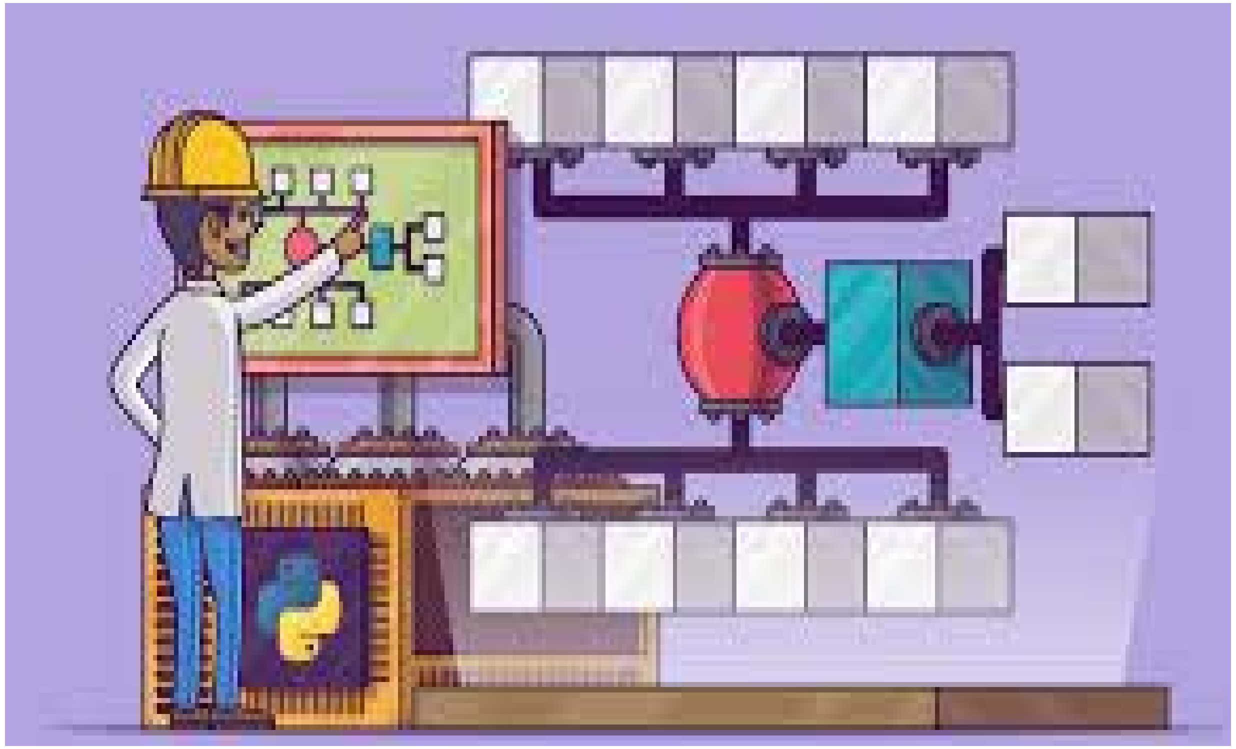
ساختمان داده ها

مدرس:
سمانه حسینی سمنانی

دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر



تحلیل الگوریتم‌ها

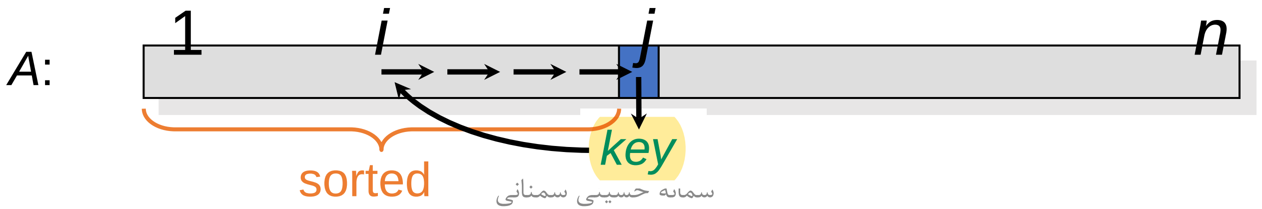
- انتخاب بین الگوریتم‌های مختلف بر اساس معیار:
  1. زمان اجرا ← پیچیدگی زمانی
  2. حافظه لازم در زمان اجرای الگوریتم ← پیچیدگی مکانی

- زمان اجرا به چه عواملی وابسته است؟ نوع پردازنده، کامپایلر، اندازه ورودی، پیچیدگی الگوریتم
- آیا محاسبه زمان اجرا بر حسب s یا ms معیار مناسبی است؟

