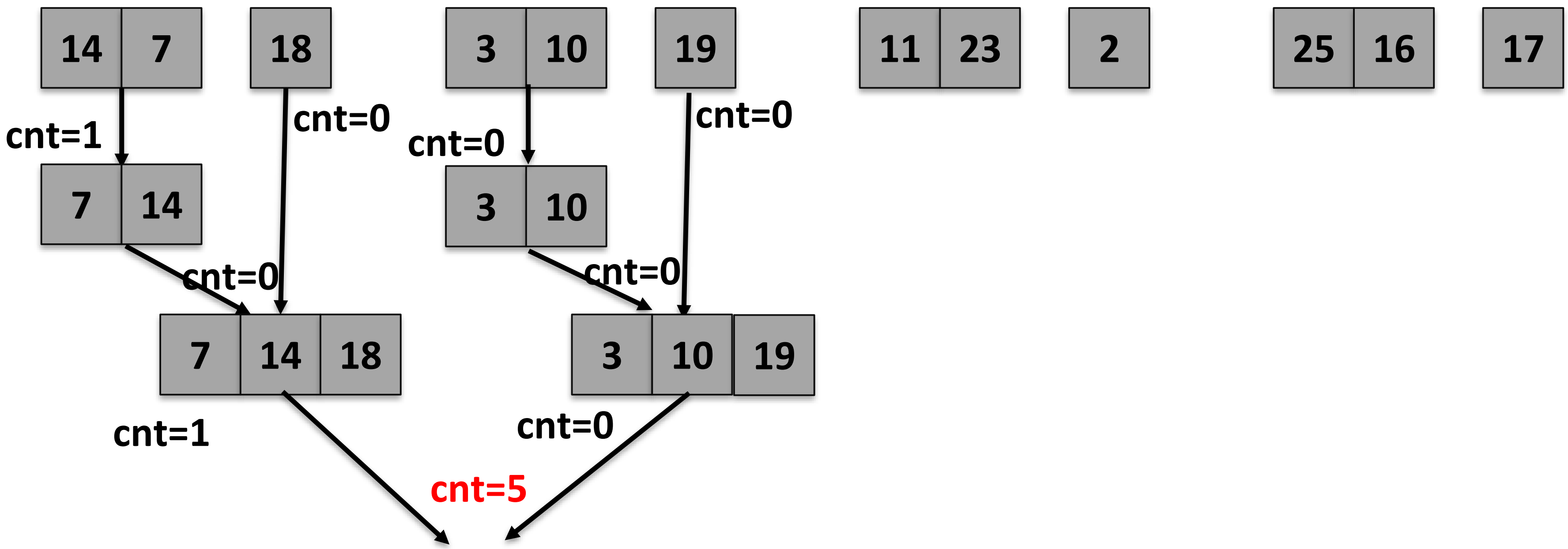
Example

Conquer



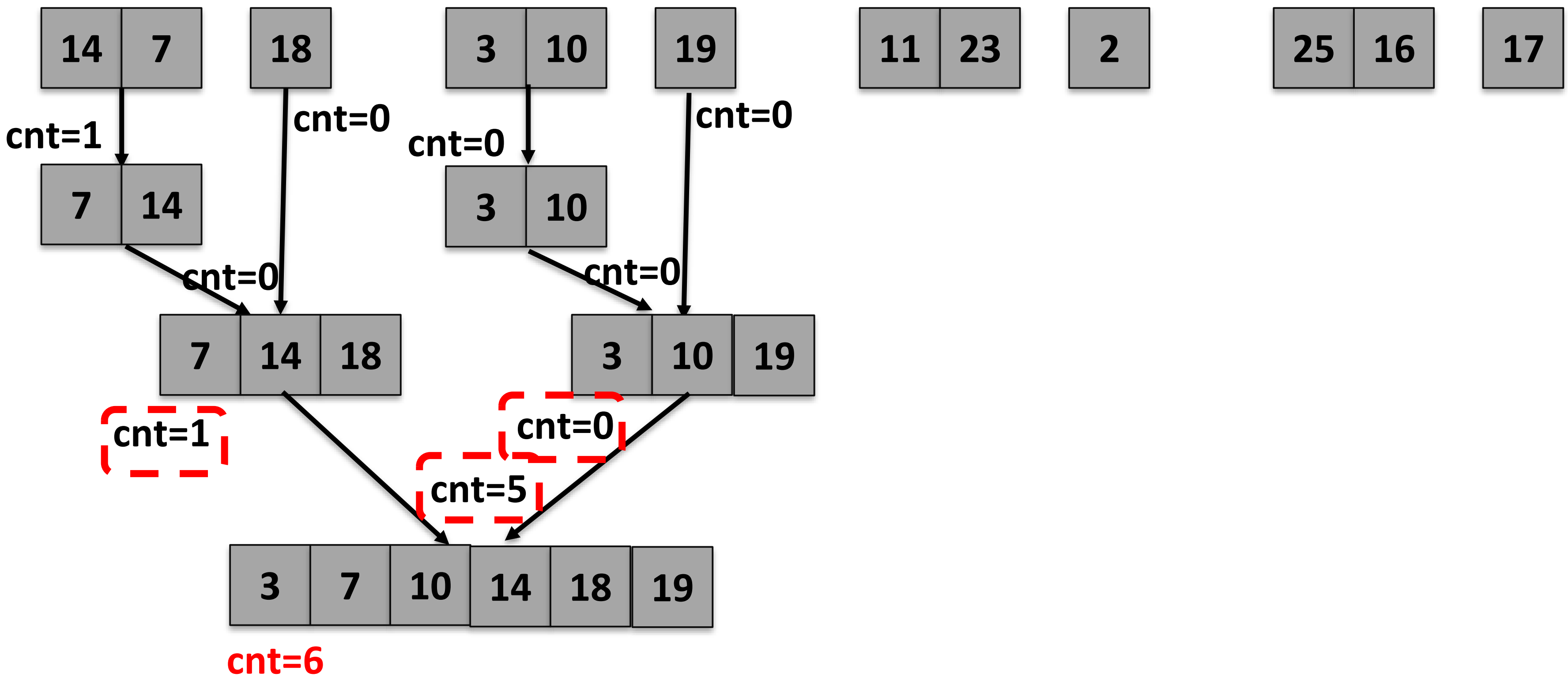
Example

Conquer



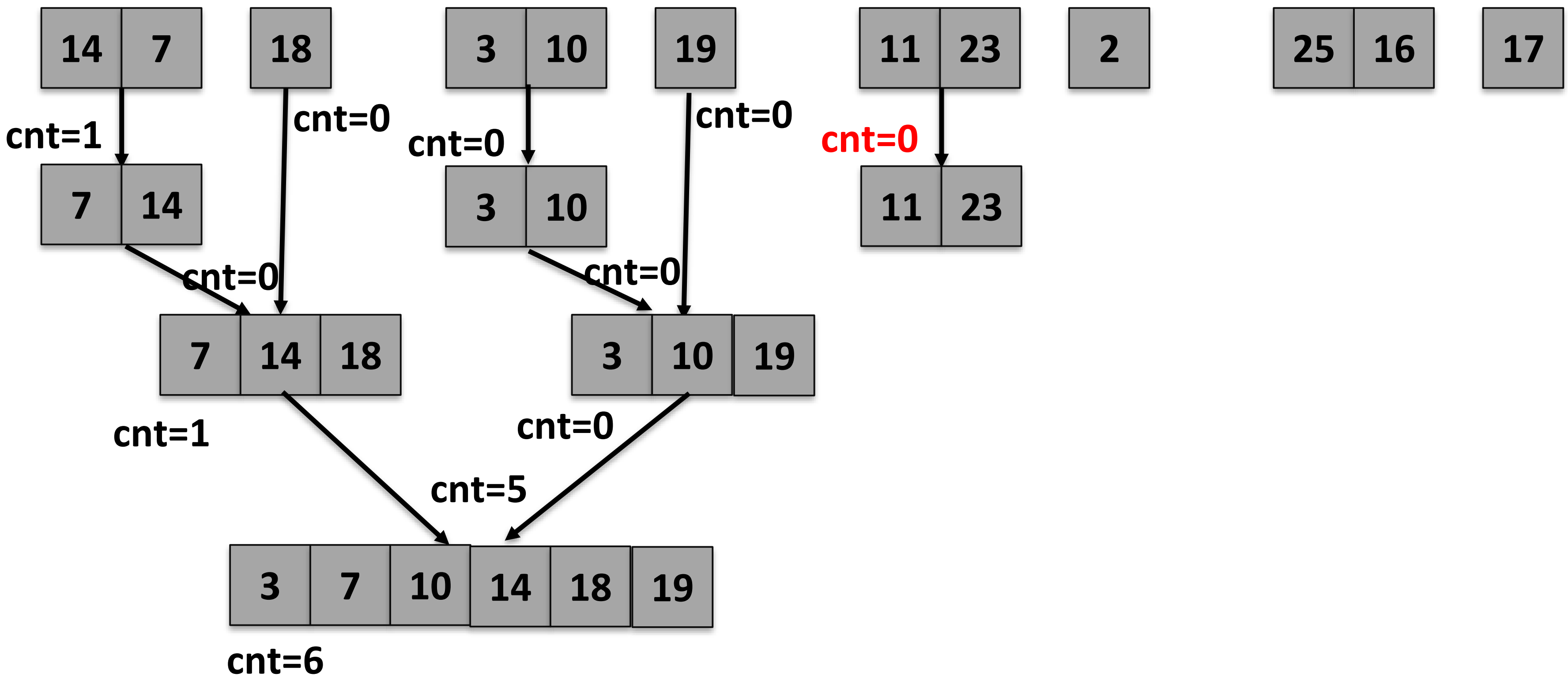
Example

Conquer



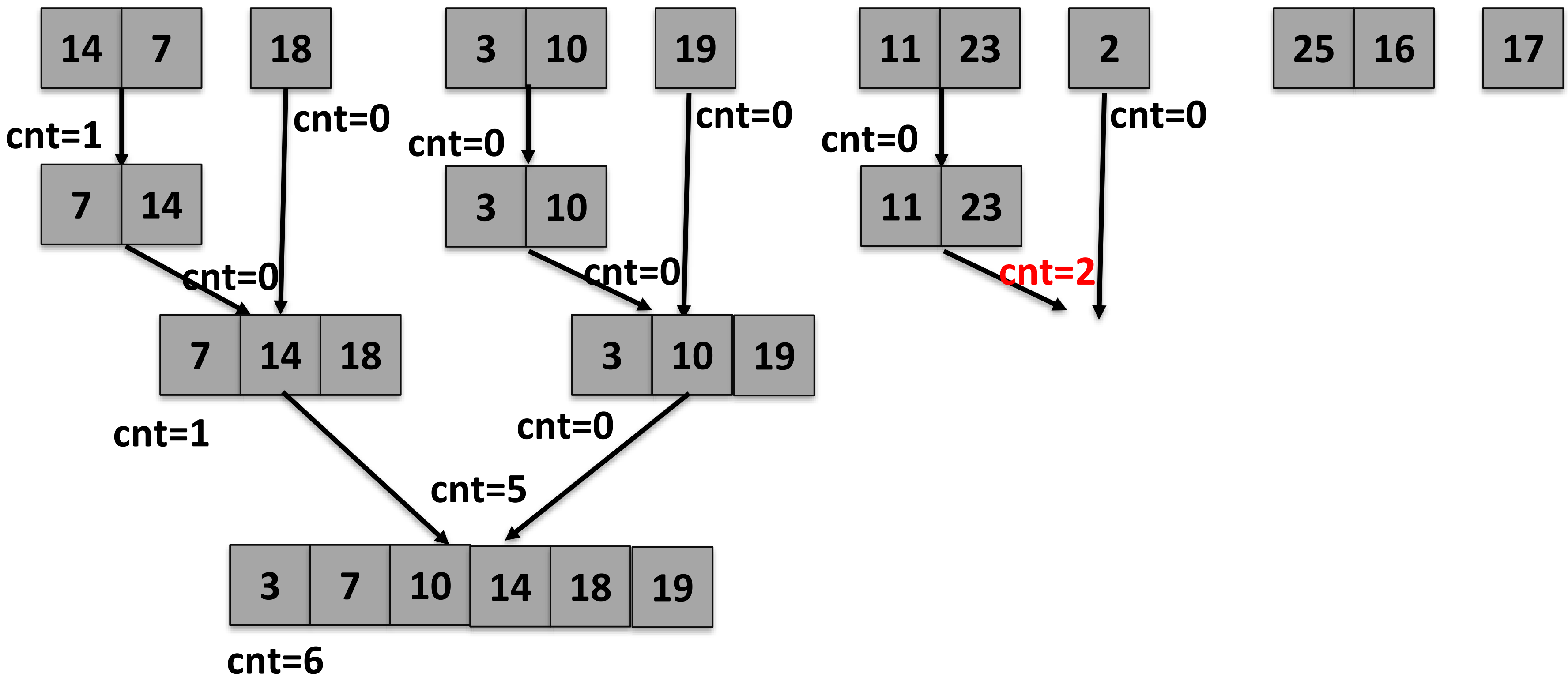
Example

Conquer



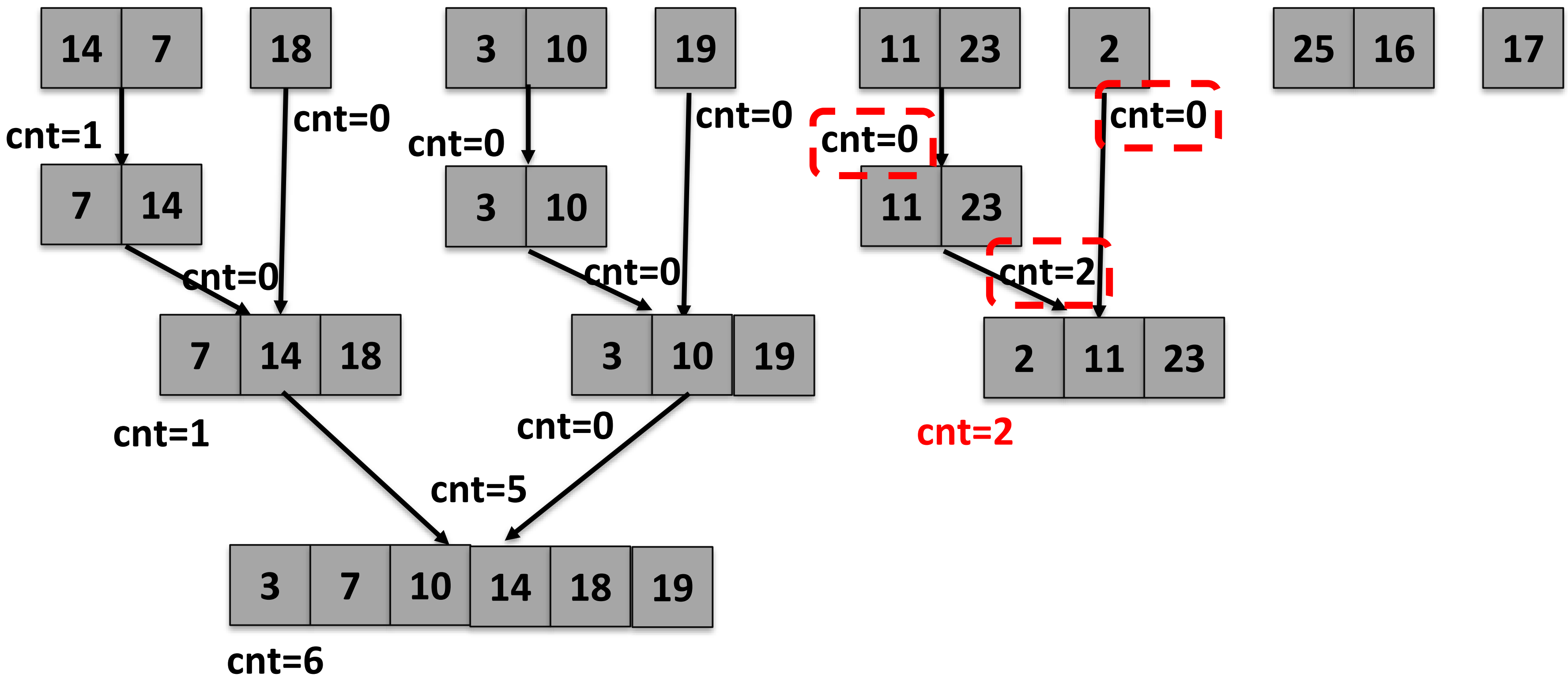
Example

Conquer



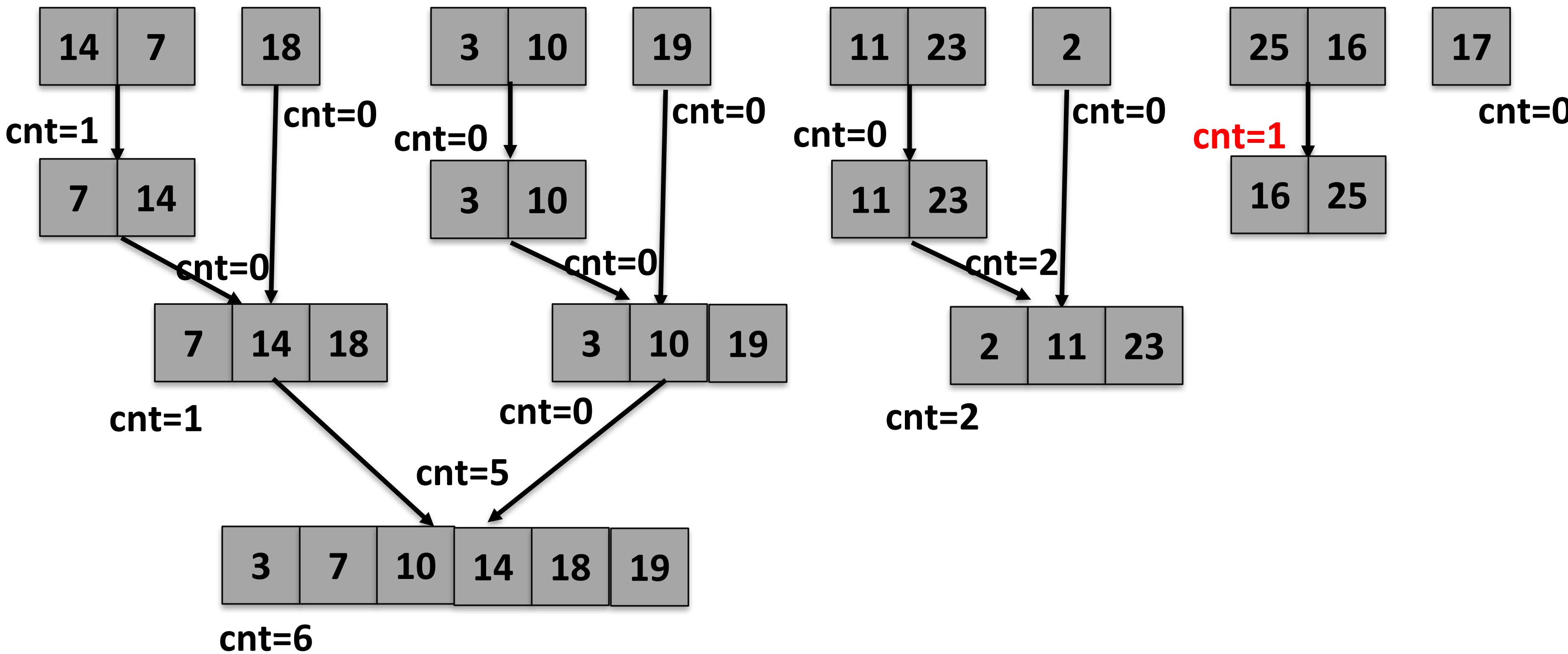
Example

Conquer



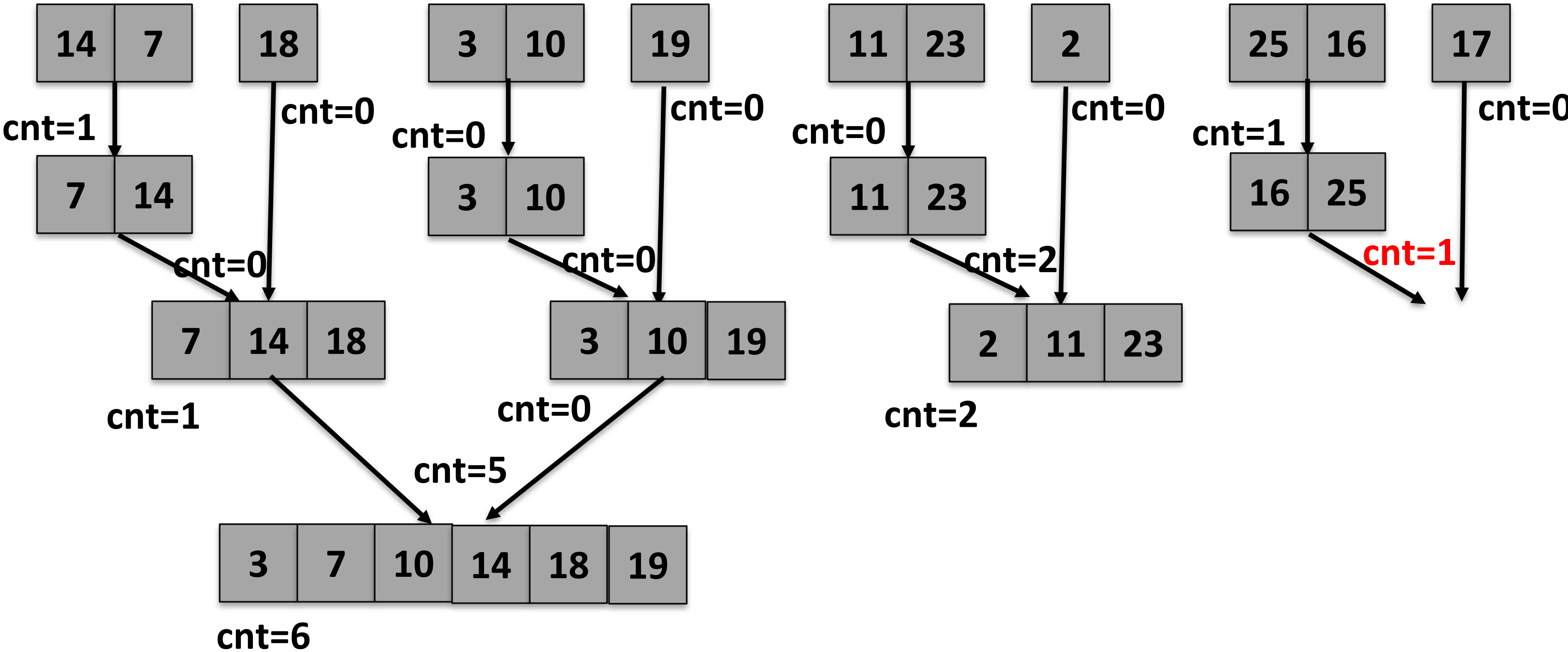
Example

Conquer



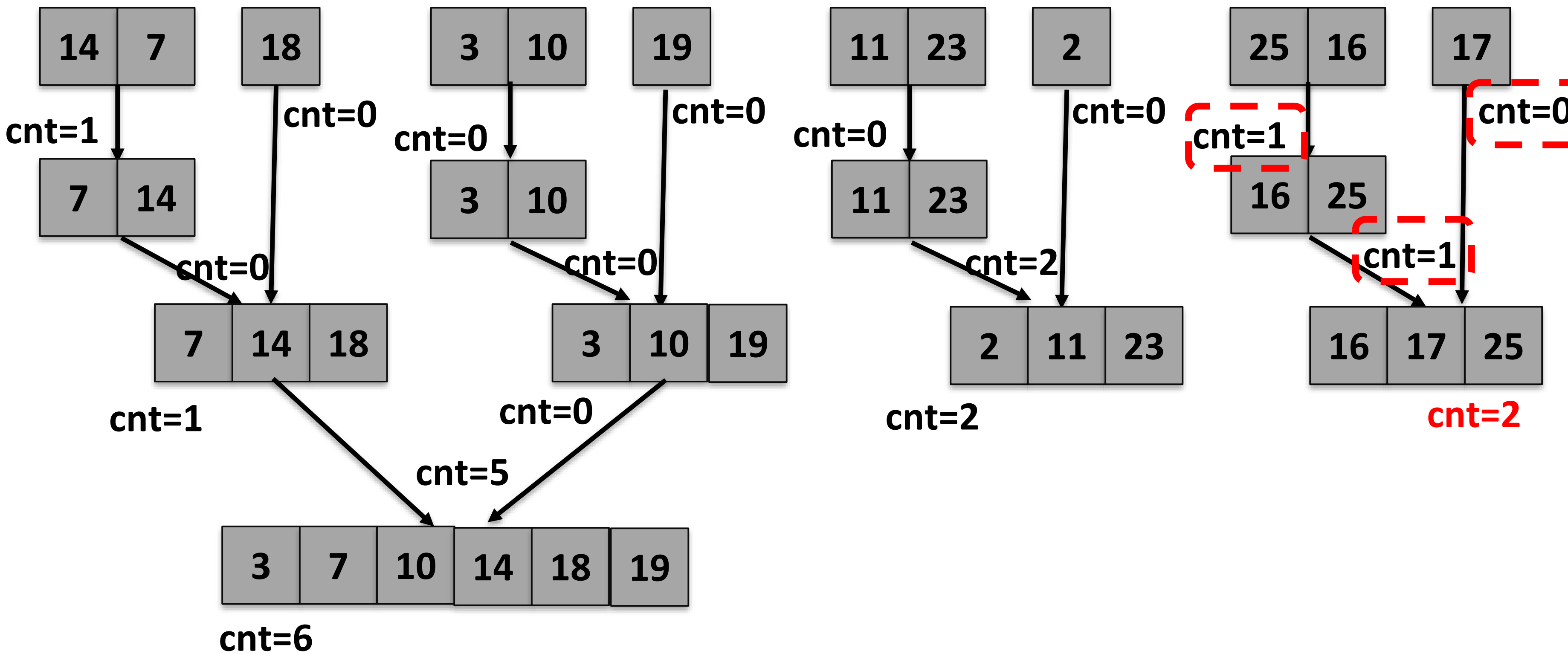
Example

Conquer



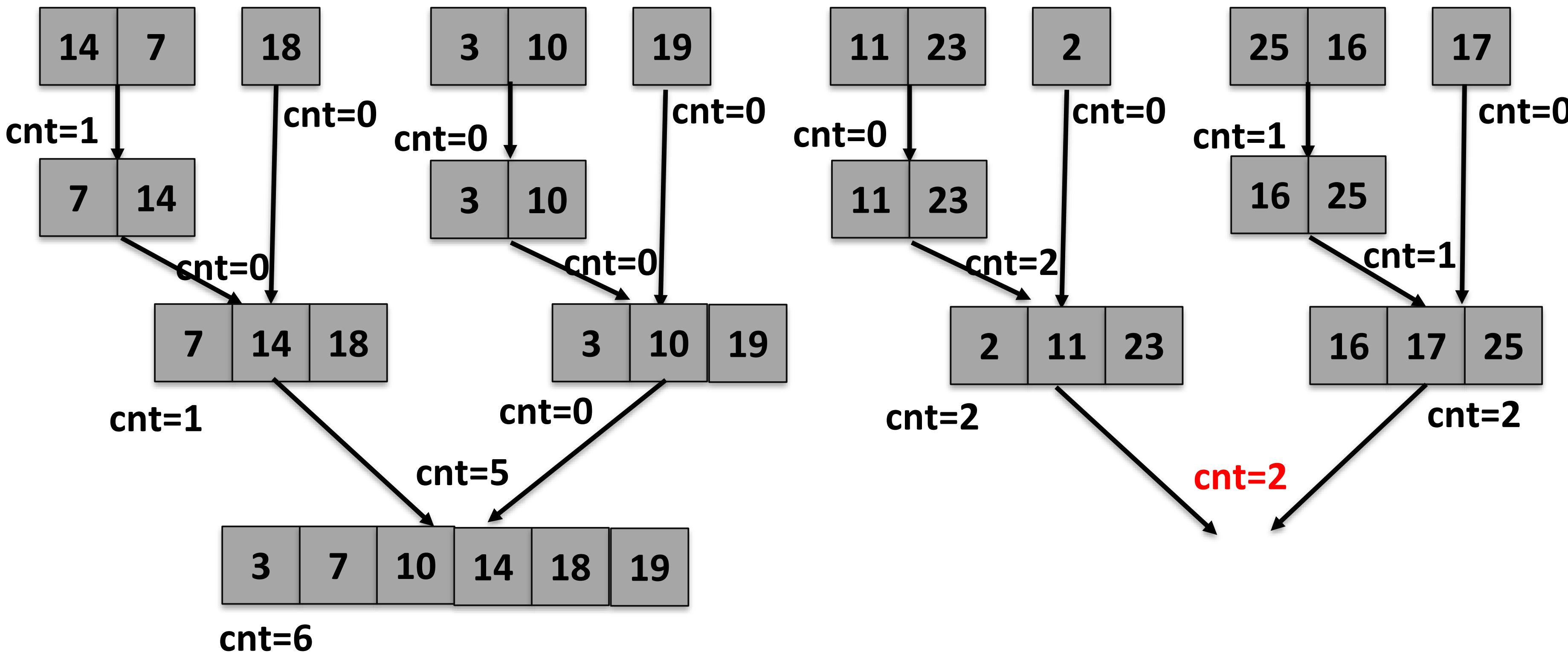
Example

Conquer



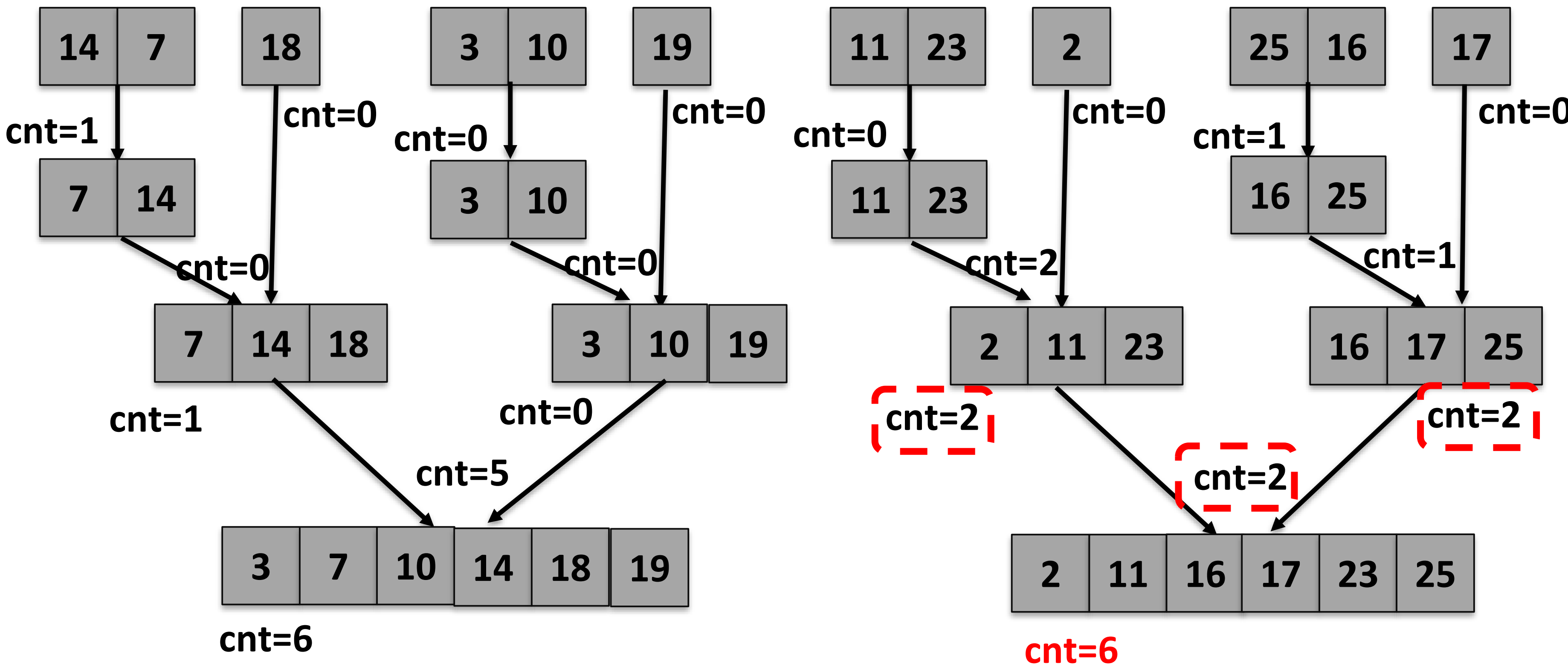
Example

Conquer



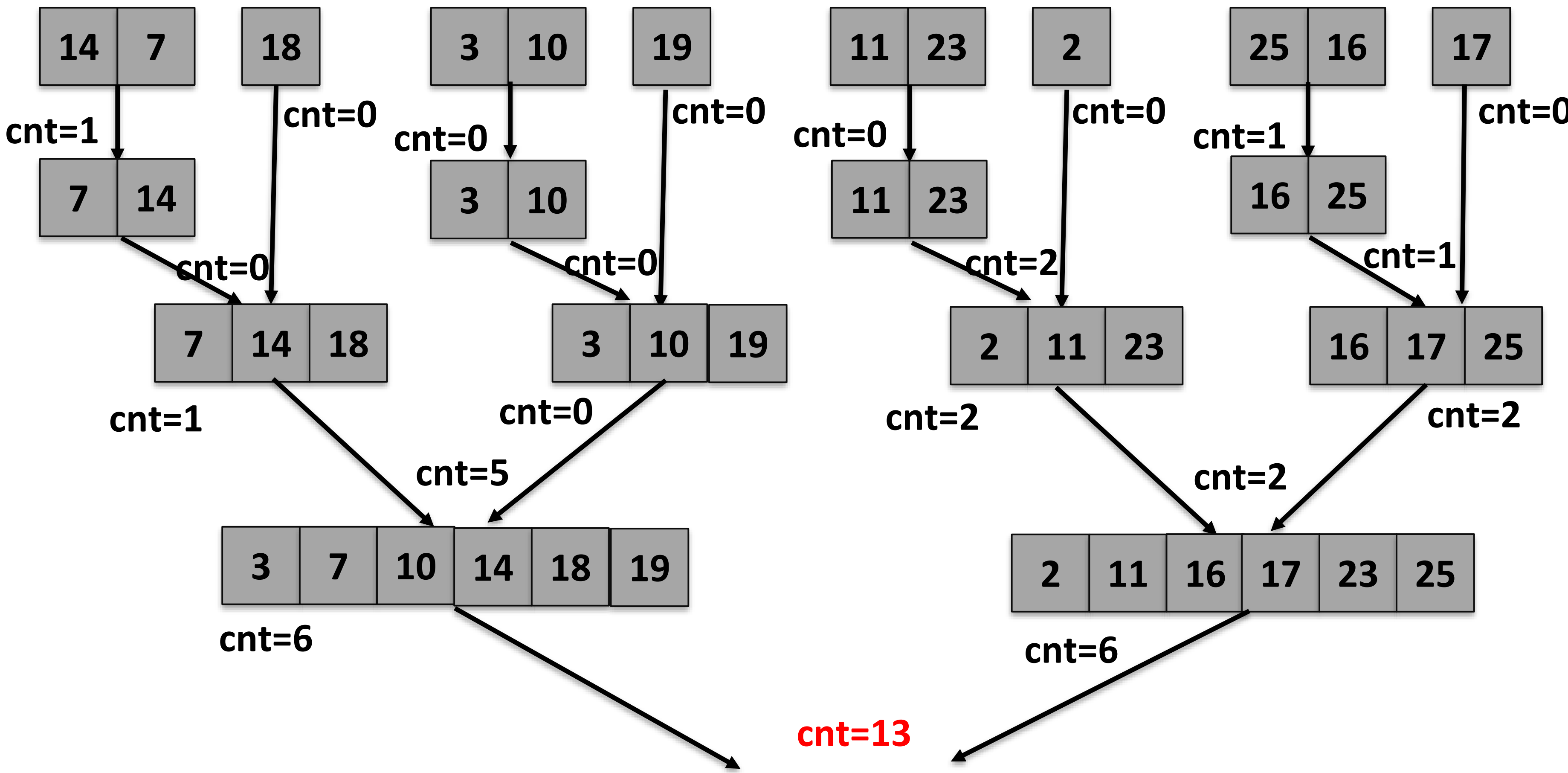
Example

Conquer



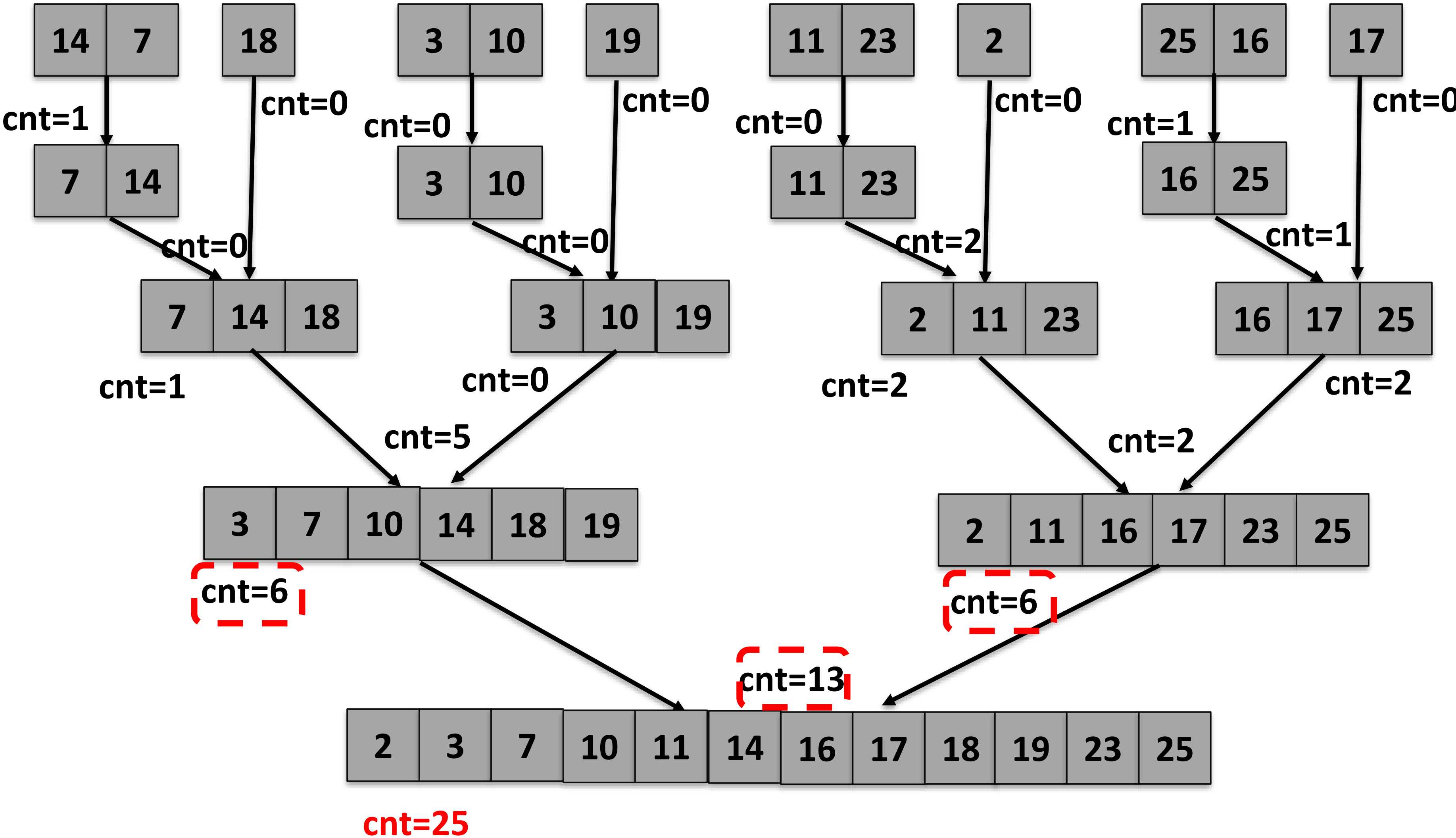
Example

Conquer



Example

Conquer



Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Analysis of the D&C Algorithm

Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Analysis of the D&C Algorithm

Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Proof. The worst-case running time T(n) satisfies the recurrence:

Analysis of the D&C Algorithm

Proposition. The sort-and-count algorithm counts the number of inversions in a permutation of size n in $O(n \log n)$ time.

Proof. The worst-case running time T(n) satisfies the recurrence:

$$T(n) = \begin{cases} O(1), & if\ n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n) & otherwise \end{cases}$$

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$
\begin{aligned}
A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\
B(x) &= b_0 + b_1 x + \cdots + b_m x^m
\end{aligned}
$$

Compute the product $A(x)B(x)$

Example

$$
\begin{aligned}
A(x) &= 1 + 2x + 3x^2 \\
B(x) &= 3 + 2x + 2x^2 \\
A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4
\end{aligned}
$$

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

Example

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

- Assume that the coefficients a_i and b_i are stored in arrays A[0…n] and B[0…m]

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

Example

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

- Assume that the coefficients a_i and b_i are stored in arrays A[0…n] and B[0…m]
- Cost: number of scalar multiplications and additions

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

- $c_k = \sum_{0 \le i \le n, 0 \le j \le m, i+j=k} a_i b_j$, for all $0 \le k \le m+n$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

- $c_k = \sum_{0 \le i \le n, 0 \le j \le m, i+j=k} a_i b_j$, for all $0 \le k \le m+n$

Definition

The vector $(c_0, c_1, \ldots, c_{m+n})$ is the convolution of the vectors $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_m)$

