Program Design and Algorithms

Part I: Divide and Conquer

Lecture 3: Maximum Contiguous Subarray Problem and Counting Inversion Problem

Yongxin Tong (童咏昕)

School of CSE, Beihang University

yxtong@buaa.edu.cn

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Introduction to Part I

- Divide-and-conquer (D&C) is an important algorithm design paradigm.

Introduction to Part I

- 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)

Introduction to Part I

- 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)

Introduction to Part I

- 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

Introduction to Part I

- 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)
  - Integer Multiplication (整数乘法)
  - Polynomial Multiplication (多项式乘法)
  - QuickSort and Partition (快速排序与划分)
  - Deterministic and Randomized Selection (确定性与随机化选择)

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Maximum Contiguous Subarray (MCS) Problem

ACME Corp[1] – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

[1] A Company that Makes Everything

Maximum Contiguous Subarray (MCS) Problem

ACME Corp[1] – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

Between years 1 and 9:

- ACME earned $-3+2+1-4+5+2-1+3-1=4$ M$

[1] A Company that Makes Everything

Maximum Contiguous Subarray (MCS) Problem

ACME Corp[1] – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

Between years 1 and 9:

- ACME earned $-3 + 2 + 1 - 4 + 5 + 2 - 1 + 3 - 1 = 4$ M$

Between years 2 and 6:

- ACME earned $2 + 1 - 4 + 5 + 2 = 6$ M$

[1] A Company that Makes Everything

Maximum Contiguous Subarray (MCS) Problem

ACME Corp[1] – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

Between years 1 and 9:

- ACME earned $-3+2+1-4+5+2-1+3-1=4$ M$

Between years 2 and 6:

- ACME earned $2+1-4+5+2=6$ M$

Between years 5 and 8:

- ACME earned $5+2-1+3=9$ M$

[1] A Company that Makes Everything

Maximum Contiguous Subarray (MCS) Problem

ACME Corp[1] – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

Between years 1 and 9:

- ACME earned $-3+2+1-4+5+2-1+3-1=4$ M$

Between years 2 and 6:

- ACME earned $2+1-4+5+2=6$ M$

Between years 5 and 8:

- ACME earned $5+2-1+3=9$ M$

Problem: Find the span of years in which ACME earned the most

[1] A Company that Makes Everything

Maximum Contiguous Subarray (MCS) Problem

ACME Corp[1] – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

Between years 1 and 9:

- ACME earned $-3+2+1-4+5+2-1+3-1=4$ M$

Between years 2 and 6:

- ACME earned $2+1-4+5+2=6$ M$

Between years 5 and 8:

- ACME earned $5+2-1+3=9$ M$

如果所有数组元素都是非负数，整个数组和肯定是最大

Problem: Find the span of years in which ACME earned the most

Answer: Year 5-8, 9 M$

[1] A Company that Makes Everything

Formal Definition

- Input: An array of reals A[1…n]

Formal Definition

- Input: An array of reals $A[1\ldots n]$
- The value of subarray $A[i\ldots j]$ is

$$V(i,j) = \sum_{x=i}^{j} A(x)$$

Formal Definition

- Input: An array of reals *A*[1…n]
- The value of subarray *A*[i…j] is

$$V(i,j) = \sum_{x=i}^{j} A(x)$$

Definition (Maximum Contiguous Subarray Problem)

Find $i \leq j$ such that $V(i,j)$ is maximized.

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

A Brute Force Algorithm

Calculate the value of $V(i, j)$ for each pair $i \leq j$ and return the maximum value

A Brute Force Algorithm

Calculate the value of $V(i, j)$ for each pair $i \leq j$ and return the maximum value

VMAX $\leftarrow A[1]$;

A Brute Force Algorithm

Calculate the value of $V(i,j)$ for each pair $i \leq j$ and return the maximum value

```
VMAX ← A[1];
for i ← 1 to n do
    for j ← i to n do
        // calculate V(i,j)
        V ← 0;
        for x ← i to j do
            V ← V + A[x];
        end
```

A Brute Force Algorithm

Calculate the value of $V(i,j)$ for each pair $i \leq j$ and return the maximum value

```
VMAX ← A[1];
for i ← 1 to n do
    for j ← i to n do
        // calculate V(i,j)
        V ← 0;
        for x ← i to j do
            V ← V + A[x];
        end
        if V > VMAX then
            VMAX ← V;
        end
    end
end
return VMAX
```

A Brute Force Algorithm

Calculate the value of $V(i,j)$ for each pair $i \leq j$ and return the maximum value

```
VMAX ← A[1];
for i ← 1 to n do
    for j ← i to n do
        // calculate V(i,j)
        V ← 0;
        for x ← i to j do
            V ← V + A[x];
        end
        if V > VMAX then
            VMAX ← V;
        end
    end
end
return VMAX
```

$O(n^3)$ arithmetic additions

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

A Data-Reuse Algorithm

Idea:

- don't need to calculate each V(*i, j*) from scratch

A Data-Reuse Algorithm

Idea:

- don't need to calculate each V(*i, j*) from scratch
- exploit the fact: $V(i,j) = \sum_{x=i}^{j} A[x] = V(i,j-1) + A[j]$

A Data-Reuse Algorithm

Idea:

- don't need to calculate each V(*i, j*) from scratch
- exploit the fact: $V(i,j) = \sum_{x=i}^{j} A[x] = V(i,j-1) + A[j]$

```
VMAX ← A[1];
for i ← 1 to n do
    V ← 0;
    for j ← i to n do
        // calculate V(i,j)
        V ← V + A[j];
```

A Data-Reuse Algorithm

Idea:

- don't need to calculate each V(*i, j*) from scratch
- exploit the fact: $V(i,j) = \sum_{x=i}^{j} A[x] = V(i,j-1) + A[j]$

```
VMAX ← A[1];
for i ← 1 to n do
    V ← 0;
    for j ← i to n do
        // calculate V(i,j)
        V ← V + A[j];
        if V > VMAX then
            VMAX ← V;
        end
    end
end
return VMAX
```

A Data-Reuse Algorithm

Idea:

- don't need to calculate each V(*i, j*) from scratch
- exploit the fact: $V(i,j) = \sum_{x=i}^{j} A[x] = V(i,j-1) + A[j]$

```
VMAX ← A[1];
for i ← 1 to n do
    V ← 0;
    for j ← i to n do
        // calculate V(i,j)
        V ← V + A[j];
        if V > VMAX then
            VMAX ← V;
        end
    end
end
return VMAX
```

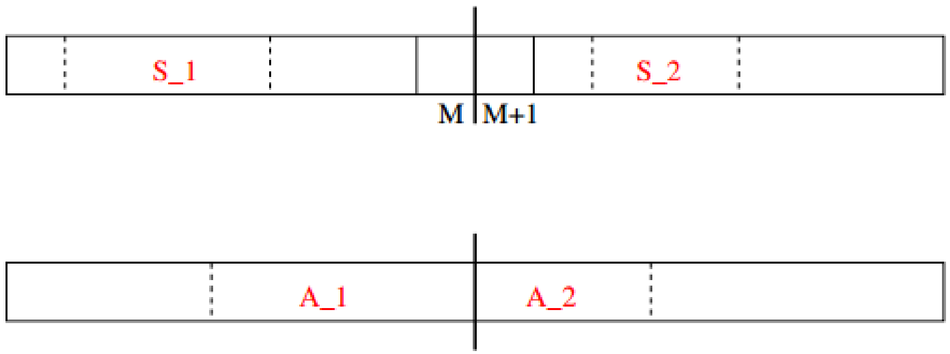
$O(n^2)$ arithmetic additions

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

A Divide-and-Conquer Algorithm

Set $m = \lfloor (n+1)/2 \rfloor$

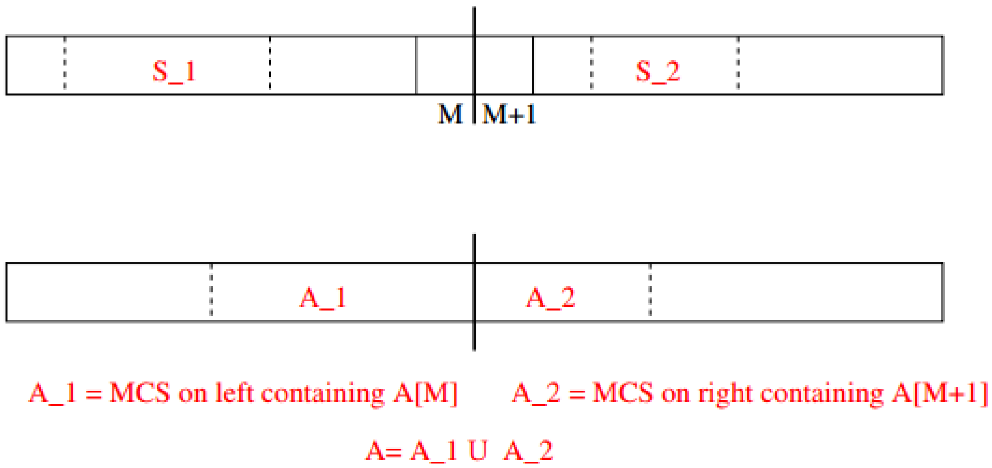


A_1 = MCS on left containing A[M] A_2 = MCS on right containing A[M+1]

A= A_1 U A_2

A Divide-and-Conquer Algorithm

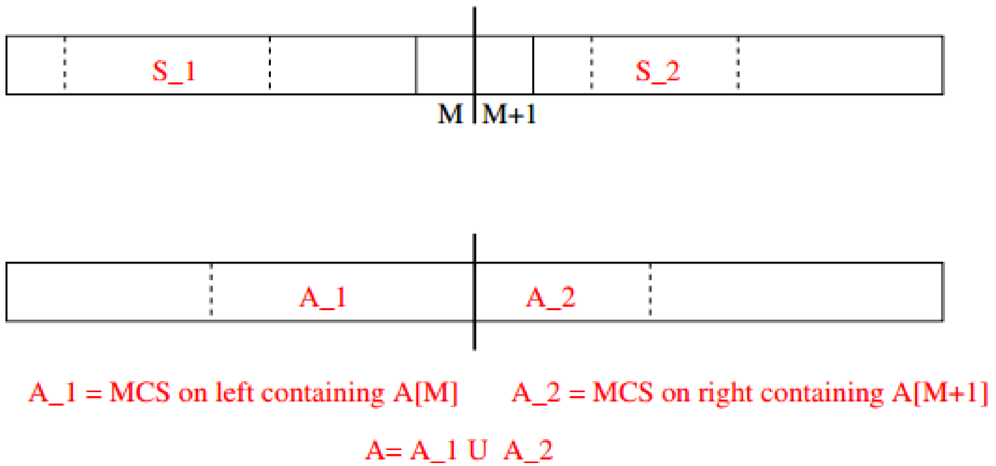
Set $m = \lfloor (n+1)/2 \rfloor$



The MCS S must be one of

A Divide-and-Conquer Algorithm

Set $m = \lfloor (n+1)/2 \rfloor$

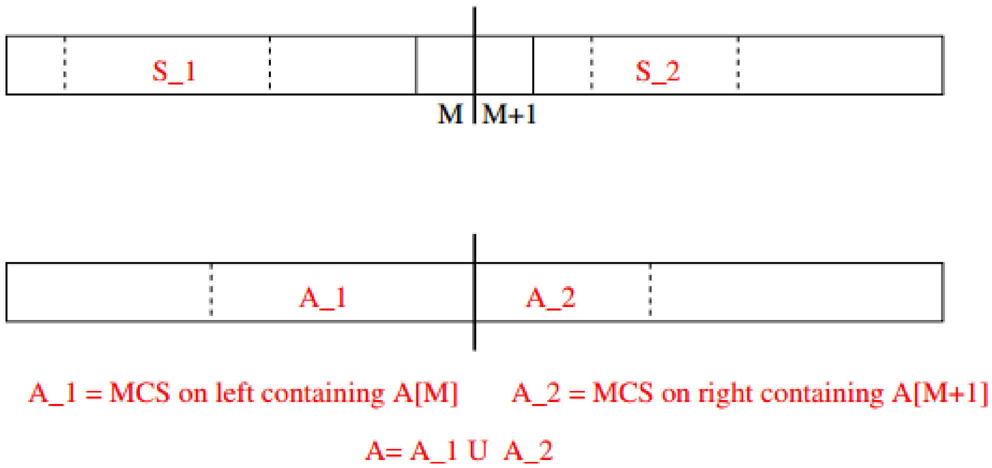


The MCS S must be one of

1. S_1: the MCS in $A[1 \ldots m]$

A Divide-and-Conquer Algorithm

Set $m = \lfloor (n+1)/2 \rfloor$

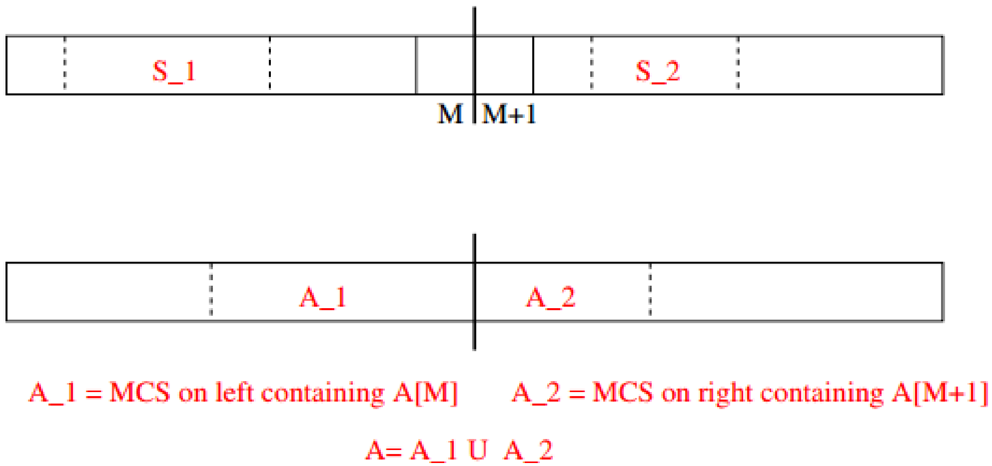


The MCS S must be one of

1. S_1: the MCS in $A[1 \ldots m]$
2. S_2: the MCS in $A[m+1 \ldots n]$

A Divide-and-Conquer Algorithm

Set $m = \lfloor (n+1)/2 \rfloor$

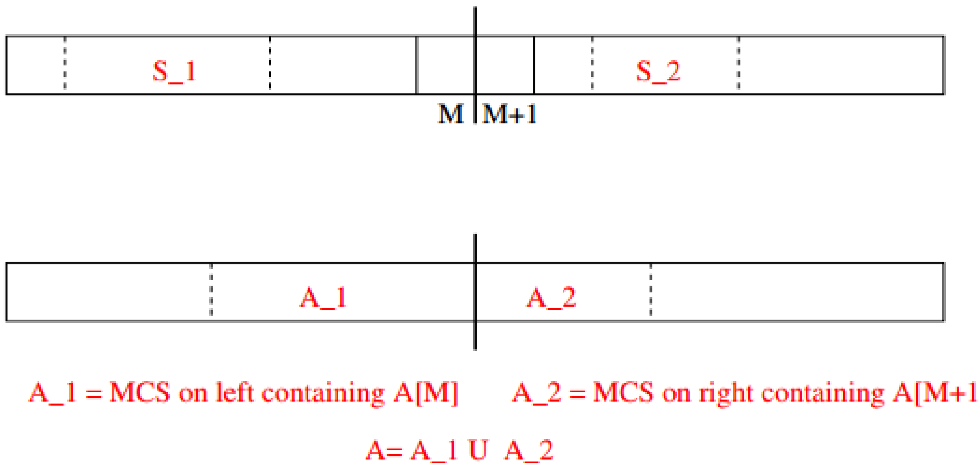


The MCS S must be one of

1. S_1: the MCS in $A[1 \ldots m]$
2. S_2: the MCS in $A[m+1 \ldots n]$
3. A: the MCS across the cut.

A Divide-and-Conquer Algorithm

Set $m = \lfloor (n+1)/2 \rfloor$



The MCS S must be one of

1. S_1: the MCS in $A[1 \dots m]$
2. S_2: the MCS in $A[m+1 \dots n]$
3. A: the MCS across the cut.

So,

最终，在S_1, S_2和A(跨越中点的最大子数组)这三种情况中选取和最大者

$$S = \text{the best among } \{S_1, S_2, A\}$$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 =$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 =$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	‖	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$

1	-5	4	2	-7	3	6	-1	‖	2	-4	7	-10	2	6	1	-3

- $A_1 =$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $A_1 = [3, 6, -1]$ and $A_2 =$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $A_1 = [3, 6, -1]$ and $A_2 = [2, -4, 7]$
- $A = A_1 \cup A_2 =$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $A_1 = [3, 6, -1]$ and $A_2 = [2, -4, 7]$
- $A = A_1 \cup A_2 = [3, 6, -1, 2, -4, 7]$

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $A_1 = [3, 6, -1]$ and $A_2 = [2, -4, 7]$
- $A = A_1 \cup A_2 = [3, 6, -1, 2, -4, 7]$

- $Value(S_1) = 9;\ Value(S_2) = 9;\ Value(A) = 13$
- solution:

An Example of Divide-and-Conquer Algorithm

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

- $S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$

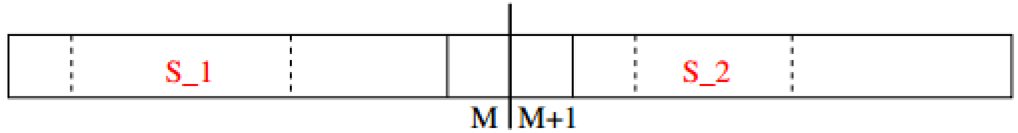
1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

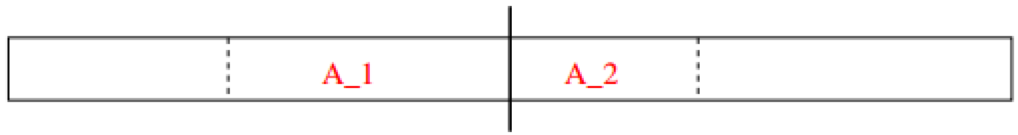
- $A_1 = [3, 6, -1]$ and $A_2 = [2, -4, 7]$
- $A = A_1 \cup A_2 = [3, 6, -1, 2, -4, 7]$

- $Value(S_1) = 9;\ Value(S_2) = 9;\ Value(A) = 13$
- solution: A

Divide: MCS across The Cut

Set $m = \lfloor (n+1)/2 \rfloor$



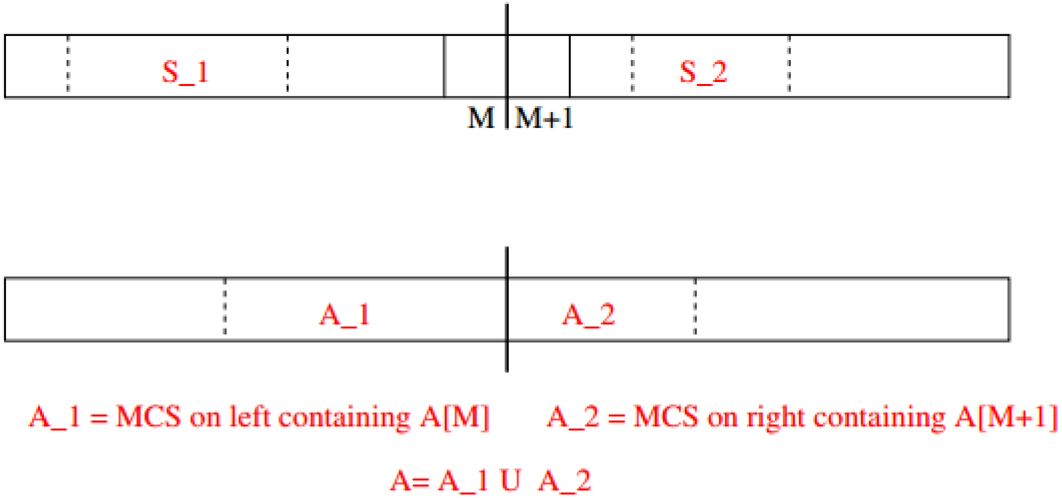


A_1 = MCS on left containing A[M]   A_2 = MCS on right containing A[M+1]

A= A_1 U A_2

Divide: MCS across The Cut

Set $m = \lfloor (n+1)/2 \rfloor$

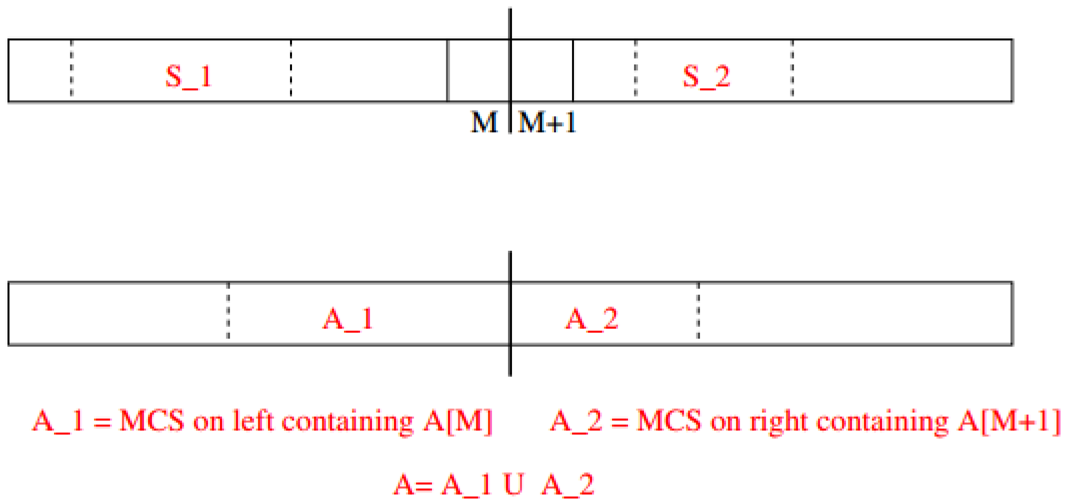


$A = A_1 \cup A_2$

- A_1: MCS among contiguous subarrays ending at A[m]

Divide: MCS across The Cut

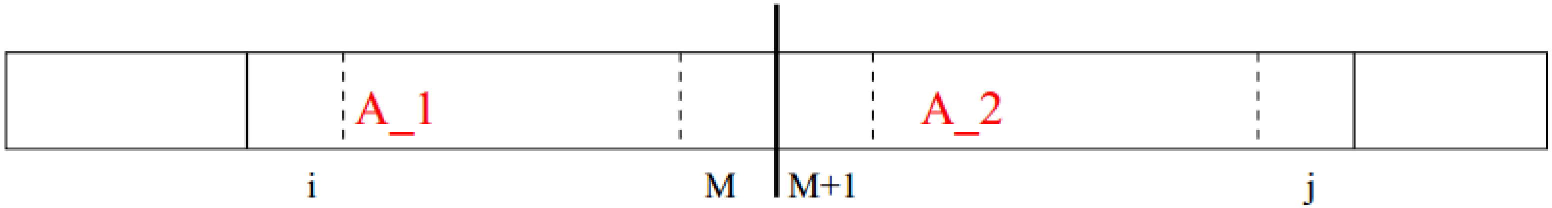
Set $m = \lfloor (n+1)/2 \rfloor$



$A = A_1 \cup A_2$

- A_1: MCS among contiguous subarrays ending at A[m]
- A_2: MCS among contiguous subarrays starting at A[m+1]

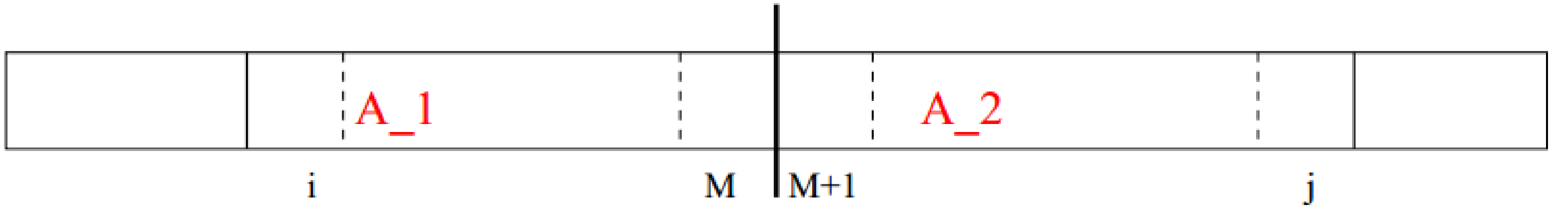
Conquer: Finding the "A_1" Subarrays



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

MAX $\leftarrow A[m]$;
SUM $\leftarrow A[m]$;

