



هرم

ساختمان داده ها و الگوریتم

مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان



مسئله

ساختمان داده ای ارائه کنید که به نحوی کارا بتواند اعمال زیر را انجام دهد:

- درج عنصر : یک عنصر جدید را در مجموعه عناصرهایش اضافه کند.
- حذف عنصر کمینه : عنصر کمینه موجود در مجموعه عناصر درج شده را پیدا کند و حذف کند.
- پرسش عنصر کمینه : عنصر کمینه موجود در مجموعه عناصر درج شده را اعلام کند.

مسئله

ساختمان داده ای ارائه

کند و حذف کند.
نم کند.

شروع	:	{}	
Add	6	:	{6}
Add	1	:	{6,1}
Add	14	:	{6, 1, 14}
Min Query		:	1
Delete		:	{6, 14}
Min Query		:	6

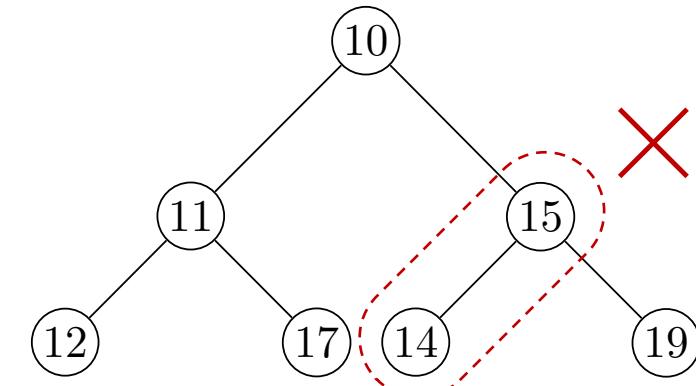
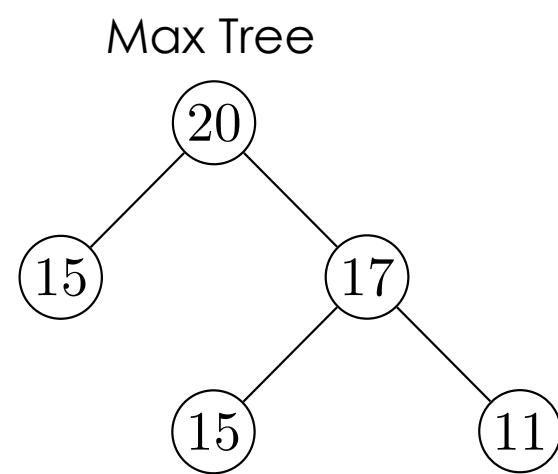
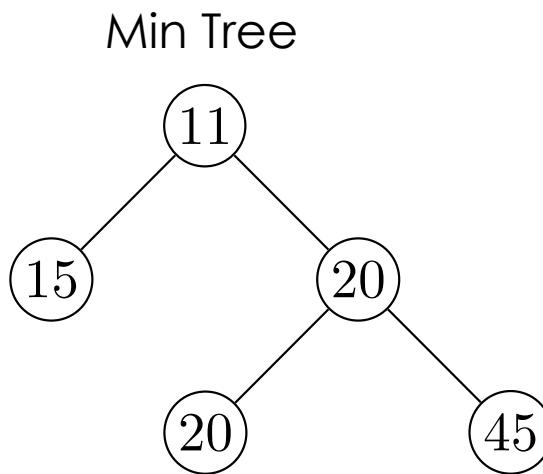
• درج عنصر

• حذف عنصر کمینه

• پرسش عنصر کمینه

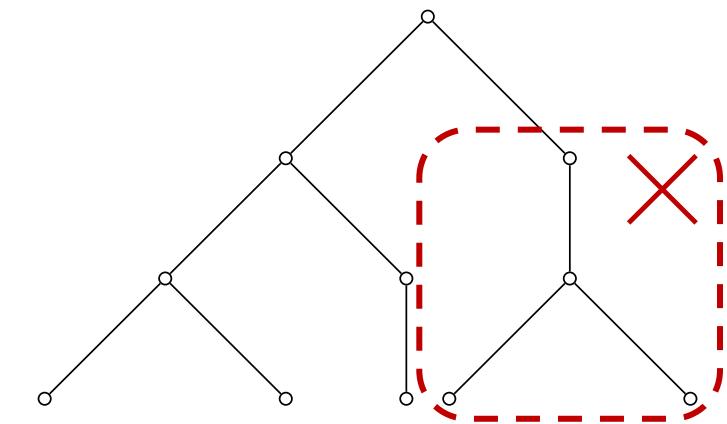
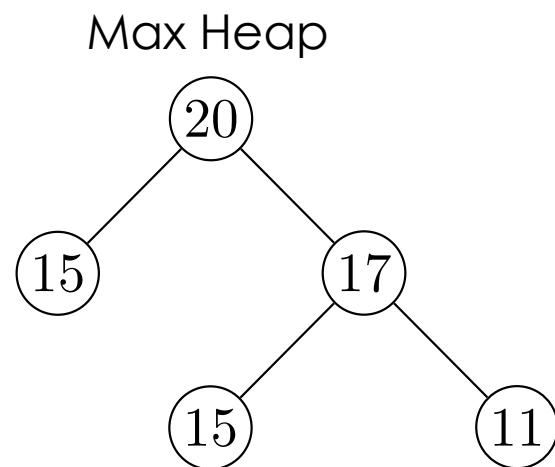
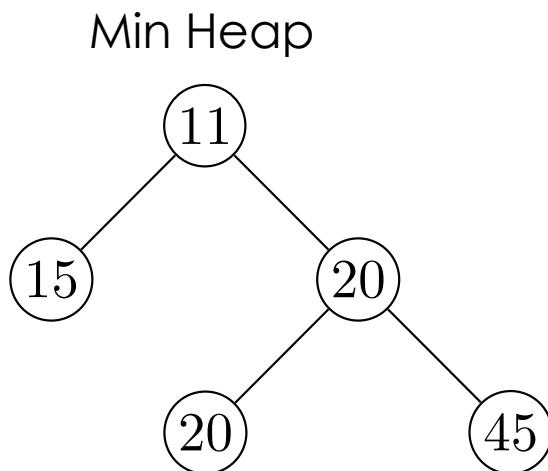
درخت نیمه مرتب (Heap، هرم)

- Max Tree : درختی که مقدار کلید هر گره آن بیشتر یا مساوی فرزندانش باشد.
- Min Tree : درختی که مقدار کلید هر گره آن کمتر یا مساوی فرزندانش باشد.



درخت نیمه مرتب (هرم، Heap)

- درخت دودویی کامل که Max Tree نیز باشد. Max Heap
- درخت دودویی کامل که Min Tree نیز باشد. Min Heap

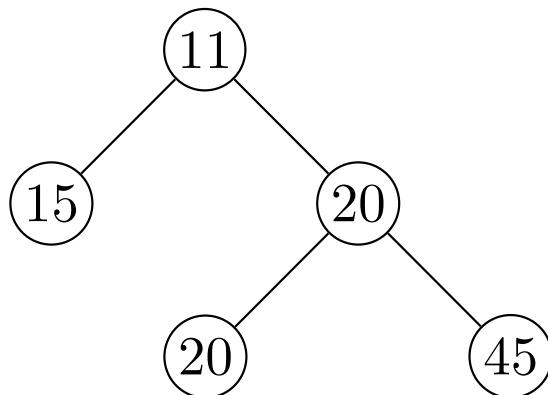


درخت نیمه مرتب (Heap، هرم)

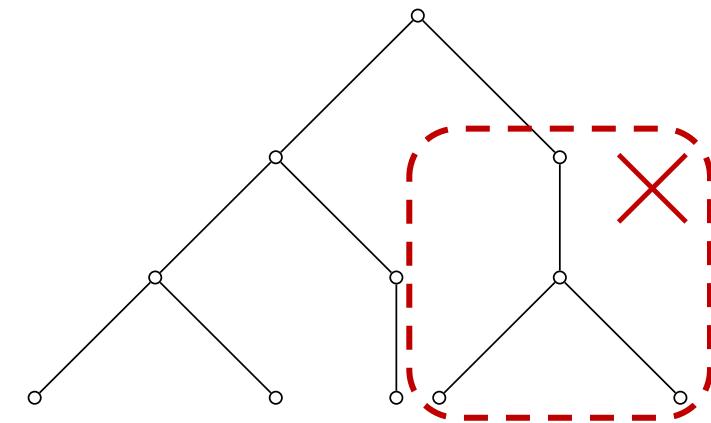
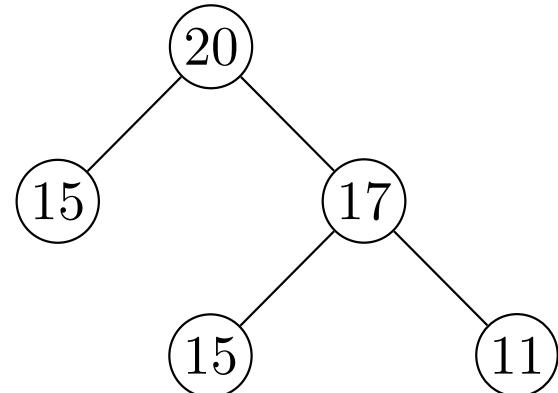
البته اگر نوع ساختار Heap مشخص نشده باشد، به صورت Max Heap پیش فرض آن را در نظر خواهیم گرفت.

- درخت دودویی کامل که Max Tree نیز باشد. Max Heap •
- درخت دودویی کامل که Min Tree نیز باشد. Min Heap •

Min Heap

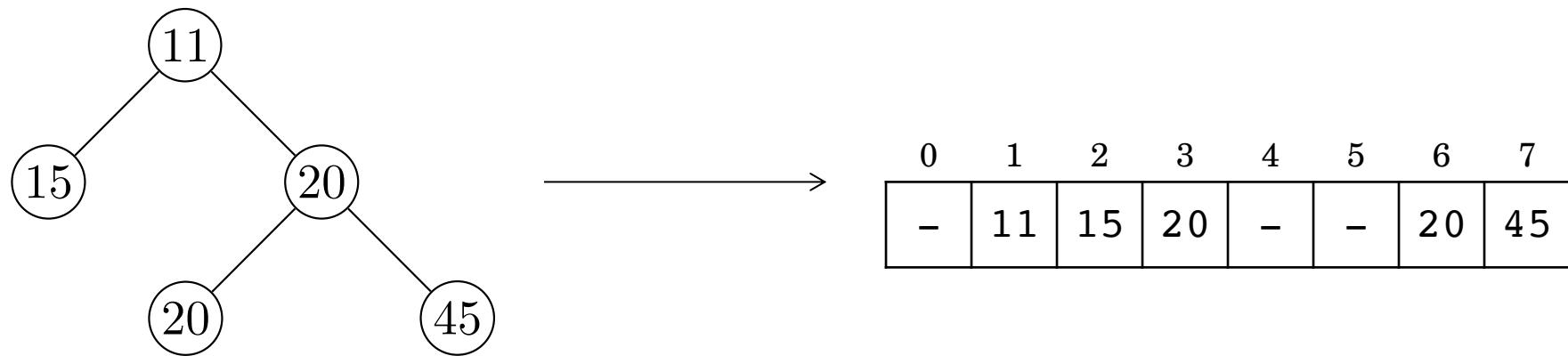


Max Heap



ذخیره سازی

از آنجا که «هرم» یک درخت دودویی کامل است، برای نمایش آن از آرایه ها استفاده می کنیم:



ذخیره سازی

از آنجا که «دخت دده» کاما است، دای نماش آن از آنها استفاده کنند:
پادآوری:

در بحث درخت‌ها پیاده‌سازی درخت دودویی با آرایه نشان داده شد؛ که شماره‌گذاری از ۱ با گره ریشه آغاز می‌شد و تمام گره‌های دیگر از سطح بعدی از چپ به راست شماره‌گذاری می‌شدند.

7

45

نکته: از آنجا که Heap یک درخت کامل است:

- برای نمایش آن از آرایه استفاده می‌شود.
- ارتفاع آن $1 + \lfloor \log_2 n \rfloor$ با فرض سطح ریشه‌ی ۱ است.

ذخیره سازی

از آنجا که « h.e » یک دخت‌دهنگار کاما است، باید نمایش آن را از آنها استفاده کنیم:

- تعداد برگ‌ها $\left\lfloor \frac{n}{2} \right\rfloor$ است.
- شماره آنها نیز $\left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \dots, n$ است.
- شماره پدر گره i ام، $\left\lfloor \frac{i}{2} \right\rfloor$ و شماره فرزندان آن در صورت وجود i و $2i+1$ است.

فرزنده راست
فرزنده چپ

7

45

جستجو در هرم

برای جستجو یک کلید دلخواه در هرم، با توجه به ذخیره سازی آنها در آرایه‌ها و مرتب نبودن عناصر، باید از روش جستجو خطی استفاده کنیم، این یعنی مرتبه اجرایی «جستجو عنصر دلخواه» $O(n)$ است.

مسئله) ساختمان داده ای ارائه کنید که به نحوی کارا بتواند اعمال زیر را انجام دهد:

- درج عنصر :
 - حذف عنصر کمینه :
 - پرسش عنصر کمینه :
- یک عنصر جدید را در مجموعه عناصرهایش اضافه کند.
عنصر کمینه موجود در مجموعه عناصر درج شده را پیدا کند و حذف کند.
عنصر کمینه موجود در مجموعه عناصر درج شده را اعلام کند.

هدف از طراحی این ساختمان داده را فراموش نکنیم!

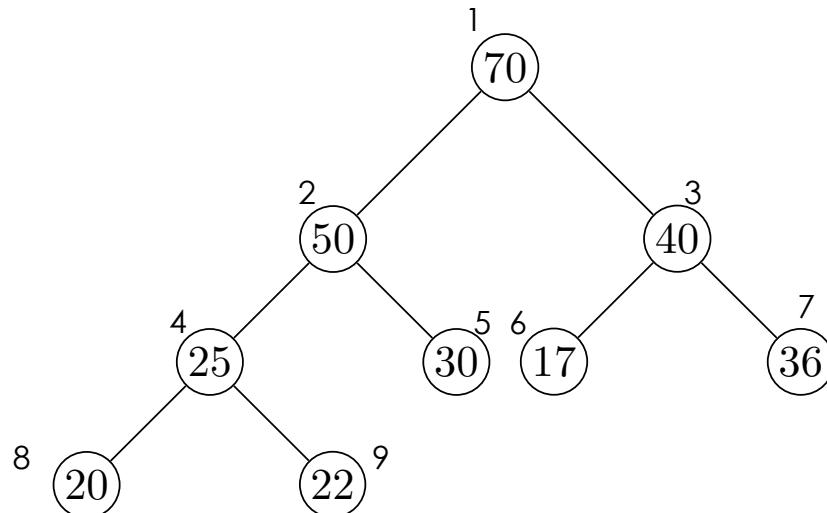
جستجو در هرم

برای یک Max Heap در این درخت $n = 9$ گره، بیشینه در ریشه بوده و گره کمینه

- جستجوی (17) در یکی از برگ‌ها با شماره‌های $5, 6, 7, 8, 9$ گردد.

$$\left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \dots, n = 5, 6, 7, 8, 9$$

خواهد بود.



گردد.

گردد.

جستجوی ب

گردد

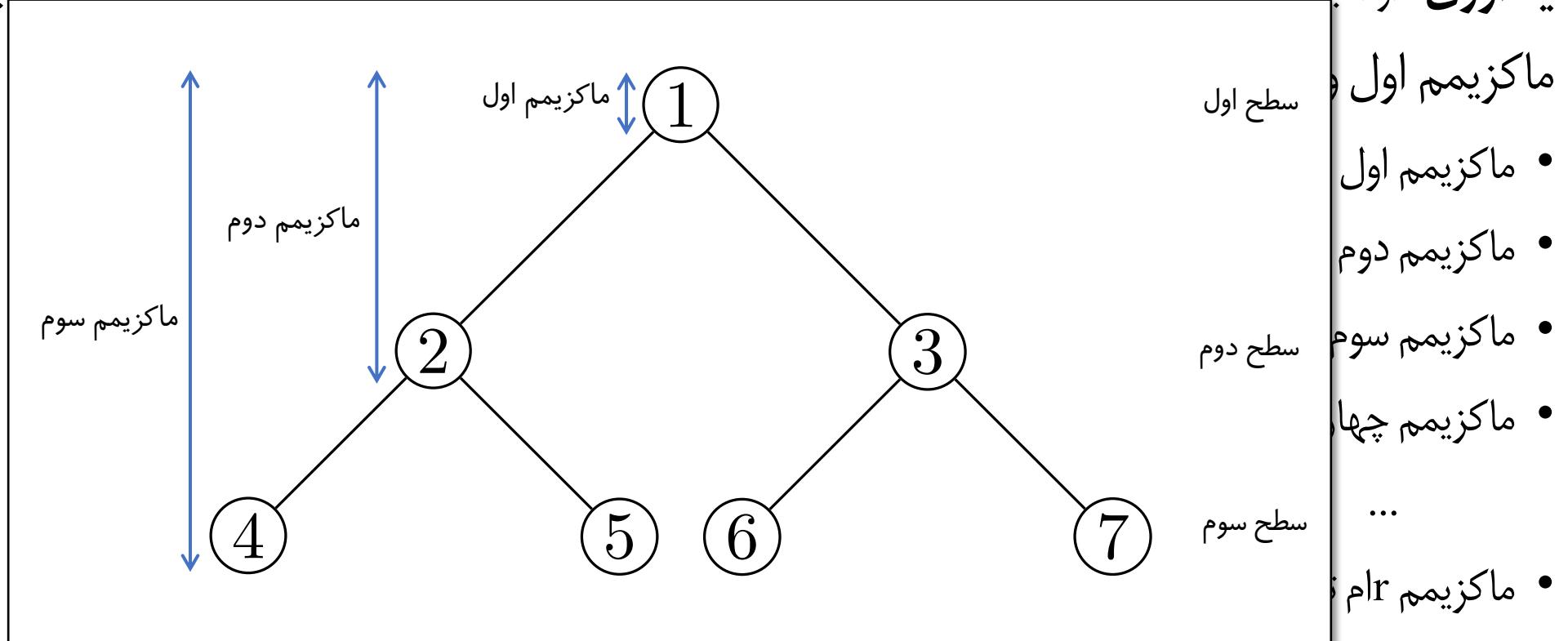
هرم و عنصر پیشینه

یادآوری: از آنجایی که Heap یک درخت کامل است، بنابراین ارتفاع (عمق) $h = \lfloor \log_2^n \rfloor + 1$ آن خواهد بود.
ماکزیمم اول و دوم و سوم و ... و r ام در درخت Max Heap با n کلید متفاوت:

- ماکزیمم اول همواره در ریشه قرار دارد
(خانه $[1]$ آرایه)
- ماکزیمم دوم تا سطح دوم می‌تواند قرار گیرد
(خانه $[2...3]$ آرایه)
- ماکزیمم سوم تا سطح سوم می‌تواند قرار گیرد
(خانه $[2...7]$ آرایه)
- ماکزیمم چهارم تا سطح چهارم می‌تواند قرار گیرد
(خانه $[2...15]$ آرایه)
- ...
- ماکزیمم r ام تا سطح r می‌تواند قرار کرد
(خانه $[2...2^{r-1}]$ آرایه)

هرم و عنصر پیشینه

یادآوری: از آنجا که H را یک درخت کاما است، بنابراین ارتفاع (عمق) آن خواهد بود.



