INSERTION SORT, MERGE SORT & RECURRENCES

Juan Mendivelso

Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

CONTENTS

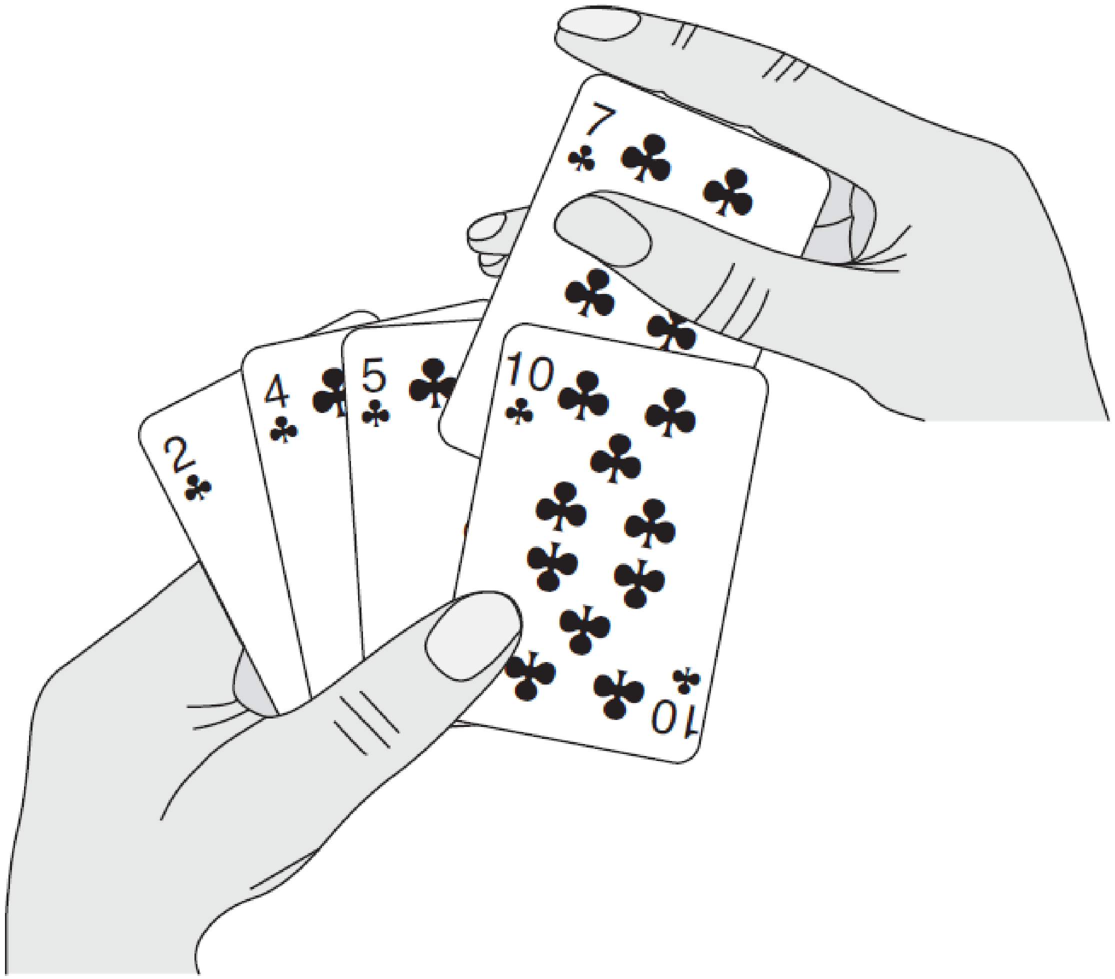
1. INSERTION SORT

SORTING PROBLEM

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

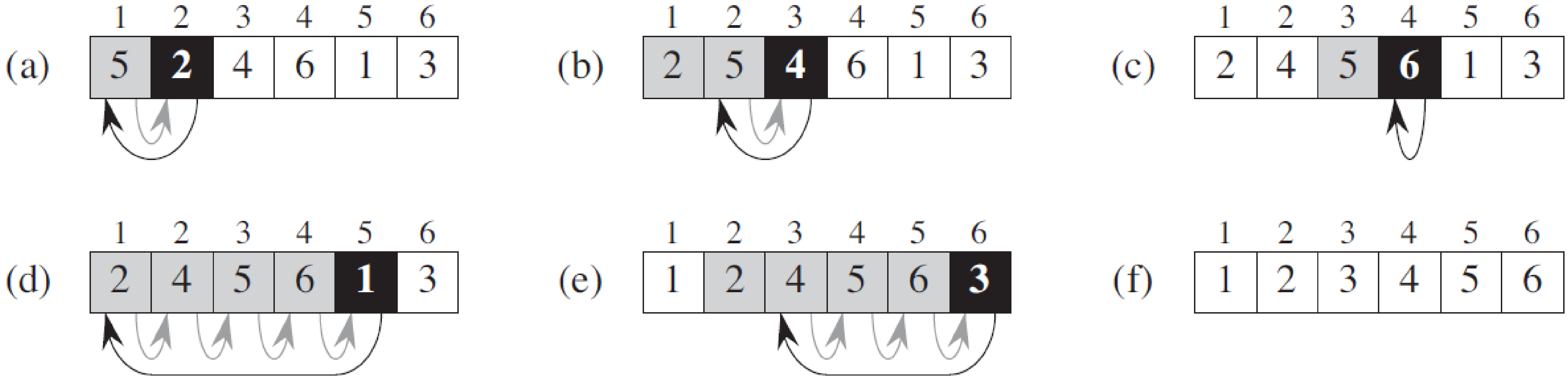
Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