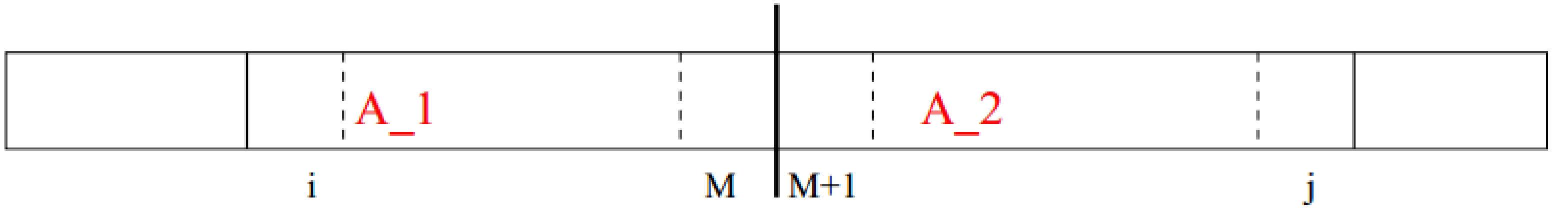
Conquer: Finding the "A_1" Subarrays



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 $+A[i]$;

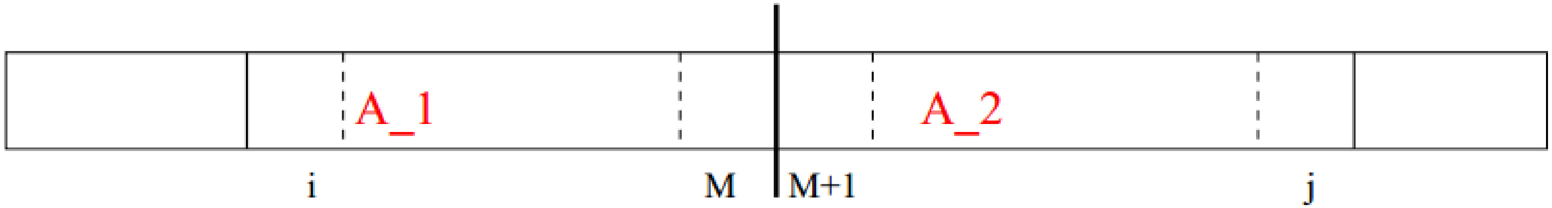
Conquer: Finding the "A_1" Subarrays



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

Conquer: Finding the "A_1" Subarrays



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;

Conquer: Finding *"A"* with A Linear Time

- There are only m sequences of the form

Conquer: Finding "*A*" with A Linear Time

- There are only *m* sequences of the form
  - A_1 can be found in $O(m)$ time

Conquer: Finding "*A*" with A Linear Time

- There are only *m* sequences of the form
  - A_1 can be found in $O(m)$ time
- Similarly, A_2 is in the form $A[m+1...j]$

Conquer: Finding "A" with A Linear Time

- There are only m sequences of the form
  - A_1 can be found in $O(m)$ time

- Similarly, A_2 is in the form $A[m+1 \ldots j]$
  - there are only $n-m$ such sequences

Conquer: Finding "*A*" with A Linear Time

- There are only *m* sequences of the form
  - A_1 can be found in $O(m)$ time

- Similarly, A_2 is in the form $A[m+1 \ldots j]$
  - there are only $n-m$ such sequences
  - A_2 can be found in $O(n-m)$ time

Conquer: Finding "A" with A Linear Time

- There are only m sequences of the form
  - A_1 can be found in $O(m)$ time

- Similarly, A_2 is in the form $A[m+1 \ldots j]$
  - there are only $n-m$ such sequences
  - A_2 can be found in $O(n-m)$ time

- $A = A_1 \cup A_2$ can be found in $O(n)$ time

Conquer: Finding "*A*" with A Linear Time

- There are only m sequences of the form
  - A_1 can be found in $O(m)$ time

- Similarly, A_2 is in the form $A[m+1 \ldots j]$
  - there are only $n-m$ such sequences
  - A_2 can be found in $O(n-m)$ time

- $A = A_1 \cup A_2$ can be found in $O(n)$ time
  - linear to the input size

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \ldots t]$ with $s \leq t$
Output: MCS of $A[s \ldots t]$

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \dots t]$ with $s \le t$
Output: MCS of $A[s \dots t]$
begin
    if $s = t$ **then return** $A[s]$;

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \ldots t]$ with $s \leq t$
Output: MCS of $A[s \ldots t]$
begin
  if $s = t$ **then return** $A[s]$;

  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \ldots t]$ with $s \leq t$
Output: MCS of $A[s \ldots t]$
begin
  if $s = t$ **then return** $A[s]$;
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;
    Find $MCS(A, m+1, t)$;

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \ldots t]$ with $s \leq t$
Output: MCS of $A[s \ldots t]$
begin
  if $s = t$ **then return** $A[s]$;
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;
    Find $MCS(A, m+1, t)$;
    Find MCS that contains both $A[m]$ and $A[m+1]$;

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \dots t]$ with $s \le t$
Output: MCS of $A[s \dots t]$
begin
  if $s = t$ **then return** $A[s]$;

  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;
    Find $MCS(A, m+1, t)$;
    Find MCS that contains both $A[m]$ and $A[m+1]$;
    return maximum of the three sequences found
  end

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \ldots t]$ with $s \leq t$
Output: MCS of $A[s \ldots t]$
begin
  if $s = t$ **then return** $A[s]$;
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;
    Find $MCS(A, m+1, t)$;
    Find MCS that contains both $A[m]$ and $A[m+1]$;
    return maximum of the three sequences found
  end
end

The Complete Divide-and-Conquer Algorithm

$MCS(A, s, t)$

Input: $A[s \dots t]$ with $s \leq t$
Output: MCS of $A[s \dots t]$
begin
  if $s = t$ **then return** $A[s]$;

  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;
    Find $MCS(A, m+1, t)$;
    Find MCS that contains both $A[m]$ and $A[m+1]$;
    return maximum of the three sequences found
  end
end

First Call: *MCS*(*A, 1, n*)

A Full Illustration of the D&C Algorithm

6 -4 7 -4 0 1

A Full Illustration of the D&C Algorithm

6 -4 7 ¦ -4 0 1

A Full Illustration of the D&C Algorithm

6 -4 7 | -4 0 1

6 -4 7   **-4 0 1**

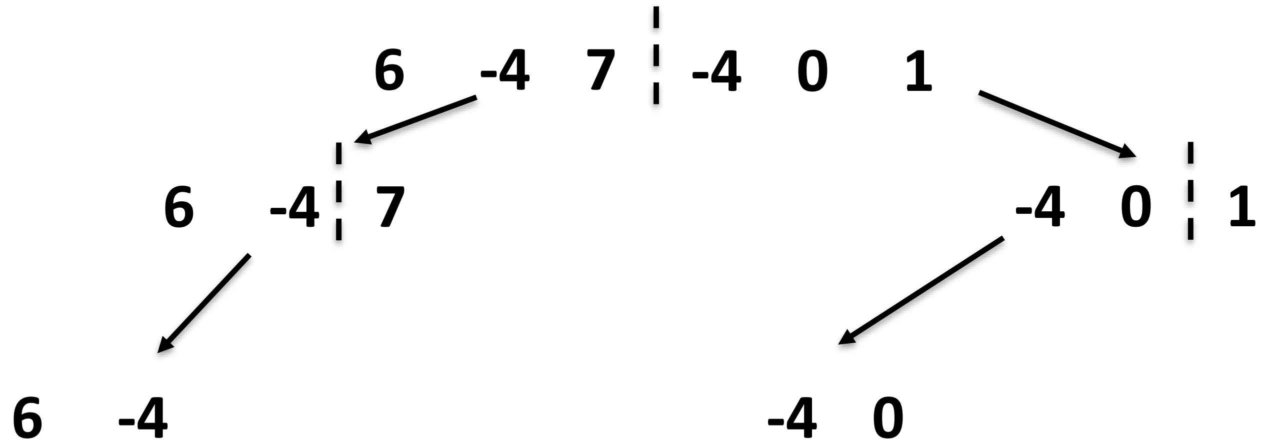
A Full Illustration of the D&C Algorithm

6 -4 7 ¦ -4 0 1

6 -4 ¦ 7

-4 0 ¦ 1

A Full Illustration of the D&C Algorithm



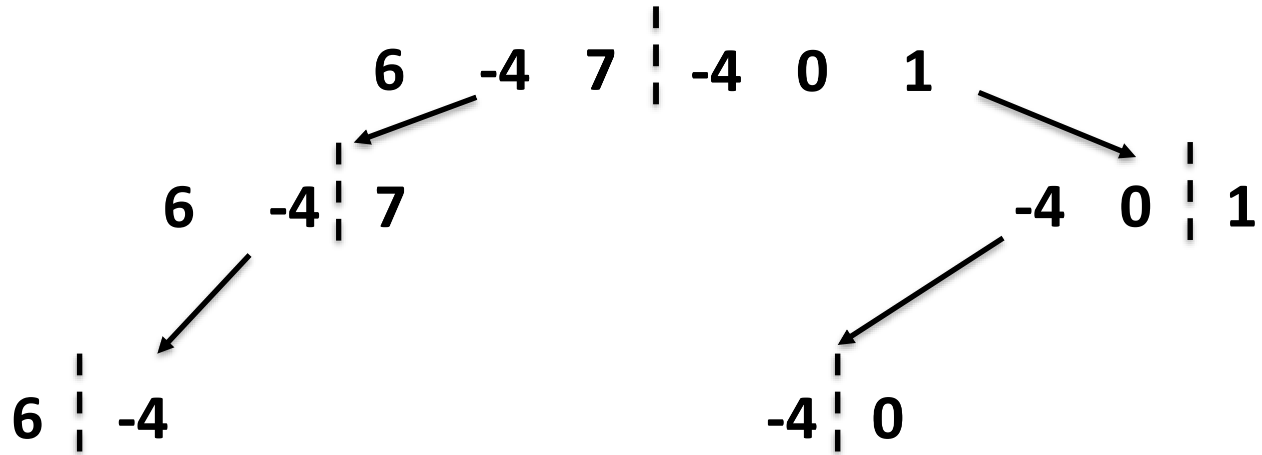
A Full Illustration of the D&C Algorithm

6 -4 7 | -4 0 1

6 -4 | 7 ; **-4 0 | 1**

6 | -4 ; **-4 | 0**

A Full Illustration of the D&C Algorithm



Divide

A Full Illustration of the D&C Algorithm

6 -4 7 ¦ -4 0 1

6 -4 ¦ 7

-4 0 ¦ 1

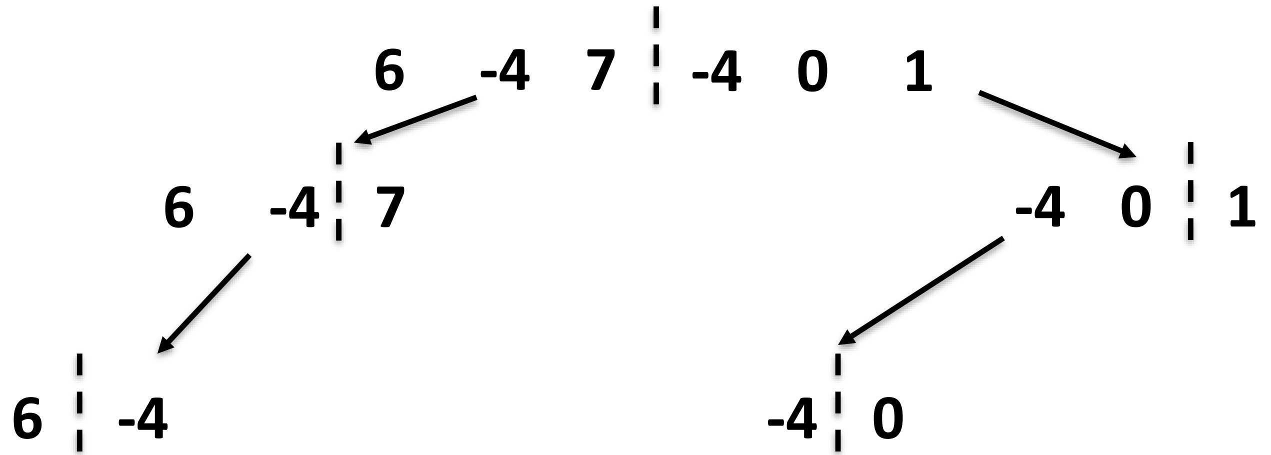
6 ¦ -4

-4 ¦ 0

Divide

Conquer

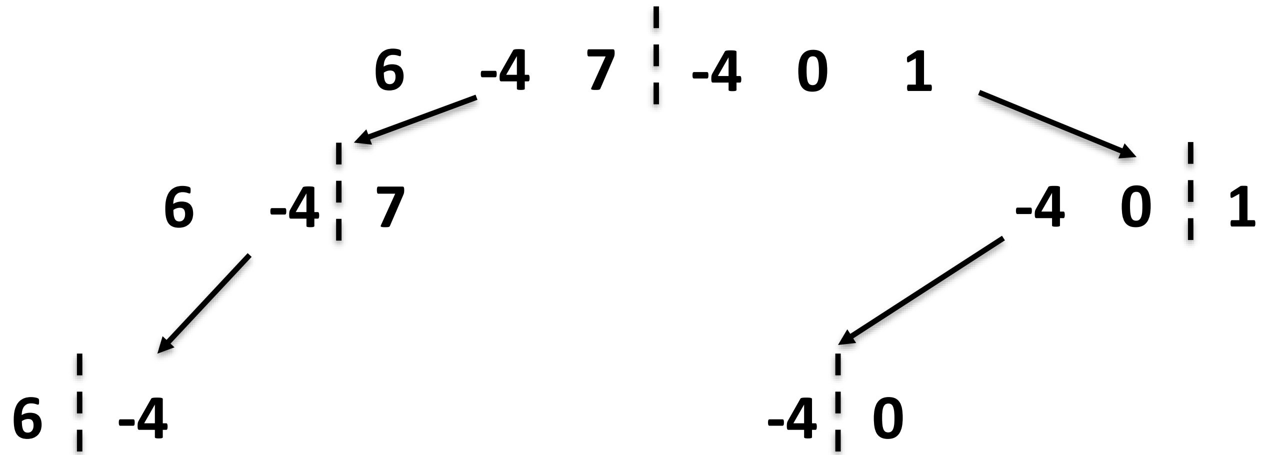
A Full Illustration of the D&C Algorithm



MCS={6} MCS={-4}

Conquer

A Full Illustration of the D&C Algorithm



Divide

Conquer

MCS={6}   MCS={-4}

A={6,-4}

Value(A)=2

A Full Illustration of the D&C Algorithm

6 -4 7 ¦ -4 0 1

6 -4 ¦ 7 **-4 0 ¦ 1**

6 ¦ -4 **-4 ¦ 0**

Divide

Conquer

MCS={6} **MCS={-4}**

A={6,-4}

Value(A)=2

MCS={6}

A Full Illustration of the D&C Algorithm

6 -4 7 ¦ -4 0 1

6 -4 ¦ 7

-4 0 ¦ 1

6 ¦ -4

-4 ¦ 0

Divide

Conquer

MCS={6} MCS={-4}

A={6,-4}

Value(A)=2

MCS={6} MCS={7}

A Full Illustration of the D&C Algorithm

6 -4 7 ¦ -4 0 1

6 -4 ¦ 7

-4 0 ¦ 1

6 ¦ -4

-4 ¦ 0

Divide

Conquer

MCS={6} **MCS={-4}**

A={6,-4}

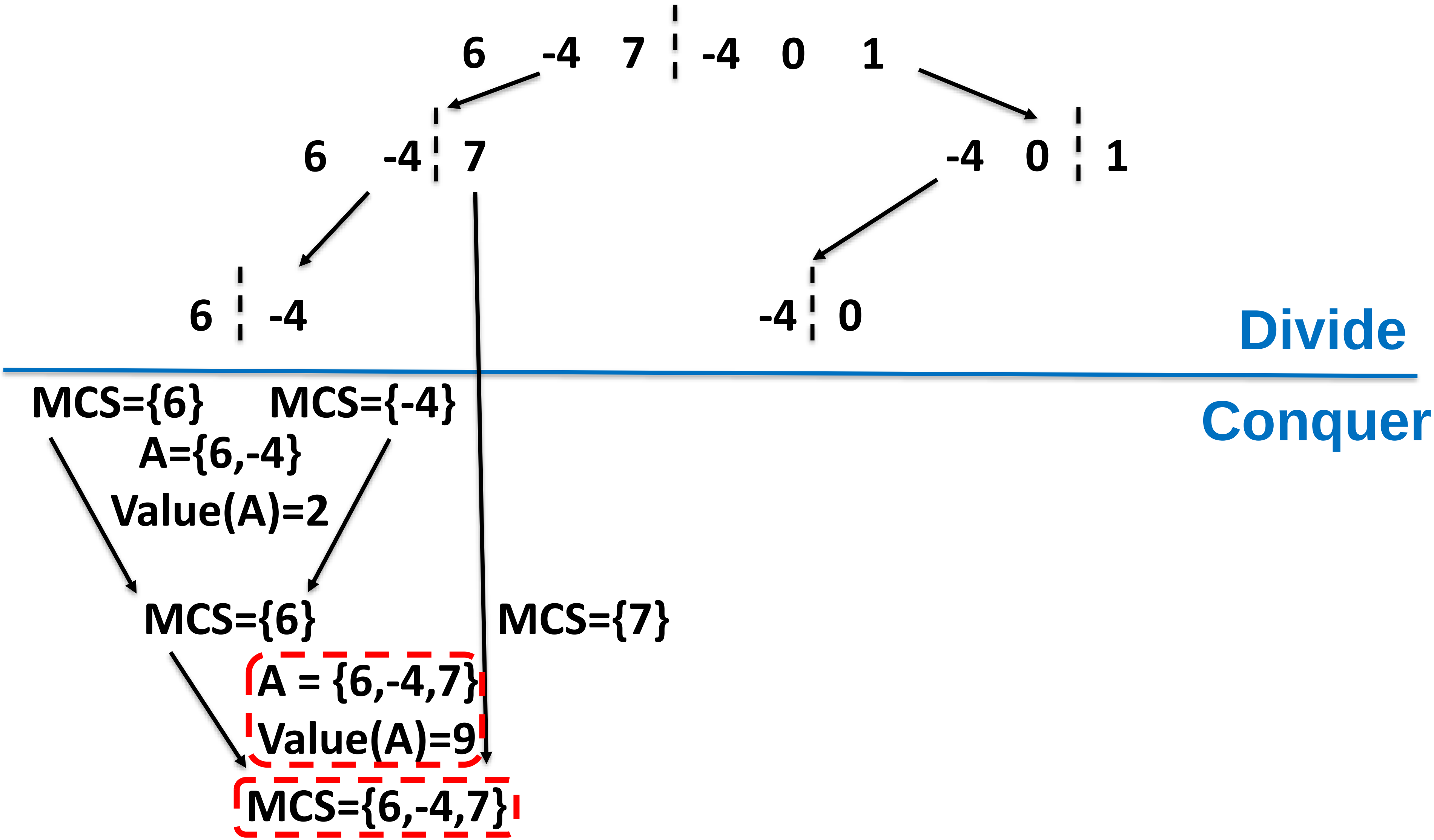
Value(A)=2

MCS={6} **MCS={7}**

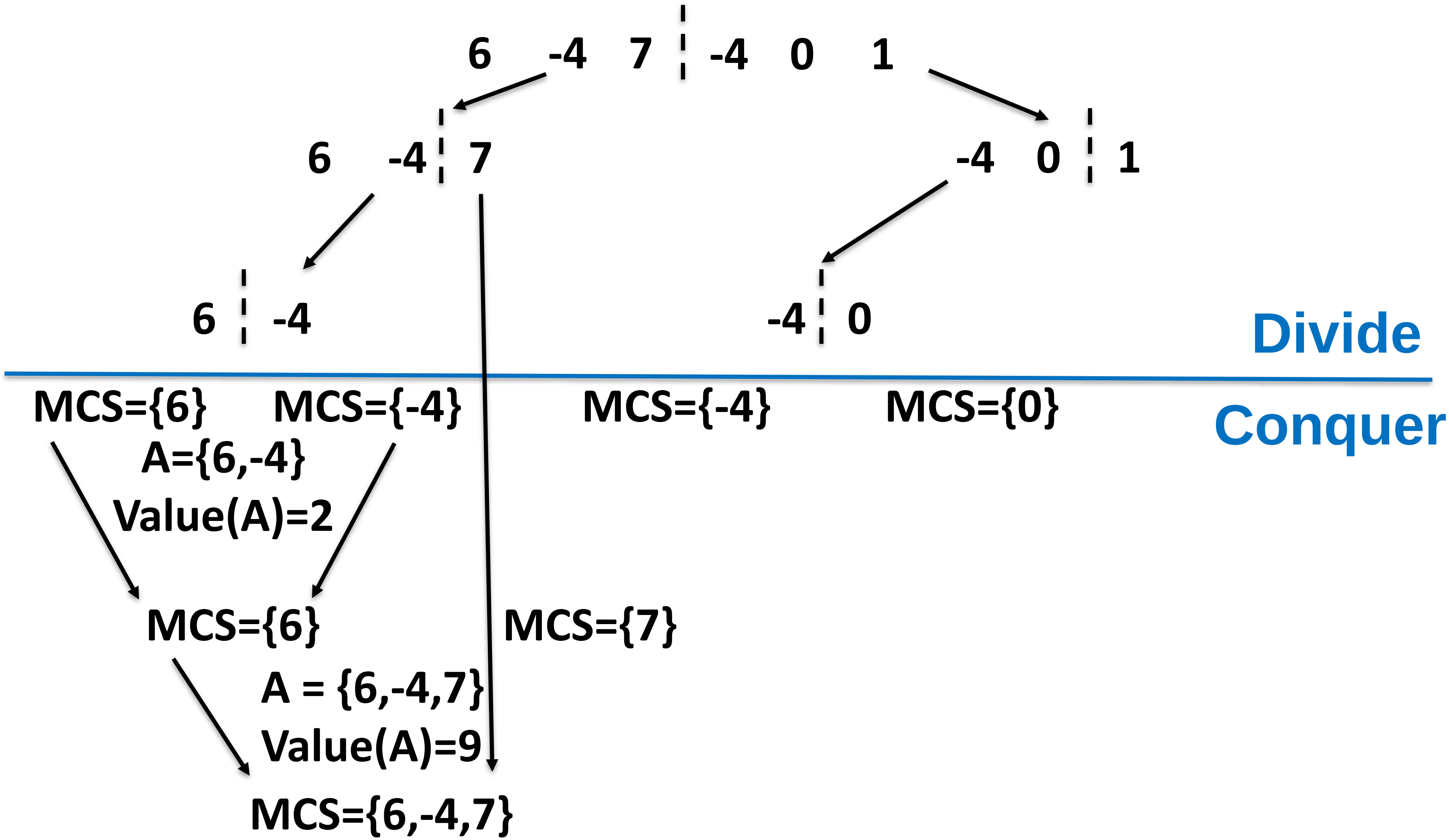
A = {6,-4,7}

Value(A)=9

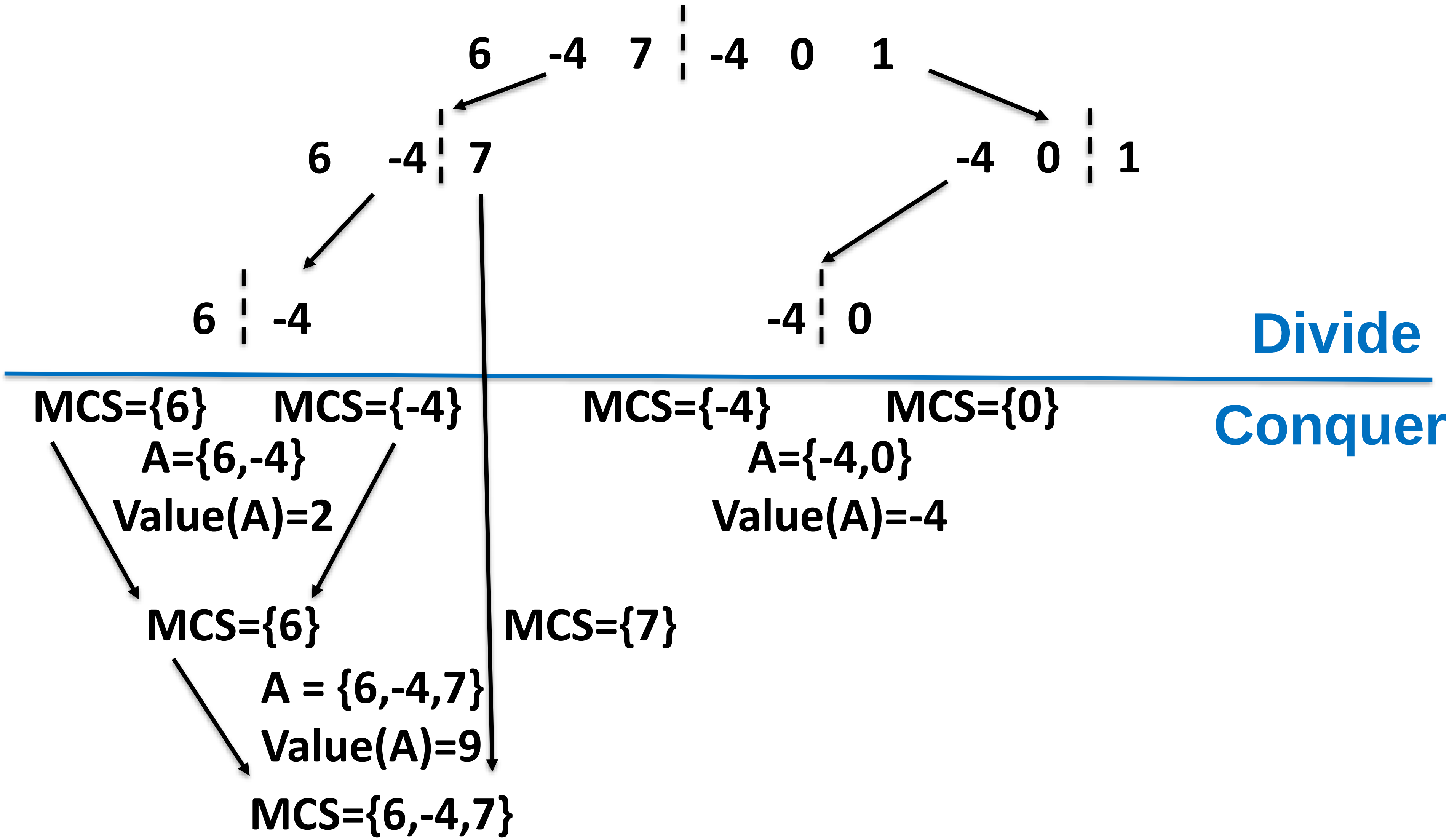
A Full Illustration of the D&C Algorithm



