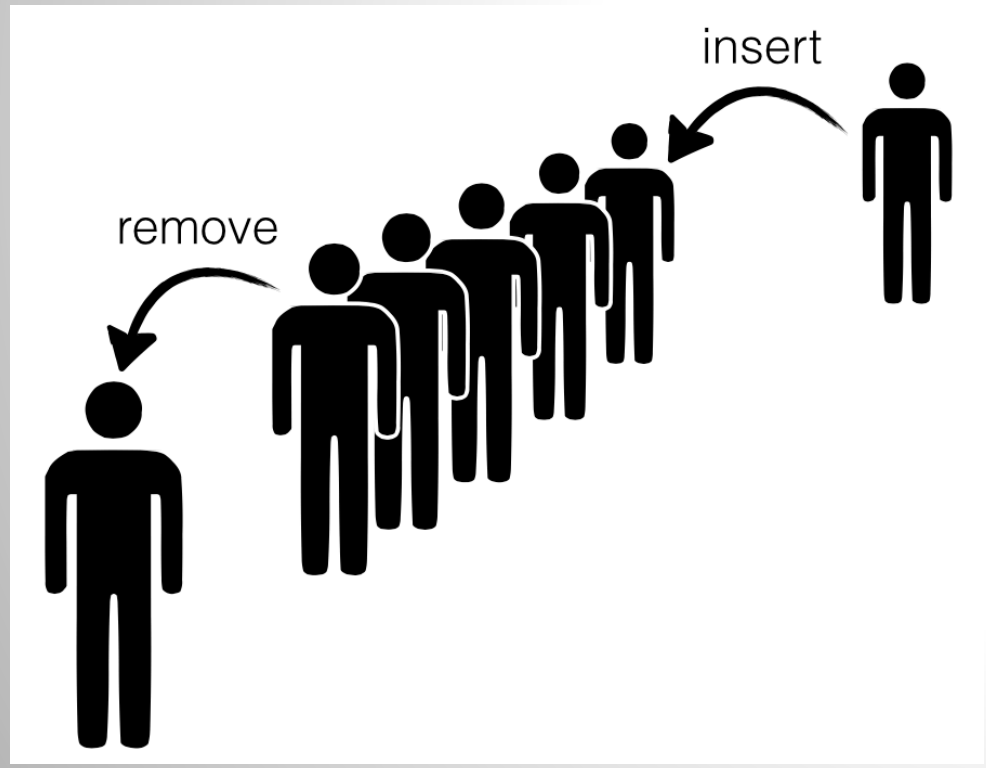
هرم

Heap

صف

- نوعی لیست است که تنها امکان اضافه کردن از انتها و حذف کردن از سر لیست را به ما می‌دهد.



- مدل First-In-First-Out

اعمالی که بر روی صف انجام می‌دهیم

- $Enqueue(Q, x)$ در انتهای صف x درج عنصر :
- $Dequeue(Q)$ حذف و بازگرداندن عنصر از ابتدای صف :
- $Front(Q)$ برگرداندن عنصر ابتدای صف بدون حذف آن :

- $isEmpty(Q)$ آیا Q خالی است :
- $Size(Q)$ برگرداندن اندازه : Q

اعمالی که بر روی صف انجام می‌دهیم

- $Enqueue(Q, x)$ در انتهای صف x درج عنصر :
- $Dequeue(Q)$ حذف و بازگرداندن عنصر از ابتدای صف :
- $Front(Q)$ برگرداندن عنصر ابتدای صف بدون حذف آن :

- $isEmpty(Q)$ آیا Q خالی است :
- $Size(Q)$ برگرداندن اندازه : Q

اگر عنصر بیشینه(کمینه) را بخواهیم ...

صف اولویت (Priority Queue)

- داده ساختاریست مشابه صف(پشته) که به هر عنصر x_i یک متغیر p_i به عنوان اولویت نسبت داده شده است که پردازش روی عنصر با بالاترین اولویت انجام می‌شود.

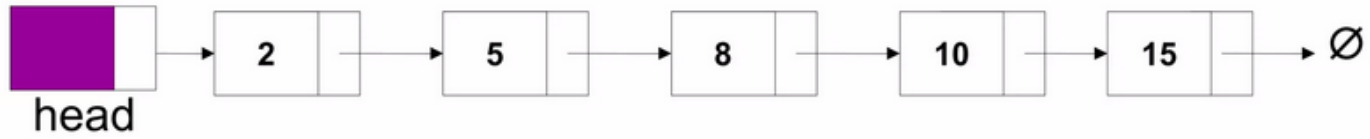
صف اولویت (Priority Queue)

- داده ساختاریست مشابه صف(پشته) که به هر عنصر x_i یک متغیر p_i به عنوان اولویت نسبت داده شده است که پردازش روی عنصر با بالاترین اولویت انجام می‌شود.

- اعمال روی صف اولویت
  - $insert(S, x, p)$ درج عنصر x با اولویت p در S
  - $max(S)$ برگرداندن عنصر با بالاترین اولویت در S
  - $extract\_max(S)$ برگرداندن و حذف عنصر با بالاترین اولویت در S

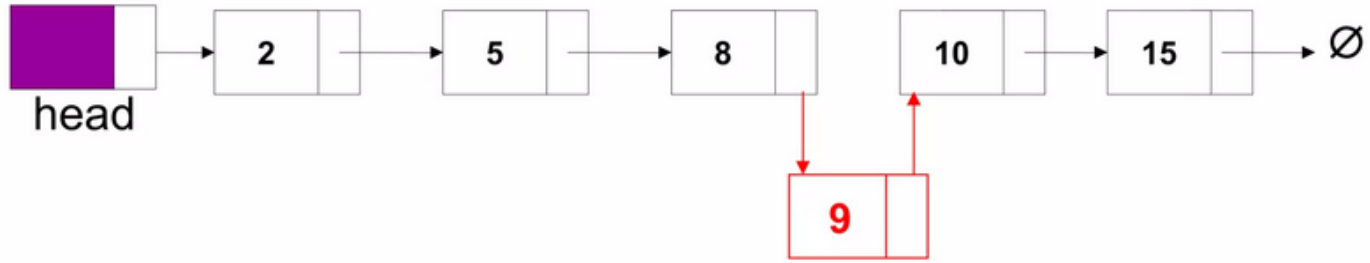
صف اولویت کمینه

- پیاده سازی با استفاده از لیست مرتب شده (از کم به زیاد)



- هزینه یافتن کمینه :؟
- هزینه حذف کردن کمینه:؟

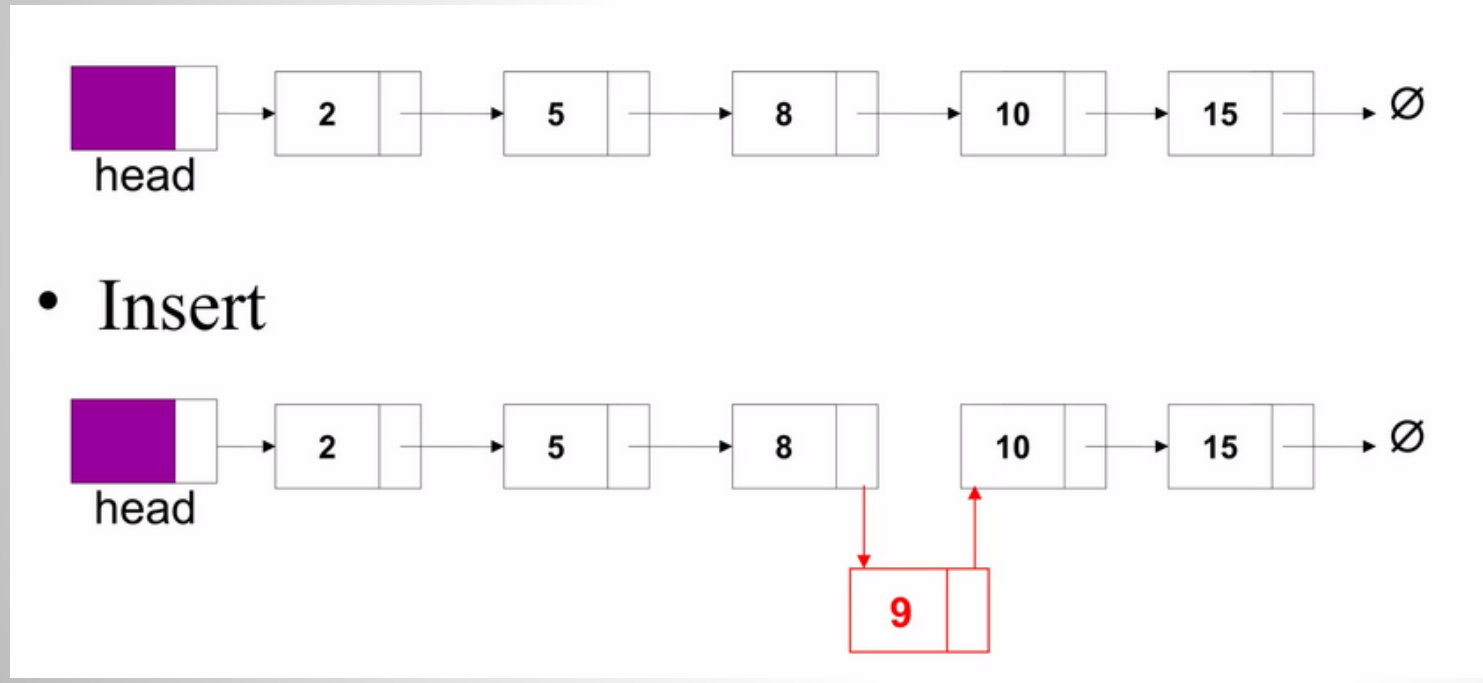
- Insert



- هزینه اضافه کردن:؟

صف اولویت کمینه

- پیاده سازی با استفاده از لیست مرتب شده



- هزینه خواندن: $O(1)$
- هزینه حذف کردن: O(1)
- هزینه اضافه کردن: $O(n)$

هرم (heap)

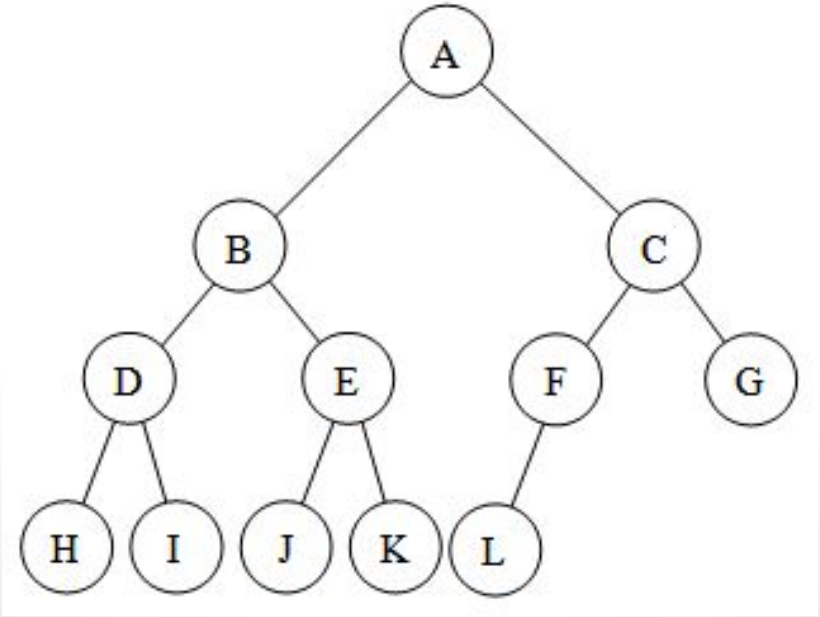
- هرم یکی از بهینه‌ترین پیاده سازی‌های صف اولویت می‌باشد، به طوری که معمولا دو کلمه "هرم" و "صف اولویت" به جای هم استفاده می‌شوند

- بهترین پیاده سازی هرم توسط یک **درخت دودویی کامل** است.

برای پیاده سازی توسط آرایه

تعریف‌ها

- **درخت kتایی کامل:** درختی که همه گره‌ها بجز عمق آخر کاملا پر هستند و در عمق آخر از چپ به راست پر شده اند.



هرم (heap)

- هرم یکی از بهینه‌ترین پیاده سازی‌های صف اولویت می‌باشد، به طوری که معمولا دو کلمه “هرم” و “صف اولویت” به جای هم استفاده می‌شوند

- بهترین پیاده سازی هرم توسط یک درخت دودویی کامل است.