برای کمینه در درخت
هم به همین صورت است.

هرم و عنصر کمینه

یادآوری: از آنجایی که Heap یک درخت کامل است، بنابراین ارتفاع (عمق) $h = \lfloor \log_2^n \rfloor + 1$ آن خواهد بود.

و مینیمم r اول و دوم و سوم و ... و r ام در درخت Min Heap با n کلید متفاوت:

(خانه $[1]$ آرایه)

• مینیمم r اول همواره در ریشه قرار دارد

(خانه $[2...3]$ آرایه)

• مینیمم r دوم تا سطح دوم می‌تواند قرار گیرد

(خانه $[2...7]$ آرایه)

• مینیمم r سوم تا سطح سوم می‌تواند قرار گیرد

(خانه $[2...15]$ آرایه)

• مینیمم r چهارم تا سطح چهارم می‌تواند قرار گیرد

...

(خانه $[2...2^{r-1}]$ آرایه)

• مینیمم r ام تا سطح r می‌تواند قرار کرد

مثال

یک Max Heap با n عنصر متمایز را در نظر بگیرید که با یک آرایه پیاده‌سازی شده است (بزرگ‌ترین عنصر در درایه اول قرار دارد). چهارمین بزرگ‌ترین عنصر در کدامیک از درایه‌های زیر می‌تواند قرار گیرد؟

۱) ۲ یا ۳

۲) ۸ تا ۱۵

۳) ۴، ۵، ۶ یا ۷

۴) همه موارد

مثال

یک Max Heap با n عنصر متمایز را در نظر بگیرید که با یک آرایه پیاده‌سازی شده است (بزرگ‌ترین عنصر در درایه اول قرار دارد). چهارمین بزرگ‌ترین عنصر در کدامیک از درایه‌های زیر می‌تواند قرار گیرد؟

۱) ۲ یا ۳

$$2^r - 1 = 2^4 - 1 = 15$$

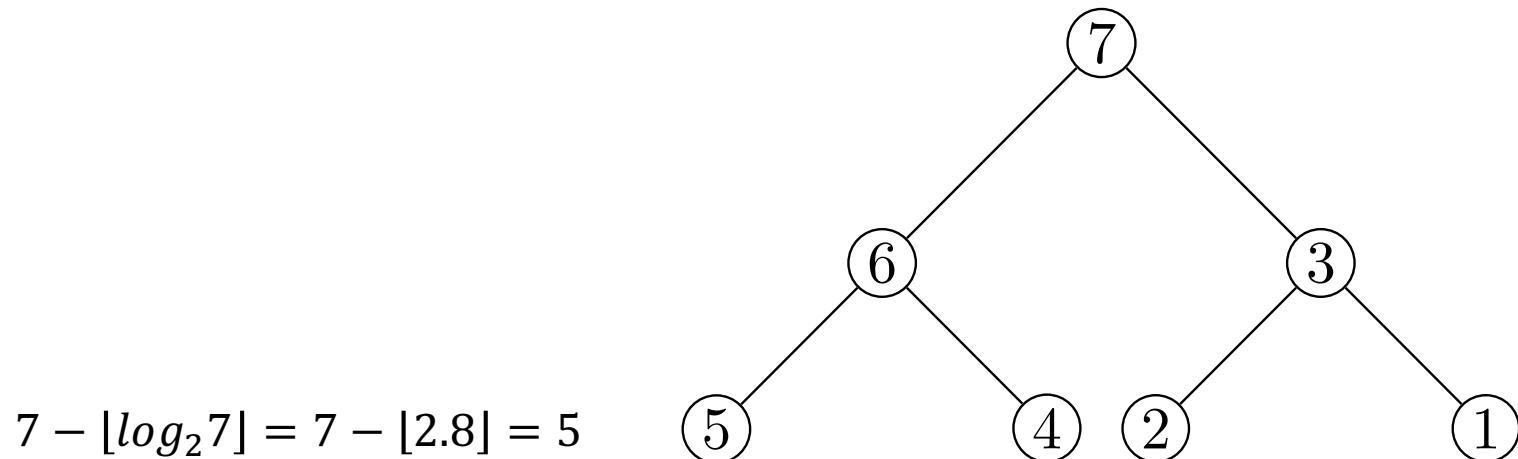
۲) ۸ تا ۱۵

۳) ۴، ۵، ۶ یا ۷

۴) همه موارد

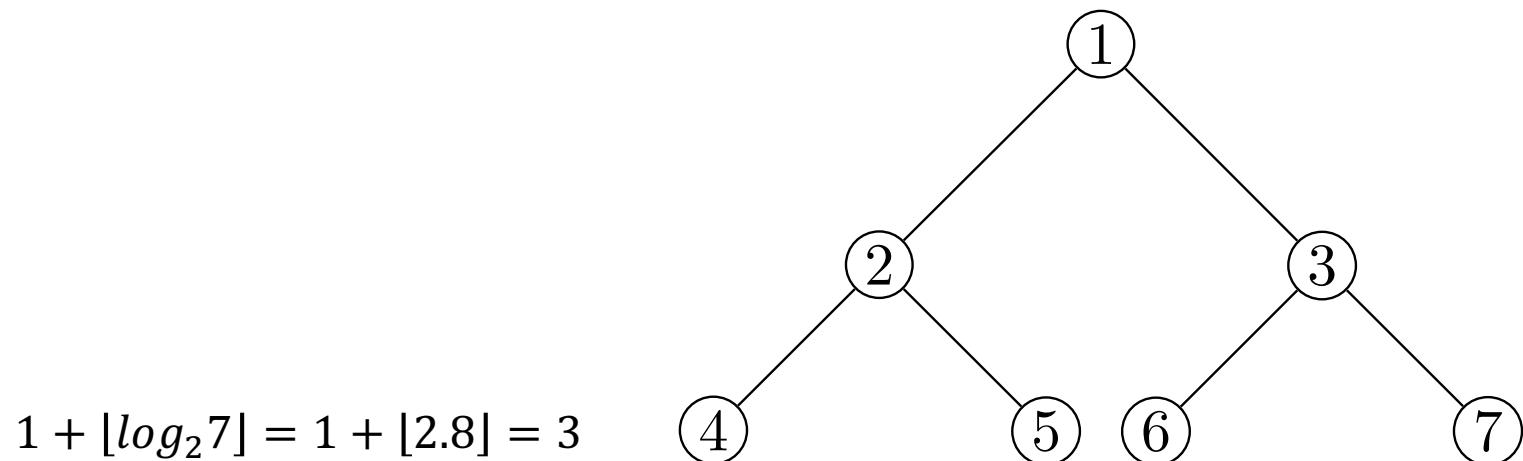
نتیجه

اگر اعداد ۱ تا n در یک Max Heap قرار داشته باشند، بزرگترین عددی که می‌توان در آخرین سطح مشاهده کرد برابر با $n - \lfloor \log_2 n \rfloor$ خواهد بود.



نتیجه

اگر اعداد ۱ تا n در یک Min Heap قرار داشته باشند، کوچکترین عددی که می‌توان در آخرین سطح مشاهده کرد برابر با $1 + \lfloor \log_2 n \rfloor$ خواهد بود.



حفظ ویژگی هرم

روال MAX-HEAPIFY برای حفظ ویژگی Max Heap به کار می‌رود. ورودی آن آرایه‌ی A و اندیس i در آرایه است.

زمانی که این روال فراخوانی می‌شود فرض می‌شود که درخت‌های دودویی مشتق شده از i_{left} و i_{right} خود به تنها یک Max Heap هستند. ولی عنصر $A[i]$ ممکن است کوچک‌تر از فرزندانش باشد. در نتیجه ویژگی Max Heap از بین می‌رود. وظیفه روال MAX-HEAPIFY این است که مقدار موجود در $A[i]$ را به سمت پایین حرکت بدهد تا درخت مشتق شده از i یک Max Heap شود.

حفظ ویژگی هرم

روال برای حفظ ویژگی Max Heap به کار می‌رود. ورودی آن آرایه‌ی A و اندیس i در آرایه است.

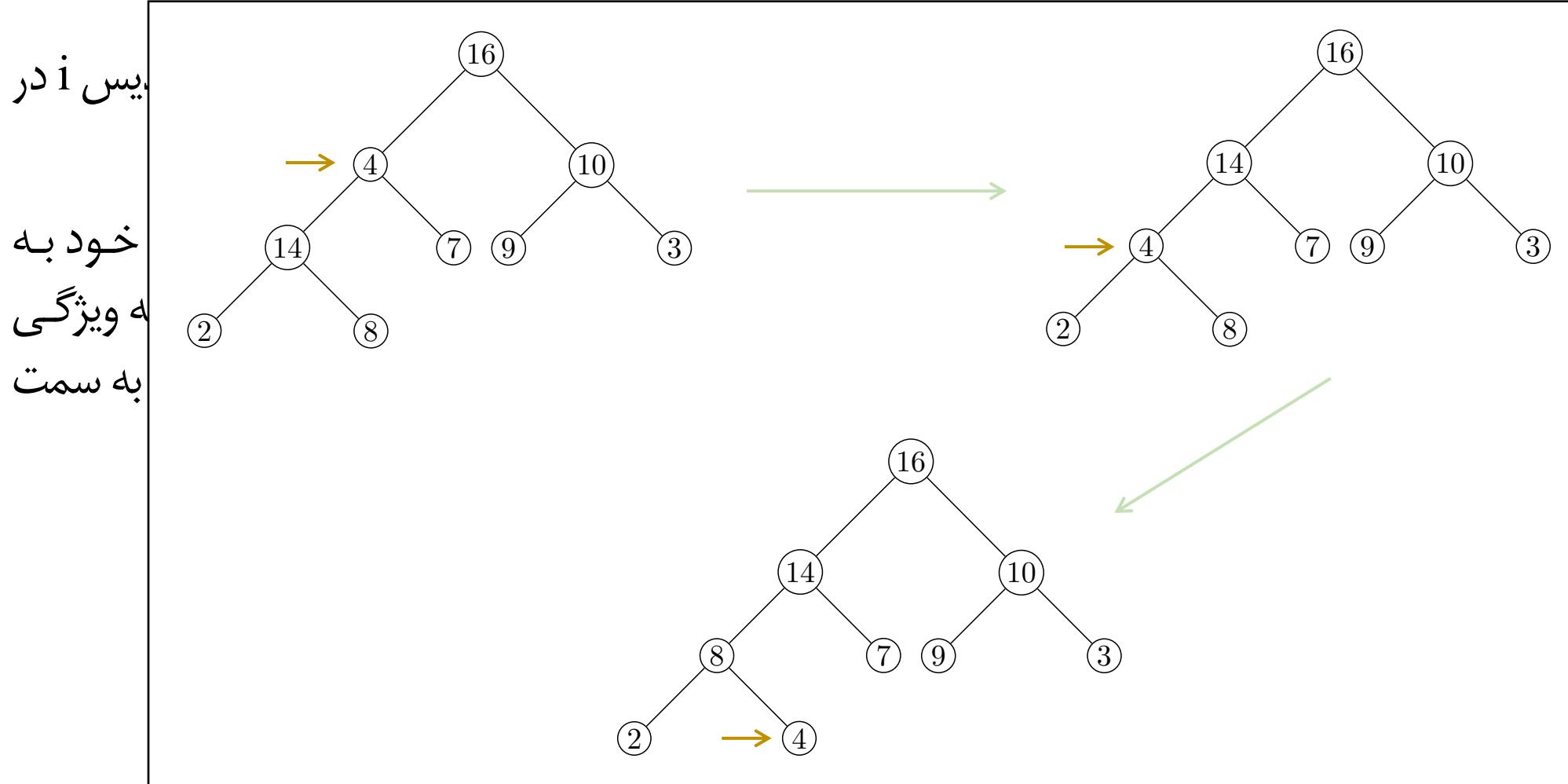
زمانی که این روال فراخواخته شد. در نتیجه ویژگی زیر را به سمت اندیس i خود به اندیس i_{left} و i_{right} منتقل می‌کند. در نتیجه ویژگی زیر را به سمت اندیس i خود در آرایه A[i] داشته باشد.

```
def MAX_HEAPIFY(A, i):
    l=i.left
    r=i.right
    max=i
    if(l<=len(A) and A[l]>A[max]):
        max=l
    if(r<=len(A) and A[r]>A[max]):
        max=r
    if(max!=i):
        swap(max,i)
    return MAX_HEAPIFY(A,max)
return 'done!'
```

زمانی که این روال فراخواخته شد. در نتیجه ویژگی زیر را به سمت اندیس i خود به اندیس i_{left} و i_{right} منتقل می‌کند. در نتیجه ویژگی زیر را به سمت اندیس i خود در آرایه A[i] داشته باشد.

زمان اجرا از مرتبه $O(\log_2 n)$ که متناسب با ارتفاع درخت است.

حفظ ویژگی هرم



ساختن هرم

در اینجا می‌خواهیم با استفاده از روال MAX-HEAPIFY یک آرایه‌ی $A[1\dots n]$ را به یک Max Heap تبدیل کنیم. Max Heap در درخت حرکت کرده و با استفاده از MAX-HEAPIFY آن را به صورت درجا به روال Build-Max-Heap تبدیل می‌کند. (توجه داشته باشید ابتدا i با اندیس آخرین گره‌ای که برگ نیست مقداردهی می‌شود.)

```
def Build_Max_Heap(A):
    heap.size=len(A)
    st=(heap.size)//2
    for i in [st, st-1, st-2, ..., 1]:
        MAX_HEAPIFY(i)
    return 'Built.'
```

شماره آخرین گره غیر برگ →

توجه: روال زمان $O(n)$ صرف می‌کند.

