طراحی الگوریتم‌ها

Introduction to Algorithm

مهدی جوانمردی

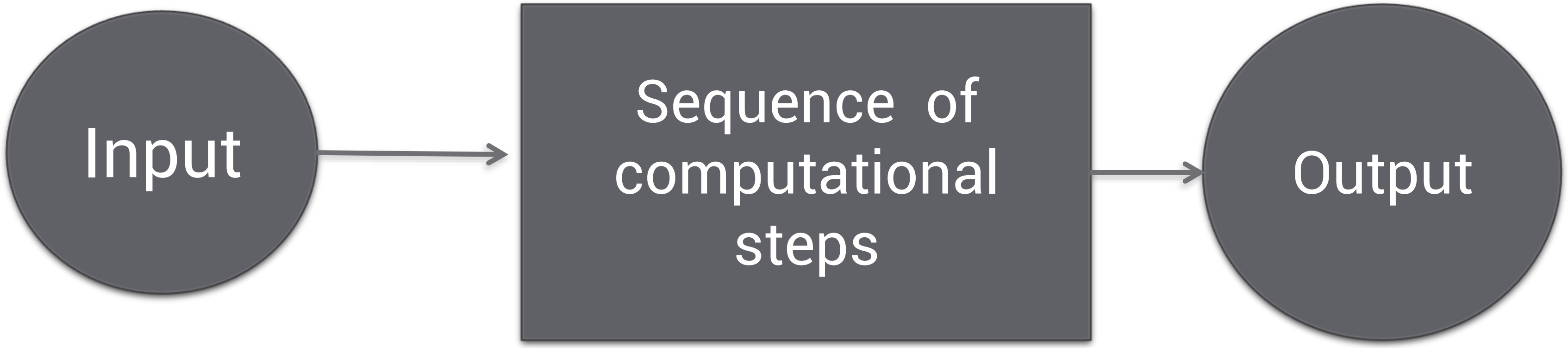
پاییز ۱۴۰۱

فصل اول: مفاهیم پایه

- الگوریتم چیست؟
- الگوریتم صحیح و غیر صحیح
- برخی کاربردهای الگوریتم
- الگوریتم بعنوان یک فناوری

الگوریتم چیست؟

An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.



مثال: مسئله مرتب‌سازی رشته اعداد

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 $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Ex. Input: sequence 31, 41, 59, 26, 41, 58

Output: sequence 26, 31, 41, 41, 58, 59

الگوریتم صحیح و الگوریتم غیر صحیح

- An algorithm is said to be correct if, for every input instance, it halts with the correct output. We say that a correct algorithm solves the given computational problem.
- An incorrect algorithm might not halt at all on some input instances, or it might halt with an answer other than the desired one.
- *Incorrect algorithms can sometimes be useful, if their error rate can be controlled. (An example of this when we study algorithms for finding large prime numbers.)*

الگوریتم بعنوان یک فناوری

- فرض: کامپیوترها بینهایت سریع و حافظه رایگان باشد
- آیا باز نیاز است درس طراحی الگوریتم بخوانیم؟
- جواب: بله!!!

- باید مطمئن شویم که الگوریتم طراحی شده در همه حالت جواب درست را تولید می کند

- و اما واقعیت: کامپیوترها سریع هستند ولی نه بینهایت!
  حافظه ارزان است ولی رایگان نیست!

منابع محدود و مشخص

فصل دوم: شروع به کار!

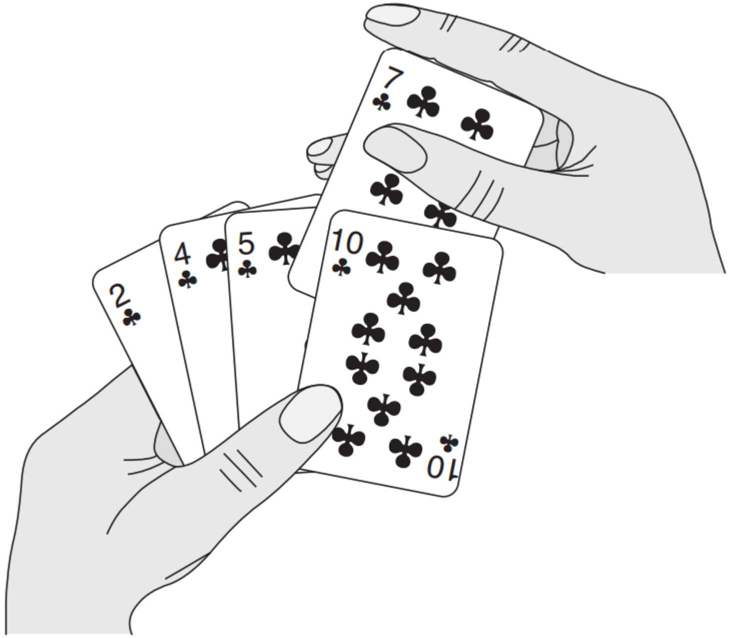
- مرتب‌سازی درجی
- تحلیل الگوریتم
- طراحی الگوریتم

