

دەم: مرتب سازى

ساختمان داده‌ها و الگوریتم

مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان



دانشگاه شهید بهشتی
دانشگاه شهید بهشتی

Kolektivy vědecké
Kolonialismus

Kolonié
Komenský, J. A. - ...

Komenský, J. A. pedagogika
Kompromisy politické

Komsomol
Kommunismus P.D.

Komunikace interpersonální
Komunismus - ...

Komunismus – přechod od sociální
Komunistická strana

13

14

15

16

17

18

Komun. Strana Číny
Kommun. Strana Německa

KS Německa – programy
Končeková – Veselá, L.

Konfetiny
Konflikt vojenský

Konflikty
Konflikt

Konspekty
Konstrukce

Konstrukce stavební
Kontrapunkt

19

20

21

22

23

24

Konfrarevoluce
Kopaná ČSR (po r. 1968)

Kopaná ČSSR
Korea lid. dem.

Kontologie

Kostky

Kouf Inbūt

25

26

27

28

29

30

Kofeni
Kovy -

Kovy – nauka o materiálu
Kovy - zpracování

Kovy - zpracování
Kozuchów

Kozy
Krakow

Krakov - průvodce
Krawc, B.

Kraby
Kreslení stavební

31

32

33

34

35

36

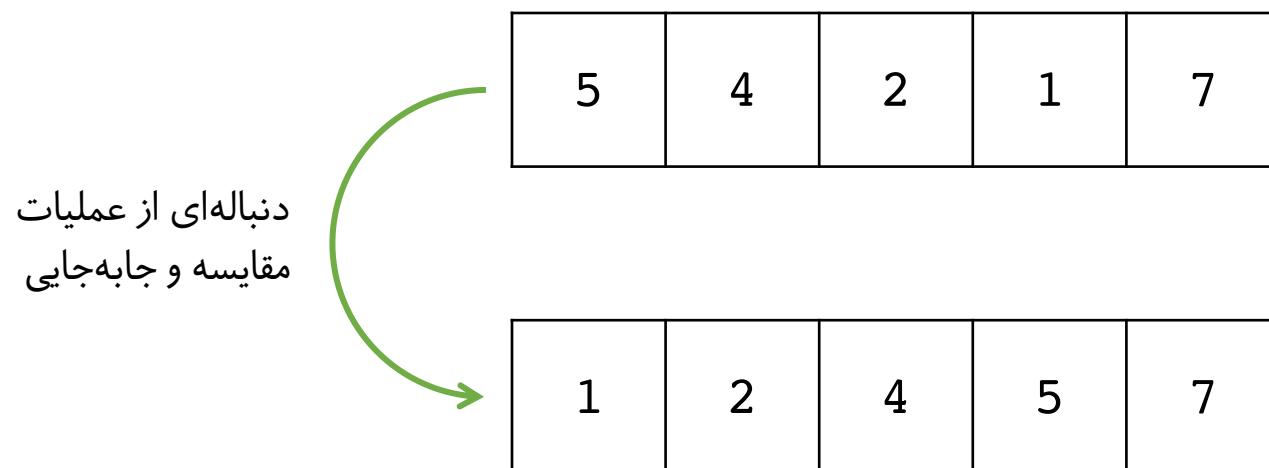
مسئله

- مجموعه از اشیاء داریم که برای آنها رابطه‌ای ترتیب‌پذیر (نه لزوماً کوچک‌تر و بزرگ‌تر) برقرار است، آنها را در لیستی به طور مرتب بچینید.



مرتب‌سازی

- عملیات مرتب‌سازی برای هر آرایه «عددی» از دو عمل مقایسه و جابه‌جایی (تعویض) تشکیل شده است.
- آن قسمت از عملیات مرتب‌سازی که مقایسه برمبنای آن انجام می‌شود، کلید نامیده می‌شود.



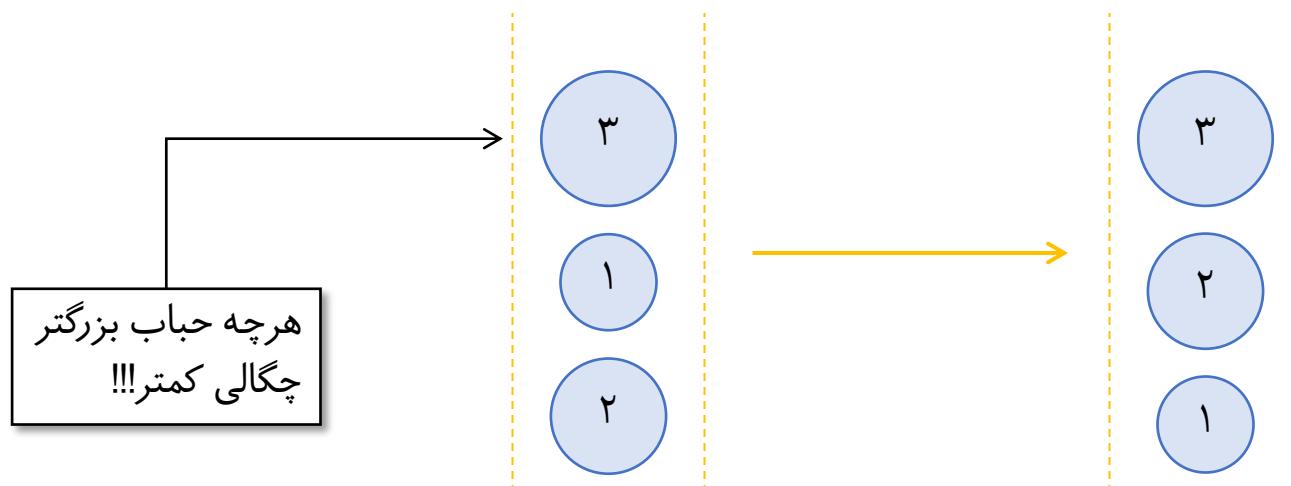
مرتب‌سازی

- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- Quick Sort
- Heap Sort
- BST Sort
- Radix Sort
- مرتب‌سازی حبابی
- مرتب‌سازی درجی
- مرتب‌سازی انتخابی
- مرتب‌سازی ادغامی
- مرتب‌سازی سریع
- مرتب‌سازی هرم
- مرتب‌سازی درخت جستجو دودویی
- مرتب‌سازی مبنایی

مرتب‌سازی حبابی

Bubble Sort

در این مرتب‌سازی در هر مرحله (گذر) با شروع از ابتدا یا انتهای هر عنصر با عنصر بعدی ($a[i]$ با $a[i+1]$) یا هر عنصر با عنصر قبلی ($a[i]$ با $a[i-1]$) مقایسه می‌شود و در صورتی که ترتیب آنها مناسب نباشد با هم جایه‌جا می‌شوند تا بهترین عنصر به سمت انتهای یا ابتدای آرایه هدایت شود در نهایت با $n-1$ گذر آرایه مرتب خواهد شد.



مرتب‌سازی حبابی

Bubble Sort

گذر	تعداد مقایسه	تعداد جابه‌جایی
1	$n-1$	$0 \leq \leq n-1$
2	$n-2$	$0 \leq \leq n-2$
\vdots	\vdots	\vdots
i	$n-i$	$0 \leq \leq n-i$
\vdots	\vdots	\vdots
$n-1$	1	$0 \leq \leq 1$
مجموع	$n(n-1)/2$	$0 \leq \text{تعداد جابه‌جایی} \leq n(n-1)/2$

بزرگترین عنصر در خانه $a[n]$ قرار می‌گیرد

دومین بزرگترین عنصر در خانه $a[n-1]$ قرار می‌گیرد

مرتب‌سازی حبابی

Bubble Sort

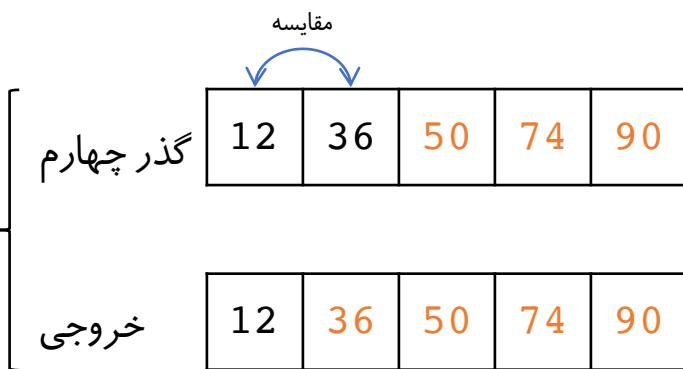
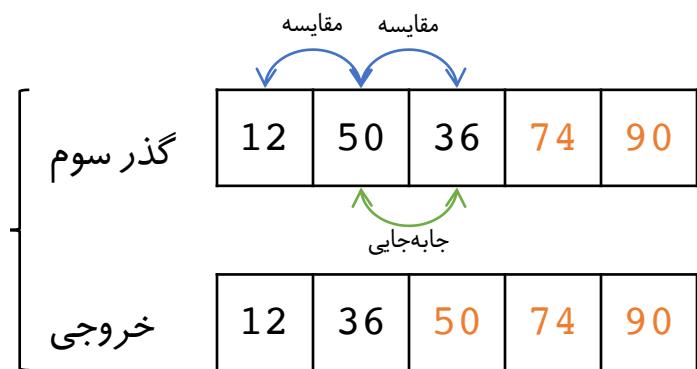
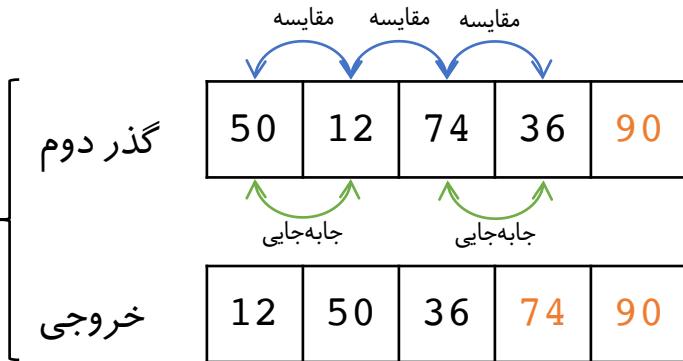
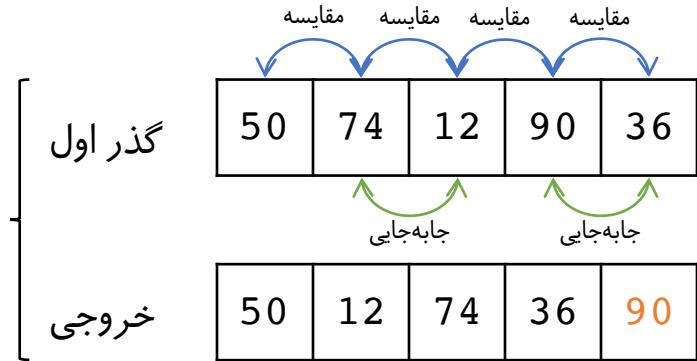
در این مرتب‌سازی در هر مرحله (گذر) با شروع از ابتدا یا انتهای هر عنصر با عنصر بعدی ($a[i]$ با $a[i+1]$) یا هر عنصر با عنصر قبلی (i) تب آنها مناسب نباشد با هم جابه‌جا می‌شوند تا بهترین عهایت با $n-1$ گذر آرایه مرتب خواهد شد.

```
def bubble_sort(a,n):
    flag=True
    i=0
    while(i<=n and flag==True):
        flag=False
        for j in range(1,n-i):
            if(a[j]>a[j+1]):
                flag=True
                temp=a[j]
                a[j]=a[j+1]
                a[j+1]=temp
        i=i+1
    return a
```

هر چه حباب بزرگتر
چگالی کمتر!!!

مرتب‌سازی حبابی

50	74	12	90	36
----	----	----	----	----



مرتب‌سازی حبابی

Bubble Sort

گذر	تعداد مقایسه	تعداد جابه‌جایی	با $n-1$ مقایسه و ۰ جابه‌جایی، بیشینه اول به خانه آخر منتقل می‌شود.	با $n-2$ مقایسه و ۰ جابه‌جایی، بیشینه دوم به ماقبل خانه آخر منتقل می‌شود.
1	$n-1$	0		
2	$n-2$	0		
:	:	:		
$n-1$	1	0		
مجموع	$n(n-1)/2$	0		

بهترین حالت: ورودی مرتب

مقایسات: $n(n-1)/2 = O(n^2)$

جابه‌جایی: $0 = O(1)$

برای $n=5$ مرتب‌سازی حبابی این شکل است:

1	2	...	$n-1$	n
---	---	-----	-------	-----

مرتب‌سازی حبابی

Bubble Sort

گذر	تعداد مقایسه	تعداد جابه‌جایی	بدرین حالت: ورودی مرتب و معکوس مقایسات: $n(n-1)/2 = O(n^2)$
1	$n-1$	$n-1$	با $n-1$ مقایسه و $n-1$ جابه‌جایی، بیشینه اول به خانه آخر منتقل می‌شود.
2	$n-2$	$n-2$	با $n-2$ مقایسه و $n-2$ جابه‌جایی، بیشینه دوم به ماقبل خانه آخر منتقل می‌شود.
\vdots	\vdots	\vdots	
$n-1$	1	1	
مجموع	$n(n-1)/2$	$n(n-1)/2$	

مرتب‌سازی درجی

Insertion Sort

در این مرتب‌سازی در هر مرحله (گذر) فقط یک کلید در داخل آرایه قابل بررسی است. بدین صورت که کلید k_i باید در بین $1-1-i$ کلید مرتب شده (k_1, \dots, k_{i-1}) به گونه‌ای قرار گیرد تا لیست حاصل به طول i مرتب بماند. در نهایت با $n-1$ گذر آرایه مرتب خواهد شد.

نکته قابل توجه در مرتب‌سازی درجی آن است که تا گذر آخر $n-1$, محل هیچ کدام از عناصر ثابت نیست.

مرتب‌سازی درجی

Insertion Sort

گذر	تعداد مقایسه	تعداد جابه‌جایی	نتیجه
1	1	$0 \leq \leq 1$	لیست $a[1..2]$ مرتب
2	$1 \leq \leq 2$	$0 \leq \leq 2$	لیست $a[1..3]$ مرتب
:	:	:	:
$n-1$	$1 \leq \leq n-1$	$0 \leq \leq n-1$	لیست $a[1..n]$ مرتب
	$n - 1 \leq \leq \frac{n(n - 1)}{2}$ مقایسات	$0 \leq \leq \frac{n(n - 1)}{2}$ مقایسات	

مرتب‌سازی درجی

Insertion Sort

در این مرتب‌سازی در هر مرحله (گذر) فقط یک کلید در داخل آرایه قابل بررسی است. بدین صورت که کلید k_i باید در بین $1-1$ - i کلید متناسبه (k_1, \dots, k_{i-1}) به گهنه‌ای، قرار گردید تا لیست حاصل به طول i مرتب بماند. در

صر ثابت نیست.

```
def Insertion_sort(a,n):
    for i in range(2,n):
        j=i
        while(a[j]<a[j-1] and j>1):
            temp=a[j]
            a[j]=a[j+1]
            a[j+1]=temp
            j=j-1

    return a
```

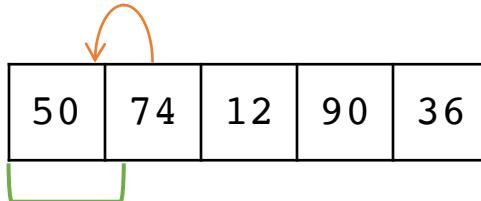
نهاشت با $n-1$ گذر آرایه

نکته قابل توجه در مرتب

مرتب‌سازی درجی

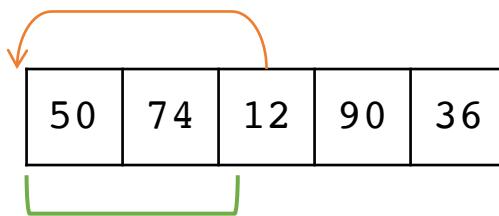
50	74	12	90	36
----	----	----	----	----

گذر اول



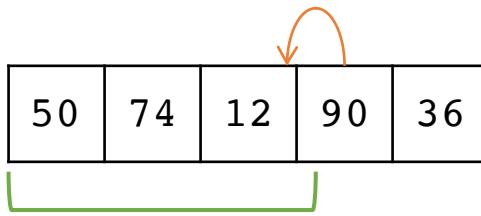
50	74	12	90	36
----	----	----	----	----

گذر دوم



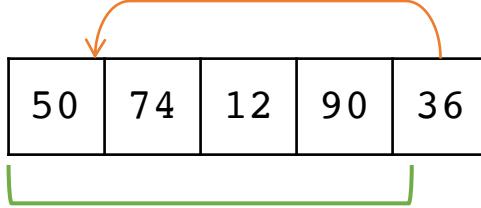
12	50	74	90	36
----	----	----	----	----

گذر سوم



12	50	74	90	36
----	----	----	----	----

گذر چهارم



12	36	50	74	90
----	----	----	----	----

مرتب‌سازی درجی

Insertion Sort

جابه‌جایی: $O(1)$

مقایسات: $O(n)$

بهترین حالت: ورودی مرتب

1	2	...	$n-1$	n
---	---	-----	-------	-----

گذر	تعداد مقایسه	تعداد جابه‌جایی
1	1	0
2	1	0
:	:	:
$n-1$	1	0
مجموع	$n(n-1)/2$	0

۱ با ۱ مقایسه شده و بدون جابه‌جایی در جای خود قرار می‌گیرد

۲ با ۲ مقایسه شده و بدون جابه‌جایی در جای خود قرار می‌گیرد

n با $n-1$ مقایسه شده و بدون جابه‌جایی در جای خود قرار می‌گیرد

مرتب‌سازی درجی

Insertion Sort

$$n(n-1)/2 = O(n^2)$$

$$ن(n-1)/2 = O(n^2)$$

بدترین حالت: ورودی مرتب و معکوس مقایسات:

n	n-1	...	2	1
---	-----	-----	---	---

گذر	تعداد مقایسه	تعداد جابه‌جایی
1	1	1
2	2	2
:	:	:
n-1	n-1	n-1
مجموع	$n(n-1)/2$	$n(n-1)/2$

n با n مقایسه شده و با 1 جابه‌جایی در جای خود قرار می‌گیرد

n-1 با n-1 مقایسه شده و با 2 جابه‌جایی در جای خود قرار می‌گیرد

1 با n-1...2 مقایسه شده و با n-1 جابه‌جایی در جای خود قرار می‌گیرد

مرتب‌سازی انتخابی

Selection Sort

در این مرتب‌سازی در هر مرحله (مثلاً گذر i) کوچک‌ترین یا بزرگ‌ترین کلید با $n-i$ مقایسه انتخاب شده سپس با i جایگزینی در محل واقعی خود قرار می‌گیرد. در نهایت با $n-1$ گذر آرایه مرتب خواهد شد.

گذر	تعداد مقایسه	تعداد جابه‌جایی
1	$n-1$	1
2	$n-2$	1
:	:	:
$n-1$	1	1
	$\frac{n(n - 1)}{2}$	$n-1$

Selection Sort

مرتب‌سازی انتخابی

در این مرتب‌سازی در هر مرحله (مثلاً گذر i) کوچک‌ترین یا بزرگ‌ترین کلید با $n-i$ مقایسه انتخاب شده سپس با i جایگایی در i -اهد شد.