INSERTION SORT ON CARDS



Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

INSERTION SORT EXAMPLE

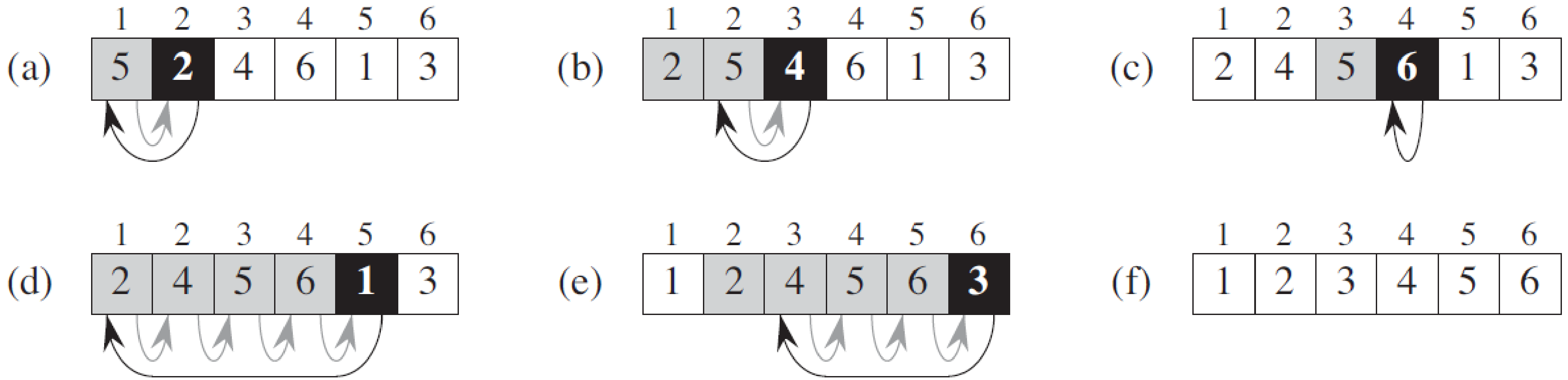


INSERTION SORT PSEUDOCODE

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

INSERTION SORT PSEUDOCODE

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

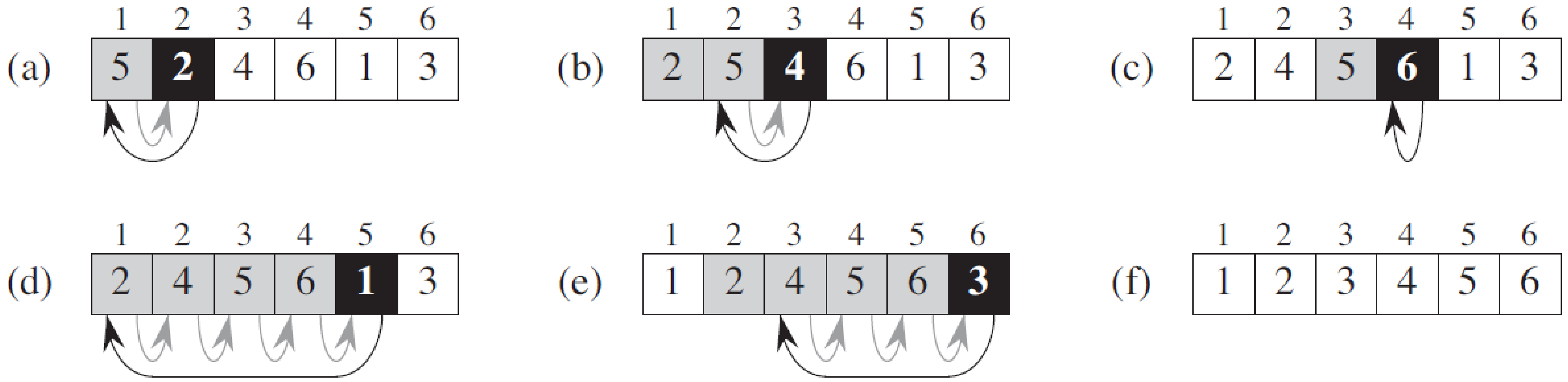


LOOP INVARIANT

At the start of each iteration of the **for** loop of lines 1–8, the subarray $A[1\,..\,j-1]$ consists of the elements originally in $A[1\,..\,j-1]$, but in sorted order.

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j - 1].
4      i = j - 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i - 1
8      A[i + 1] = key
```



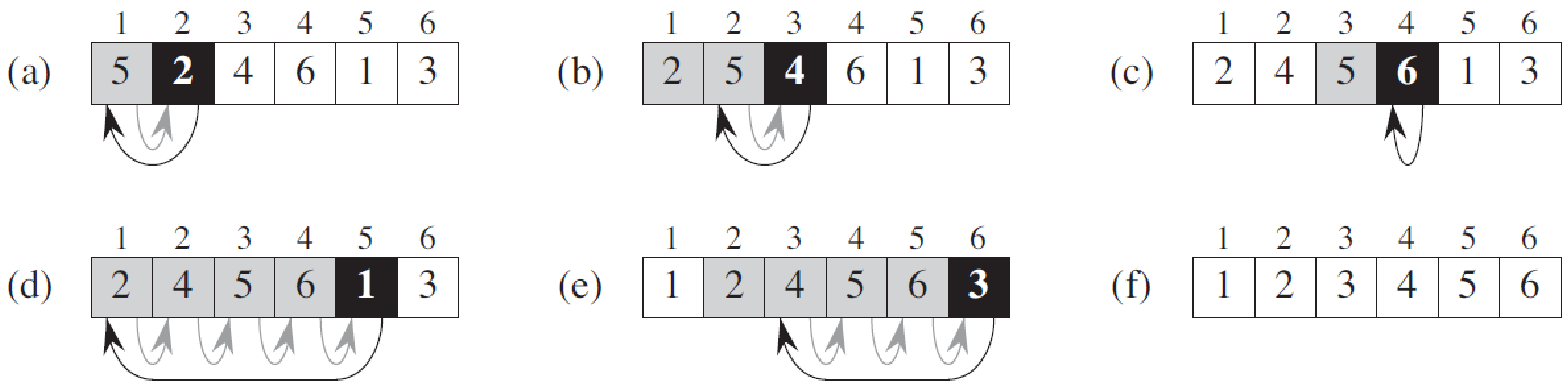
LOOP INVARIANT

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

At the start of each iteration of the **for** loop of lines 1–8, the subarray $A[1 \,..\, j-1]$ consists of the elements originally in $A[1 \,..\, j-1]$, but in sorted order.

- Initialization? Maintainance & Termination?



SPACE REQUIRED

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```


Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

SPACE REQUIRED

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

- $\Theta(1)$
- in-place

RUNNING TIME

- t_j:number of times line 5 is executed for of j

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
4 $i = j - 1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

GENERAL CASE

- t_j:number of times line 5 is executed for of j

INSERTION-SORT(*A*)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
4 $i = j - 1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

BEST CASE

INSERTION-SORT(A)

		cost	times
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

- t_j:number of times line 5 is executed for of j
- t_j=1

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

WORST CASE

- t_j:number of times line 5 is executed for of j
- t_j= j

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) \,.$$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \,. \end{aligned}$$

AVERAGE CASE

INSERTION-SORT(A)

		cost	times
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

- t_j:number of times line 5 is executed for of j
- t_j= j/2

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

T(n)=???

ASYMPTOTIC ANALYSIS

- Let T(n) denote the running time of Insertion Sort.
- Fill the following table by determining, in each cell, which Δ in {θ,Ω,O,ω,o} will make the expression T(n) = Δ(f(n)) true.

Case/ f(n)	Δ(1)	Δ(n)	Δ(n^2)	Δ(n^3)
Best Case				
Worst Case				
Average Case				
General Case				

ASYMPTOTIC ANALYSIS

- Let T(n) denote the running time of Insertion Sort.
- Fill the following table by determining, in each cell, which Δ in {θ,Ω,O,ω,o} will make the expression T(n) = Δ(f(n)) true.

Case/ f(n)	Δ(1)	Δ(n)	$\Delta(n^2)$	$\Delta(n^3)$
Best Case				
Worst Case				
Average Case				
General Case				

ANALYSIS OF ALGORITHMS

- What is the complexity of the following algorithm in the best, worst and general case?

```
Misterio1(n){
	for i=1 to n{
		k = i
		while k > 1{
			k=k/2
		}
	}
}
```

ANALYSIS OF ALGORITHMS

- What is the complexity of the following algorithm in the best, worst and general case?

