هرم و مرتب سازی هرمی
HEAP AND HEAP SORT

فرشته دهقانی

سرفصل مطالب

- معرفی هرم
- عملیات های هرم
- پیاده سازی هرم
- روش های ساختن هرم
- مرتب سازی هرمی و دیگر کاربرد ها

معرفی هرم

- مثال:
- اورژانس یک بیمارستان را در نظر بگیرید که همیشه میخواهد به اورژانسی ترین بیمار رسیدگی کند. برای این کار باید بتواند به صورت سریع اورژانسی‌ترین بیمار را شناسایی کند. همچنین باید بتواند افرادی جدیدی که وارد می‌شوند را سریعا در محل خود در صف قرار دهد.

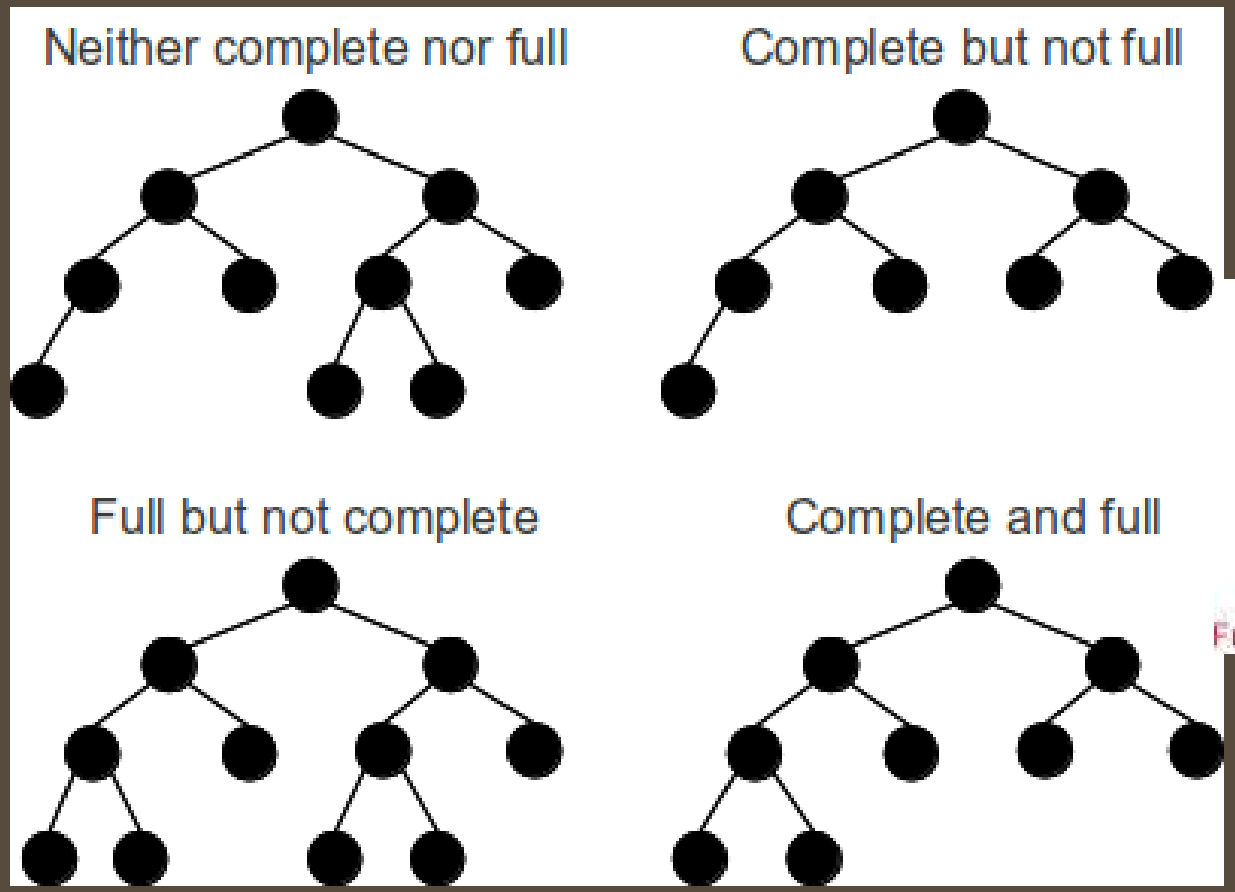
- سوال: این کار توسط یک آرایه همیشه مرتب چقدر زمان بر است

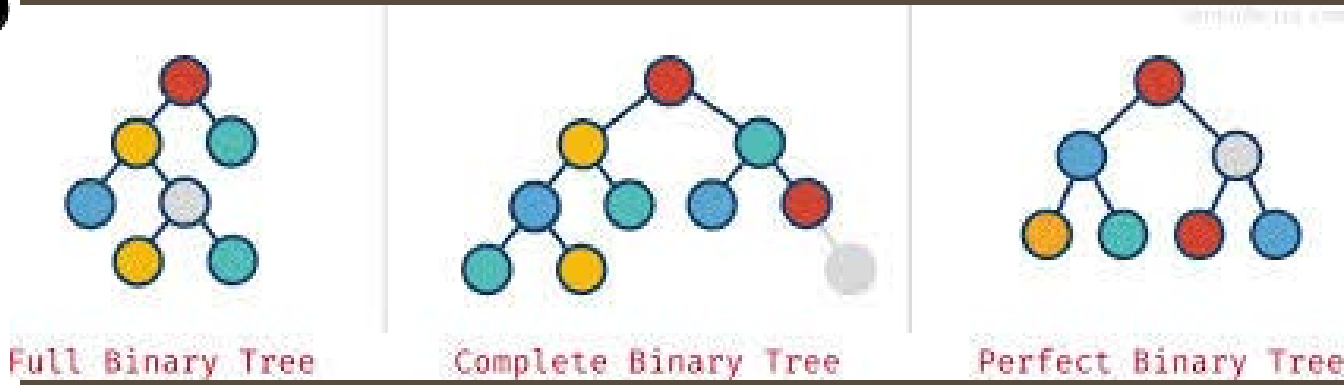
implementation	insert	delMax	max
Unordered array	1	N	N
Ordered array	N	1	1
goal	log N	log N	log N