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

- $c_k = \sum_{0 \le i \le n, 0 \le j \le m, i+j=k} a_i b_j$, for all $0 \le k \le m+n$

Definition

The vector $(c_0, c_1, \ldots, c_{m+n})$ is the convolution of the vectors $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_m)$

- We need to calculate convolutions. This is a major problem in digital signal processing

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

A Direct (Brute Force) Approach

To ease analysis, assume n = m.

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$

A Direct (Brute Force) Approach

To ease analysis, assume n = m.

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j\,, for\ all\ 0 \le k \le 2n$$

A Direct (Brute Force) Approach

To ease analysis, assume n = m.

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i, j \le n, i+j=k} a_i b_j \, , for\ all\ 0 \le k \le 2n$$

Direct approach: Compute all c_k's using the formula above.

A Direct (Brute Force) Approach

To ease analysis, assume n = m.

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \leq i,j \leq n, i+j=k} a_i b_j \,, for\ all\ 0 \leq k \leq 2n$$

Direct approach: Compute all c_k's using the formula above.

- Total number of multiplications: $O(n^2)$

A Direct (Brute Force) Approach

To ease analysis, assume n = m.

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i, j \le n, i+j=k} a_i b_j \, , for \; all \; 0 \le k \le 2n$$

Direct approach: Compute all c_k's using the formula above.

- Total number of multiplications: $O(n^2)$
- Total number of additions: $O(n^2)$

A Direct (Brute Force) Approach

To ease analysis, assume n = m.

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j \,, for\ all\ 0 \le k \le 2n$$

Direct approach: Compute all c_k's using the formula above.