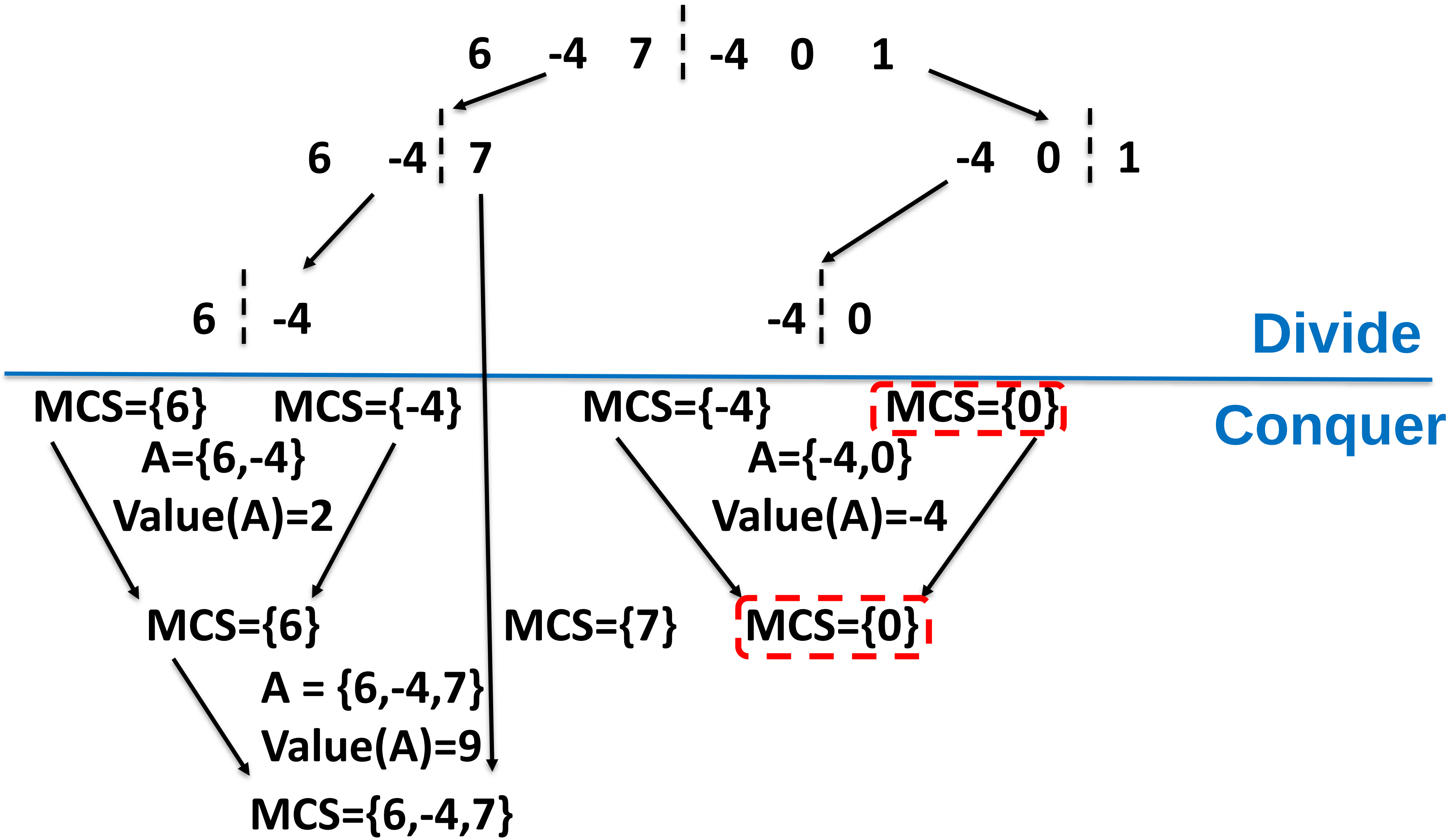
A Full Illustration of the D&C Algorithm



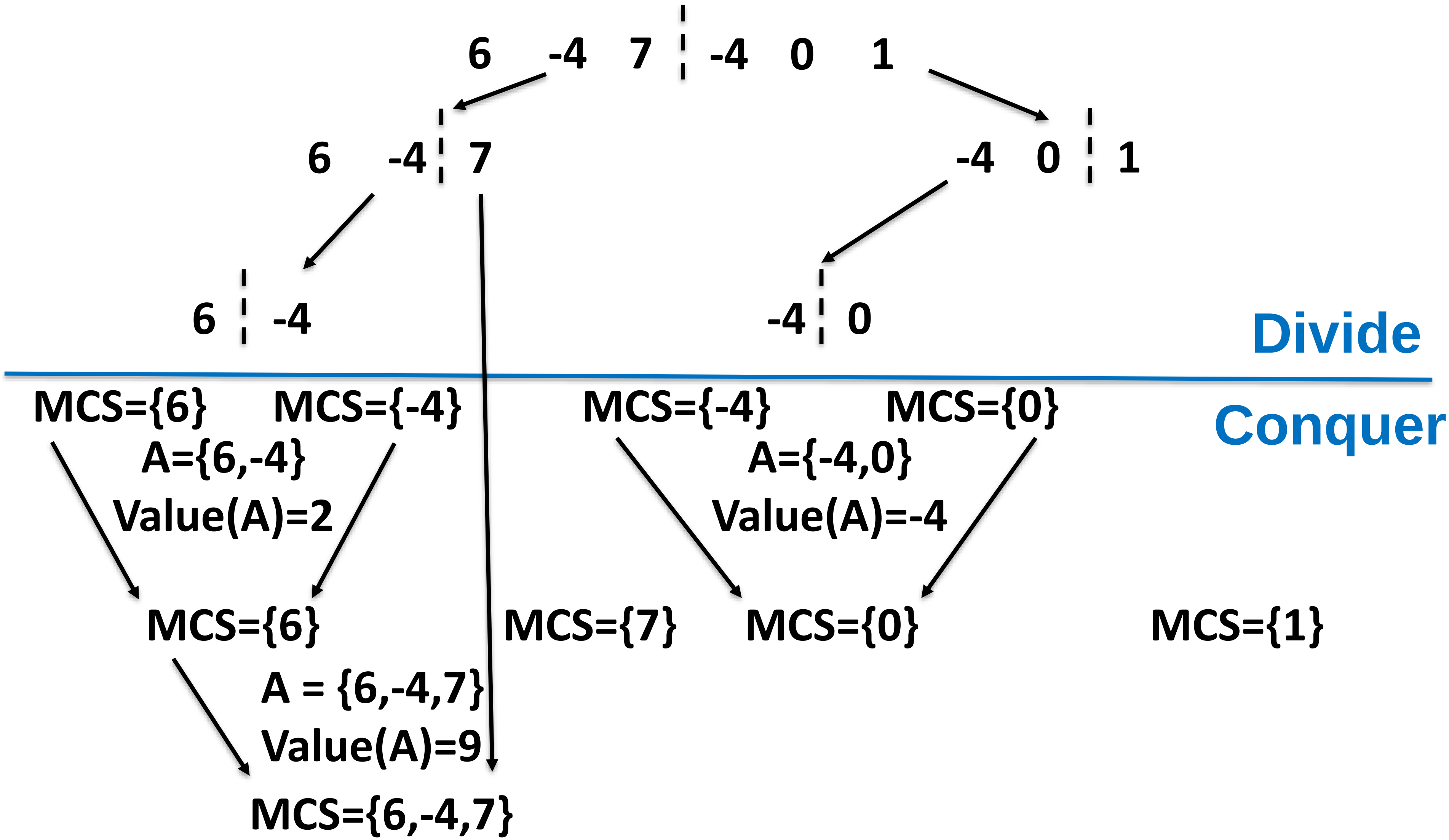
A Full Illustration of the D&C Algorithm



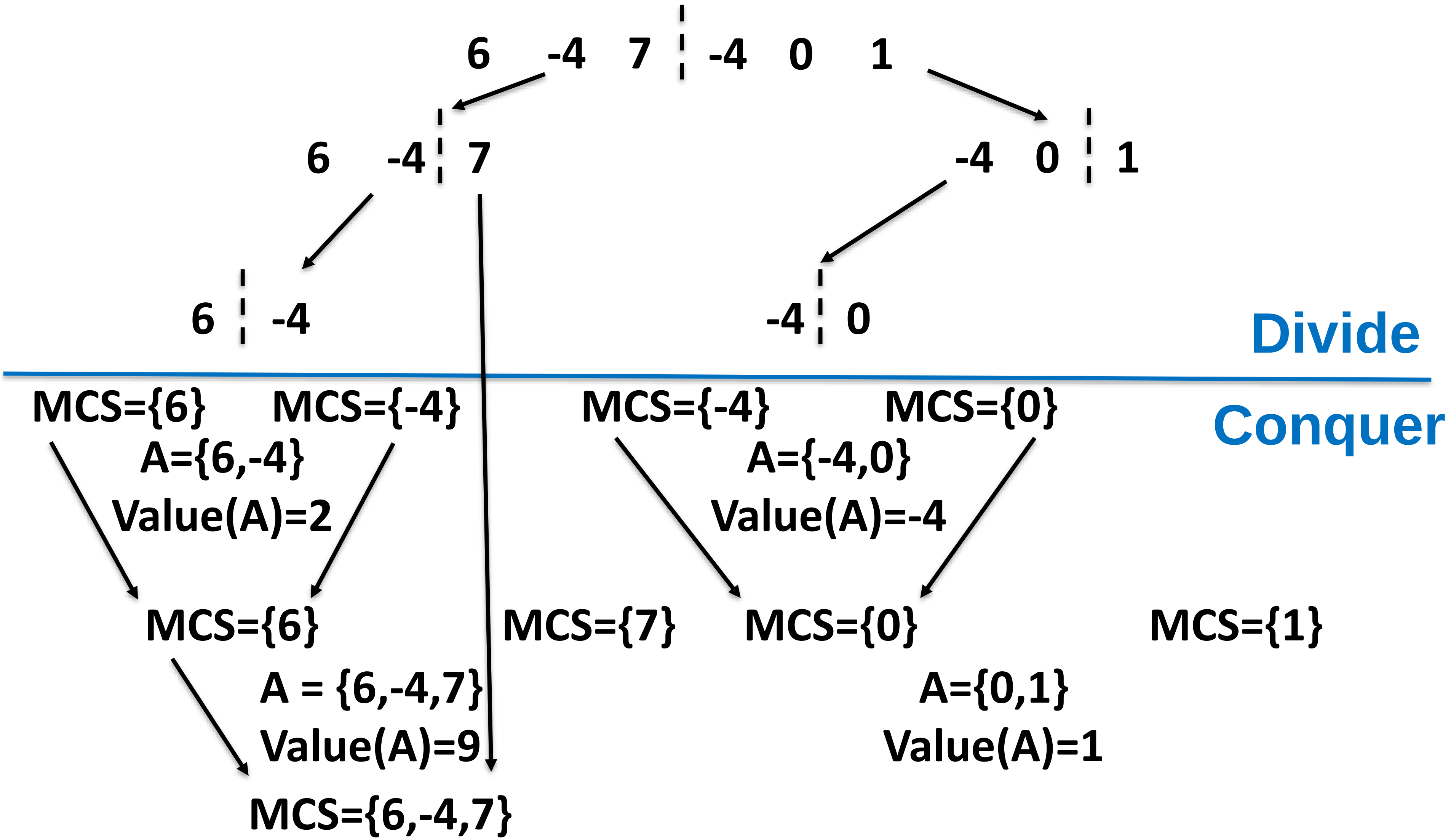
A Full Illustration of the D&C Algorithm



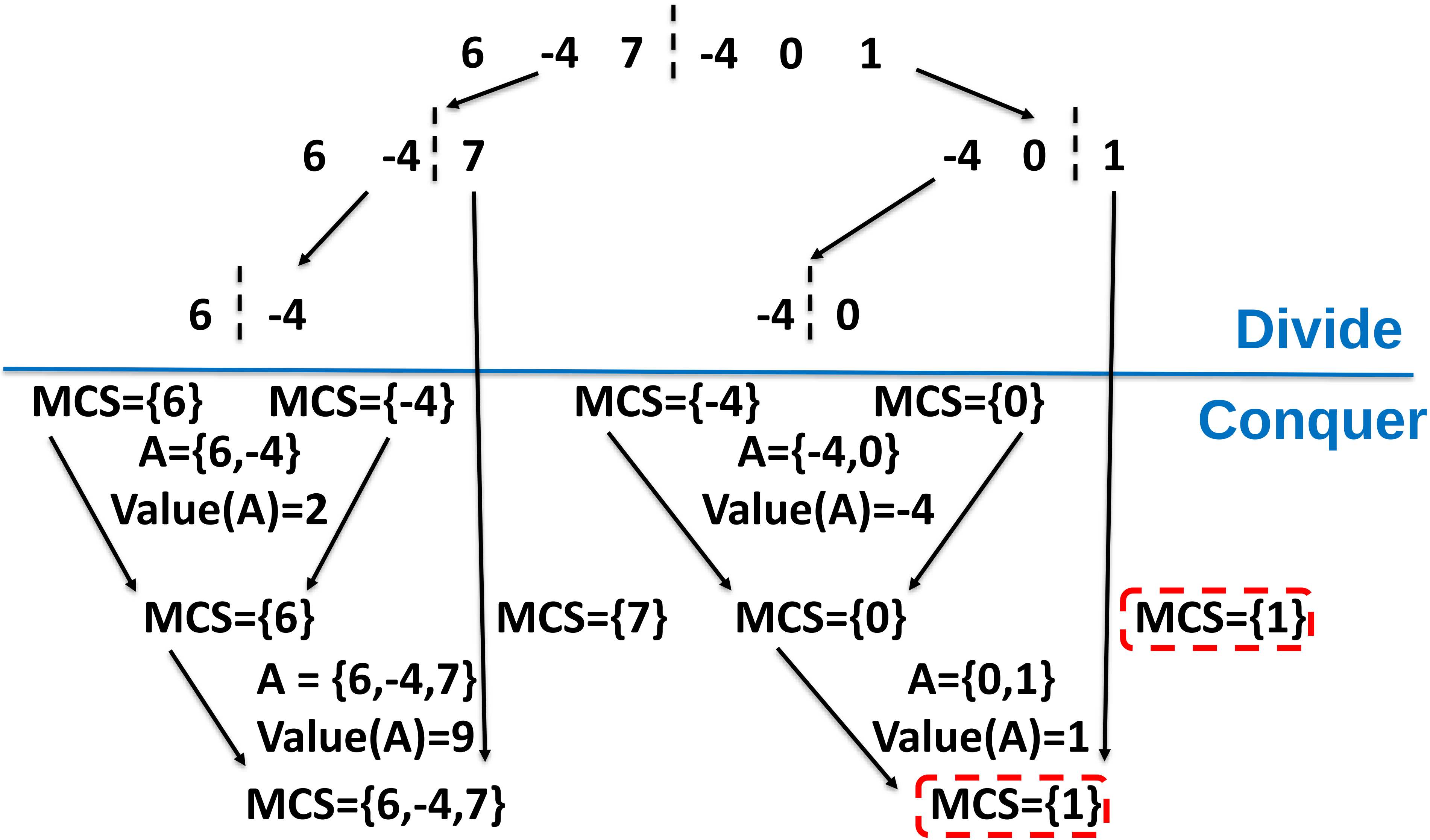
A Full Illustration of the D&C Algorithm



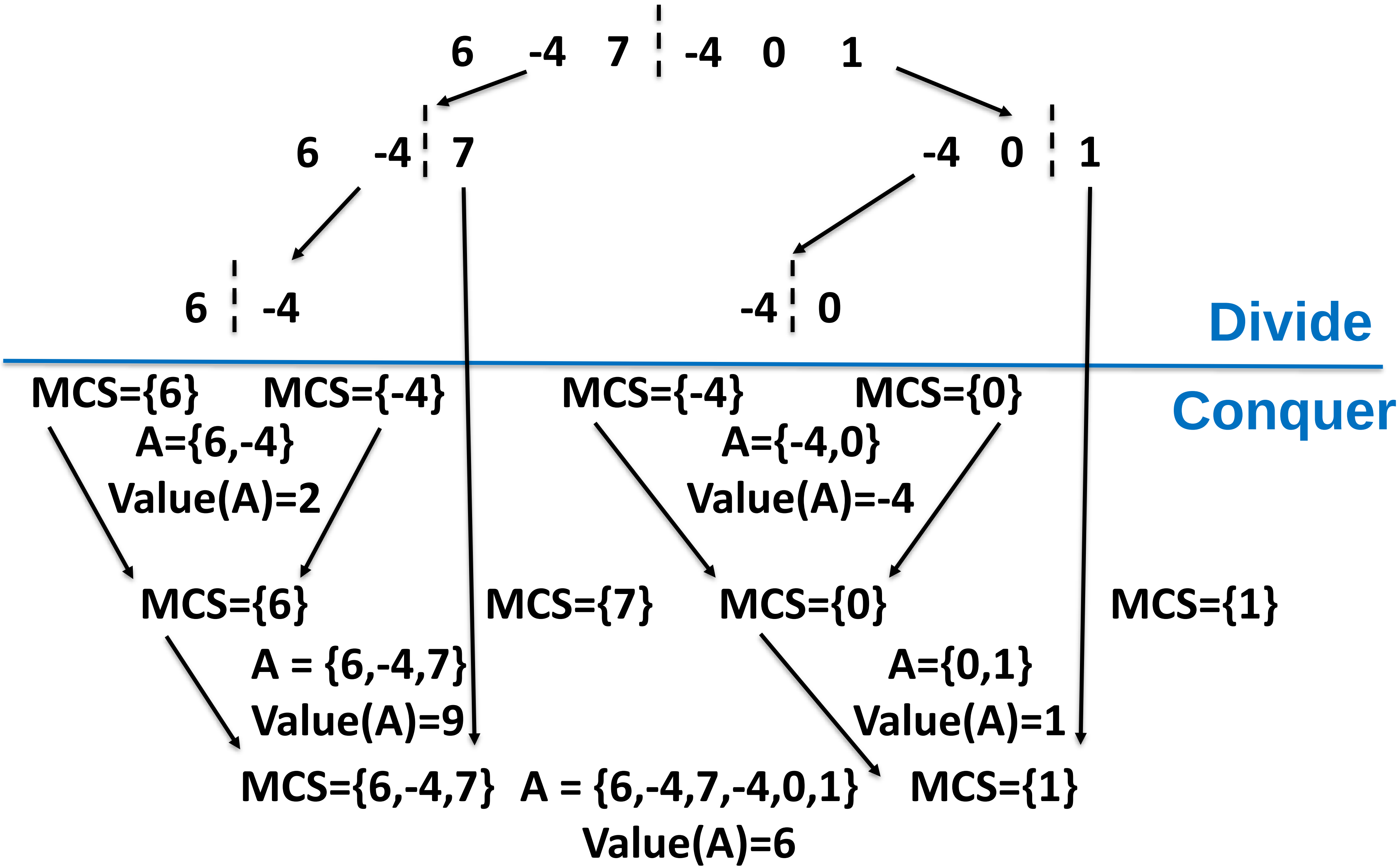
A Full Illustration of the D&C Algorithm



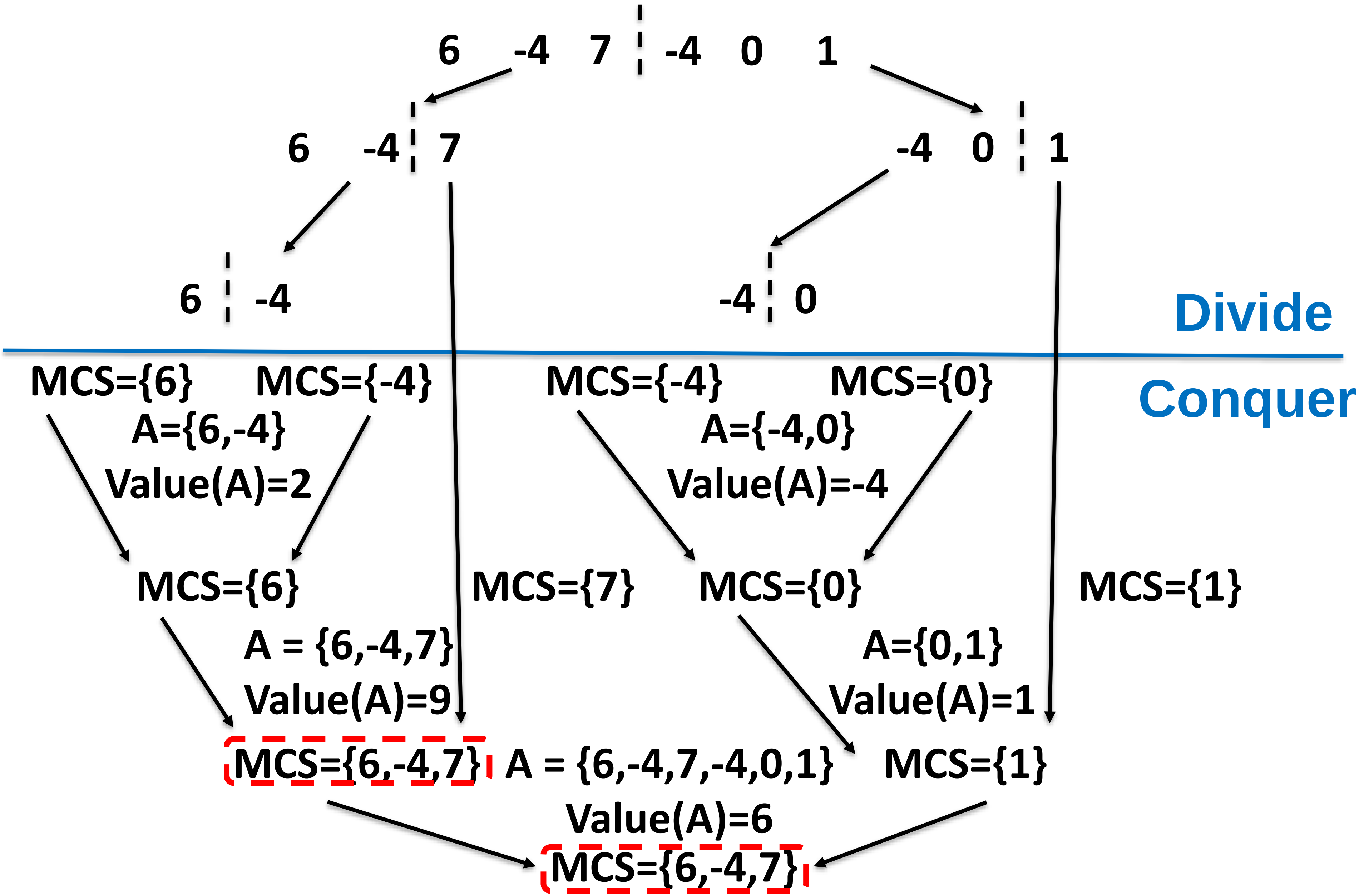
A Full Illustration of the D&C Algorithm



A Full Illustration of the D&C Algorithm



A Full Illustration of the D&C Algorithm



Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

Analysis of the D&C Algorithm

- *n*: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

```
begin
    if s = t then return A[s] // O(1)
.
```

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin

  if $s = t$ **then return** $A[s]$ // O(1)

  else

    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;

    Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$

.

Analysis of the D&C Algorithm

- *n*: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin
    if $s = t$ **then return** $A[s]$ // O(1)
    else
        $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
        Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
        Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$

.

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin
  if $s = t$ **then return** $A[s]$ // O(1)
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
    Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$
    Find MCS that contains *both* $A[m]$ and $A[m+1]$; // O(n)

.

Analysis of the D&C Algorithm

- *n*: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run *MCS*(*A, s, t*)

begin
  if $s = t$ **then return** $A[s]$ // O(1)

  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
    Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$
    Find MCS that contains *both* $A[m]$ and $A[m+1]$; // O(n)
    return maximum of the three sequences found // O(1)

.