Insertion Sort | مرتب‌سازی درجی

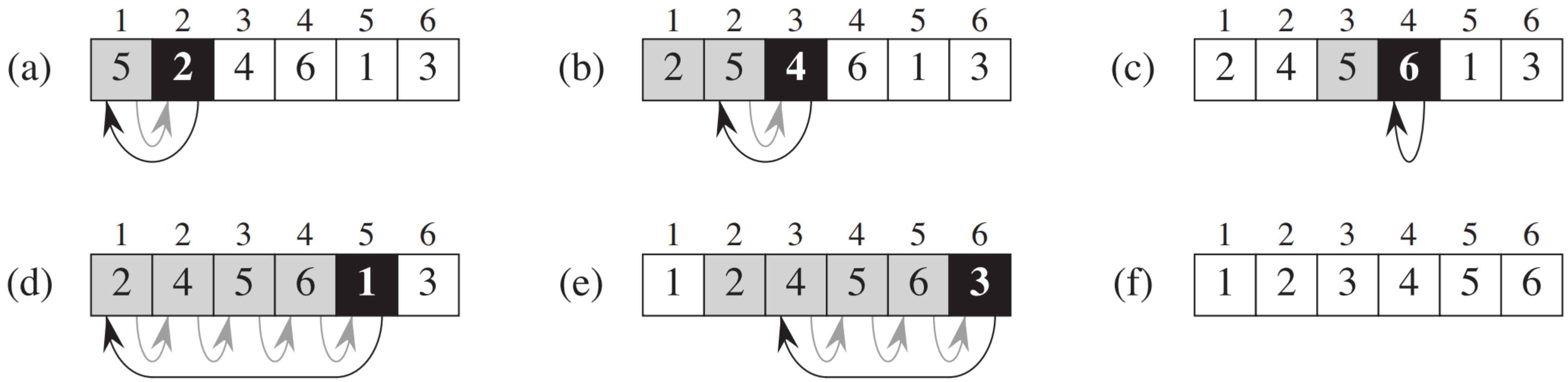
Efficient algorithm for sorting a small number of elements:

- We start with an empty left hand and the cards face down on the table.
- We then remove one card at a time from the table and insert it into the correct position in the left hand. To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left.

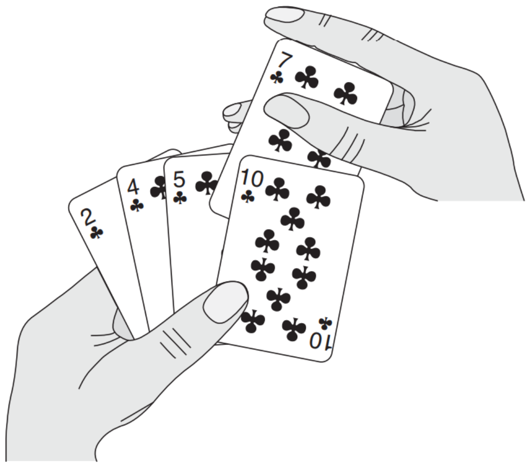
```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```



مثال مرتب‌سازی درجی

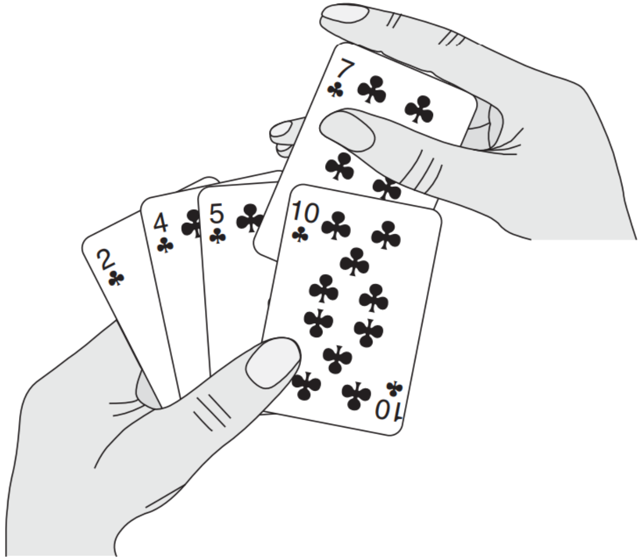



```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```



اثبات صحت الگوریتم مرتب‌سازی درجی


```
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
```



مشابه استقراء ریاضی

Loop Invariant

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

اثبات صحت الگوریتم با استفاده از مستقل از حلقه

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

loop invariants: understand why an algorithm is correct

We must show three things about a loop invariant:

- **Initialization:** It is true prior to the first iteration of the loop
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration
- **Termination:** When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

فوائد استفاده از شبه کد

```
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
```

تحلیل الگوریتم‌ها

- Analyzing an algorithm: for an input size,
  - measure memory (space)
  - measure computational time (running time).
- Input size: depends on the problem:
  - Sorting: number of items in the input; array size,… O(n)
  - Big integer (multiplying, …): number of bits to represent the input in binary notation O(log n)
  - Two numbers: input of a graph can be O(n,m), number of vertices and number of edges.
- Running time:
  - A constant amount of time is required to execute each line
  - each execution of the i^{th} line takes time c_i , where c_i is a constant.