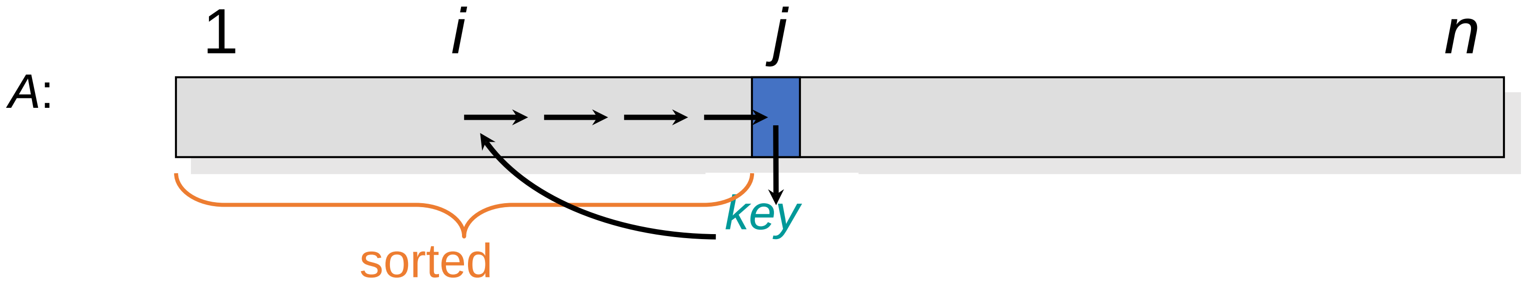
- نیازمند معیاری مستقل از جزییات سخت افزار و نرم افزار کامپیوتر اجرا کننده الگوریتم هستیم.

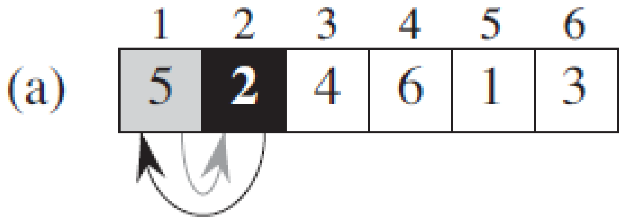
الگوریتم Insertion sort

```
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





Insertion sort الگوریتم

```
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
```

- The algorithm sorts the input numbers ***in place***

تحلیل پیچیدگی زمانی الگوریتم Insertion sort