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin
    if $s = t$ **then return** $A[s]$ // O(1)
    else
        $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
        Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
        Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$
        Find MCS that contains *both* $A[m]$ and $A[m+1]$; // O(n)
        return maximum of the three sequences found // O(1)
    end
end

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin
  if $s = t$ **then return** $A[s]$ // O(1)
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
    Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$
    Find MCS that contains *both* $A[m]$ and $A[m+1]$; // O(n)
    return maximum of the three sequences found // O(1)
  end
end

$$T(1) = O(\quad)$$

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin
  if $s = t$ **then return** $A[s]$ // O(1)
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
    Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$
    Find MCS that contains *both* $A[m]$ and $A[m+1]$; // O(n)
    return maximum of the three sequences found // O(1)
  end
end

$$T(1) = O(1)$$

Analysis of the D&C Algorithm

- n: problem size ($n = t - s + 1$)
- $T(n)$: time needed to run $MCS(A, s, t)$

begin
 if $s = t$ **then return** $A[s]$ // O(1)
 else
  $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
  Find $MCS(A, s, m)$; // $T(\lceil \frac{n}{2} \rceil)$
  Find $MCS(A, m+1, t)$; // $T(\lfloor \frac{n}{2} \rfloor)$
  Find MCS that contains *both* $A[m]$ and $A[m+1]$; // O(n)
  return maximum of the three sequences found // O(1)
 end
end

$$T(1) = O(1)$$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) \quad \text{for } n > 1$$

Analysis of the D&C Algorithm

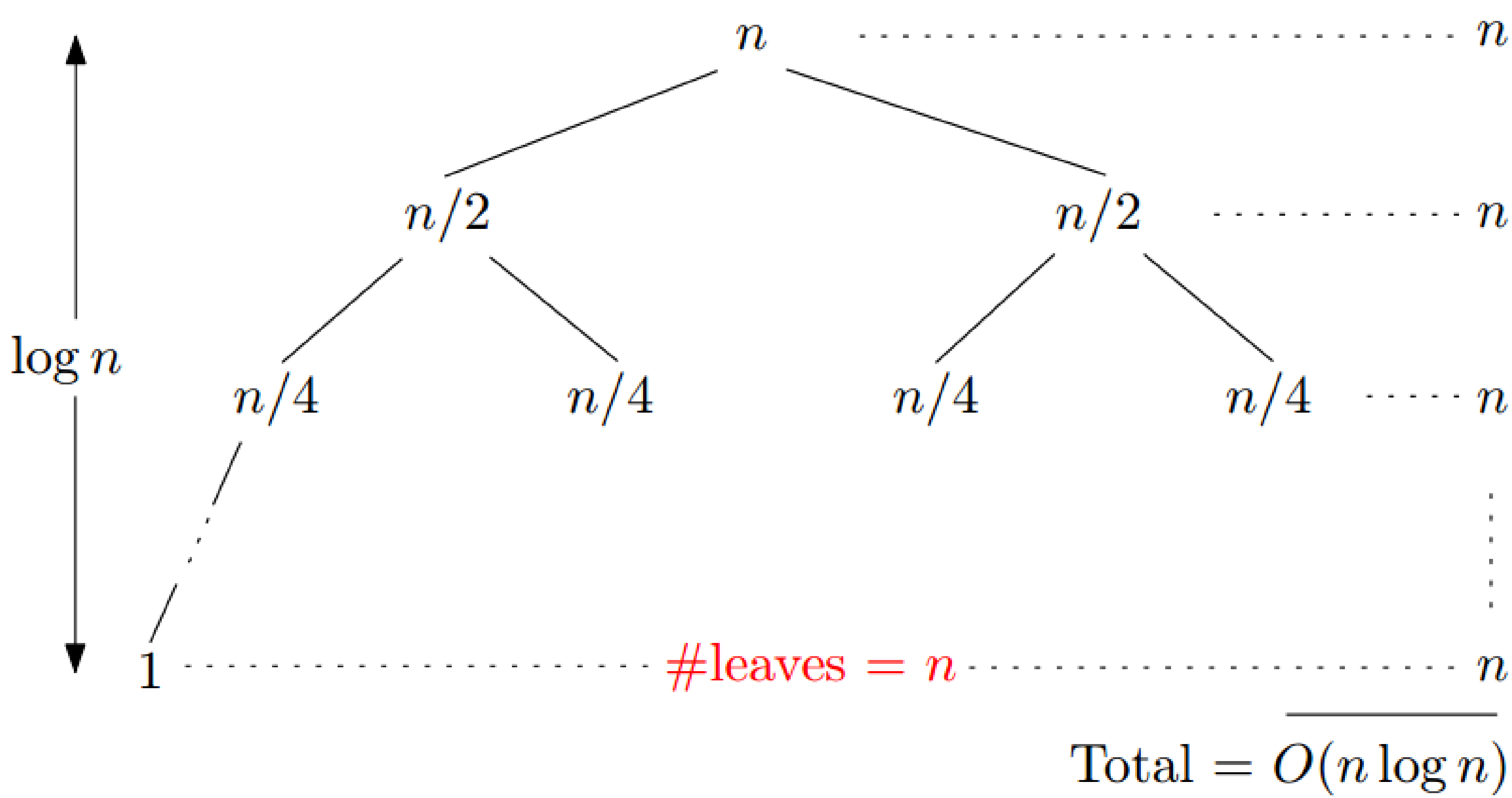
To simplify the analysis, we assume that n is a power of 2

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

Analysis of the D&C Algorithm

To simplify the analysis, we assume that n is a power of 2

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



Summary

In the MCS problem, we saw 3 different algorithms for solving the maximum contiguous subarray problem

- A O(n^3) brute force algorithm

Summary

In the MCS problem, we saw 3 different algorithms for solving the maximum contiguous subarray problem

- A $O(n^3)$ brute force algorithm
- A $O(n^2)$ algorithm that reuses data

Summary

In the MCS problem, we saw 3 different algorithms for solving the maximum contiguous subarray problem

- A $O(n^3)$ brute force algorithm
- A $O(n^2)$ algorithm that reuses data
- A $O(n \log n)$ divide-and-conquer algorithm

Summary

In the MCS problem, we saw 3 different algorithms for solving the maximum contiguous subarray problem

- A $O(n^3)$ brute force algorithm
- A $O(n^2)$ algorithm that reuses data
- A $O(n \log n)$ divide-and-conquer algorithm

Can you solve the problem in $O(n)$ time?

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A 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: $a_1, a_2, \ldots, a_n$.

	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

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A 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

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 ← 0;
for i ← 1 to L.length do
    for j ← i + 1 to L.length do
        if L[i] > L[j] then
            r ← r + 1;
        end
    end
end
return r;
```

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

Outline

- Introduction to Part I
- Maximum Contiguous Subarray Problem
  - Problem definition
  - A brute force algorithm
  - A data-reuse algorithm
  - A divide-and-conquer algorithm
  - Analysis of the divide-and-conquer algorithm
- Counting Inversions Problem
  - Problem definition
  - A brute force algorithm
  - A 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 $A[1 \ldots n]$, 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∈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

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**

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

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?

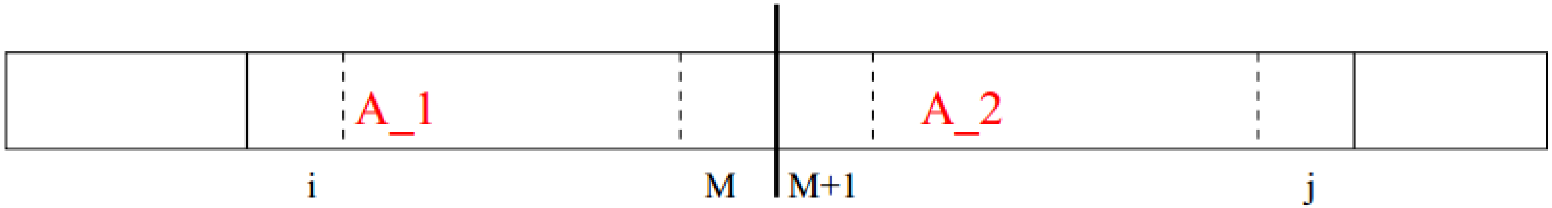
List A

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 $A[i \dots m]$, $V(i, m) = V(i + 1, m) + A[i]$

```
MAX ← A[m];
SUM ← A[m];
for i ← m − 1 downto 1 do
    SUM ← SUM + A[i];
    if SUM > MAX then
        MAX ← SUM;
    end
end
A₁=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 ∈ 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

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

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,

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.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

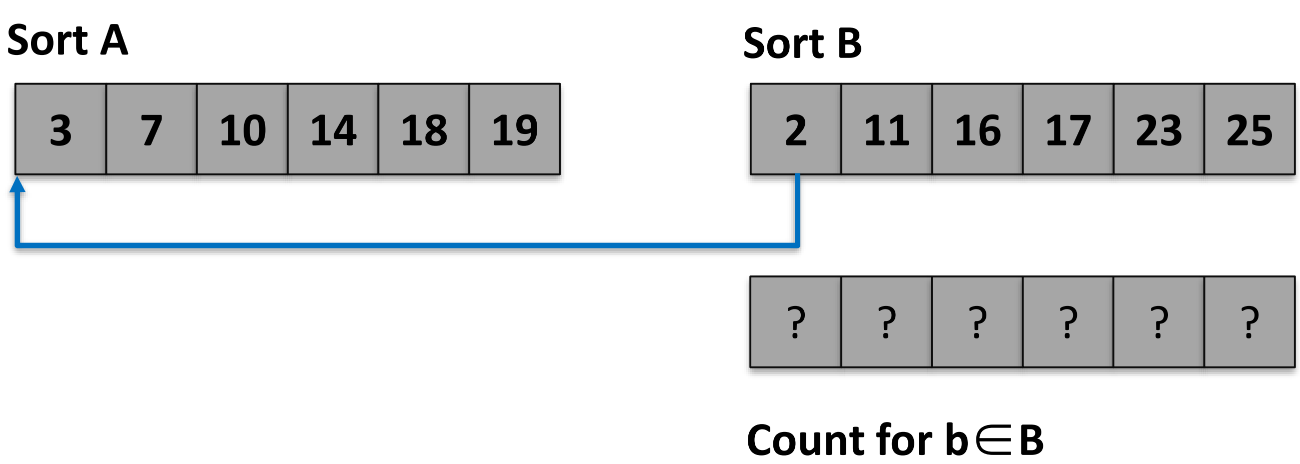
How to combine two subproblems?

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.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

6	?	?	?	?	?

Count for b∈B

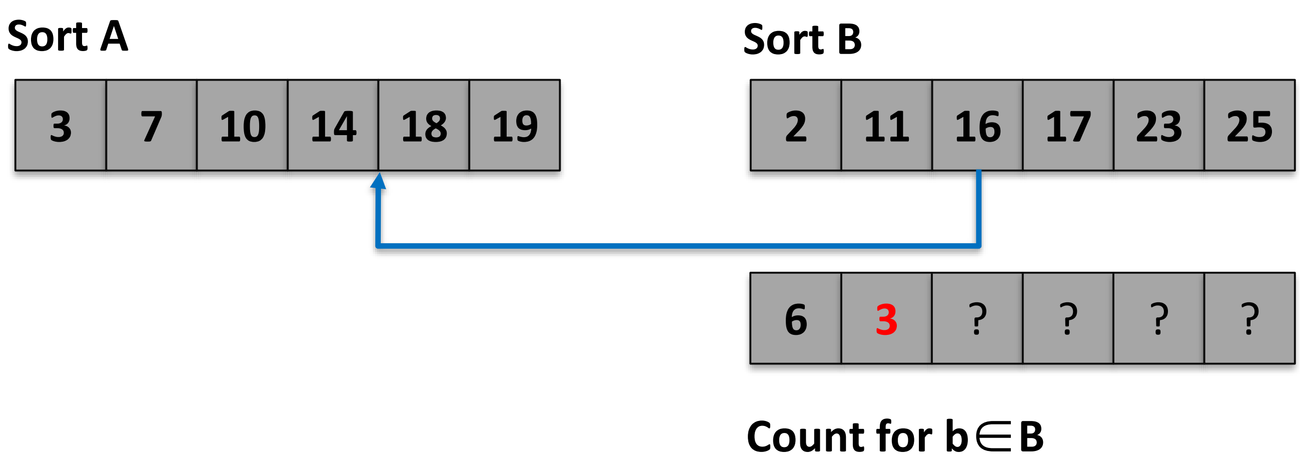
How to combine two subproblems?

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.



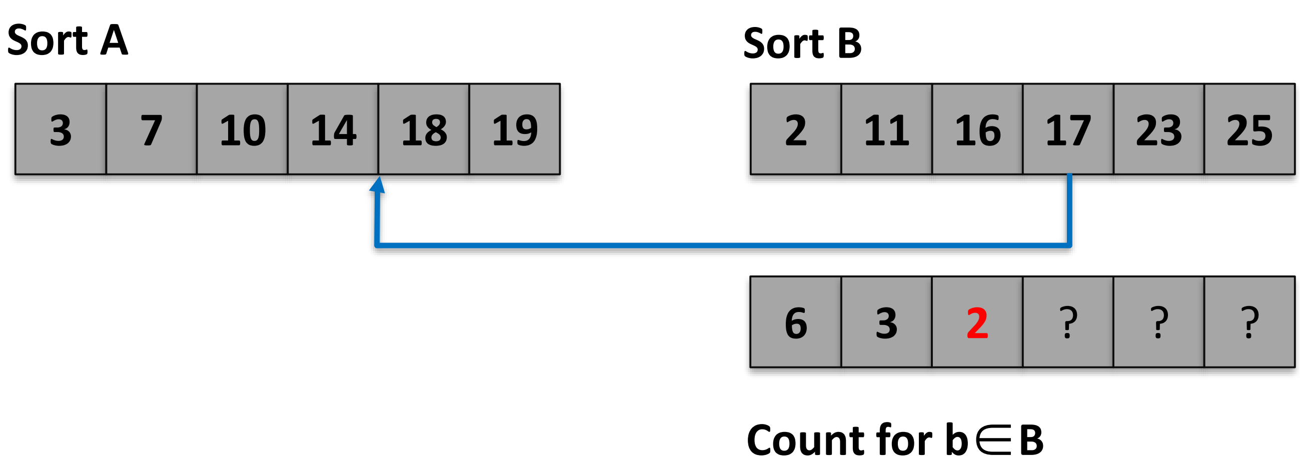
How to combine two subproblems?

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.



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	?	?

Count for b∈B

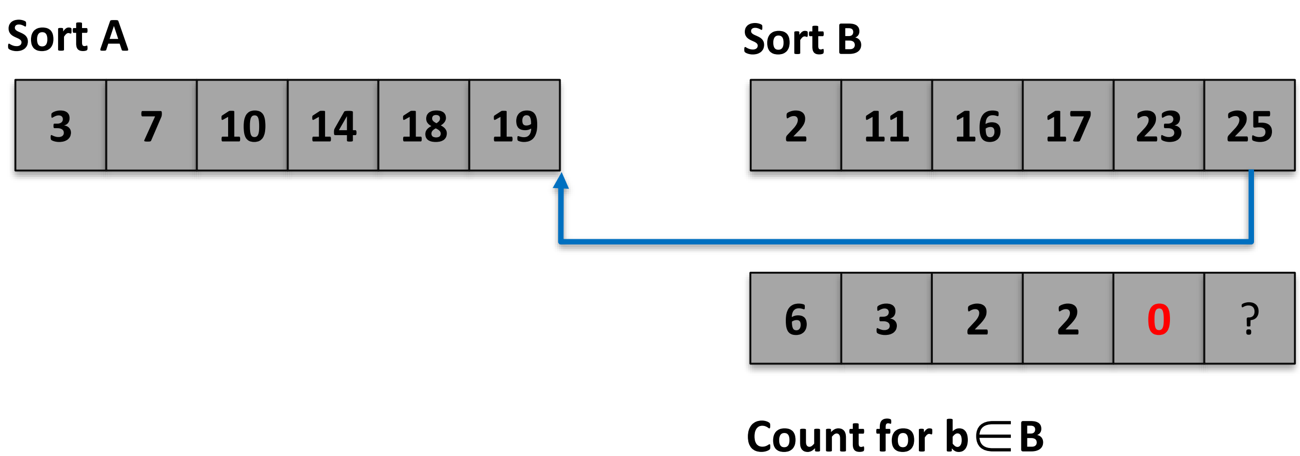
How to combine two subproblems?

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	0

Count for b∈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.

Sort A

3	7	10	14	18	19

Sort B

2	11	16	17	23	25

6	3	2	2	0	0

Count for $b \in B$

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

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

Combine two subproblems: Improvement

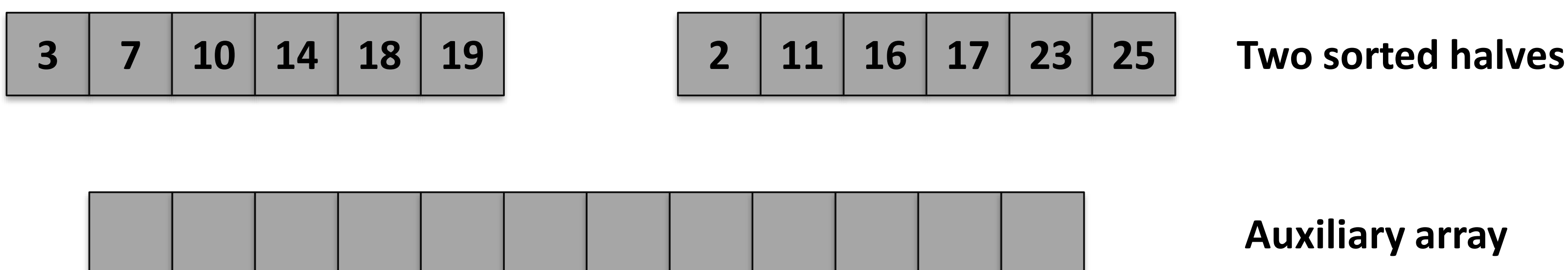
Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- 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.

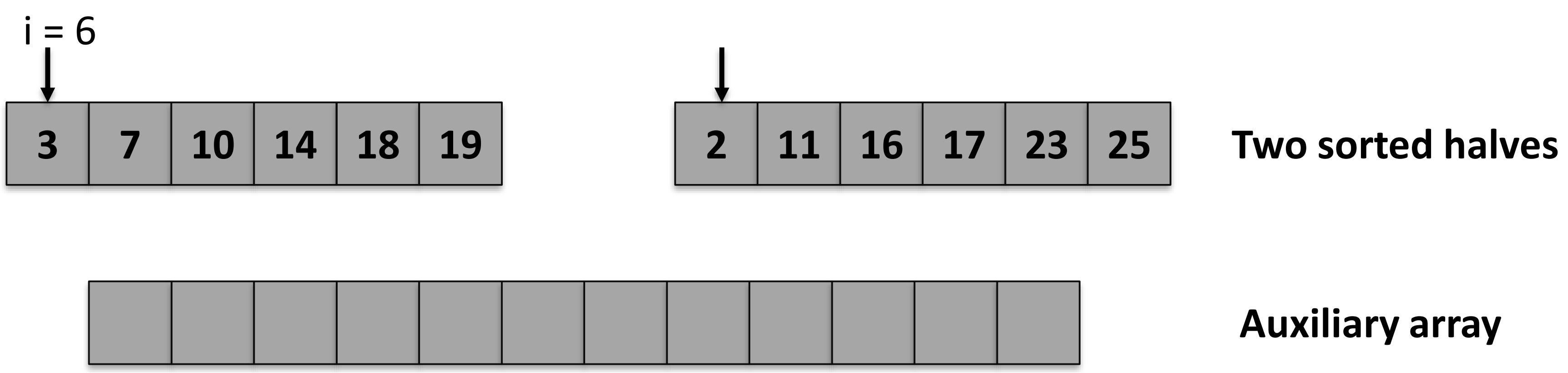
- Scan A and B from left to right.
- Compare a_i and b_j.
  - If $a_i < b_j$, then a_i is not inverted with any element left in B.
  - If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.



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.

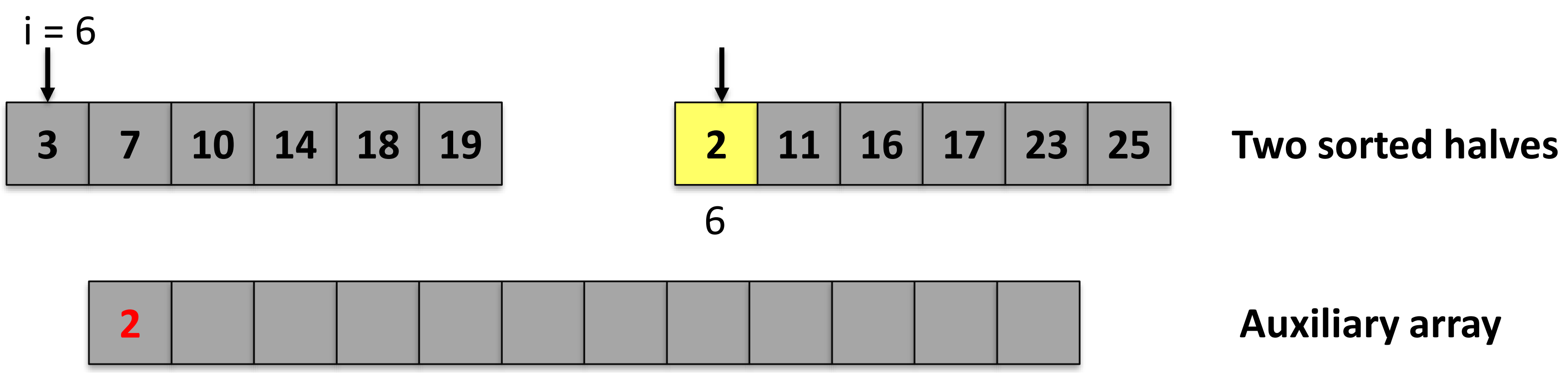


Total:

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.

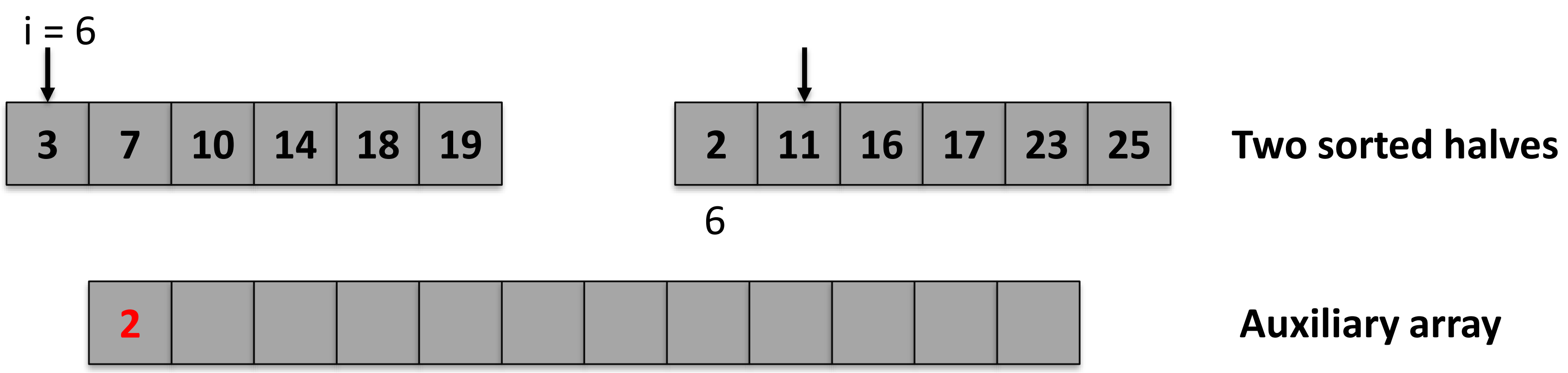


Total: 6

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.

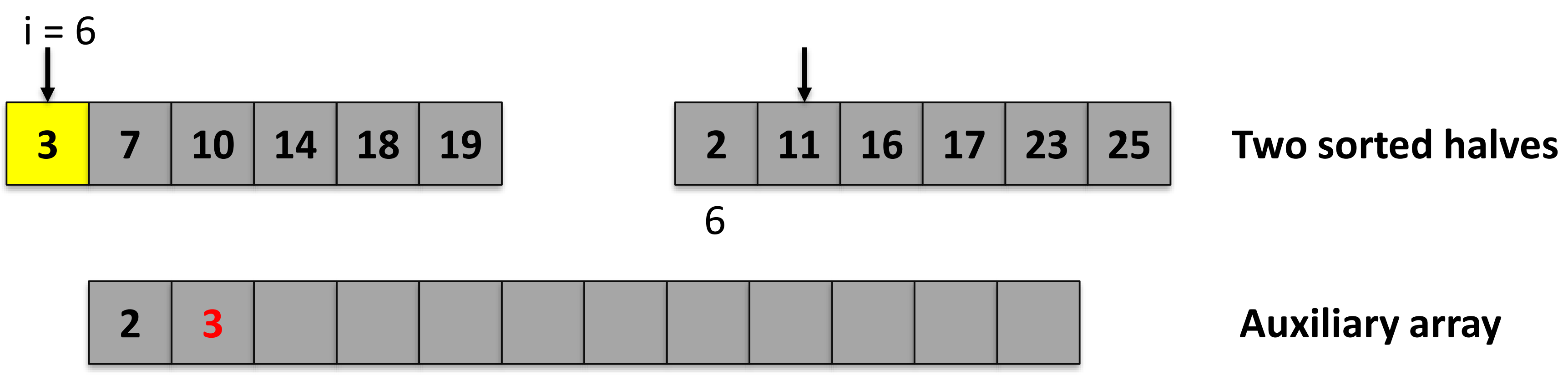


Total: 6

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.