تحلیل زمانی مرتب‌سازی درجی

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

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

تحلیل زمانی مرتب‌سازی درجی

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

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

مثال مرتب‌سازی

Ex. Input: sequence 31, 41, 59, 26, 41, 58

Output: sequence 26, 31, 41, 41, 58, 59

تحلیل بهترین حالت

Best case	The array is already sorted

1	2	3	4	5	6
1	2	3	4	5	6

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

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

تحلیل بدترین حالت

Worst case	The array is in reverse sorted order

1	2	3	4	5	6
6	5	4	3	2	1

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

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

and

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

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

بدترین زمان اجرا و زمان اجرای متوسط

- Worst-case running time:
  - the longest running time for any input of size n:
    - Upper bound on the running time for any input
    - For some algorithms, the worst case occurs fairly often
    - The "average case" is often roughly as bad as the worst case
- Average-case or expected running time:
  - Technique of probabilistic analysis
  - Assume that all inputs of a given size are equally likely
  - Difficult to analyze

☆ متوسط زمان اجرای مرتب سازی درجی؟

تحلیل زمانی مرتب‌سازی درجی

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

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

مثال مرتب‌سازی

Ex. Input: sequence 31, 41, 59, 26, 41, 58

Output: sequence 26, 31, 41, 41, 58, 59

تحلیل بهترین حالت

Best case	The array is already sorted

1	2	3	4	5	6
1	2	3	4	5	6

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

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

تحلیل بدترین حالت

Worst case	The array is in reverse sorted order

1	2	3	4	5	6
6	5	4	3	2	1

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

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

and

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

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

بدترین زمان اجرا و زمان اجرای متوسط

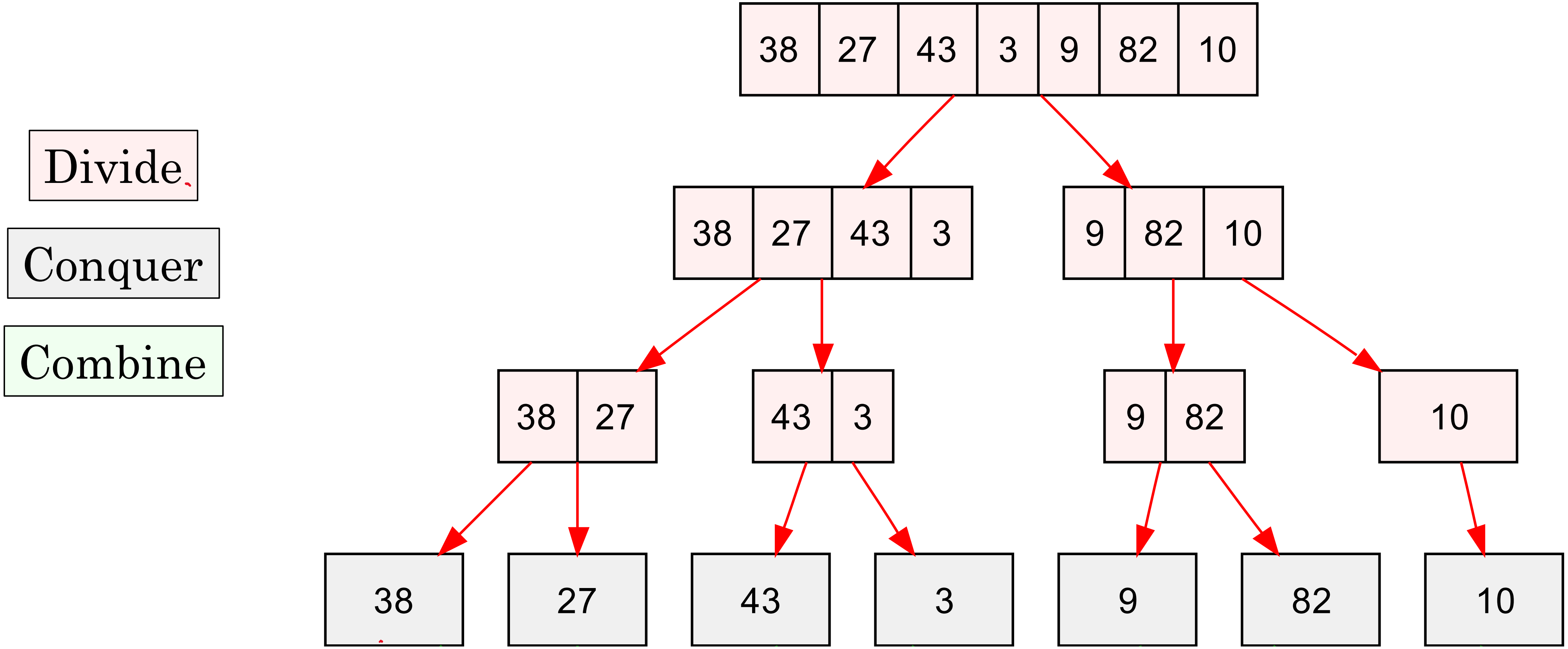
- Worst-case running time:
  - the longest running time for any input of size n:
    - Upper bound on the running time for any input
    - For some algorithms, the worst case occurs fairly often
    - The "average case" is often roughly as bad as the worst case
- Average-case or expected running time:
  - Technique of probabilistic analysis
  - Assume that all inputs of a given size are equally likely
  - Difficult to analyze