ساختن هرم

```
def BuildMaxHeap(A):
    head = 0
    size = len(A)
    for i in range(size // 2 - 1, -1, -1):
        MaxHeapify(A, i)
    return A
```

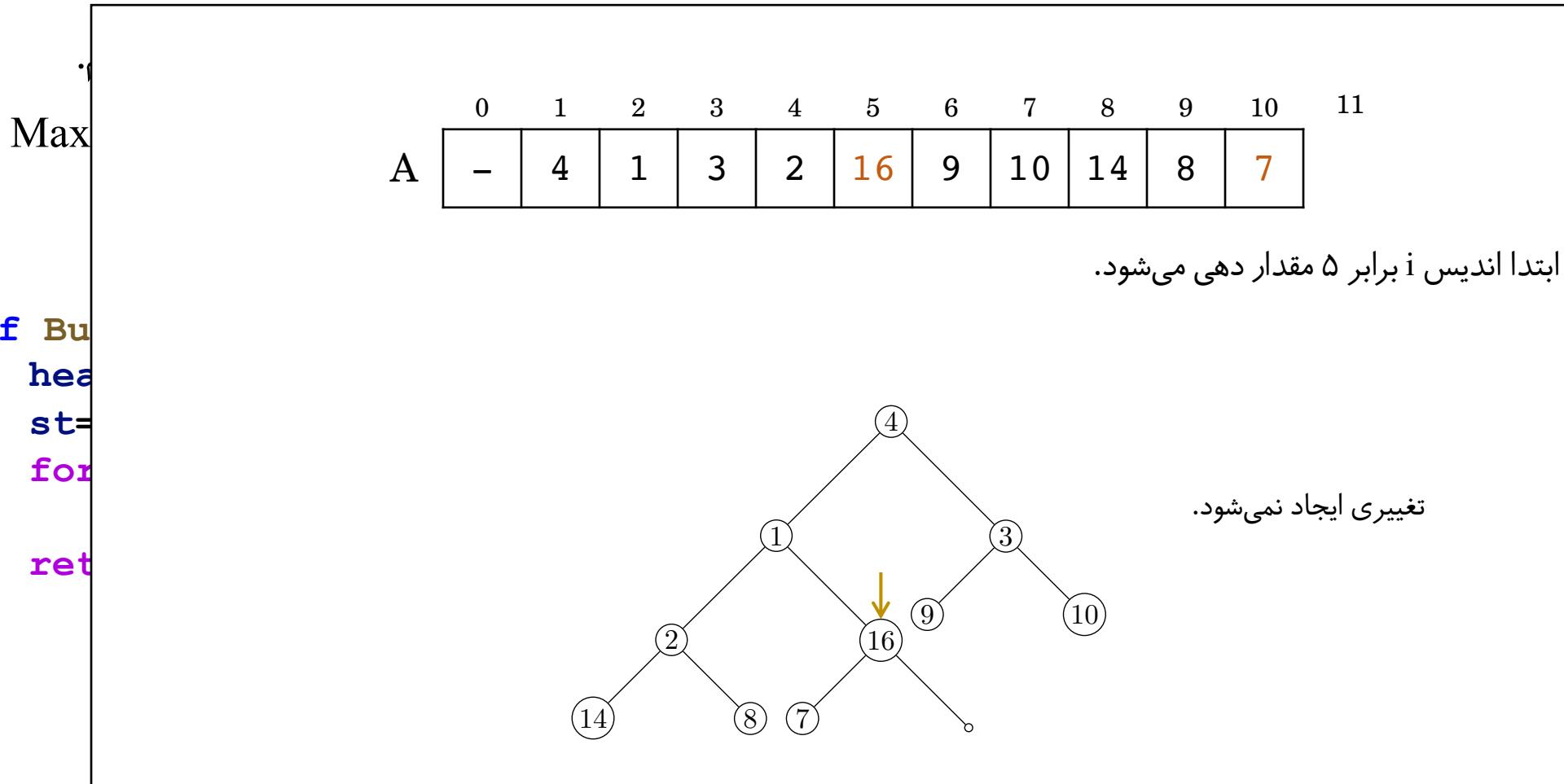
A

	0	1	2	3	4	5	6	7	8	9	10
	-	4	1	3	2	16	9	10	14	8	7

در اینجا می‌توانیم روال تبدیل می‌کنیم.

توجه: روال

ساختن هرم



ساختن هرم

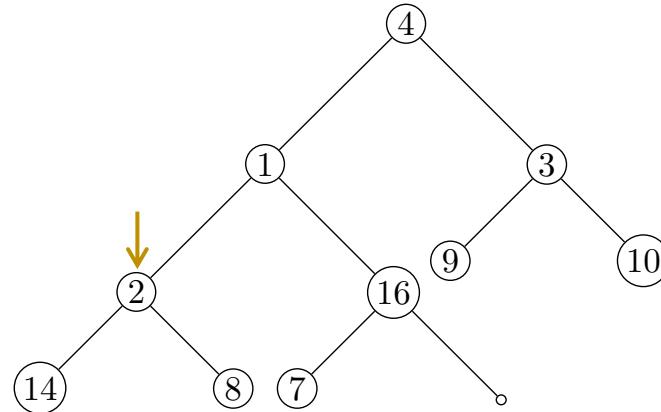
در اینجا می
روال [eap تبدیل می

Max

A	0	1	2	3	4	5	6	7	8	9	10
.	-	4	1	3	2	16	9	10	14	8	7

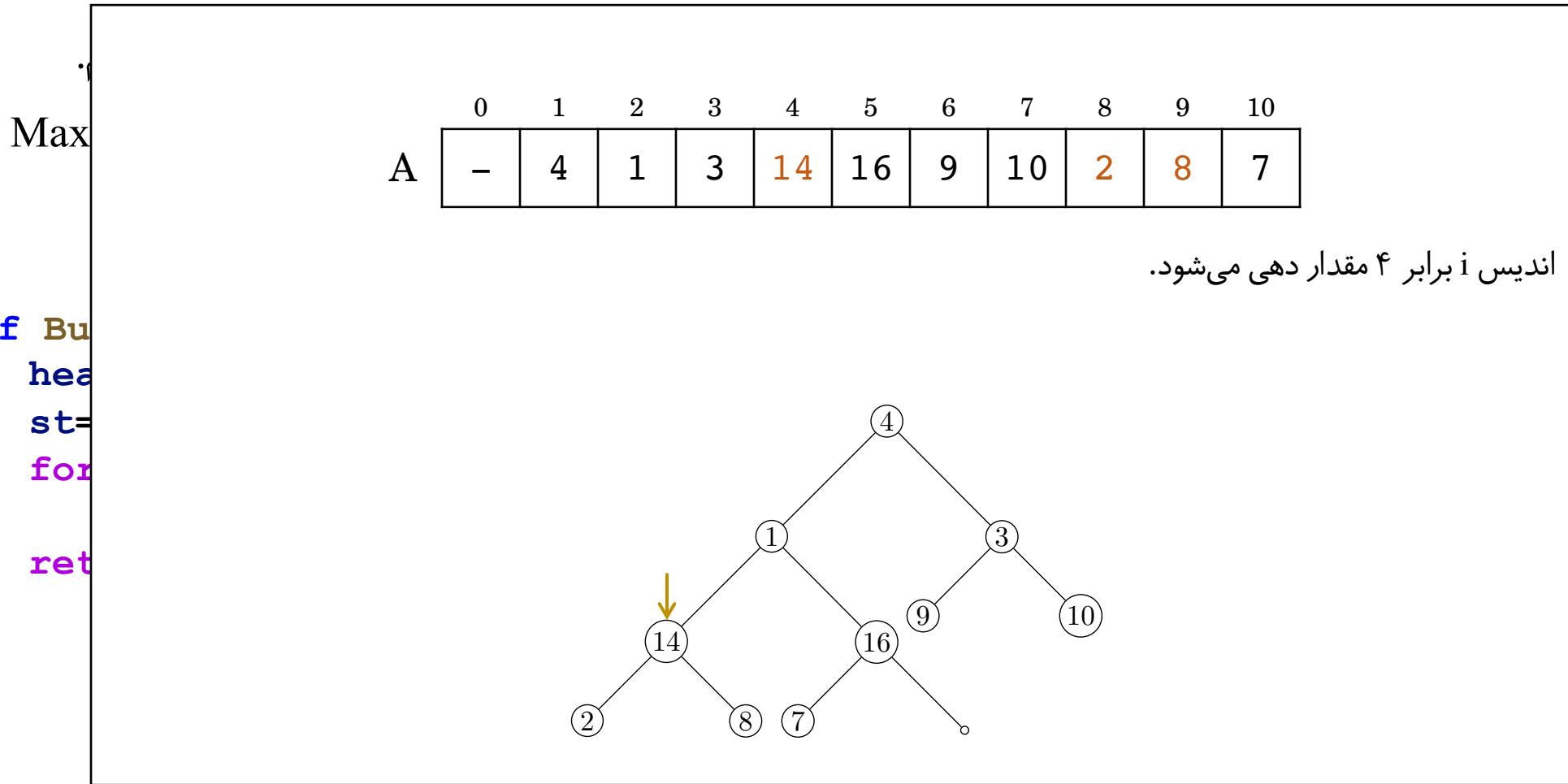
اندیس ۱ برابر ۴ مقدار دهی می شود.

```
def Bu  
hea  
st=
```



توجه: روال

ساختن هرم



ساختن هرم

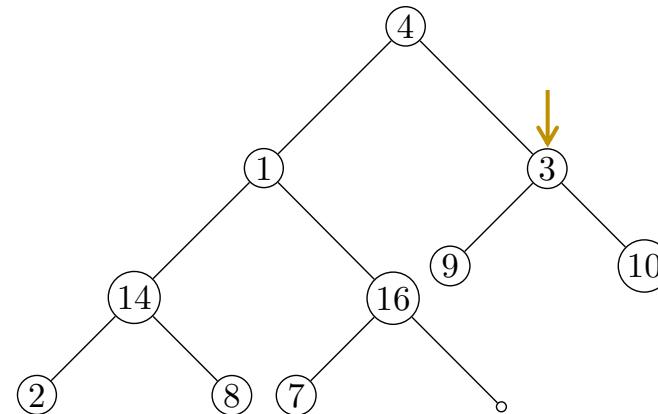
در اینجا می‌
روال [eap تبدیل می‌

Max

A	0	1	2	3	4	5	6	7	8	9	10
.	-	4	1	3	14	16	9	10	2	8	7

اندیس ۱ برابر ۳ مقدار دهی می‌شود.

```
def Bu  
hea  
st=
```



توجه: روال

ساختن هرم

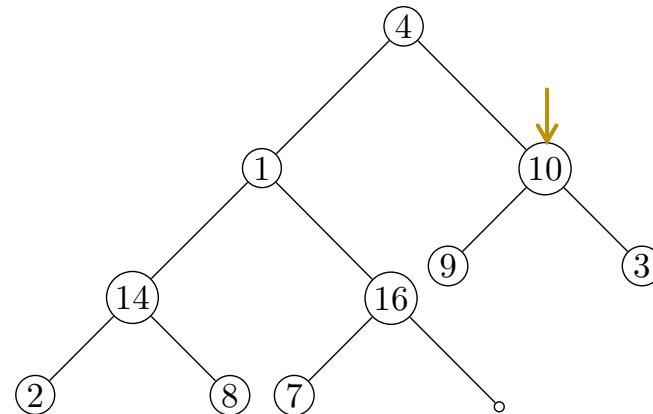
در اینجا می
روال [eap تبدیل می

Max

A	0	1	2	3	4	5	6	7	8	9	10
.	-	4	1	10	14	16	9	3	2	8	7

اندیس ۱ برابر ۳ مقدار دهی می شود.

```
def Bu  
hea  
st=
```



توجه: روال

ساختن هرم

در اینجا می‌
روال heap
تبديل می‌

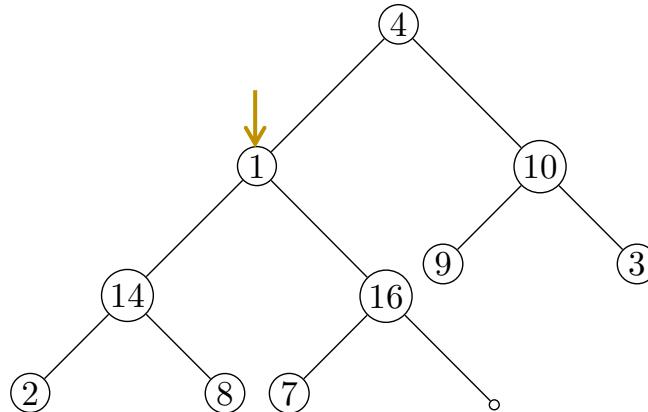
Max

A

0	1	2	3	4	5	6	7	8	9	10
-	4	1	10	14	16	9	3	2	8	7

اندیس ۱ برابر ۲ مقدار دهی می‌شود.

```
def BuildHeap(A, n):  
    head = 0  
    start = 0  
    for i in range(0, n):  
        if A[i] < A[head]:  
            A[i], A[head] = A[head], A[i]  
            head = i  
    return A
```



توجه: روال

ساختن هرم

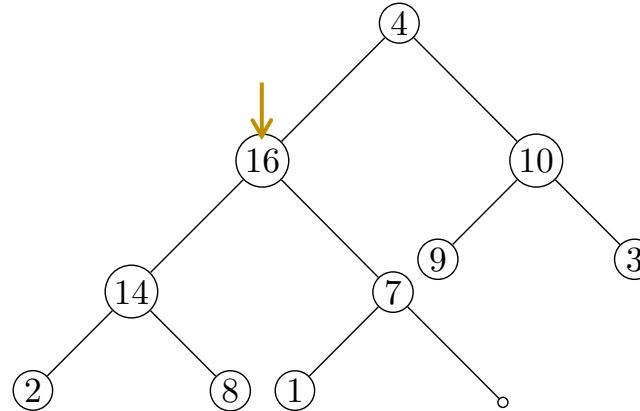
در اینجا می‌
روال [eap تبدیل می‌

Max

A	0	1	2	3	4	5	6	7	8	9	10
.	-	4	16	10	14	7	9	3	2	8	1

اندیس ۱ برابر ۲ مقدار دهی می‌شود.