- **هرم بیشینه(کمینه):** یک درخت دودویی کامل است، که مقدار هر گره بیشتر از مقدار فرزنداش باشد.

- ارتفاع هرم (درخت دودویی کامل) چقدر است؟

هرم (heap)

- هرم یکی از بهینه‌ترین پیاده سازی‌های صف اولویت می‌باشد، به طوری که معمولا دو کلمه "هرم" و "صف اولویت" به جای هم استفاده می‌شوند

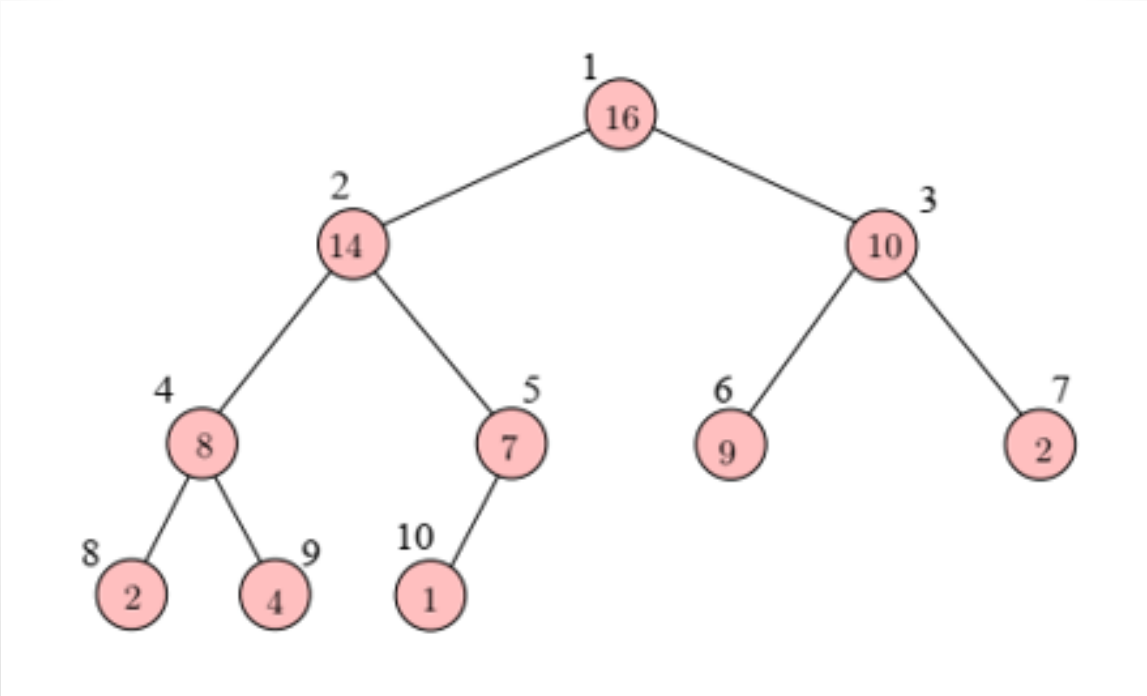
- بهترین پیاده سازی هرم توسط یک درخت دودویی کامل است.

- **هرم بیشینه(کمینه):** یک درخت دودویی کامل است، که مقدار هر گره بیشتر از مقدار فرزنداش باشد.

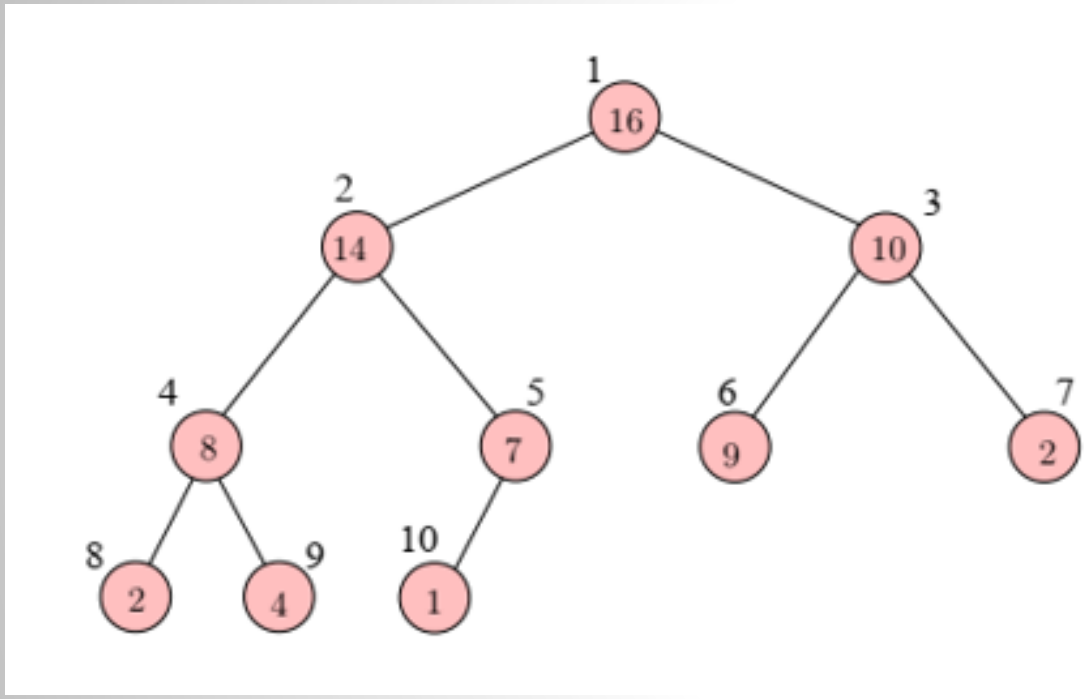
- ارتفاع هرم (درخت دودویی کامل) چقدر است؟ $\log n$

هرم (heap)

- **هرم بیشینه(کمینه):** یک درخت دودویی کامل است، که مقدار هر گره بیشتر(کمتر) از مقدار فرزنداش باشد.



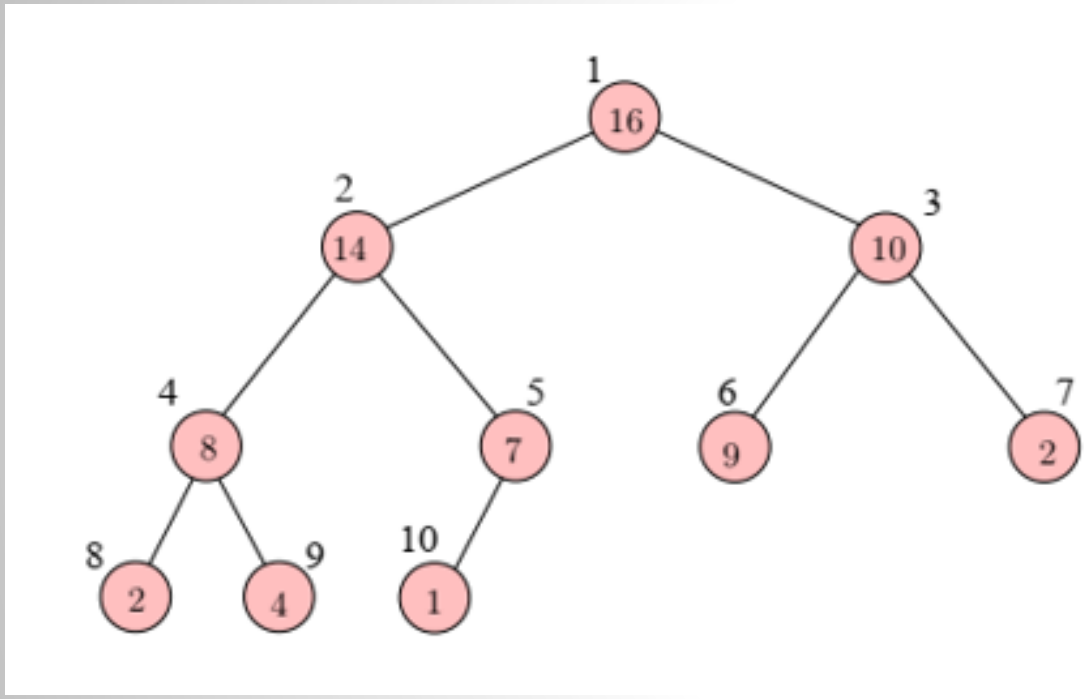
پیاده سازی



- فقط ساختار هرم مورد بحث است
  - توابع روی هرم در بخش بعد بحث می‌شود

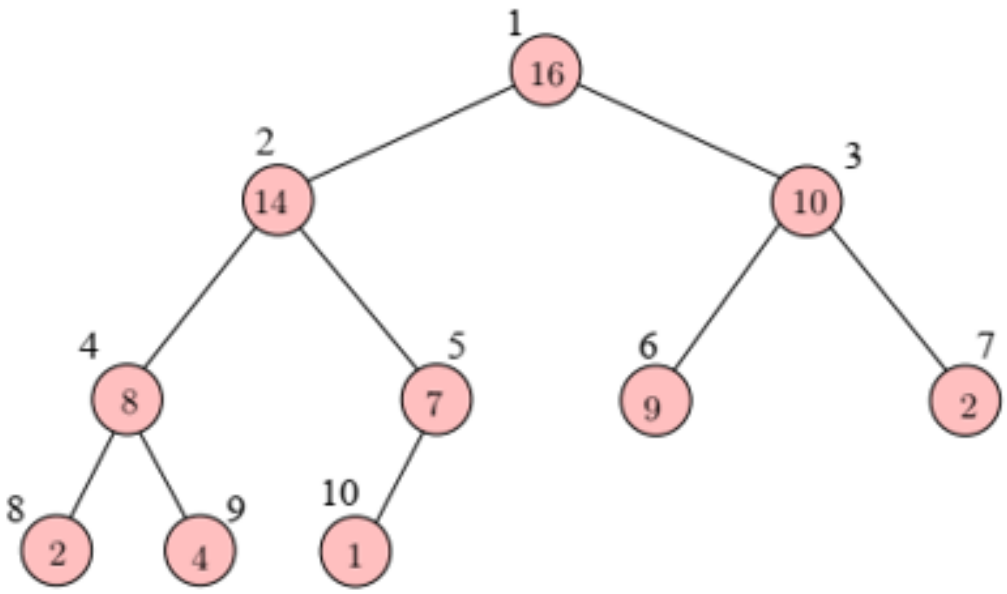
- با استفاده از اشاره‌گر:

پیاده سازی



- فقط ساختار هرم مورد بحث است
  - توابع روی هرم در بخش بعد بحث می‌شود
- با استفاده از اشاره‌گر: معادل درخت