☆ متوسط زمان اجرای مرتب سازی درجی؟

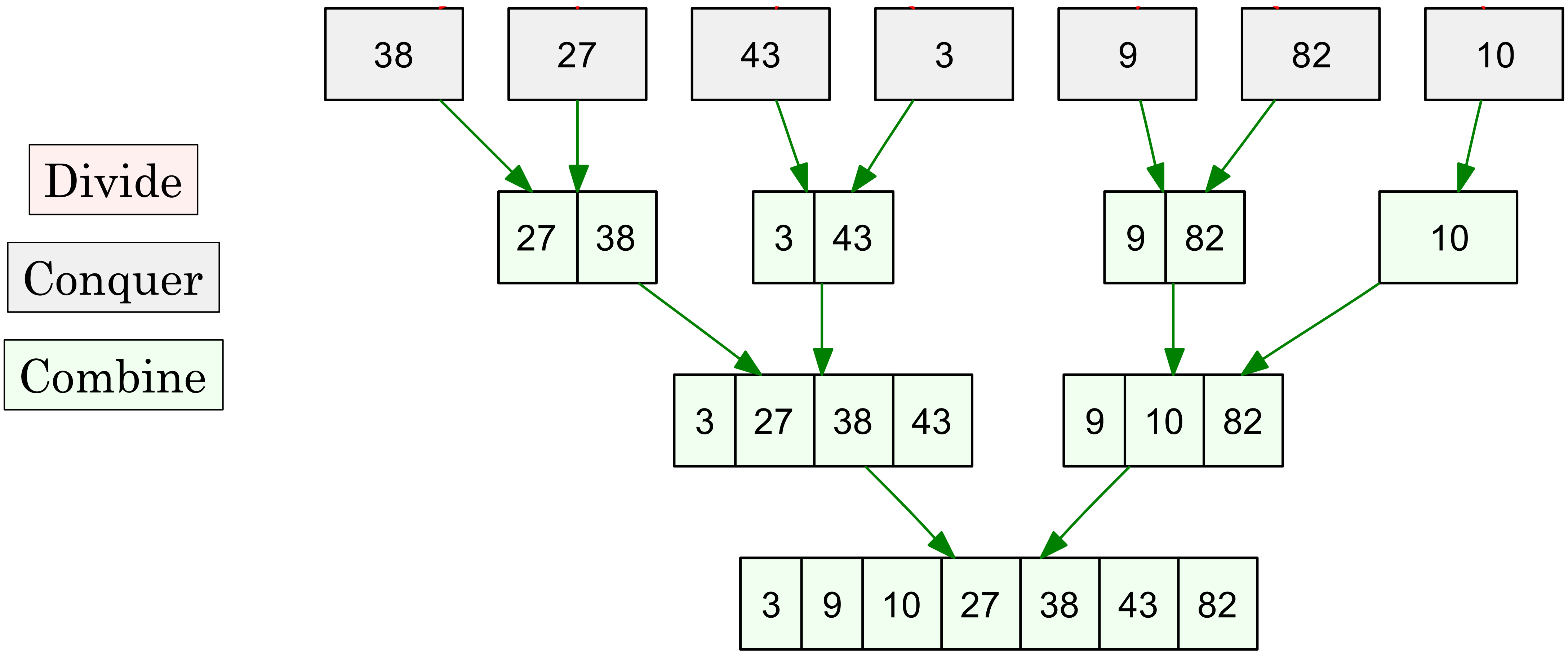
تکنیک‌های طراحی الگوریتم و الگوریتم‌های تقسیم و حل

- Insertion sort: **Incremental approach** A[1 .. j-1] A[j] → A[1 .. j]
- **The divide-and-conquer approach**:
  - **Divide** the problem into a number of subproblems (similar to original problem).
  - **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
  - **Combine** the solutions to the subproblems into the solution for the original problem.
- **Recursive structure:** to solve a given problem, they call themselves recursively one or more times to deal with closely related subproblems.

مثال تقسیم و حل: Merge Sort

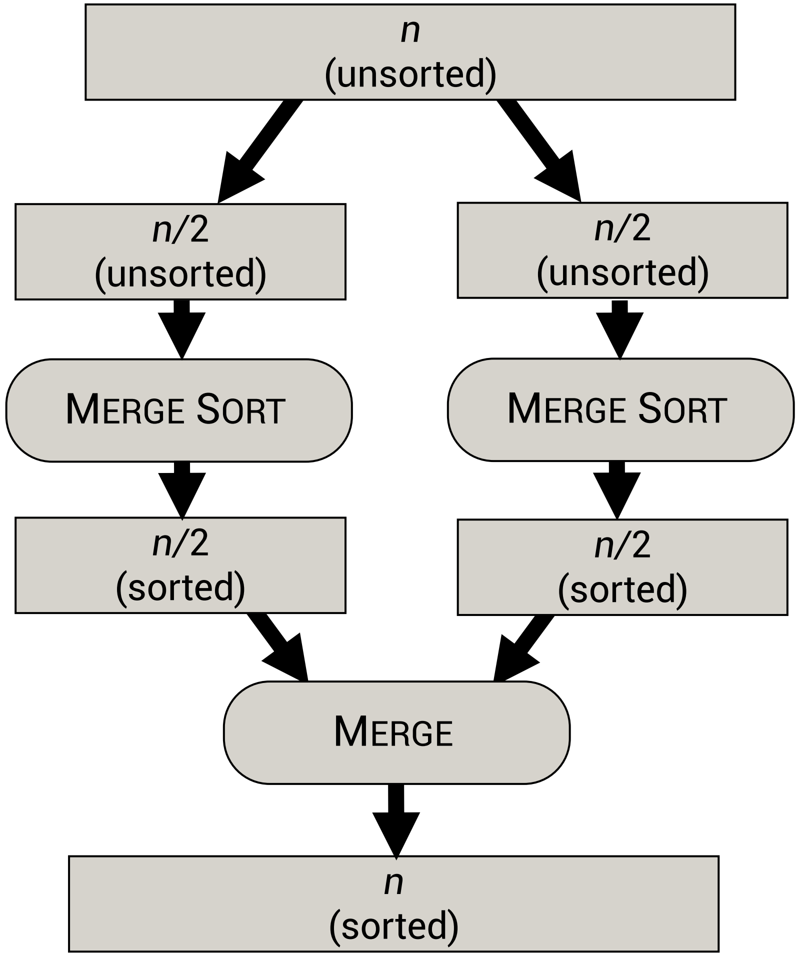


مثال تقسیم و حل: Merge Sort



Merge Sort | الگوریتم مرتب‌سازی ادغامی

- **Divide:** Divide the n-elements sequence to be sorted into two subsequences of n/2 elements each
- **Conquer:** Sort the two subsequences recursi -vely using merge sort
- **Combine:** Merge the two sorted subsequences to produce the sorted answer.



