Heapsort

برخی ویژگی‌های Heap sort

۱- مانند الگوریتم merge sort در بدترین حالت $O(n \lg n)$ است

۲- مانند الگوریتم insertion sort یک الگوریتم in place است.

برخی ویژگی‌های data structure ی درخت Heap

۱- درخت Heap یک درخت باینری نزدیک به کامل است. (almost-complete / nearly-complete)

۲- ارتفاع درخت heap برابر $\Theta(\lg n)$ است.

* ۳- یک درخت heap را می‌توان به راحتی در یک آرایه جا داد و ذخیره کرد (به صورت sorted)

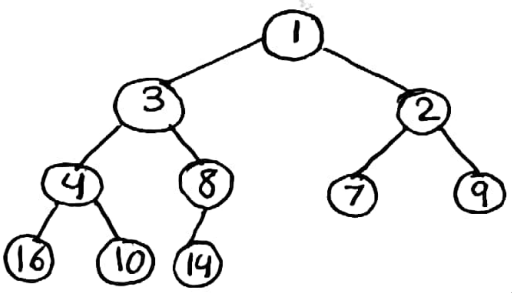
Definitions:

دو نوع درخت heap وجود دارد:

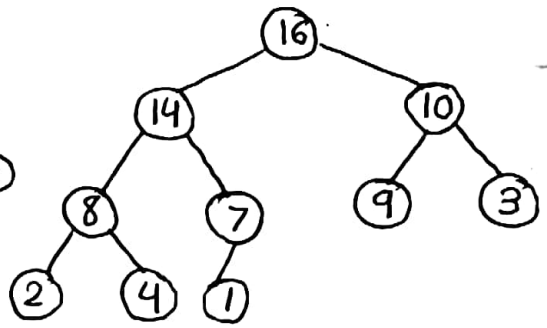
max-heap trees: برای تمام node های واقع در درخت داریم (به جز root) بدین معنا که node ی i کوچکتر یا مساوی پدرش است و ریشه شامل بزرگترین عنصر است.

min-heap trees: برای تمام node های واقع در درخت به جز ریشه داریم: node ی i بزرگتر یا مساوی پدرش است و در نتیجه root شامل کوچکترین عنصر است.

min-heap



max-heap



: convention

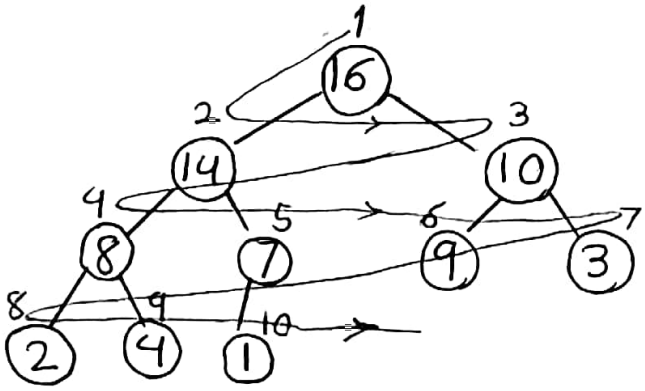
$i \rightarrow$ node اندیس $\Rightarrow$ PARENT(i) $\rightarrow$ اندیس پدر i

---

* برای ذخیره node های heap tree A از یک آرایه A استفاده می‌کنیم

Example

max-heap tree



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

در این صورت داریم :

Root of tree is always $A[1]$

Parent of $A[i] = A[\lfloor \frac{i}{2} \rfloor]$

left child of $A[i] = A[2i]$

Right child of $A[i] = A[2i+1]$

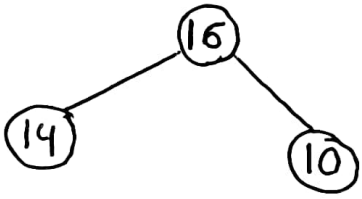
Example :

Parent of $A[9] \rightarrow A[4]$

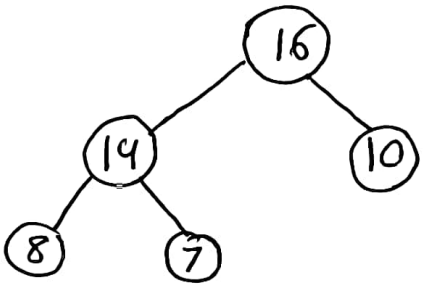
Right child of $A[3] \rightarrow A[7]$

ساختن درخت از روی آرایه هم بسیار ساده است :