پیاده سازی



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

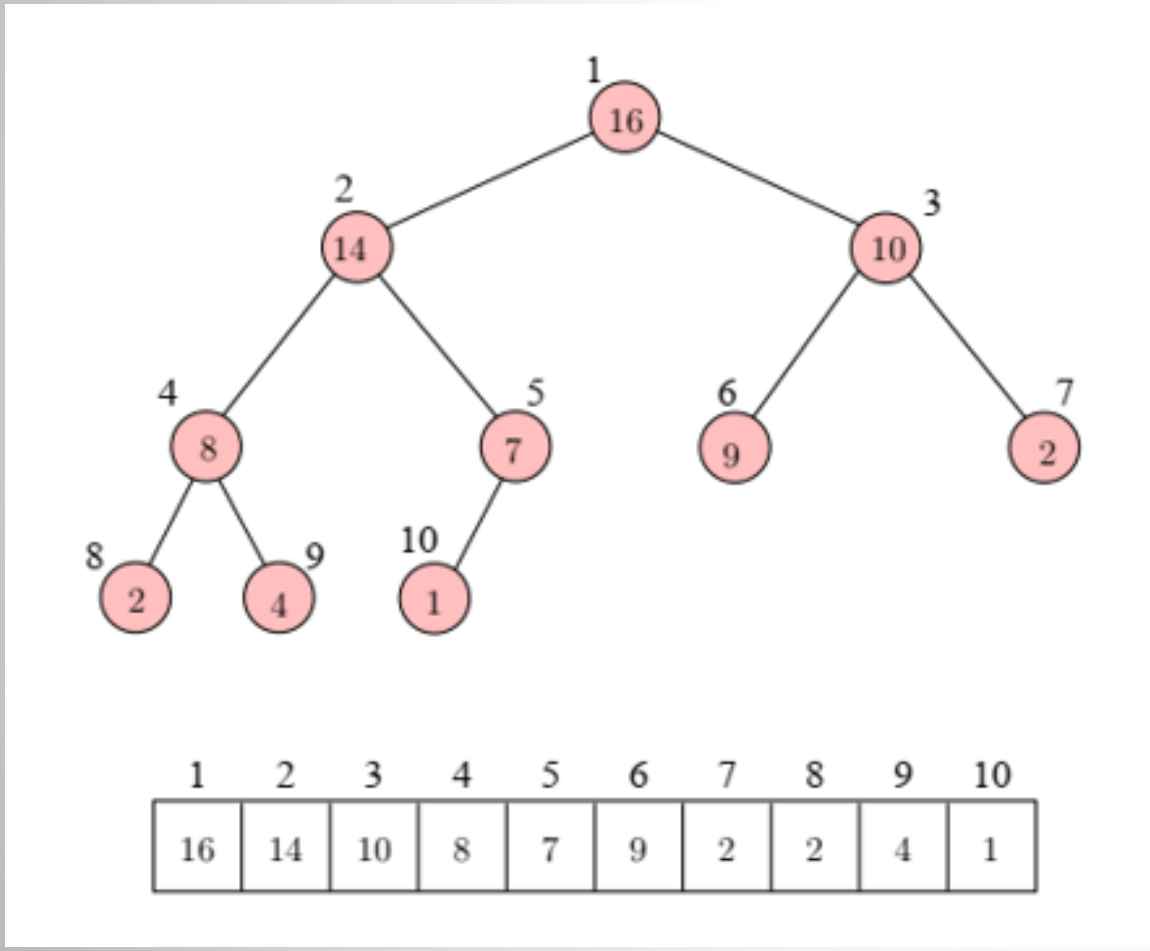
- فقط ساختار هرم مورد بحث است
  - توابع روی هرم در بخش بعد بحث می‌شود

- با استفاده از آرایه $A[1 \ldots n]$:
  - ریشه در $A[1]$

  - فرزند چپ گره i در ؟
  - فرزند راست گره i در ؟

  - پدر گره i در ؟

پیاده سازی



- فقط ساختار هرم مورد بحث است
  - توابع روی هرم در بخش بعد بحث می‌شود

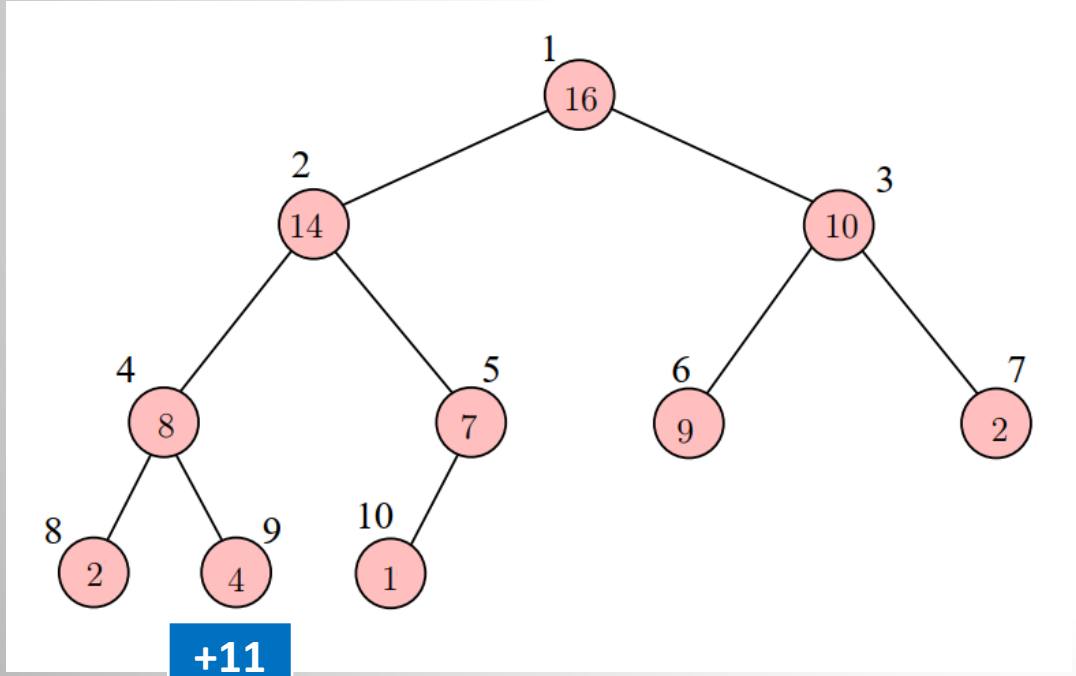
- با استفاده از آرایه $A[1 \ldots n]$:
  - ریشه در $A[1]$

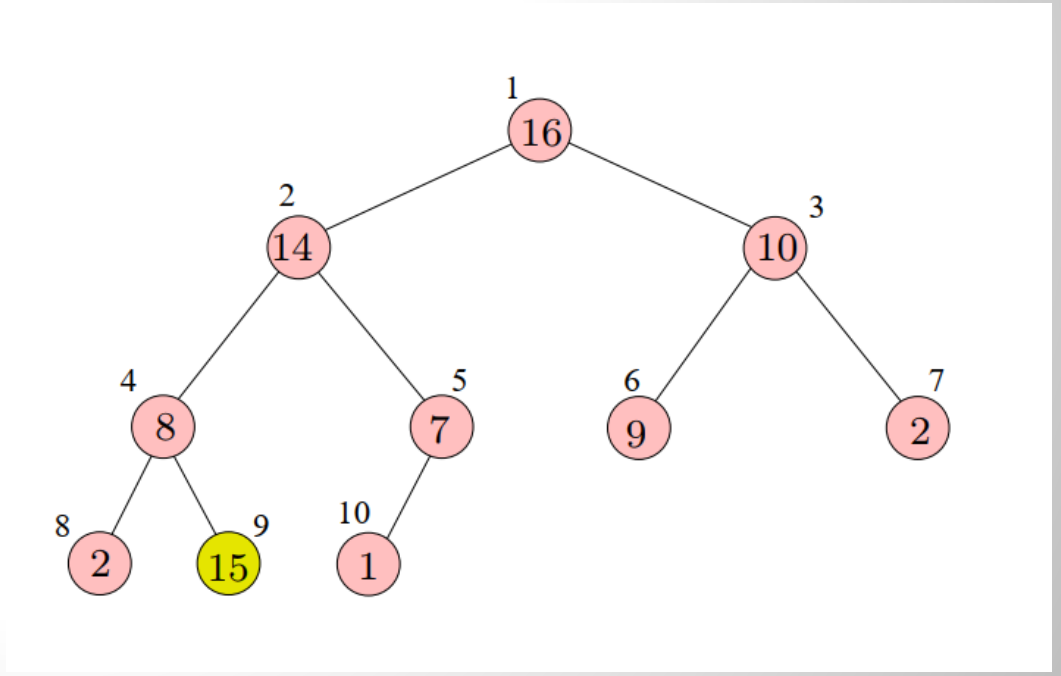
  - فرزند چپ گره i در $A[2i]$
  - فرزند راست گره i در $A[2i + 1]$

  - پدر گره i در $A[\lfloor \frac{i}{2} \rfloor]$

اضافه کردن مقدار یک گره

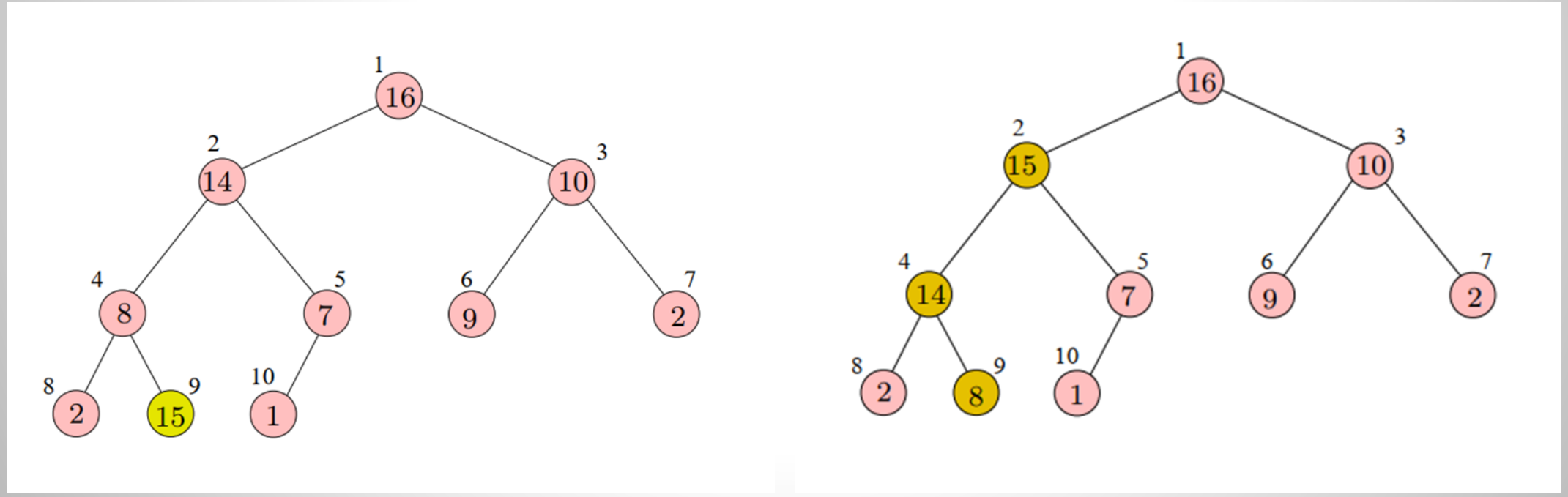
- فرض کنیم مقدار ۴ را ۱۱ واحد اضافه کنیم
- روش ؟