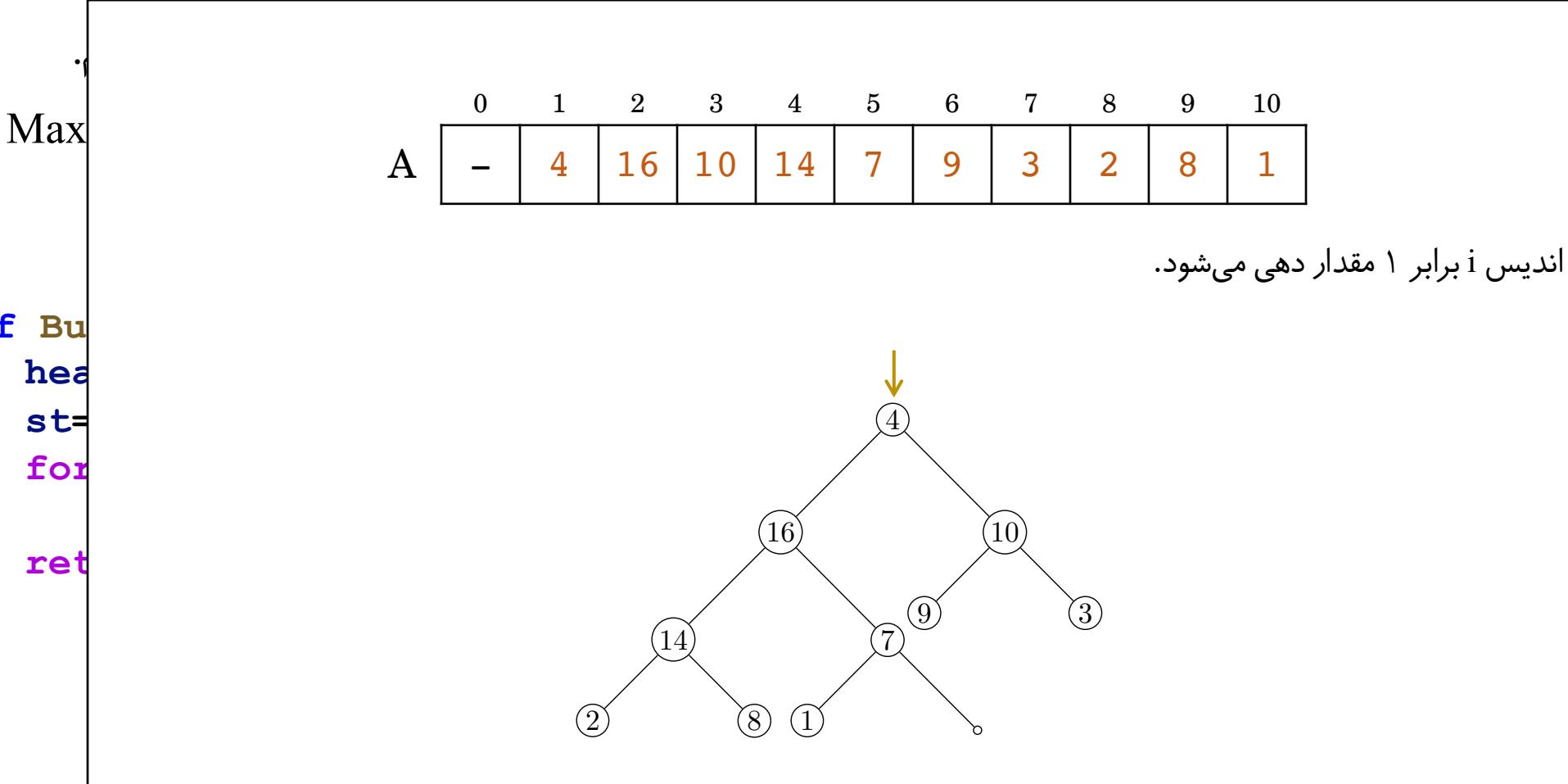
```
def Bu  
hea  
st=
```



توجه: روال

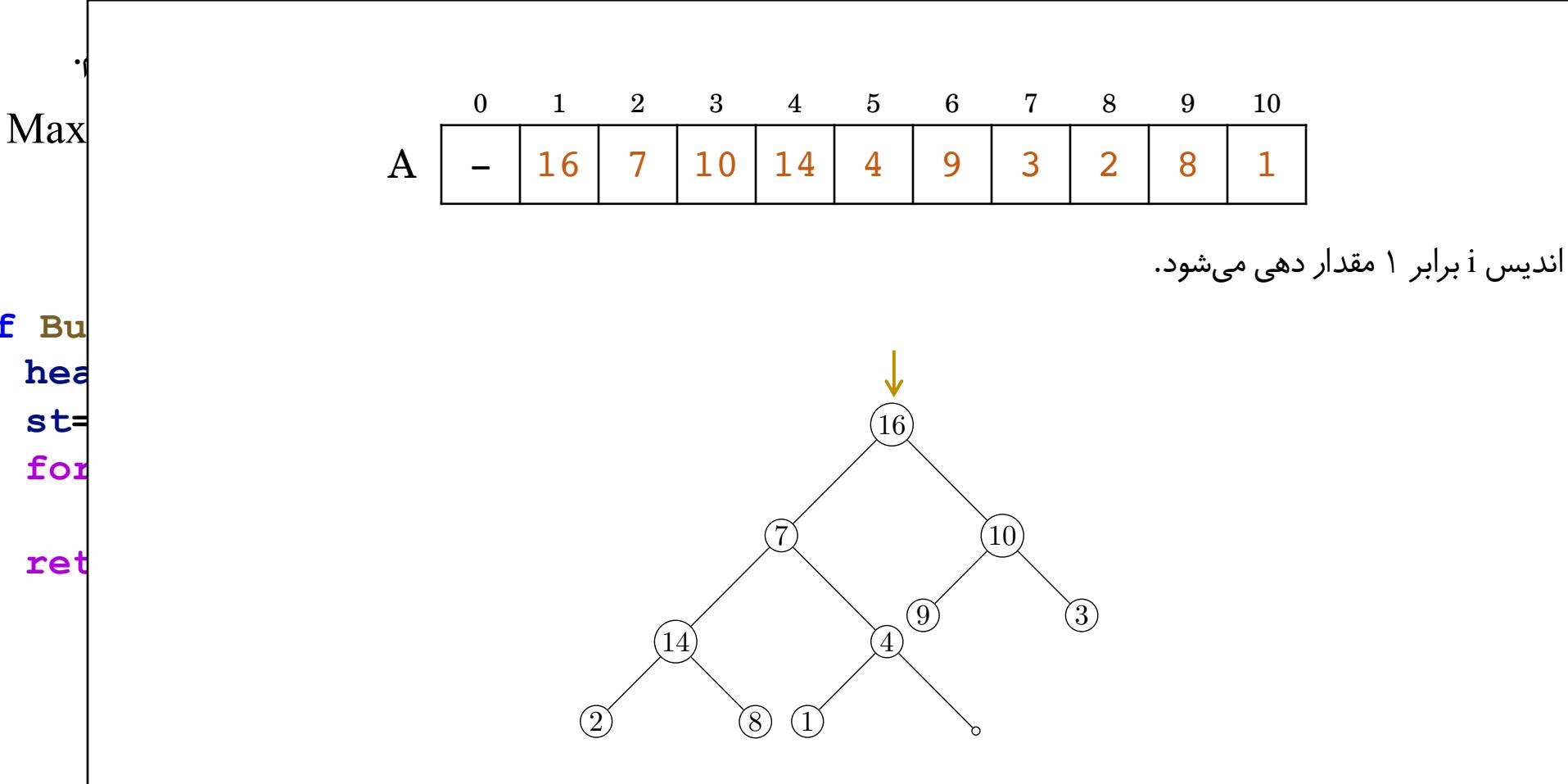
ساختن هرم

در اینجا می
روال [eap تبدیل می



ساختن هرم

در اینجا می‌
روال [eap تبدیل می‌



حذف عنصر پیشینه

این روال در Max Heap عنصر با بزرگ‌ترین کلید را از درخت حذف کرده و برمی‌گرداند.

از آنجا که در Max Heap بزرگ‌ترین عنصر در ریشه قرار دارد. برای این کار ابتدا ریشه را از درخت خارج و سپس آخرین عنصر هرم(آرایه) را در ریشه قرار می‌دهد. با این کار فرزندان ریشه Max Heap باقی خواهند ماند اما عنصر ریشه جدید ممکن است ویژگی Max Heap را نداشته باشد.

از این رو برای بازیابی هرم، یک MAX-HEAPIFY به روی عنصر ریشه کافیست.

حذف عنصر پیشینه

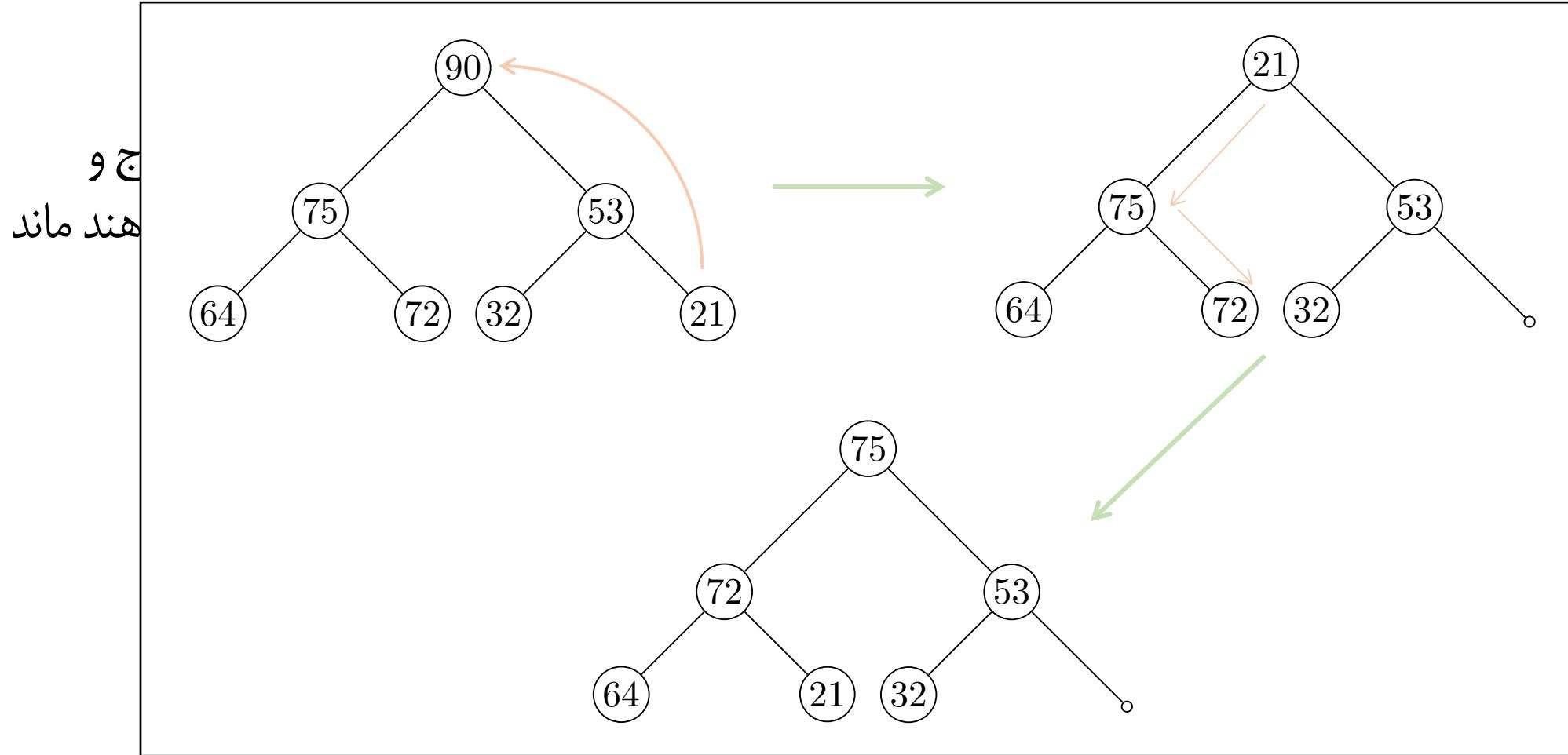
این روال در Max Heap عنصر با بزرگ‌ترین کلید را از درخت حذف کرده و برمی‌گرداند.
از آنجا که در Max Heap بزرگ‌ترین عنصر در ریشه قرار دارد. برای این کار ابتدا ریشه را از درخت خارج و باقی ماند.

```
def Extract_MAX(A):
    if(len(A)<1):
        return 'Heap is empty'
    result=A[1]
    A[1]=A[len(A)-1]
    len(A)=len(A)-1
    MAX_HEAPIFY(A,1)
    return result
```

سپس آخرین عنصر هم
اما عنصر ریشه جدید م
از این رو برای بازیابی ه

پس زمان اجرا این عمل از $O(\log n)$ است.

حذف عنصر پیشینه



افزایش مقدار گره

اين روال کلید عنصری از Max Heap با اندیس i را به مقداری بيشتر از قبلش افزایش می‌دهد. چون ممکن است با افزایش کلید $A[i]$ درخت ويژگی Max-Heap را از دست بدهد.

اين تابع مسیر اين گره تا ريشه را برای يافتن مكان مناسب برای اين کلید می‌پيماید. در طی اين مسیر، کلید افزایش يافته مکررا با پدرش مقایسه می‌شود. اگر اين کلید از پدرش بزرگ‌تر باشد. با آن جا به جا می‌شود. در غير اين صورت روال به پایان می‌دهد.

```
def HEAP_INCREASE_KEY(A, i, key) :
    A[i] = key
    par = i // 2 # parent(i)
    while (i > 0 and A[par] < A[i]):
        swap(A[i], A[par])
        i = i // 2 # i = parent(i)
    return 'done.'
```

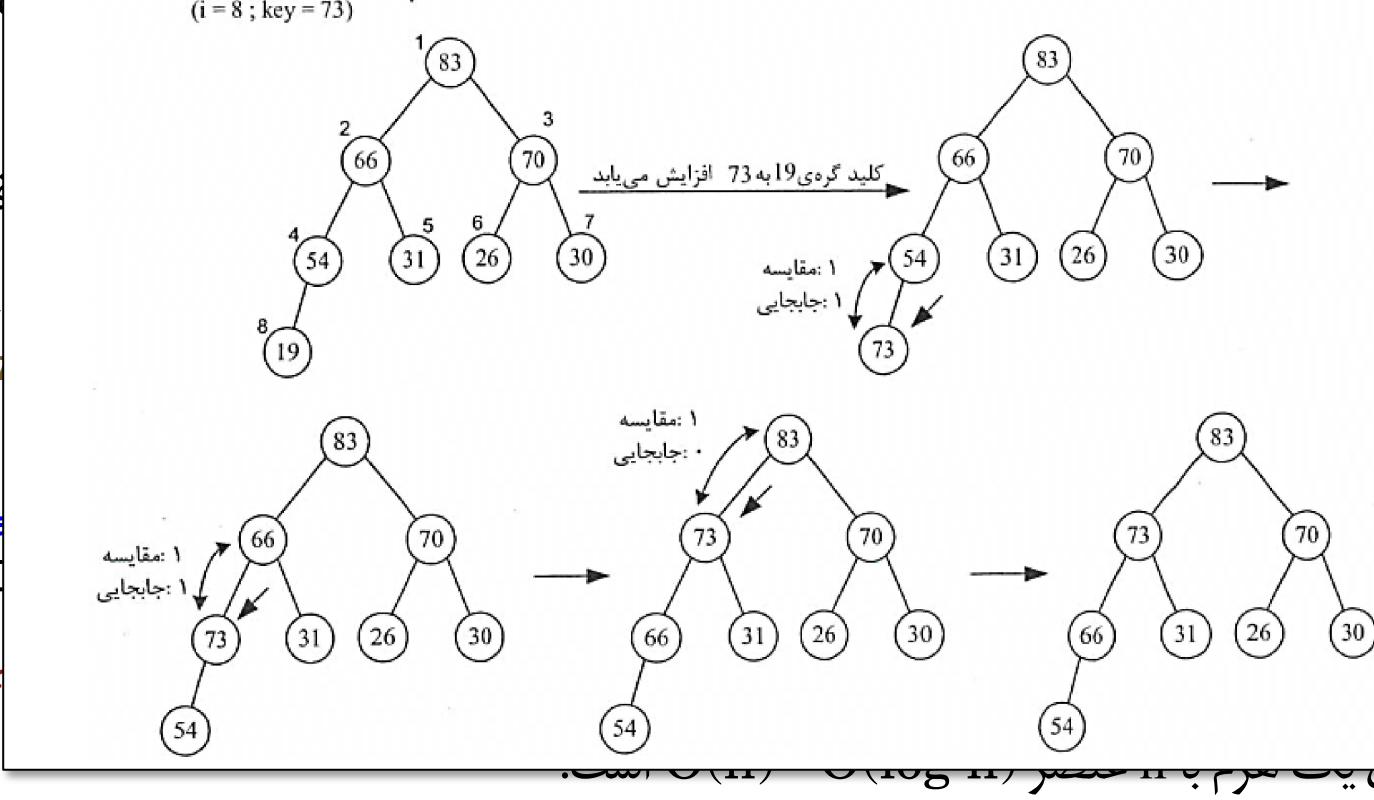
زمان اجرای اين روال روی يك هرم با n عنصر $O(h) = O(\log n)$ است.

افزایش مقدار گره

این روال کلید عنصری از Max Heap است. اندیشه مقادیر شرکت افزاش یعنی ممکن است با افزایش کلید

افزایش یافته مکررا با پایان می‌دهد.

```
def HEAP_INCREASER(A, i):
    A[i] = key
    par = i // 2 #
    while(i > 0 and A[i] > A[par]):
        swap(A, i, par)
        i = i // 2
    return 'done'
```



زمان اجرای این روال روی یک سری بزرگ از این روش

p درخت ویژگی A[i]

این تابع مسیر این گره پدرش مقایسه می‌شود

افزایش مقدار گره

این روال کلید عنصری از Max Heap با اندیس i را به مقداری بیشتر از قبلش افزایش می‌دهد. چون ممکن است با افزایش کلید $A[i]$ درخت ویژگی Max-Heap را از دست بدهد.

مکررا با

```
def HEAP_DECREASE_KEY(A, i, key):
    A[i] = key
    MAX_HEAPIFY(A,i)
    return 'done.'

def HEAP_INCREASE_KEY(A, i, key):
    swap(A[i],A[par])
    i=i//2 # i = parent(i)
    return 'done.'
```

این تابع مد برای کاهش مقدار یک گره، کافیست تا از MAX-HEAPIFY استفاده کنیم!
پدرس مقابله

زمان اجرای این روال روی یک هرم با n عنصر $O(h) = O(\log n)$ است.

درج گره در هرم

ابتدا گره‌ای جدید با کلید ∞ - به عنوان آخرین گره (در انتهای آرایه) در درخت درج می‌شود. سپس با استفاده از روند افزایش کلید، مقدار گره به مقدار مورد نظر افزایش یافته و در مکان صحیح خود قرار می‌گیرد.

```
def insert(A, key):
    A.append(-∞)
    HEAP_INCREASE_KEY(A, len(A), key)
    return 'done.'
```

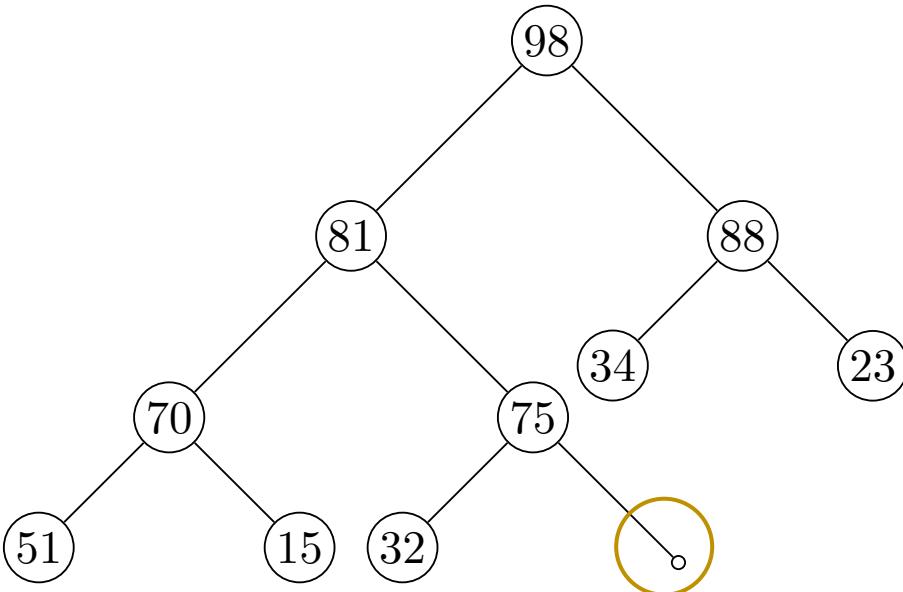
با توجه به آن که گره درج شونده ممکن است تاریشه جابه‌جا شود، زمان درج حداقل $O(h)$ خواهد بود. از طرفی ارتفاع یک هرم حداقل $1 + \lfloor \log n \rfloor$ است. بنابرای زمان درج یک گره در هرم $O(\log n)$ خواهد بود.

درج گره در هرم

ابتدا گره
سپس با
می‌گیرد.

برای مثال میخواهیم در هرم-بیشینه زیر گره‌ای با مقدار ۸۶ درج کنیم.