گذر	تعداد مقایسه
1	$n-1$
2	$n-2$
:	:
$n-1$	1
	$\frac{n(n - 1)}{2}$

```
def Insertion_sort(a,n):
    for i in range(1,n):
        minimum=inf
        index=-1
        for j in range(i,n):
            if(a[j]<min):
                minimum=a[j]
                index=j
        a[index]=a[i]
        a[i]=minimum
    return a
```

$n-1$

مرتب‌سازی انتخابی

50	74	12	90	36
----	----	----	----	----

گذر اول

50	74	12	90	36
----	----	----	----	----

12	74	50	90	36
----	----	----	----	----

گذر دوم

12	74	50	90	36
----	----	----	----	----

12	36	50	90	74
----	----	----	----	----

گذر سوم

12	36	50	90	74
----	----	----	----	----

12	36	50	90	74
----	----	----	----	----

گذر چهارم

12	36	50	90	74
----	----	----	----	----

12	36	50	74	90
----	----	----	----	----

مرتب‌سازی انتخابی

Selection Sort

گذر	تعداد مقایسه	تعداد جابه‌جایی	برای همه حالات یکسان است.
1	$n-1 = O(n)$	1	با 1 مقایسه، کمینه اول(عدد 1) انتخاب می‌شود و با 1 جابه‌جایی در جای خودش [1]a قرار می‌گیرد.
2	$n-2$	1	با 2 مقایسه، کمینه دوم(عدد 2) انتخاب می‌شود و با 1 جابه‌جایی در جای خودش [2]a قرار می‌گیرد.
:	:	:	
$n-1$	1	1	با 1 مقایسه، کمینه n ام(عدد n) انتخاب می‌شود و با 1 جابه‌جایی در جای خودش [n]a قرار می‌گیرد.
	$\frac{n(n - 1)}{2}$	$n-1$	مقایسات: $n(n-1)/2 = O(n^2)$

مرتب‌سازی ادغامی

Merge Sort

مرتب‌سازی ادغامی ۲ ایده اصلی را با هم ترکیب می‌کند تا زمان اجرایش تقویت شود:

- یک لیست کوچک از گام‌های کمتری برای مرتب‌کردن نسبت به یک لیست بزرگ استفاده می‌کند.
- برای مرتب کردن دو لیست مرتب‌شده نسبت به دو لیست نامرتب گام‌های کمتری نیاز می‌باشد به عنوان مثال اگر این لیست‌ها مرتب باشند شما مجبور هستید تا هر لیست را فقط یکبار پیمایش کنید.
- آنها را دو لیست‌پیوندی مرتب‌شده در نظر بگیرید و با الگوریتم ادغام گفته شده از $O(n)$ ادغام کنید.

Merge Sort

مرتب‌سازی ادغامی

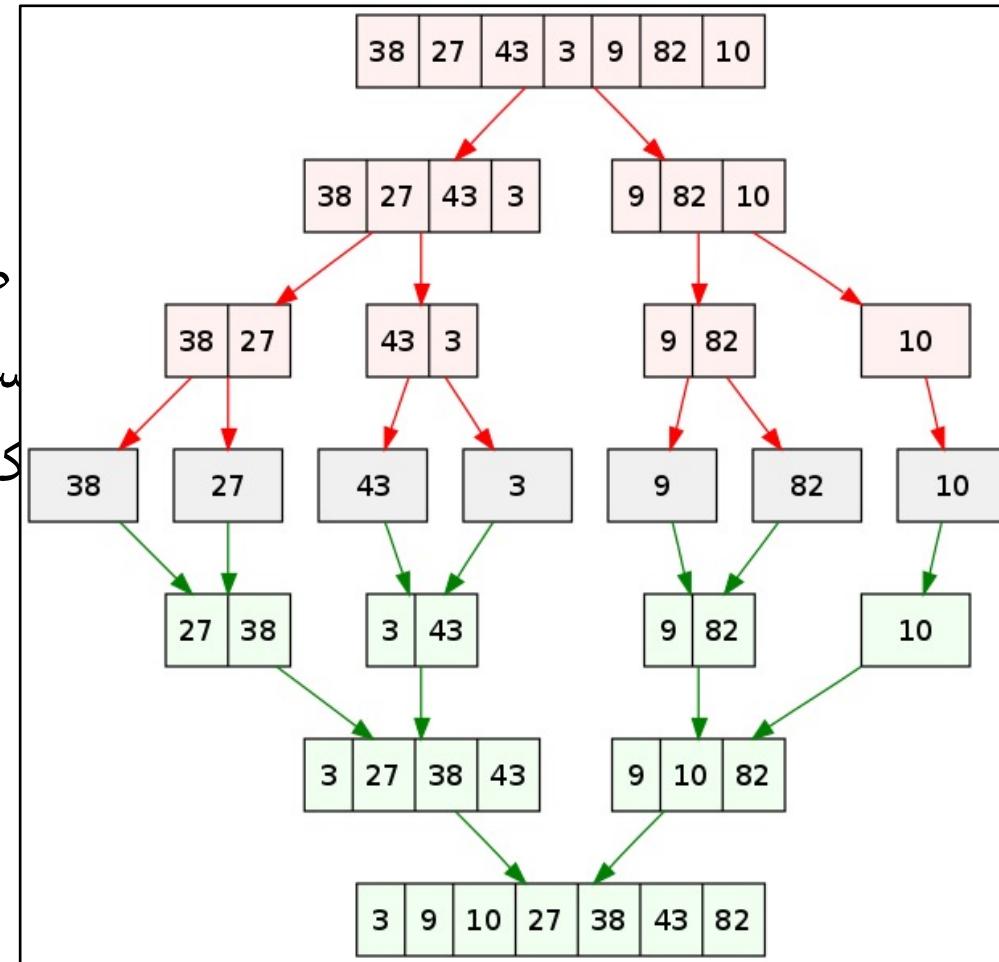
به طور عمومی یک الگوریتم مرتب‌سازی ادغامی بدین صورت کار می‌کند:

- .i. اگر طول لیست ۰ یا ۱ باشد آن پیش از این مرتب شده است در غیر این صورت لیست نامرتب را به دو زیرلیست که اندازه آنها در حدود نصف سایز لیست اولیه است تقسیم می‌کند.
- .ii. هر زیر لیست را به طور بازگشتی با صدا کردن merge sort مرتب می‌کند.
- .iii. دو تا دو تا زیر لیست‌ها را از آخر ادغام می‌کند تا به یک لیست برسد.

Merge Sort

مرتب‌سازی ادغامی

صورة
ت اولیه است تقسیم می‌کند.
کند.



به طور عمومی یک الگوریتم

- اگر طول لیست ۰ باشد لیست نامرتب را به همراه زیر لیست را بهمراه دو تا دو تا زیر لیست
- .ii
- .iii
- .iv

Merge Sort

مرتب‌سازی ادغامی

به طور عمومی یک الگوریتم مرتب‌سازی ادغامی بدین صورت کار می‌کند:

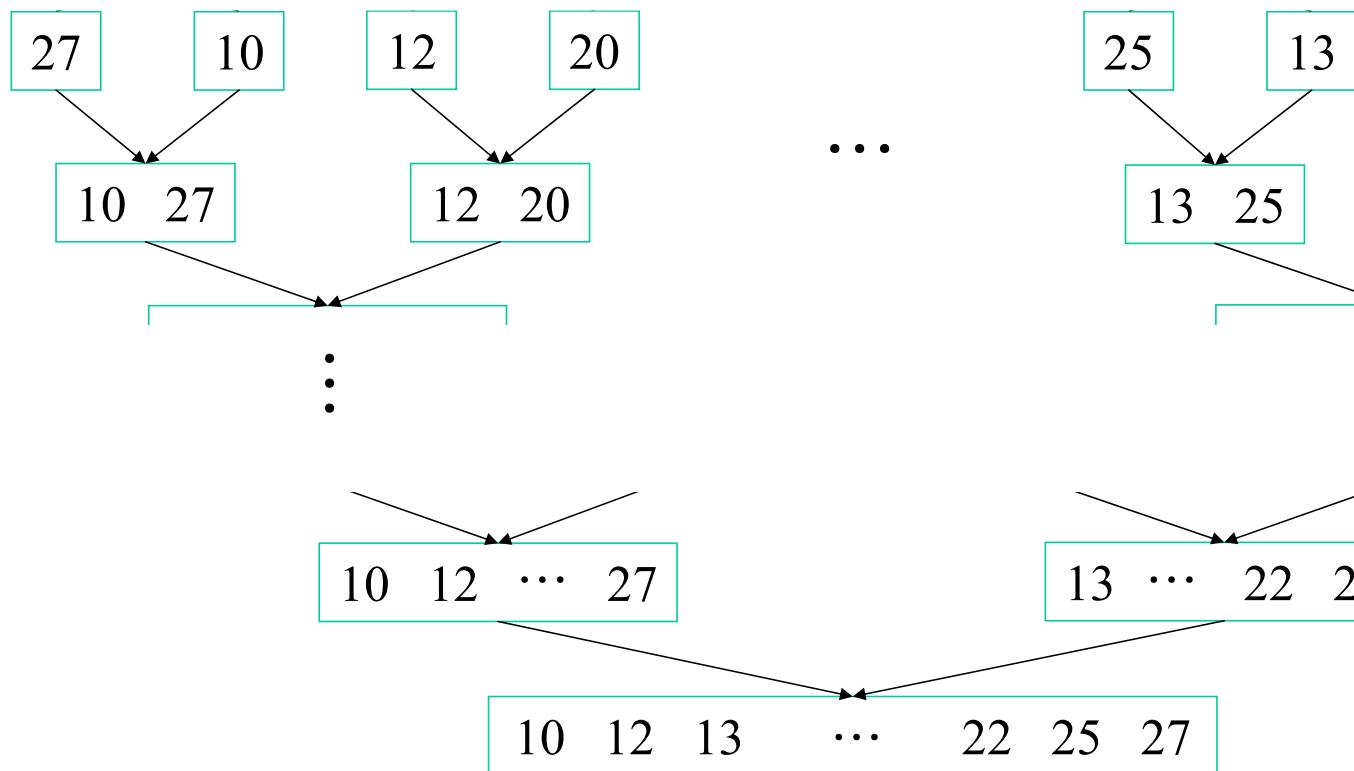
- .i. اگر طول لیست ۱ باشد آن پیش از این مرتب شده است در غیر این صورت لیست نامرتب است تقسیم می‌کند.
- .ii. لیست نامرتب
- .iii. هر زیر لیست
- .iv. دو تا دو تا ز

```
def merge_sort(a,n):  
    if(n<=1):  
        return a  
    mid=n/2  
    l=merge_sort(a[0:mid],n/2)  
    r=merge_sort(a[mid:n],n/2)  
    a=merge(l,r) ←  
    return a
```

همان تابع ادغام برای دو
لیست پیوندی مرتب

Merge Sort

مرتب‌سازی ادغامی

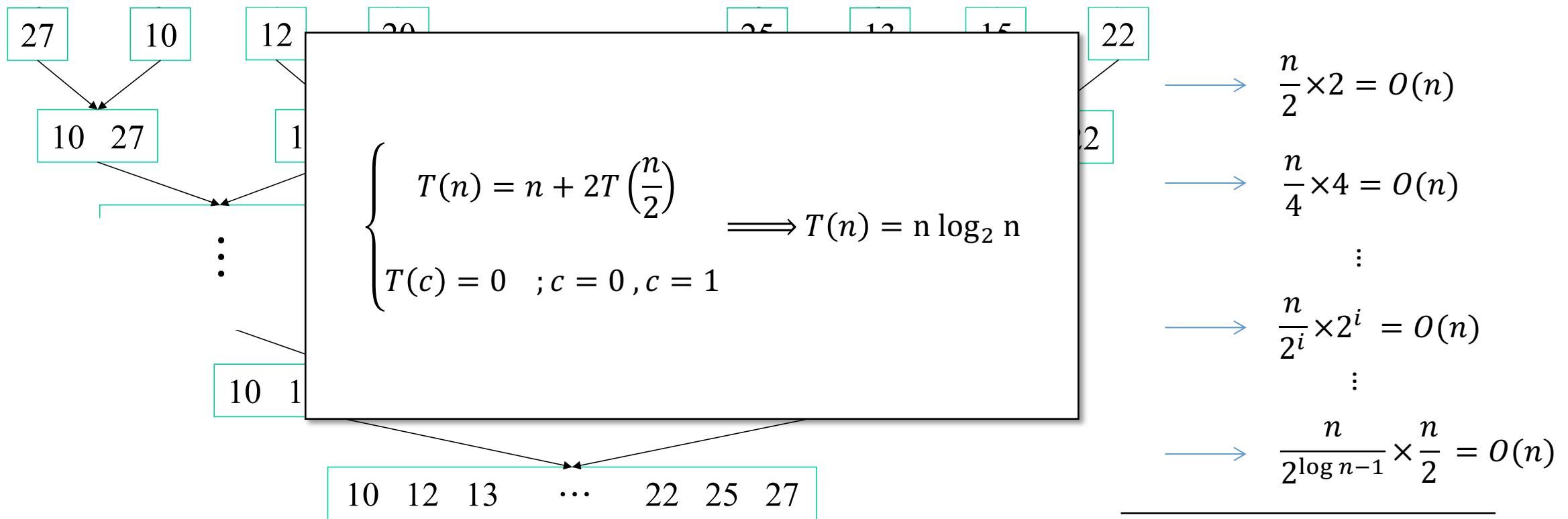


$$\begin{aligned} \frac{n}{2} \times 2 &= O(n) \\ \frac{n}{4} \times 4 &= O(n) \\ &\vdots \\ \frac{n}{2^i} \times 2^i &= O(n) \\ &\vdots \\ \frac{n}{2^{\log n - 1}} \times \frac{n}{2} &= O(n) \end{aligned}$$

$$\log_2 n \times O(n) = O(n \log n)$$

Merge Sort

مرتب‌سازی ادغامی



مرتب‌سازی سریع

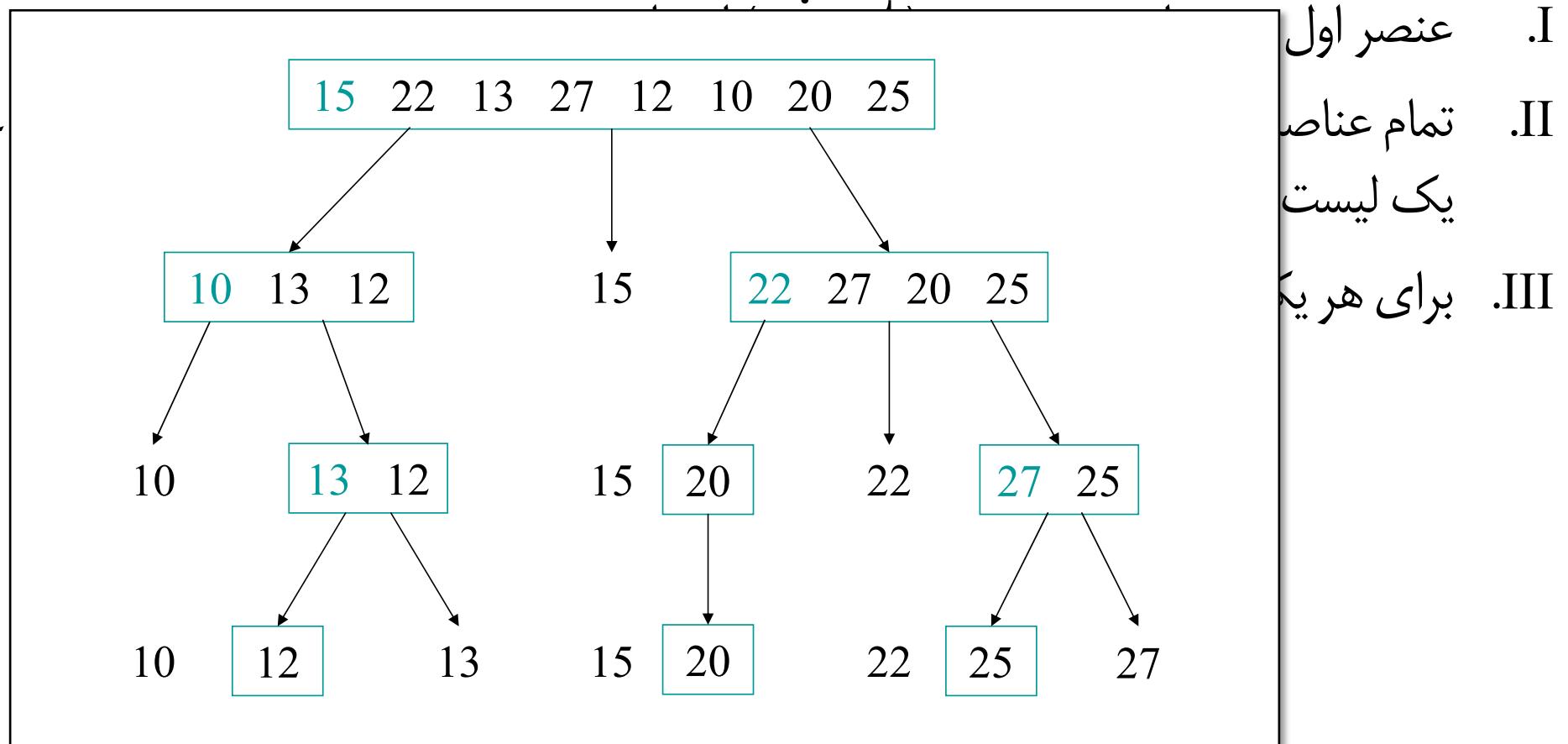
Quick Sort

- I. عنصر اول به عنوان عنصر محور(pivot) انتخاب می‌شود.
- II. تمام عناصر با محور مقایسه می‌شوند و با عناصر کوچکتر یک لیست تشکیل می‌شود و با عناصر بزرگتر یک لیست دیگر.
- III. برای هر یک از لیست‌های بدست آمده عملیات قبلی تکرار می‌شود.

مرتب‌سازی سریع

Quick Sort

عناصر بزرگتر



مرتب‌سازی سریع

Quick Sort

```
def quick_sort(a,l,r):
    if(l<r):
        i=l
        j=r+1
        pivot=a[left]
        repeat:
            repeat:
                i=i+1
                until a[i]>=pivot
            repeat:
                j=j-1
                until a[j]<=pivot
            if i < j:
                swap(a[i],a[j])
        until i>=j
        swap(a[left],a[j])
        quick_sort(a,left,j-1)
        quick_sort(a,j+1,right)
```

شده.
د.
یم.
می شود.

- روش انجام کار در یک مرتب‌سازی سریع:
- ۱- عنصر اول را به طور اندیس های ۰ و ۱ بگذارید.
 - ۲- اندیس های ۰ و ۱ را با شروع از اندیس ۰: و اندیس ۱: از مقدار اینگاه متفاوت نگیرید.
 - ۳- بعد از اجرای مرحله اول (الف) اگر $j > i$ آنگاه مقدار اندیس j را با مقدار اندیس i تعادل نمایید.

مرتب‌سازی سریع

Quick Sort

- روش انجام کار
- ۱- عنصر اول را انتخاب کنید.
 - ۲- اندیس‌های i و j را از ۰ و ۹ آغاز کنید.
 - ۳- بعد از اجرای مرحله دو، اگر $i < j$ باشد، مرحله دو را از مجدد اجرا کنید.
 - ۴- اگر $i = j$ باشد، مرحله دو را از مجدد اجرا کنید.

40	20	10	80	60	50	7	30	100
0	1	2	3	4	5	6	7	8

انتخاب عنصر محور:

40	20	10	80	60	50	7	30	100
0	1	2	3	4	5	6	7	8

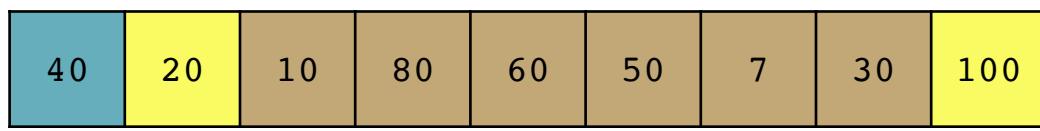
مرتب‌سازی سریع

Quick Sort

- 1- یک زیر آرایه تنها عناصر بزرگتر یا مساوی pivot را نگه داری می کند.
- 2- یک زیر آرایه دیگر، عناصری را نگه می دارد که کوچتر از pivot باشند.