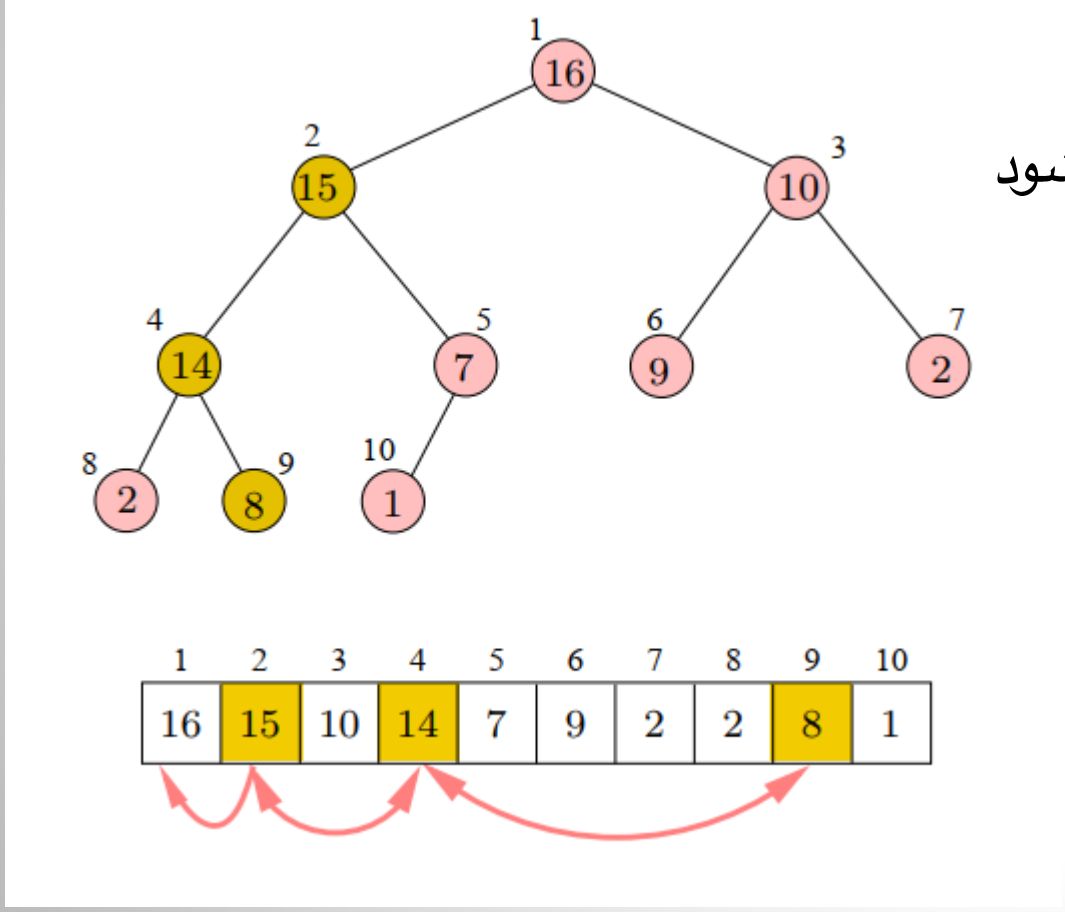


اضافه کردن مقدار یک گره

- گره تغییر یافته در هر مرحله:
  - اگر از پدرش بزرگتر باشد با آن جابه‌جا می‌شود



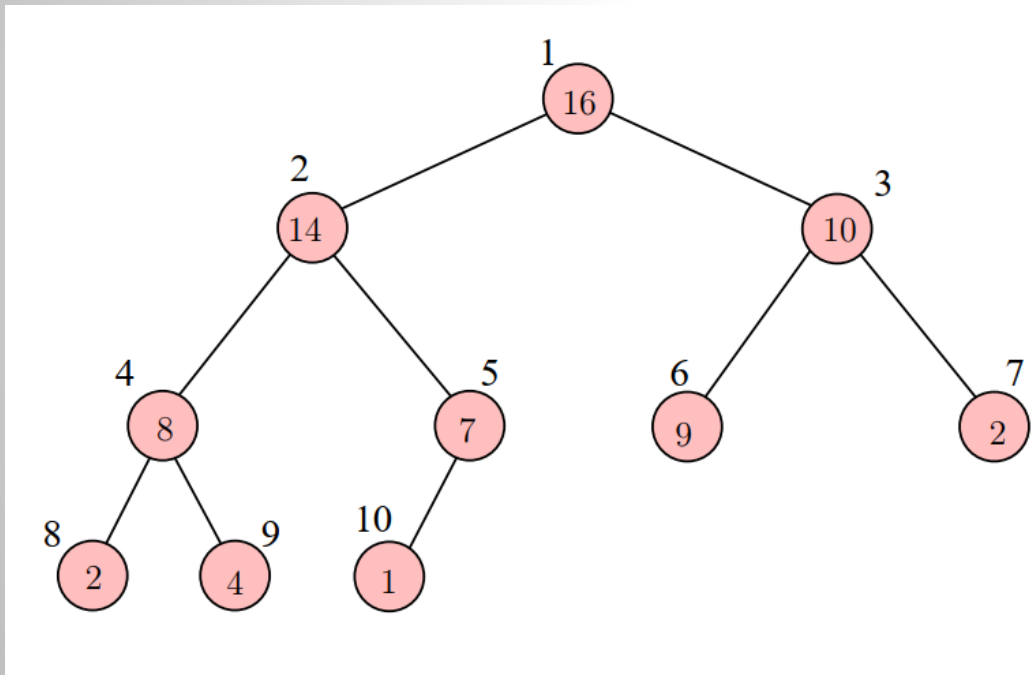
اضافه کردن مقدار یک گره



- گره تغییر یافته در هر مرحله:
  - اگر از پدرش بزرگتر باشد با آن جابه‌جا می‌شود
- چرا؟
  - آیا از فرزندان راست پدرش هم بزرگتر است؟
  - آیا پدر از نوه‌های خود نیز بزرگتر است؟
  - ...

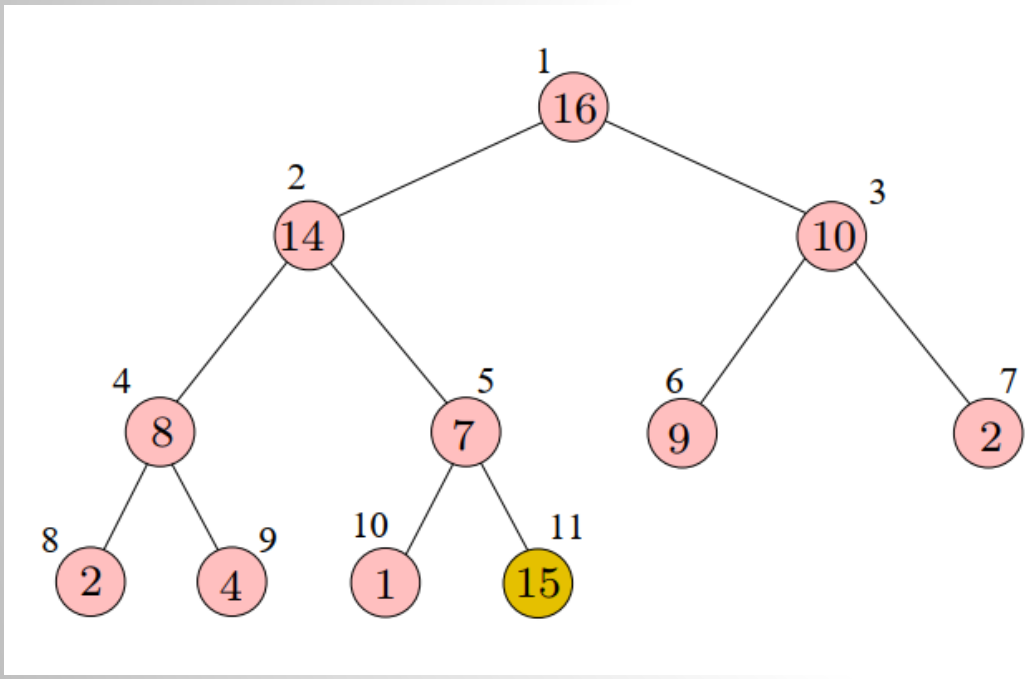
افزودن یک گره جدید

- فرض کنیم یک گره جدید با مقدار ۱۵ را می‌خواهیم به هرم زیر اضافه کنیم.

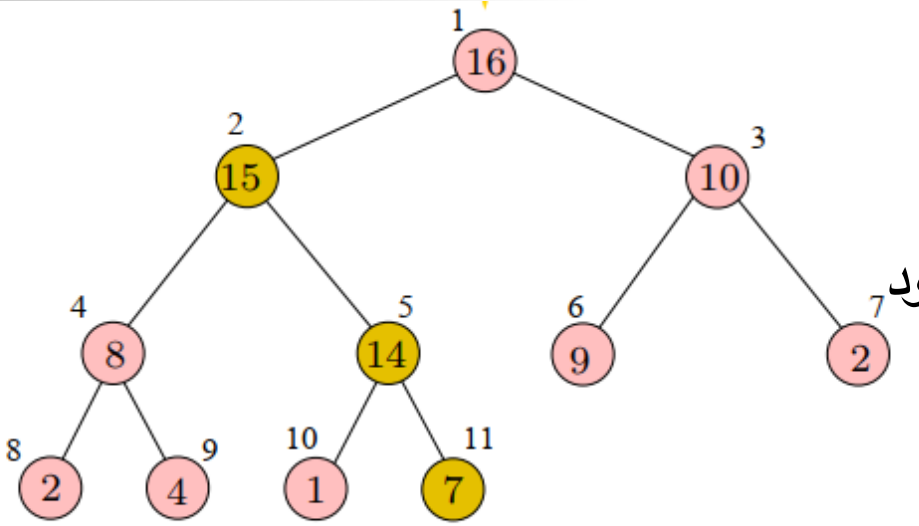


افزودن یک گره جدید

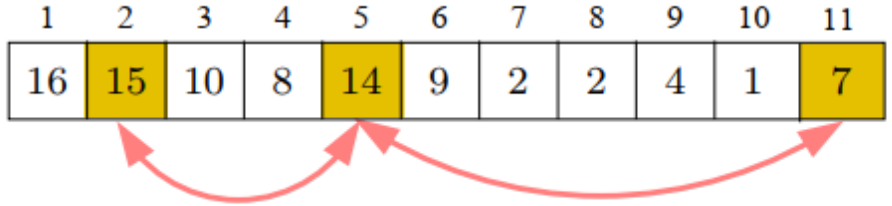
- گره به ته درخت اضافه شده:



افزودن یک گره جدید



- گره به ته درخت اضافه شده:
- گره تغییر یافته در هم مرحله:
  - اگر از پدرش بزرگتر باشد با آن جابه‌جا می‌شود

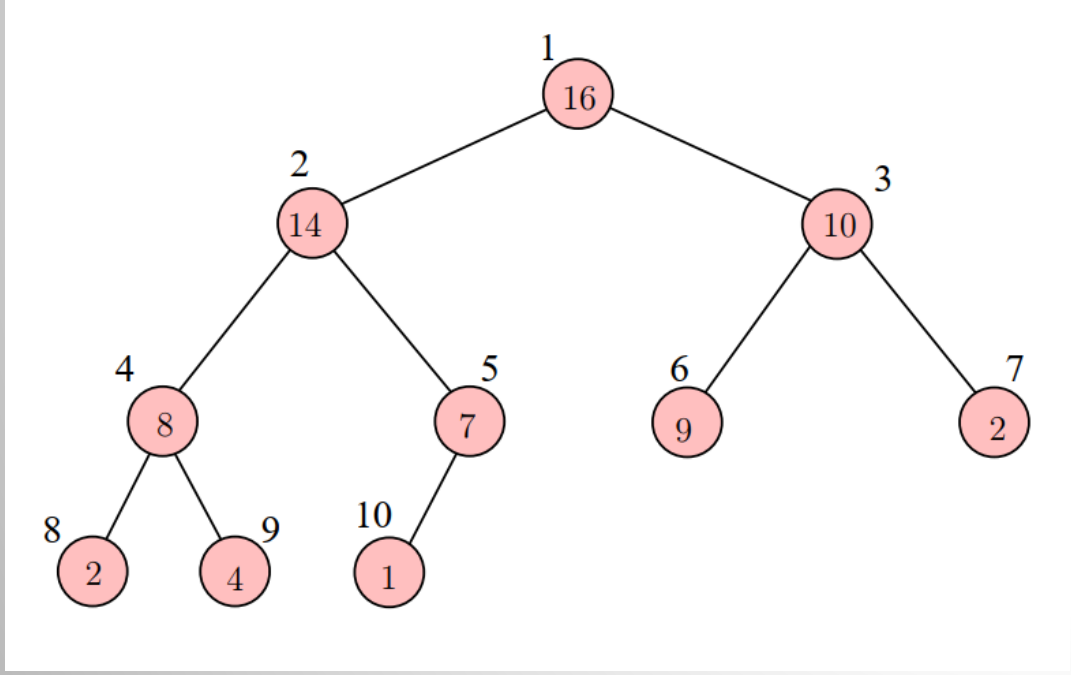


- انگار گره $\infty-$ به ته آرایه اضافه شده
  - حال مقدار آن را افزایش می‌دهیم.

حذف بزرگ‌ترین عنصر

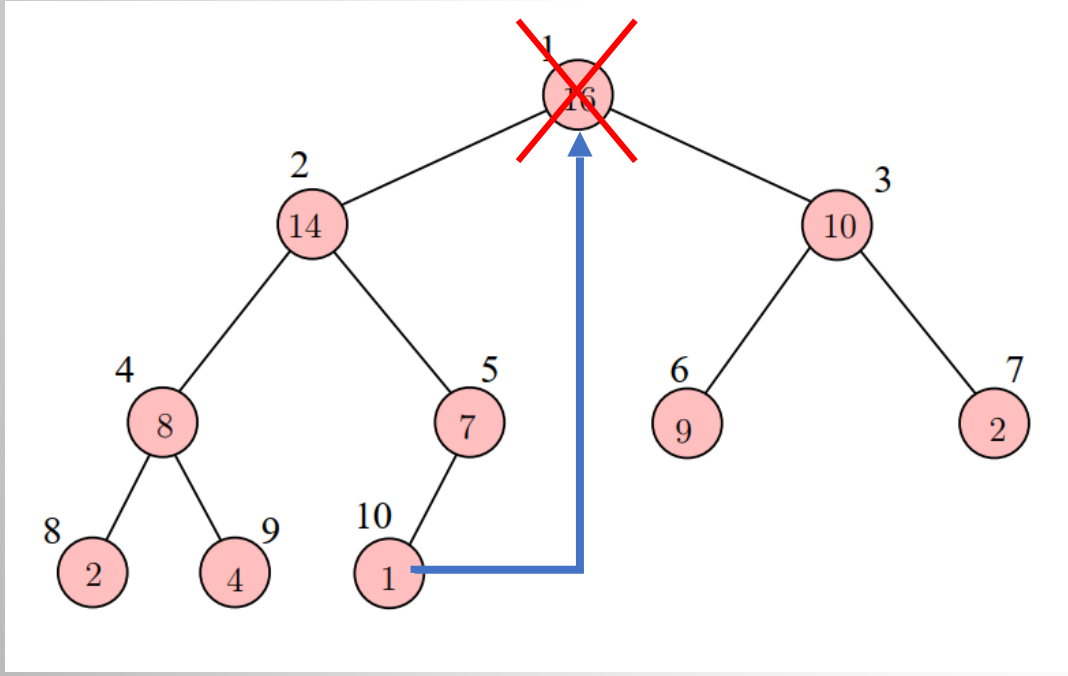
- فرض کنیم بزرگ‌ترین گره (گره ۱۶) را می‌خواهیم حذف کنیم.