```
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
```

تحلیل پیچیدگی زمانی الگوریتم Insertion sort

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

- Three possible cases:
  - Best case
  - Worst case
  - Average case

Best case

- The best case occurs if the array is already sorted
- $t_j = 1 \; for \; j = 2,3, \dots, n$ and the best-case running time is

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

- $an + b$ for constants a and b that depend on the statement costs c_i;
- ***linear function*** of n.

Worst case

- The worst case occurs if the array is in reverse sorted order
- $t_j = j \; for \; j = 2,3,\ldots,n$ as we must compare each element with each element in the entire sorted subarray:

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

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

Worst case

- $an^2 + bn + c$ for *constants* a, b *and* c that depend on the statement costs c_i;
- ***quadratic function*** of n.

Average case

- $t_j = j/2 \; for \; j = 2,3,\ldots,n$

$$\sum_{j=2}^{n} j/2 = \frac{n(n+1)}{4} - 1$$

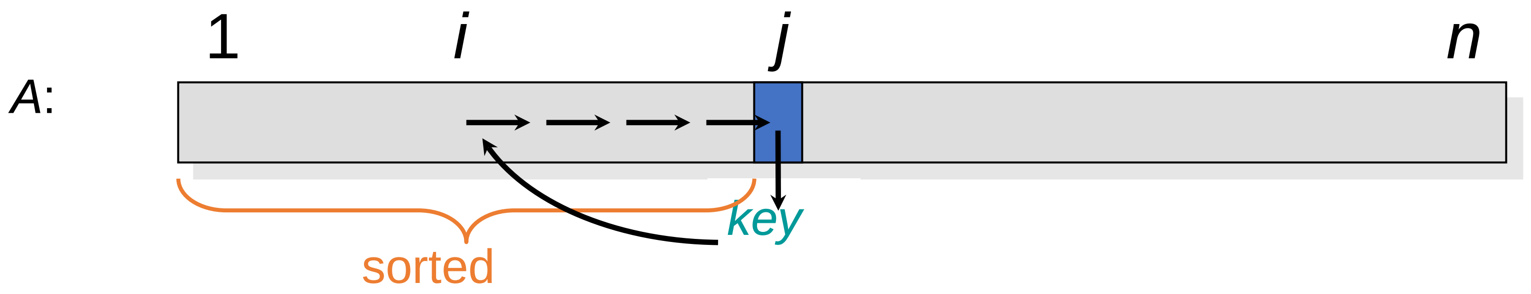
$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{4} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{4}\right) + c_7\left(\frac{n(n-1)}{4}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{4} + \frac{c_6}{4} + \frac{c_7}{4}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{4} - \frac{c_6}{4} - \frac{c_7}{4} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

Worst case vs Average case analysis

- $an^2 + bn + c$ for *constants a, b and c* that depend on the statement costs c_i;
- ***quadratic function*** of n.

- Worse case is more important for us because:
  - It provides a guarantee that the algorithm will never take any longer.
  - For some algorithms, the worst case occurs fairly often.