روش انجام کار
۱- عنصر اول ر
۲- اندیس های
اندیس i: با
اندیس j: با
۳- بعد از اجرای
الف) اگر $j < i$ آر
ب) اگر $i > j$

pivot=0

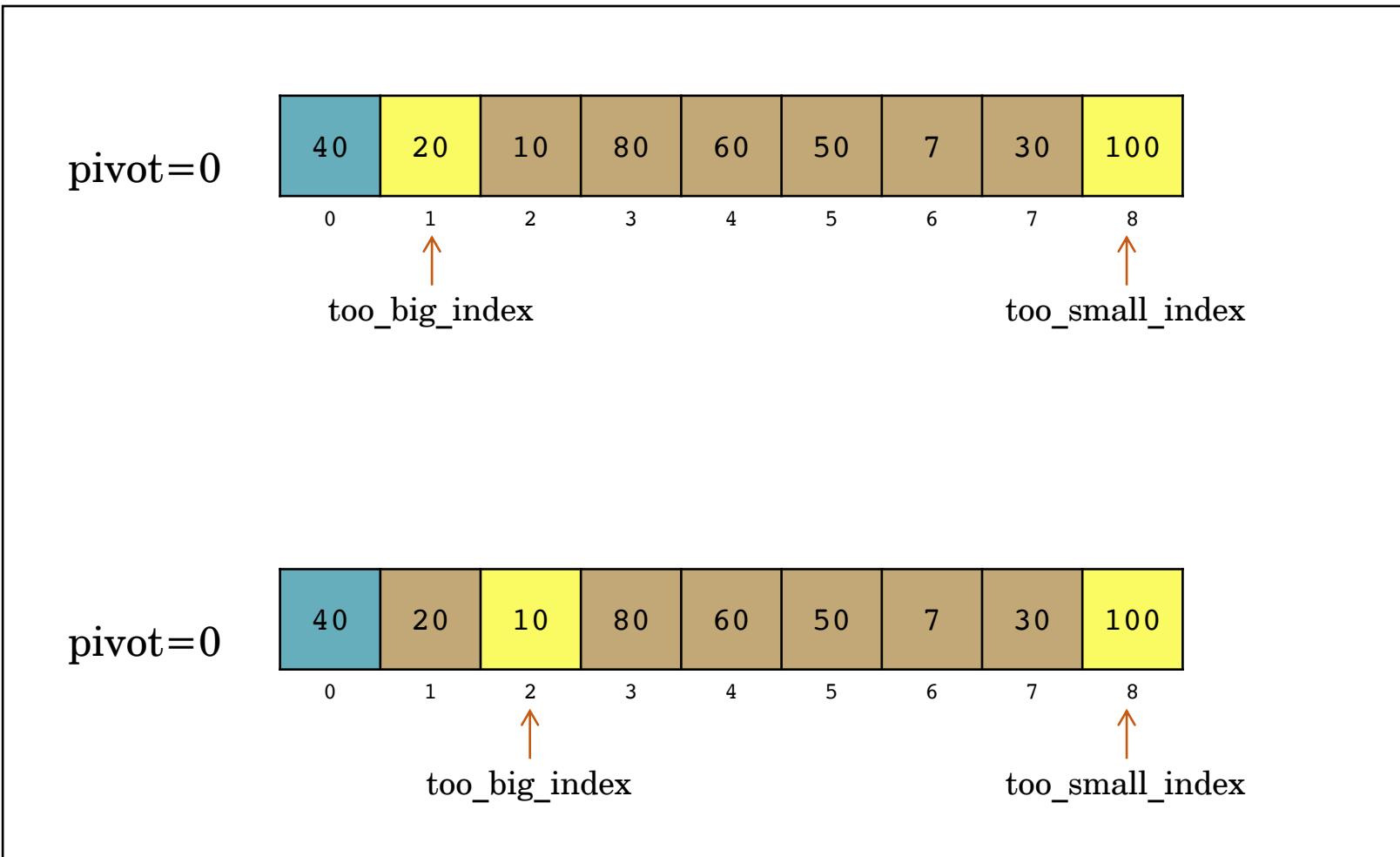


too_big_index

too_small_index

مرتب‌سازی سریع

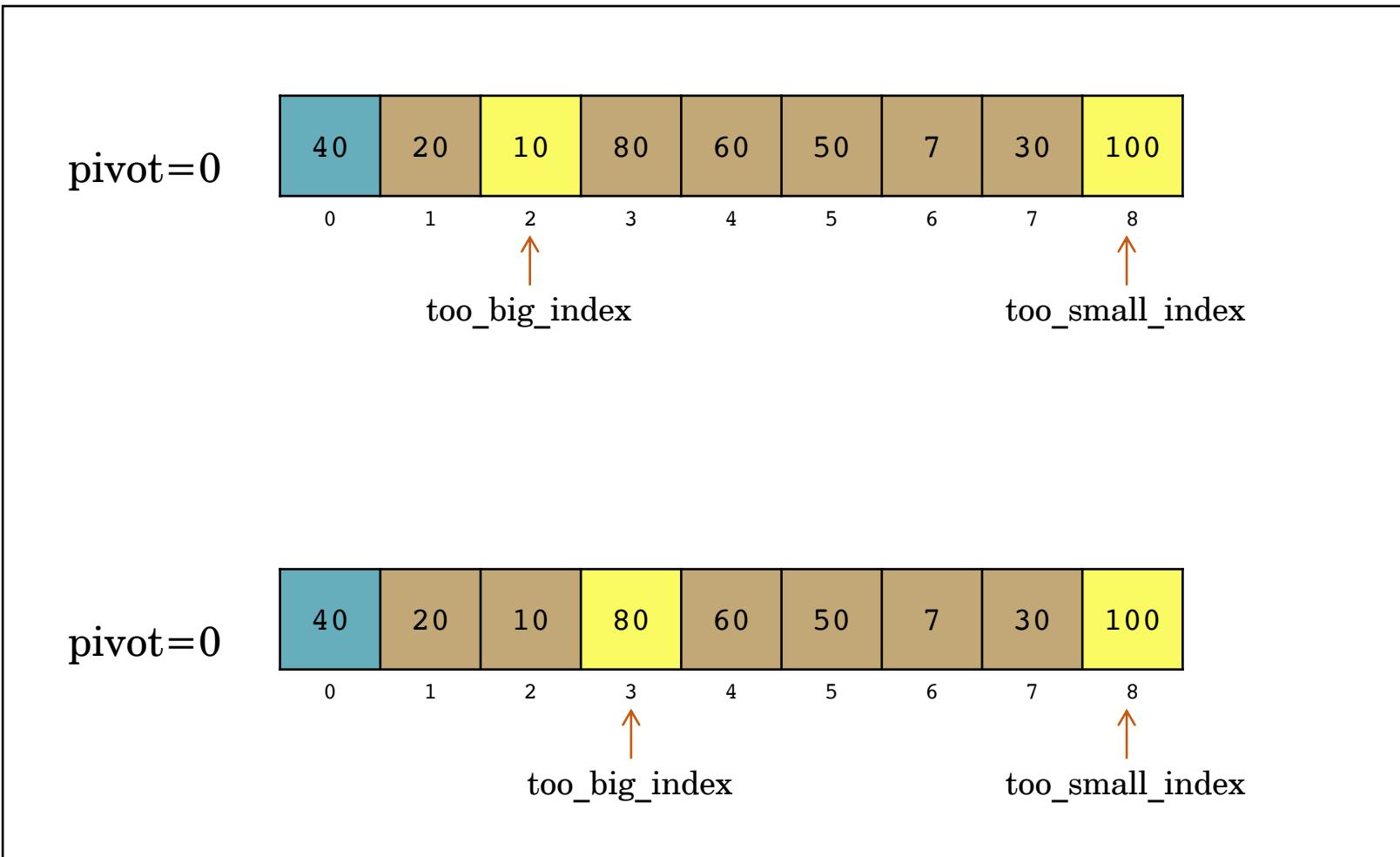
Quick Sort



- روش انجام کار
- ۱- عنصر اول را از اندیس های i : با اندیس j : با
 - ۲- بعد از اجرای (الف) اگر $j < i$ آنرا (ب) اگر $j > i = j$ با

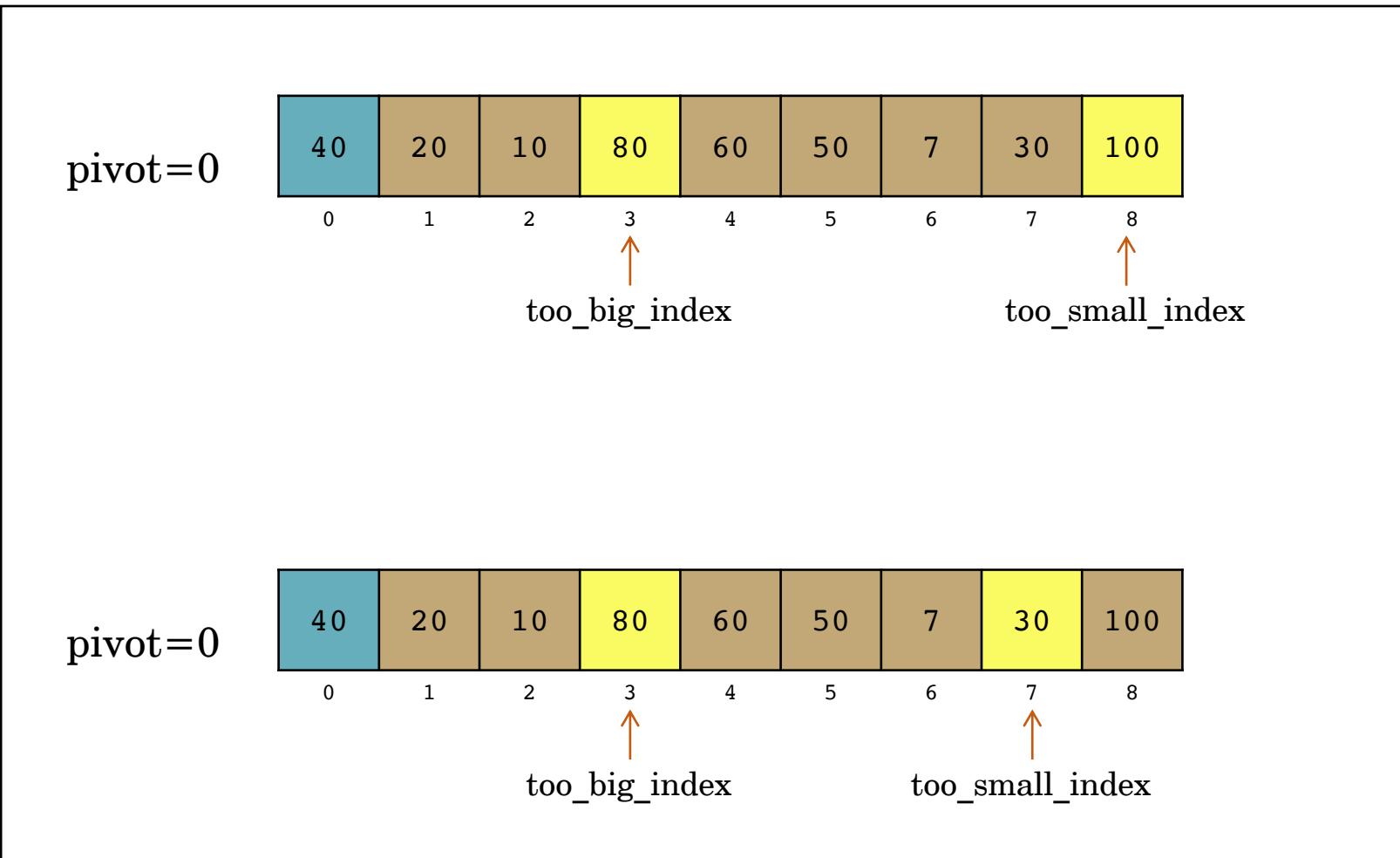
مرتب‌سازی سریع

Quick Sort



مرتب‌سازی سریع

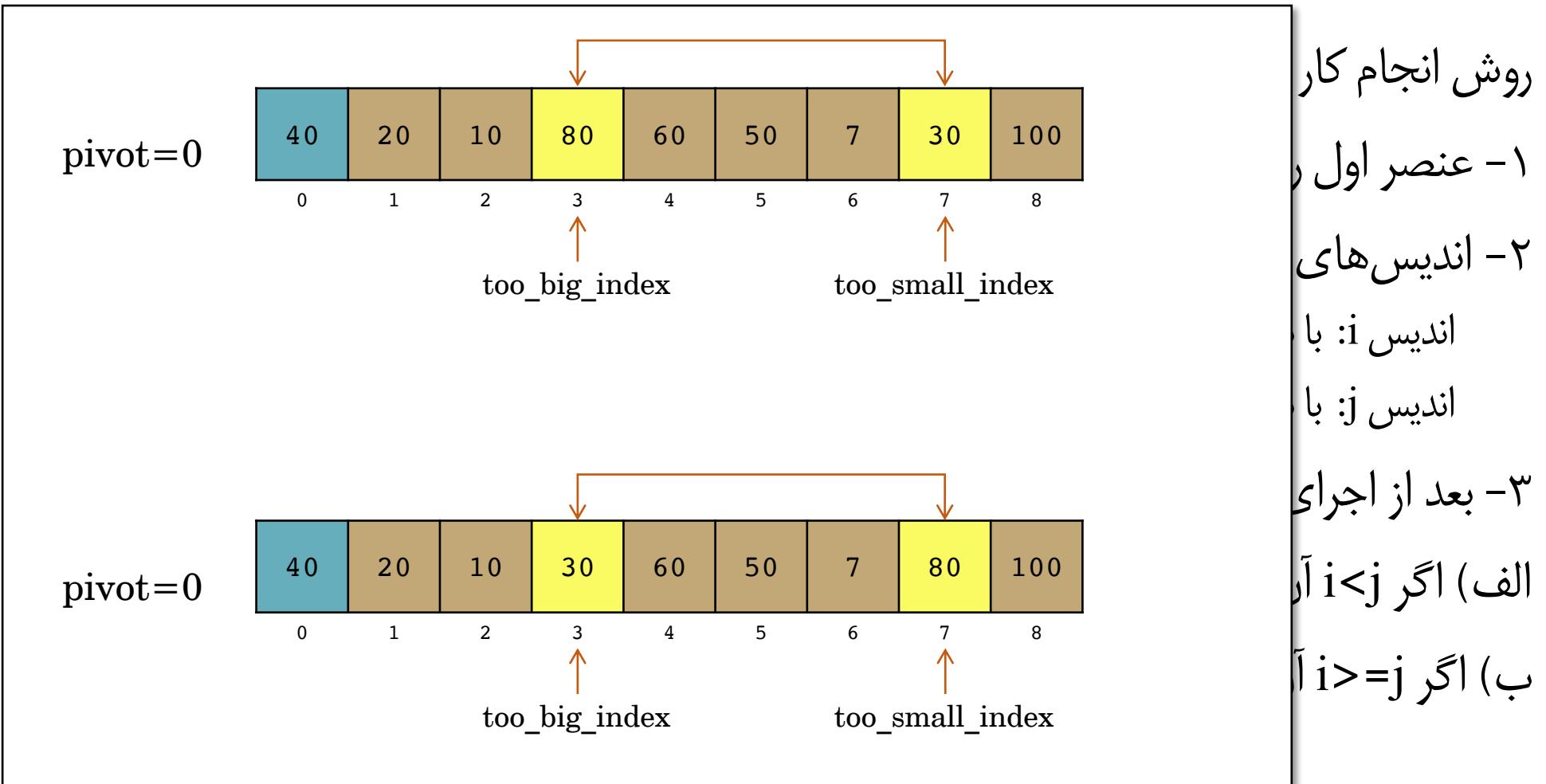
Quick Sort



- روش انجام کار
- ۱- عنصر اول را اندیس‌هایی از i: با اندیس j: با
 - ۲- بعد از اجرای (الف) اگر $j < i$ آنرا (ب) اگر $i > j$ آنرا