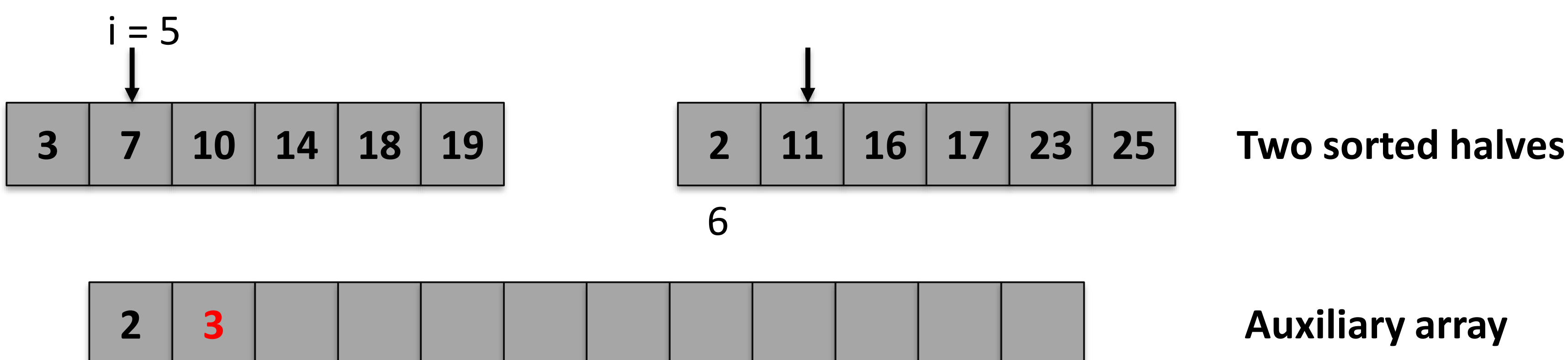


Total: 6

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.

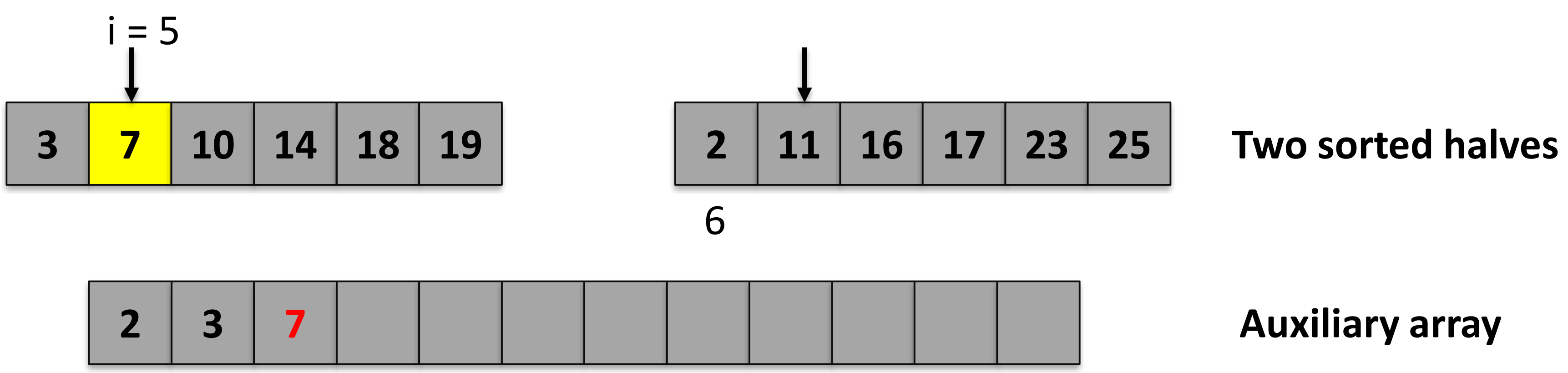


Total: 6

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.

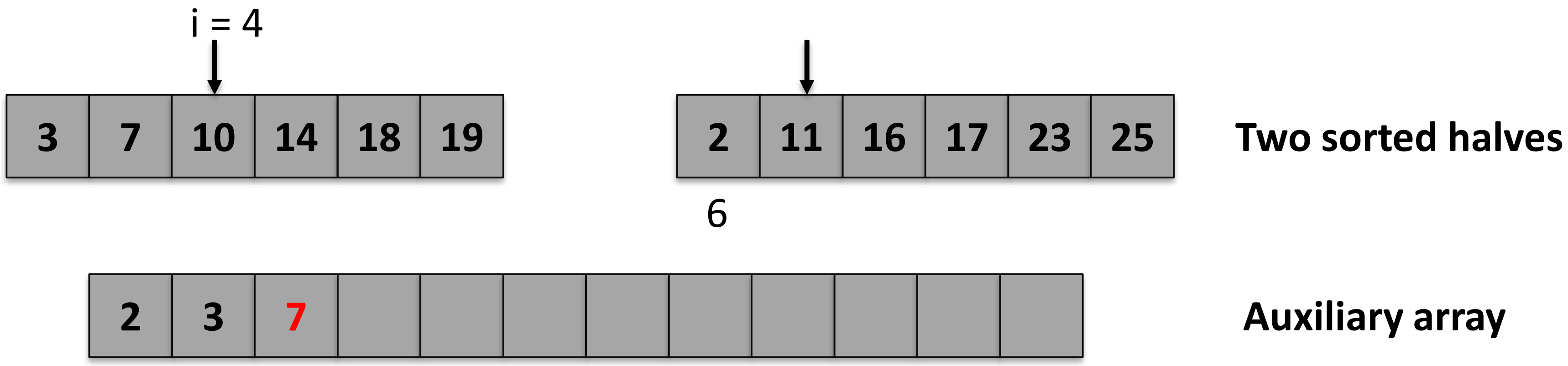


Total: 6

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.

i = 4

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

6

Two sorted halves

2	**3**	**7**	**10**								

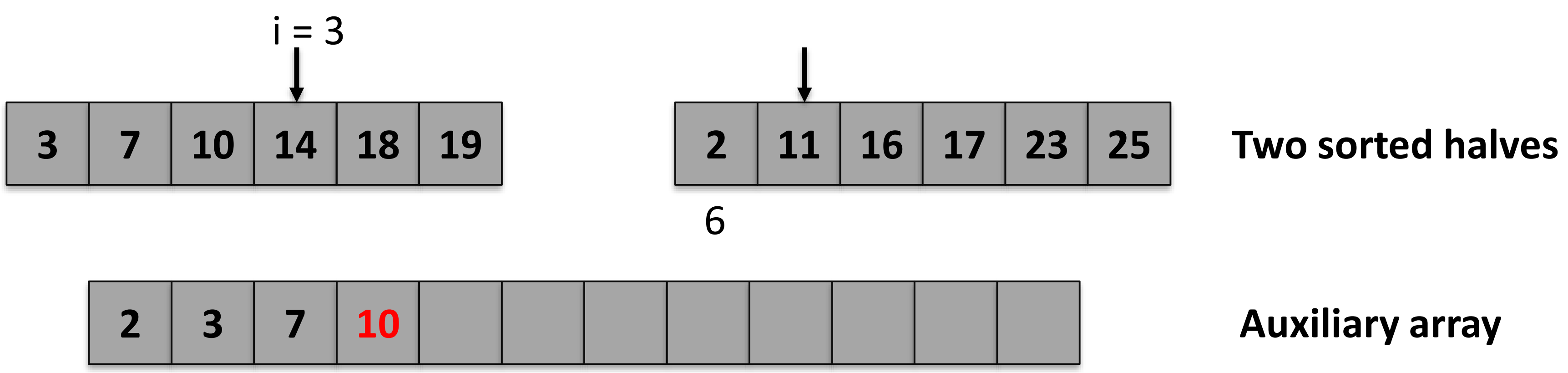
Auxiliary array

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.

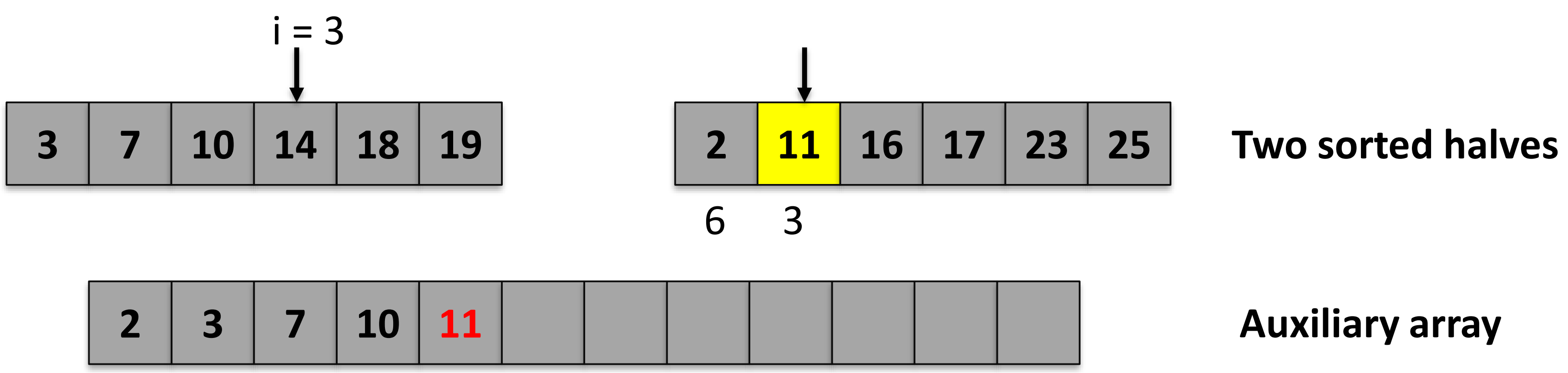


Total: 6

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.

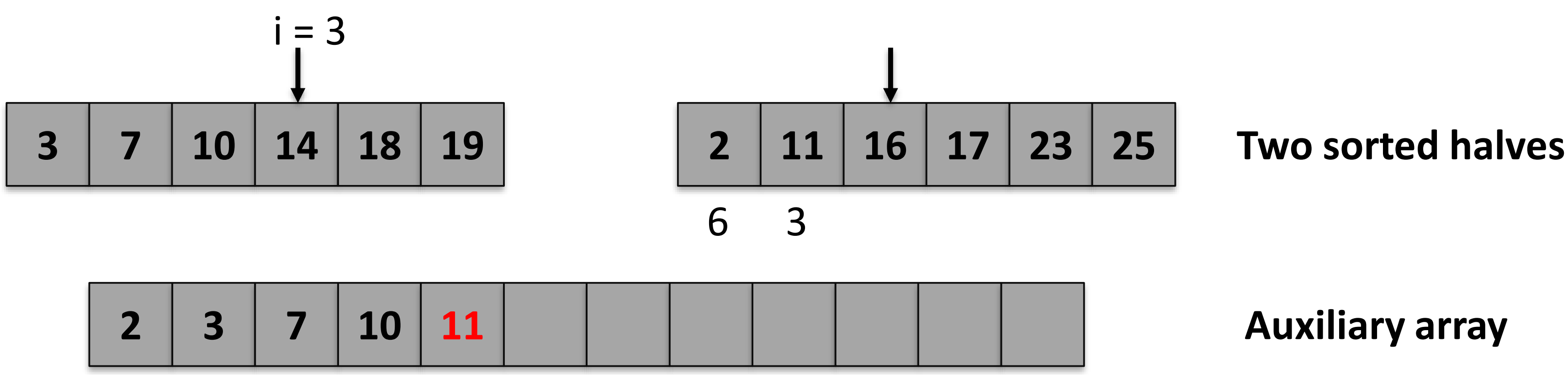


Total: 6+3

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.

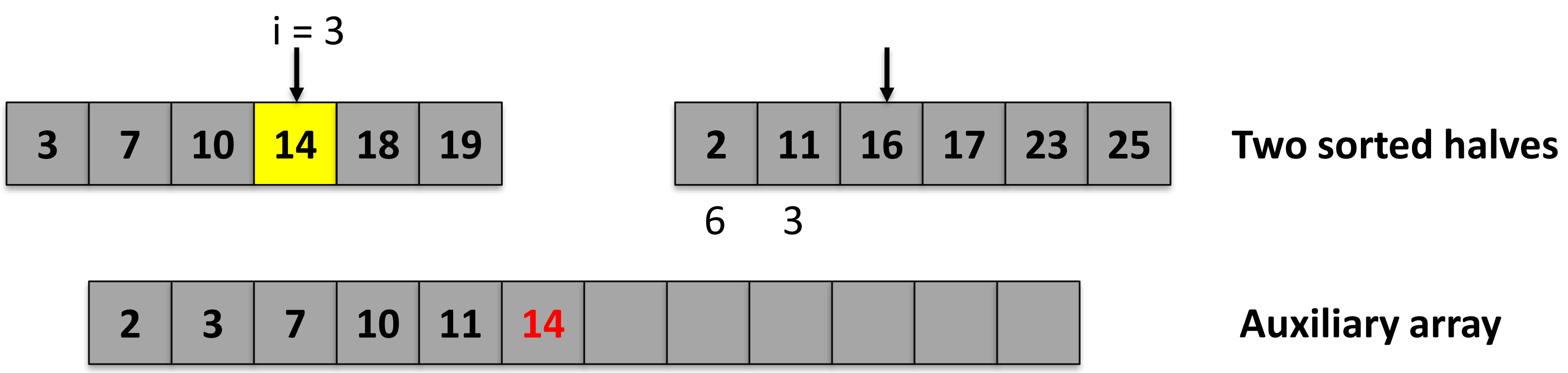


Total: 6+3

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.

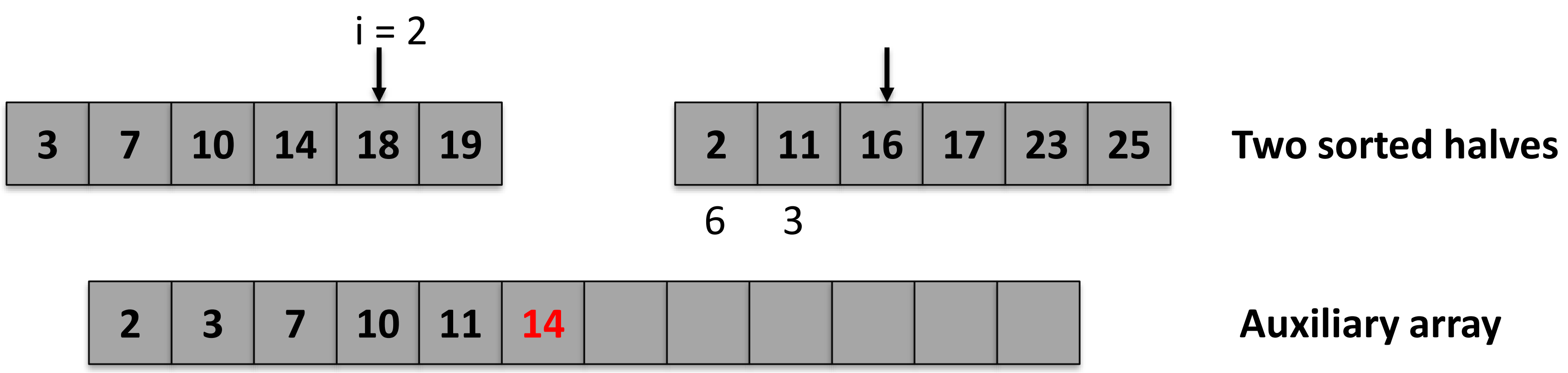


Total: 6+3

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.

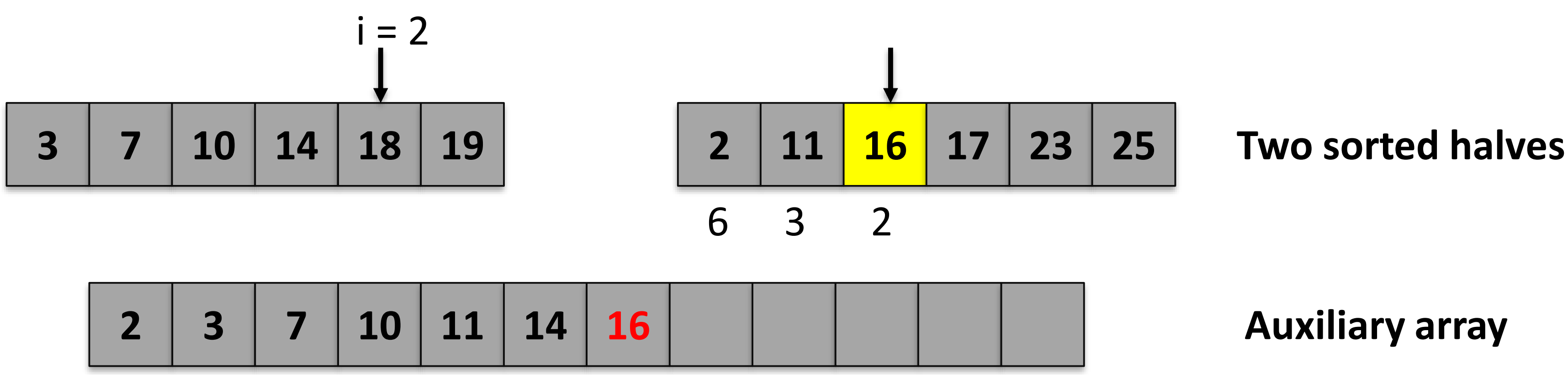


Total: 6+3

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.

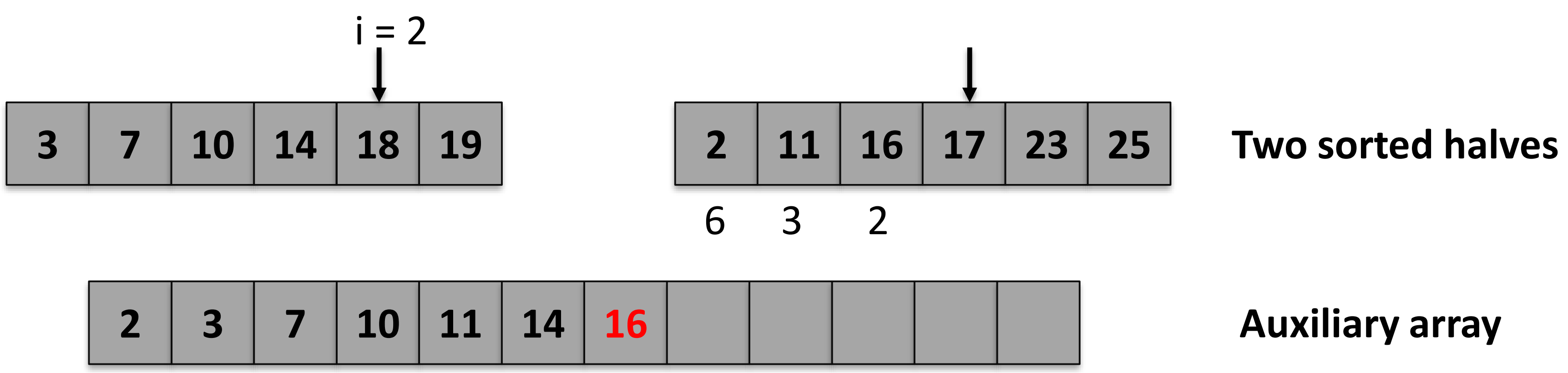


Total: 6+3+2

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ 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 ∈ 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	2	2		

Two sorted halves

2	3	7	10	11	14	16	17				

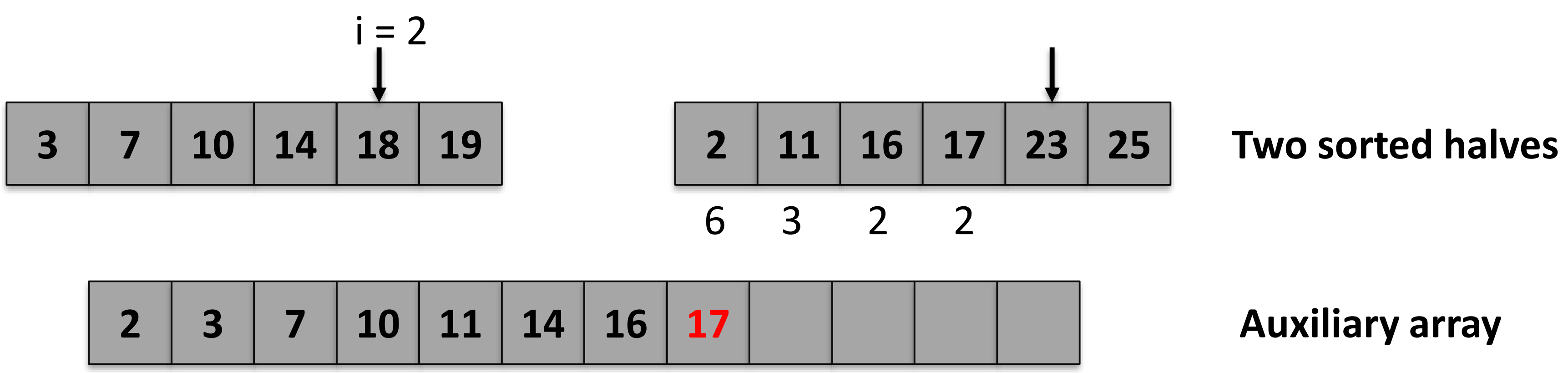
Auxiliary array

Total: 6+3+2+2

Combine two subproblems: Improvement

Count inversions (a, b) with a ∈ 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.

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	18			

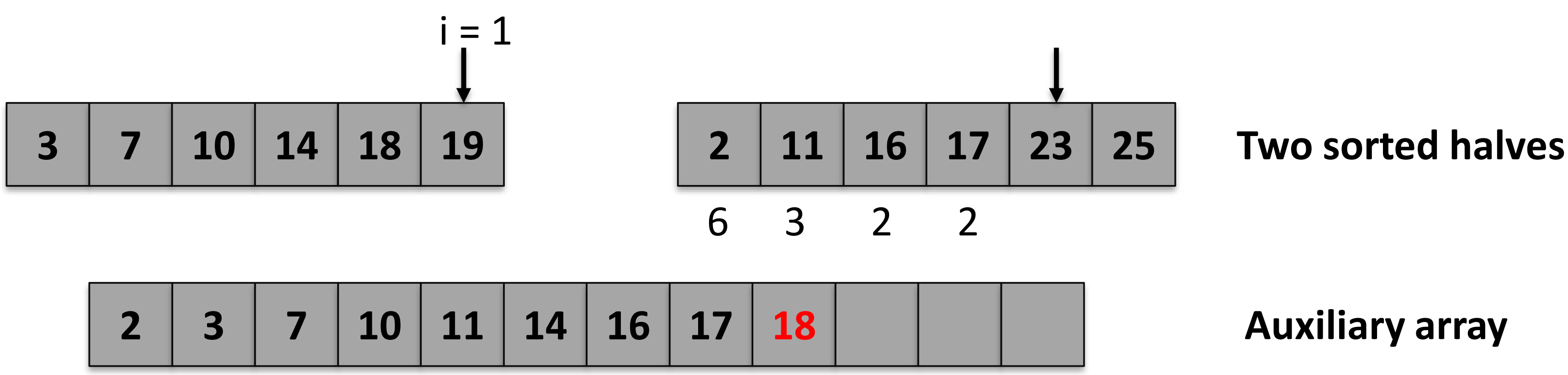
Auxiliary array

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.

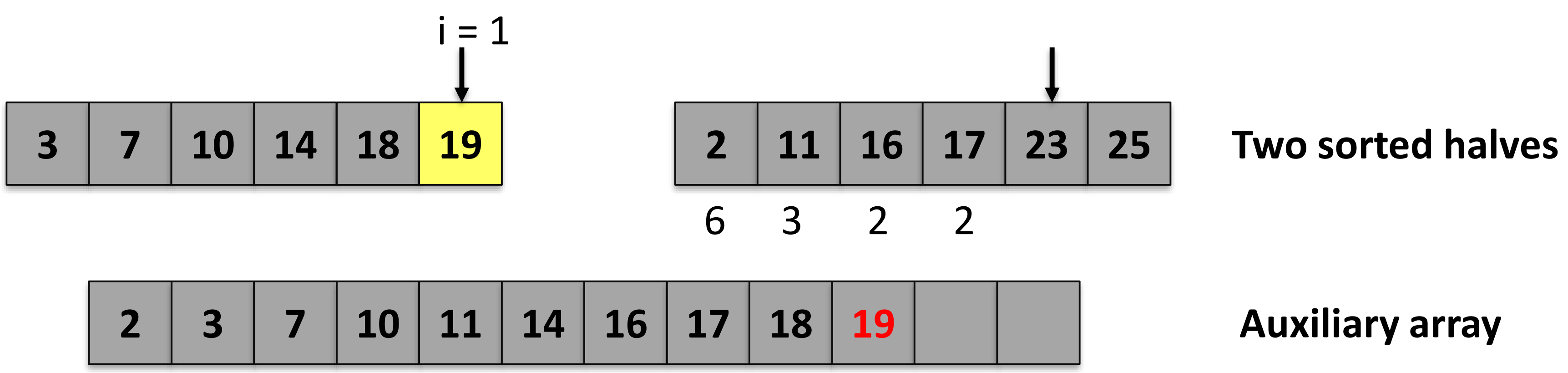


Total: 6+3+2+2

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.