  - The “average case” is often roughly as bad as the worst case.

آیا می‌توان هزینه Insertion Sort را کاهش داد؟



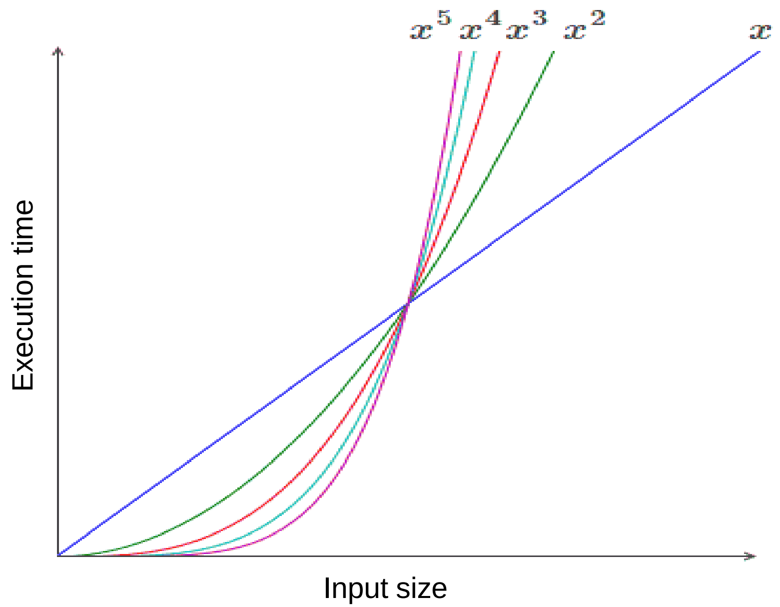
Order of growth درجه رشد توابع

- Insertion sort: $\longrightarrow$ $\boldsymbol{\theta(n^2)}$
  - Best case: $an + b$ $\xrightarrow{\text{Ignore a \& b}}$ order of growth: n
  - Worst case: $an^2 + bn + c$ $\xrightarrow{\text{Ignore a, bn+c}}$ order of growth: n^2
  - Average case: $an^2 + bn + c$ $\xrightarrow{\text{Ignore a, bn+c}}$ order of growth: n^2

- Why we can ignore such term?
  - lower-order terms are relatively insignificant for large values of n.
  - constant factors are less significant than the rate of growth in determining computational efficiency for large inputs

Order of growth درجه رشد توابع

- one algorithm is more efficient than another if its worstcase running time has a lower order of growth