هرم
HEAP

- داده ساختار هرم: یک درخت ریشه‌دار کامل (Complete Tree)
- رئوس به گونه ای قرار گرفته اند که ارزش هر راس از ارزش بچه‌هایش بیشتر مساوی (یا کمتر مساوی)
- درخت کامل :تمام سطوح درخت به غیر از احتمالا آخرین سطح پر بوده و برگ های سطح آخر از چپ به راست قرار گرفته اند.
- با توجه به این که درخت دودویی و کامل است، میتوان نتیجه گرفت که ارتفاع هرم همیشه برابر $\lfloor logn \rfloor$
- نکته : ساختار درختی هرم لزومی ندارد دودویی باشد ولی در این درس نوع دودویی آن را بررسی و استفاده می‌کنیم.

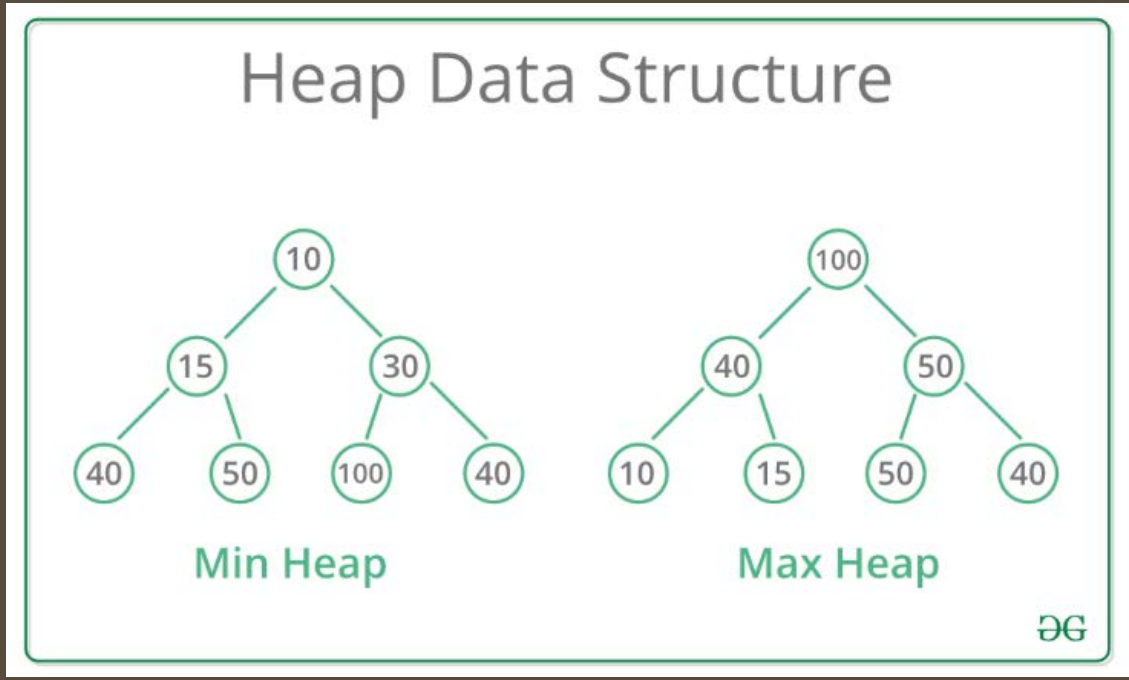
درخت کامل


Neither complete nor full
Complete but not full
Full but not complete
Complete and full


Full Binary Tree
Complete Binary Tree
Perfect Binary Tree

انواع هرم

- هرم بیشینه: ارزش هر راس از ارزش بچه هایش بزرگتر مساوی است.
- هرم کمینه: ارزش هر راس از ارزش بچه هایش کمتر مساوی است.