- Total number of multiplications: $O(n^2)$
- Total number of additions: $O(n^2)$
- Complexity: $O(n^2)$

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) =$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) +$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a_n}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) =$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x)x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x)x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) +$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} \\ +$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}}$$
$$+A_1(x)B_0(x)x^{\frac{n}{2}} +$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(\mathrm{x}) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} + A_1(x)B_0(x)x^{\frac{n}{2}} + A_1(x)B_1(x)x^n$$

The First Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$A_1(x) = \mathrm{a}_{\frac{\mathrm{n}}{2}} + \mathrm{a}_{\frac{\mathrm{n}}{2}+1}\mathrm{x} + \cdots + \mathrm{a}_{\mathrm{n}}\mathrm{x}^{\frac{\mathrm{n}}{2}}$$

$$A(x) = A_0(x) + A_1(x) x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x) x^{\frac{n}{2}}$$

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} + A_1(x)B_0(x)x^{\frac{n}{2}} + A_1(x)B_1(x)x^n$$

The original problem (of size n) is divided into 4 problems of input size n/2

Example

$$A(x) = 2 + 5x + 3x^2 + x^3 - x^4$$
$$B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$$
$$A(x)B(x) = 2 + 9x + 17x^2 + 23x^3 + 34x^4$$
$$+39x^5 + 19x^6 + 3\mathrm{x}^7 - 6\mathrm{x}^8$$
$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2$$
$$A(x) = A_0(x) + A_1(x)x^2$$

Example

$$A(x) = 2 + 5x + 3x^2 + x^3 - x^4$$
$$B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$$
$$A(x)B(x) = 2 + 9x + 17x^2 + 23x^3 + 34x^4$$
$$+39x^5 + 19x^6 + 3\text{x}^7 - 6\text{x}^8$$
$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2$$
$$A(x) = A_0(x) + A_1(x)x^2$$
$$B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2$$
$$B(x) = B_0(x) + B_1(x)x^2$$

Example

$$A(x) = 2 + 5x + 3x^2 + x^3 - x^4$$
$$B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$$
$$A(x)B(x) = 2 + 9x + 17x^2 + 23x^3 + 34x^4$$
$$+39x^5 + 19x^6 + 3\mathrm{x}^7 - 6\mathrm{x}^8$$
$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2$$
$$A(x) = A_0(x) + A_1(x)x^2$$
$$B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2$$
$$B(x) = B_0(x) + B_1(x)x^2$$
$$A_0(x)B_0(x) = 2 + 9x + 10x^2$$
$$A_1(x)B_1(x) = 6 + 11x + 19x^2 + 3x^3 - 6x^4$$
$$A_0(x)B_1(x) = 4 + 16x + 27x^2 + 30x^3$$
$$A_1(x)B_0(x) = 3 + 7x + x^2 - 2x^3$$

Example

$$A(x) = 2 + 5x + 3x^2 + x^3 - x^4$$
$$B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$$
$$A(x)B(x) = 2 + 9x + 17x^2 + 23x^3 + 34x^4$$
$$+39x^5 + 19x^6 + 3\text{x}^7 - 6\text{x}^8$$
$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2$$
$$A(x) = A_0(x) + A_1(x)x^2$$
$$B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2$$
$$B(x) = B_0(x) + B_1(x)x^2$$
$$A_0(x)B_0(x) = 2 + 9x + 10x^2$$
$$A_1(x)B_1(x) = 6 + 11x + 19x^2 + 3x^3 - 6x^4$$
$$A_0(x)B_1(x) = 4 + 16x + 27x^2 + 30x^3$$
$$A_1(x)B_0(x) = 3 + 7x + x^2 - 2x^3$$
$$A_0(x)B_1(x) + A_1(x)B_0(x) = 7 + 23x + 28x^2 + 28x^3$$

$$A_0(x)B_0(x) + \big(A_0(x)B_1(x) + A_1(x)B_0(x)\big)x^2 + A_1(x)B_1(x)x^4$$

Example

$$A(x) = 2 + 5x + 3x^2 + x^3 - x^4$$
$$B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$$
$$A(x)B(x) = 2 + 9x + 17x^2 + 23x^3 + 34x^4$$
$$+39x^5 + 19x^6 + 3\mathrm{x}^7 - 6\mathrm{x}^8$$
$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2$$
$$A(x) = A_0(x) + A_1(x)x^2$$
$$B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2$$
$$B(x) = B_0(x) + B_1(x)x^2$$
$$A_0(x)B_0(x) = 2 + 9x + 10x^2$$
$$A_1(x)B_1(x) = 6 + 11x + 19x^2 + 3x^3 - 6x^4$$
$$A_0(x)B_1(x) = 4 + 16x + 27x^2 + 30x^3$$
$$A_1(x)B_0(x) = 3 + 7x + x^2 - 2x^3$$
$$A_0(x)B_1(x) + A_1(x)B_0(x) = 7 + 23x + 28x^2 + 28x^3$$

$$A_0(x)B_0(x) + \big(A_0(x)B_1(x) + A_1(x)B_0(x)\big)x^2 + A_1(x)B_1(x)x^4$$
$$= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 + 19x^6 + 3x^7 - 6x^8$$

The First Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- Compute $A_0(x)B_0(x), A_0(x)B_1(x), A_1(x)B_0(x), \mathrm{A}_1(\mathrm{x})\mathrm{B}_1(\mathrm{x})$

The First Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- Compute $A_0(x)B_0(x), A_0(x)B_1(x), A_1(x)B_0(x), \mathrm{A}_1(\mathrm{x})\mathrm{B}_1(\mathrm{x})$ by recursively calling the algorithm 4 times

The First Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- Compute $A_0(x)B_0(x), A_0(x)B_1(x), A_1(x)B_0(x), \mathrm{A}_1(\mathrm{x})\mathrm{B}_1(\mathrm{x})$ by recursively calling the algorithm 4 times

Combine

- Add the following four polynomials

$$A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} \\ + A_1(x)B_0(x)x^{\frac{n}{2}} \\ + A_1(x)B_1(x)x^{n}$$

The First Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- Compute
  $A_0(x)B_0(x), A_0(x)B_1(x), A_1(x)B_0(x), \mathrm{A}_1(\mathrm{x})\mathrm{B}_1(\mathrm{x})$
  by recursively calling the algorithm 4 times

Combine

- Add the following four polynomials

$$
\begin{aligned}
A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} \\
+ A_1(x)B_0(x)x^{\frac{n}{2}} \\
+ A_1(x)B_1(x)x^{n}
\end{aligned}
$$

- Takes O(n) operations

The First Divide-and-Conquer Algorithm

$\mathrm{PolyMulti1}(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$;
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}}$;

The First Divide-and-Conquer Algorithm

$\text{PolyMulti1}(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}};$
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}};$
$U(x) \leftarrow \text{PolyMulti1}(A_0(x), B_0(x)); // T(n/2)$
$V(x) \leftarrow \text{PolyMulti1}(A_0(x), B_1(x)); // T(n/2)$
$W(x) \leftarrow \text{PolyMulti1}(A_1(x), B_0(x)); // T(n/2)$
$Z(x) \leftarrow \text{PolyMulti1}(A_1(x), B_1(x)); // T(n/2)$

The First Divide-and-Conquer Algorithm

PolyMulti1$(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$;
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}}$;
$U(x) \leftarrow$PolyMulti1$(A_0(x), B_0(x))$;$//T(n/2)$
$V(x) \leftarrow$PolyMulti1$(A_0(x), B_1(x))$;$//T(n/2)$
$W(x) \leftarrow$PolyMulti1$(A_1(x), B_0(x))$;$//T(n/2)$
$Z(x) \leftarrow$PolyMulti1$(A_1(x), B_1(x))$;$//T(n/2)$
return $(U(x) + [V(x) + W(x)]x^{\frac{n}{2}} + Z(x)x^n)$;$//O(n)$

The First Divide-and-Conquer Algorithm

PolyMulti1$(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}}$;
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{\frac{n}{2}}$;
$U(x) \leftarrow$PolyMulti1$(A_0(x), B_0(x))$;$//T(n/2)$
$V(x) \leftarrow$PolyMulti1$(A_0(x), B_1(x))$;$//T(n/2)$
$W(x) \leftarrow$PolyMulti1$(A_1(x), B_0(x))$;$//T(n/2)$
$Z(x) \leftarrow$PolyMulti1$(A_1(x), B_1(x))$;$//T(n/2)$
return $(U(x) + [V(x) + W(x)]x^{\frac{n}{2}} + Z(x)x^n)$;$//O(n)$

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

Analysis of Running Time

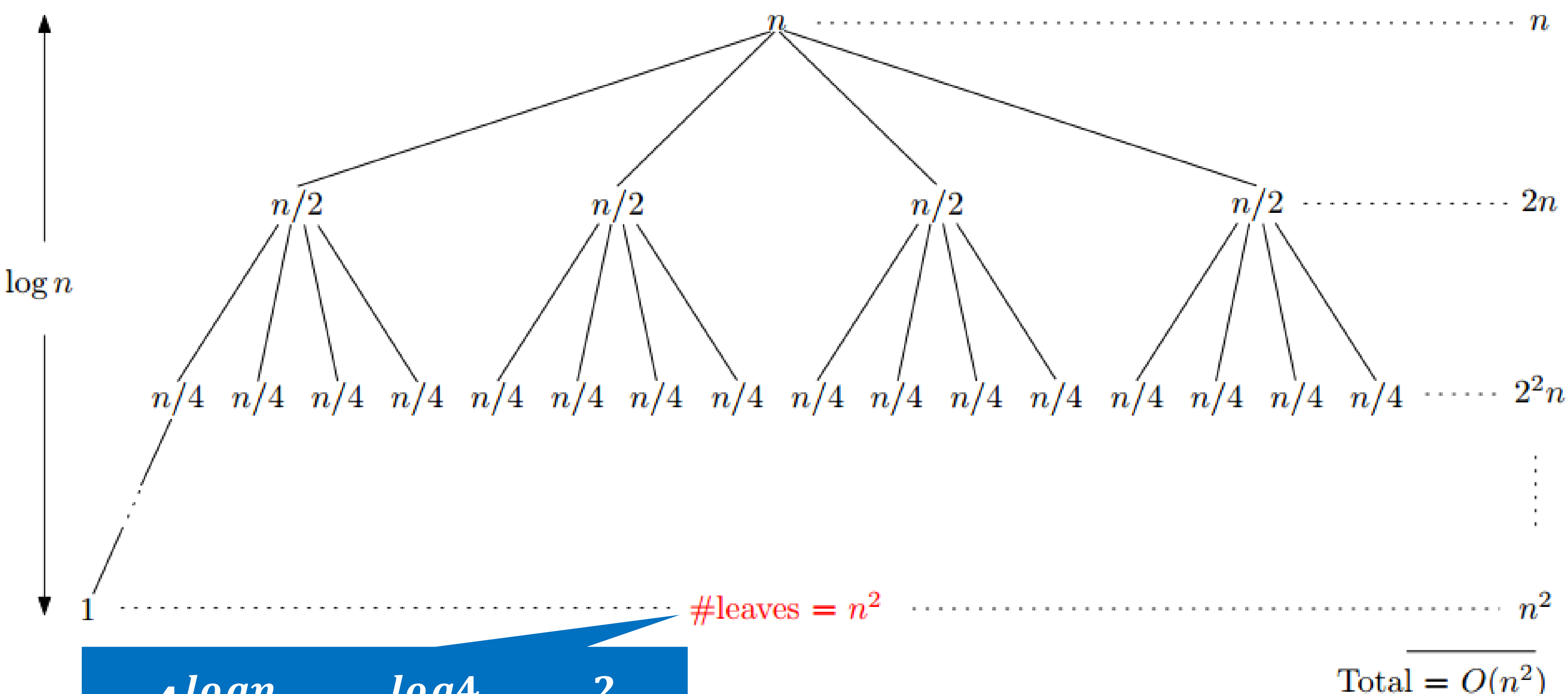
Assume that n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