- for large enough inputs e.g. $\theta(n^2)$ algorithm will run more quickly in the worst case than , $\theta(n^3)$ algorithm

merge sort مرتب سازی ادغامی

- Input array:
  - 5, 2, 3, 7, 4, 9, 12, 1, 6, 10
- Two sorted part:
  - 2, 3, 4, 5, 7
  - 1, 6, 9, 10, 12

Calculate these two sorted arrays using the same method

Recursive

- Merge two parts:
  - 1, 2, 3, 4, 5, 6, 7, 9, 10, 12

مرتب سازی ادغامی merge sort

MERGE-SORT(A, p, r)

if $p < r$

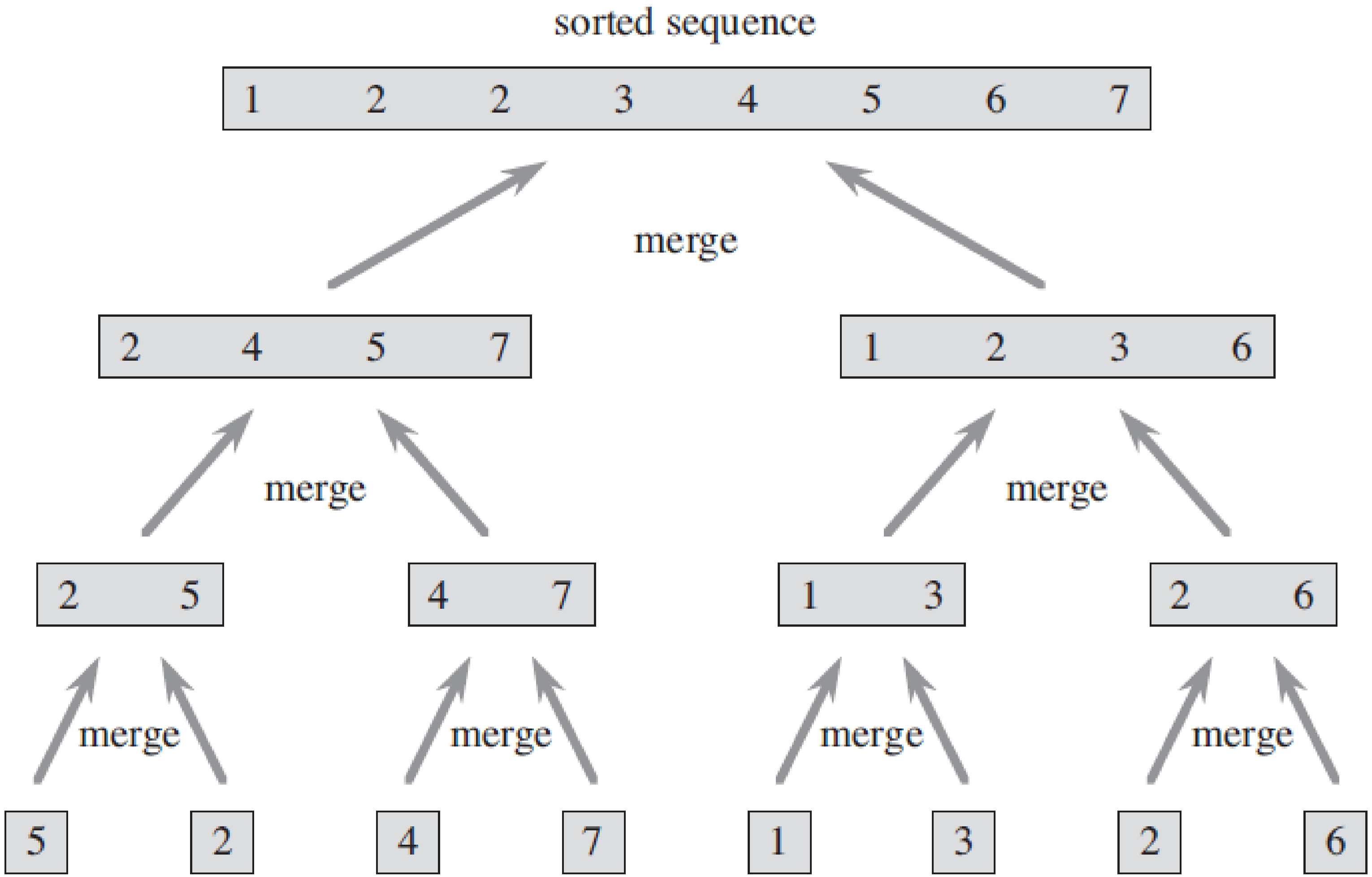
$\quad q = \lfloor (p + r)/2 \rfloor$

$\quad$ MERGE-SORT(A, p, q)

$\quad$ MERGE-SORT$(A, q + 1, r)$

$\quad$ MERGE(A, p, q, r)

merge sort مرتب سازی ادغامی



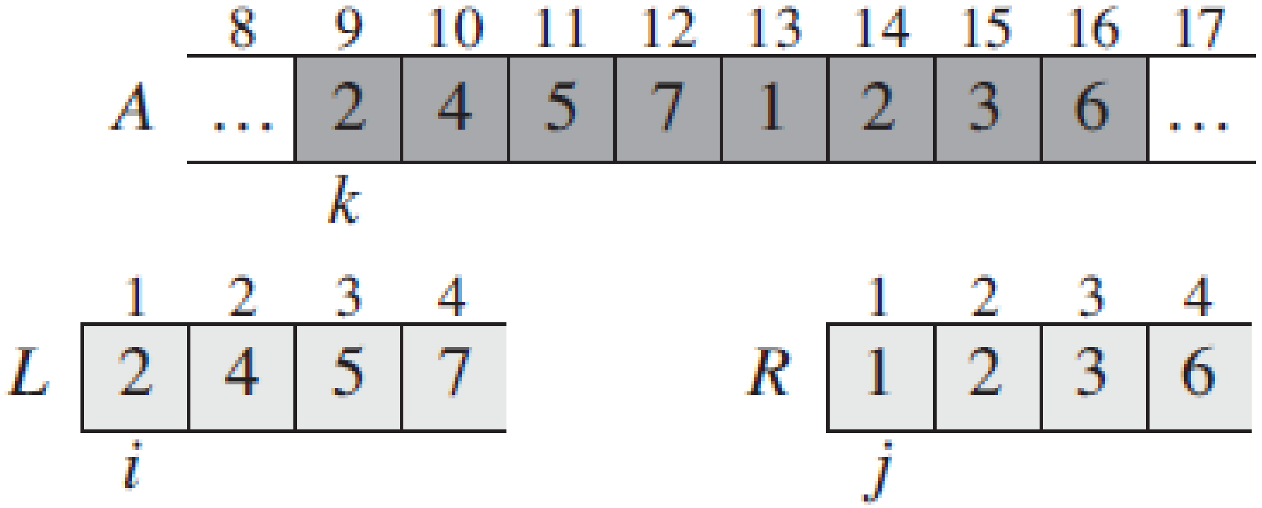
merge sort مرتب سازی ادغامی

- Two sorted part:
  - L: 2, 3, 4, 5, 7
  - R: 1, 6, 9, 10, 12
- Merge two parts:
  - A: 1, 2, 3, 4, 5, 6, 7, 9, 10, 12

MERGE(A, p, q, r)

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