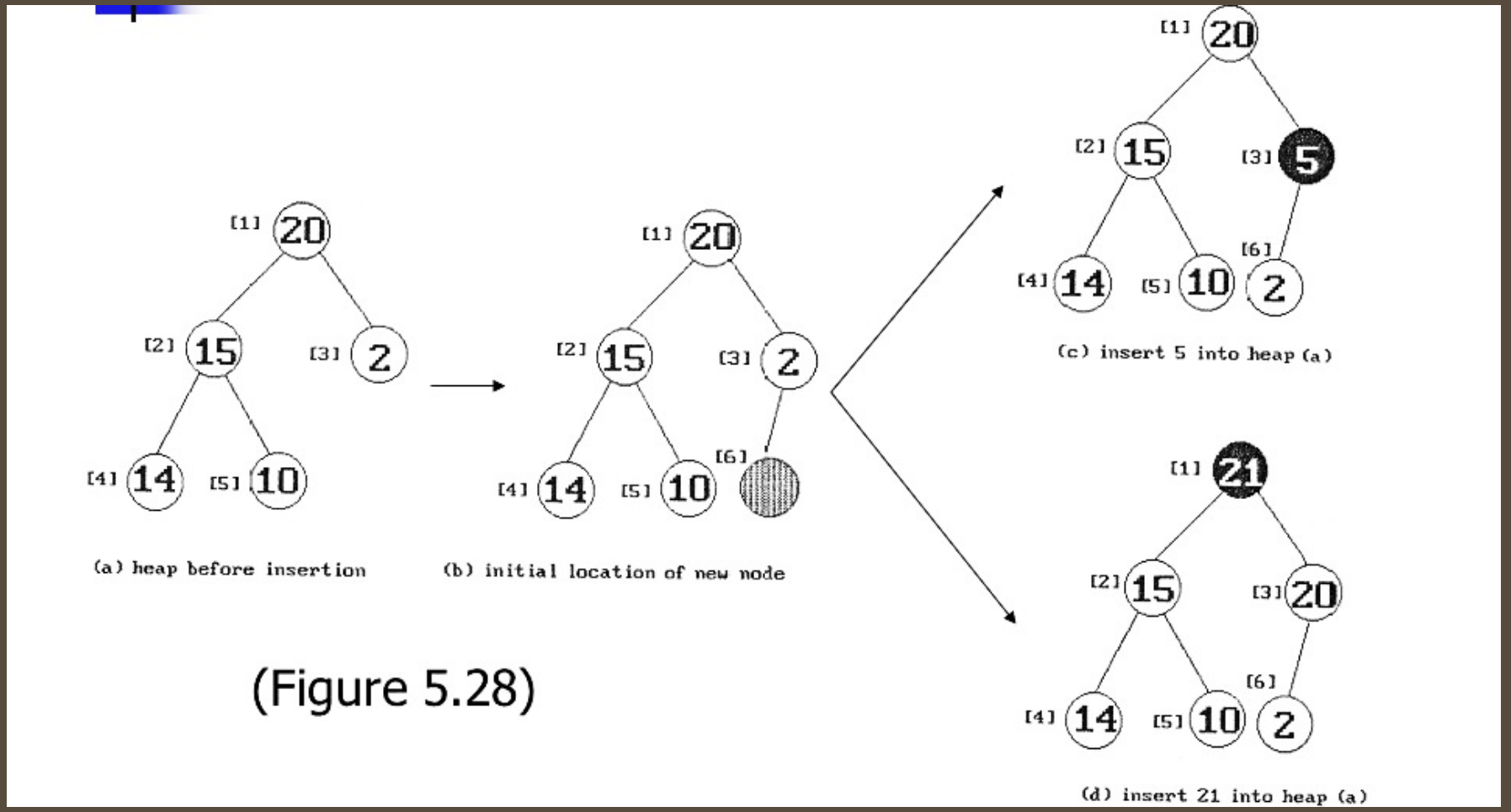
عملیات هرم

- هرم بیشینه:
  - درج
  - پیدا کردن عضو بیشینه
  - حذف عضو بیشینه
- (مشابه با هرم کمینه)

درج

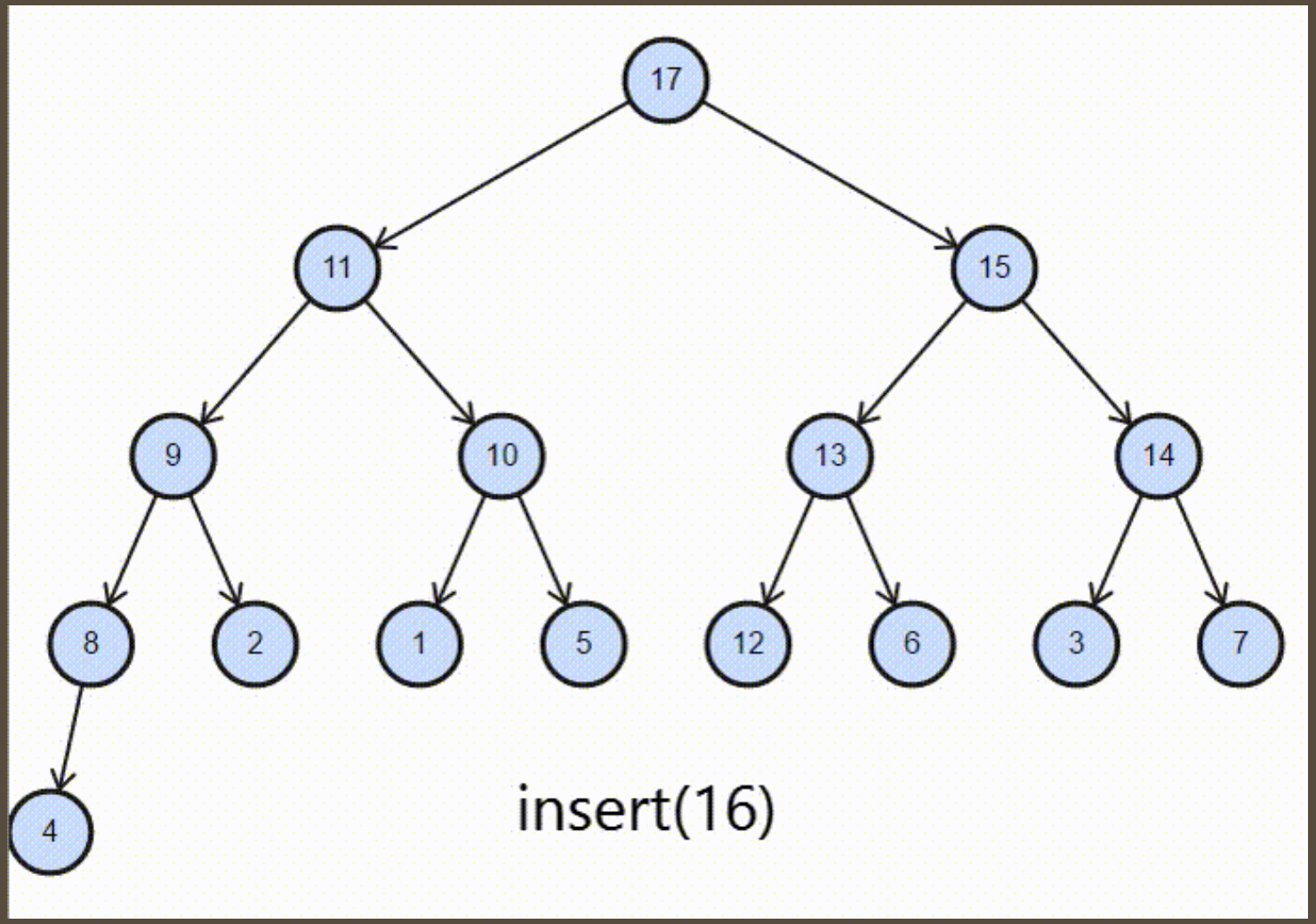
- برای درج یک عضو در هرم، ابتدا آن را در اولین مکان خالی قرار می‌دهیم(با حفظ کامل بودن درخت) سپس با تعدادی عمل جابجایی (bubble-up) عضو جدید را به مکان درستش منتقل می‌کنیم.
- در bubble-up عضو مورد نظر را با پدر خود مقایسه می‌کنیم و در صورتی که از آن بیشتر بود، جای آن دو را با هم عوض می‌کنیم. این کار را آنقدر ادامه می‌دهیم تا عضو مورد نظر از پدر خود کوچک‌تر شود، یا آنکه خودش ریشه درخت شود.
- چون هر عمل جابجایی عضو جدید را یک سطح در درخت بالاتر میبرد، bubble-up حداکثر به اندازه ارتفاع درخت یا همان logn است