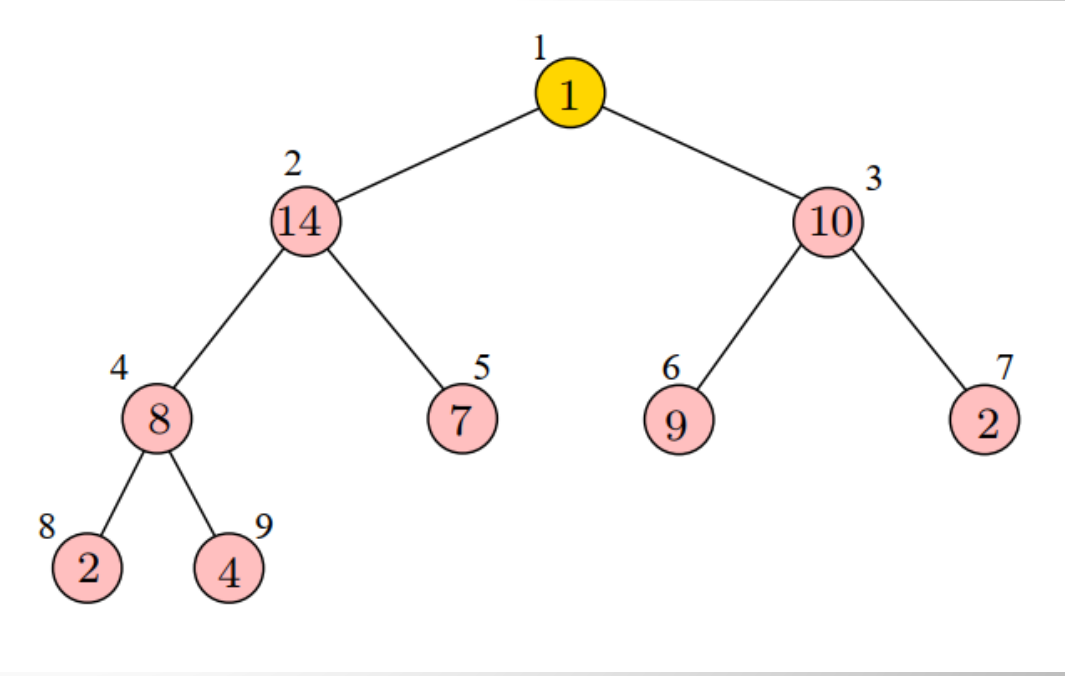
- روش ؟



حذف بزرگ‌ترین عنصر

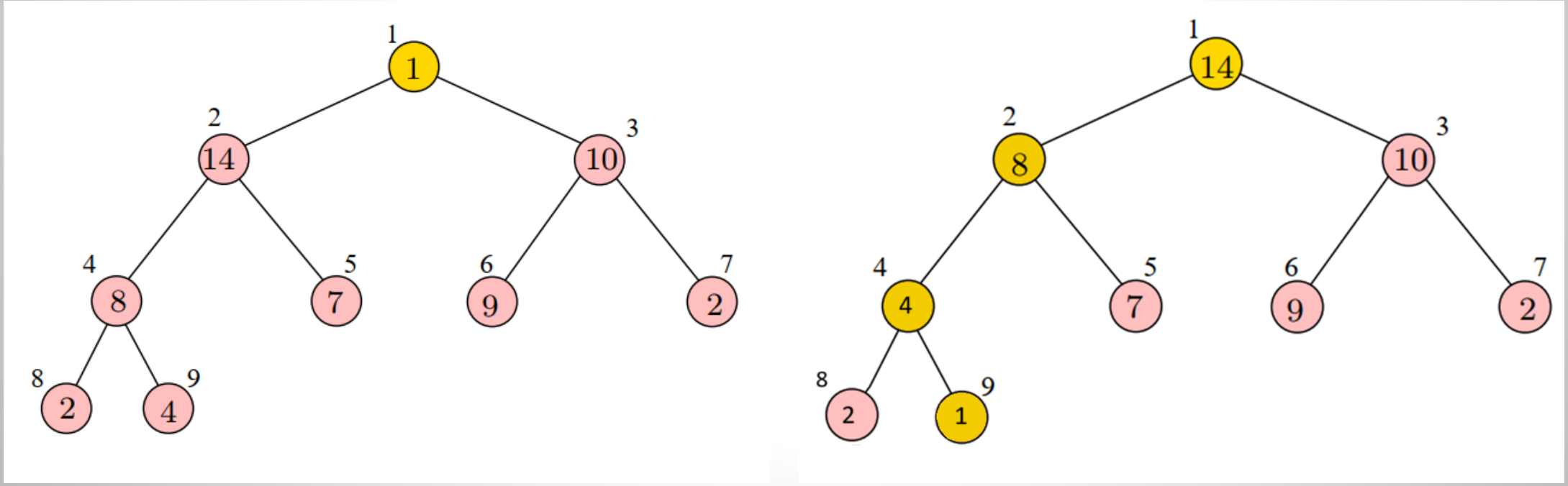
- عنصر انتهای درخت به محل ریشه می‌آوریم





حذف بزرگ‌ترین عنصر

- گره تغییر یافته در هم مرحله:
  - اگر از فزرندیش کوچکتر باشد، با بزرگ‌ترین فرزندش جابه‌جا می‌کنیم.



ساخت هرم روی ورودی با اندازه n

- ورودی: 12,5,11,3,10,2,9,4,8,15,7,6
- هدف ساخت هرم کمینه.

- روش ؟

ساخت هرم روی ورودی با اندازه n

- ورودی: 12,5,11,3,10,2,9,4,8,15,7,6
- هدف ساخت هرم کمینه.

- روش اول:
  - مرتب سازی با الگوریتم "مرتب سازی سریع"
  - زمان اجرا: ؟

ساخت هرم روی ورودی با اندازه n

- ورودی: 12,5,11,3,10,2,9,4,8,15,7,6
- هدف ساخت هرم کمینه.

- روش اول:
  - مرتب سازی با الگوریتم "مرتب سازی سریع"
  - زمان اجرا: $O(n\log n)$

ساخت هرم روی ورودی با اندازه n

- ورودی: 12,5,11,3,10,2,9,4,8,15,7,6
- هدف ساخت هرم کمینه.

- روش دوم:
  - یک هرم خالی در نظر گرفته
  - در هر مرحله یکی از داده‌ها را به آن اضافه کنیم
  - زمان اجرا: ؟

ساخت هرم روی ورودی با اندازه n

- ورودی: 12,5,11,3,10,2,9,4,8,15,7,6
- هدف ساخت هرم کمینه.

- روش دوم:
  - یک هرم خالی در نظر گرفته
  - در هر مرحله یکی از داده‌ها را به آن اضافه کنیم
  - زمان اجرا: $O(n\log n)$

ساخت هرم روی ورودی با اندازه n

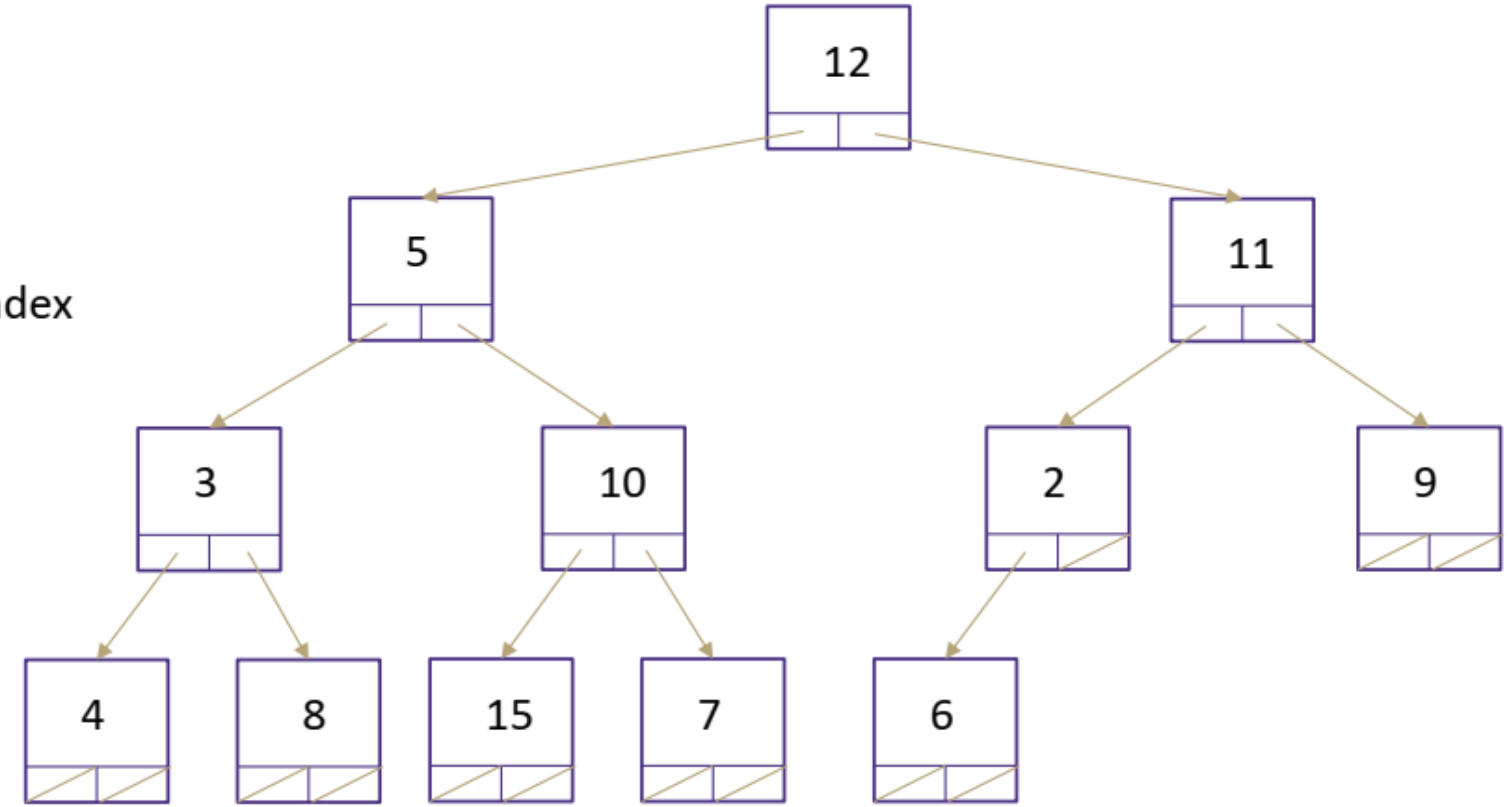
- ورودی: 12,5,11,3,10,2,9,4,8,15,7,6
- هدف ساخت هرم کمینه.