د قرار
def in
A.a
HEAD
ret
یک هرم



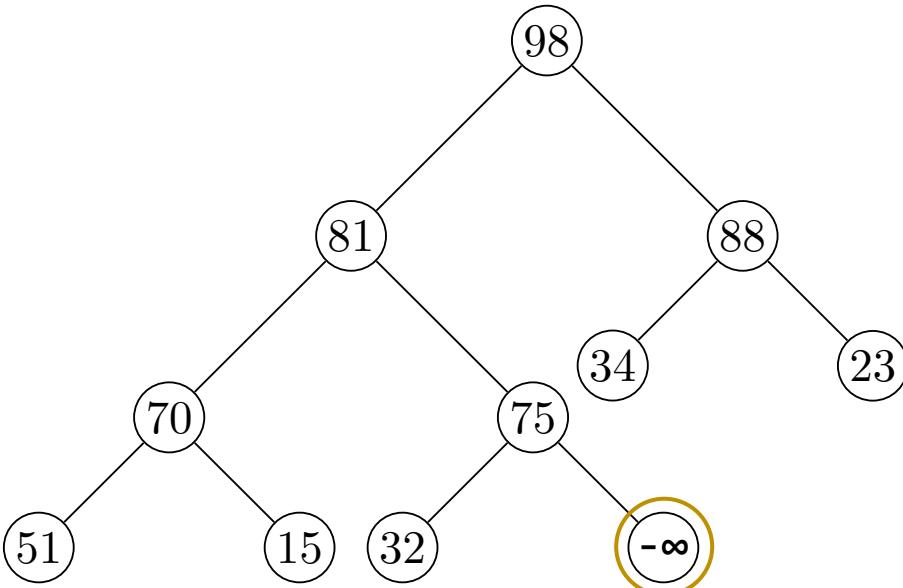
با توجه به
حداکثر 1

درج گره در هرم

```
def in  
A.a  
HEA  
ret
```

د قرار

یک هرم

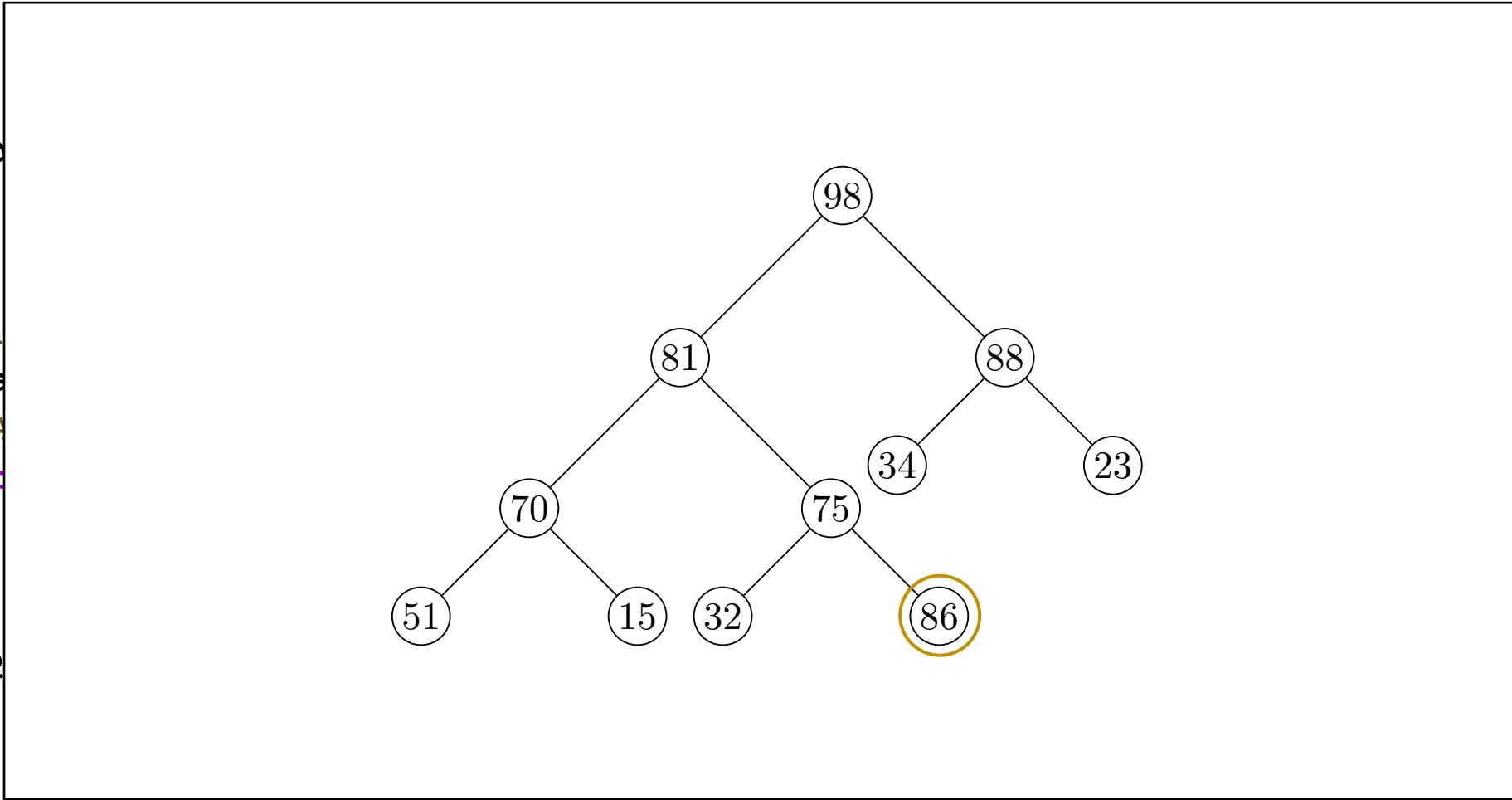


ابتدا گره سپس با می‌گیرد.

با توجه به حداقل 1

درج گره در هرم

```
def inser(A,a,HEAD):
    ret
```

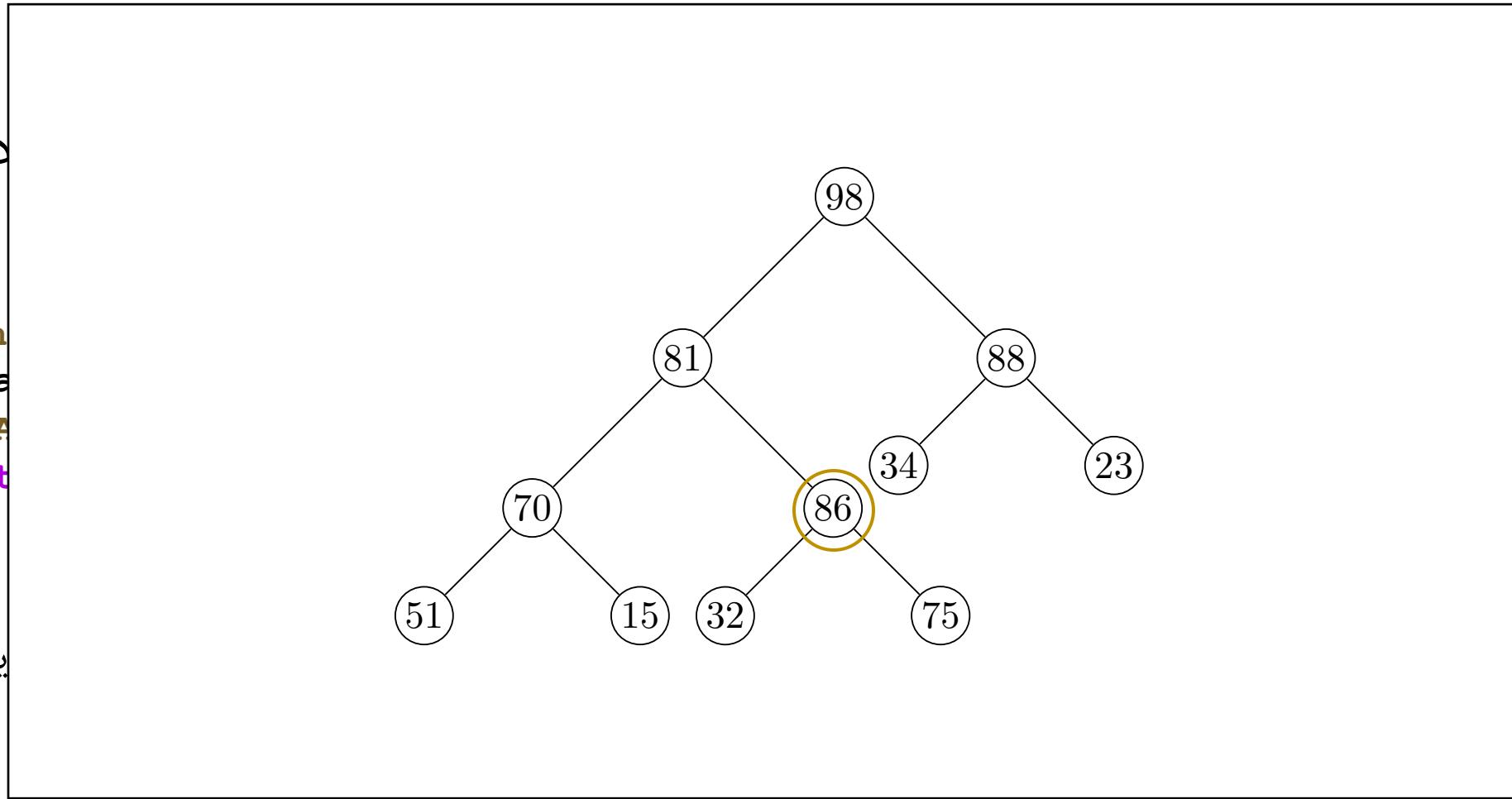


ابتدا گره
سپس با
می‌گیرد.

با توجه به
حداکثر 1

درج گره در هرم

```
def inser(A, a, HEAD):
    ret
```

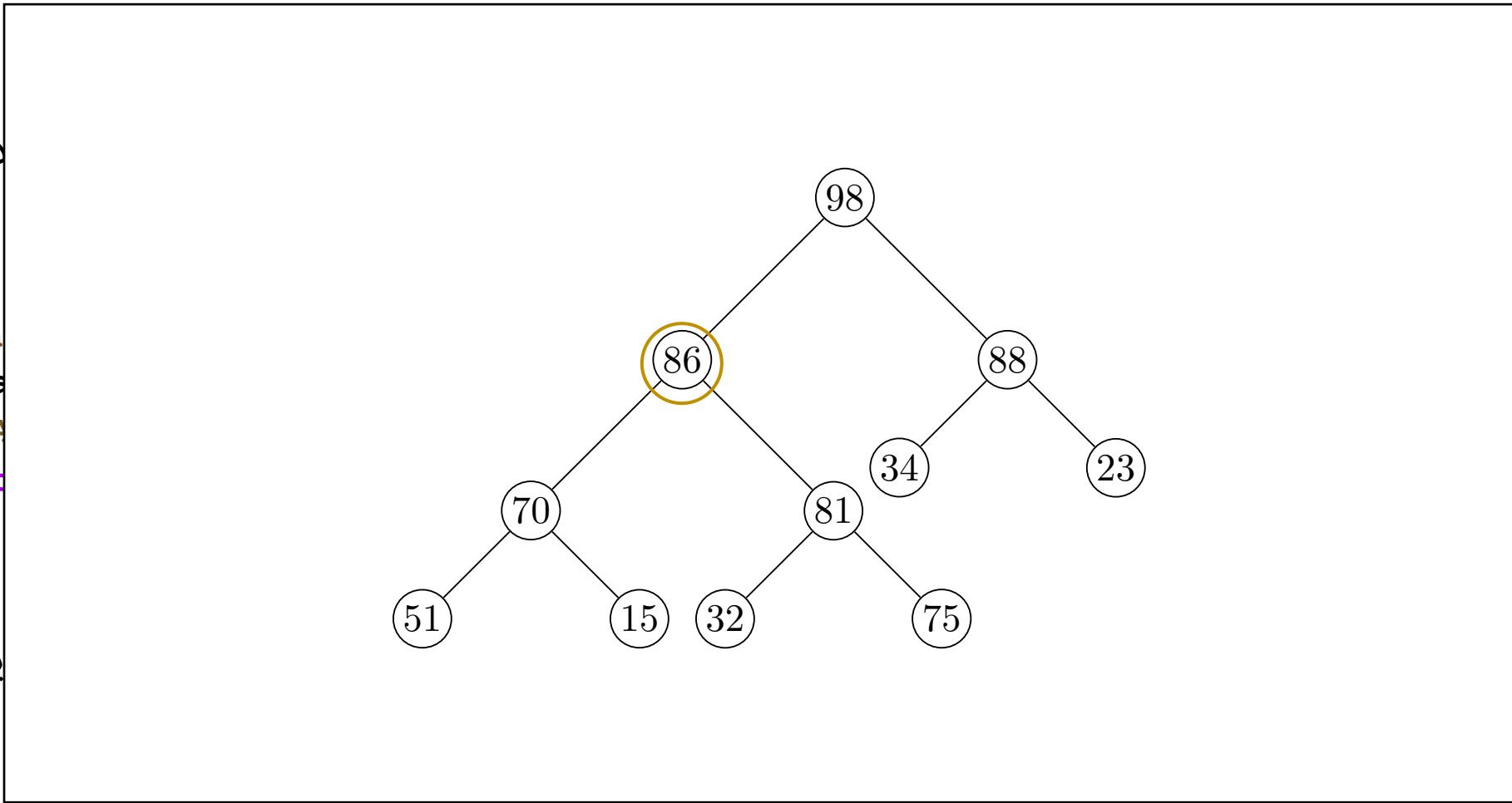


ابتدا گره
سپس با
می‌گیرد.

با توجه به
حداکثر 1

درج گره در هرم

```
def inser(A,a,HEAD):
    ret
```



ابتدا گره
سپس با
می‌گیرد.

با توجه به
حداکثر 1

مرتب سازی بر اساس هرم

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 $[n]A$ 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: $O(n) + (n-1)O(\log n) = O(n \log n)$