```
Misterio2(n){
	for i=2 to n{
		InsertionSort(A,i)
	}
}
```

2. MERGE SORT

Divide & Conquer

- **Divide** the problem into subproblems.
- **Conquer** the subproblems by solving them recursively.
- **Combine** the solution of such problems to get the solution of the original problem.

Divide & Conquer in Merge Sort

- **Divide** the array to sort into two subarrays.
- **Conquer** by sorting such subarrays recursively.
- **Combine** such sorted subarrays into a sorted array by using the **Merge** procedure.

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

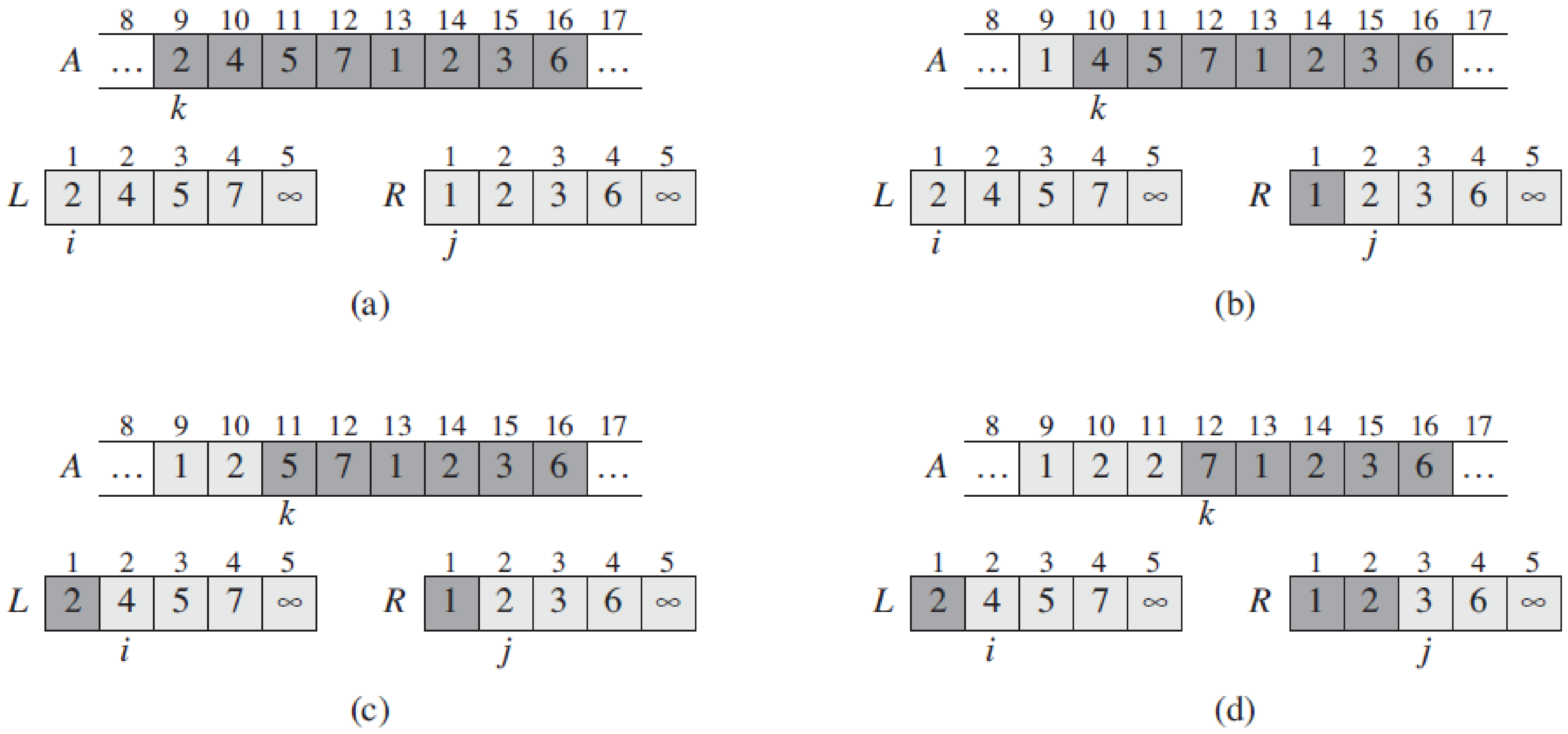

Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