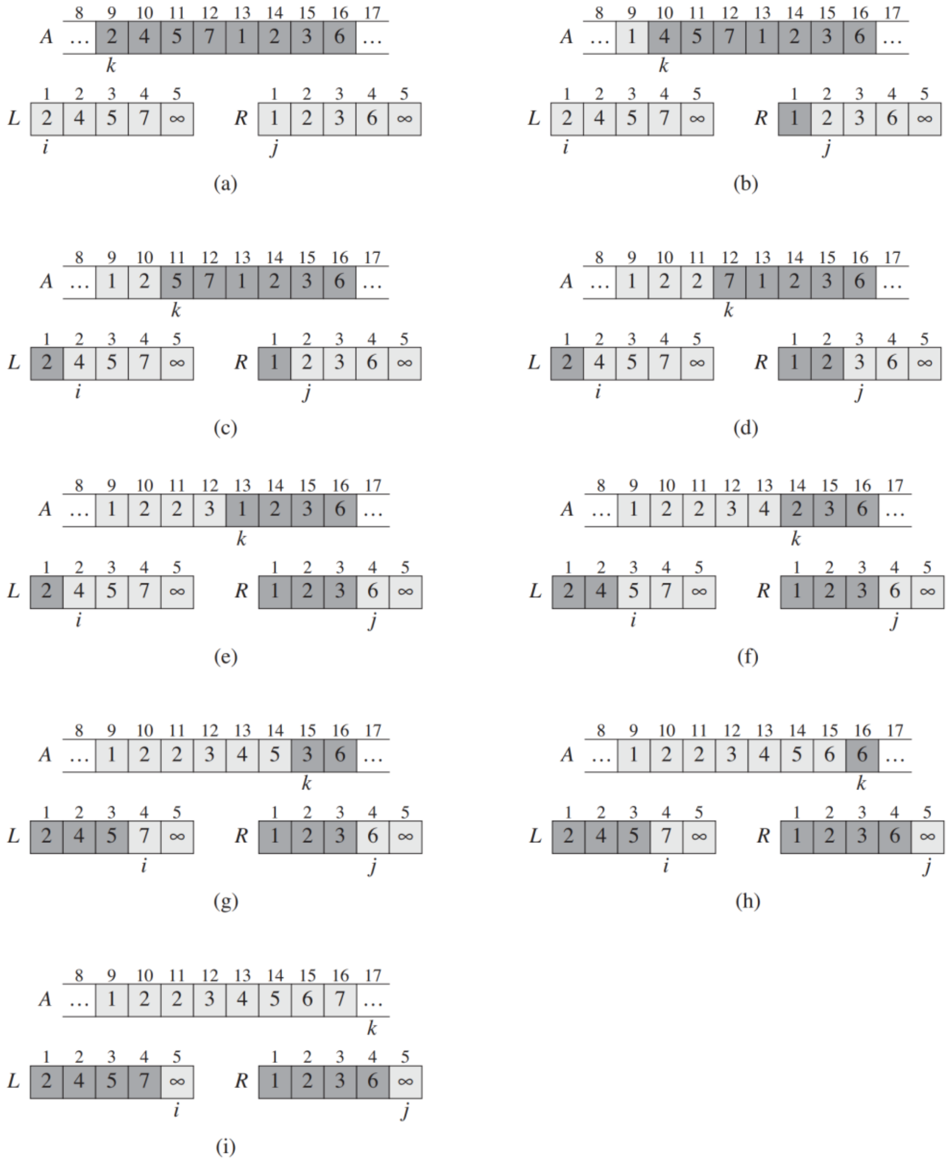

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Divide
Conquer
Conquer
Combine

الگوریتم مرتب‌سازی ادغامی | Merge Sort

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
```



Merge Sort | الگوریتم مرتب‌سازی ادغامی