مرتب سازی هرم

در مرتب سازی هرم ابتدا با استفاده از روال Build-Max-Heap آرایه ورودی $A[1...n]$ به Max Heap تبدیل می شود. از آنجا که بزرگ ترین عنصر آرایه در ریشه قرار دارد، می تواند با یک جابه جایی 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
```

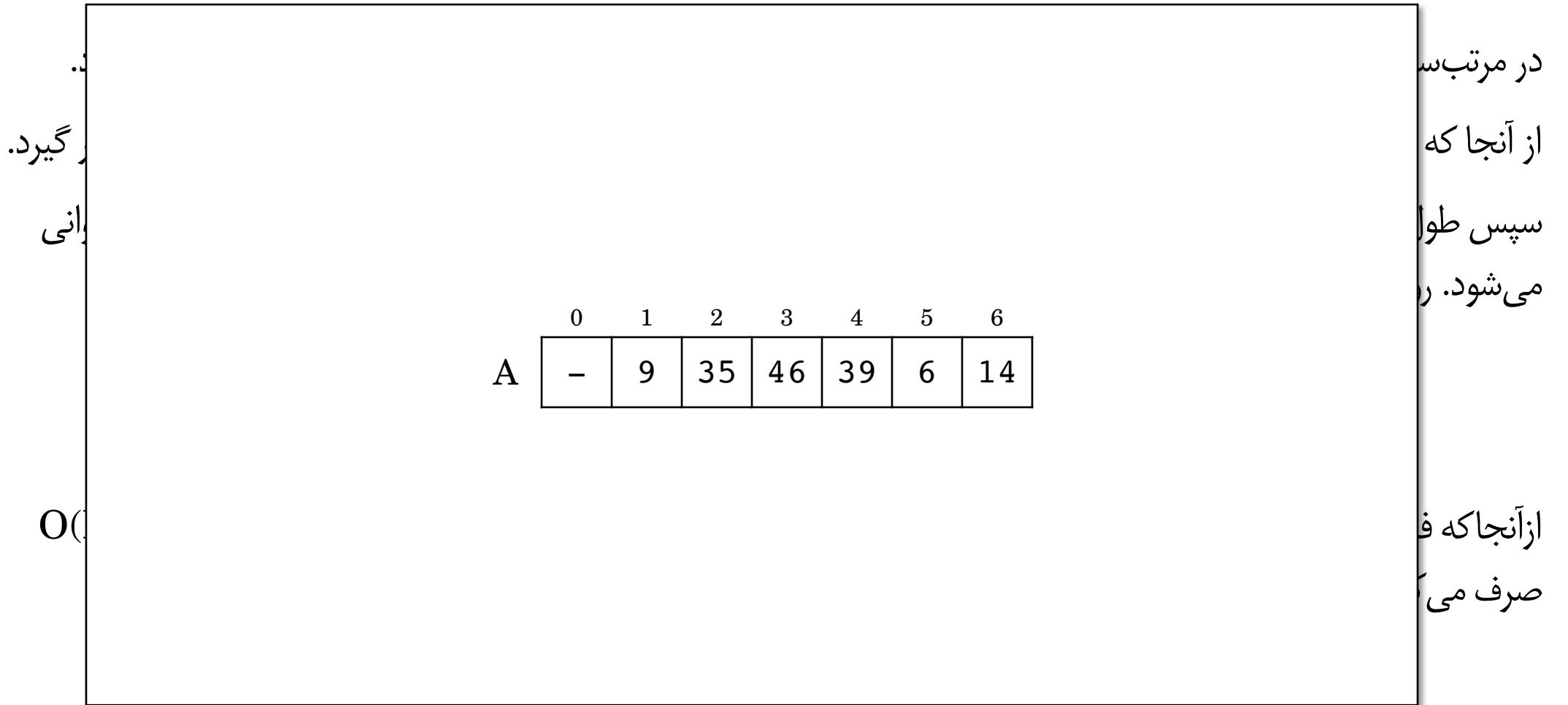
$O(\log n)$ زمان

سپس طول هرم یکی کم می شود. روال این روند را تا

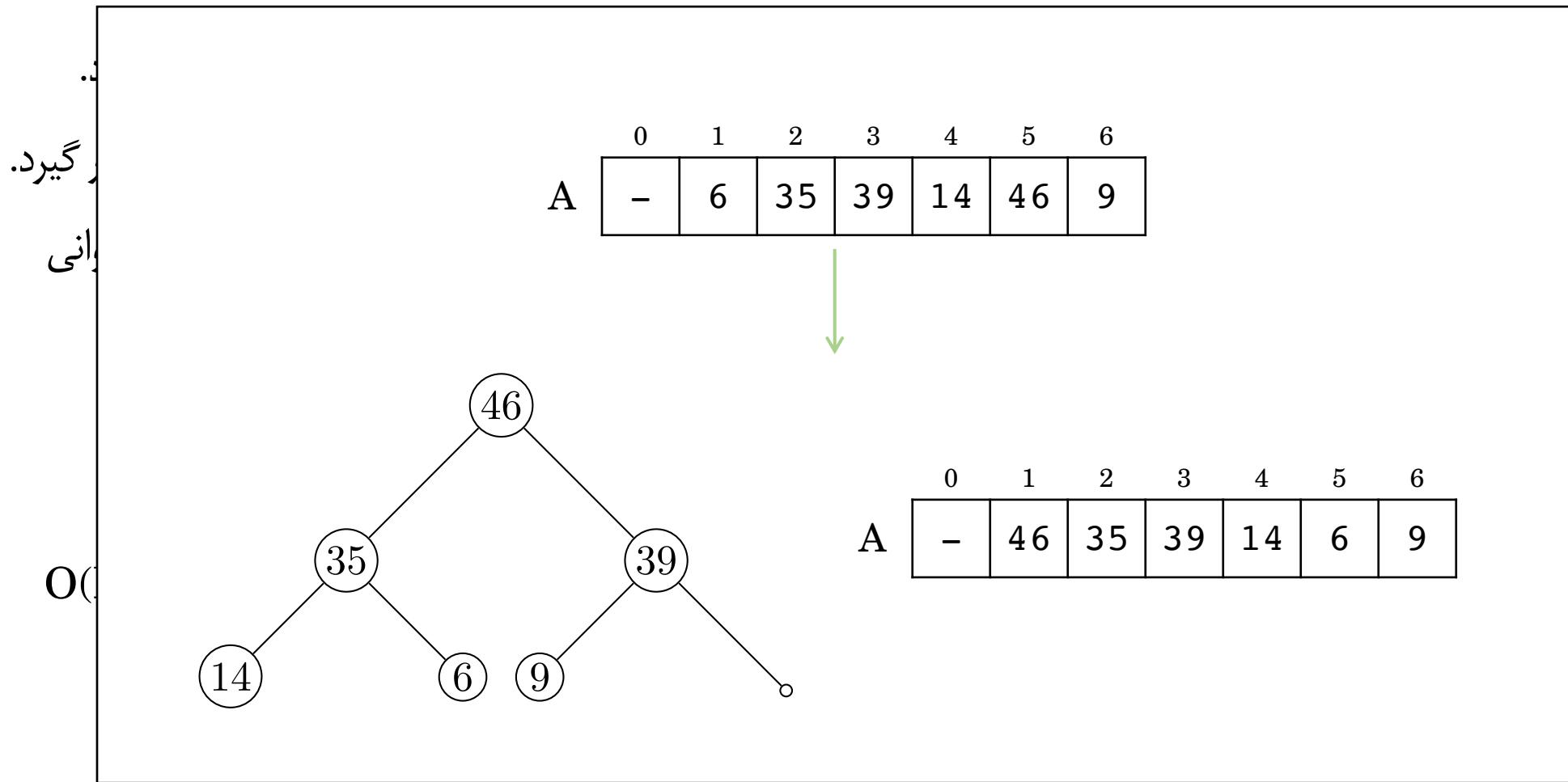
از آنجا که فراخوانی Heap