مثال



(Figure 5.28)

مثال



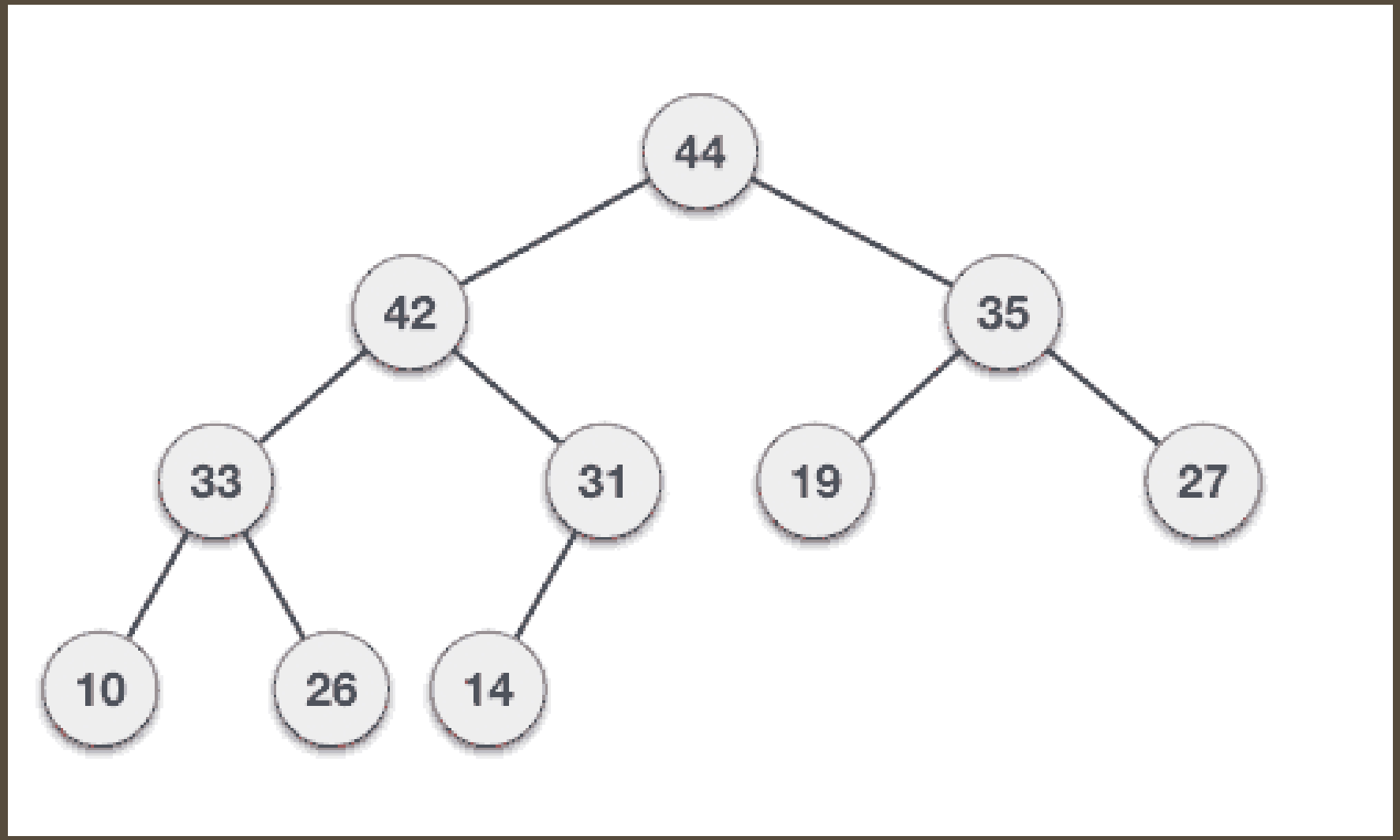
یافتن عضو بیشینه

- ریشه درخت همیشه حاوی عنصر بیشینه است
- از مرتبه O(1)