Example of Merge Sort

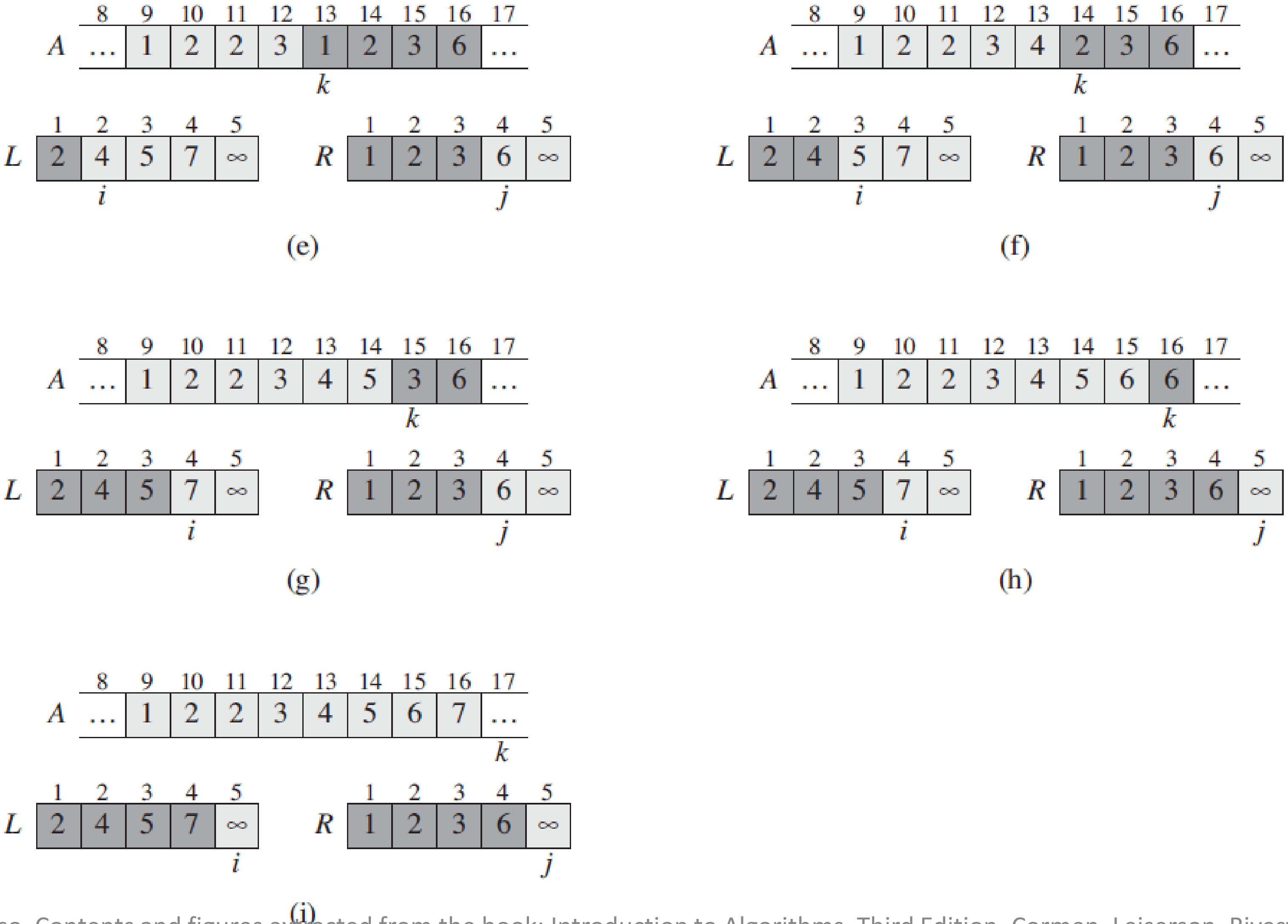
- Let's consider the array 5,2,4,7,1,3,2,6
- **Divide** p=1, r=8, q=4. The subarrays are 5,2,4,7 and 1,3,2,6.
- **Conquer** those subarrays recursively: 2,4,5,7 and 1,2,3,6
- **Combine** such sorted subarrays **Merge** procedure.

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

Example of Merge



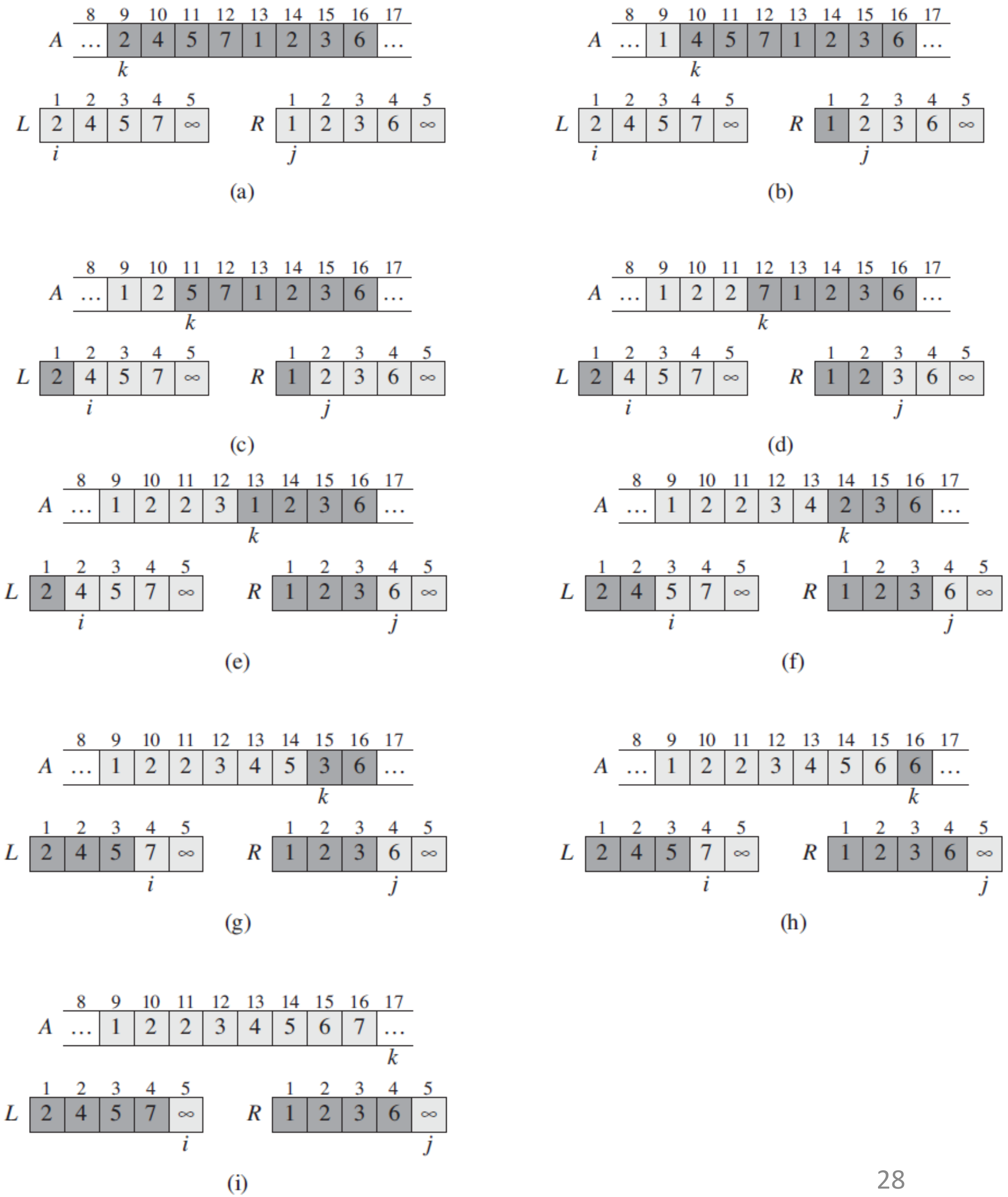
Example of Merge



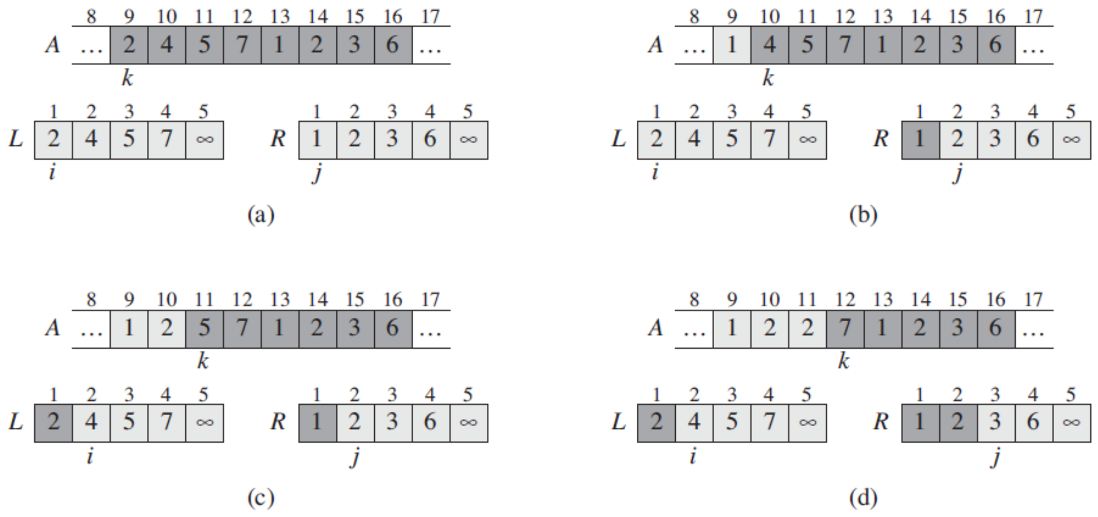
Merge

MERGE(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```



Loop Invariant

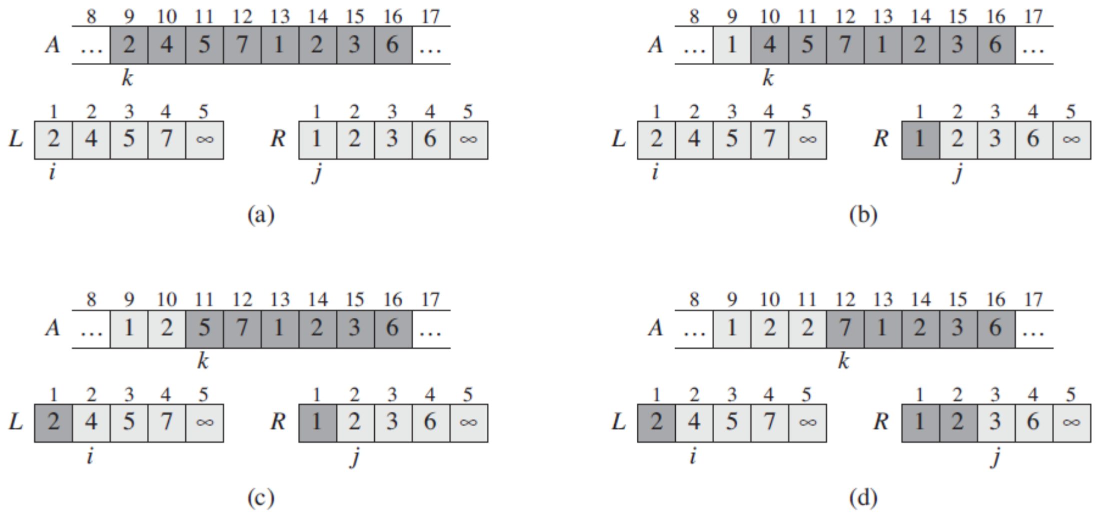


MERGE(A, p, q, r)