```
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

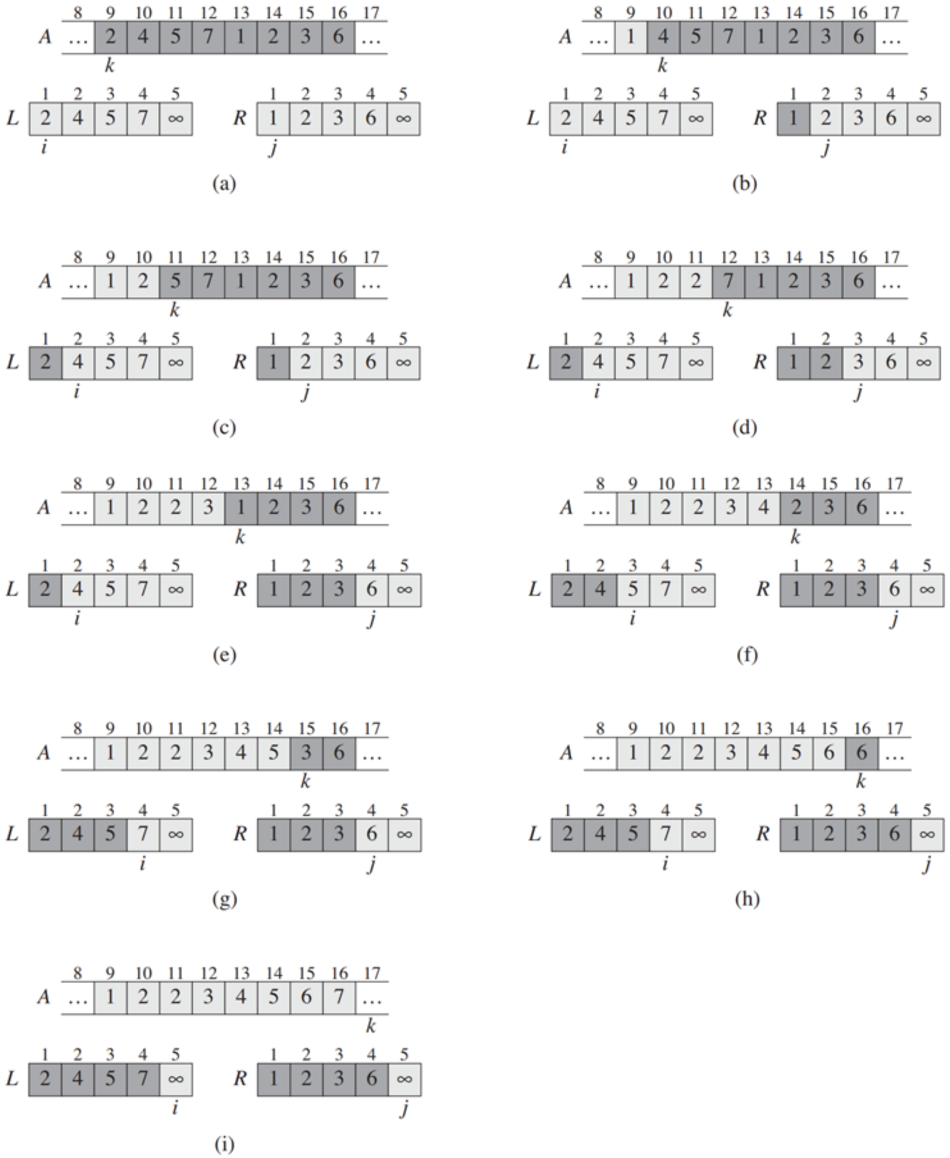
$A[p .. k-1]$

$L[1 .. n_1+1]$

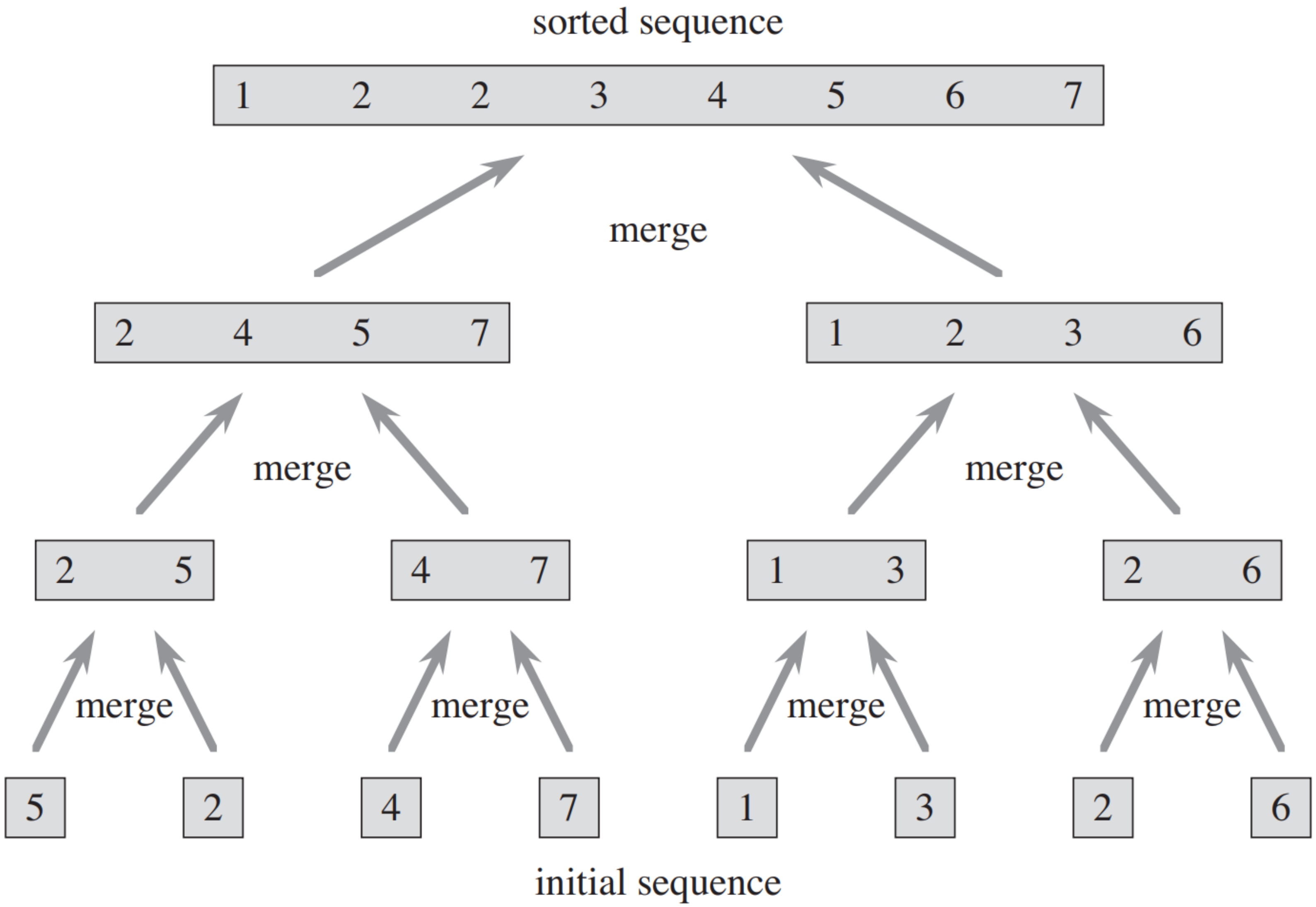
$R[1 .. n_2+1]$

$L[i]$

$R[j]$



مثال مرتب‌سازی ادغامی



تحلیل زمانی الگوریتم‌های تقسیم و حل

- **Divide:** D(n) = Θ(1).
- **Conquer:** solve two subproblems, each of size n/2, which contributes 2T (n/2) to the running time.
- **Combine:** the MERGE procedure on an n-element subarray takes time Θ(n), so C(n) = Θ(n).

Merge(A,p,q,r)

$\Theta(1)$:
$n_1 = q - p + 1$