حذف عنصر بیشینه

- برای حذف عضو بیشینه در هرم، ریشه را از درخت حذف میکنیم و آخرین عضو هرم(چپ ترین برگ در پایین ترین عمق) را در جایگاه ریشه قرار می‌دهیم. سپس با تعدادی عمل جابجایی (bubble-down) این عضو را به مکان درست منتقل می‌کنیم. (همچنان درخت کامل است)
- در bubble-down، عضو مورد نظر را با بزرگترین فرزند خود مقایسه می‌کنیم و در صورتی که از آن کمتر بود، جای آن دو را با هم عوض می‌کنیم. این کار را آنقدر ادامه می‌دهیم تا عضو مورد نظر از هر دو فرزند خود کوچکتر شود، یا آنکه خودش برگ درخت شود.
- چون هر عمل جابجایی عضو جدید را یک سطح در درخت پایین‌تر می‌برد، bubble-down حداکثر به اندازه ارتفاع درخت یا همان O(logn)

حذف عنصر بیشینه



مقایسه هرم و آرایه مرتب

- درج
  - هرم: O(logn)
  - آرایه: O(n)
- حذف بیشینه
  - هرم: O(logn)
  - آرایه: O(1)
- یافتن عضو بیشینه
  - هرم: O(1)
  - آرایه: O(1)

پیاده سازی هرم با استفاده از آرایه

- هرم را می‌توان با استفاده از آرایه پیاده‌سازی کرد. به این صورت که اگر یک عضو در خانه iام آرایه قرار گرفت (i=1,...n)
- برای سهولت پیاده سازی اندیس صفر را خالی میگذارند
- فرزندان چپ و راستش(در صورت وجود) به ترتیب در خانه های 2i و 2i+1

ساخت درخت HEAP – روش اول

- با شروع از اول لیست، اعداد را یک به یک در هرم bubble_up می‌کنیم. با این روش در هر مرحله هرم تولید شده با عناصری که تا آن لحظه در هرم bubble_upشده‌اند یک هرم درست خواهد بود. این کار را آنقدر ادامه می‌دهیم تا همه عناصر در محل درستشان قرار گیرند. دقت کنید این روش مانند آن است که عناصر را یکی یکی در یک هرم خالی درج کنیم.

ساخت درخت HEAP

ساخت درخت HEAP

O(nlgn)

تحلیل زمانی

$$\sum_{i=1}^{\log n} i.2^i = 1 \times 2 + 2 \times 2^2 + \ldots + \log n \times 2^{\log n}$$

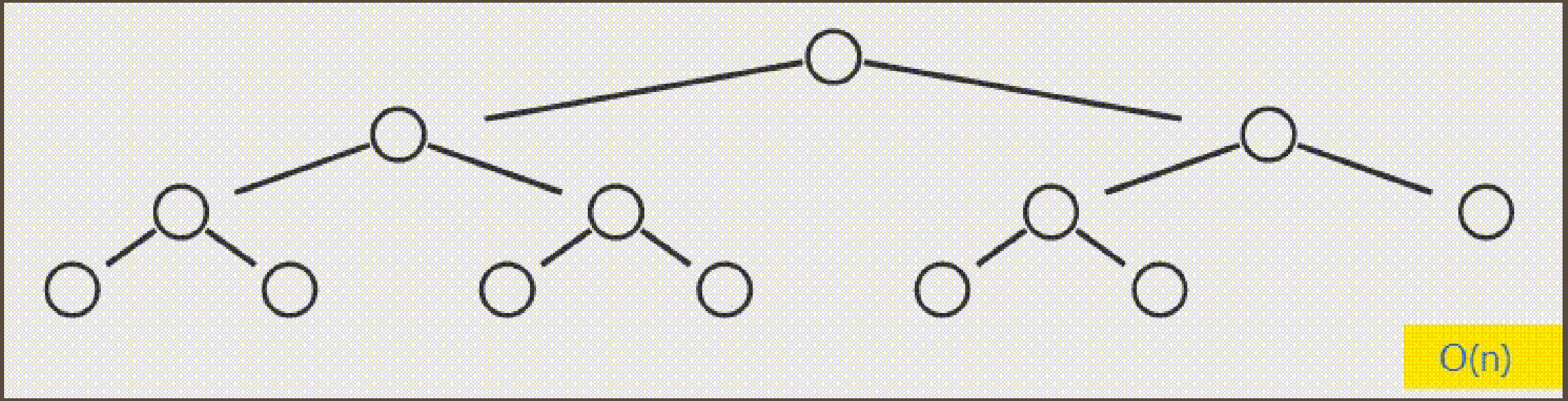
$$= 2 + 2^2 + 2^3 + \ldots + 2^{\log n}$$
$$+ 2^2 + 2^3 + \ldots + 2^{\log n}$$
$$+ 2^3 + \ldots + 2^{\log n}$$
$$\ldots$$
$$+ 2^{\log n}$$

$$= (2^{\log n+1} - 2) + (2^{\log n+1} - 2^2) + (2^{\log n+1} - 2^3) + \ldots + (2^{\log n+1} - 2^{\log n})$$

$$= 2^{\log n+1}.\log n - \sum_{i=1}^{\log n} 2^i = 2n.\log n - (2^{\log n+1} - 2) \in O(n \log n)$$