```
1   n1 = q - p + 1
2   n2 = r - q
3   let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
4   for i = 1 to n1
5       L[i] = A[p + i - 1]
6   for j = 1 to n2
7       R[j] = A[q + j]
8   L[n1 + 1] = ∞
9   R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

At the start of each iteration of the **for** loop of lines 12–17, the subarray $A[p..k-1]$ contains the $k-p$ smallest elements of $L[1..n_1+1]$ and $R[1..n_2+1]$, in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

Loop Invariant

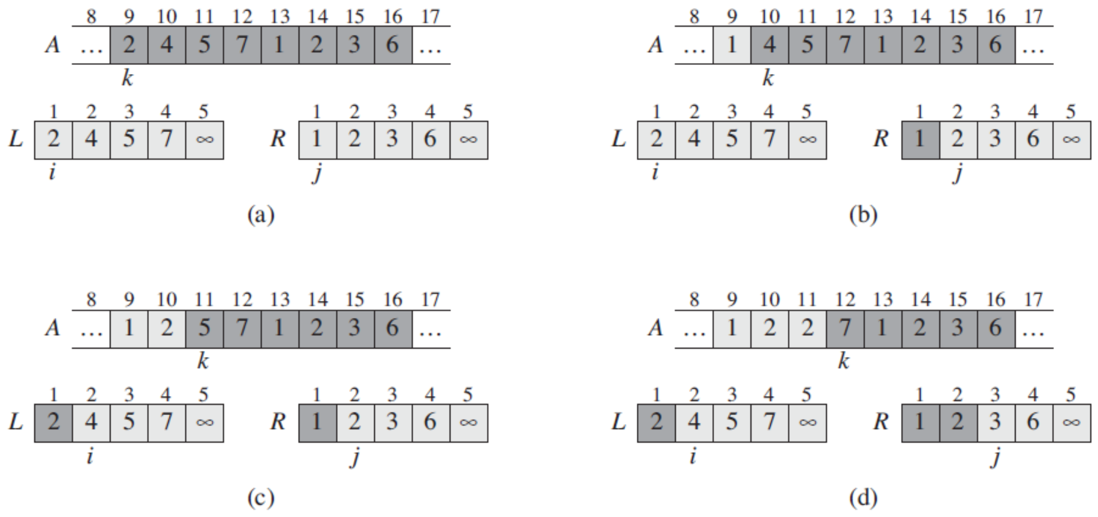



```
MERGE(A, p, q, r)
1   n1 = q − p + 1
2   n2 = r − q
3   let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
4   for i = 1 to n1
5       L[i] = A[p + i − 1]
6   for j = 1 to n2
7       R[j] = A[q + j]
8   L[n1 + 1] = ∞
9   R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]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

At the start of each iteration of the **for** loop of lines 12–17, the subarray $A[p \mathinner{..} k-1]$ contains the $k - p$ smallest elements of $L[1 \mathinner{..} n_1 + 1]$ and $R[1 \mathinner{..} n_2 + 1]$, in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

- Initialization, Maintenance, Termination?

Space Complexity

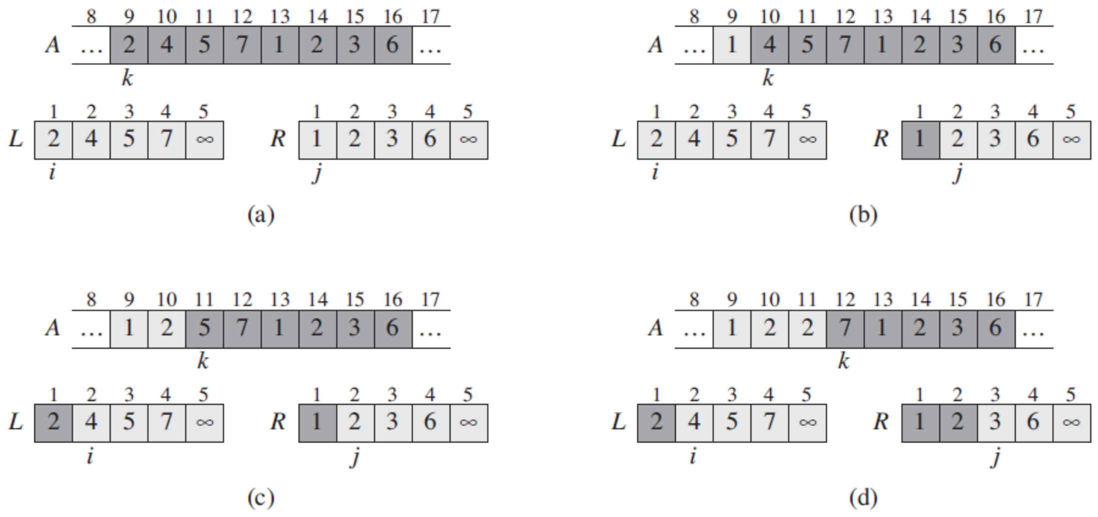


MERGE(A, p, q, r)

$n_1 = q - p + 1$
$n_2 = r - q$
let $L[1\,..\,n_1 + 1]$ and $R[1\,..\,n_2 + 1]$ be new arrays
for $i = 1$ **to** n_1
    $L[i] = A[p + i - 1]$
for $j = 1$ **to** n_2
    $R[j] = A[q + j]$
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$
for $k = p$ **to** r
    if $L[i] \le R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else $A[k] = R[j]$
        $j = j + 1$

- Space complexity of Merge?
- In-place?

Space Complexity



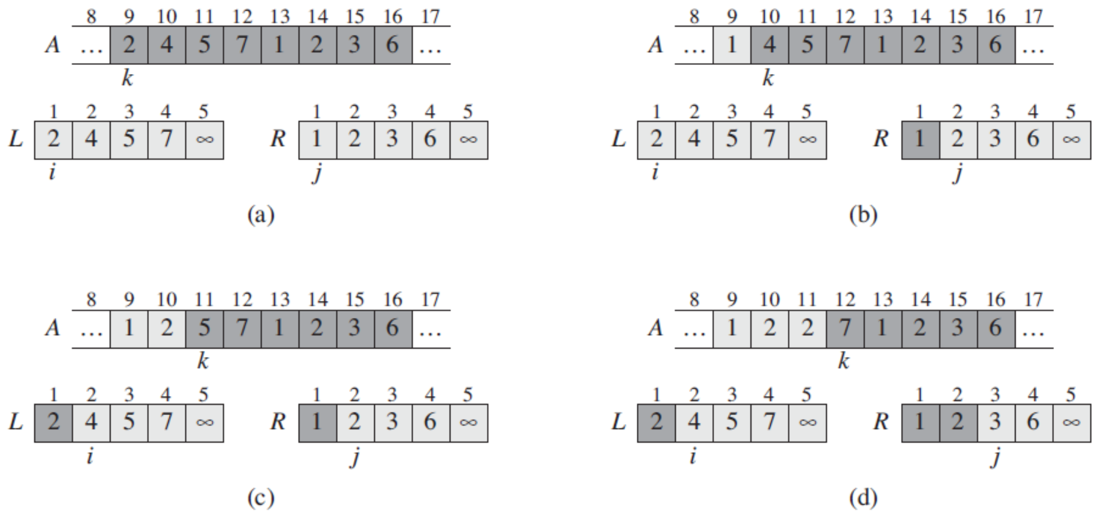
MERGE(A, p, q, r)

```
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

- Space complexity of Merge?
  - Θ(n)
- In-place?
  - No

Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

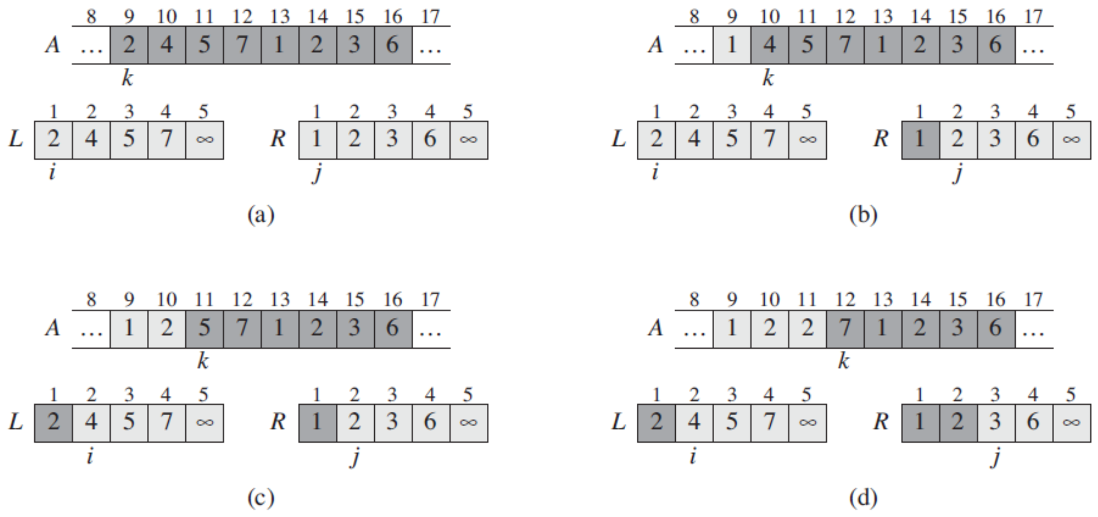
Time Complexity




```
MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

- Best case?
- Wort case?
- General Case?

Time Complexity

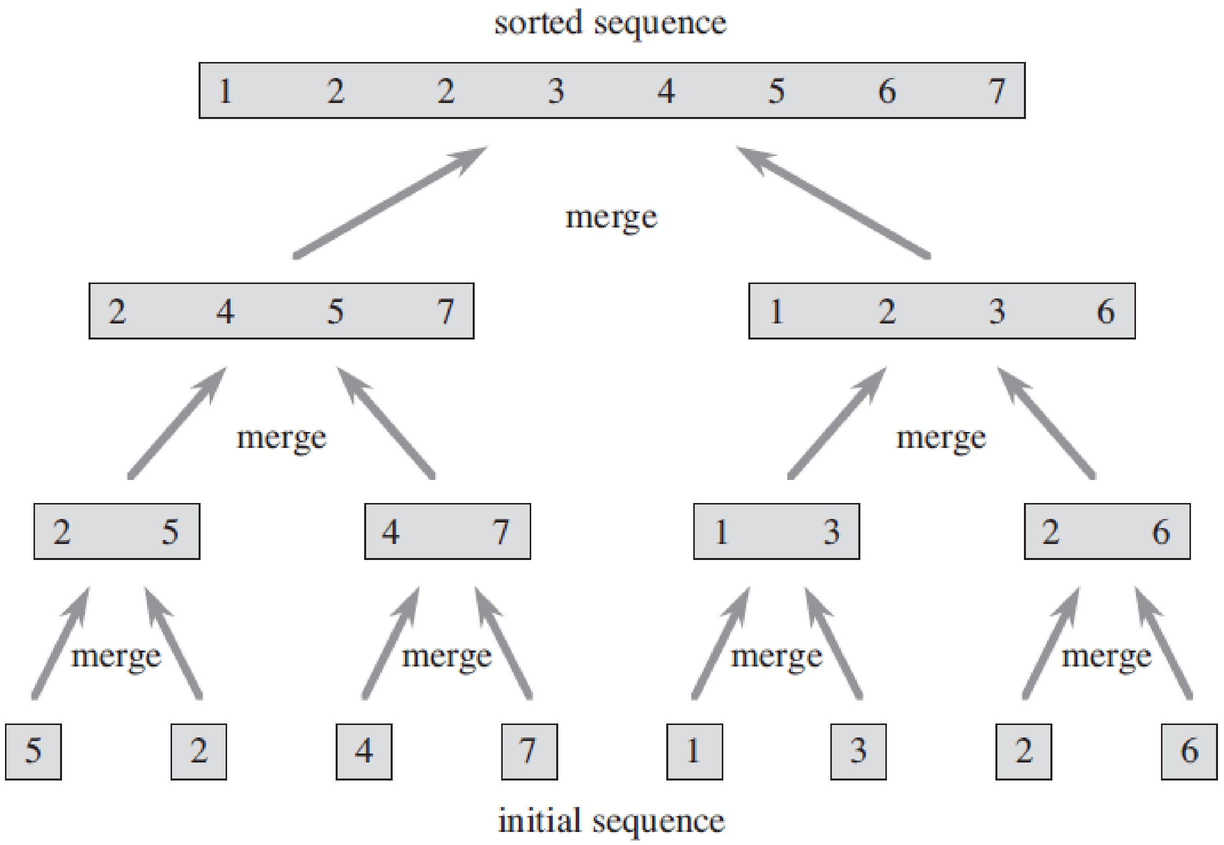