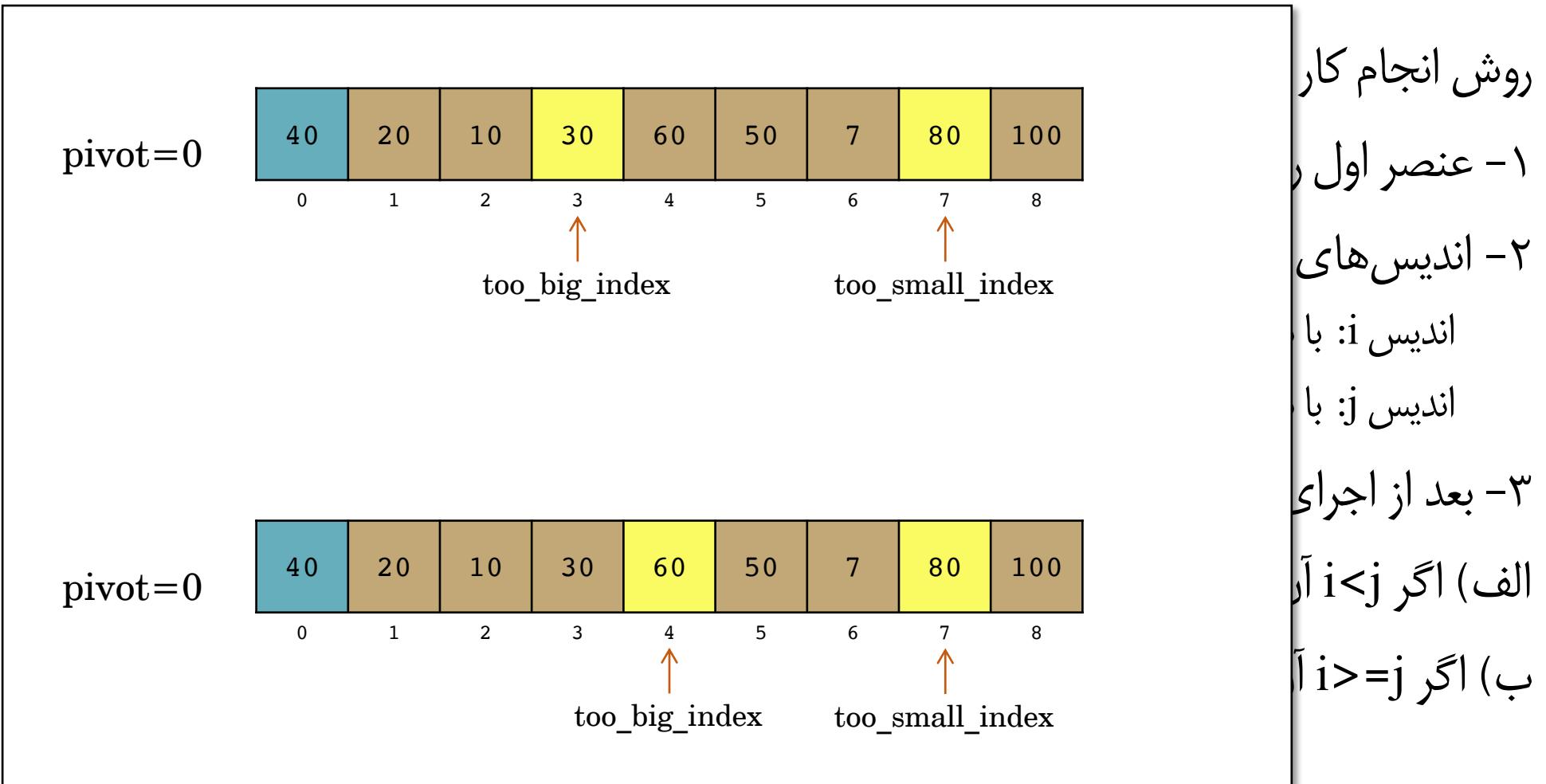
مرتب‌سازی سریع

Quick Sort



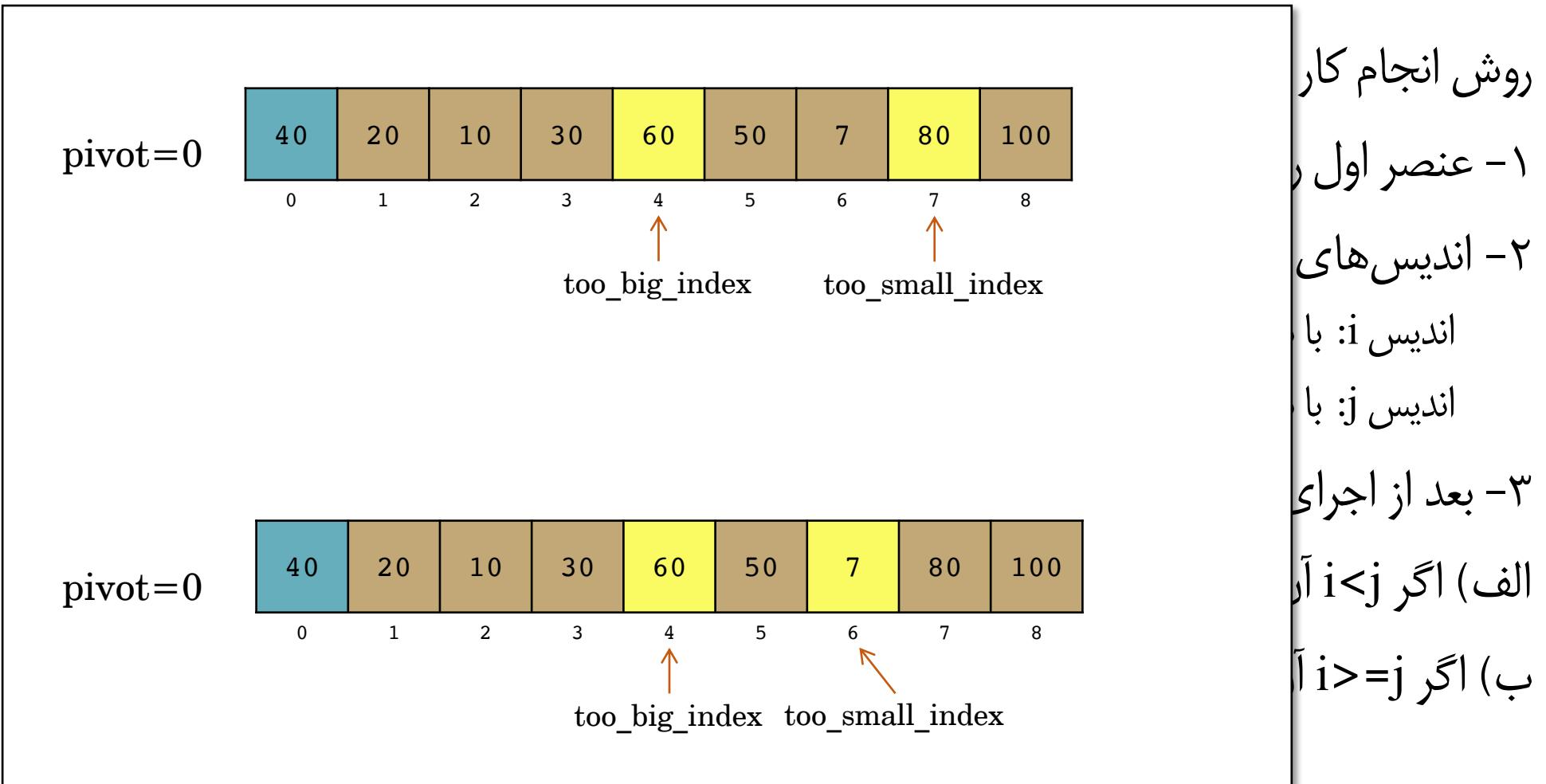
مرتب‌سازی سریع

Quick Sort



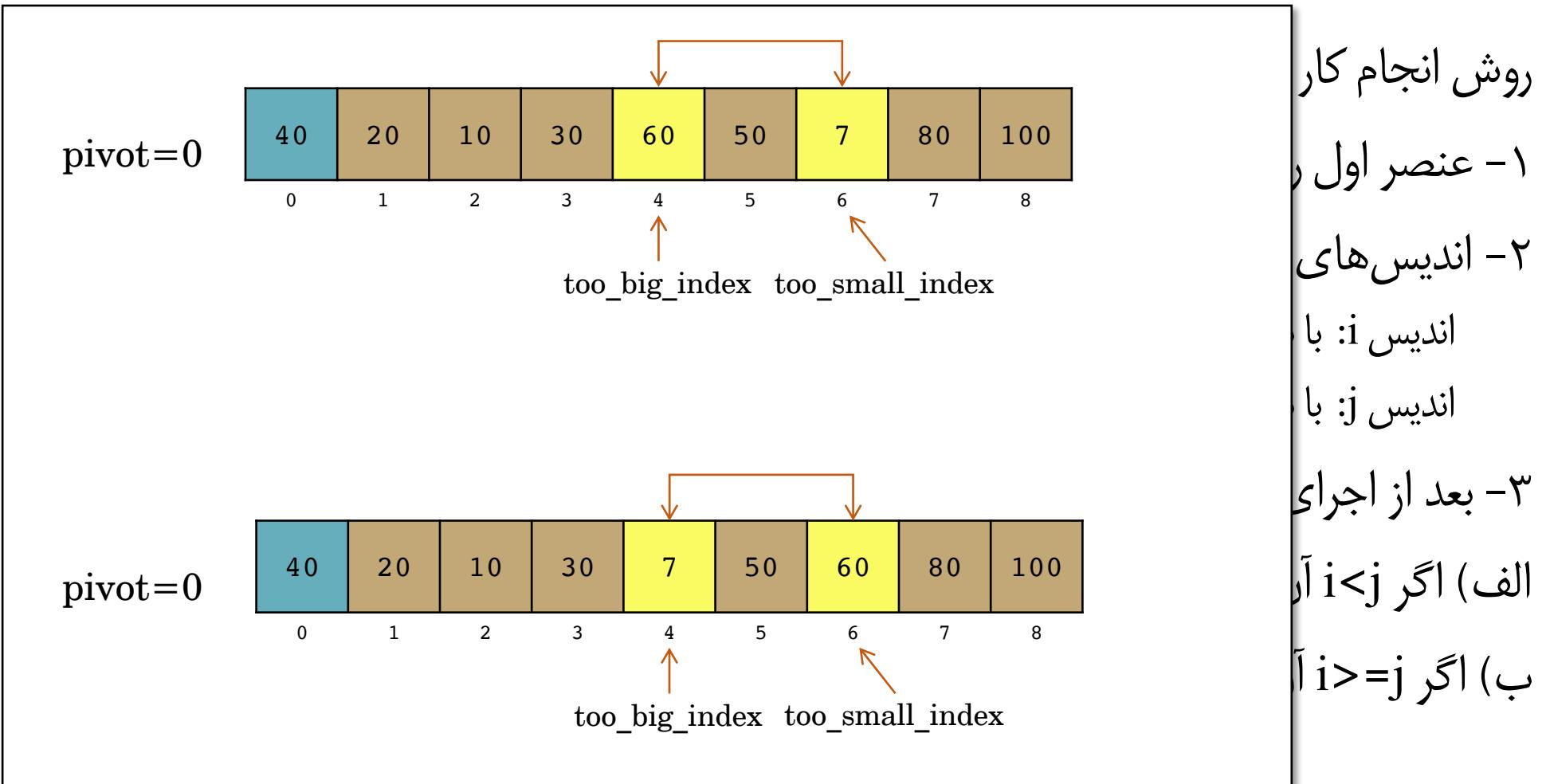
Quick Sort

مرتب‌سازی سریع



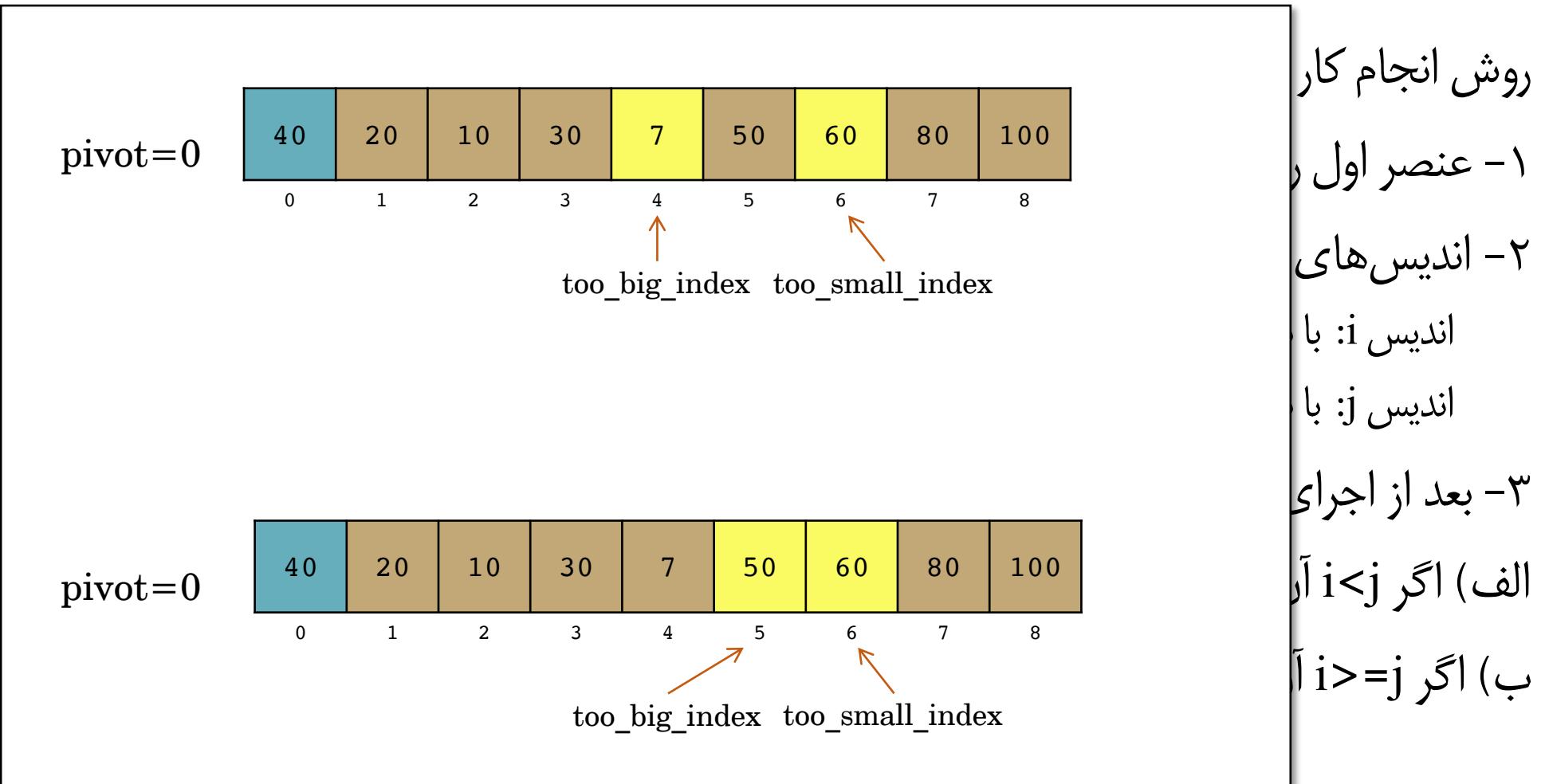
مرتب‌سازی سریع

Quick Sort



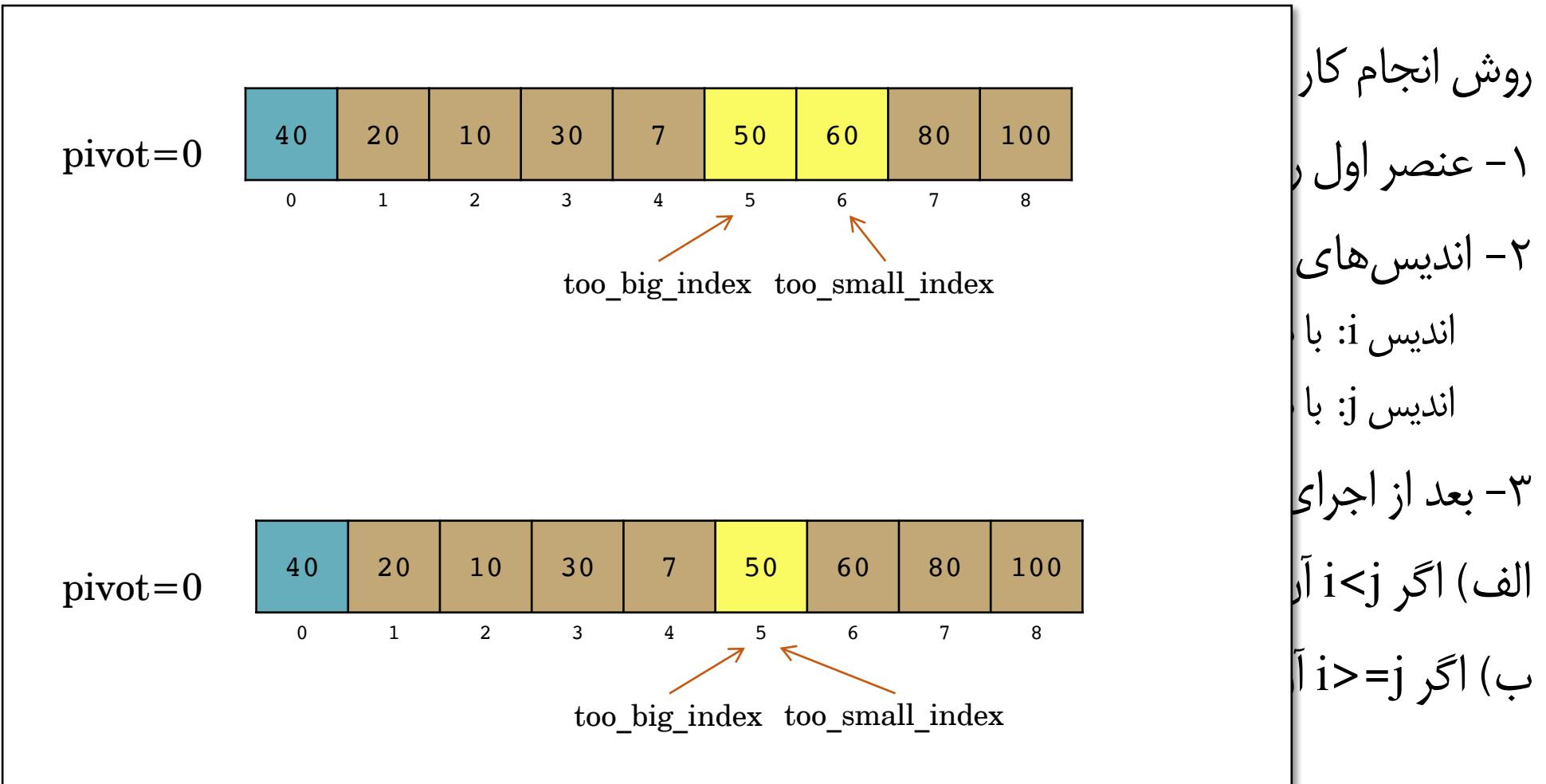
مرتب‌سازی سریع

Quick Sort



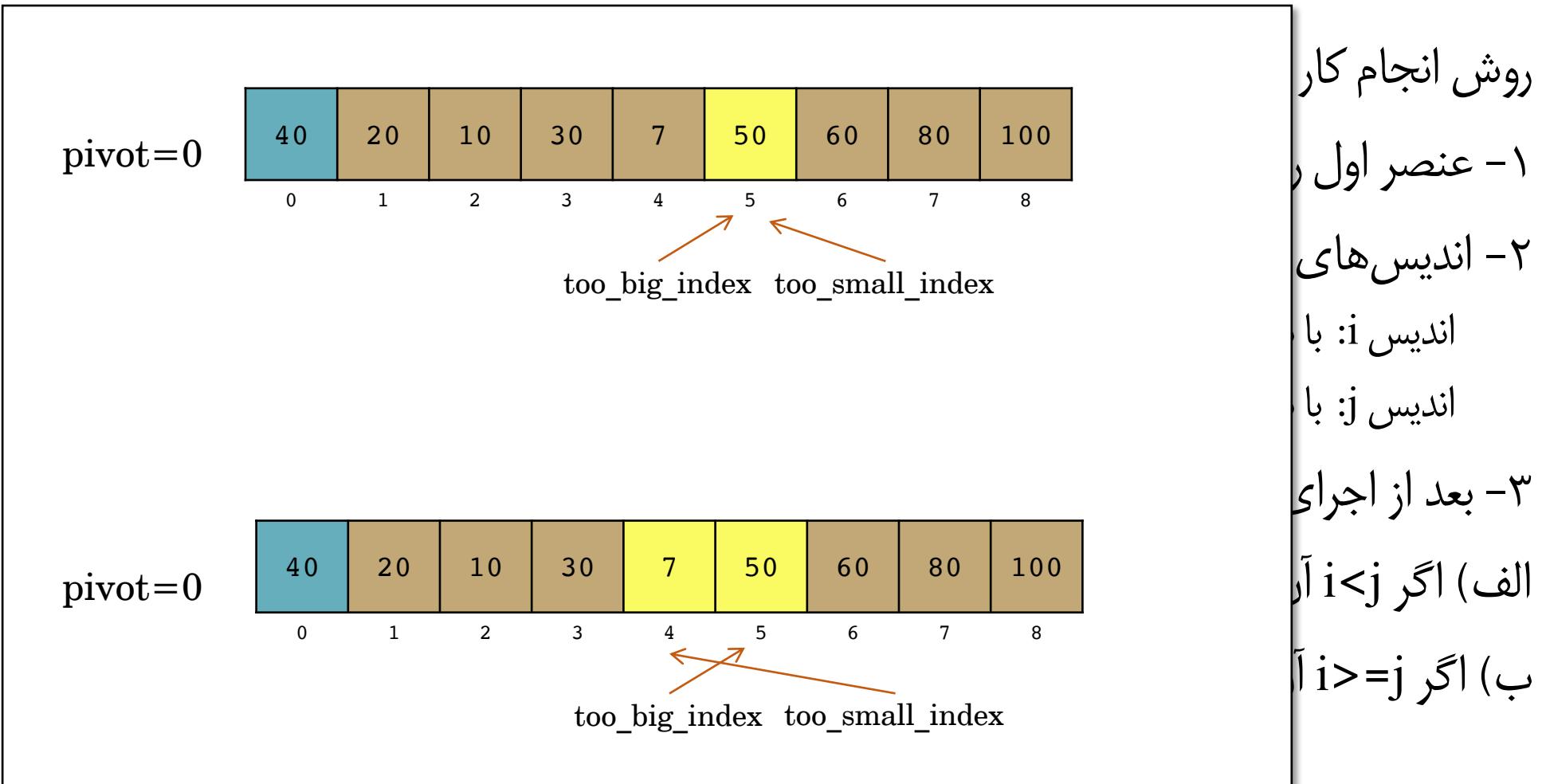
مرتب‌سازی سریع

Quick Sort



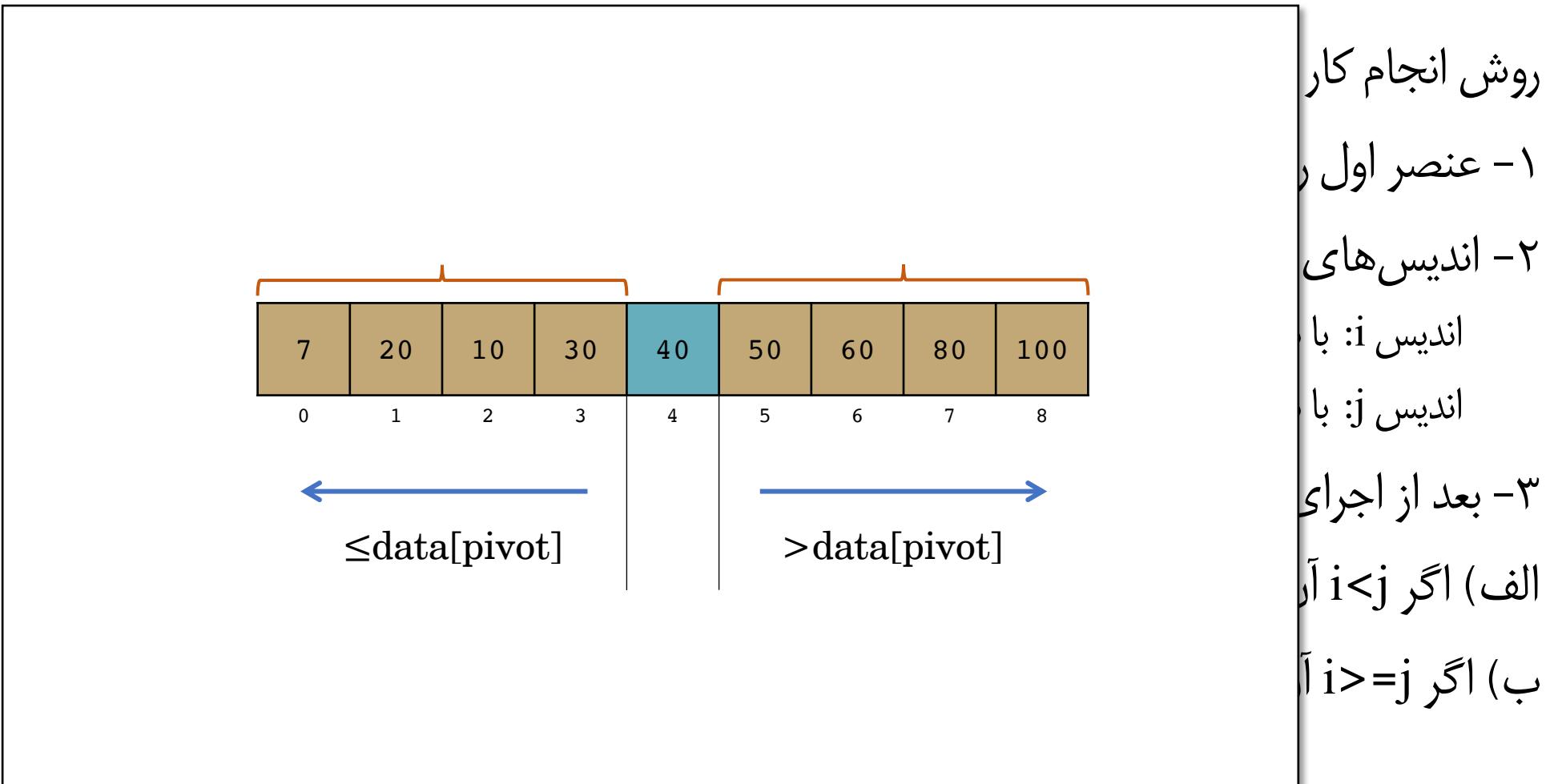
مرتب‌سازی سریع

Quick Sort



مرتب‌سازی سریع

Quick Sort



مرتب‌سازی سریع

Quick Sort

بهترین حالت

در صورتی که عنصر pivot میانه باشد، آن‌گاه آرایه به دو قسمت مساوی تقسیم شده و حداقل عمق بازگشت $\log_2 n$ خواهد بود در این وضعیت به فضای پشته $O(\log_2 n)$ نیاز خواهد بود.