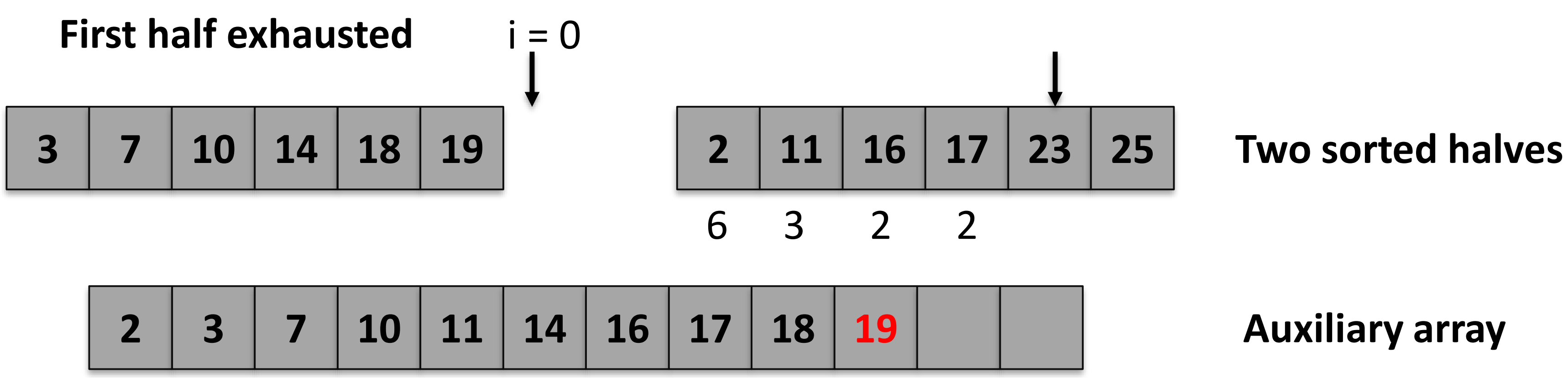


Total: 6+3+2+2

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.



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.

i = 0

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

Two sorted halves

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

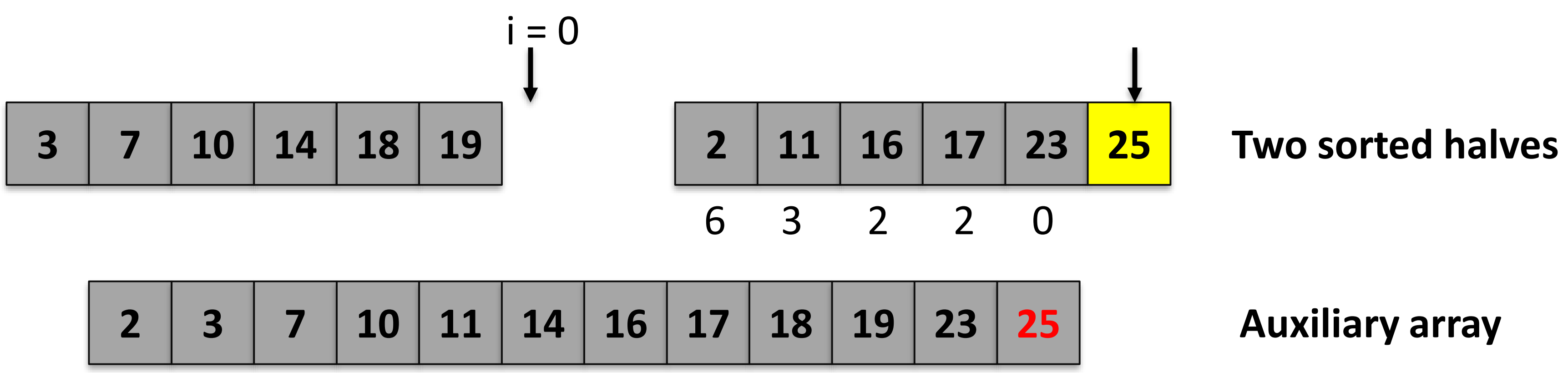
Auxiliary array

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.