$n_2 = r - q$

$\Theta(n)$:
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
    $R[j] = A[q + j]$

$\Theta(1)$:
$L[n_1 + 1] = \infty$
$R[n_2 + 1] = \infty$
$i = 1$
$j = 1$

$\Theta(n)$:
for $k = p$ **to** r
    if $L[i] \le R[j]$
        $A[k] = L[i]$
        $i = i + 1$
    else $A[k] = R[j]$
        $j = j + 1$

MERGE-SORT(A,p,r)

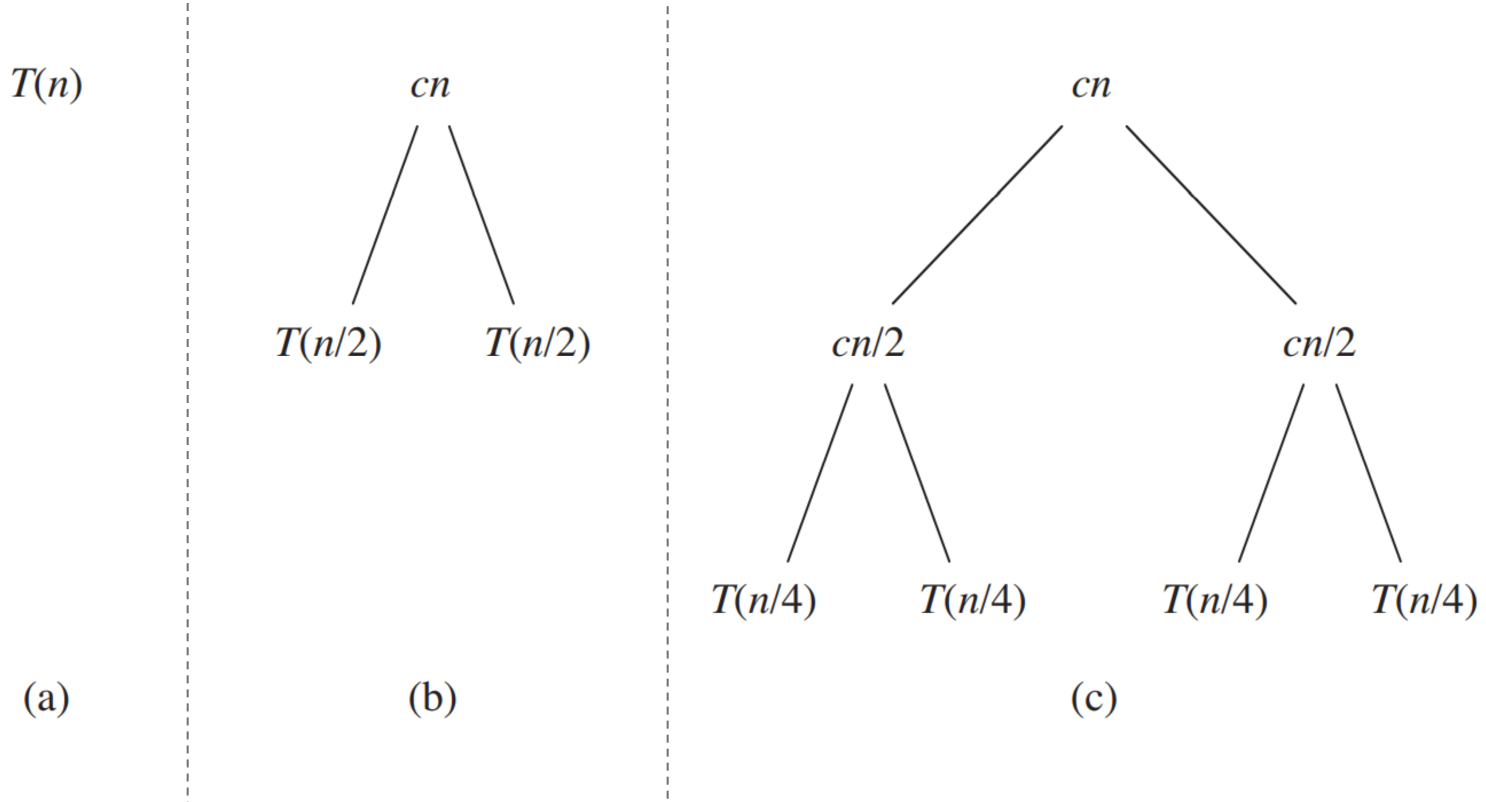
if $p < r$
    $\Theta(1)$: $q = \lfloor (p + r)/2 \rfloor$
    $T(n/2)$: MERGE-SORT(A, p, q)
    $T(n/2)$: MERGE-SORT$(A, q + 1, r)$
    $\Theta(n)$: MERGE(A, p, q, r)