بدترین حالت

در صورتی که عنصر pivot داده ماکزیمم یا مینیمم باشد آن‌گاه آرایه به دو قسمت $(0, n-1)$ برای چپ و راست تقسیم می‌شود؛ که در این حالت عمق بازگشت n خواهد شد و فضای پشته $O(n)$ خواهد بود.

مرتب‌سازی سریع

Quick Sort

بدترین حالت

در صورتی که عنصر $pivot$ داده ماکزیمم یا مینیمم باشد آن‌گاه آرایه به دو قسمت $(0, n-1)$ برای چپ و راست تقسیم می‌شود؛ که در این حالت عمق بازگشت n خواهد شد و فضای پشته $O(n)$ خواهد بود.

بدترین وضعیت در مرتب‌سازی سریع

بدترین وضعیت برای این الگوریتم زمانی است که داده به صورت مرتب یا مرتب معکوس وارد شوند چون هر بار که عنصر اول، محور انتخاب می‌شود؛ بقیه داده‌ها یا سمت چپ و یا سمت راست آن قرار می‌گیرد.

در این وضعیت داریم:

$$O(n^2) \quad \xleftarrow{\text{مرتبه زمانی:}} \quad \frac{n(n-1)}{2} \quad \text{تعداد مقایسه‌ها:}$$

مثال

فرض کنید بخواهیم روش «مرتب‌سازی سریع» را برای آرایه زیر به کار ببریم. در اولین مرحله از کار شکل آرایه مطابق کدام گزینه خواهد بود؟

25, 41, 32, 9, 16, 21

41, 23, 25, 9, 21, 16 (۱)

9, 16, 21, 25, 32, 41 (۲)

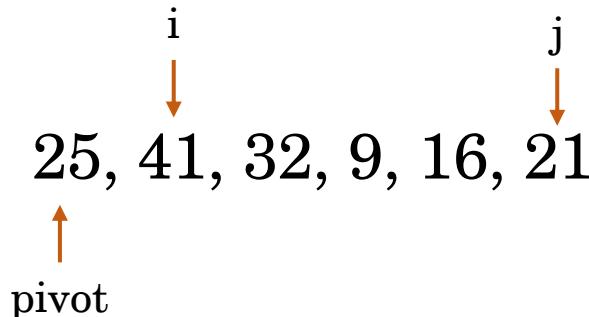
9, 21, 16, 25, 32, 41 (۳)

21, 16, 9, 25, 32, 41 (۴)

مثال

فرض کنید بخواهیم روش «مرتب‌سازی سریع» را برای آرایه زیر به کار ببریم. در اولین مرحله از کار شکل آرایه مطابق کدام گزینه‌^۱ پس از $A[i]$ با $A[j]$ جایه‌جا می‌شود.

25, 41,



, 16 (۱)

2, 41 (۲)

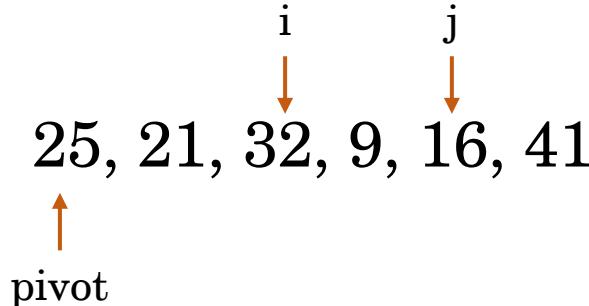
2, 41 (۳)

2, 41 (۴)

مثال

فرض کنید بخواهیم روش «مرتب‌سازی سریع» را برای آرایه زیر به کار ببریم. در اولین مرحله از کار شکل آرایه مطابق کدام گزینه‌^۱ پس از $A[i]$ با $A[j]$ جایه‌جا می‌شود.

25, 41,



, 16 (۱)

2, 41 (۲)

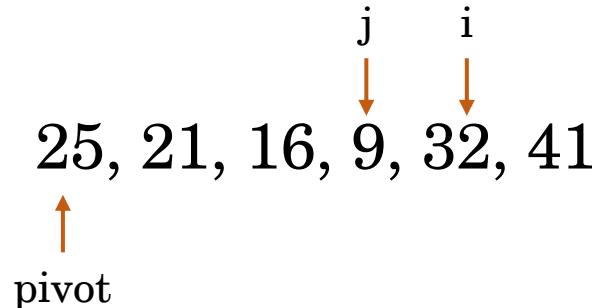
2, 41 (۳)

2, 41 (۴)

مثال

فرض کنید بخواهیم روش «مرتب‌سازی سریع» را برای آرایه زیر به کار ببریم. در اولین مرحله از کار شکل آرایه مطابق کدام گزینه‌ی $j < i$ پس $A[j \dots i]$ با pivot جایه‌جا می‌شود و مرحله اول به اتمام می‌رسد.

25, 41,



, 16 (۱)

2, 41 (۲)

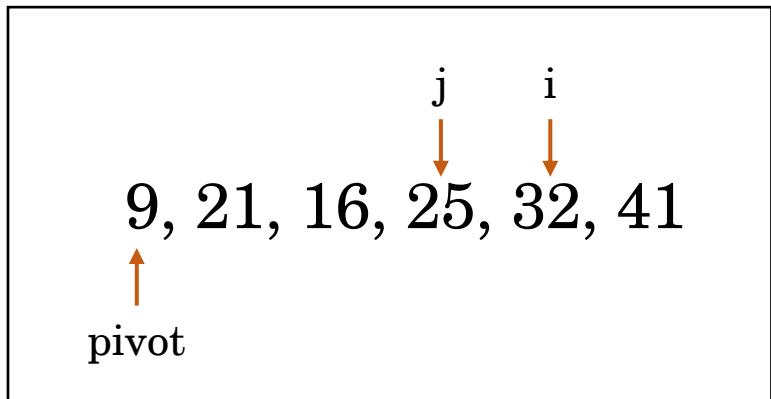
2, 41 (۳)

2, 41 (۴)

مثال

فرض کنید بخواهیم روش «مرتب‌سازی سریع» را برای آرایه زیر به کار ببریم. در اولین مرحله از کار شکل آرایه مطابق کدام گزینه خواهد بود؟

25, 41, 32, 9, 16, 21



41, 23, 25, 9, 21, 16 (۱)

9, 16, 21, 25, 32, 41 (۲)

9, 21, 16, 25, 32, 41 (۳)

21, 16, 9, 25, 32, 41 (۴)

BST Sort

مرتب‌سازی درختی

یک روش برای مرتب‌سازی n کلید:

- با استفاده از عناصر داده شده یک درخت جستجوی دودویی ایجاد کن؛
- درخت حاصل را به روش میان‌ترتیب پیمایش کن.

BST Sort

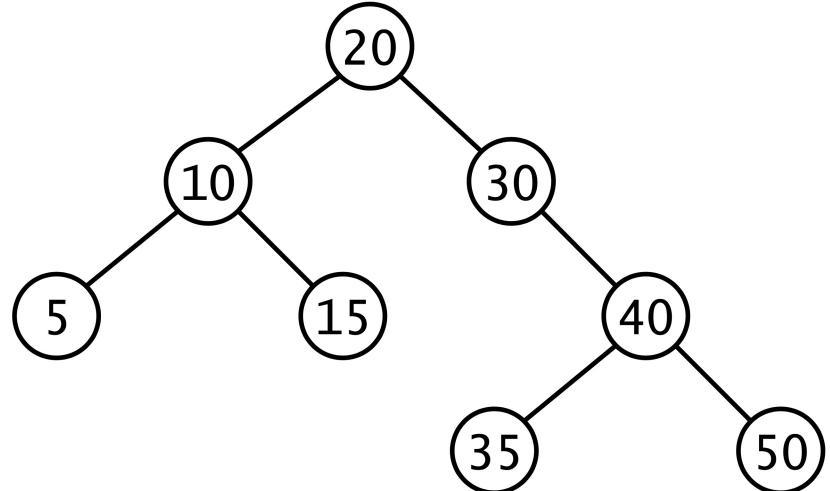
مرتب‌سازی درختی

15, 50, 35, 5, 40, 10, 30, 20



یک روش برای مرتب‌سازی n کلید:

- با استفاده از عناصر داده شده یک درخت جستجوی دودویی ایجاد کن؛
- درخت حاصل را به روش میان‌ترتیب پیمایش کن.



→ 5, 10, 15, 20, 30, 35, 40, 50

مرتب‌سازی هرم

Heap Sort

در مرتب‌سازی هرم ابتدا با استفاده از روال Build-Max-Heap آرایه ورودی $A[1...n]$ به Max Heap تبدیل می‌شود. از آنجا که بزرگ‌ترین عنصر آرایه در ریشه قرار دارد، می‌تواند با یک جابه‌جایی با $A[n]$ در مکان صحیح (نهایی) خود قرار گیرد. سپس طول هرم یکی کم شده و برای حفظ ویژگی Max Heap الگوریتم MAX-HEAPIFY برای ریشه جدید فراخوانی می‌شود. روال این روند را تا هرم با اندازه ۲ تکرار می‌کند.

از آنجا که فراخوانی MAX-HEAPIFY زمان $O(n)$ و هر یک از $n-1$ فراخوانی MAX-HEAPIFY زمان $O(\log n)$ صرف می‌کنند؛ زمان مرتب‌سازی هرم برابر است با: $O(n) + (n-1)O(\log n) = O(n \log n)$

مرتب‌سازی هرم

Heap Sort

در مرتب‌سازی هرم ابتدا با استفاده از روال Build-Max-Heap آرایه ورودی $A[1...n]$ به Max Heap تبدیل می‌شود. از آنجا که بزرگ‌ترین عنصر آرایه در ریشه قرار دارد، می‌تواند با یک جابه‌جایی با $A[n]$ در مکان صحیح (نهایی) خود قرار گیرد.

بیشه جدید فراخوانی

```
def Heap_Sort(A):
    Build_Max_Heap(A)
    n=len(A)
    for i in [n,n-1,...,2]:
        swap(A[1],A[i])
        len(A)=len(A)-1
        MAX_HEAPIFY(A,1)
    return A
```

$O(\log n)$ زمان

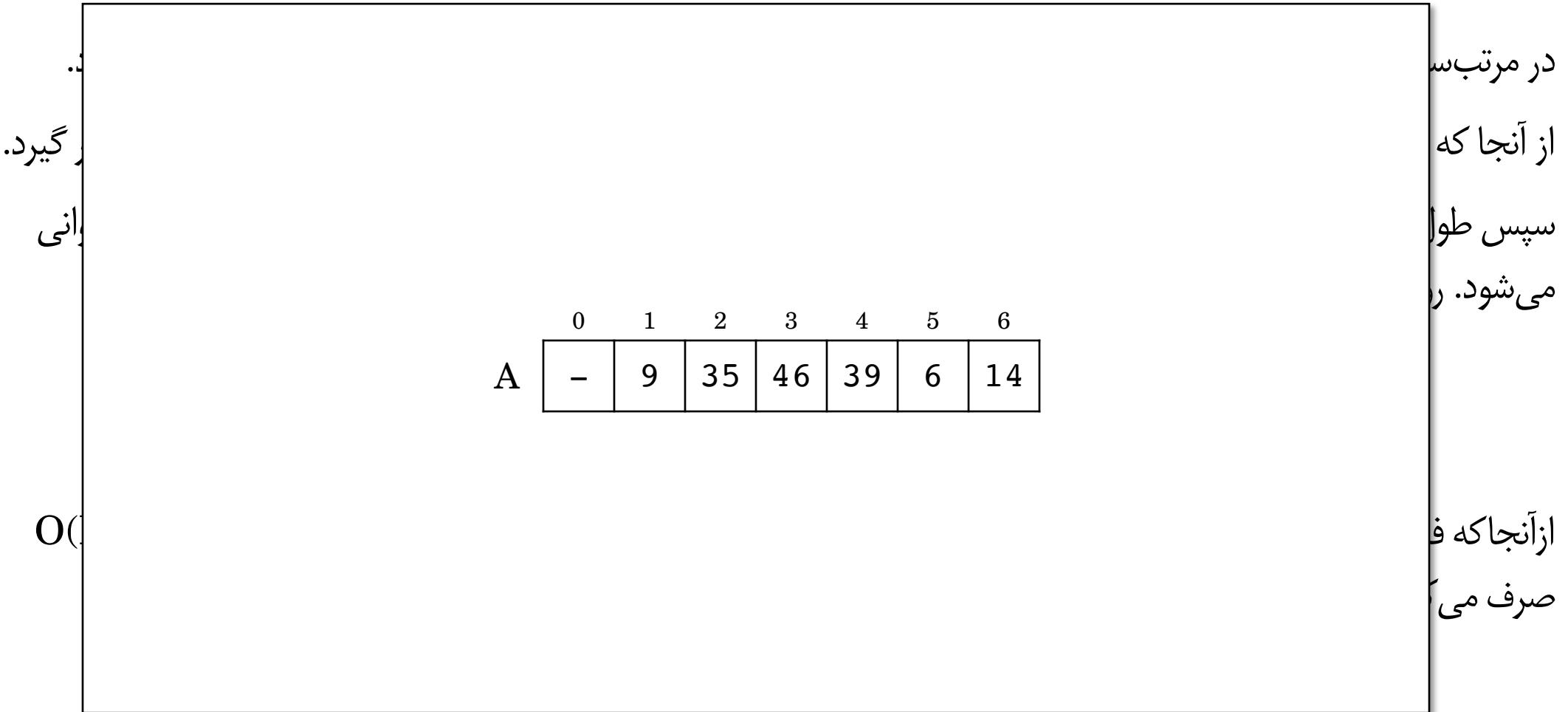
سپس طول هرم یکی کم می‌شود. روال این روند را تا

از آنجا که فراخوانی Heap

صرف می‌کنند؛ زمان مرتب‌سازی هرم برابر است با: $O(n) + (n-1)O(\log n) = O(n \log n)$