```
MERGE(A, p, q, r)
n₁ = q − p + 1
n₂ = r − q
let L[1 .. n₁ + 1] and R[1 .. n₂ + 1] be new arrays
for i = 1 to n₁
    L[i] = A[p + i − 1]
for j = 1 to n₂
    R[j] = A[q + j]
L[n₁ + 1] = ∞
R[n₂ + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

- Best case?
  - Θ(n)
- Wort case?
  - Θ(n)
- General Case?
  - Θ(n)

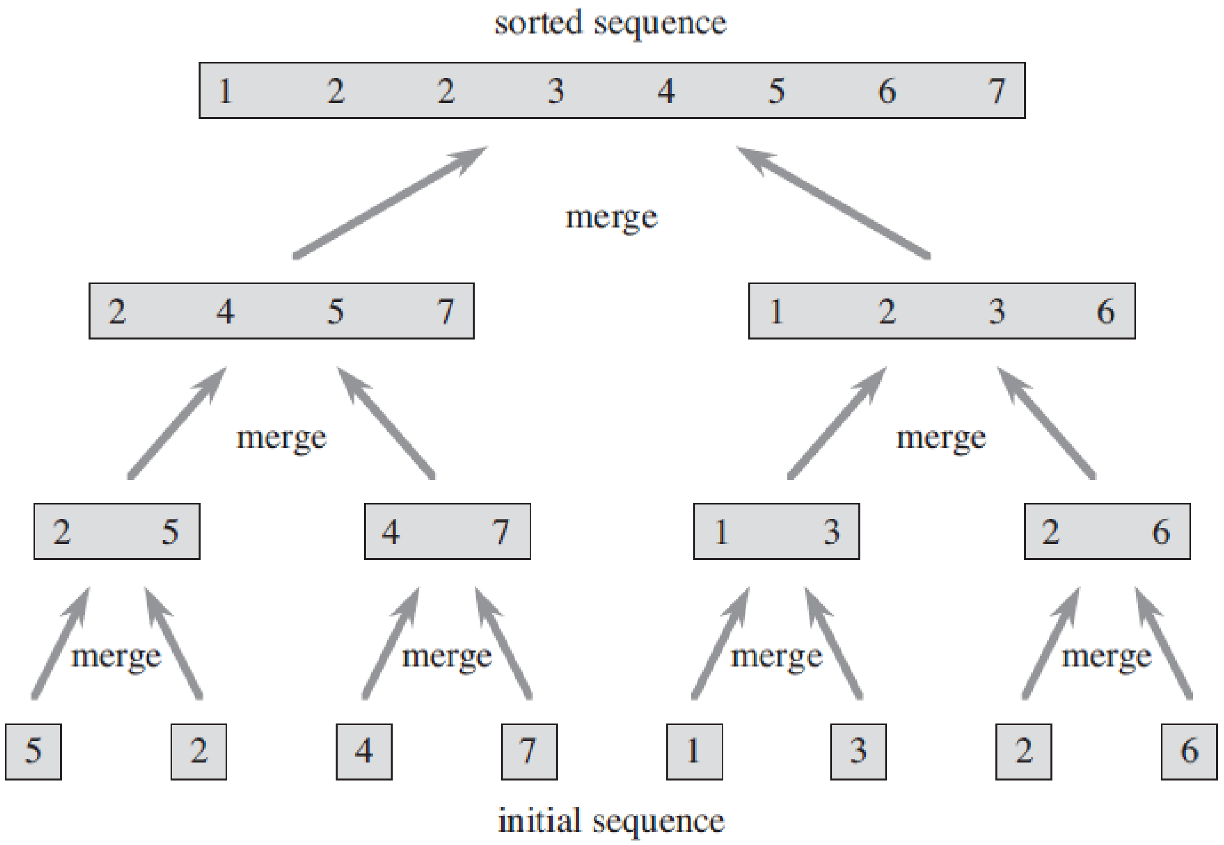
Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

Back to the Merge Sort Example




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

Running Time of Merge Sort



- How can we calculate it?

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

3. Recurrences

Recurrence

- Useful to calculate the complexity of an algorithm with recursive calls.
- The time for a length-n problem, T(n), is expressed in terms of the time of such problem for smaller inputs.
- For instance, for Divide & Conquer algorithms, the recurrence is:

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

- The division can have overlapping subproblems.
  - Example: n=18, a=5, b=3

More on Recurrences

- Every recurrence has a definition for the base case, but it is often omitted.
- It’s not necessarily defining even partitions.
- It doesn’t even have to be a fraction of the input.
- We can have inequalities of recurrences too.
- We often omit floors and ceilings.

Back to Merge Sort Running Time

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

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

Solution of Recurrences

- The recursion tree.
- The substitution method.
- The master method.

The recursion tree method on Divide & Conquer algorithms

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

- Draw a tree where the root is the cost of the independent (not recursive) cost of the original problem (of size n), i.e D(n)+C(n).
- The number of branches of each node is the number of subproblems in the recursion, i.e. **a**.
- The size of a child problem is the size of its parent divided by **b**.

The recursion tree method on Divide & Conquer algorithms

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

1. Draw a tree where
    - the root is the cost of the independent (not recursive) cost of the original problem (of size n), i.e D(n)+C(n).
    - The number of branches of each node is the number of subproblems in the recursion, i.e. **a**.
    - The size of a child problem is the size of its parent divided by **b**.