ساخت درخت HEAP – روش دوم

- با شروع از **آخر لیست** ، عناصر را یک به یک در هرم bubble_down میکنیم. با این کار تعدادی هرم کوچک ساخته میشود که در مراحل بعد با هم ترکیب میشوند تا هرم اصلی را بسازند.


O(n)

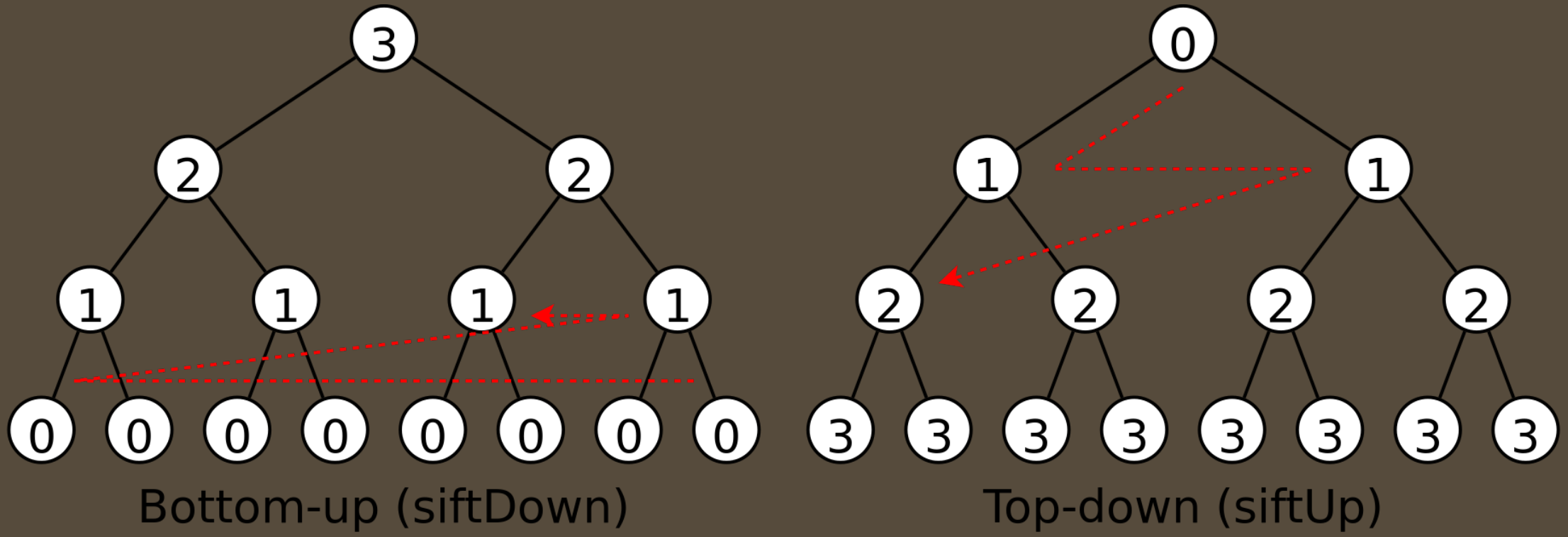
تحلیل زمانی

$$\sum_{i=0}^{\log n} (\log n - i).2^i = \log n \sum_{i=0}^{\log n} 2^i - \sum_{i=0}^{\log n} i.2^i$$

$$= \log n(2^{\log n+1} - 1) - (\log n.2^{\log n+1} - (2^{\log n+1} - 2)) = 2n - \log n - 2$$