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

تحلیل زمانی با درخت بازگشتی

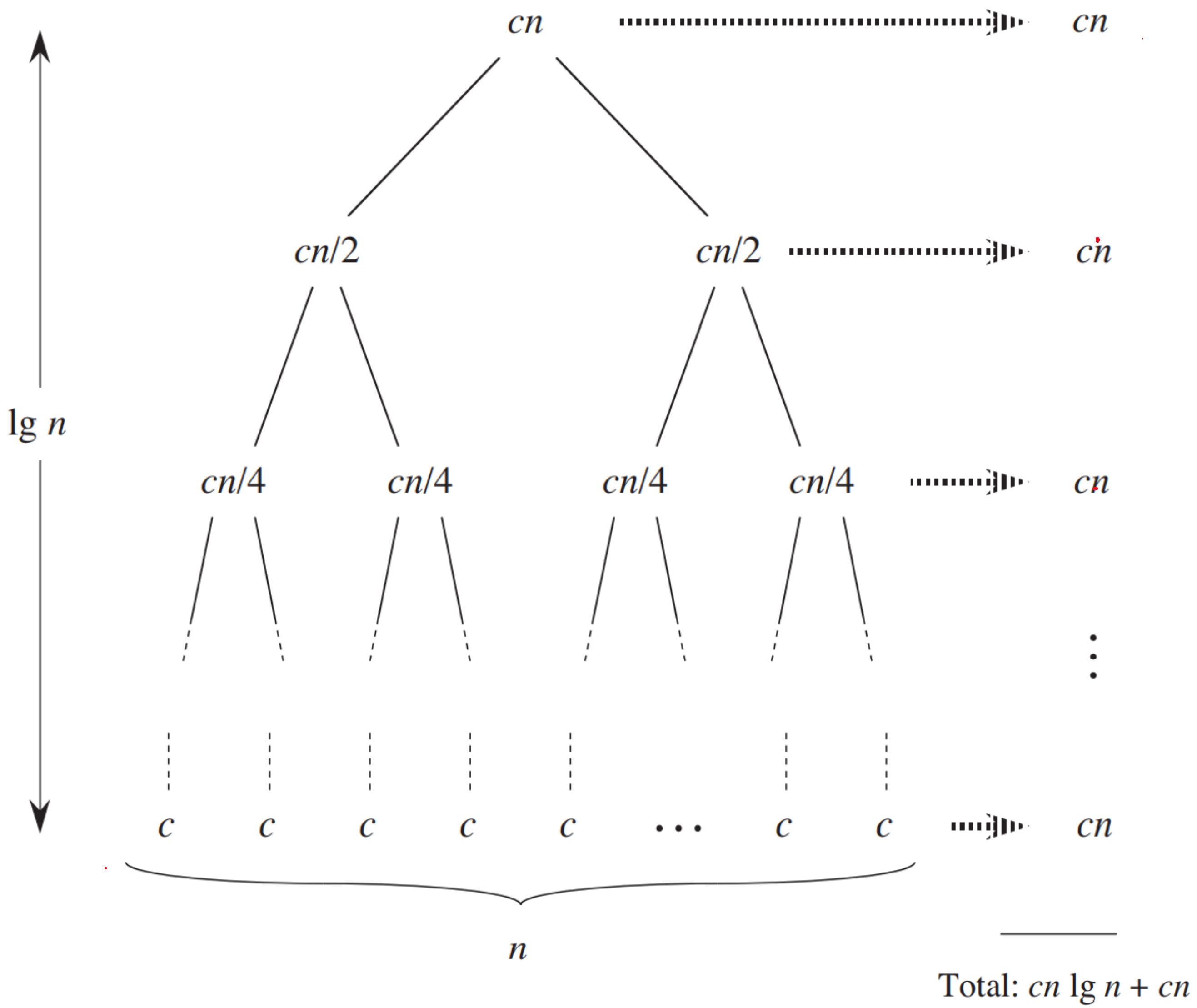
$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2\,T(n/2) + \theta(n) & \text{if } n > 1 \end{cases}$$

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



تحلیل زمانی با درخت بازگشتی

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



تمرین: تحلیل زمانی روش تقسیم و حل

Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

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

is $T(n) = n \lg n$.

راه حل تمرین

$$T(n) = \begin{cases} 2 & \text{if } n = 2\,, \\ 2T(n/2) + n & \text{if } n = 2^k\text{, for } k > 1 \end{cases} \longrightarrow T(n) = n \lg n$$

The base case is when $n = 2$, and we have $n \lg n = 2 \lg 2 = 2 \cdot 1 = 2$.

For the inductive step, our inductive hypothesis is that $T(n/2) = (n/2) \lg(n/2)$. Then

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2)\lg(n/2) + n \\ &= n(\lg n - 1) + n \\ &= n \lg n - n + n \\ &= n \lg n\,, \end{aligned}$$

which completes the inductive proof for exact powers of 2.