- آیا می‌توان در زمان خطی این کار را کرد؟

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index

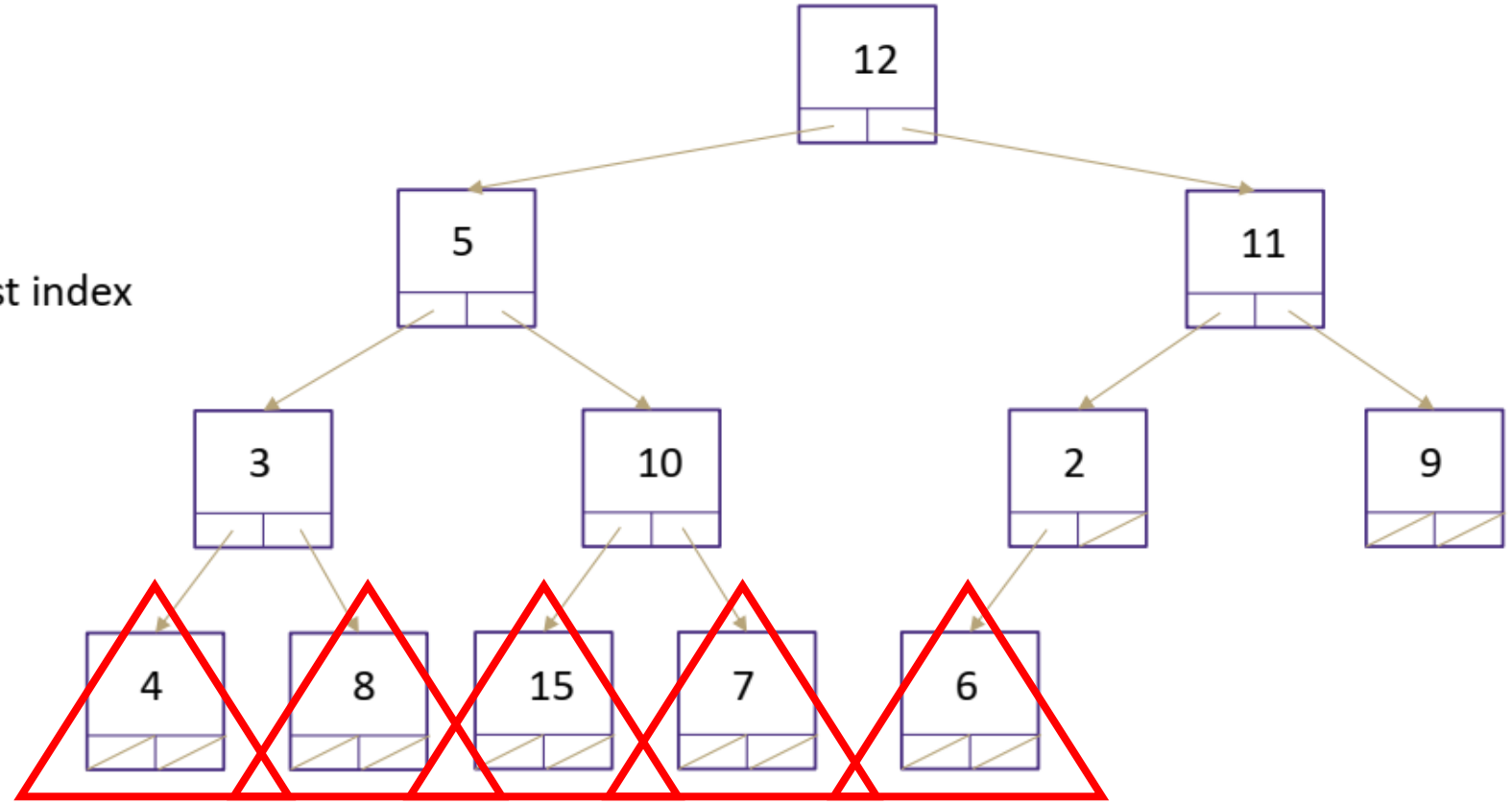


0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	10	2	9	4	8	15	7	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index



همه زیر درخت های این مرحله، یک هیپ هستند

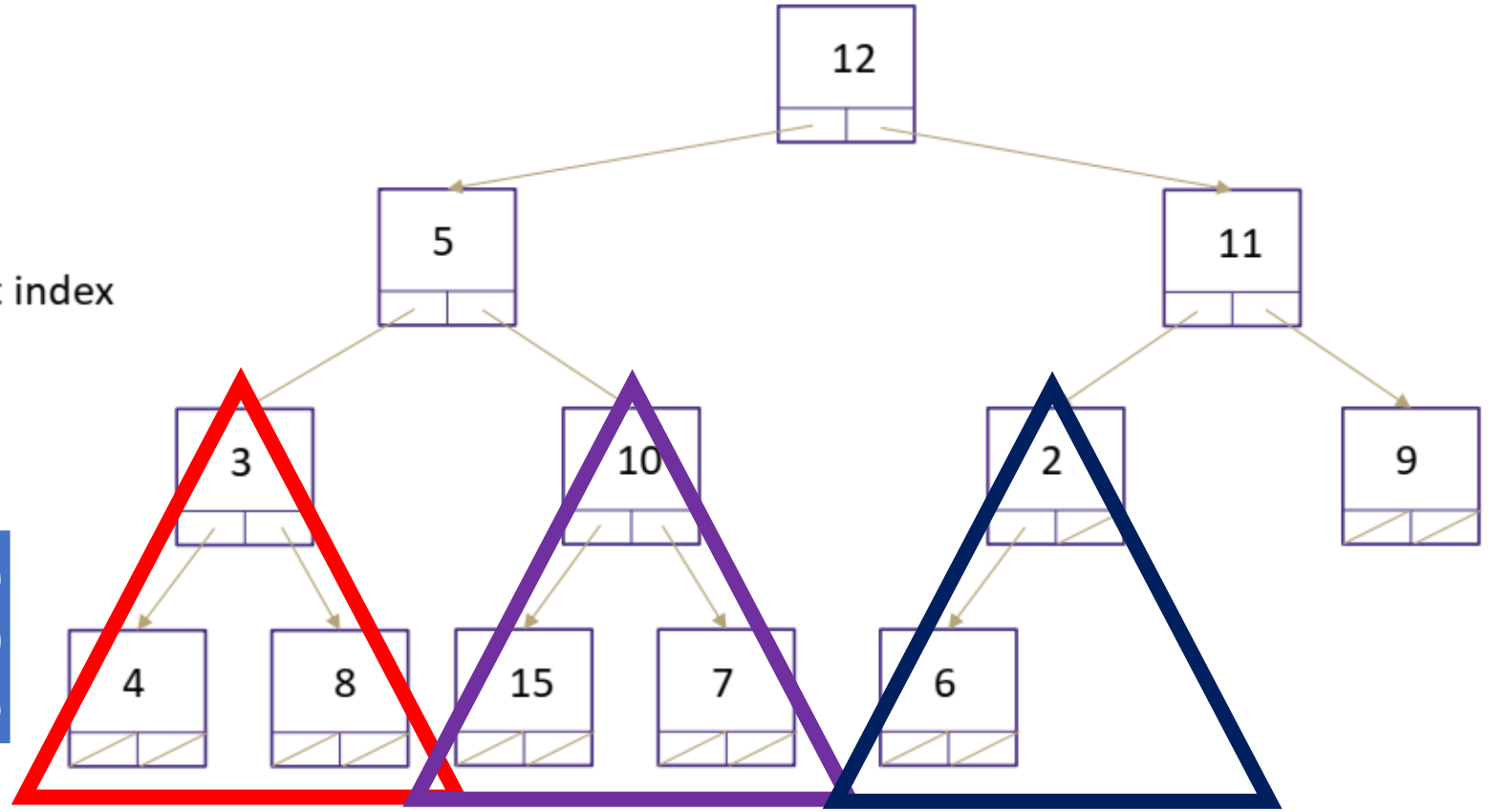
0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	10	2	9	4	8	15	7	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index

در هر مرحله ریشه زیر درخت را با کوچکترین فرزند جابه‌جا کرده تا کل زیر درخت مرحله، هیپ شود



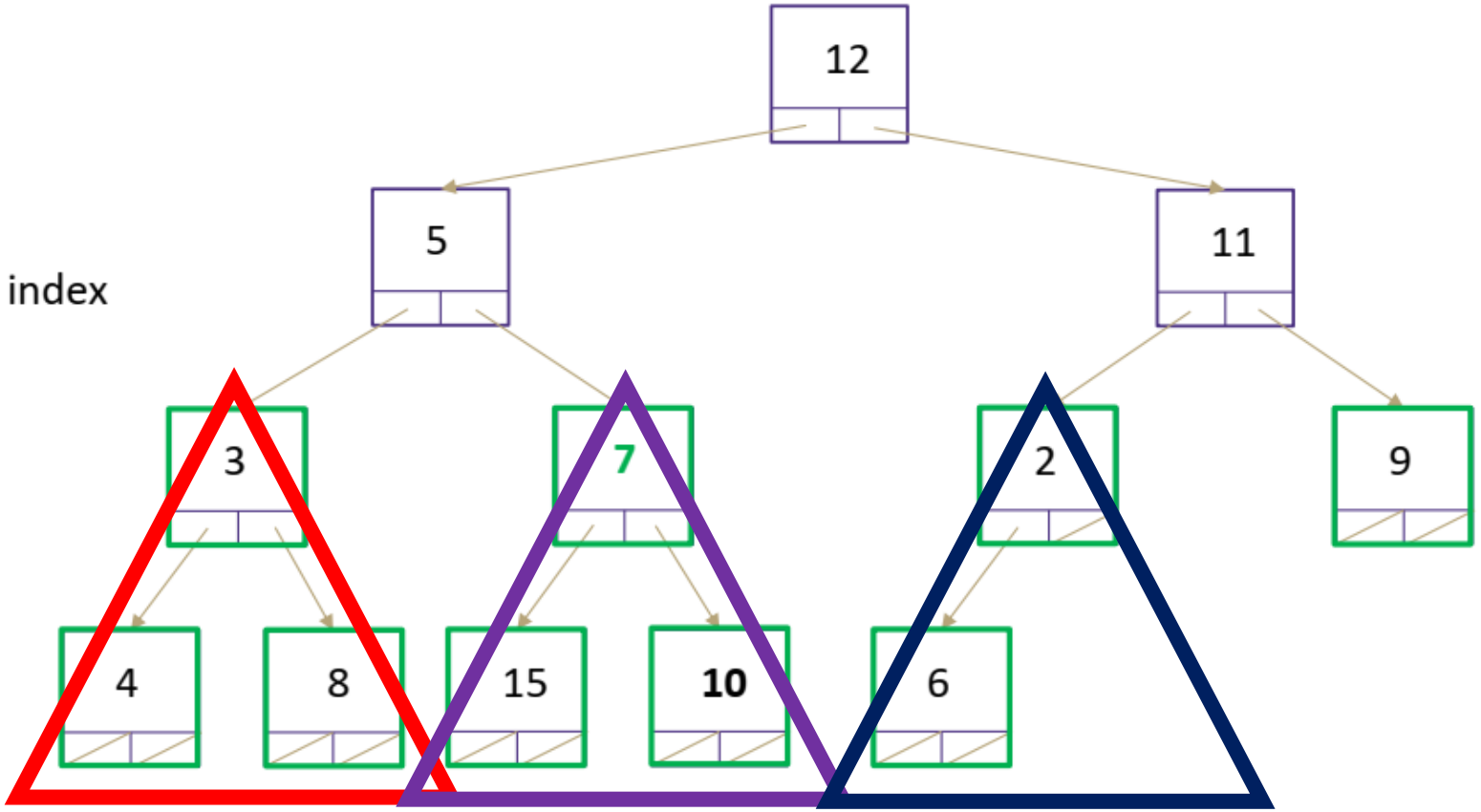
0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	10	2	9	4	8	15	7	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3