$$\in O(n)$$

مقایسه تعداد جابجایی ها در دو روش

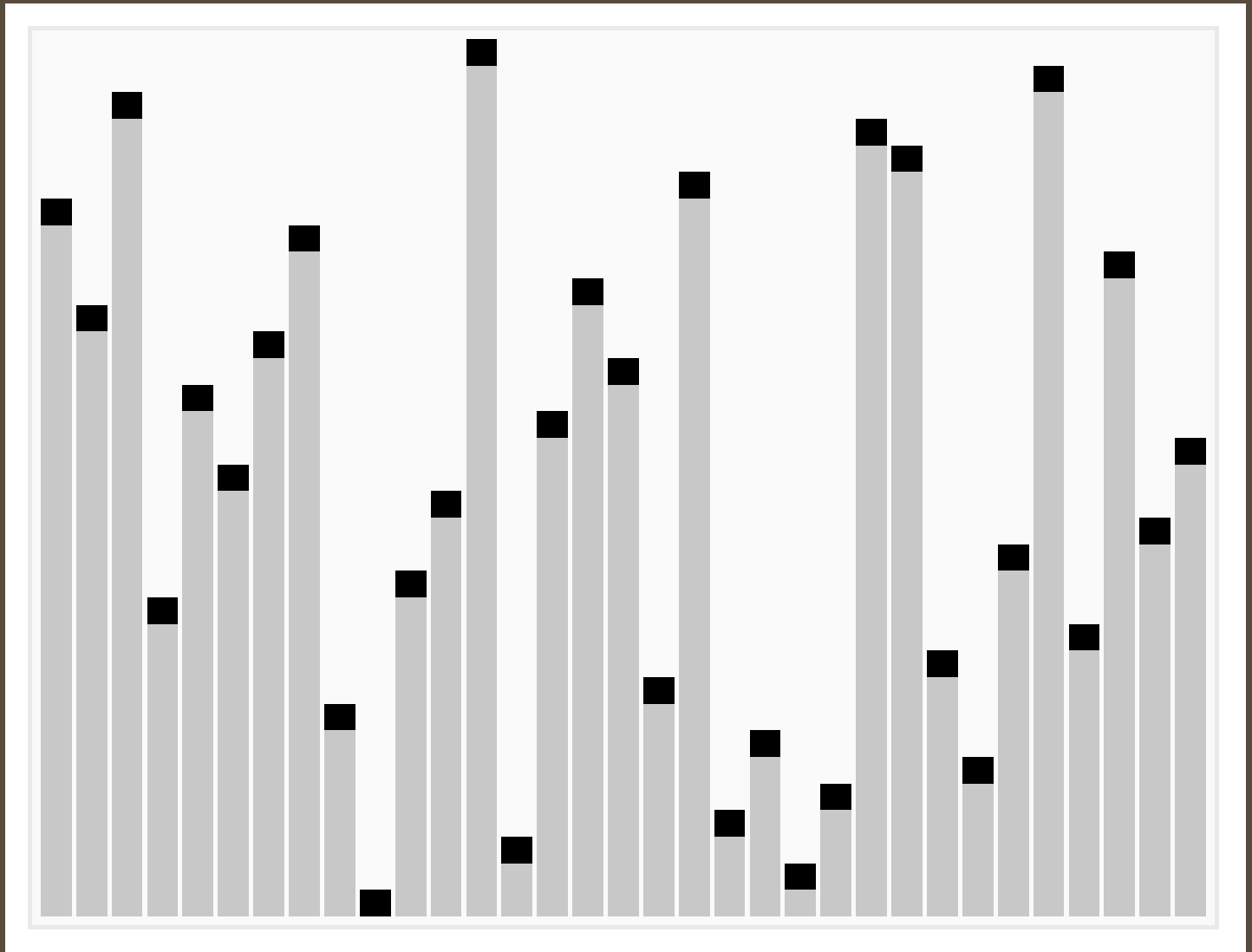


The number in the circle indicates the maximum times of swapping required when adding the node to the heap.

کاربرد هرم : مرتب سازی هرمی
HEAP SORTING

- یکی از کاربردهای مهم هرم، مرتب سازی هرمی است.
- در این روش اعدادی که می‌خواهیم مرتب کنیم را درون یک هرم (مثلا هرم بیشینه) قرار می‌دهیم. (ساخت درخت هرمی با مرتبه O(n))
- سپس کافیست مقدار بیشینه را از هرم بخوانیم و آن را ثبت و از هرم حذفش کنیم (جایگزینی اخرین فرزند با عنصر ماکزیمم و دوباره هرمی کردن درخت از مرتبه O(logn)) و این کار را آنقدر ادامه می‌دهیم تا هرم خالی شود. (O(n+nlogn)=O(nlogn))

HEAP SORTING



SORTING ALGORITHM	TIME COMPLEXITY		
	BEST CASE	AVERAGE CASE	WORST CASE
Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Heap Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Quick Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N^2)$
Radix Sort	$\Omega(N k)$	$\Theta(N k)$	$O(N k)$
Count Sort	$\Omega(N + k)$	$\Theta(N + k)$	$O(N + k)$
Bucket Sort	$\Omega(N + k)$	$\Theta(N + k)$	$O(N^2)$

اهمیت مرتب سازی هرمی

- اهمیت. یک روش مرتب سازی درجا که در بدترین حالت از مرتبه O(Nlg N) است.
- مرتب سازی ادغامی: خیر، مصرف حافظه ی کمکی خطی
- مرتب سازی سریع: خیر، در بدترین حالت از مرتبه درجه دوم
- مرتب سازی هرمی: بله!

- سخن آخر. مرتب سازی هرمی از لحاظ حافظه و زمان بهینه است، اما:
- در حالت متوسط اندکی کندتر از مرتب سازی سریع است.
- پایدار نیست!

تمرین

- به کمک هرم کمینه و به روش مرتب سازی هرمی، لیست ورودی را مرتب کنید و آن را برگردانید.

کاربرد هرم: صف اولویت
PRIORITY QUEUE

- صف اولویت داده ساختاری شبیه صف و پشته است با این تفاوت که برخلاف صف و پشته که اولویت ورود و خروج را بر حسب زمان ورود تعیین می‌کنند، می‌تواند برای ورود و خروج عناصر اولویت‌های دیگری هم تعیین کند.
- یک روش پیاده‌سازی صف اولویت با استفاده از هرم است.