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
```

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,

Combine two subproblems: Improvement

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

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( ) time where n is the number of elements in A and B.

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

Review of The Complete MCS Algorithm

$MCS(A, s, t)$

Input: $A[s \dots t]$ with $s \leq t$
Output: MCS of $A[s \dots t]$
begin
  if $s = t$ **then return** $A[s]$;
  else
    $m \leftarrow \lfloor \frac{s+t}{2} \rfloor$;
    Find $MCS(A, s, m)$;
    Find $MCS(A, m+1, t)$;
    Find MCS that contains both $A[m]$ and $A[m+1]$;
    return maximum of the three sequences found
  end
end

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 *is empty* **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 *is empty* **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 *is empty* **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);$//T(\lfloor \frac{n}{2} \rfloor)$
$(r_L, L) \leftarrow$

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if L *is empty* **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

The Complete Divide-and-Conquer Algorithm

Sort-and-Count(L)

Input: L
Output: r_L, L
if L *is empty* **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 *is empty* **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

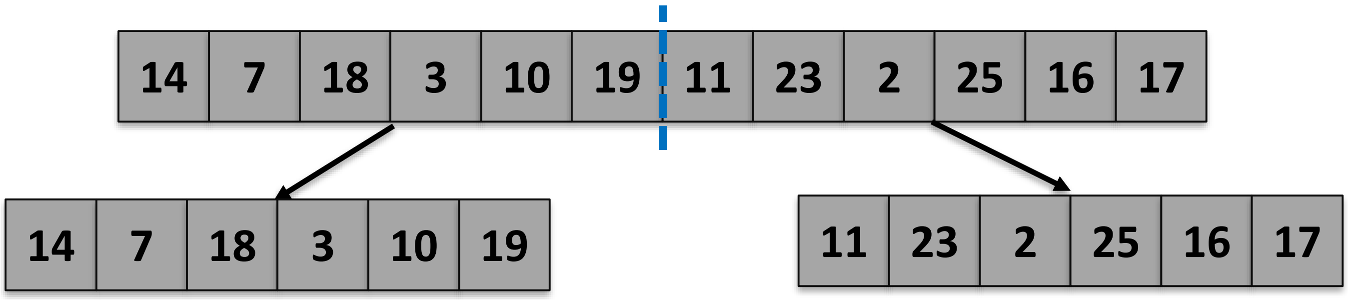
Example

Divide

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

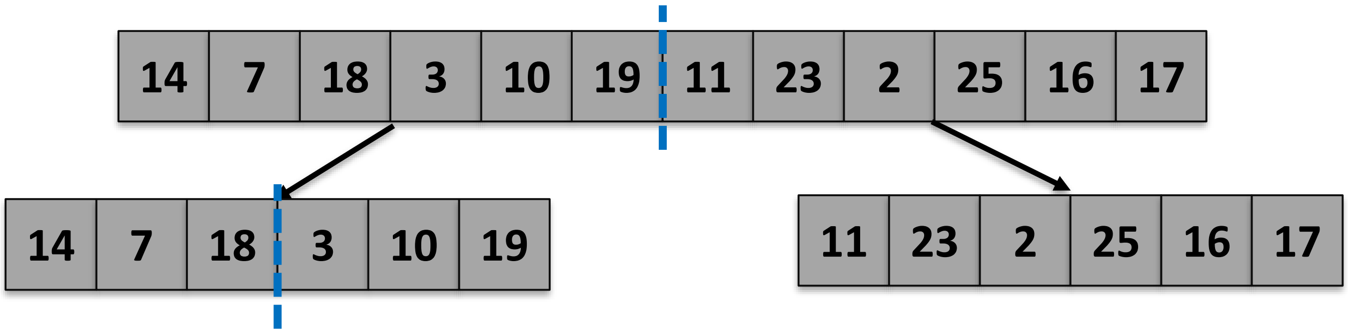
Example

Divide



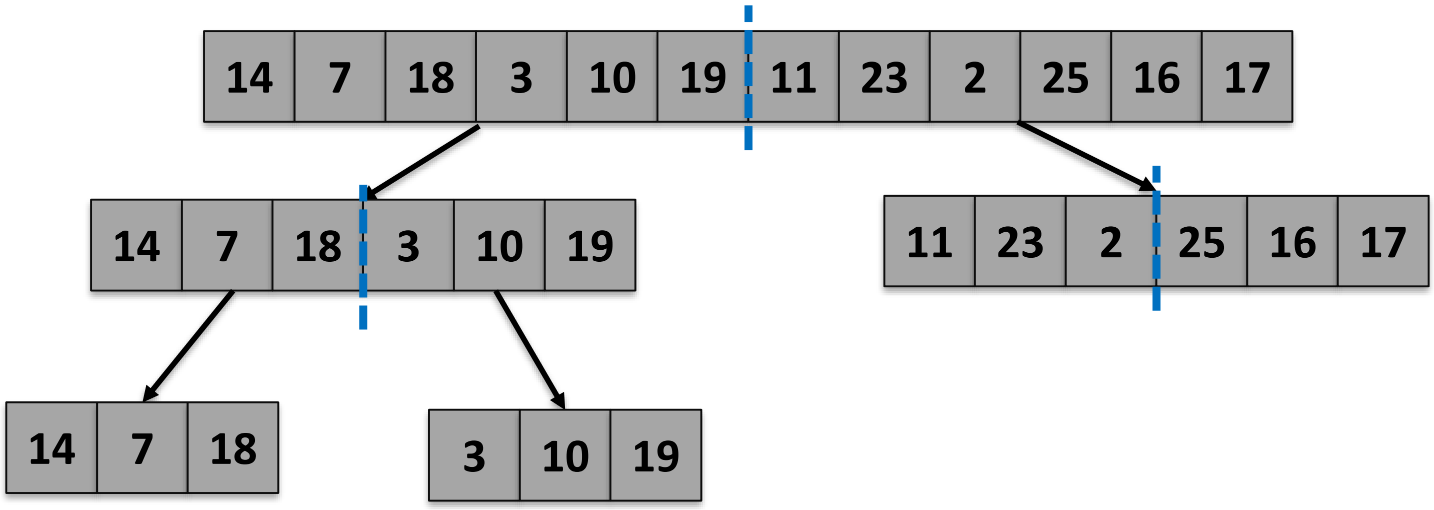
Example

Divide



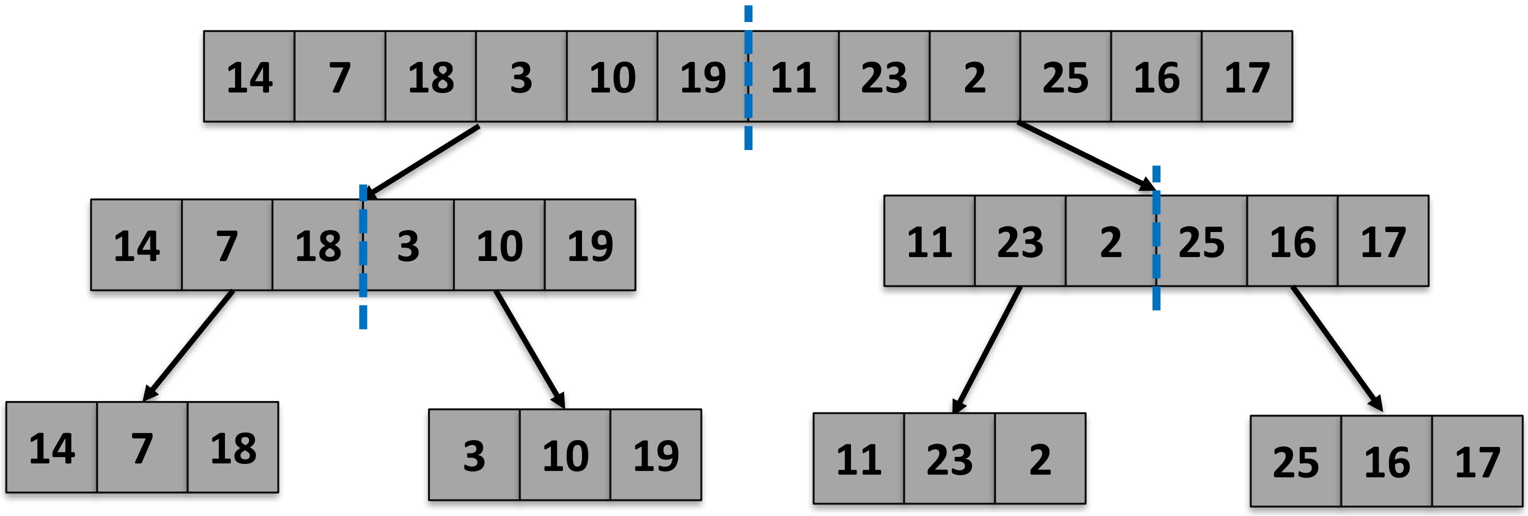
Example

Divide



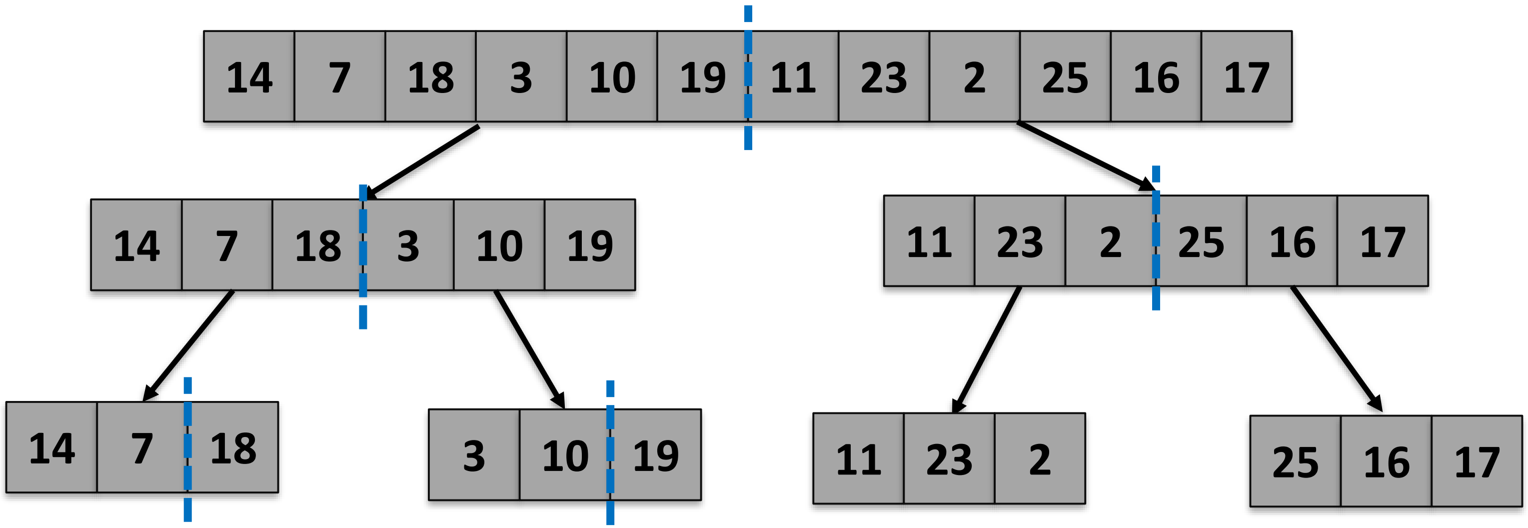
Example

Divide