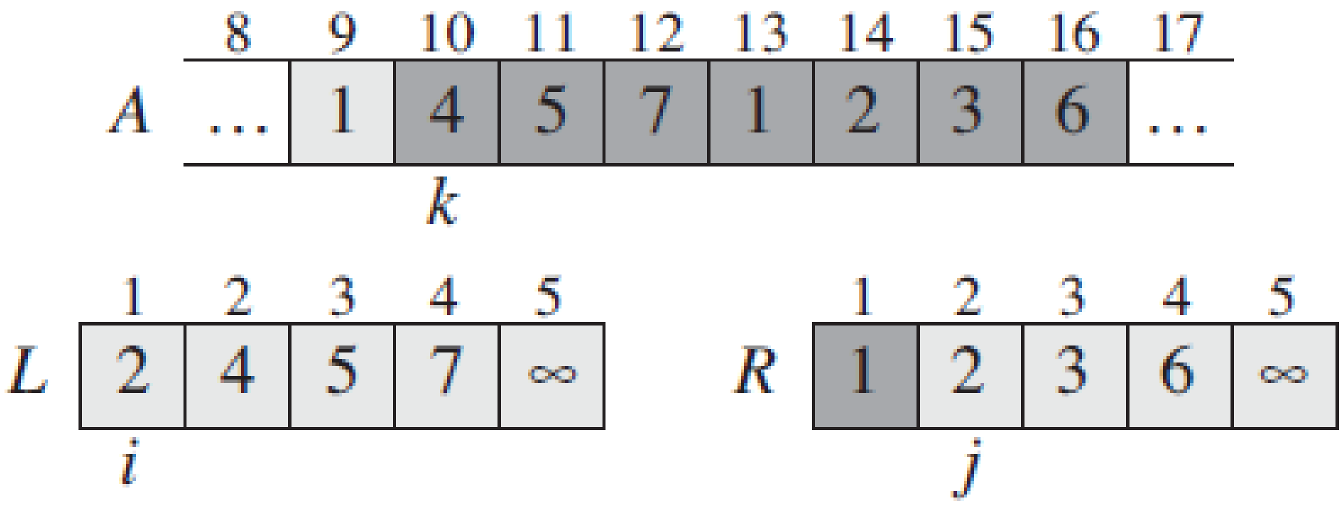
merge sort مرتب سازی ادغامی



(a)

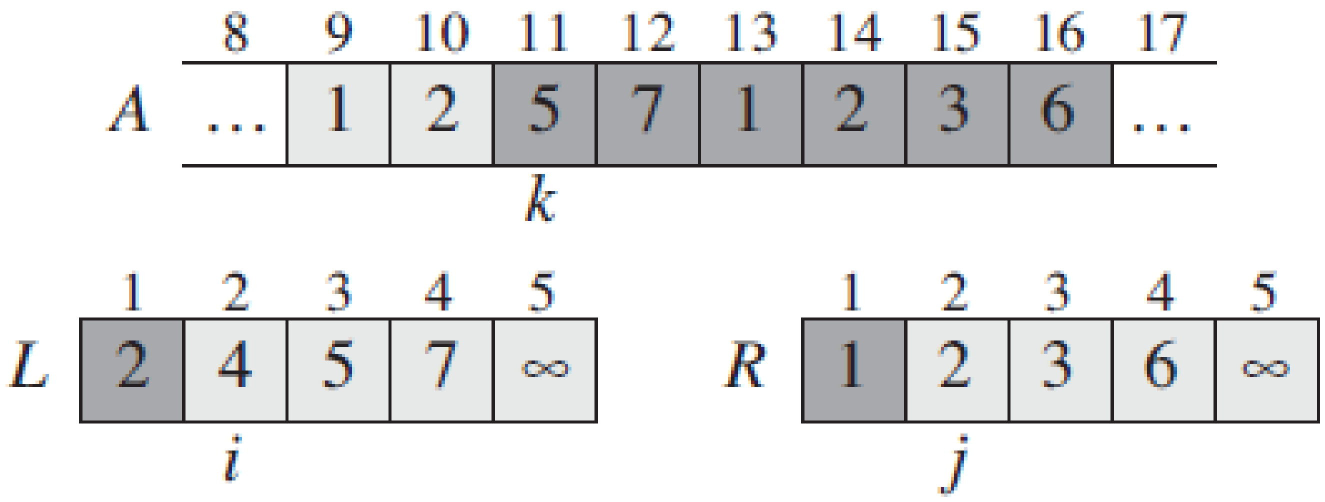
- To avoid having to check whether either array is empty in each basic step, place at the end of each array a ***sentinel*** number, which contains a special value that we use to simplify our code.

مرتب سازی ادغامی merge sort



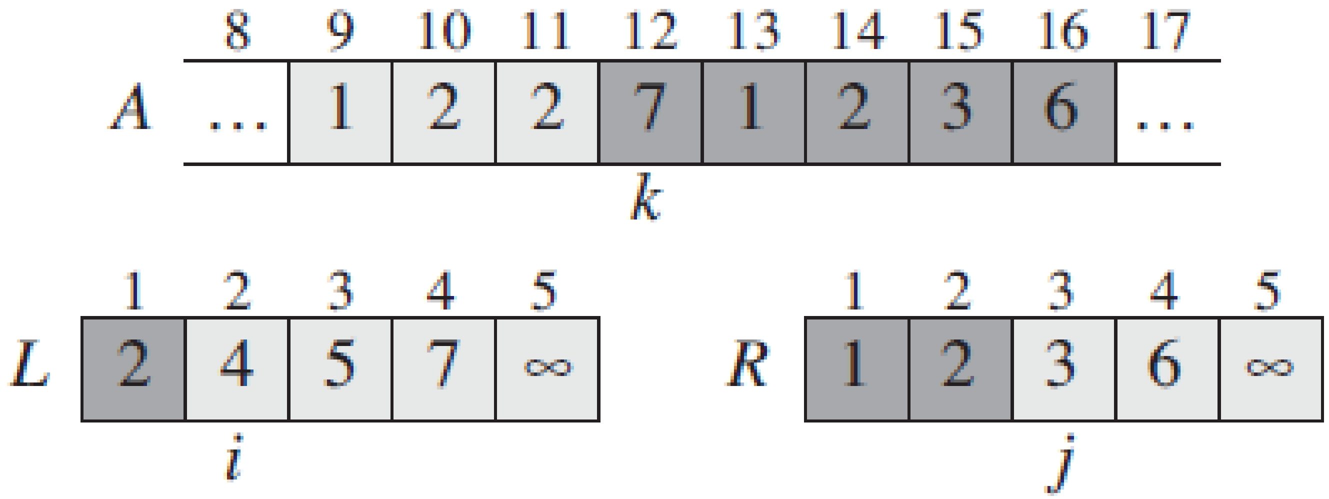
(b)

مرتب سازی ادغامی merge sort



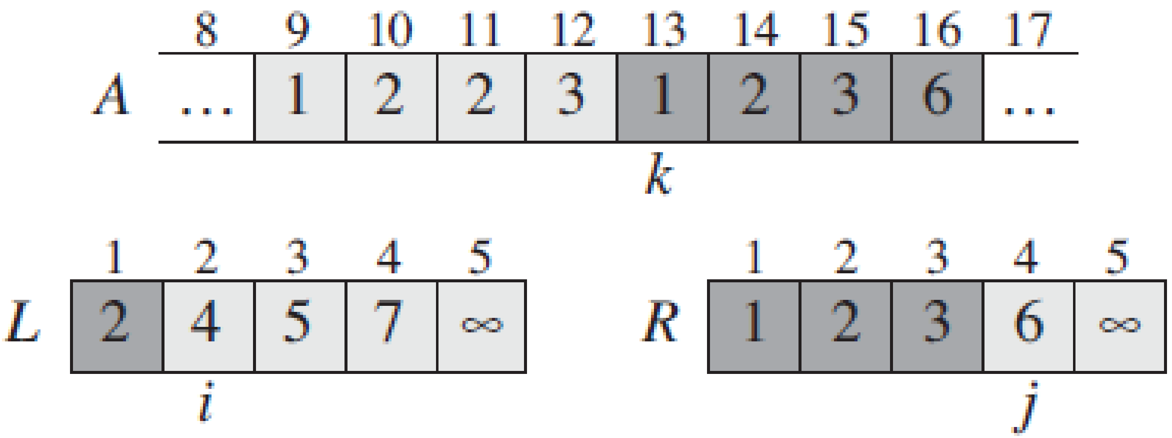
(c)

مرتب سازی ادغامی merge sort



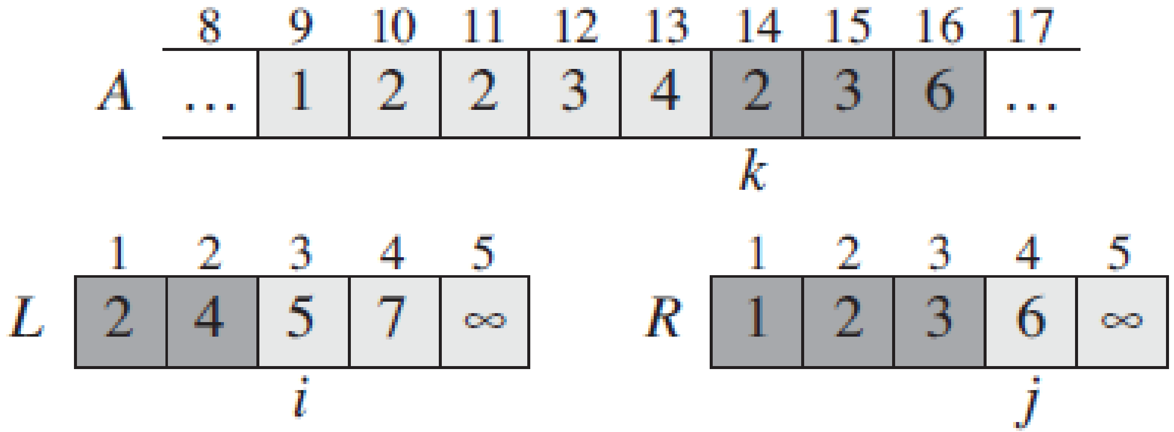
(d)

مرتب سازی ادغامی merge sort



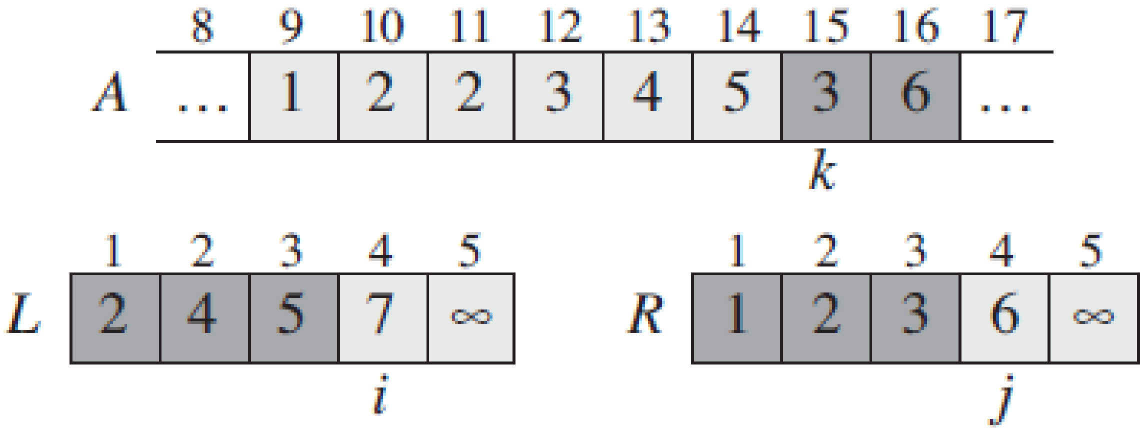
(e)

مرتب سازی ادغامی merge sort



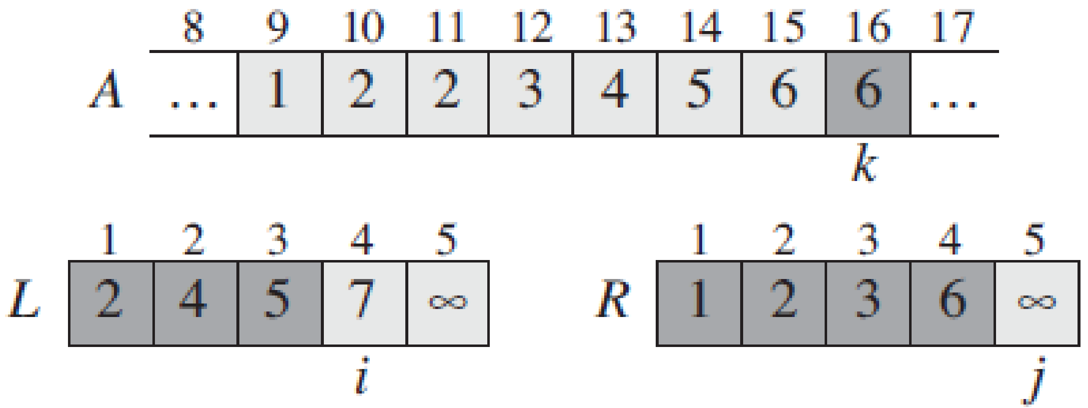
(f)

مرتب سازی ادغامی merge sort



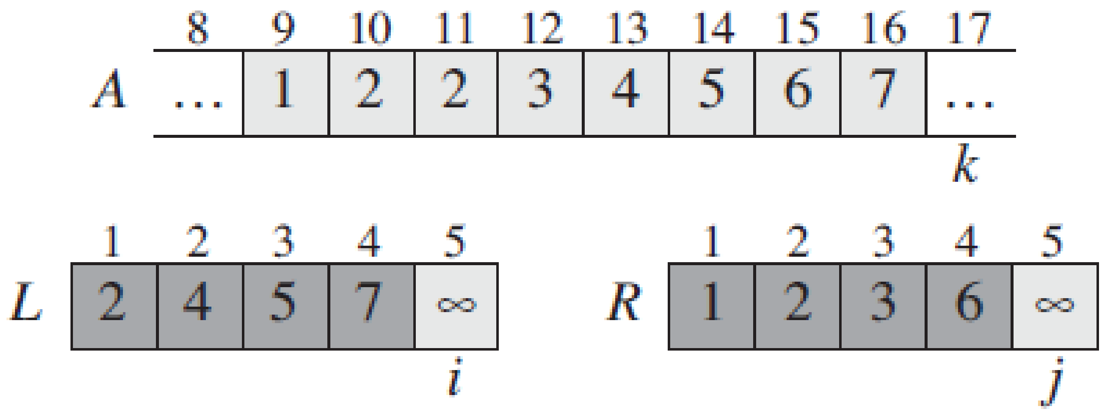
(g)

مرتب سازی ادغامی merge sort



(h)

مرتب سازی ادغامی merge sort



(i)

merge sort مرتب سازی ادغامی

MERGE(A, p, q, r)