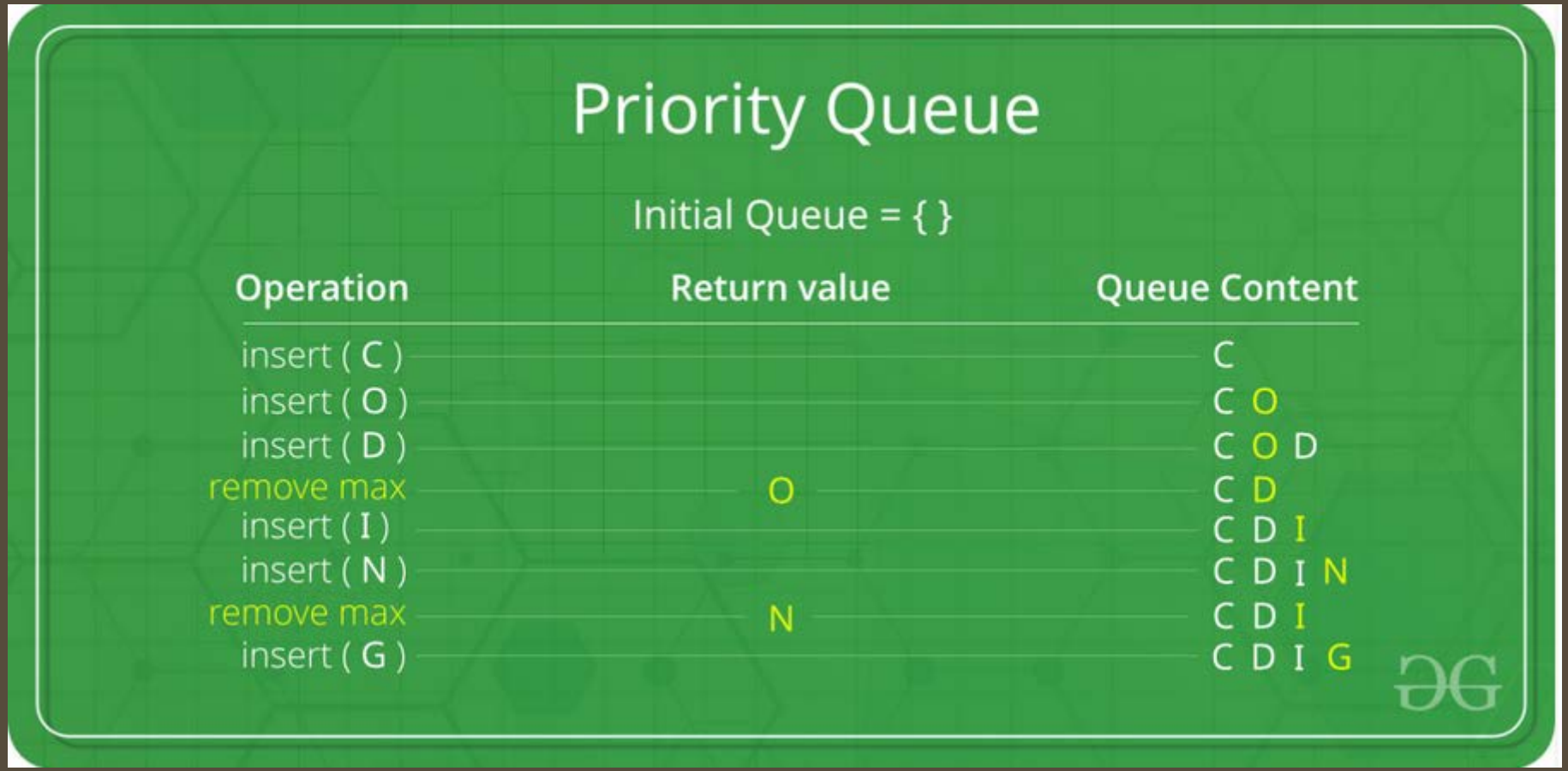
مقایسه داده ساختارها

- پشته. حذف عنصری که دیرتر از بقیه اضافه شده است.
- صف. حذف عنصری که زودتر از بقیه اضافه شده است.
- صف تصادفی. حذف یک عنصر به صورت تصادفی.
- صف اولویت. حذف بزرگ ترین(یا کوچک ترین) عنصر

کاربرد صف اولویت

- یافتن M بزرگ ترین عنصر در دنبال های از N عنصر در حال جریان.
- تشخیص کلاهبرداری: جدا کردن تراکنشهای بزرگ
- نگهداری فایل ها: یافتن بزرگترین فایل ها و دایرکتوری ها
- در الگوریتم های مبتنی بر گراف مثل دایجسترا (یافتن کوتاهترین مسیر)، پرایم (گراف پوشای کمینه)

مثال صف اولویت



صف اولویت

- هر عنصر ورودی در صف، یک اولویت دارد
- المانی که اولویت بالاتری دارد، زودتر از المان با اولویت کمتر از صف خارج می شود (Dequeue)
- اگر دو المان اولویت مشابهی داشته باشند، بر اساس ترتیب ورودشان از صف خارج می شوند

عملیات رایج در صف اولویت
و مرتبه آنها در صورت پیاده سازی با استفاده از هرم

- **insert(item, priority):** اضافه کردن یک عنصر با اولویت مشخص شده (O(logn))
- **getHighestPriority():** برگرداندن آیتم با بالاترین اولویت (O(1))
- **deleteHighestPriority():** حذف آیتم با بالاترین اولویت (O(logn))
- پیاده سازی با استفاده از هرم اولویت بالاتر نسبت به آرایه مرتب شده یا لینک لیست

تمرین

- پیاده سازی صف اولویت دارای توابع زیر
- Is_empty()
- Is_full()
- **insert(item, priority)**
- **getHighestPriority()**
- **deleteHighestPriority()**
- **Get_size()**

تمرین

- پیاده سازی توابع صف اولویت با استفاده از هرم