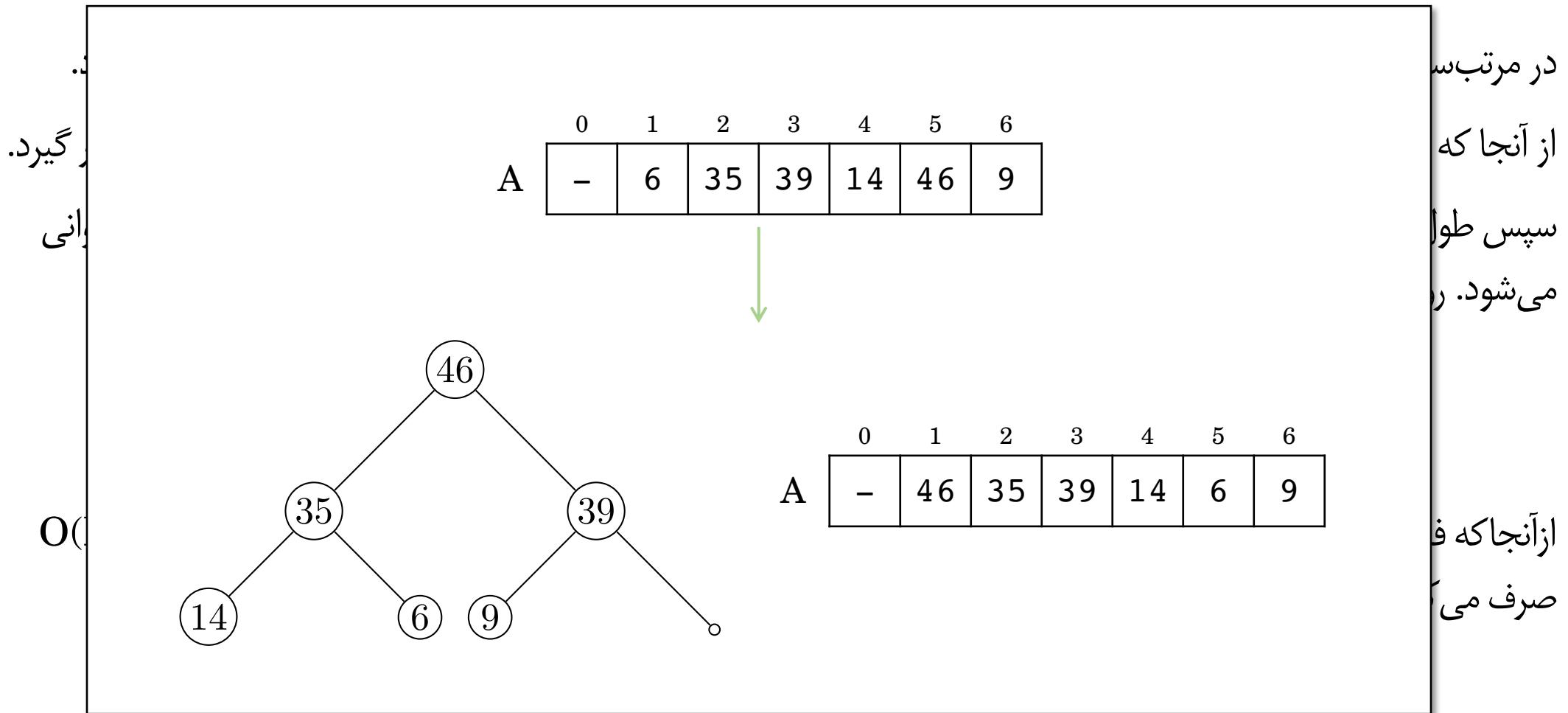
مرتب‌سازی هرم

Heap Sort



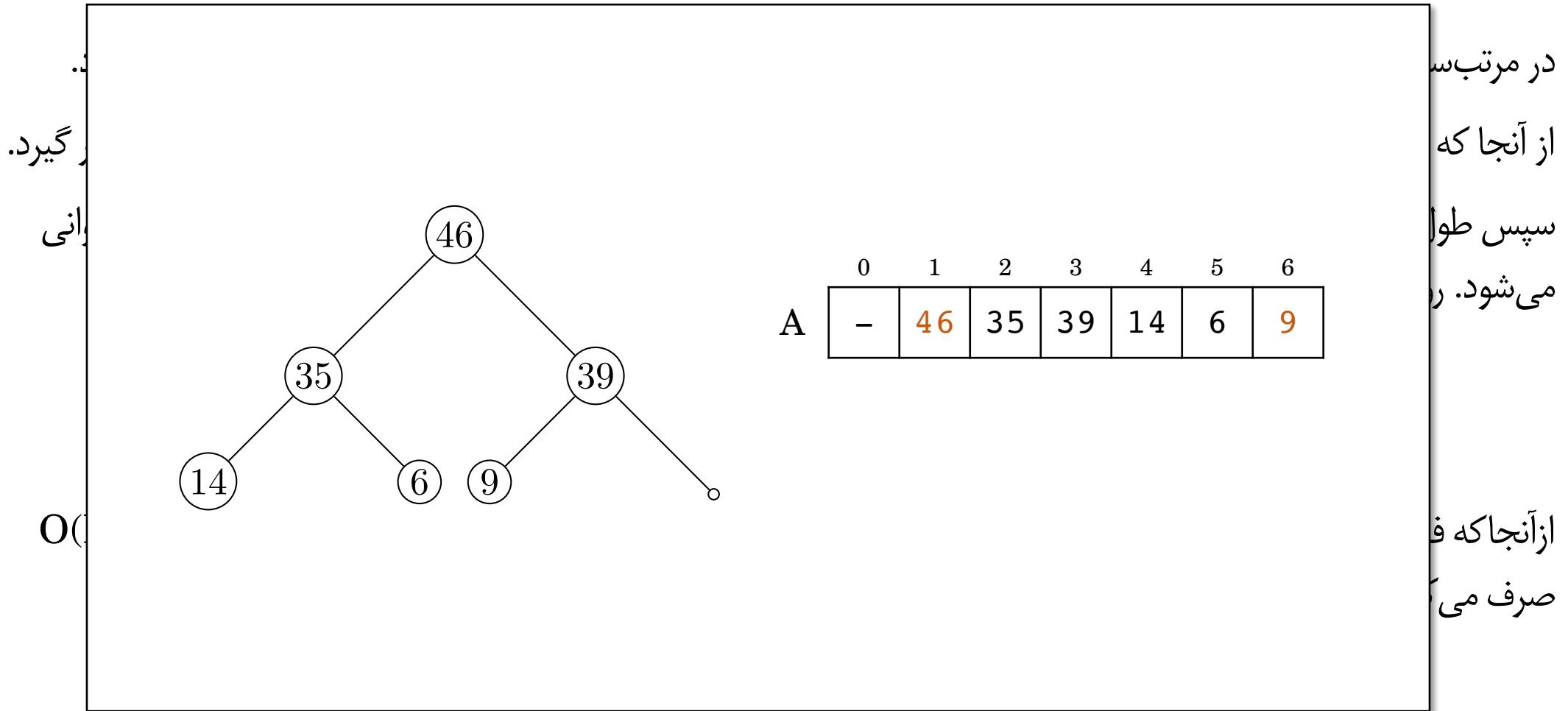
Heap Sort

مرتب‌سازی هرم



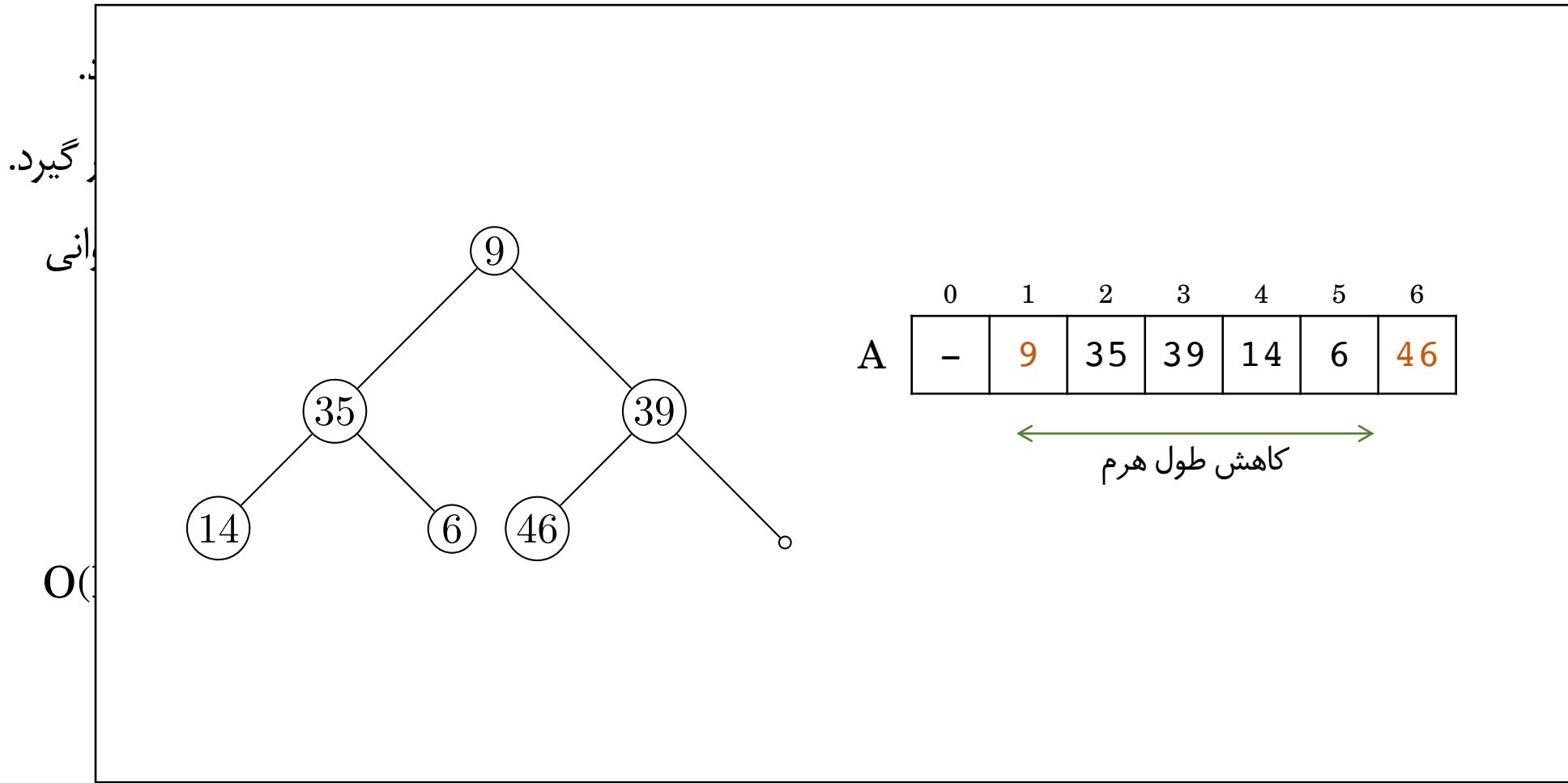
مرتب‌سازی هرم

Heap Sort



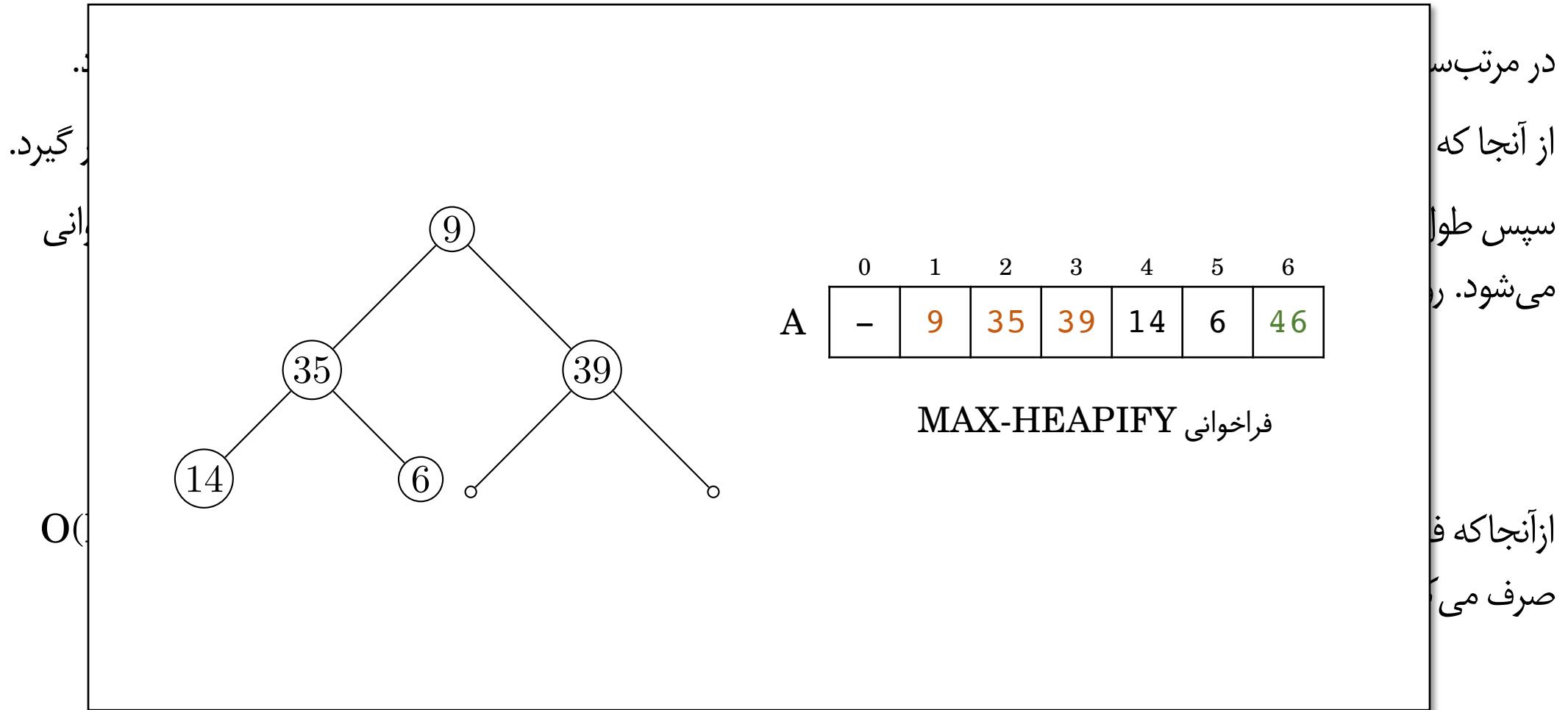
مرتب‌سازی هرم

Heap Sort



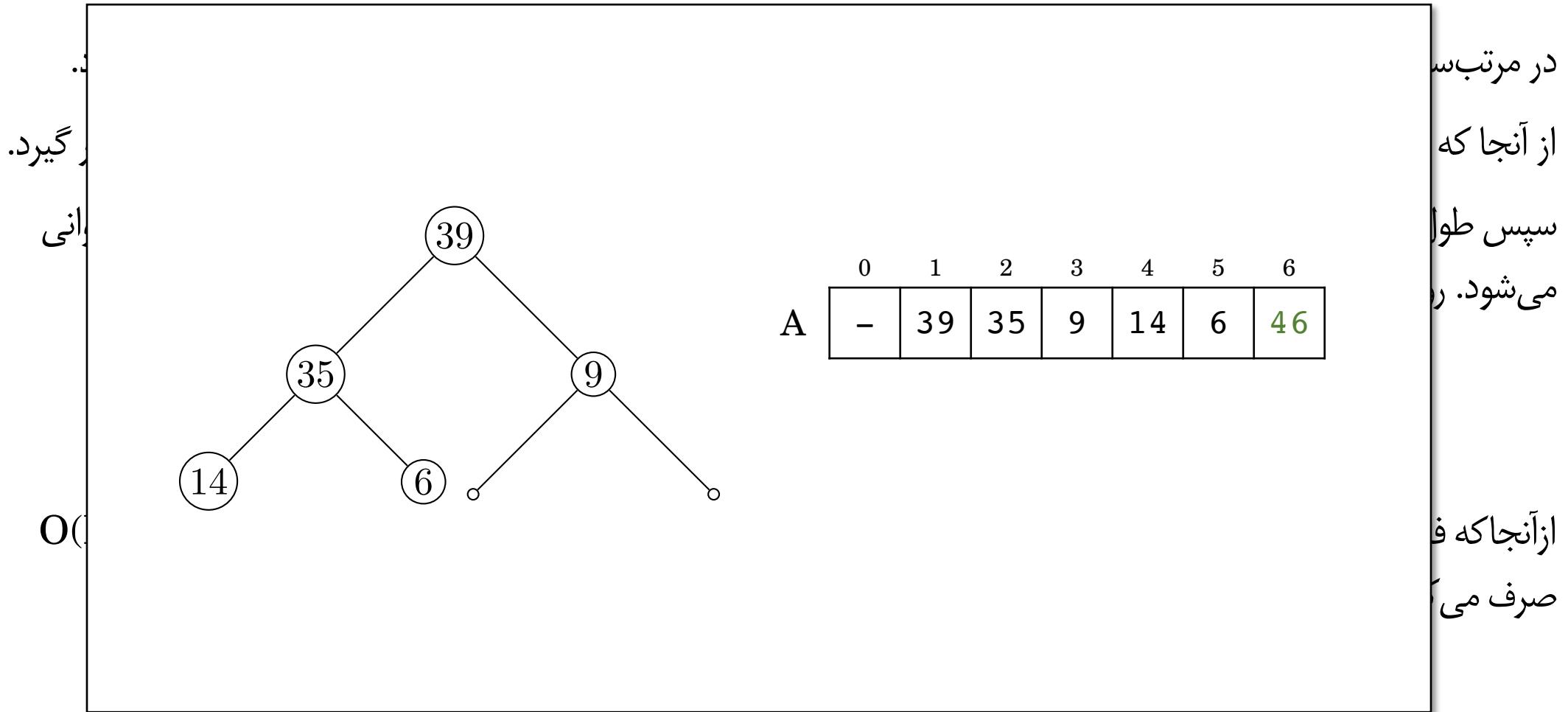
مرتب‌سازی هرم

Heap Sort



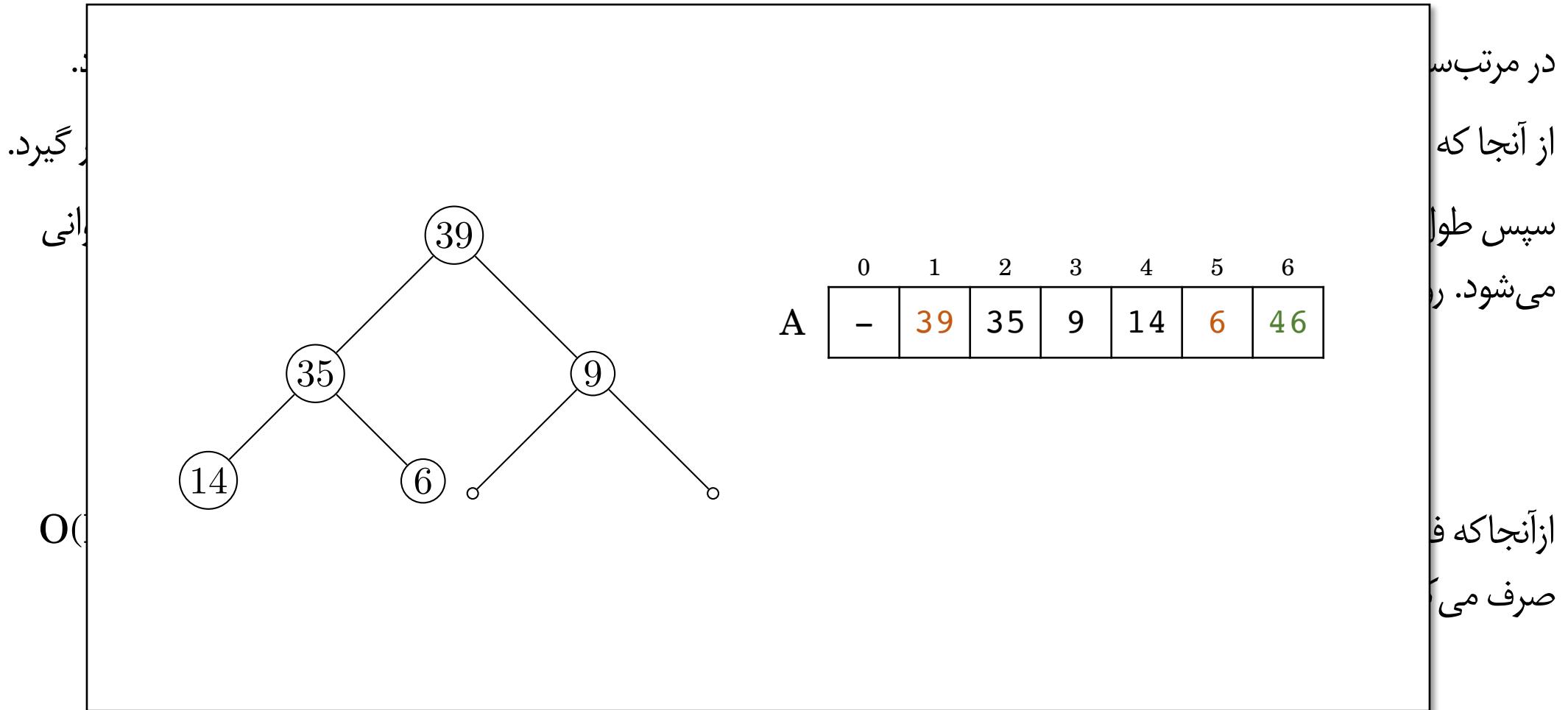
مرتب‌سازی هرم

Heap Sort



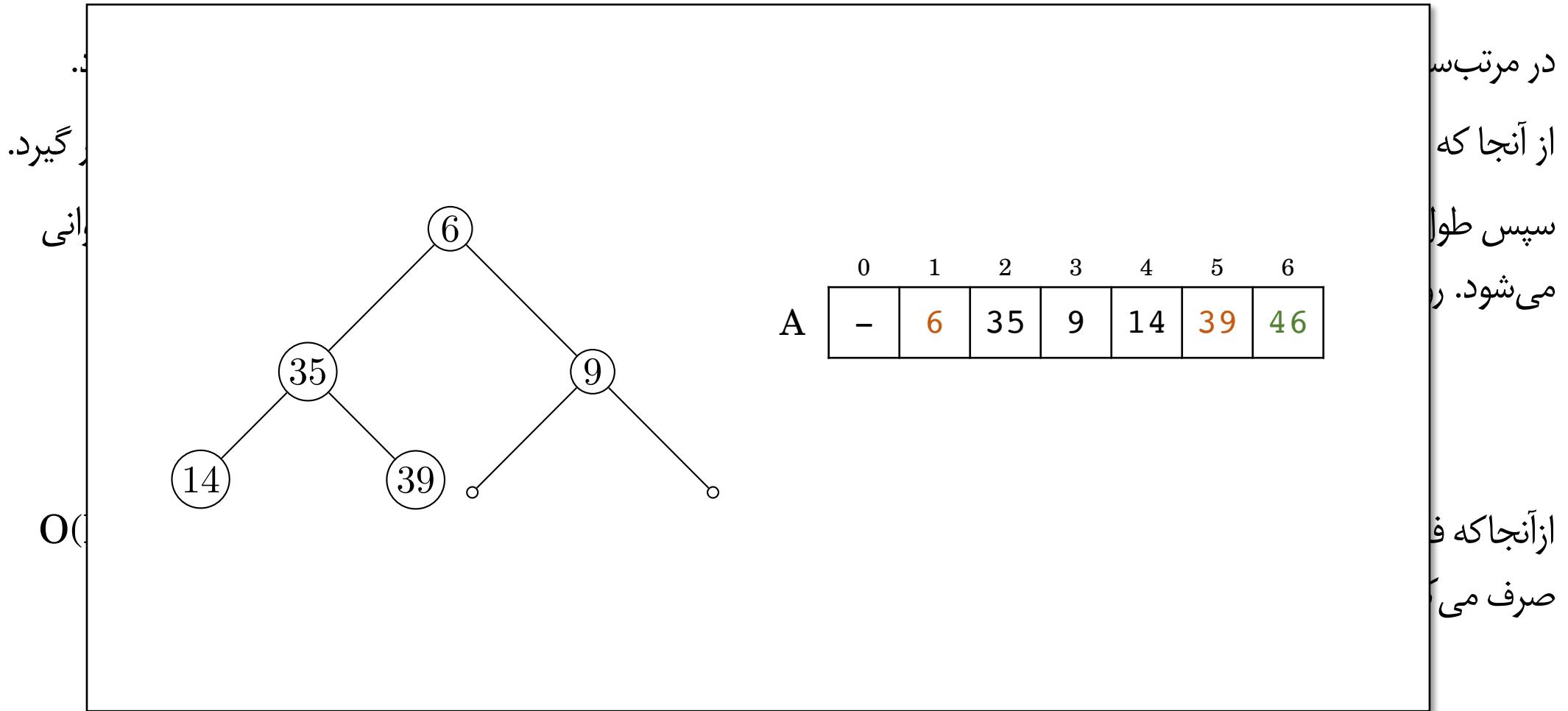
مرتب‌سازی هرم

Heap Sort



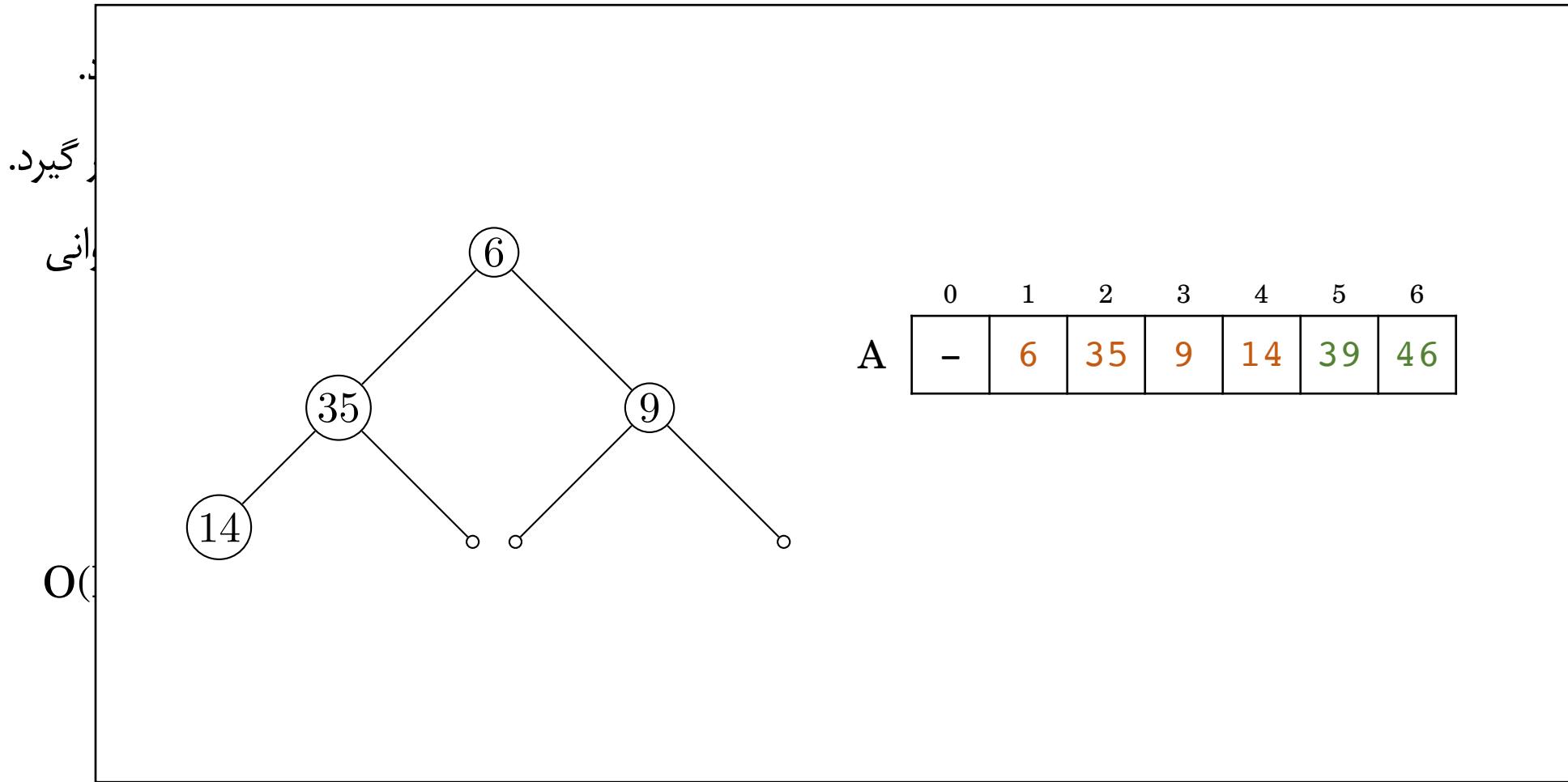
مرتب‌سازی هرم

Heap Sort



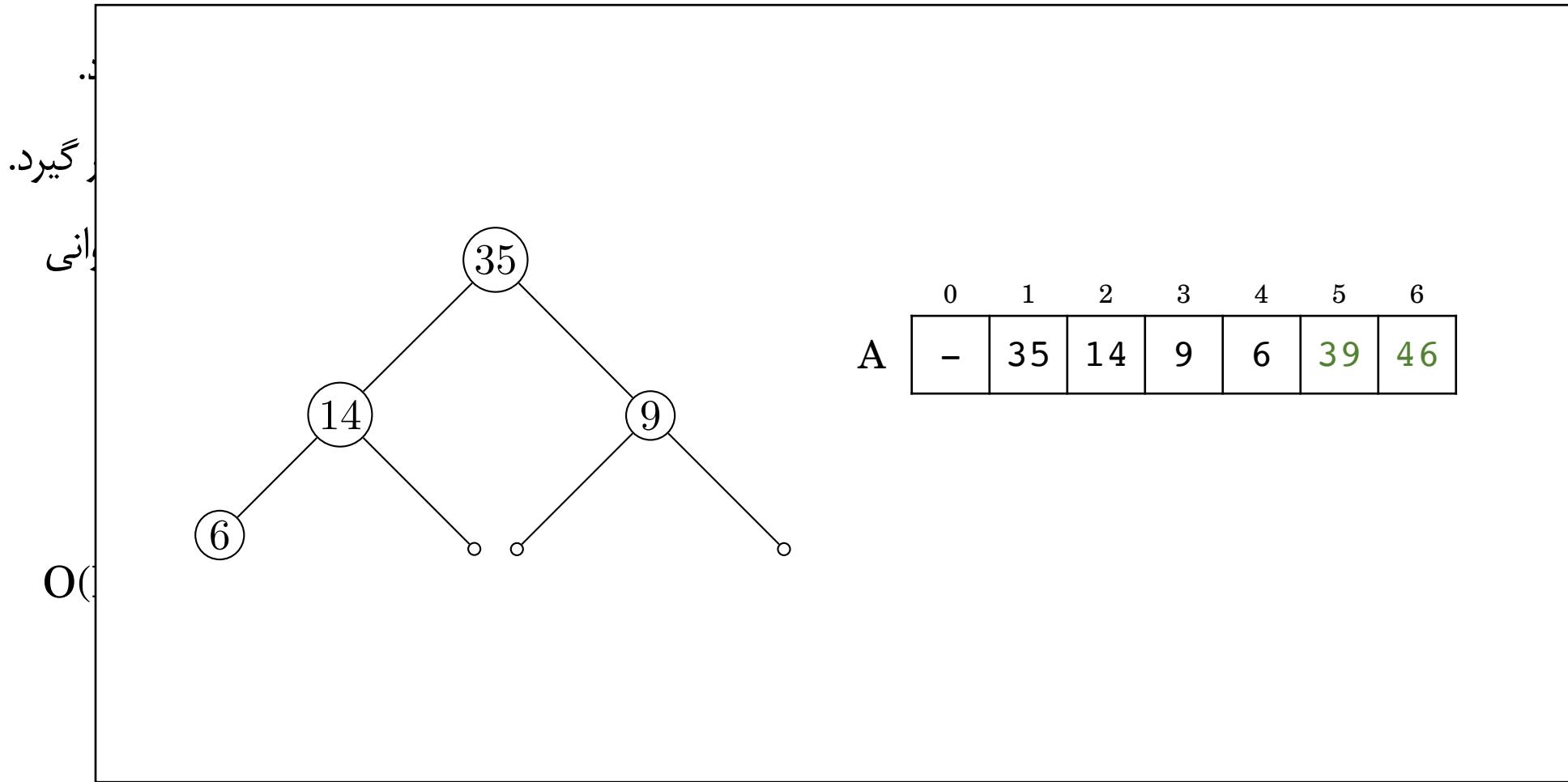
مرتب‌سازی هرم

Heap Sort