همه زیر درخت های این مرحله، یک هیپ هستند



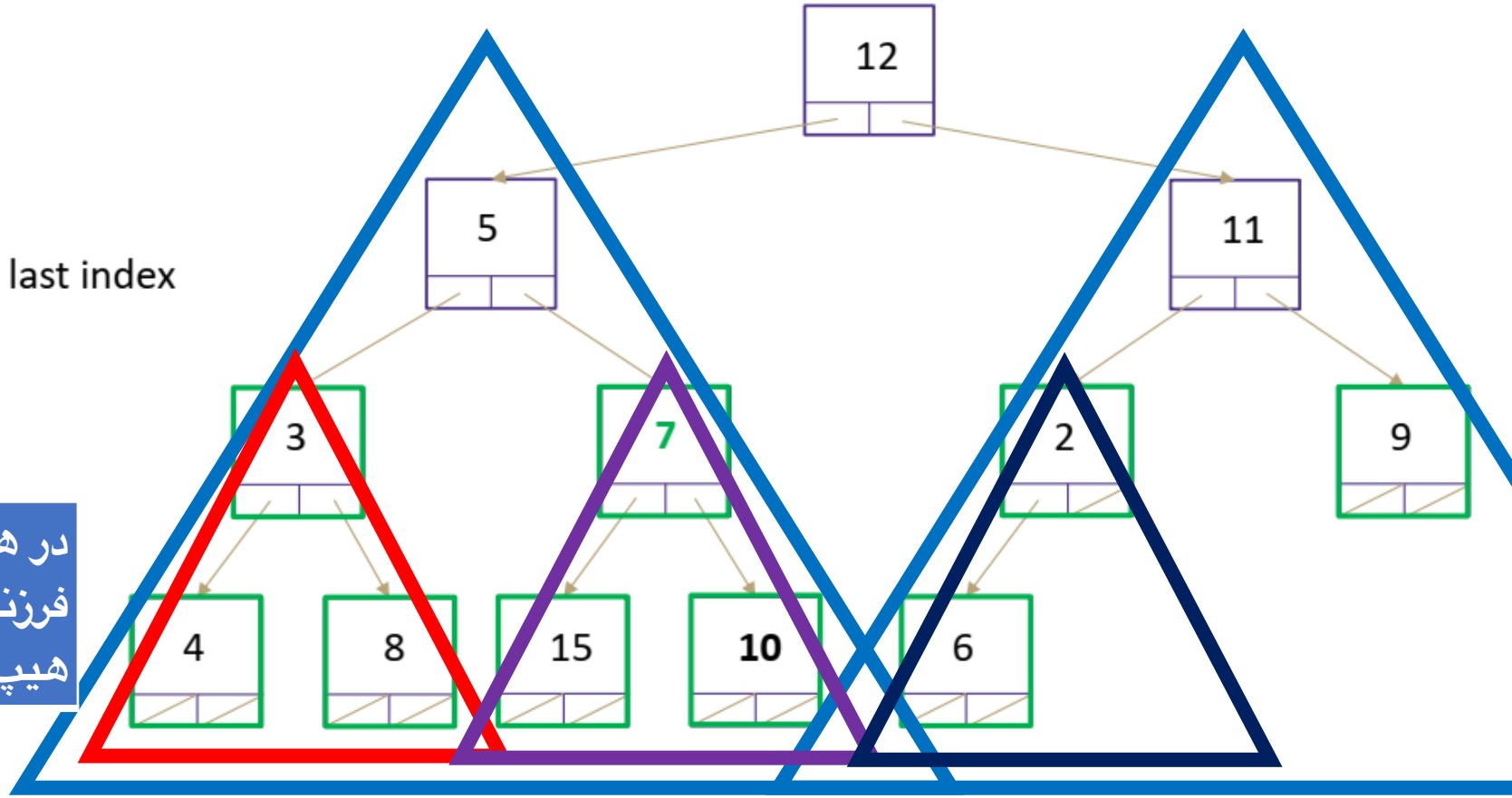
0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	7	2	9	4	8	15	10	6		

Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3

در هر مرحله ریشه زیر درخت را با کوچکترین فرزند جابه‌جا کرده تا کل زیر درخت مرحله، هیپ شود



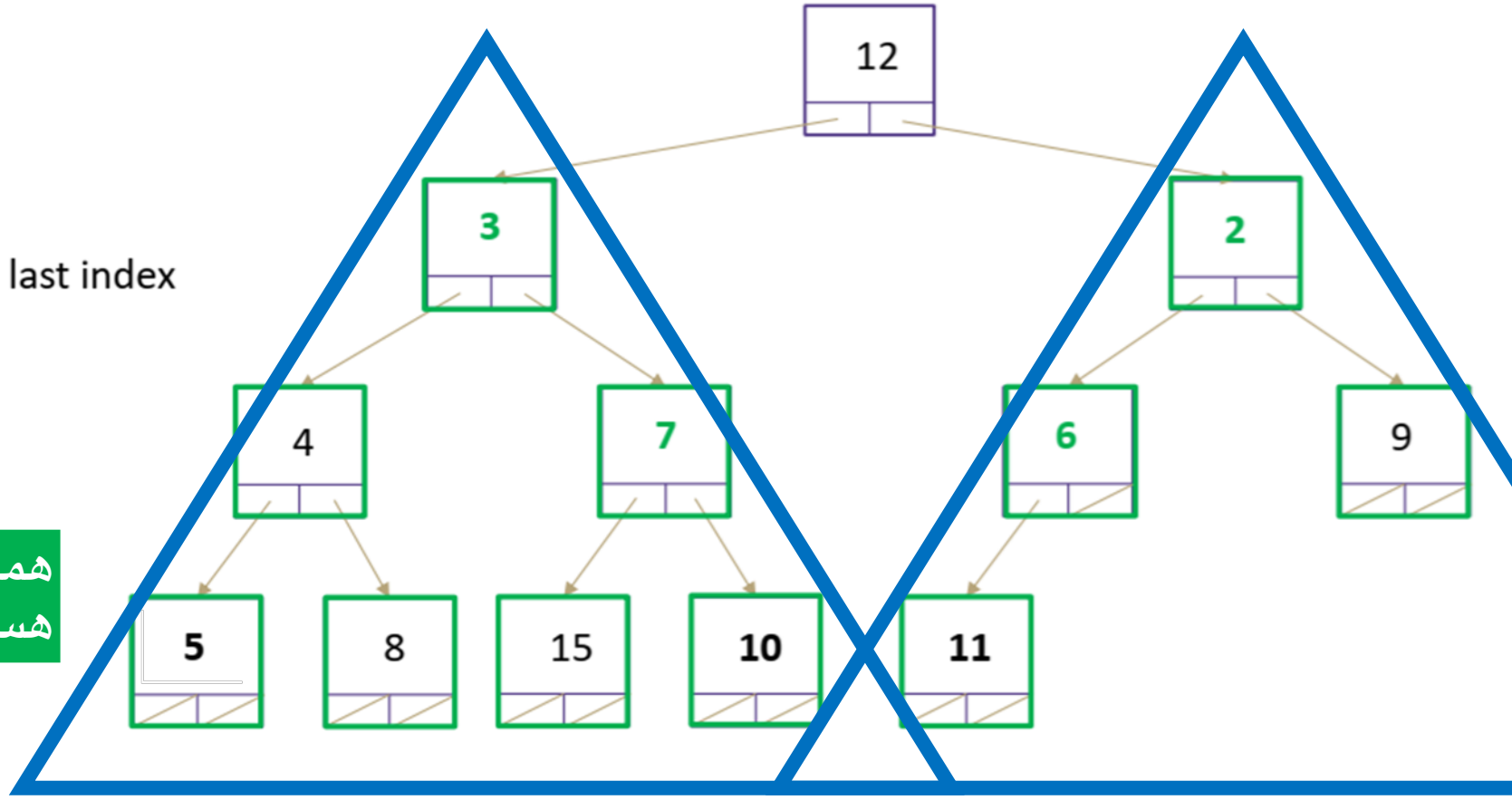
0	1	2	3	4	5	6	7	8	9	10	11	12	13
12	5	11	3	7	2	9	4	8	15	10	6		

Floyd’s buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3
    3. percolateDown level 2

همه زیر درخت های این مرحله، یک هیپ هستند

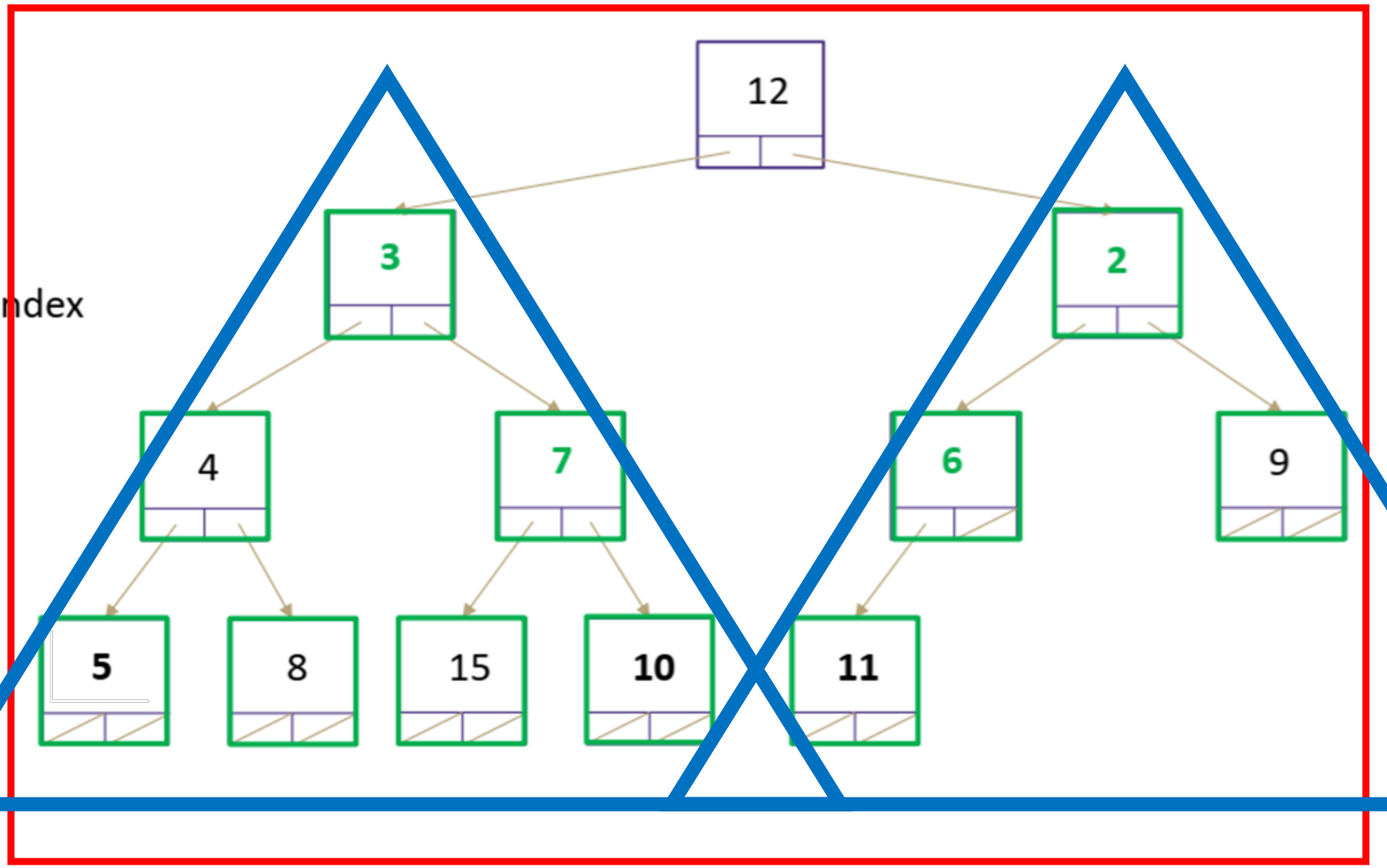


Floyd’s buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3
    3. percolateDown level 2