Analysis of Running Time

Assume that n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

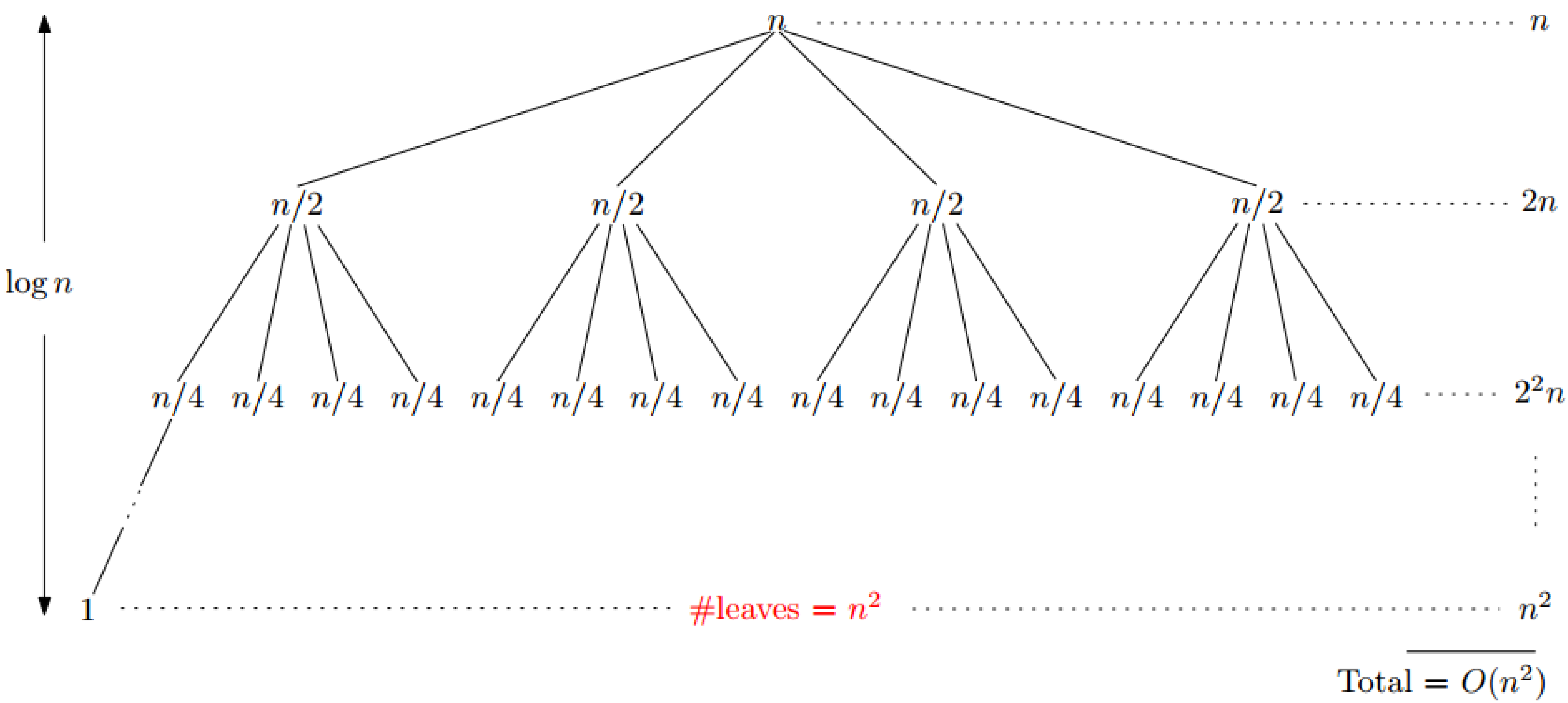


$$4^{\log n} = n^{\log 4} = n^2$$

Analysis of Running Time

Assume that n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Same order as the brute force approach! No improvement!

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - The first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Two Observations

Observation 1:

What we really need are the following 3 terms:

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1?$$

Instead of the following 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_1?$$

Two Observations

Observation 1:

What we really need are the following 3 terms:

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1?$$

Instead of the following 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_1?$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

Two Observations

Observation 1:

What we really need are the following 3 terms:

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1?$$

Instead of the following 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_1?$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0B_0 \\ Z &= A_1B_1 \end{aligned}$$

Two Observations

Observation 1:

What we really need are the following 3 terms:

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1?$$

Instead of the following 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_1?$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0B_0 \\ Z &= A_1B_1 \end{aligned}$$

- *U and Z are what we originally wanted*

Two Observations

Observation 1:

What we really need are the following 3 terms:

$$A_0B_0,\ A_0B_1 + A_1B_0,\ A_1B_1?$$

Instead of the following 4 terms:

$$A_0B_0,\ A_0B_1,\ A_1B_0,\ A_1B_1?$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0B_0 \\ Z &= A_1B_1 \end{aligned}$$

- *U and Z are what we originally wanted*
- $A_0B_1 + A_1B_0 = Y - U - Z$

The improved Divide-and-Conquer Algorithm

$\text{PolyMulti2}(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{n-\frac{n}{2}};$
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1};$
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{n-\frac{n}{2}};$

The improved Divide-and-Conquer Algorithm

PolyMulti2$(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{n-\frac{n}{2}}$;
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{n-\frac{n}{2}}$;
$Y(x) \leftarrow$PolyMulti2$(A_0(x) + A_1(x), B_0(x) + B_1(x))$;$//T(n/2)$
$U(x) \leftarrow$PolyMulti2$(A_0(x), B_0(x))$;$//T(n/2)$
$Z(x) \leftarrow$PolyMulti2$(A_1(x), B_1(x))$;$//T(n/2)$

The improved Divide-and-Conquer Algorithm

PolyMulti2$(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{n-\frac{n}{2}}$;
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{n-\frac{n}{2}}$;
$Y(x) \leftarrow$PolyMulti2$(A_0(x) + A_1(x), B_0(x) + B_1(x))$;$//T(n/2)$
$U(x) \leftarrow$PolyMulti2$(A_0(x), B_0(x))$;$//T(n/2)$
$Z(x) \leftarrow$PolyMulti2$(A_1(x), B_1(x))$;$//T(n/2)$
return $(U(x) + [Y(x) - U(x) - Z(x)]x^{\frac{n}{2}} + Z(x)x^{2\frac{n}{2}})$;$//O(n)$

The improved Divide-and-Conquer Algorithm

PolyMulti2$(A(x), B(x))$

Input: $A(x), B(x)$
Output: $A(x) \times B(x)$
$A_0(x) \leftarrow a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$A_1(x) \leftarrow a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{n-\frac{n}{2}}$;
$B_0(x) \leftarrow b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;
$B_1(x) \leftarrow b_{\frac{n}{2}} + b_{\frac{n}{2}+1} x + \cdots + b_n x^{n-\frac{n}{2}}$;
$Y(x) \leftarrow$PolyMulti2$(A_0(x) + A_1(x), B_0(x) + B_1(x))$;$//T(n/2)$
$U(x) \leftarrow$PolyMulti2$(A_0(x), B_0(x))$;$//T(n/2)$
$Z(x) \leftarrow$PolyMulti2$(A_1(x), B_1(x))$;$//T(n/2)$
return $(U(x) + [Y(x) - U(x) - Z(x)]x^{\frac{n}{2}} + Z(x)x^{2\frac{n}{2}})$;$//O(n)$