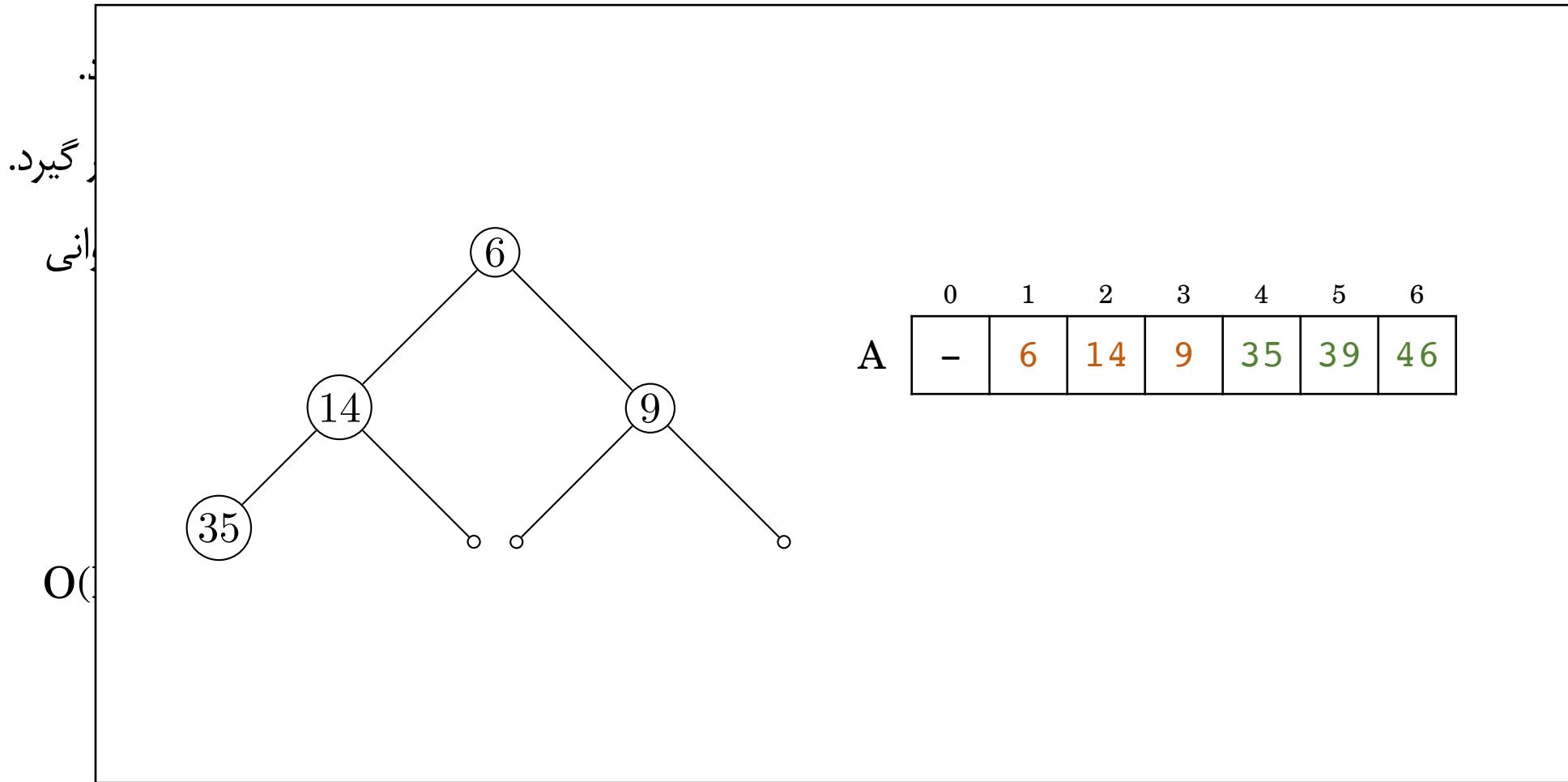
مرتب‌سازی هرم

Heap Sort



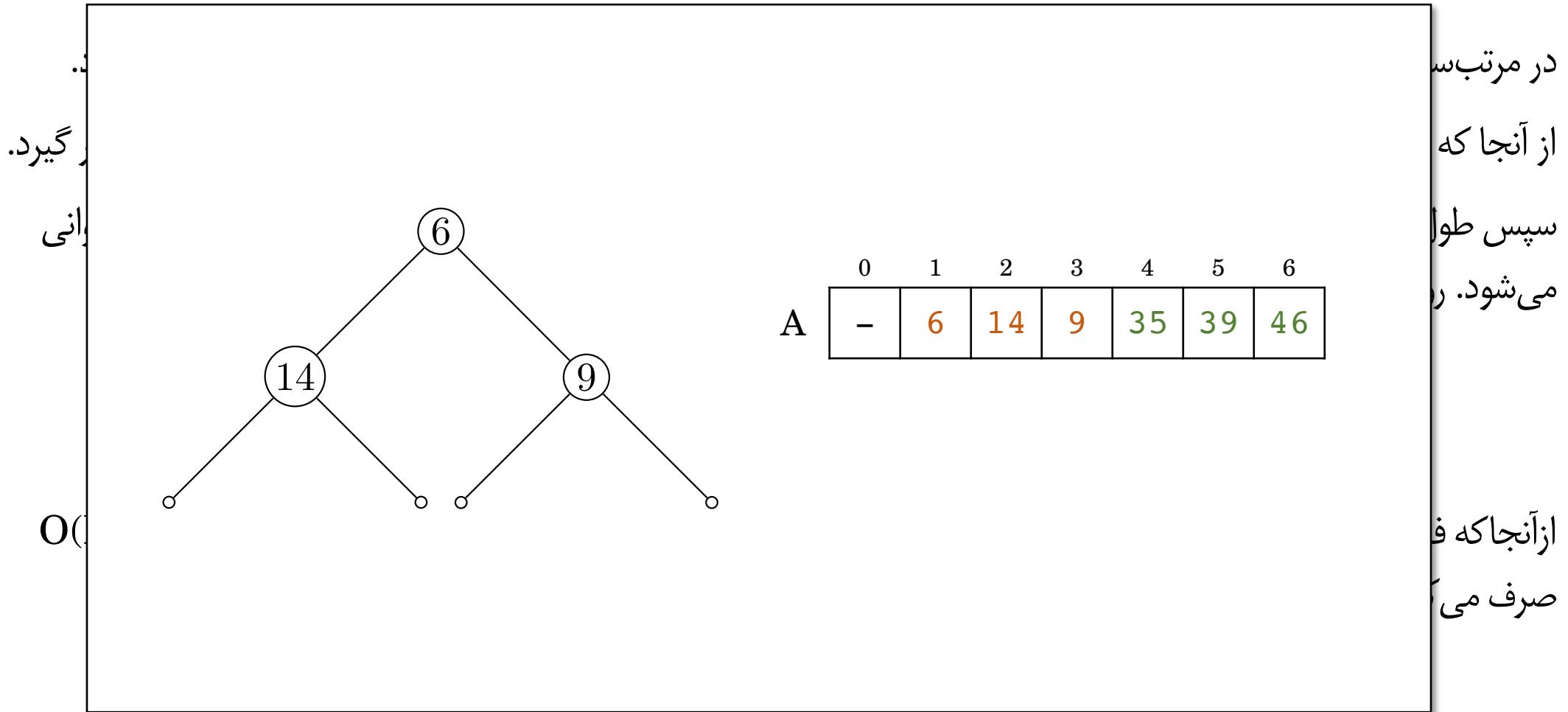
مرتب‌سازی هرم

Heap Sort



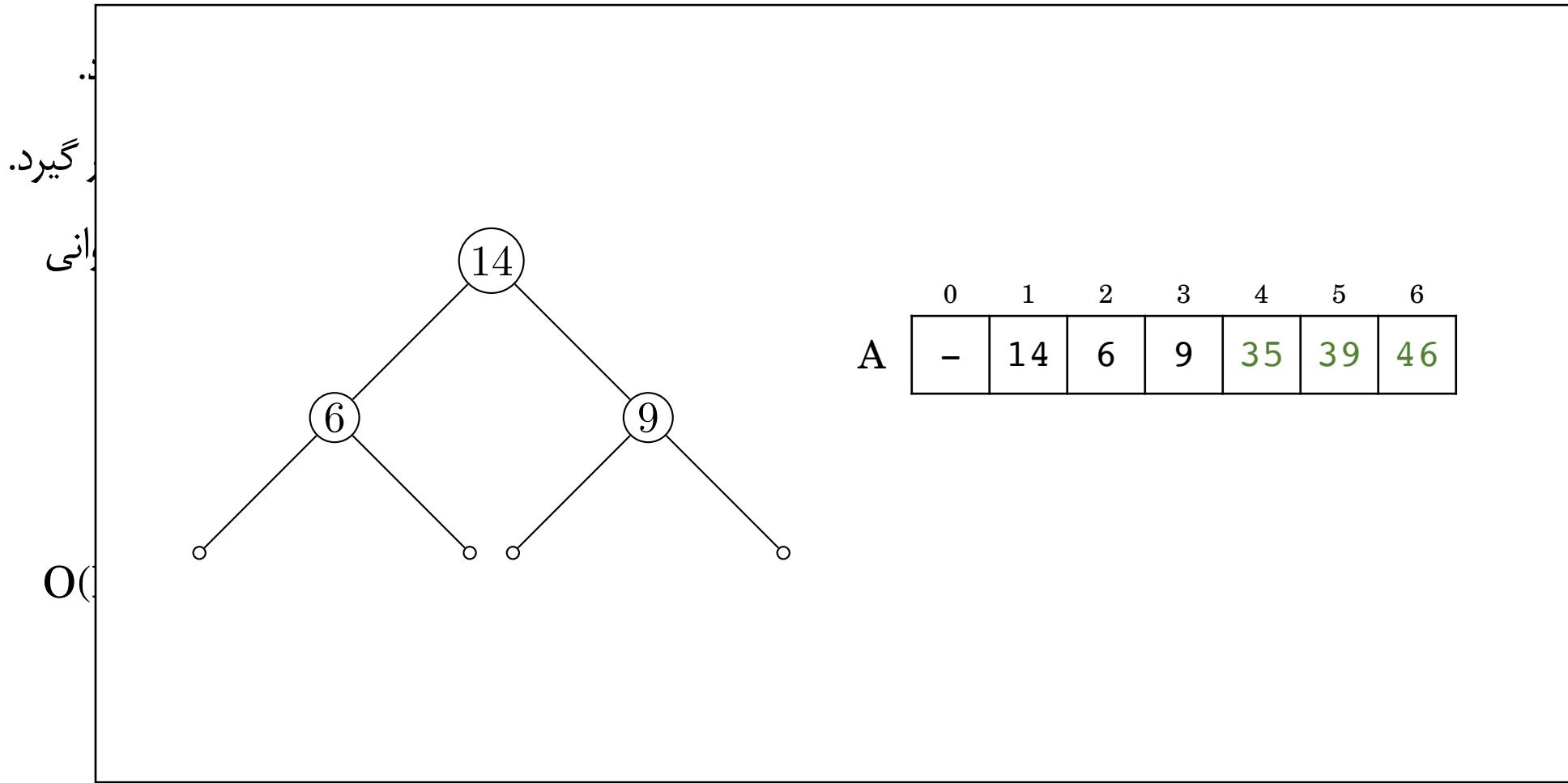
مرتب‌سازی هرم

Heap Sort



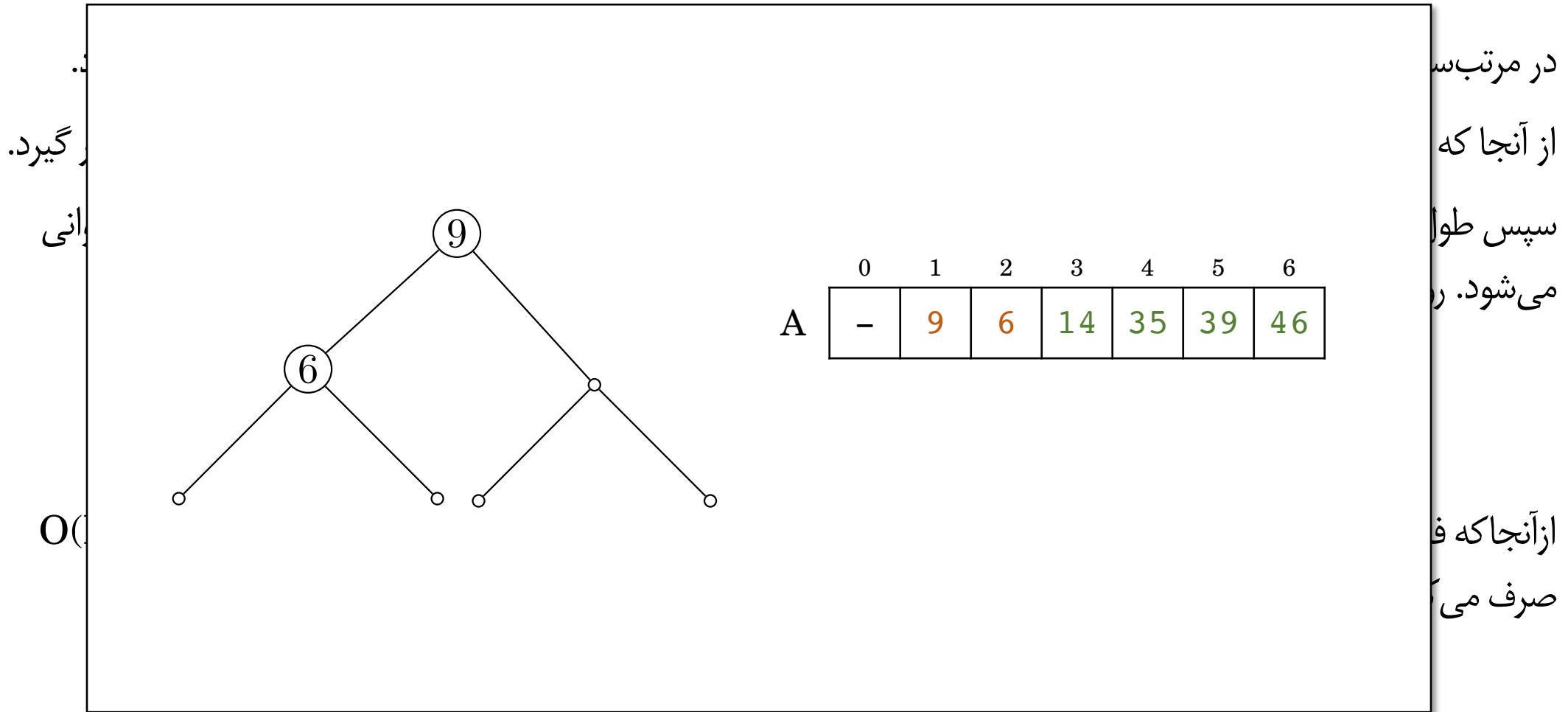
مرتب‌سازی هرم

Heap Sort



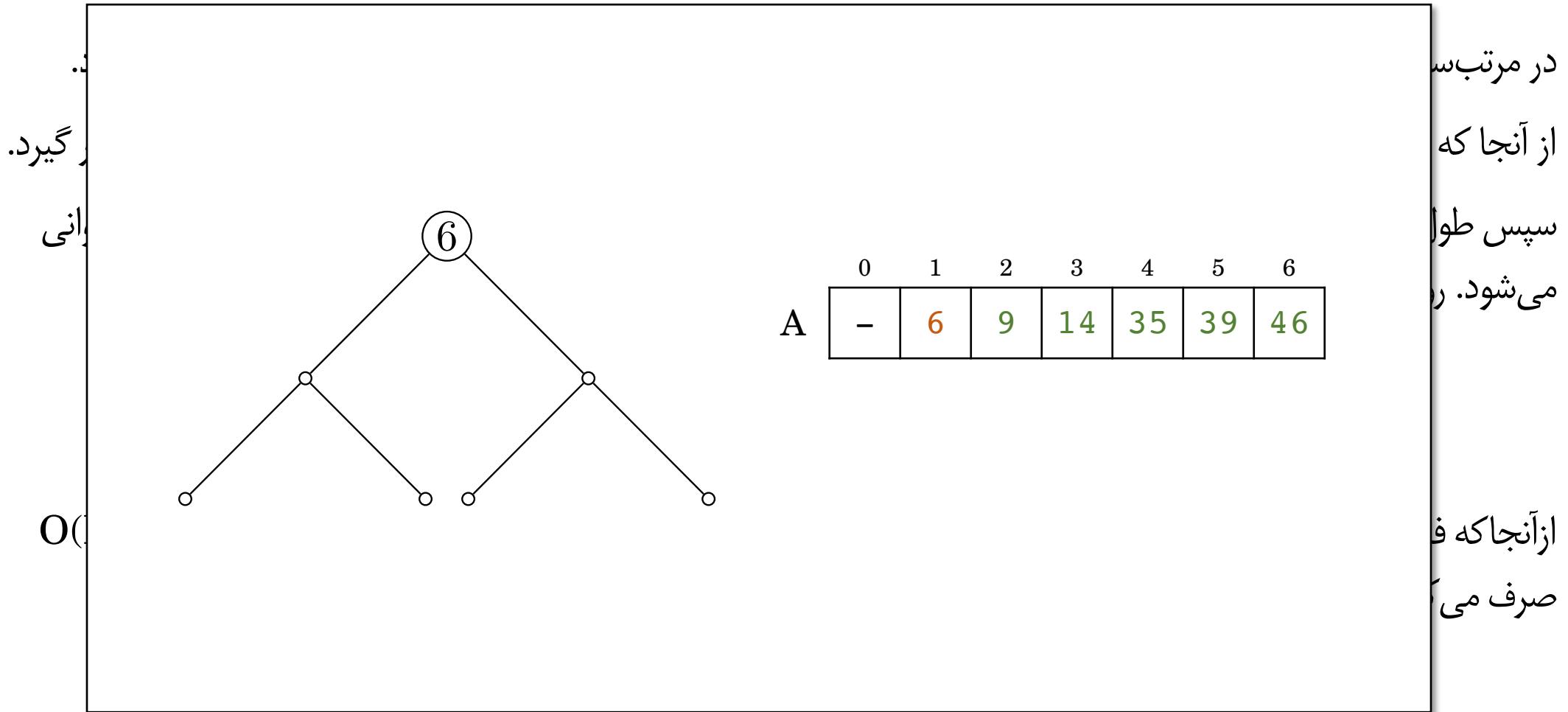
مرتب‌سازی هرم

Heap Sort



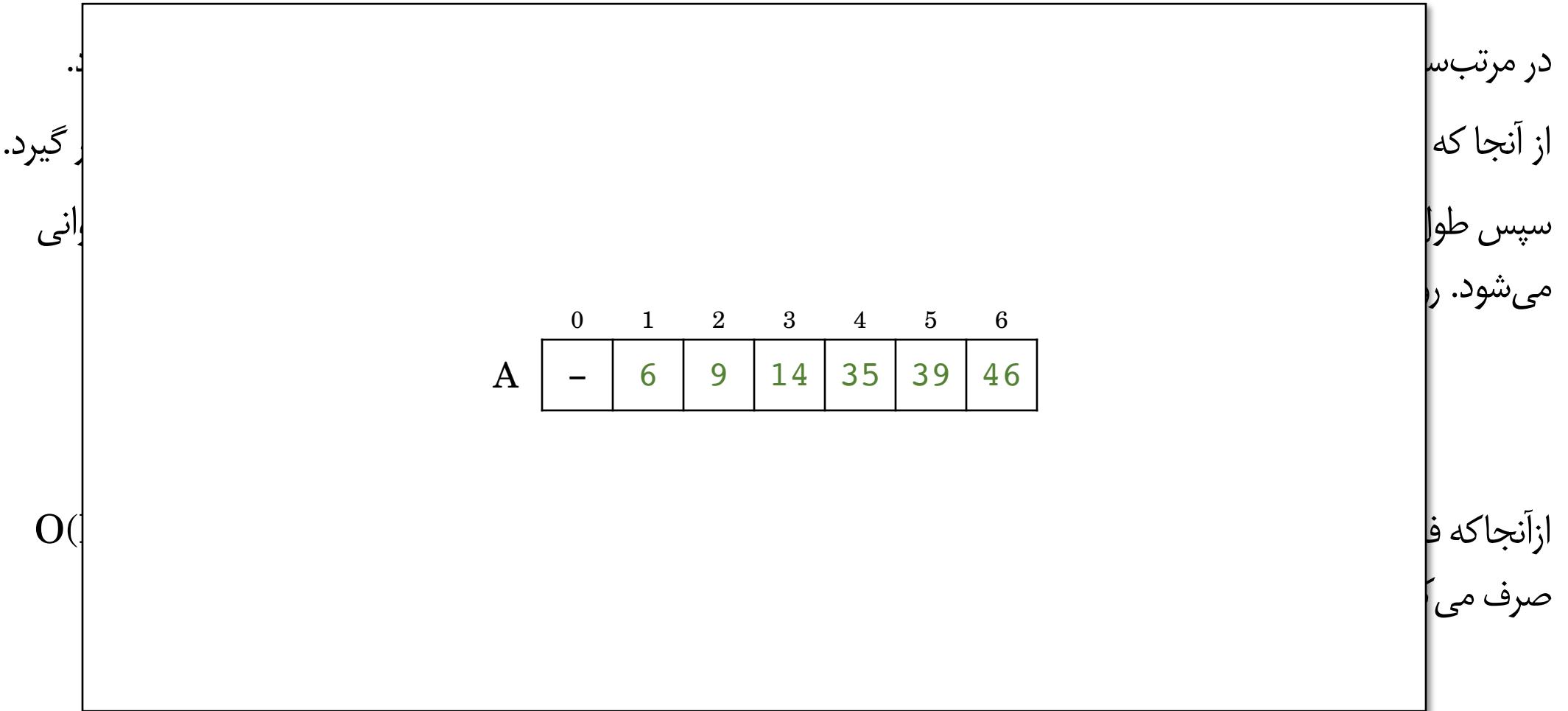
مرتب‌سازی هرم

Heap Sort



مرتب‌سازی هرم

Heap Sort



مرتب‌سازی مبنا

Radix Sort

در صورتی که آرایه n تایی با اعدادی حداقل d رقمی در مبنای r وجود داشته باشد.

مثال: در آرایه $n=8$ تایی زیر اعداد حداقل $d=3$ رقمی و در مبنای $r=10$ می‌باشد:

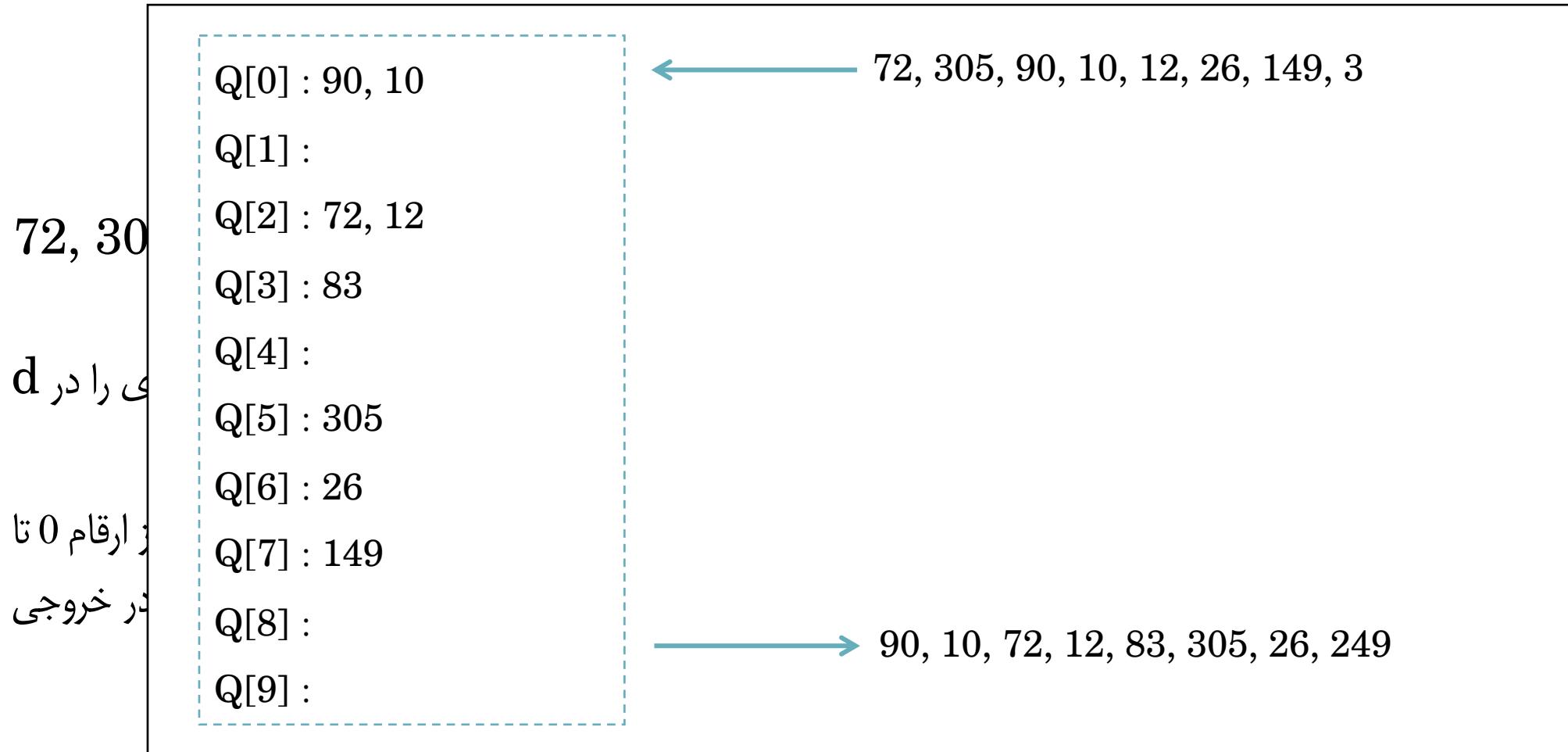
72, 305, 90, 10, 12, 26, 149, 3

الگوریتم مرتب‌سازی مبنا با شروع از کم ارزش‌ترین رقم اعداد(یکان) و به کمک r صفحه ($Q[0\dots 9]$) عمل مرتب‌سازی را در d گذر انجام می‌دهد.

در هر گذر، اعداد آرایه را به ترتیب از چپ به راست خوانده و با توجه به آنکه کم ارزش‌ترین رقم (یکان) در آنها کدام یک از ارقام 0 تا $r-1$ است. در یکی از صفحه‌ای $[0\dots r-1] Q$ قرار می‌دهد. در انتهای صفحه‌ها را به ترتیب از $[0] Q$ تا $[r-1] Q$ در خروجی گذر قرار می‌دهد.

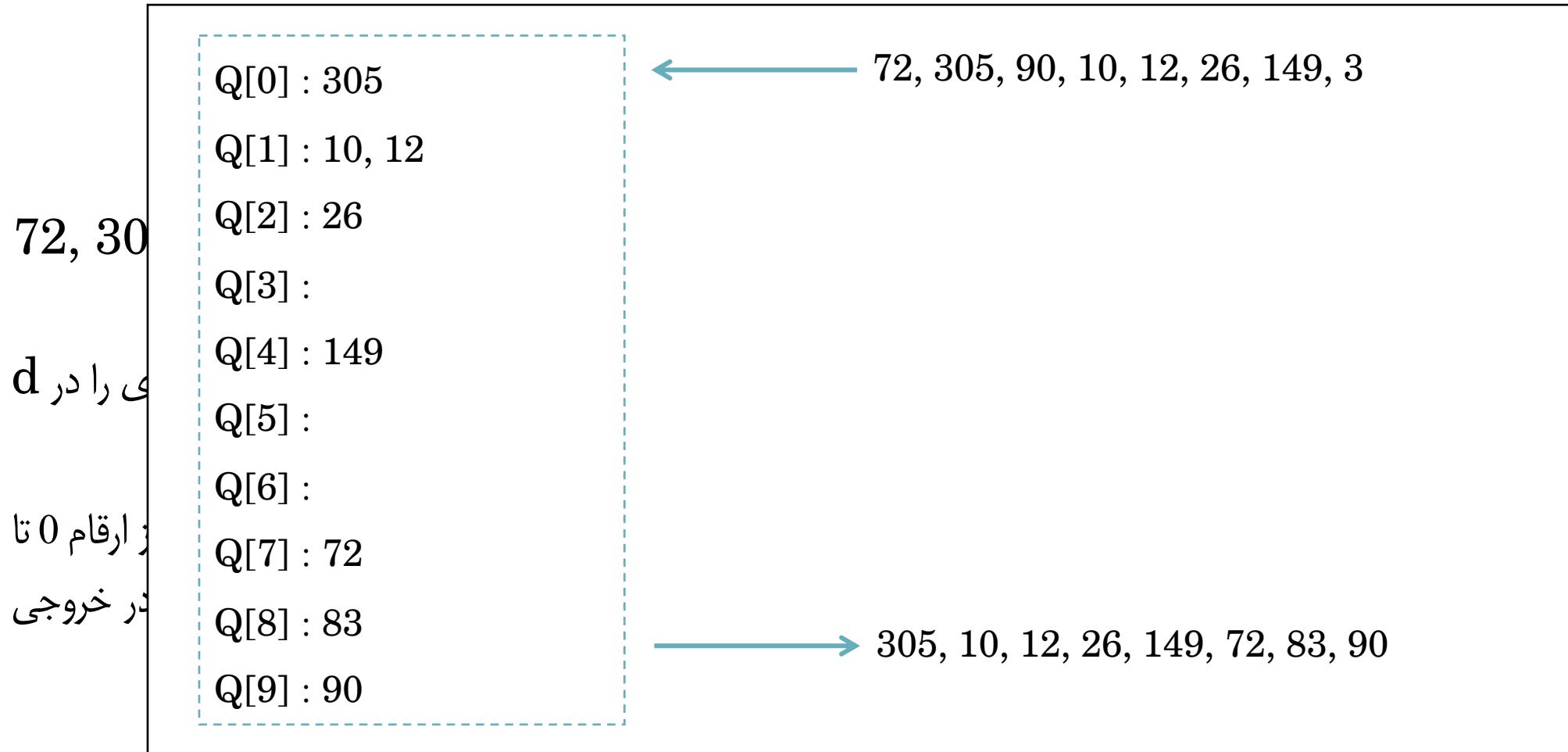
مرتب‌سازی مبنا

Radix Sort



مرتب‌سازی مبنا

Radix Sort



مرتب‌سازی مبنا

Radix Sort