The recursion tree method on Divide & Conquer algorithms

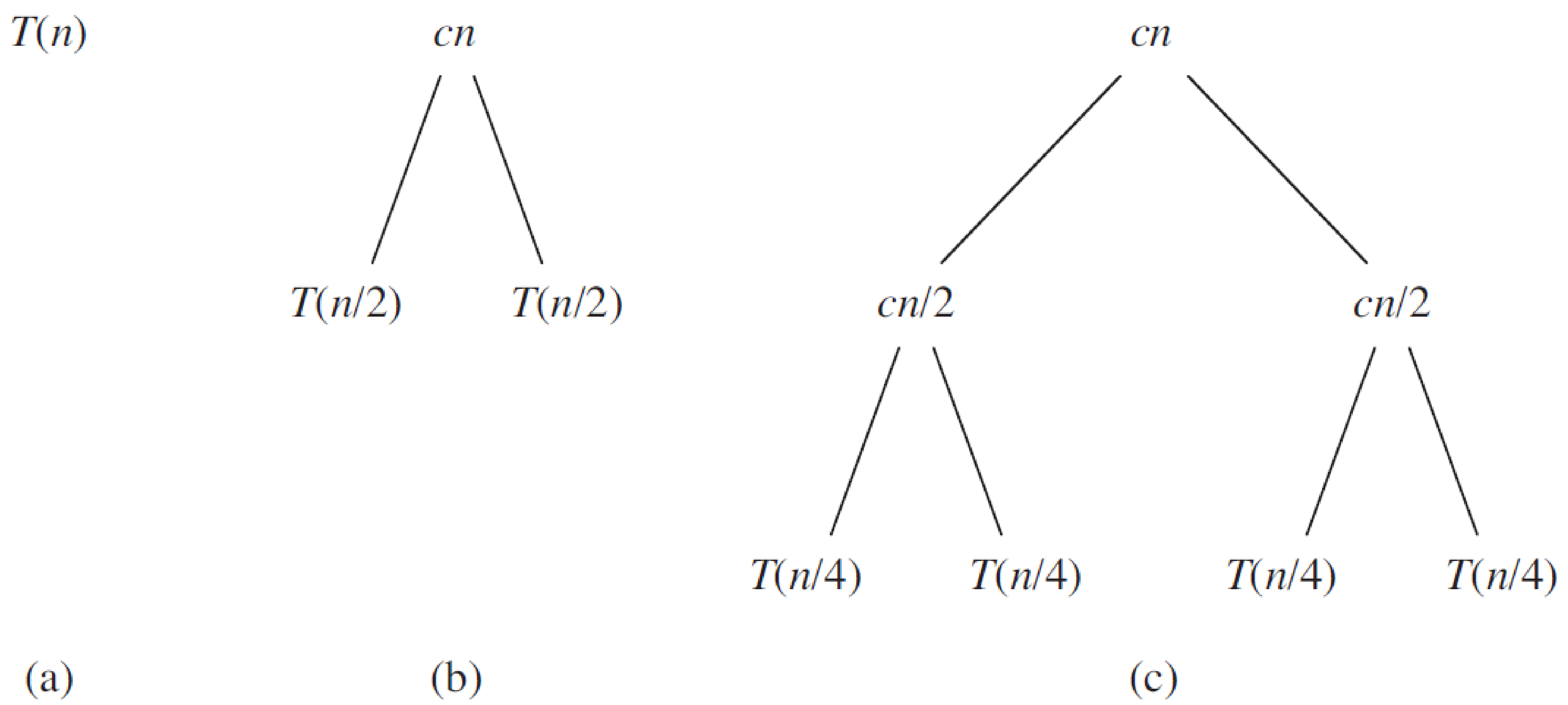
$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \text{ ,} \\ aT(n/b) + D(n) + C(n) & \text{otherwise .} \end{cases}$$

2. Find the following information about level i of the tree:
    - size of a subproblem
    - cost of a subproblem
    - number of subproblems
    - total cost
3. Find in which level the leaves (base case) are.
4. Find the recursive cost and the base case cost.
5. Choose the one that is larger.

Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

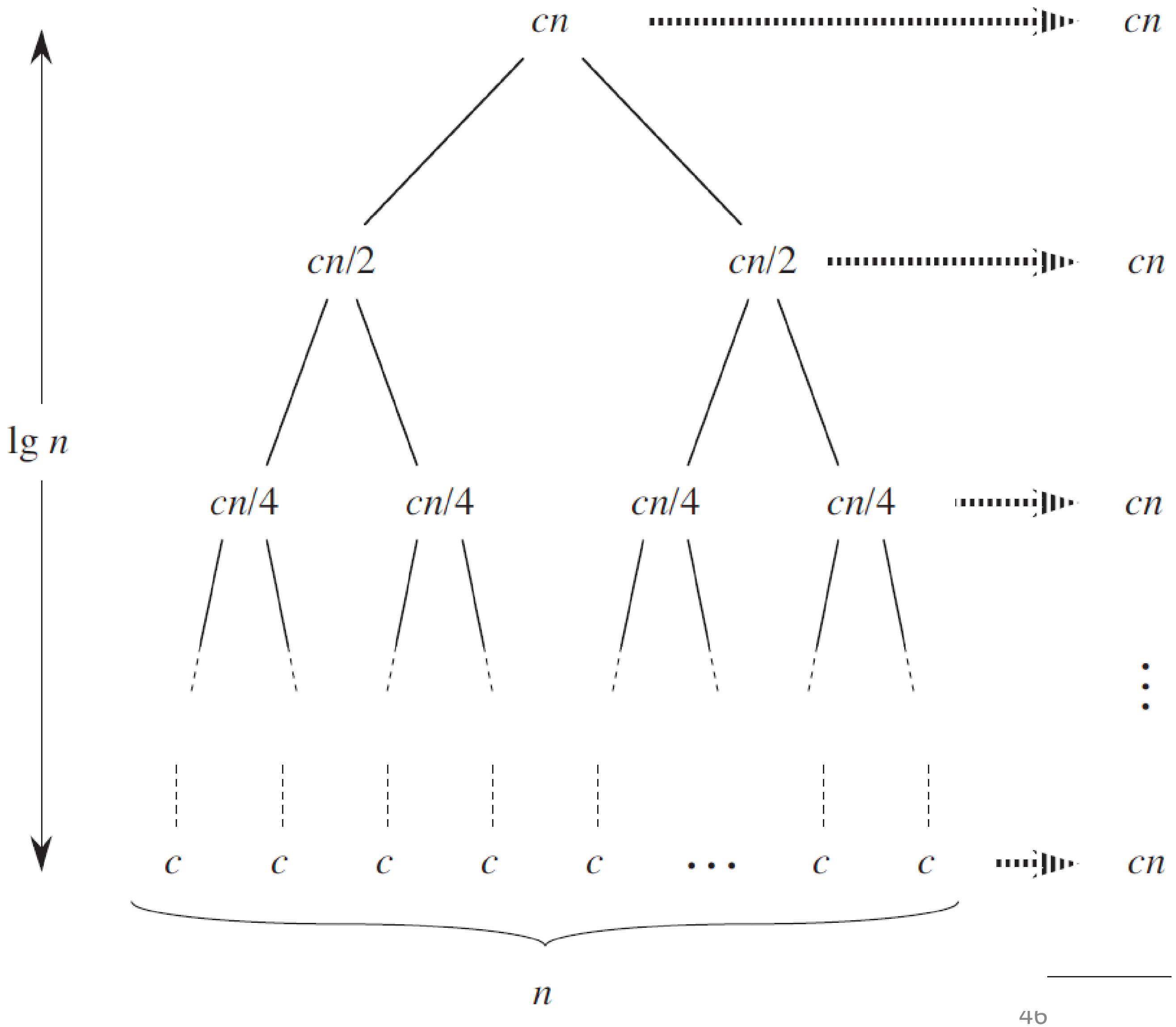
Recursion Tree for Merge Sort

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



Recursion Tree for Merge Sort

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



Find the following information about level i of the tree:

- size of a subproblem
- cost of a subproblem
- number of subproblems
- total cost

Find in which level the leaves (base case) are.