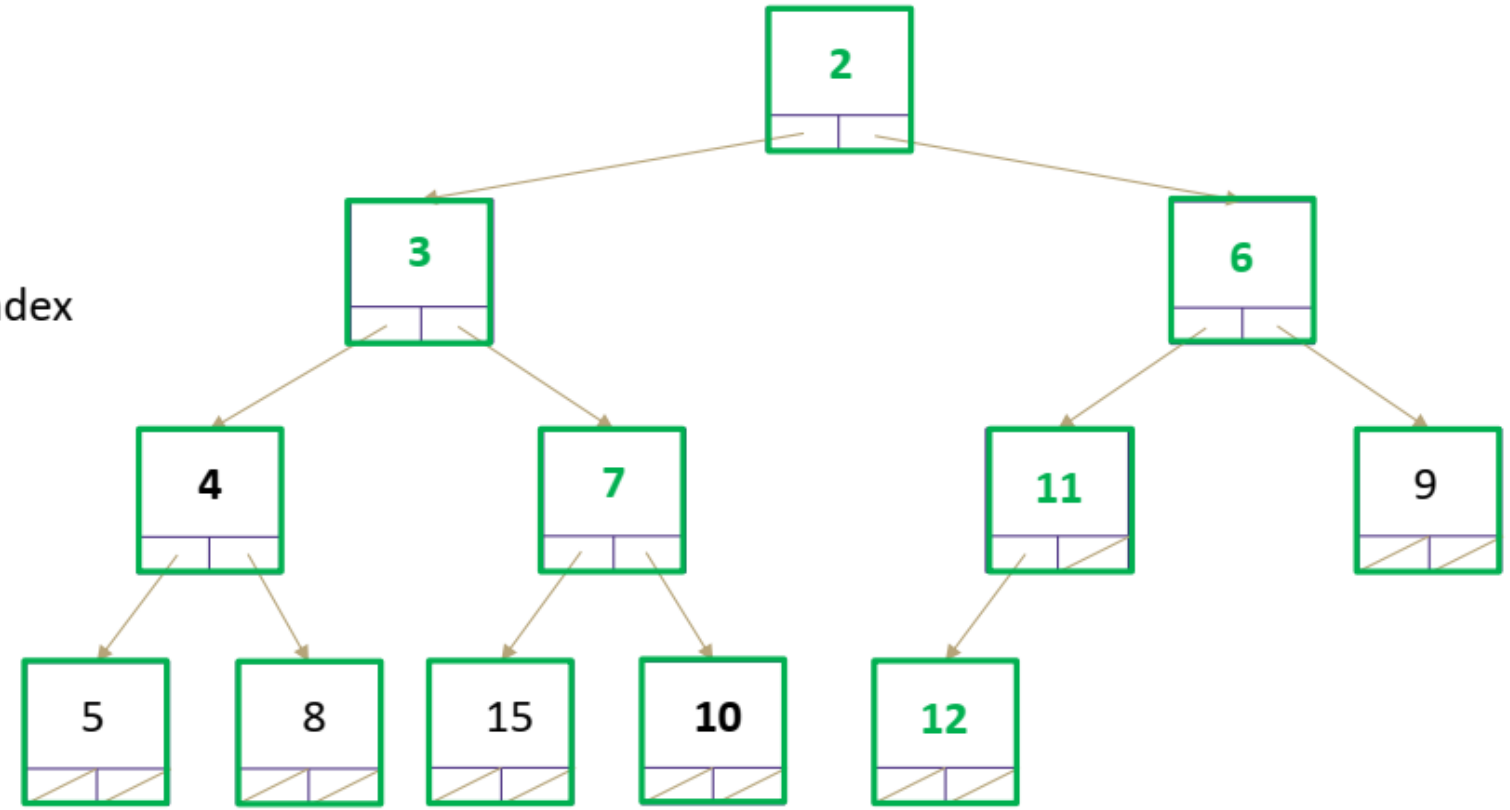
در هر مرحله ریشه زیر درخت را با کوچکترین فرزند جابه‌جا کرده تا کل زیر درخت مرحله، هیپ شود



Floyd's buildHeap algorithm

Build a tree with the values:
12, 5, 11, 3, 10, 2, 9, 4, 8, 15, 7, 6

1. Add all values to back of array
2. percolateDown(parent) starting at last index
    1. percolateDown level 4
    2. percolateDown level 3
    3. percolateDown level 2
    4. percolateDown level 1



0	1	2	3	4	5	6	7	8	9	10	11	12	13
2	3	6	4	7	11	9	5	8	15	10	12		

تحلیل

- در جدول زیر هزینه گره‌ها با ارتفاع i و تعداد آنها نشان داده شده است.

ارتفاع گره‌ها	تعداد گره‌ها در این ارتفاع	هزینه هر گره در این ارتفاع

- هزینه کل: $1\left(\frac{n}{2}\right) + 2\left(\frac{n}{4}\right) + 3\left(\frac{n}{8}\right) + \ldots + \log n(\frac{n}{n})$

تحلیل

- هزینه کل:

$$1\left(\frac{n}{2}\right)+2\left(\frac{n}{4}\right)+3\left(\frac{n}{8}\right)+\ldots+\log n\left(\frac{n}{n}\right)=$$

$$n\left(\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\ldots\right)=$$

$$n\left(1+\frac{1}{2}+\frac{1}{4}+\ldots\right)=2n\in O(n)$$

مسئله ۱

- گزارش تعداد اعداد بزرگتر از k در لیستی با اندازه n

- روش ۱: بررسی کل اعداد
  - هزینه ؟

مسئله ۱

- گزارش تعداد اعداد بزرگتر از k در لیستی با اندازه n

- روش ۱: بررسی کل اعداد
  - هزینه $O(n)$

مسئله ۱

- گزارش تعداد اعداد بزرگتر از k در لیستی با اندازه n

- روش ۲: مرتب سازی اعداد، سپس گزارش هر پرس و جو
  - هزینه هر پرس و جو ؟
  - هزینه پیش پردازش ؟

مسئله ۱

- گزارش تعداد اعداد بزرگتر از k در لیستی با اندازه n

- روش ۲: مرتب سازی اعداد، سپس گزارش هر پرس و جو
  - هزینه هر پرس و جو $O(t)$
  - هزینه پیش پردازش $O(n\log n)$

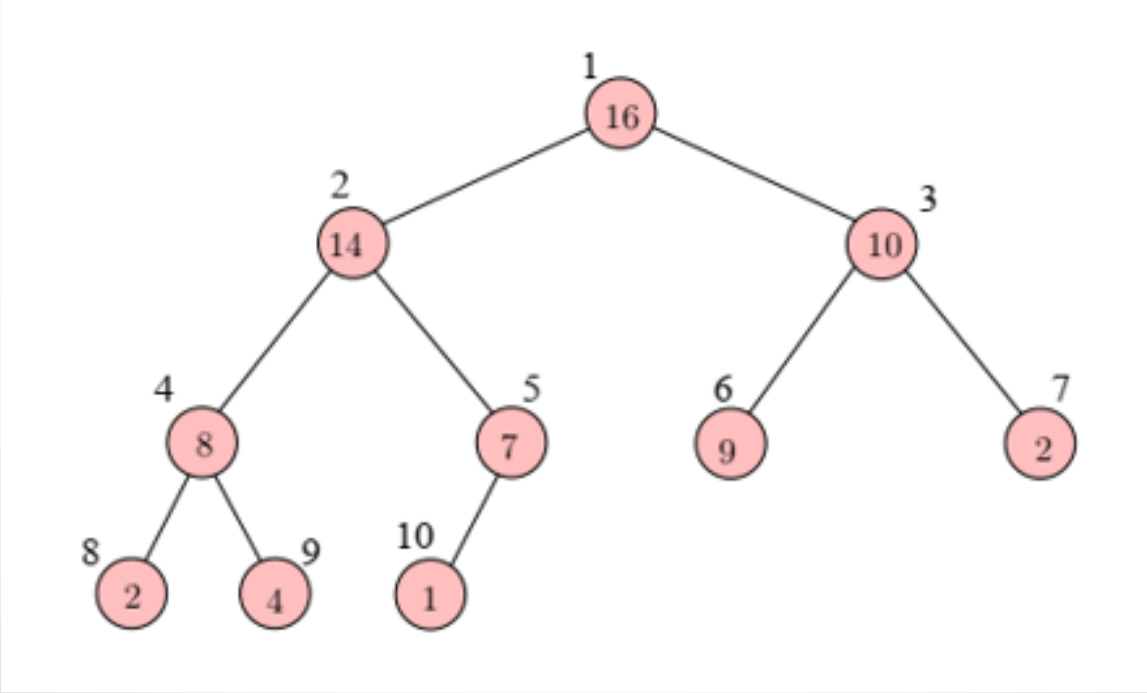
مسئله ۱

- گزارش تعداد اعداد بزرگ‌تر از k در لیستی با اندازه n

- روش ۳: ابتدا از اعداد یک هرم بیشینه می‌سازیم، سپس تعداد اعداد بزرگ‌تر از k را می‌شماریم

مسئله ۱

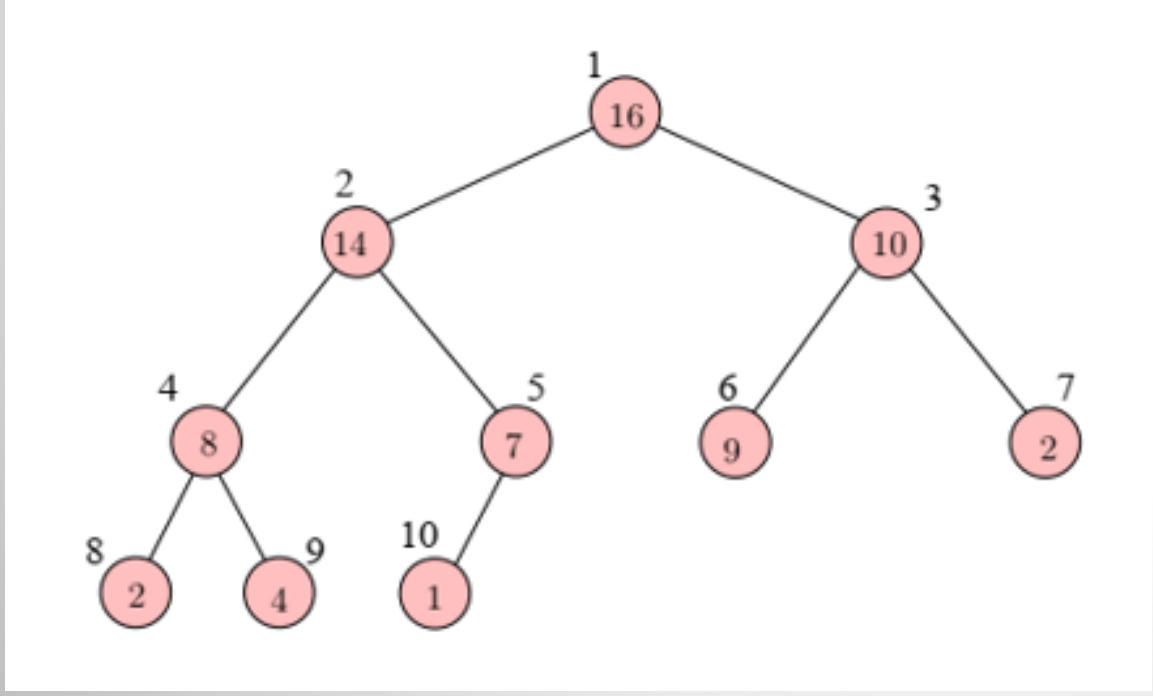
- مثال $k = 11$



مسئله ۱

- گزارش تعداد اعداد بزرگتر از k در لیستی با اندازه n

- روش ۳: ابتدا از اعداد یک هرم بیشینه می‌سازیم، سپس تعداد اعداد بزرگتر از k را می‌شماریم
  - هزینه هر پرس و جو ؟
  - هزینه پیش پردازش ؟



مسئله ۱

- گزارش تعداد اعداد بزرگ‌تر از k در لیستی با اندازه n

- روش ۳: ابتدا از اعداد یک هرم بیشینه می‌سازیم، سپس تعداد اعداد بزرگ‌تر از k را می‌شماریم
  - هزینه هر پرس و جو $O(t)$
    - چون فقط فرزندان t عنصر گزارش شده را بررسی می‌کنیم
  - هزینه پیش پردازش $O(n)$

مسئله ۲

- مرتب سازی با استفاده از هرم
  - ورودی: n عدد با نظم دلخواه
  - خروجی: n عدد مرتب شده

- روش: ؟

مسئله ۲

- مرتب سازی با استفاده از هرم
  - ورودی: n عدد با نظم دلخواه
  - خروجی: n عدد مرتب شده

- روش:
  - ساخت هیپ
  - n بار حذف بزرگ‌ترین عنصر

مسئله ۲

- مرتب سازی با استفاده از هرم
  - ورودی: n عدد با نظم دلخواه
  - خروجی: n عدد مرتب شده

- روش:
  - ساخت هیپ (هزینه ؟)
  - n بار حذف بزرگ‌ترین عنصر (هزینه ؟)

مسئله ۲

- مرتب سازی با استفاده از هرم
  - ورودی: n عدد با نظم دلخواه
  - خروجی: n عدد مرتب شده

- روش:
  - ساخت هیپ (هزینه $O(n)$)
  - n بار حذف بزرگترین عنصر (هزینه $O(n\log n)$)