صرف می کنند؛ زمان مرتب سازی هرم برابر است با: $O(n) + (n-1)O(\log n) = O(n \log n)$

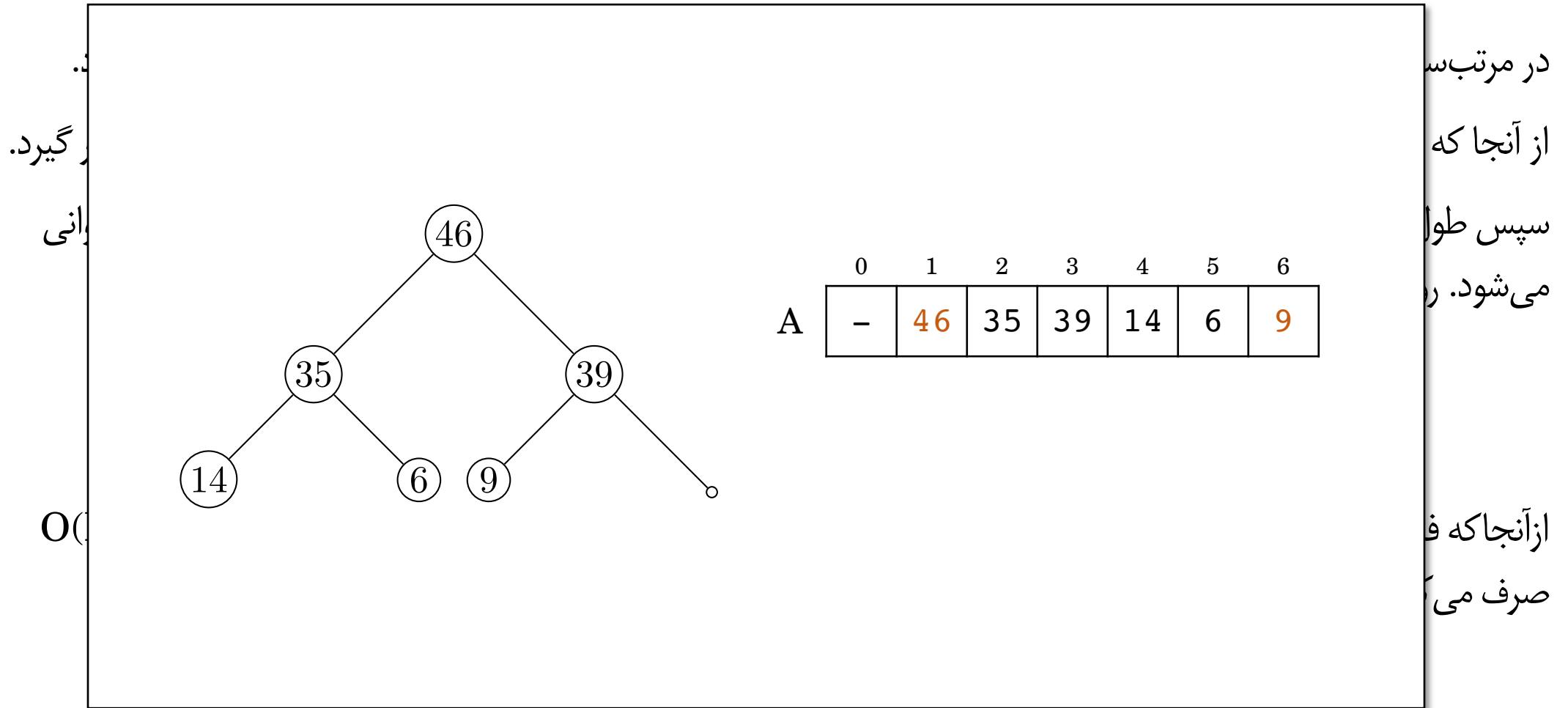
مرتب سازی هرم



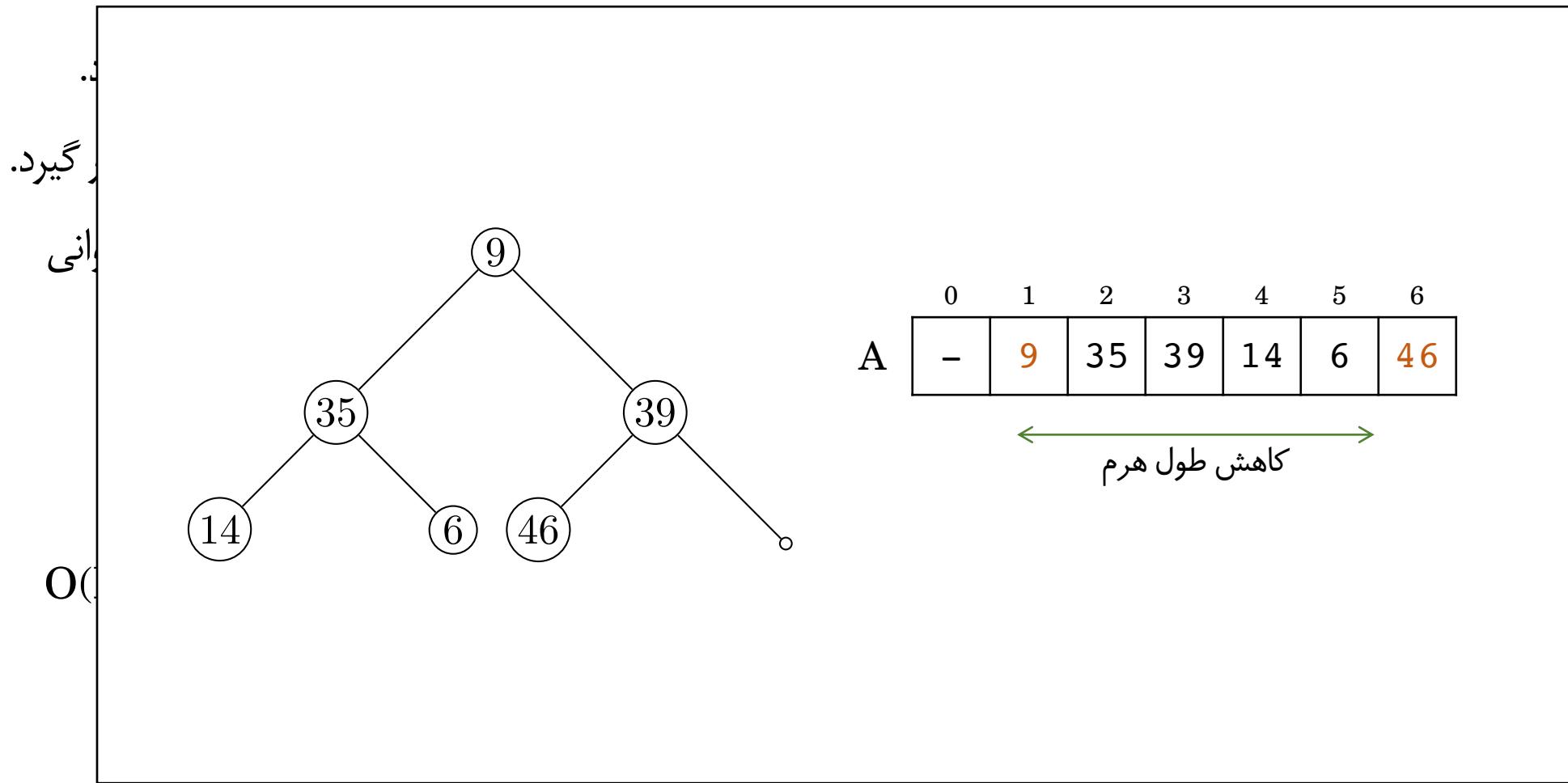
مرتب سازی هرم



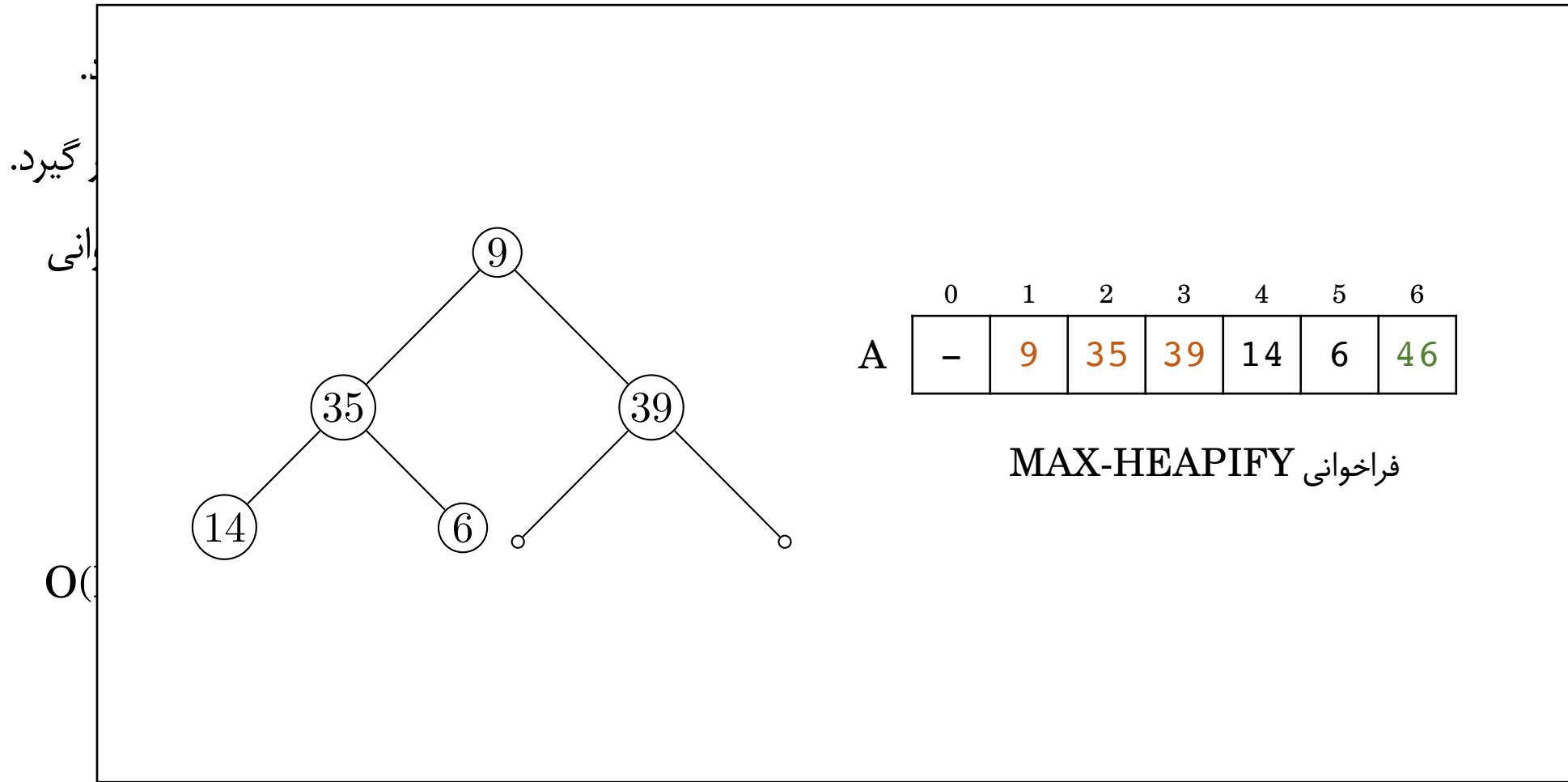
مرتب سازی هرم



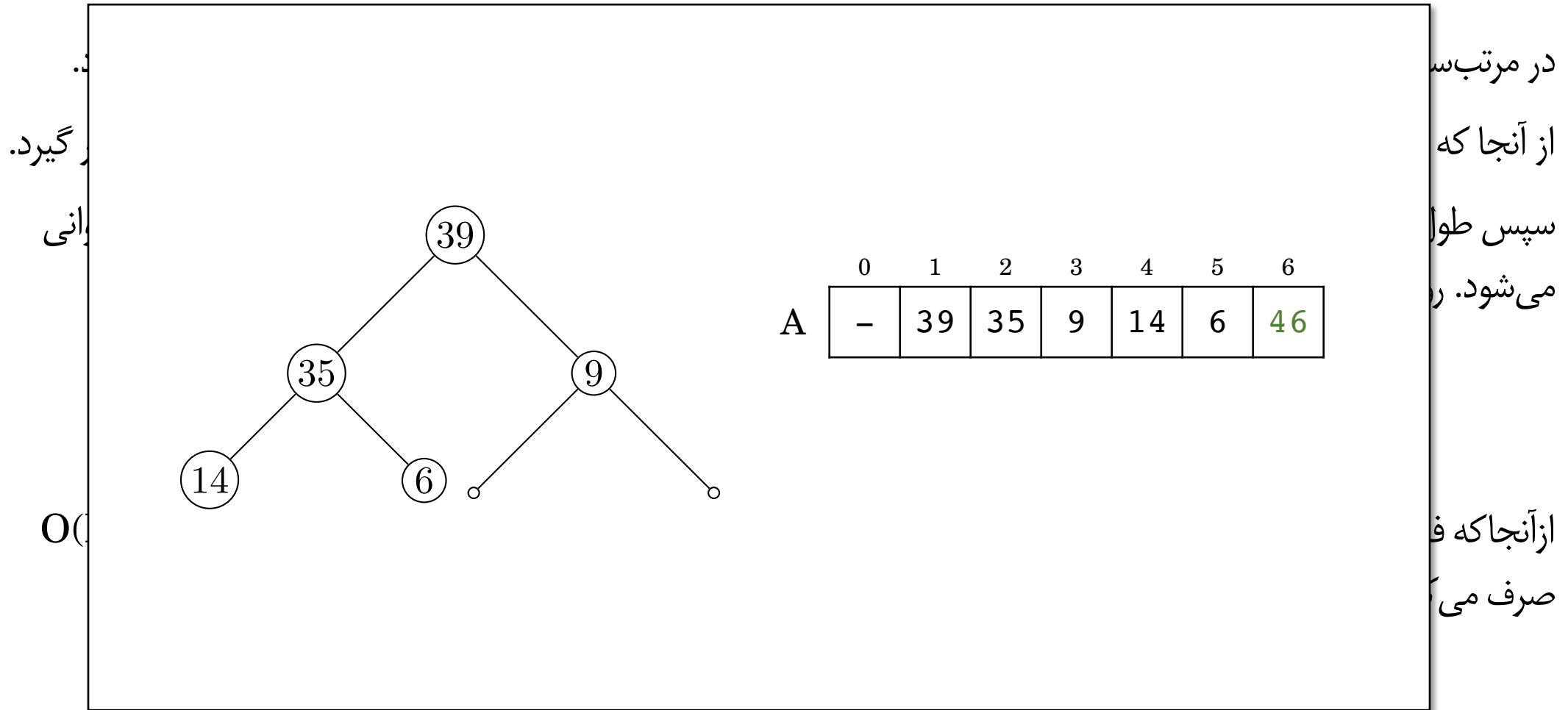
مرتب سازی هرم



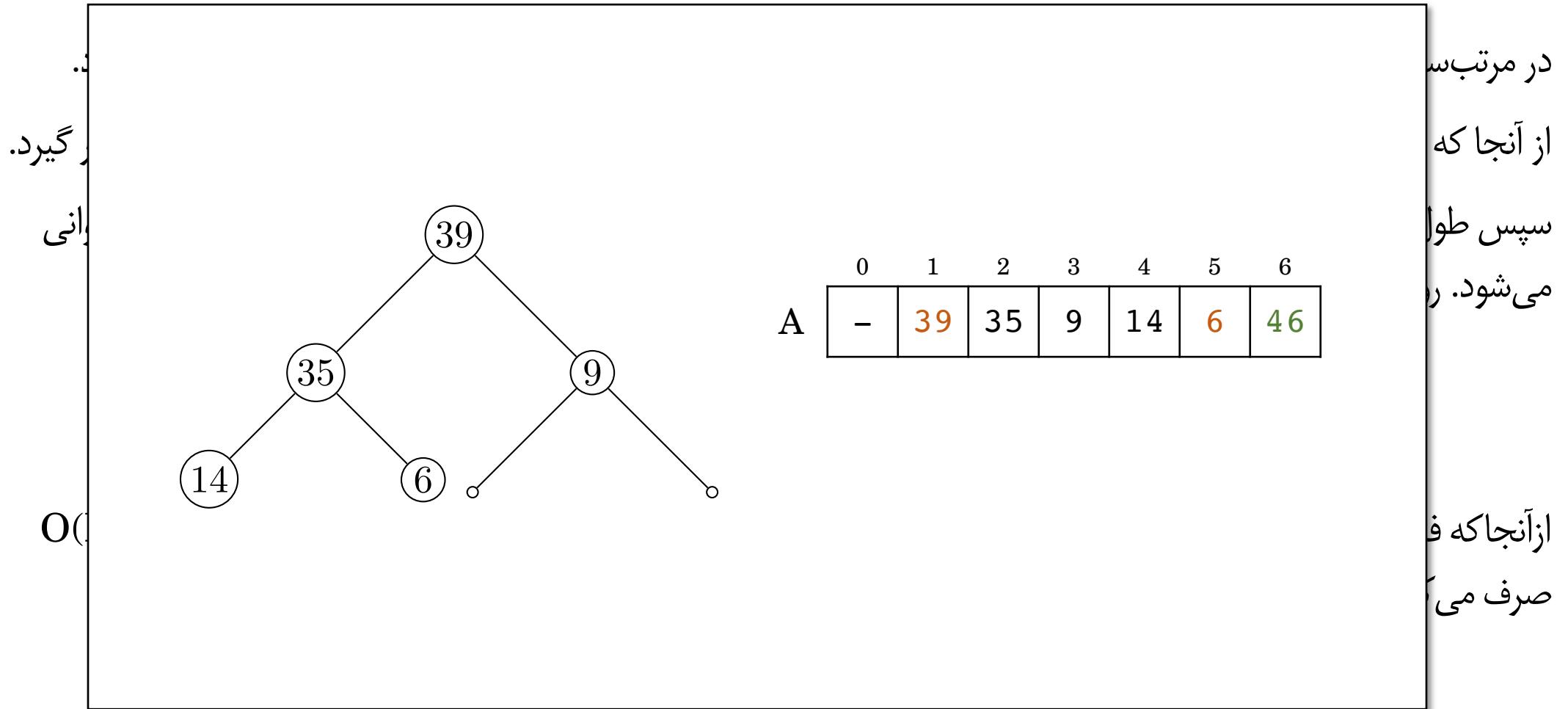
مرتب سازی هرم



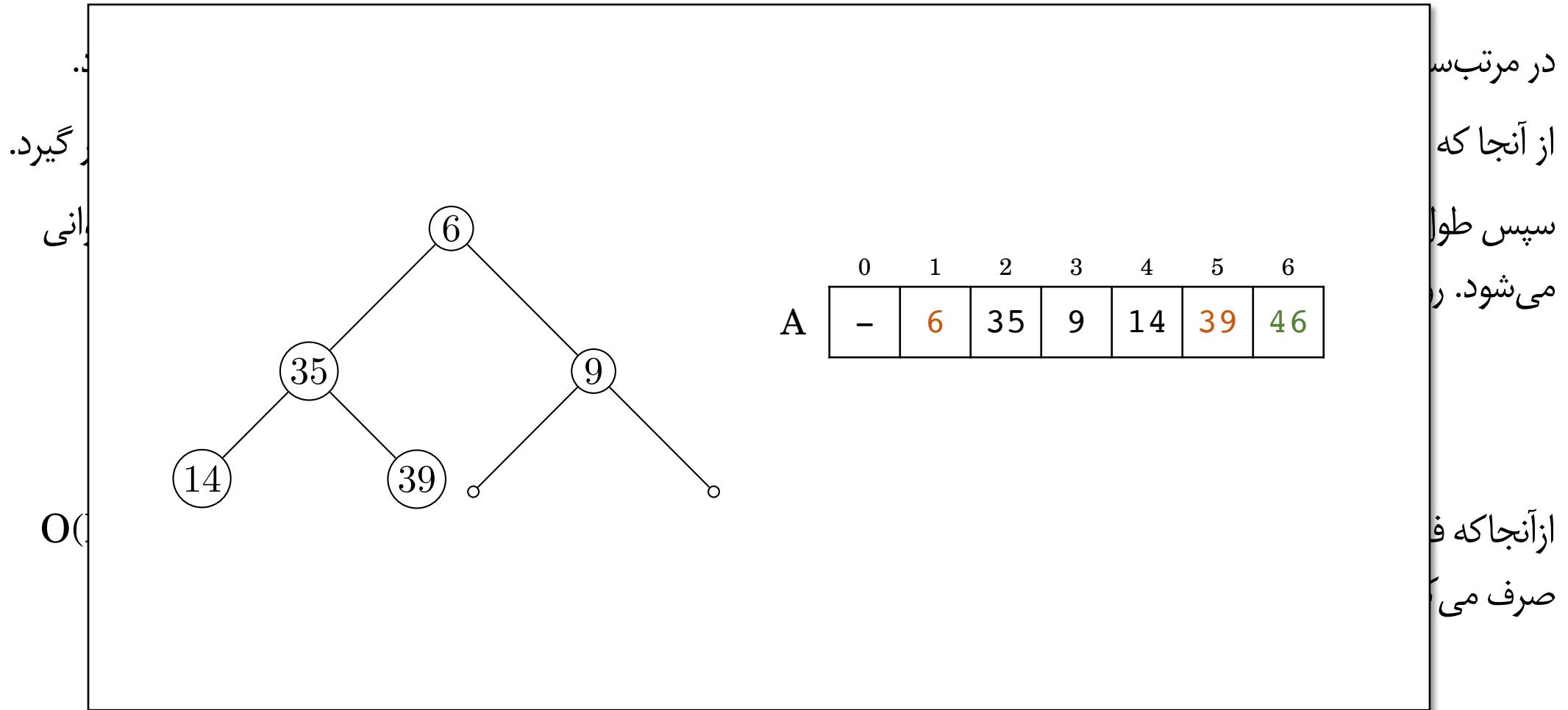
مرتب سازی هرم