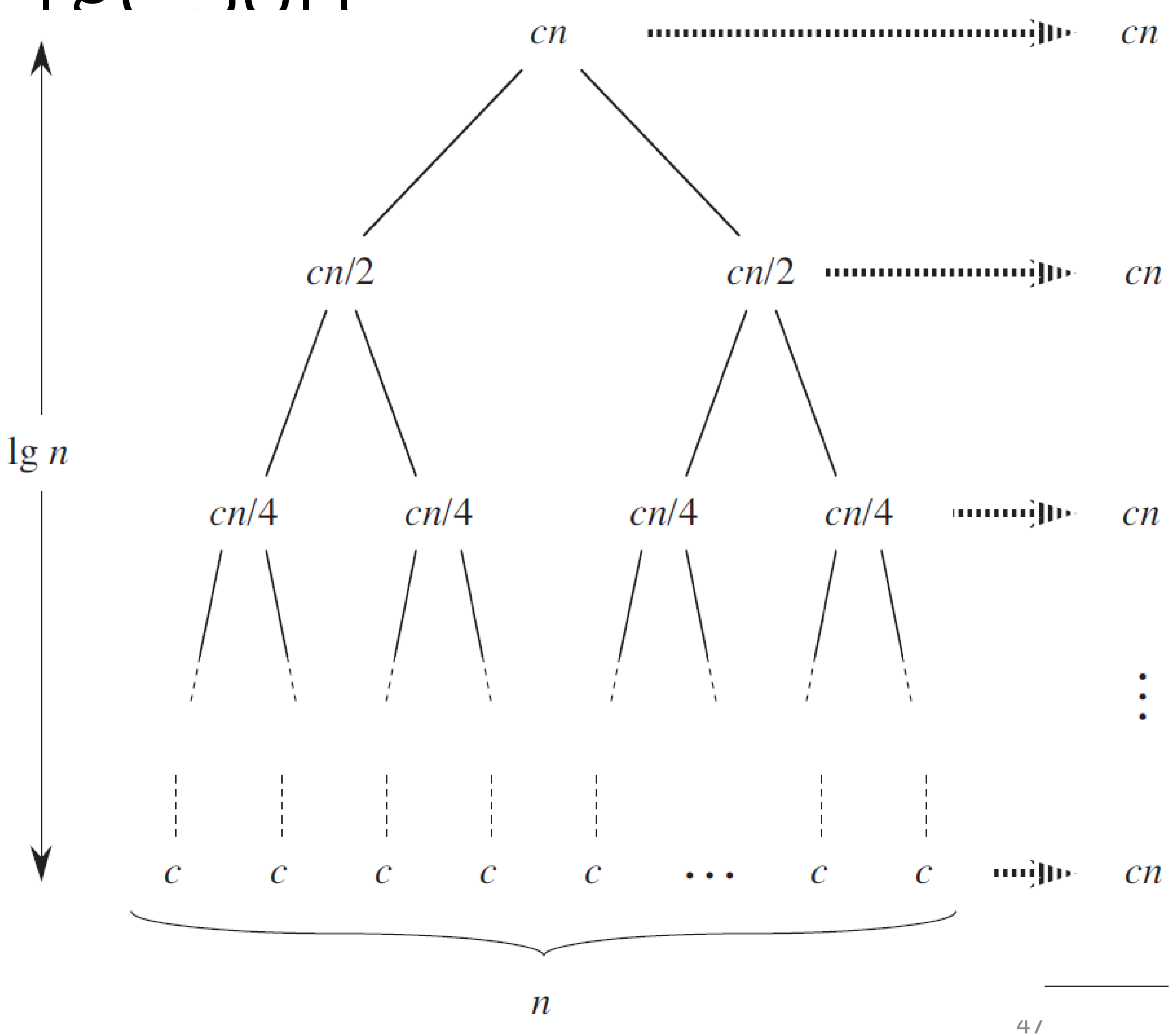
Recursion Tree for Merge Sort

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

Recursive Cost?

Base case cost?

Solution?



Examples

- $T(n) = 3T(n/4) + cn$
- $T(n) = 3T(n/4) + cn^2$.
- $T(n) = T(n/3) + T(2n/3) + cn$.

Substitution Method

- Guess a solution T(n) = f(n) and prove it inductively.
- Assume that such solution holds for the subproblems established by the recurrence.
- Use such assumptions in the definition of the recurrence to prove it holds for n as well.

The Substitution Method for Merge Sort Recurrence

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

- Guess: T(n) = O (n lg n).
- Prove that T(n) ≤ cn lg n.
- Assume that

$$T(\lfloor n/2 \rfloor) \le c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$$

Recursive proof:

$$\begin{aligned} T(n) &\le 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\ &\le cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\le cn \lg n , \end{aligned}$$

The Substitution Method for Merge Sort Recurrence

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

- Guess: T(n) = O (n lg n).
- Prove that T(n) ≤ cn lg n.
- Assume that

$$T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$$

Proving boundary conditions:

- For the sake of argument, assume that T(1) = 1.
- T(1) = 1 ≤ c(1) lg (1) fails.
- T(2) = 2T(1) + 2 = 4 ≤ c(2) lg (2) = 2c.
- T(3) = 2T(1) + 3 = 5 ≤ c(3) lg (3) = 3c lg 3
- Note that for c ≥ 2, n_0=2, T(n) ≤ cn lg n for all n ≥n_0.
- The base case of the proof doesn't need to be the base case of the recurrence.

Guessing correct solutions

- Use the solution given by the recursion tree. The substitution proof is necessary it the tree analysis was not accurate.
- Beyond that, there is not general strategy.
- Experience and creativity are required.
- Solutions to similar recurrences can often be used.
  - Example: $T(n) = 2T(n/2+17) + cn$.

Guessing correct solutions

- Another strategy: find evident upper and lower bounds and then refine them.
- For instance, for $T(n) = 2T(\lfloor n/2 \rfloor) + n$
  - It is clear that $T(n) = \Omega(n)$, so we can test $T(n) = O(n^2)$.
  - We can refine the solution between n and n^2 until we reach $O(n \lg n)$.

Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

Subtracting a lower-order term.

- Sometimes the inductive proof does not work even though the guessed solution is right.
- Then, it is necessary to subtract a lower-order term.
- For instance, $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

$$T(n) \le cn$$

$$\begin{aligned} T(n) &\le c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 , \end{aligned}$$

$$T(n) \le cn - d$$

$$\begin{aligned} T(n) &\le (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1 \\ &= cn - 2d + 1 \\ &\le cn - d , \end{aligned}$$

Presentation made by Juan Mendivelso. Contents and figures extracted from the book: Introduction to Algorithms, Third Edition. Cormen, Leiserson, Rivests and Stein. The MIT Press. 2009.

Avoid pitfalls

- The proof needs to be accurate.
- For example, $T(n) = 2T(\lfloor n/2 \rfloor) + n$

$$\begin{aligned} T(n) &\le 2(c\lfloor n/2 \rfloor) + n \\ &\le cn + n \\ &= O(n) , \qquad \Longleftarrow \textit{wrong!!} \end{aligned}$$

Changing variables

$$T(n) = 2T\left(\left\lfloor \sqrt{n} \right\rfloor\right) + \lg n$$

$$T(2^m) = 2T(2^{m/2}) + m \qquad\qquad m = \lg n$$

$$S(m) = 2S(m/2) + m$$

$$S(m) = O(m \lg m)$$

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$$

Example

- Prove the guesses provided by the recursion tree to
  - $T(n) = 3T(n/4) + cn$
  - $T(n) = 3T(n/4) + cn^2$.

Master Method

Provides a cookbook method for solving recurrences of the form

$$T(n) = aT(n/b) + f(n) ,$$

where $a \geq 1$ and $b > 1$ are positive constants and f(n) is an asymptotically positive function.

For short, f(n) and $n^{\log b\,(a)}$ are compared. The one that is polynomially bigger is the solution to the recurrence. If they are equal, the solution is f(n)(lg n).

Master Method

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Example of Case 1

$$T(n) = 9T(n/3) + n$$

$$a = 9,\ b = 3.\ f(n) = n$$

$$n^{\lg_b a} = n^{\lg_3 9} = n^2$$

$$f(n) = \mathcal{O}(n) = \mathcal{O}(n^{2-1}) = \mathcal{O}(n^{\lg_b a - \epsilon})$$

$$\epsilon = 1$$

$$T(n) = \theta(n^2)$$

Example of Case 2

$$T(n) = T(2n/3) + 1$$

$$a = 1,\ b = \tfrac{3}{2},\ f(n) = 1$$

$$n^{\lg_b a} = n^{\lg_{\frac{3}{2}} 1} = n^0 = 1$$

$$f(n) = 1 = \theta(1) = \theta(n^{\lg_b a})$$

$$T(n) = \theta(\lg n)$$

Example of Case 3

$$T(n) = 3T(n/4) + n \lg n$$

$$a = 3,\ b = 4,\ f(n) = n \lg n$$

$$n^{\lg_b a} = n^{\lg_4 3} = n^{0,793}, \quad f(n) = \Omega(n) = \Omega(n^{\lg_4 3+\epsilon}),\ \epsilon \simeq 0,207$$

- Regularity test: $3f(n/4) \leq cf(n)$

$$
\begin{aligned}
3f(n/4) &= 3\tfrac{n}{4}\lg(n/4) \\
&= \tfrac{3}{4}n(\lg(n) - \lg(4)) \\
&= \tfrac{3}{4}n(\lg n - \tfrac{3}{2}n \leq \tfrac{3}{4}n\lg n = \tfrac{3}{4}f(n)) \\
\Rightarrow \qquad & T(n) = \theta(n \lg n)
\end{aligned}
$$

Cases not covered by the Master Method

$$T(n) = 2T(n/2) + n \lg n$$

$$a = 2,\ b = 2,\ f(n) = n \lg n$$

$$n^{\lg_2 2} \equiv n \quad f(n) = \Omega(n)\Omega(n^{\lg_b a})$$

$$\exists \epsilon > 0 : f(n) = \Omega(n^{1+\epsilon})?$$

BIBLIOGRAPHY

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press. 2009.
- Images of the Master Method by Julio Cesar Lopez.