```
n1 = q - p + 1                                  subarray A[p..q]
n2 = r - q                                      A[q+1..r].
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
```

تحلیل پیچیدگی مرتب سازی 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
```

زمان ثابت

$\Theta(n_1 + n_2) = \Theta(n)$

$\Theta(n)$

تحلیل پیچیدگی مرتب سازی ادغامی merge sort

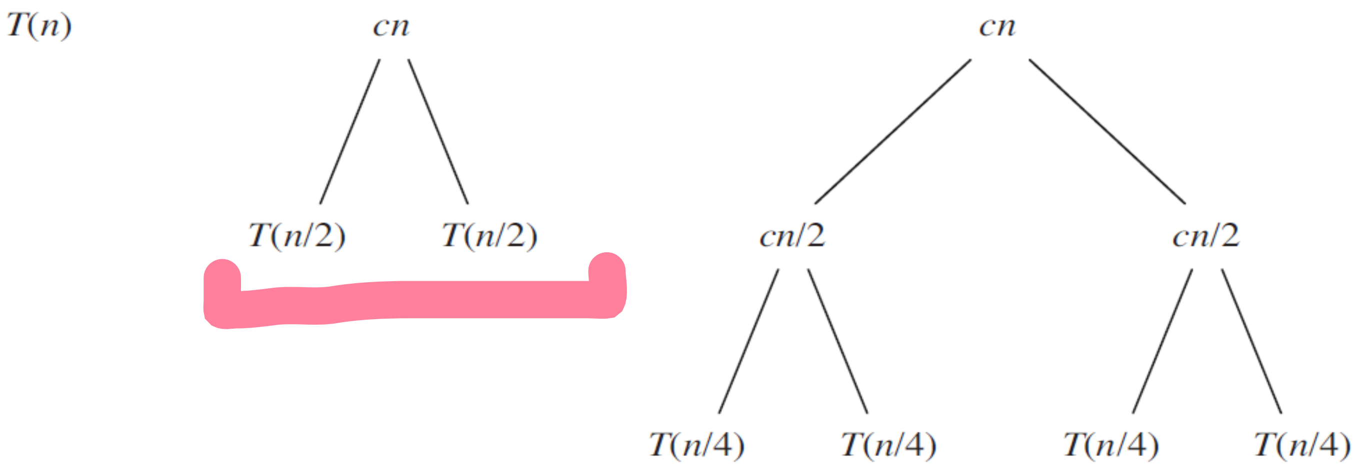
```
MERGE-SORT(A, p, r)
  if p < r
      q = ⌊(p + r)/2⌋
      MERGE-SORT(A, p, q)
      MERGE-SORT(A, q + 1, r)
      MERGE(A, p, q, r)
```

- Recursion equation for recursive algorithms

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

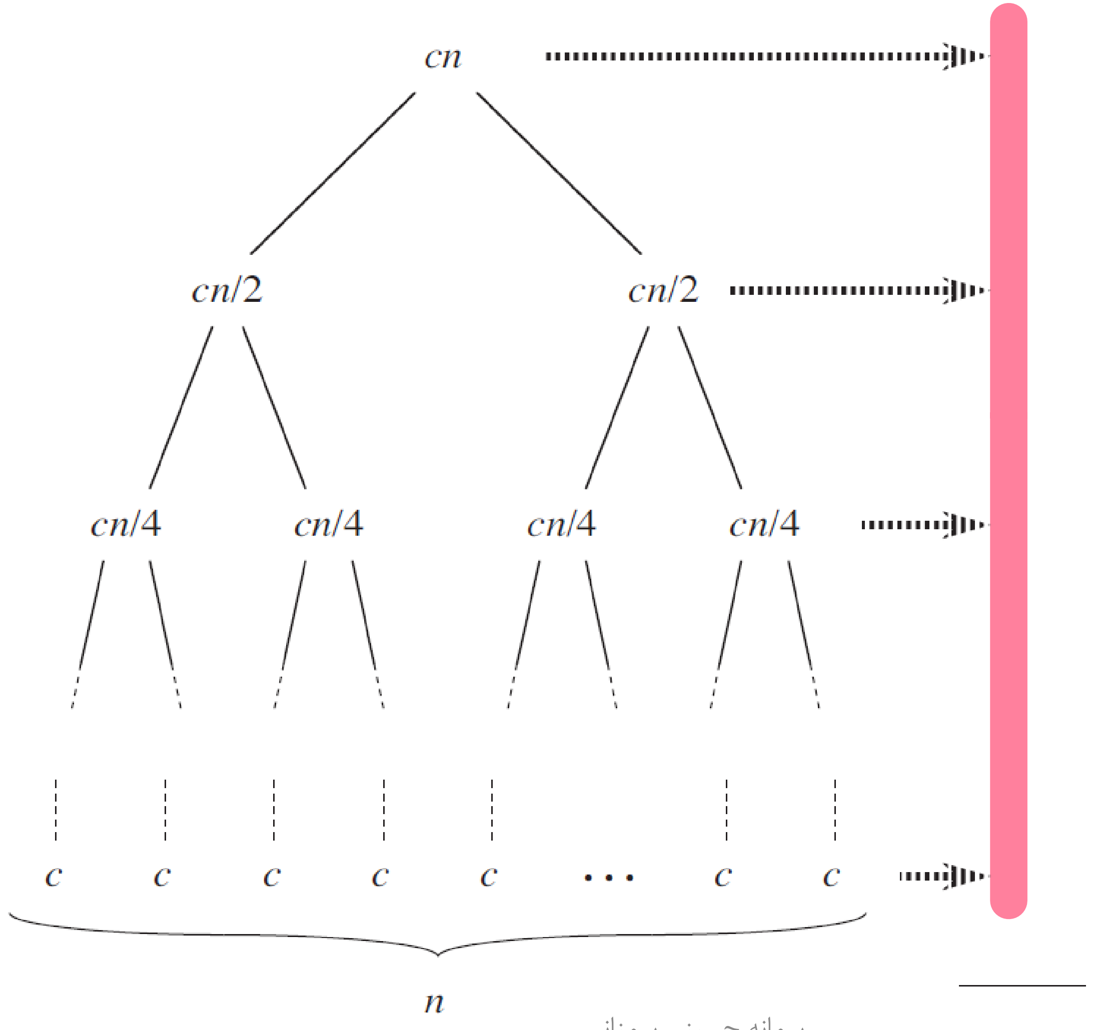
تحلیل پیچیدگی مرتب سازی ادغامی merge sort

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



(a) (b) (c)

تحلیل پیچیدگی مرتب سازی ادغامی merge sort



Total:

$$cn(\lg n + 1) = cn \lg n + cn$$

$$\Theta(n \lg n).$$