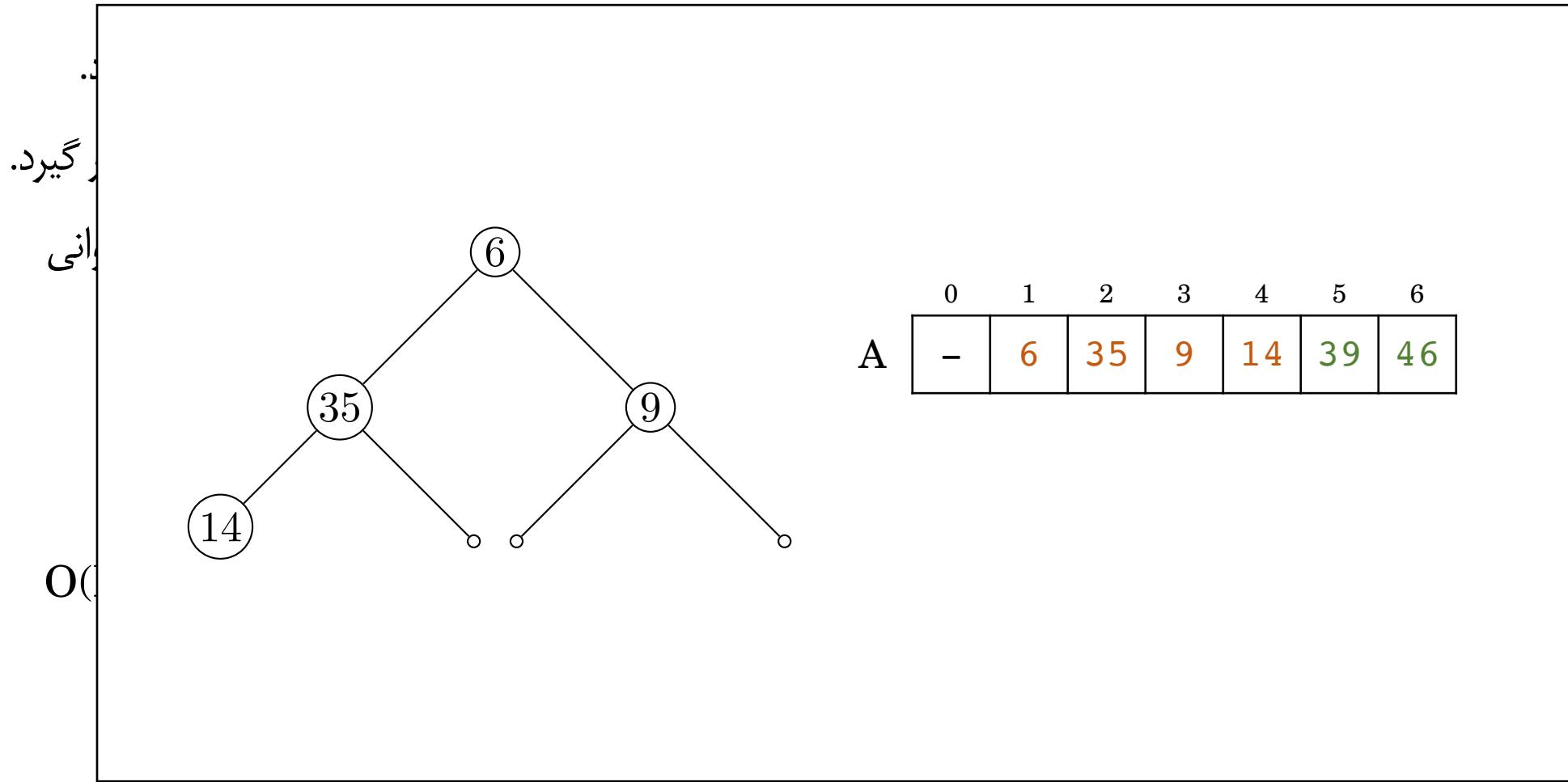
مرتب سازی هرم



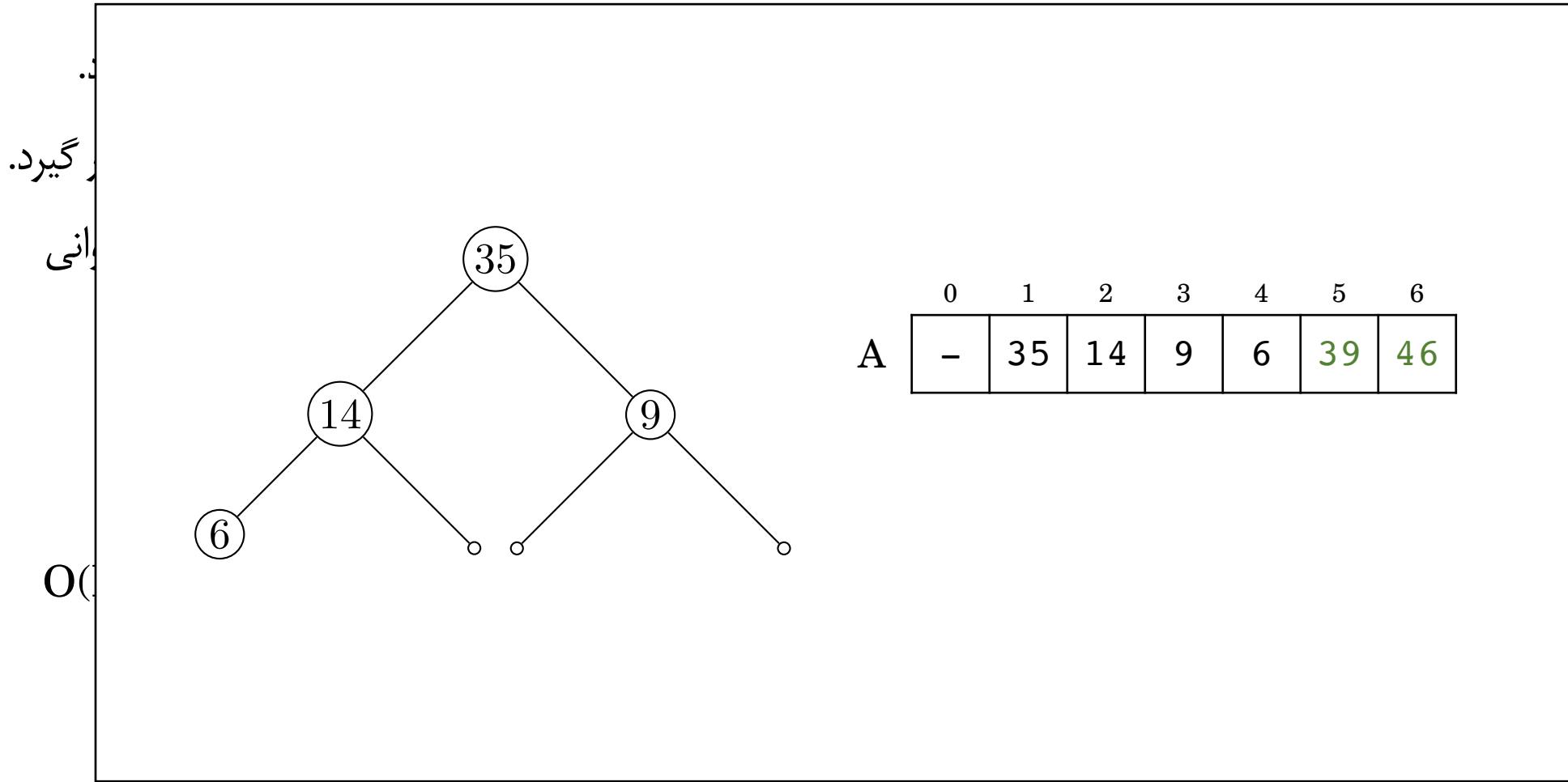
مرتب سازی هرم



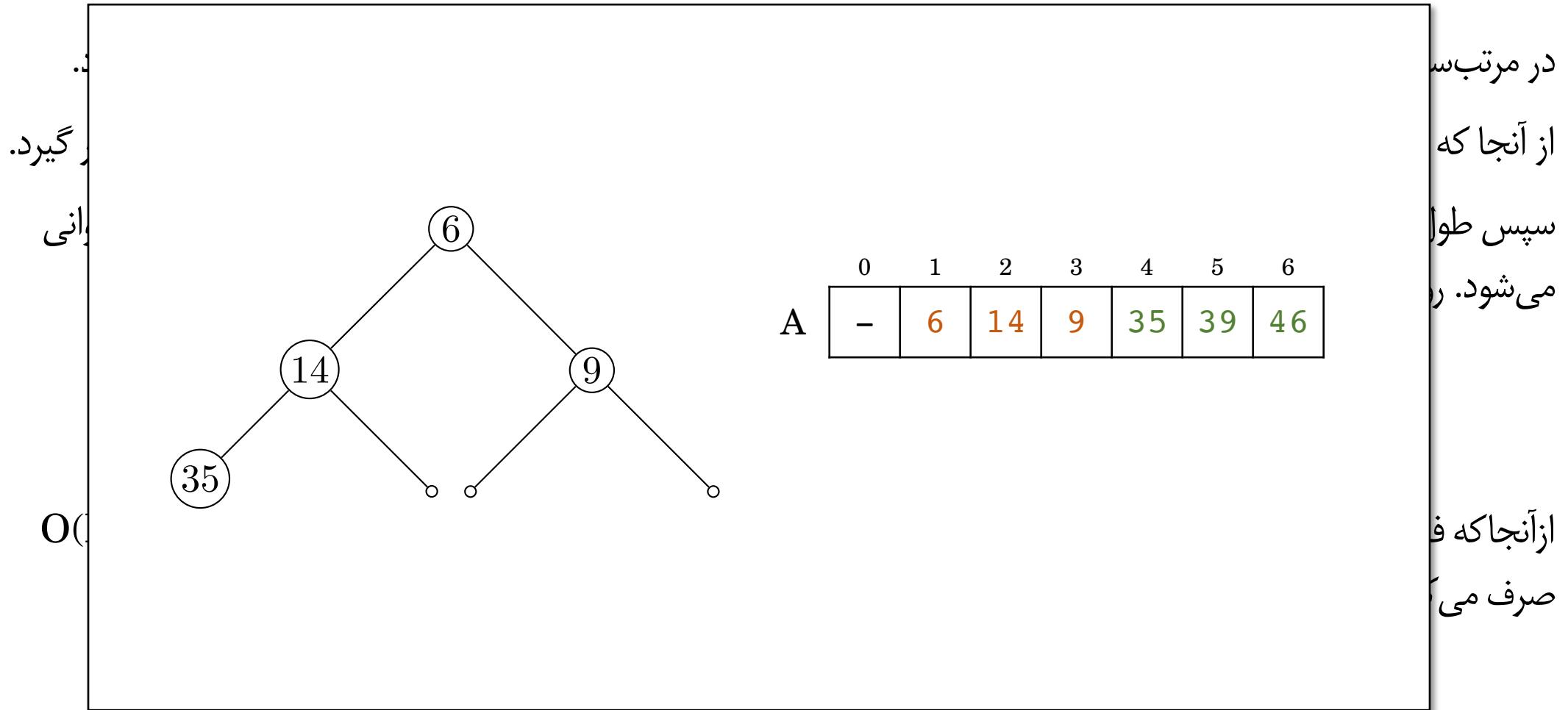
مرتب سازی هرم



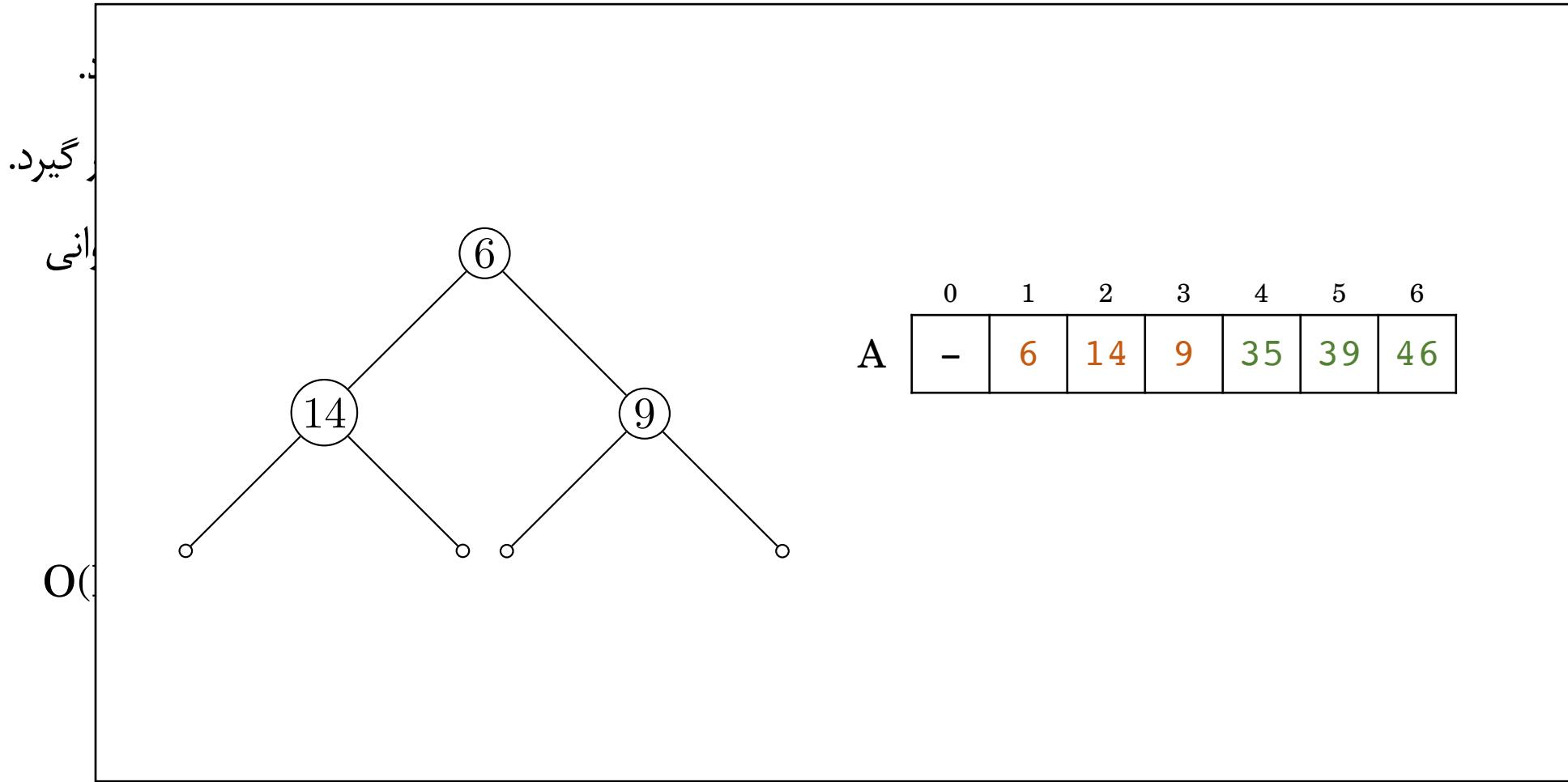
مرتب سازی هرم



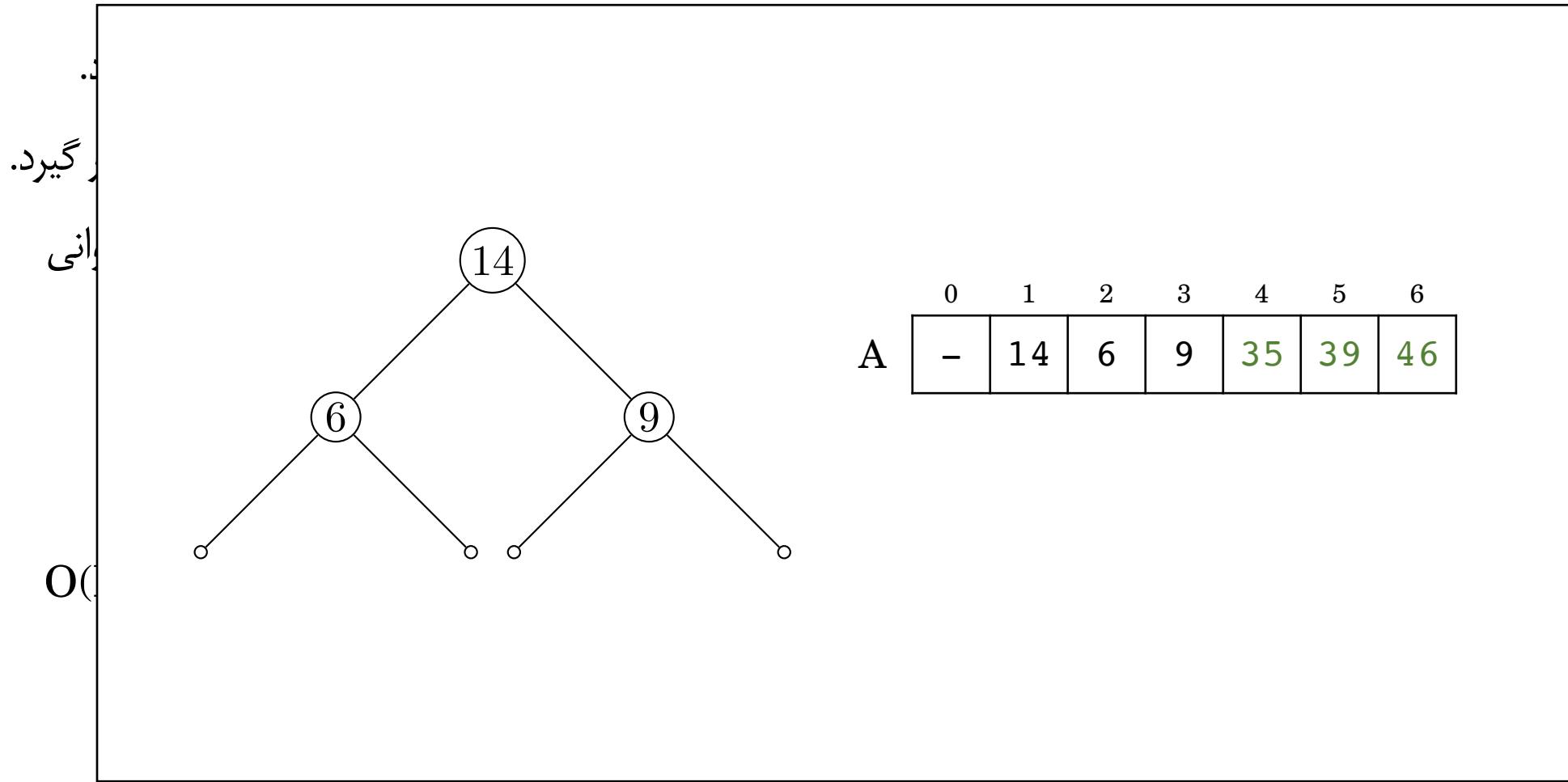
مرتب سازی هرم



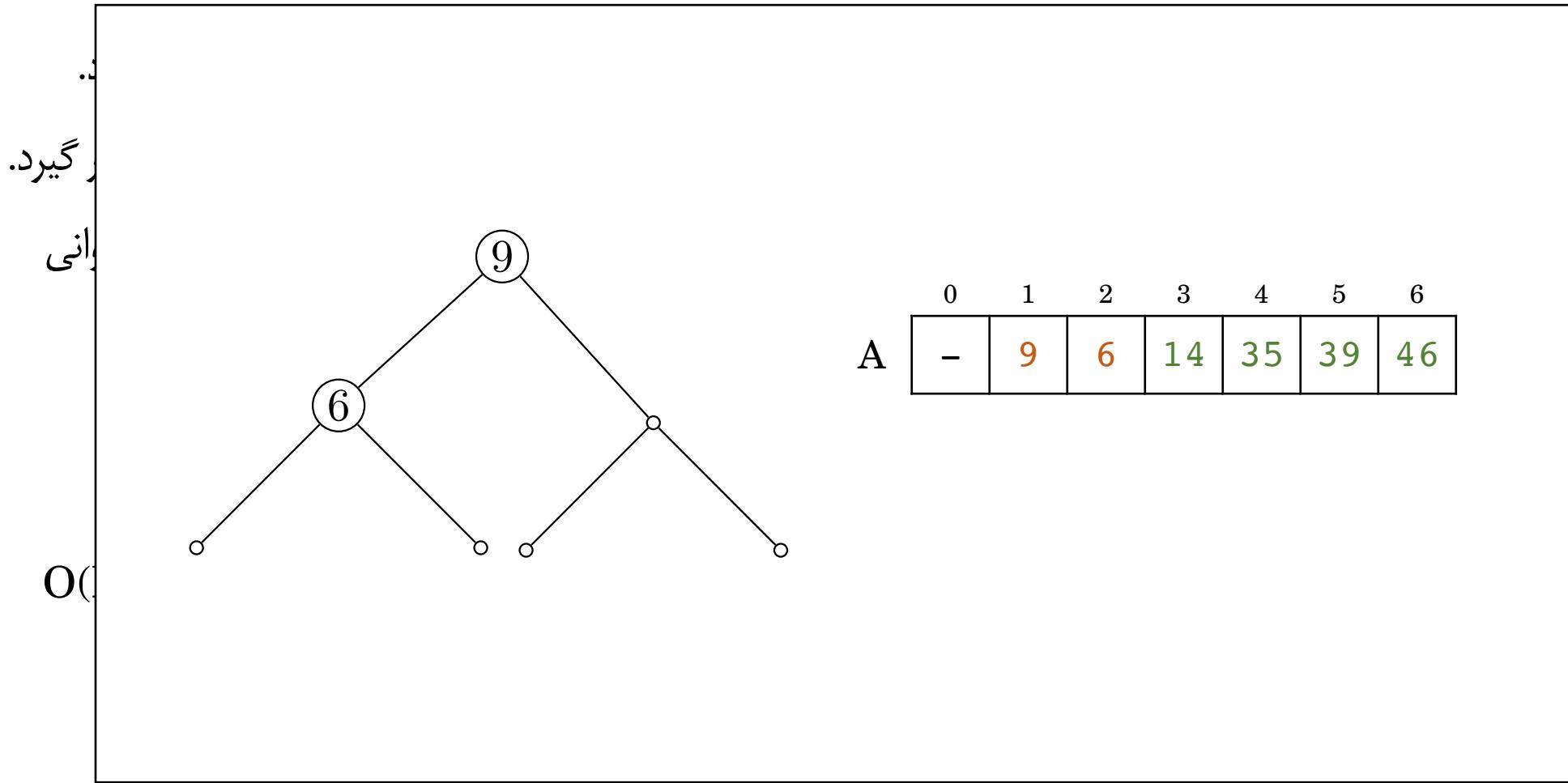
مرتب سازی هرم



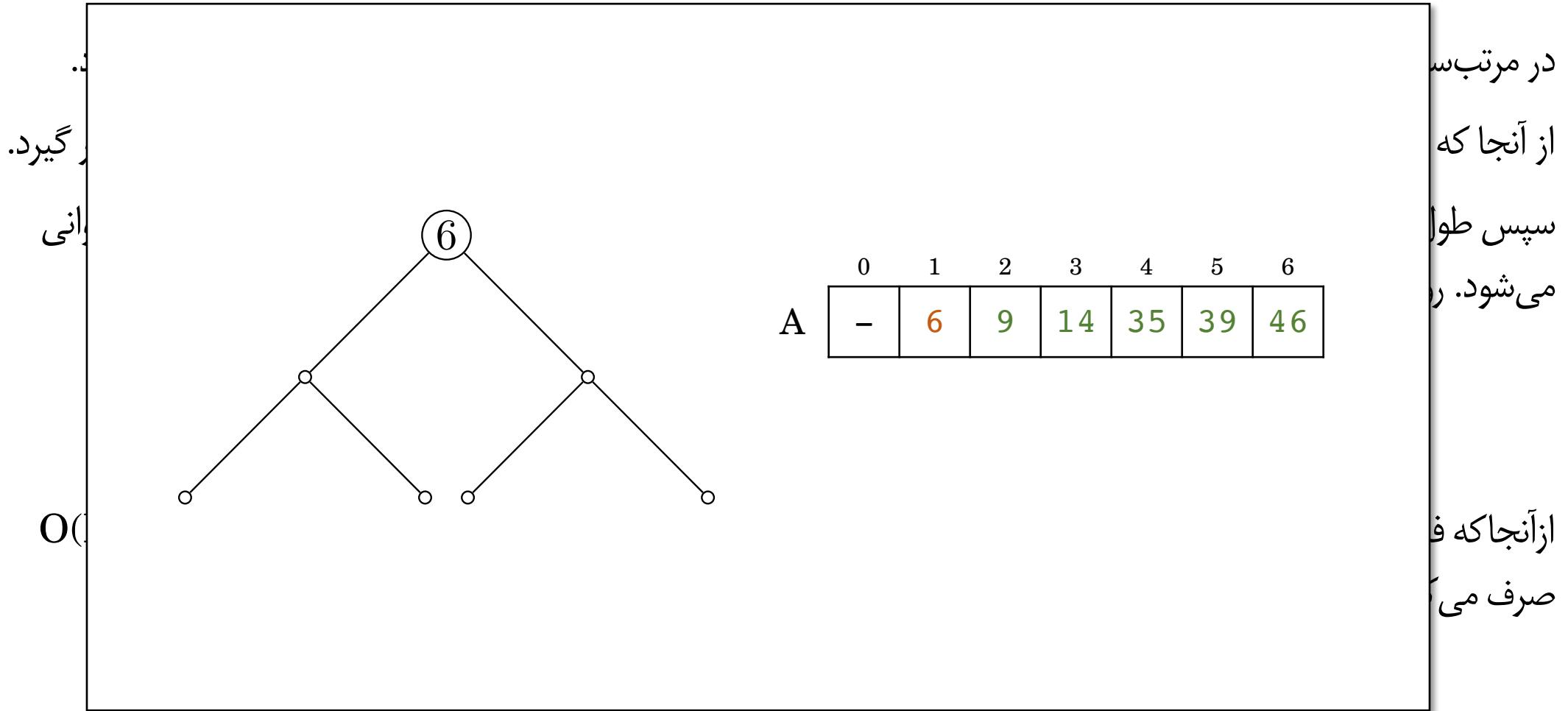
مرتب سازی هرم



مرتب سازی هرم



مرتب سازی هرم



مرتب سازی هرم