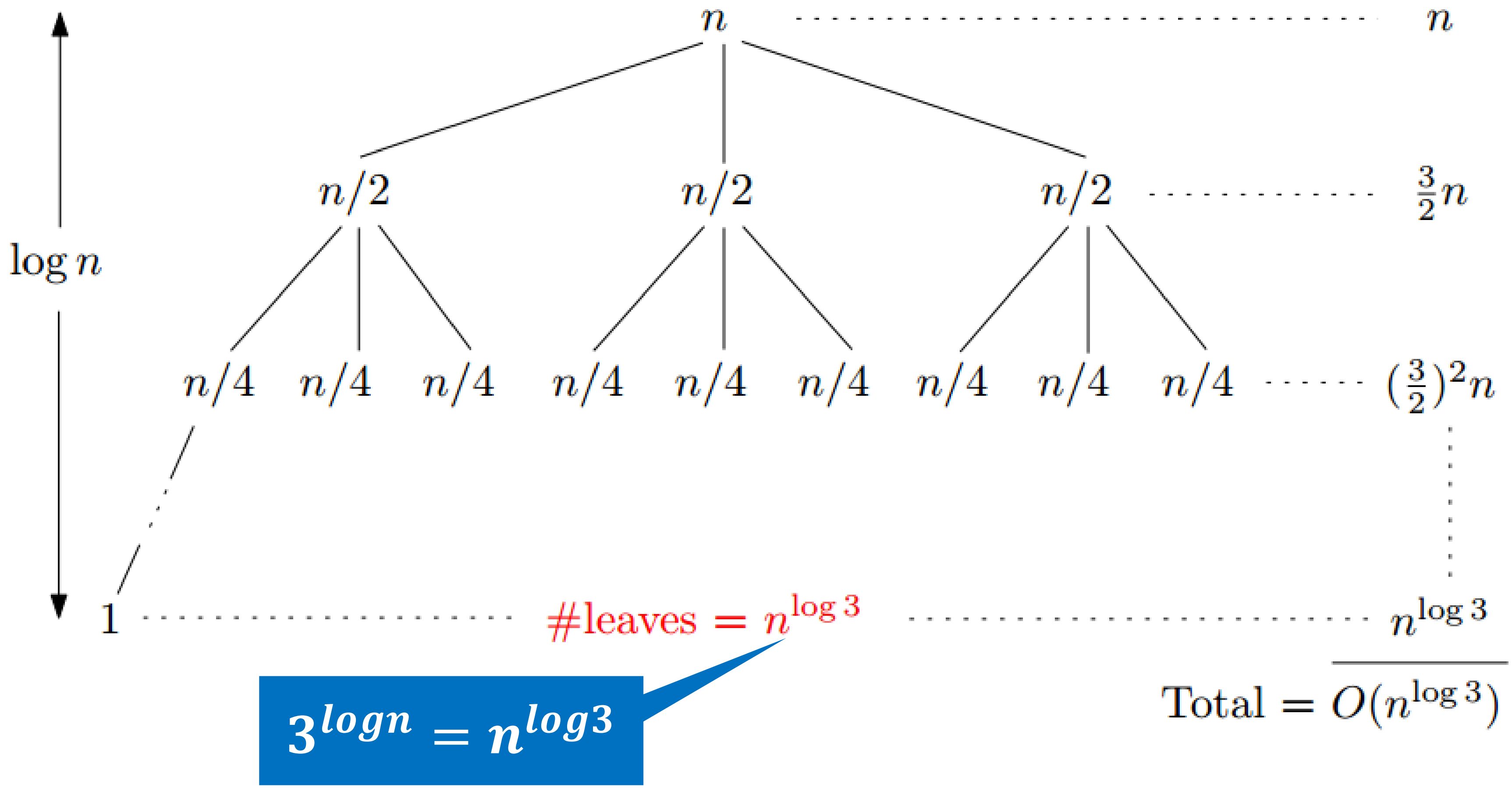
$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

Running Time of the Improved Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

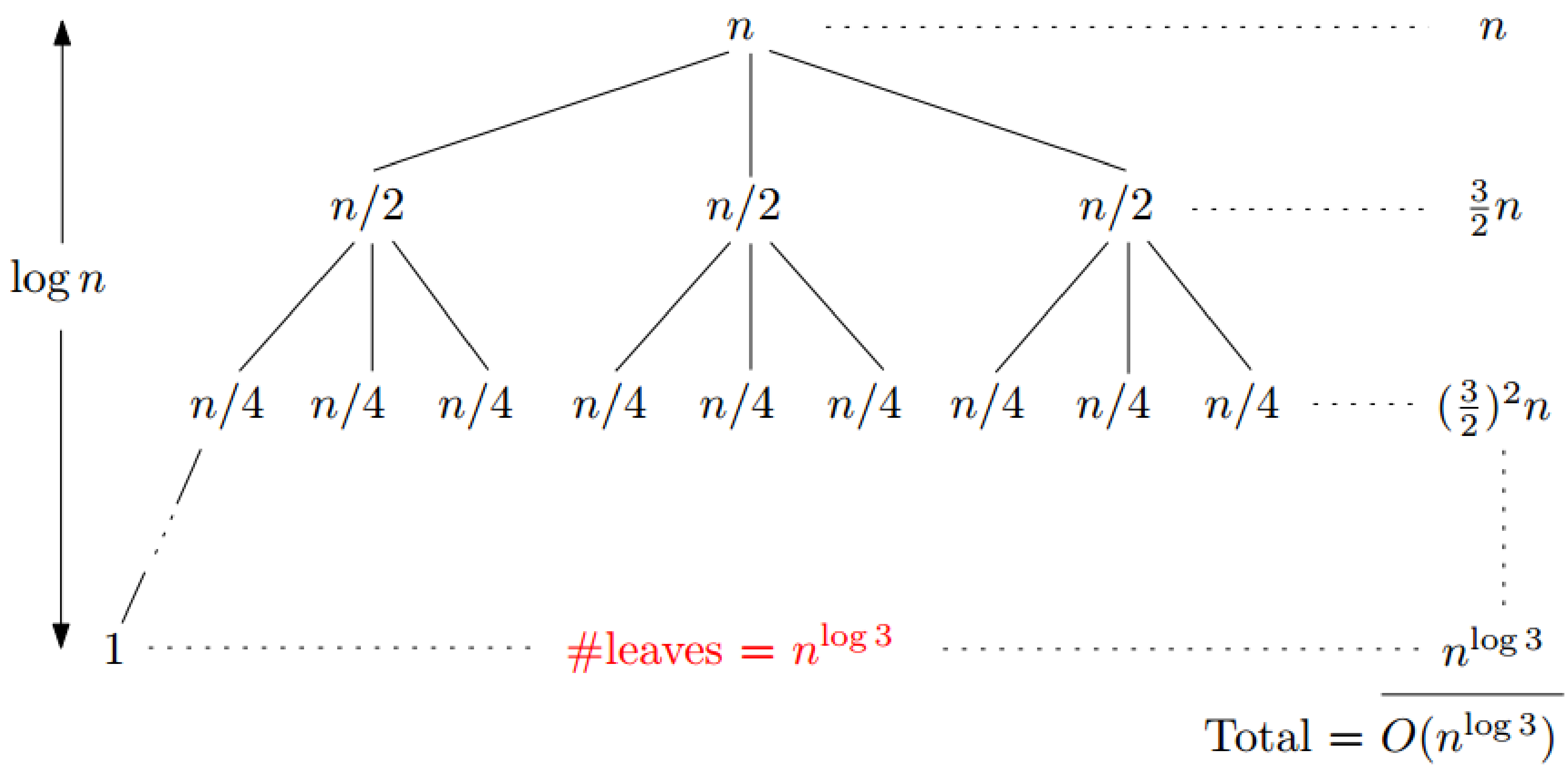
Running Time of the Improved Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



Running Time of the Improved Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$



The second method is much better!

Outline

- Review to Divide-and-Conquer Paradigm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Polynomial Multiplication Problem
  - Problem definition
  - A brute force algorithm
  - A first divide-and-conquer algorithm
  - An improved divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Analysis of the D&C algorithm

- The divide-and-conquer approach does not always give you the best solution
  - Our original algorithm was just as bad as brute force

Analysis of the D&C algorithm

- The divide-and-conquer approach does not always give you the best solution
  - Our original algorithm was just as bad as brute force
- There is actually an $O(n \log n)$ solution to the polynomial multiplication problem

Analysis of the D&C algorithm

- The divide-and-conquer approach does not always give you the best solution
  - Our original algorithm was just as bad as brute force
- There is actually an O(n log n) solution to the polynomial multiplication problem
  - It involves using the Fast Fourier Transform algorithm as a subroutine
  - The FFT is another classic divide-and-conquer algorithm(check Chapt 30 in CLRS if interested)

Analysis of the D&C algorithm

- The divide-and-conquer approach does not always give you the best solution
  - Our original algorithm was just as bad as brute force
- There is actually an O(n log n) solution to the polynomial multiplication problem
  - It involves using the Fast Fourier Transform algorithm as a subroutine
  - The FFT is another classic divide-and-conquer algorithm(check Chapt 30 in CLRS if interested)
- The idea of using 3 multiplications instead of 4 is used in large-integer multiplications
  - A similar idea is the basis of the classic Strassen matrix multiplication algorithm (CLRS 4.2)

谢谢

BDA
Beihang University