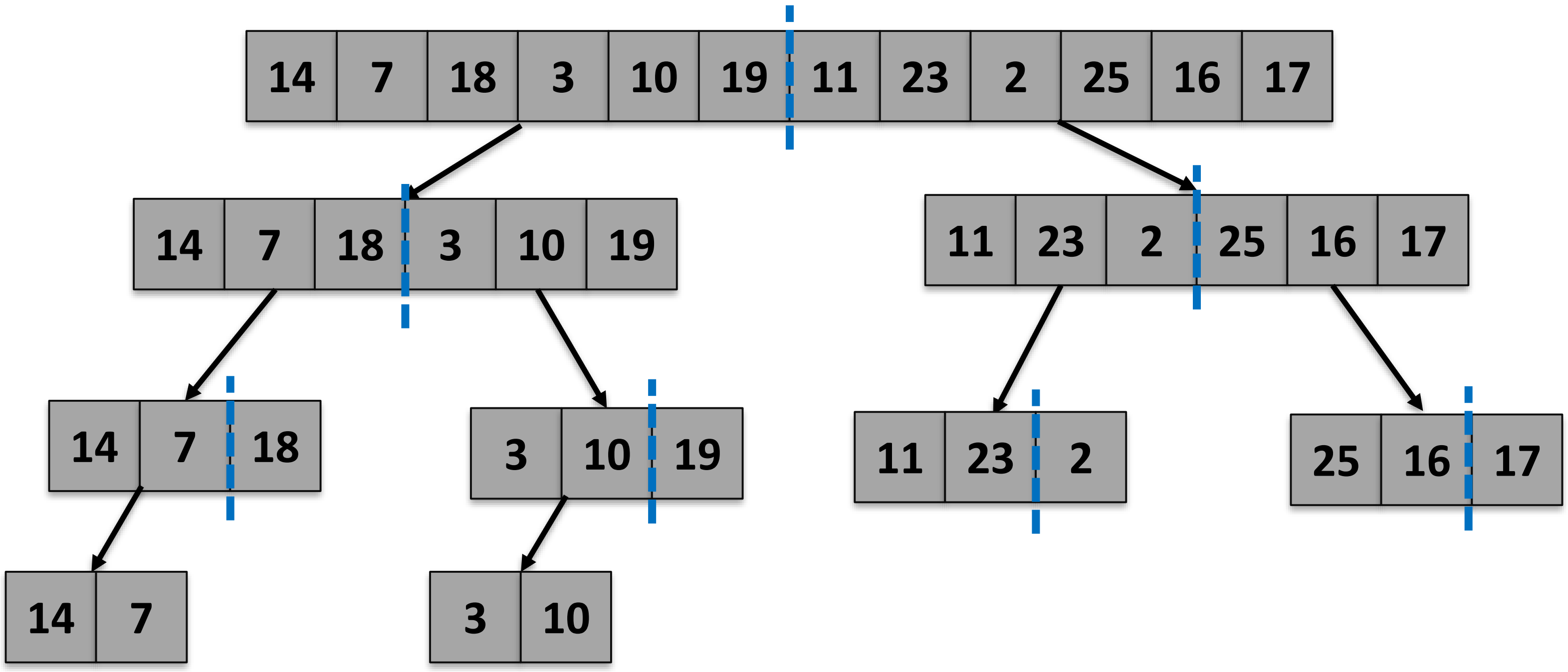
Example

Divide



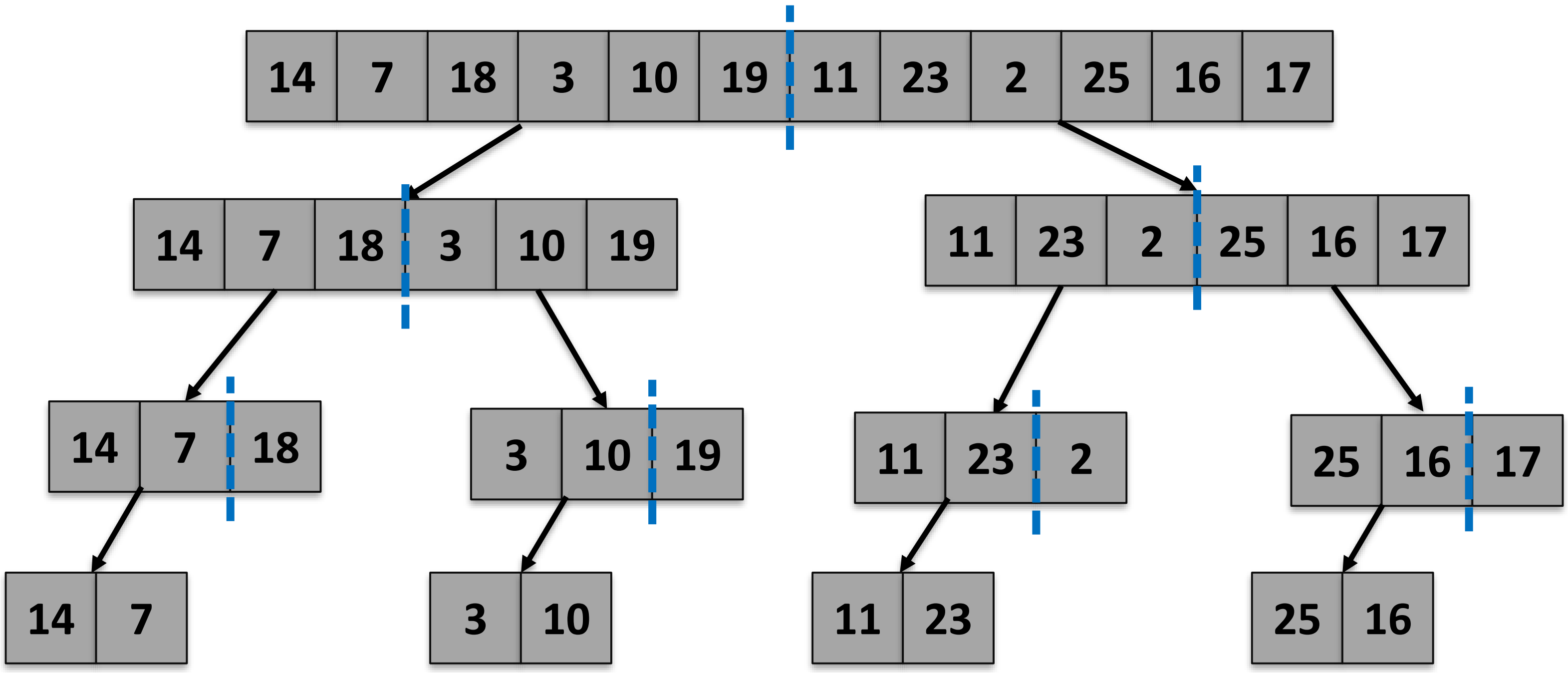
Example

Divide



Example

Divide



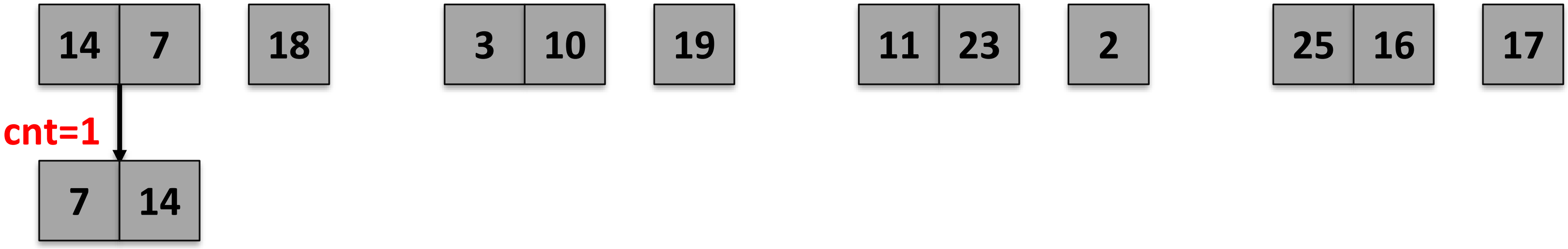
Example

Conquer

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

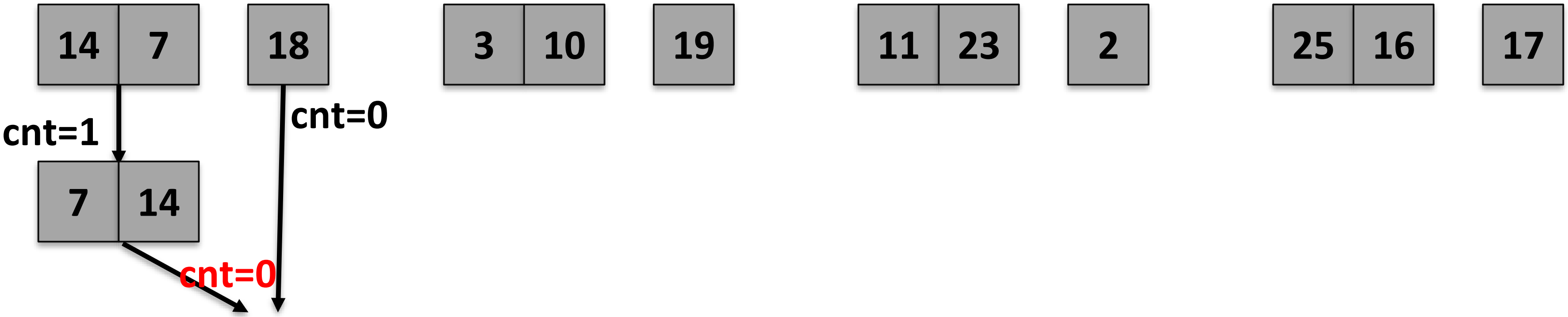
Example

Conquer



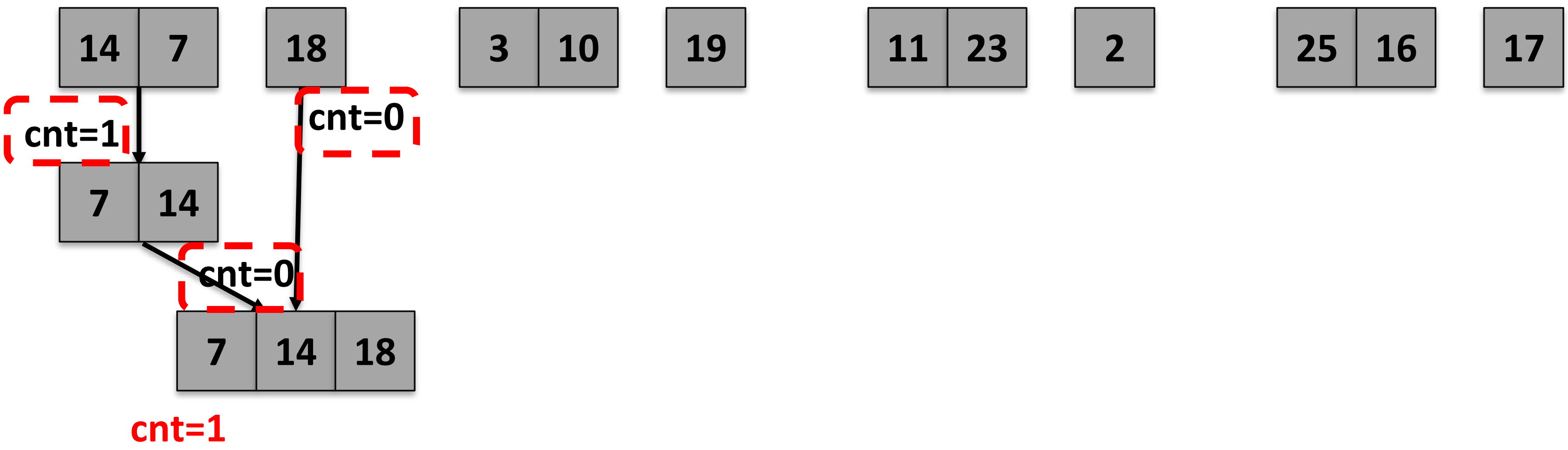
Example

Conquer



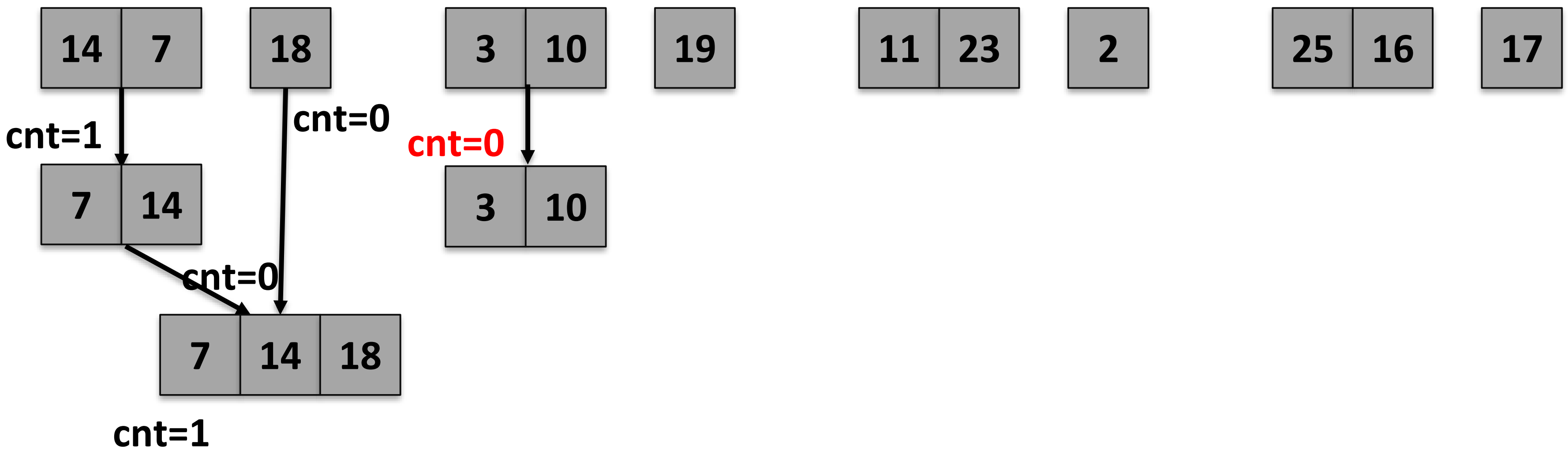
Example

Conquer



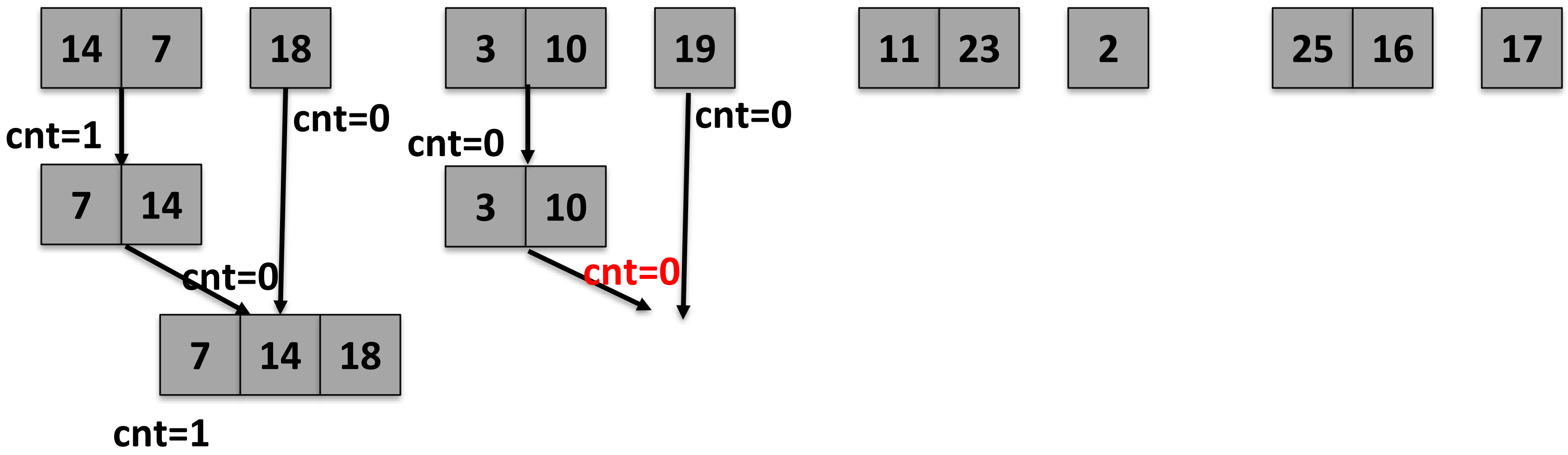
Example

Conquer



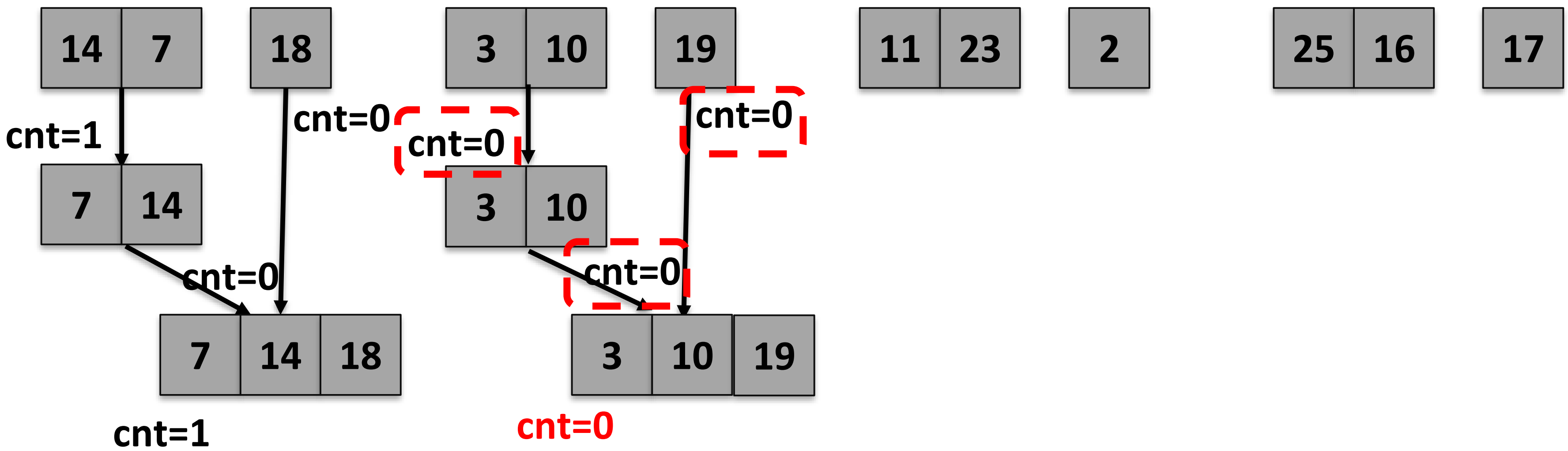
Example

Conquer



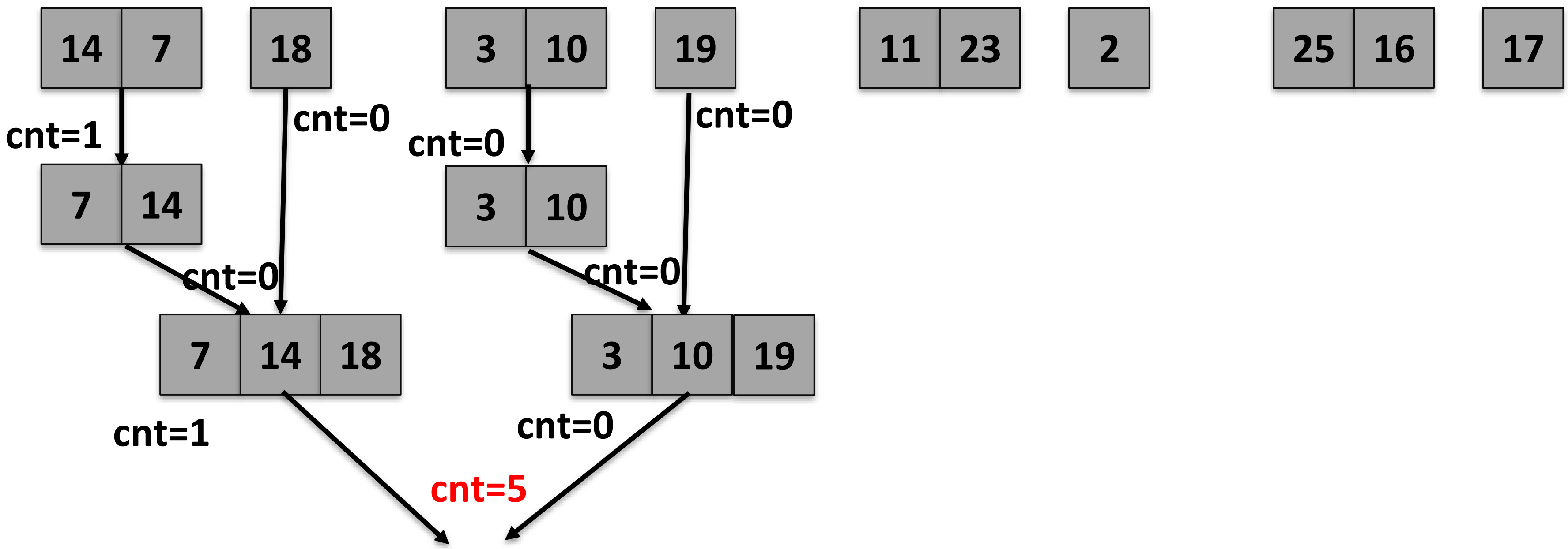
Example

Conquer



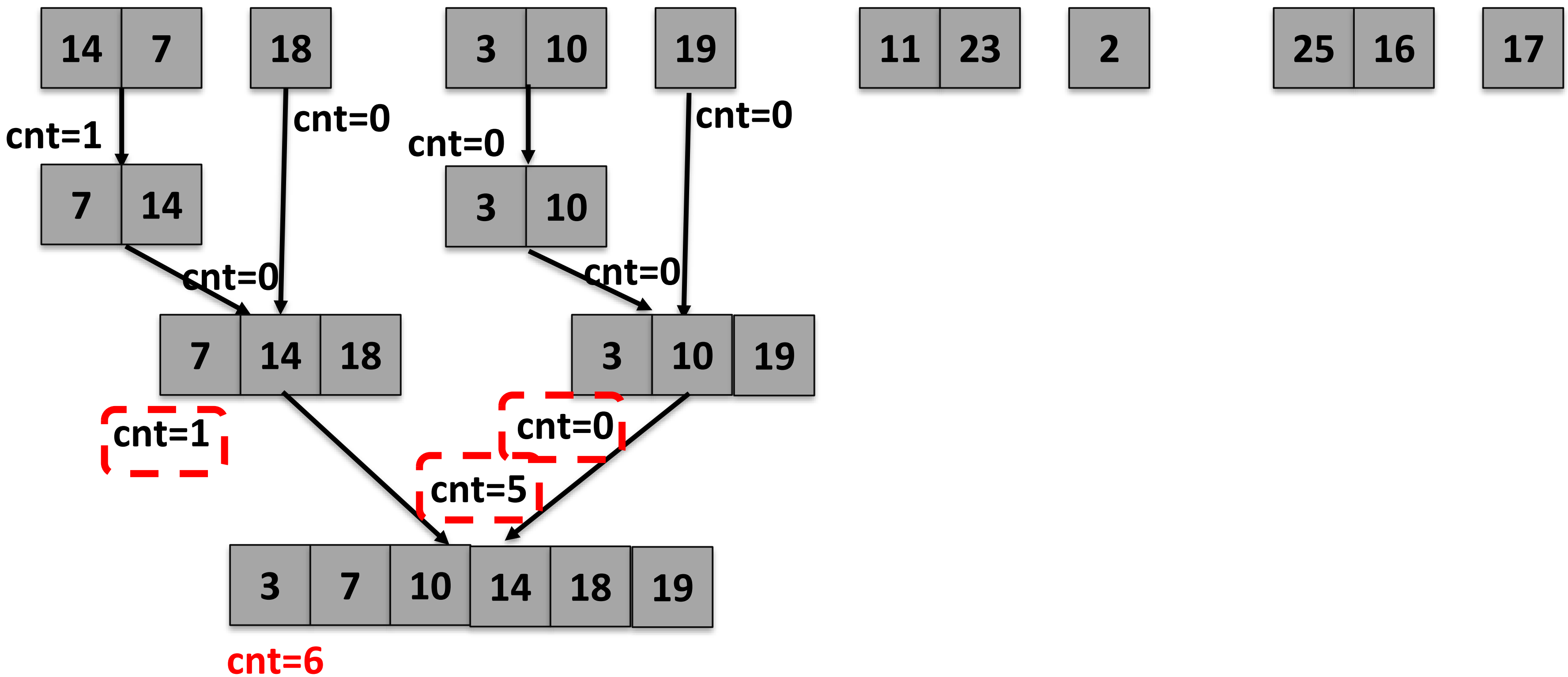
Example

Conquer



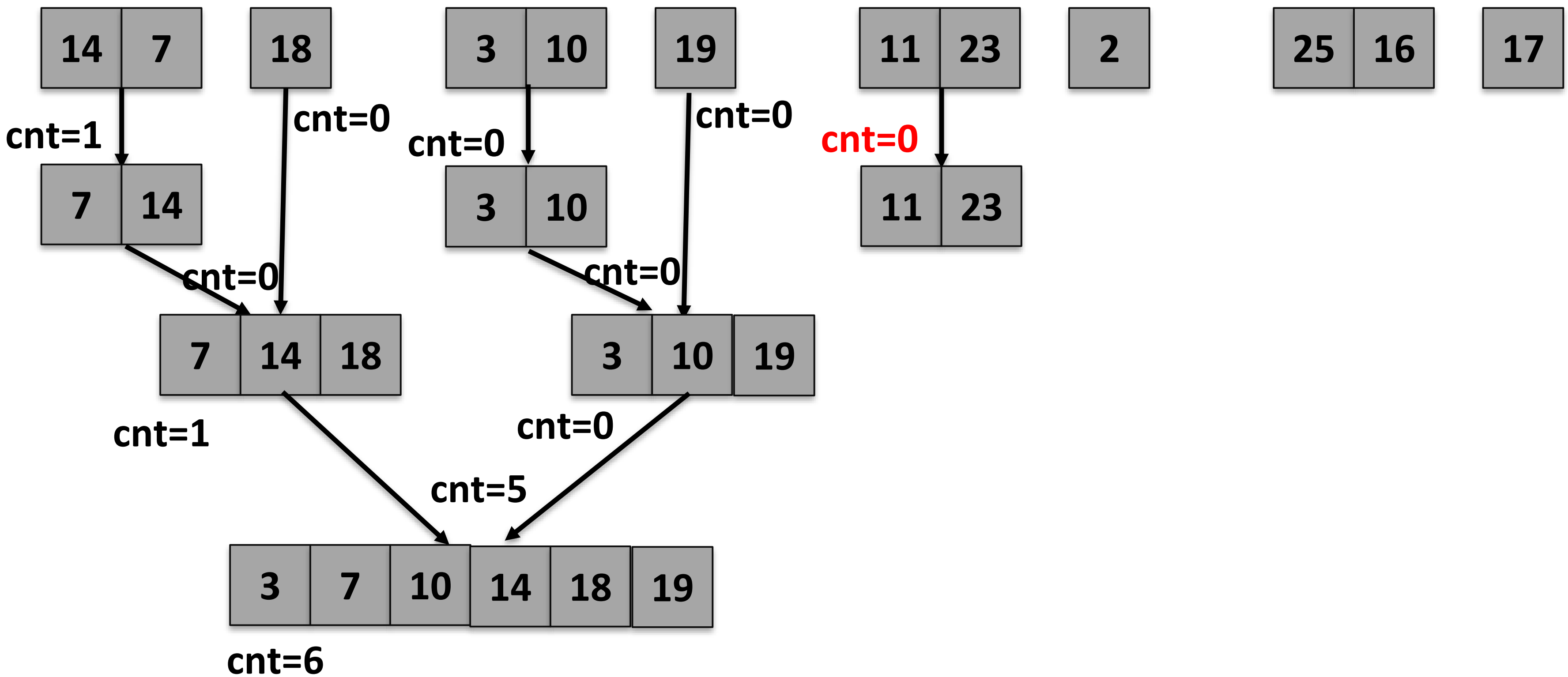
Example

Conquer



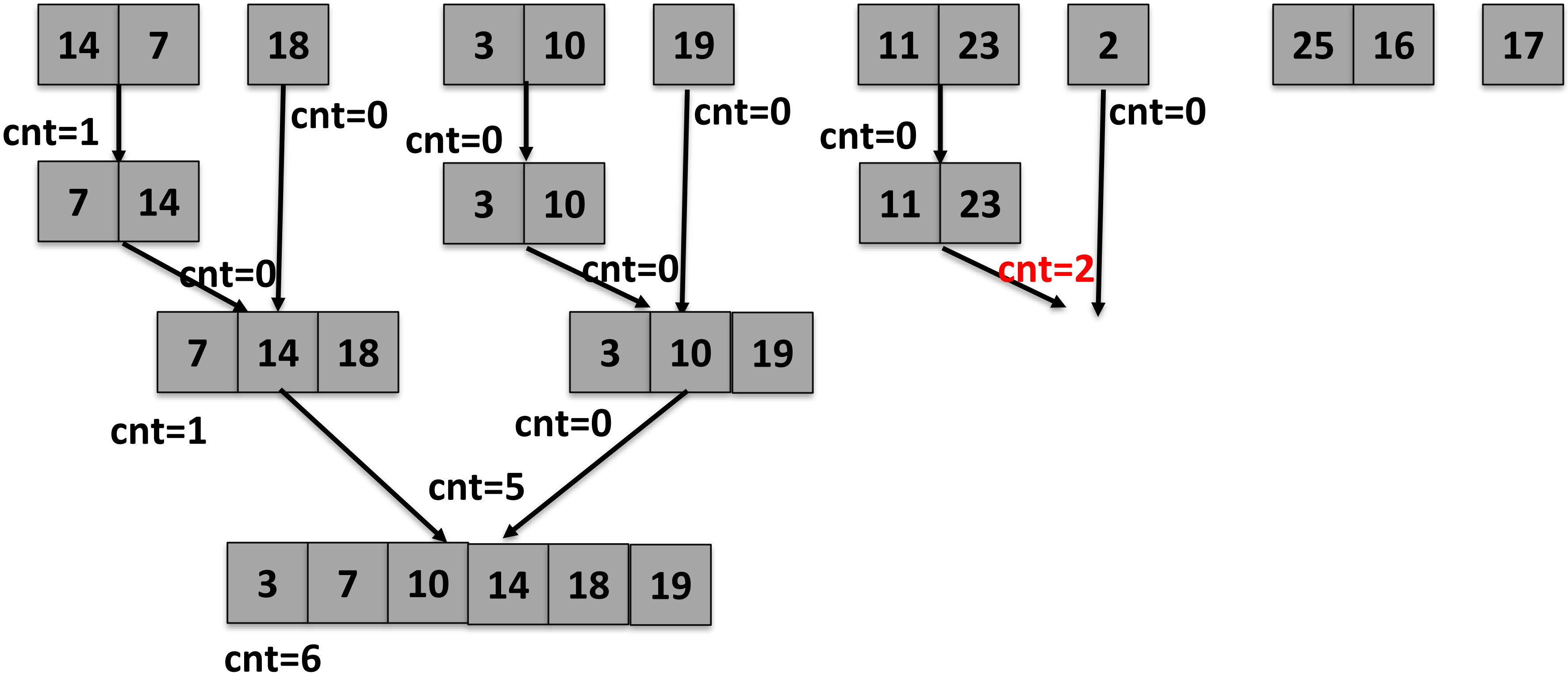
Example

Conquer



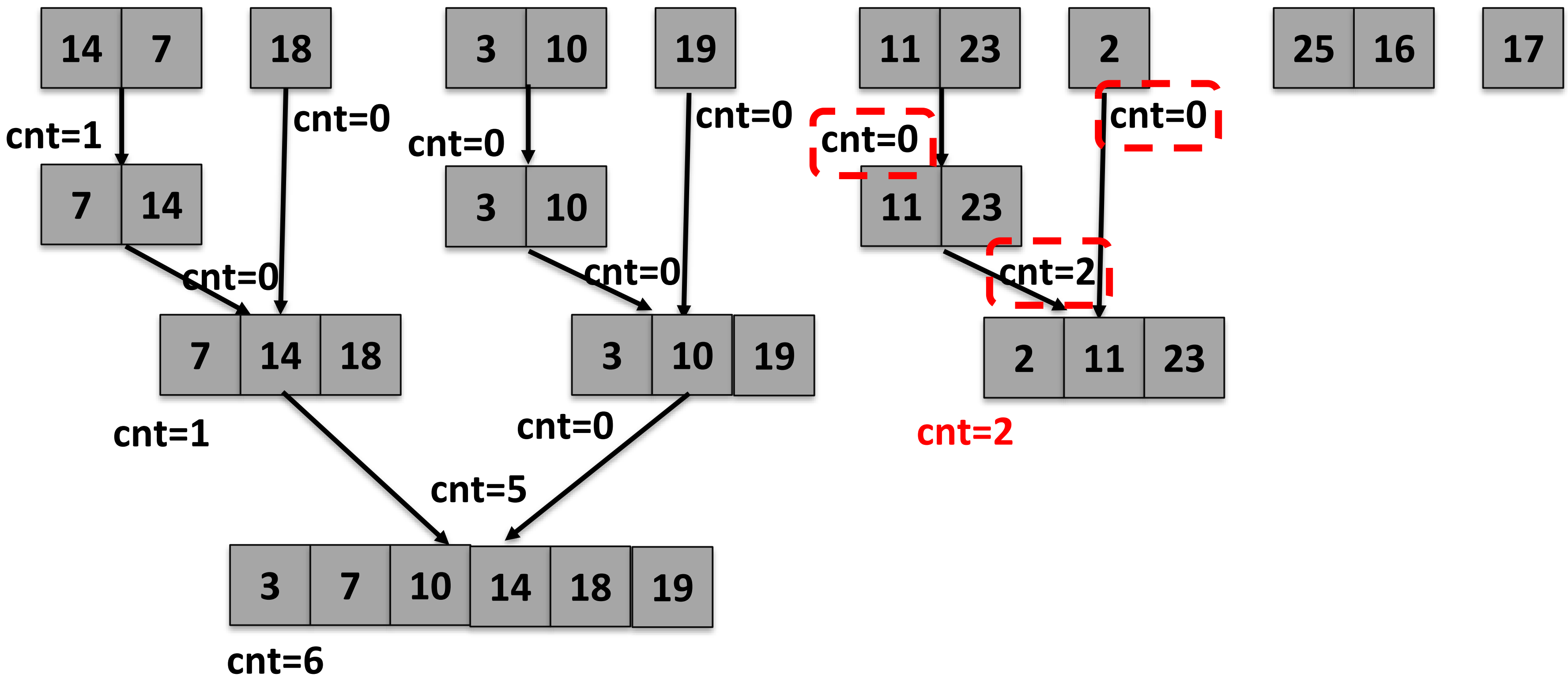
Example

Conquer



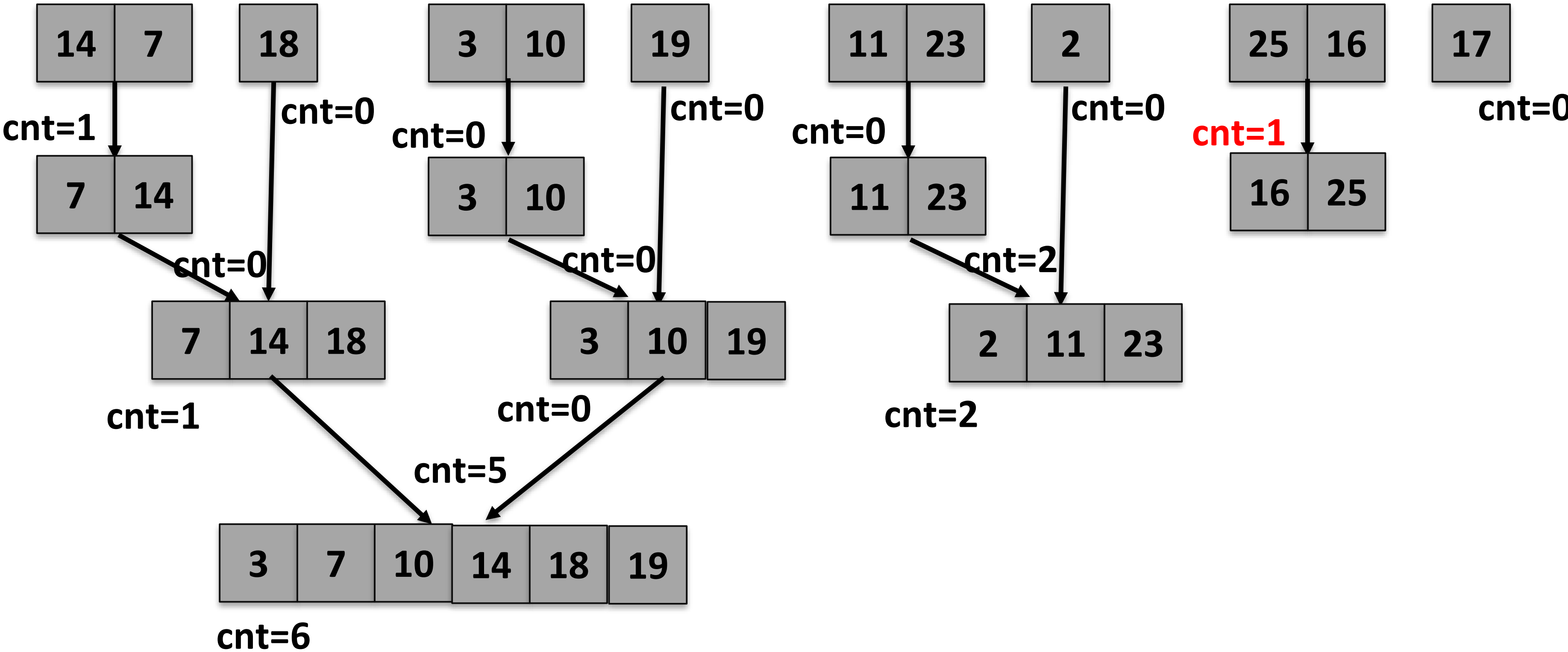
Example

Conquer



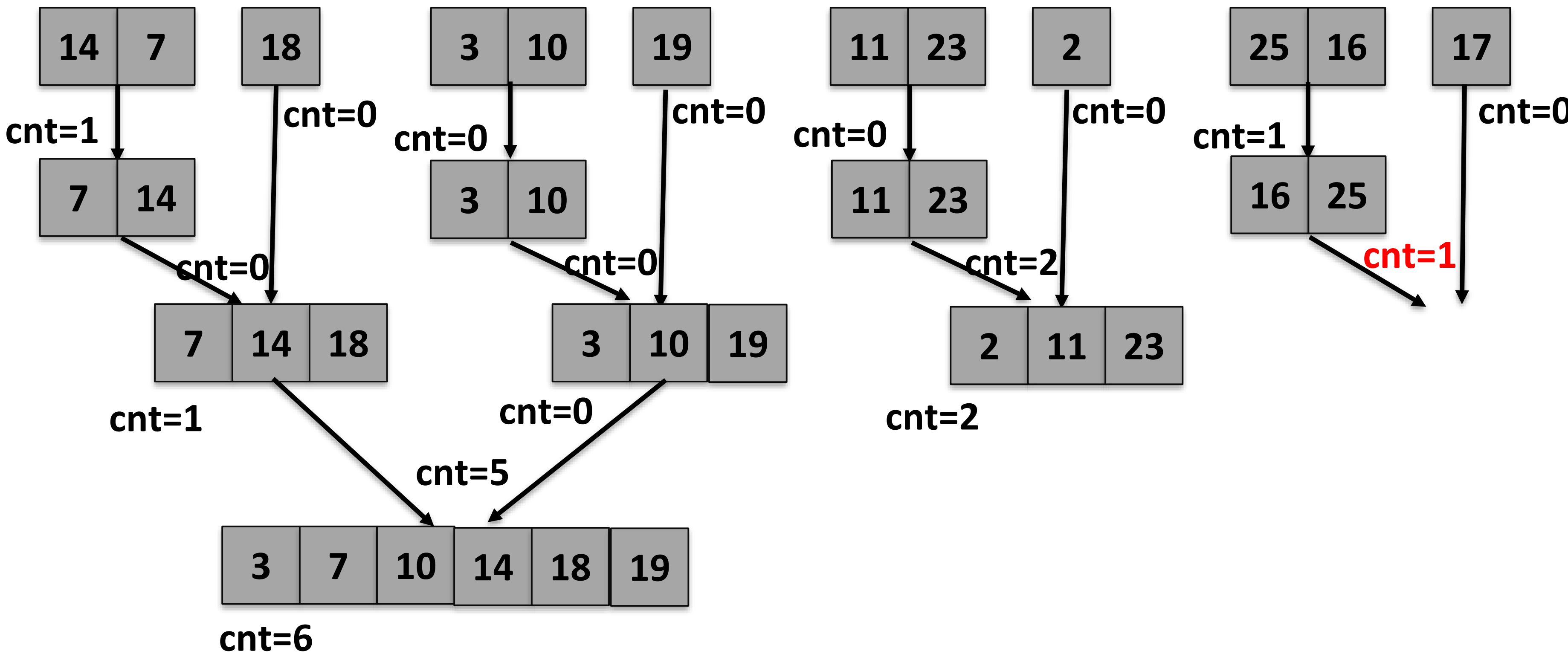
Example

Conquer



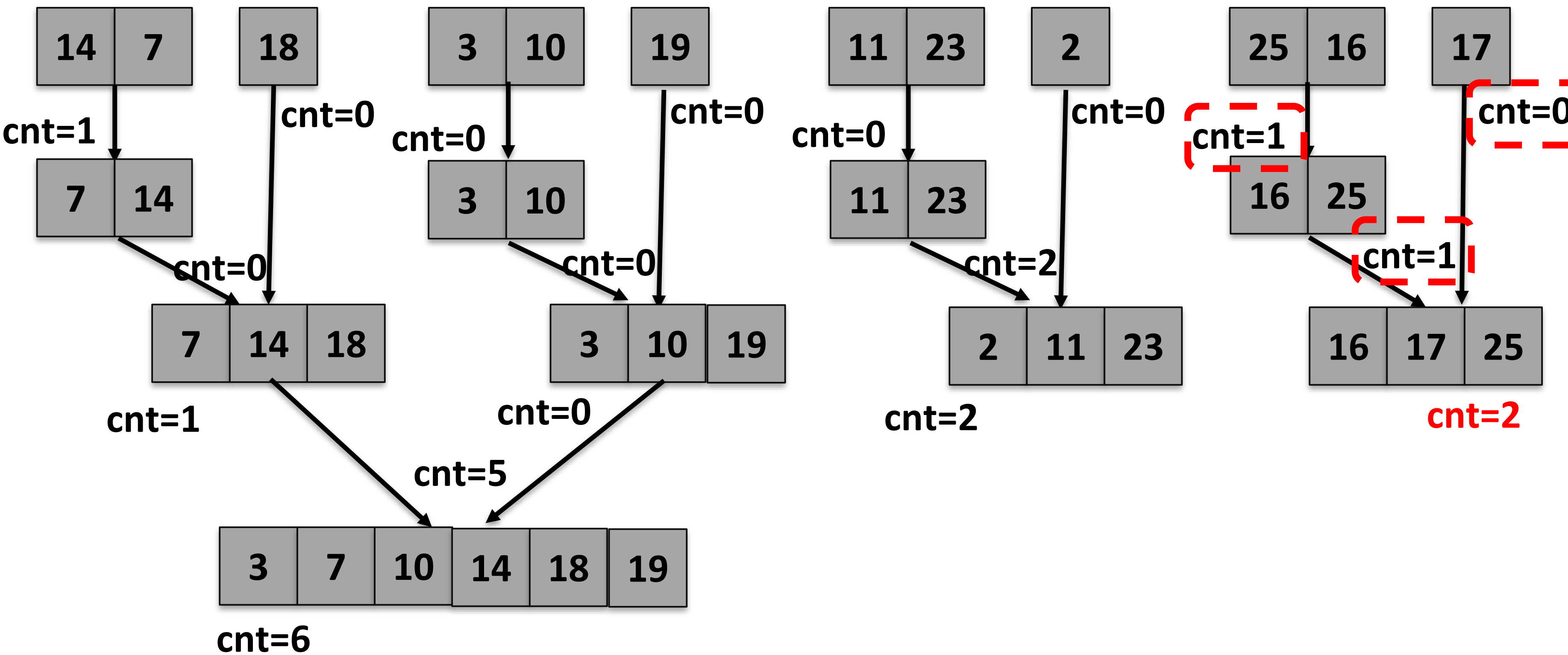
Example

Conquer



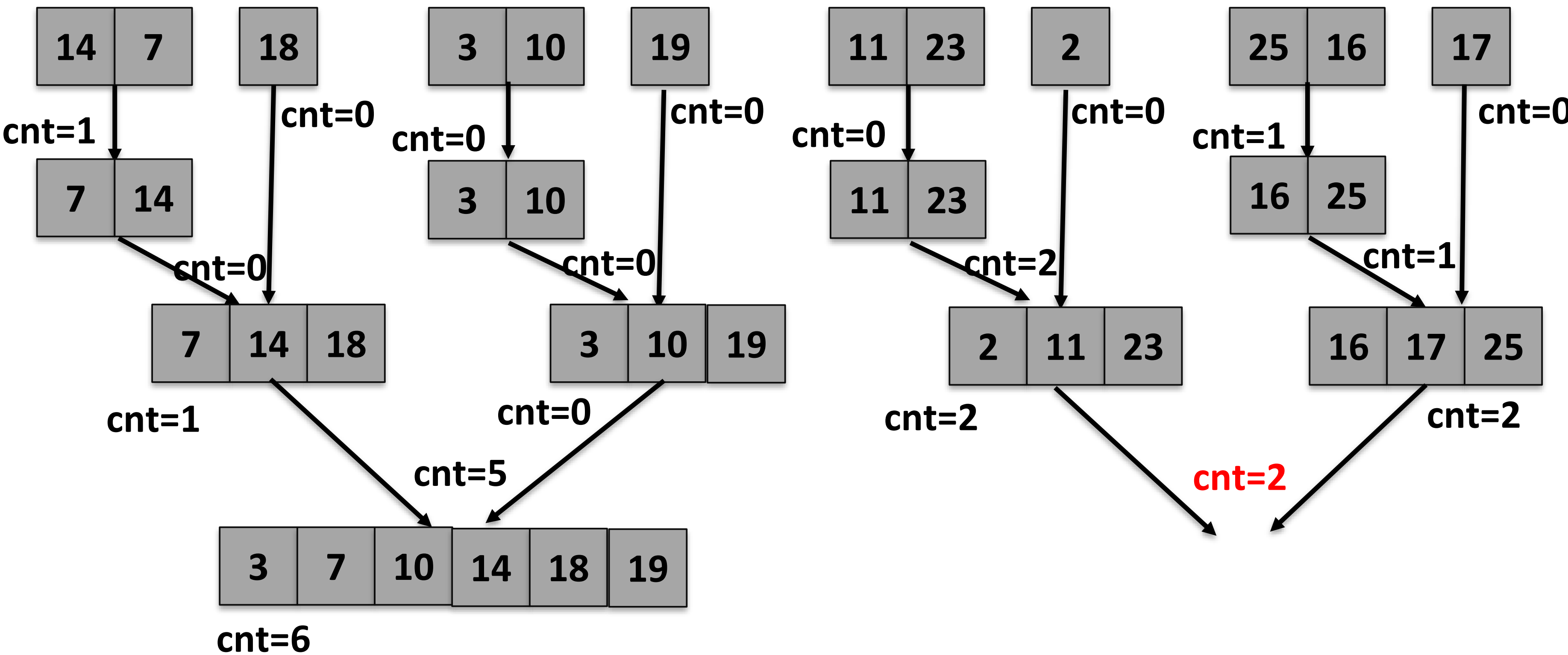
Example

Conquer



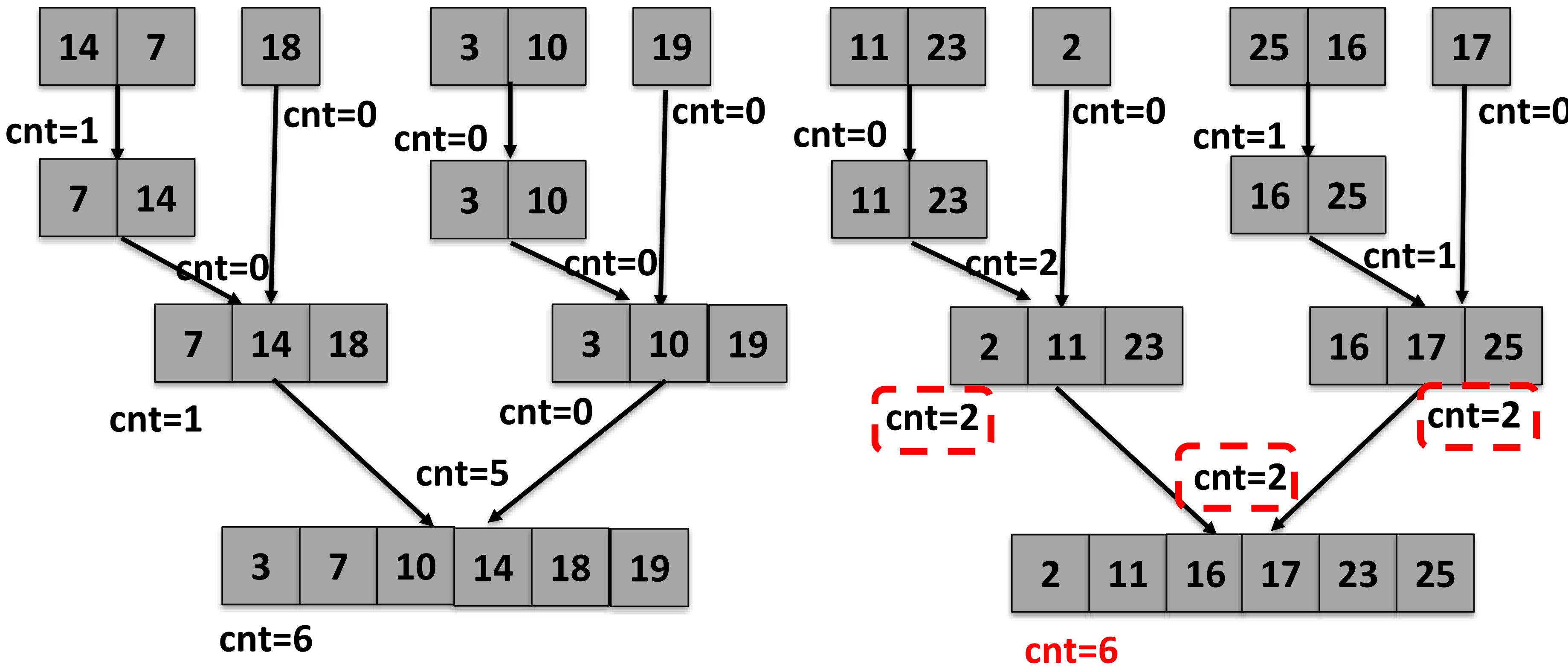
Example

Conquer



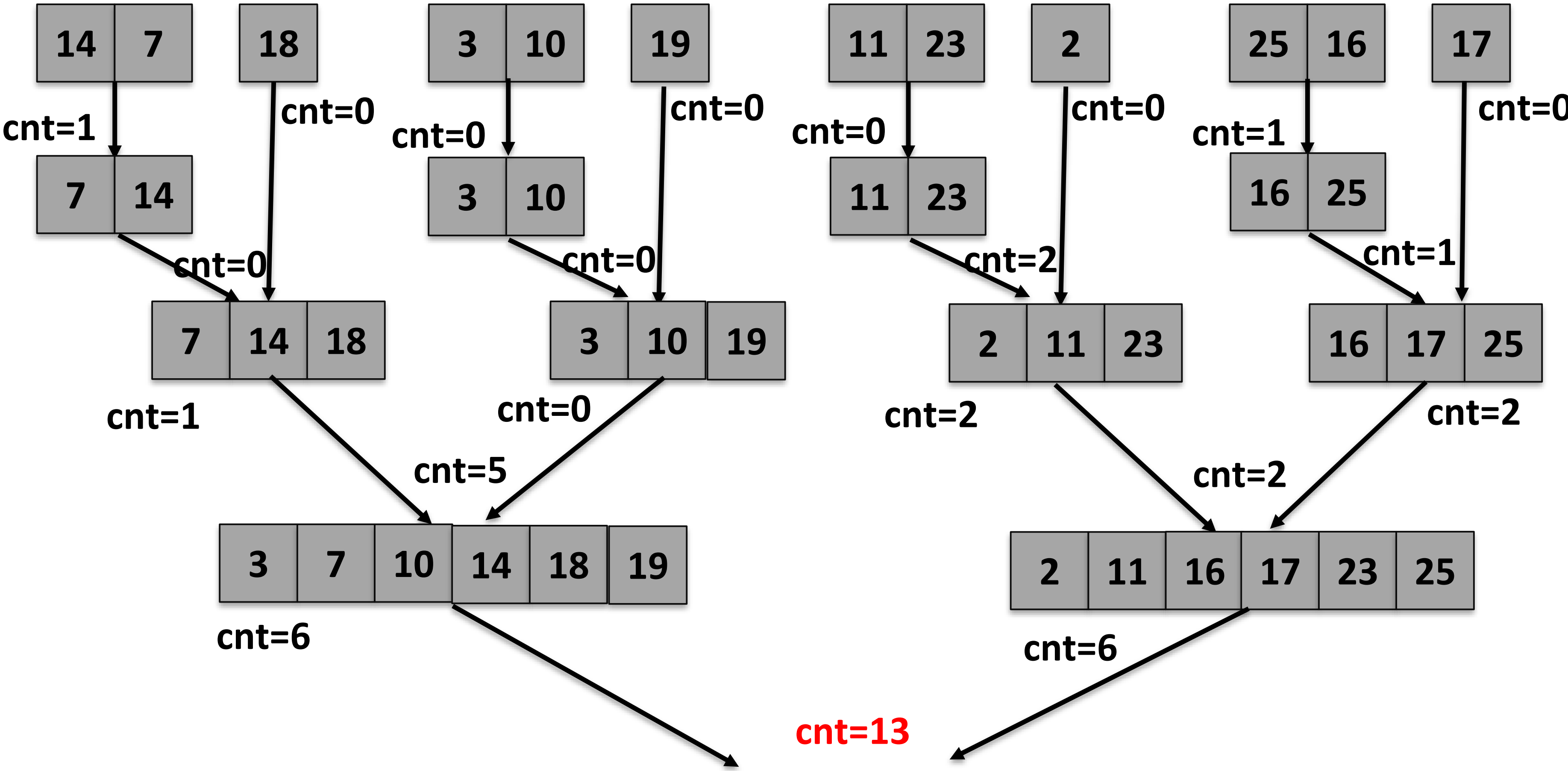
Example

Conquer



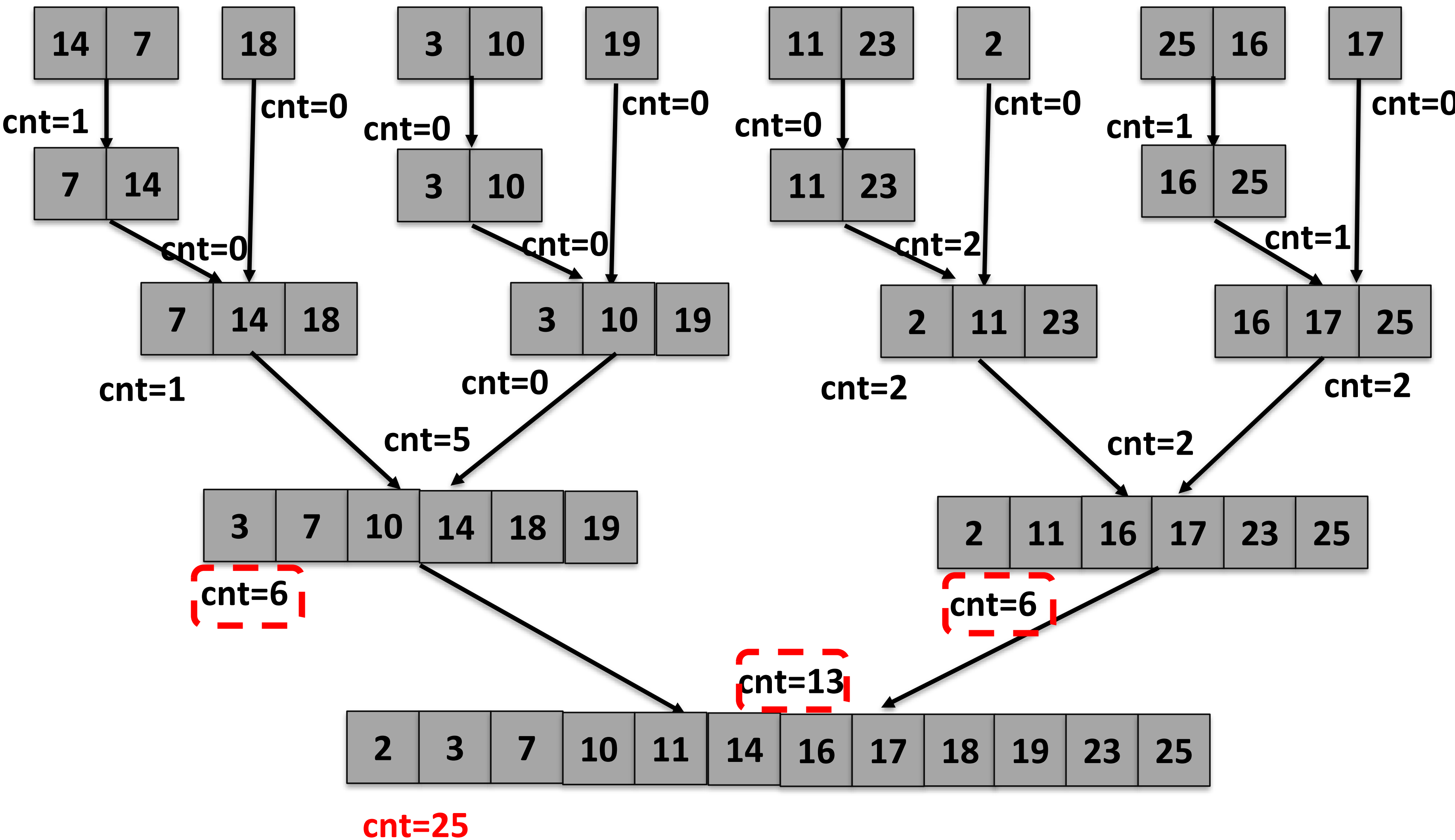
Example

Conquer



Example

Conquer



Outline

- Review to Divide-and-Conquer Paradigm
- 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
- Counting Inversion Problem
  - Problem definition
  - A brute force algorithm
  - A 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:

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


dank u
ju faleminderit
Tack
谢谢
Tak
mulţumesc
Asante
kiitos
Gracias
Salamat!
Terima kasih
Aliquam
Merci
Dankie
Obrigado
ありがとう
köszönöm
grazie
Aliquam
Go raibh maith agat
děkuii
Thank you
gam