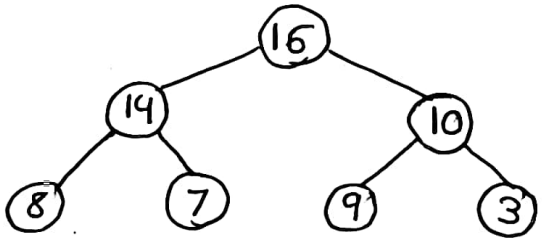
16	14	10	8	7	9	3	2	4	1



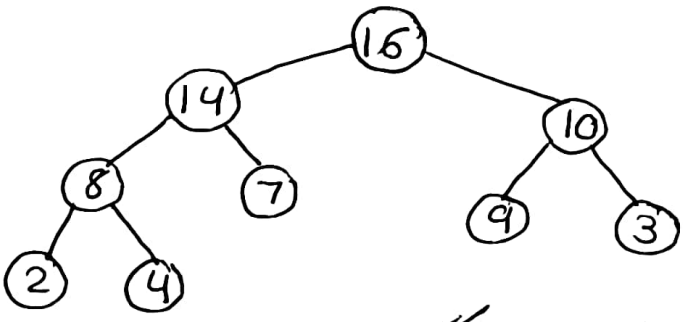
16	14	10	8	7	9	3	2	4	1



16	14	10	8	7	9	3	2	4	1



16	14	10	8	7	9	3	2	4	1



و یک مرحله دیگر

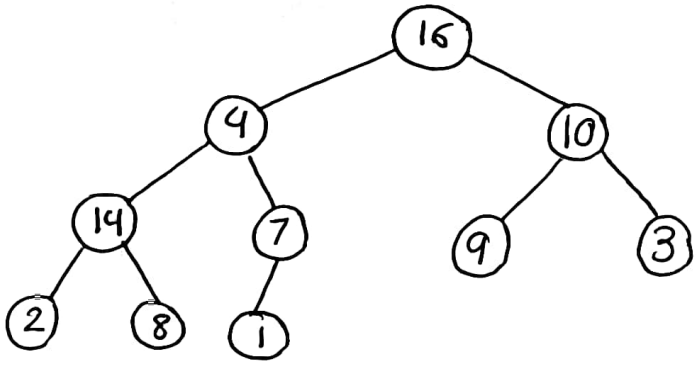
چند نکته و یادآوری:

- با استقرای ریاضی (induction) می‌توان ثابت کرد که root یک max-heap همواره شامل عنصر maximum است.
- در حالت general، heap ها می‌توانند به جای binary، درخت k-ary به ازای $k \geq 2$ باشند.

Maintaining the heap property:

چگونه یک درخت binary ی nearly complete را به max-heap tree تبدیل کنیم (MAX-HEAPIFY)

MAX-HEAPIFY پروسیجری است که با ویرایش یک درخت آن را max-heap تبدیل می‌کند.



* در درخت فوق ممکن است یک node ی i از فرزندانش کوچکتر باشد.

دو فرض:
- درخت نزدیک به کامل است.
- درخت در آرایه ذخیره شده و انتظار داریم، نتیجه نیز در آرایه ذخیره شده داده شود.

16	4	10	14	7	9	3	2	8	1

↓

یک max-heap نیست.

Max-HEAPIFY (A, i, n)

آرایه‌ای که درخت در آن است

بر روی زیردرخت با ریشه i کار می‌کند.

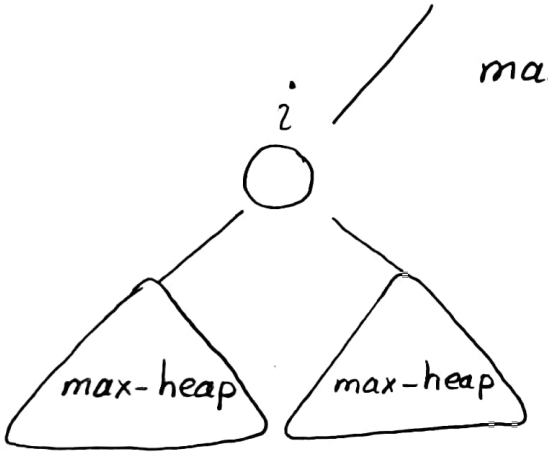
(heap-size[A])

تعداد کل node های درخت

نحوه عملکرد:

این پروسیجر، اگر left subtrees و right ی A[i]، max-heap باشند

با انجام این پروسیجر درختی که ریشه آن A[i] است، max-heap می‌شود



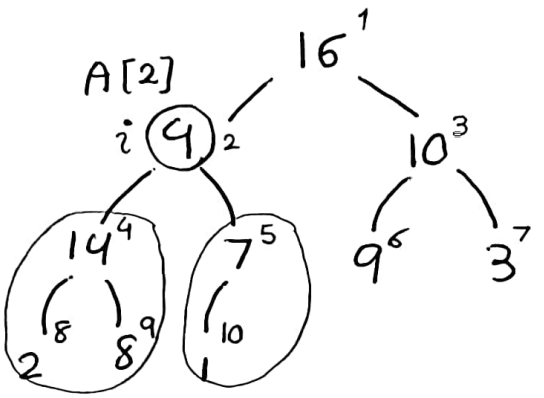
دو زیردرخت گره i ام max-heap هستند

⇓

در پایان، کل زیردرخت با ریشه i max-heap می‌شود.

```
MAX-HEAPIFY (A, i, n)
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ n and A[l] > A[i]
        then largest ← l
        else largest ← i
if r ≤ n and A[r] > A[largest]
        then largest ← r
if largest ≠ i
        then exchange A[i] ↔ A[largest]
                MAX-HEAPIFY (A, largest, n)
```

MAX-HEAPIFY (A , i=2, n=10) روند اجرای



max-heap subtrees ✓

$i \leftarrow 2$ $\quad A[i] = 4$

$l \leftarrow 4$ $\quad A[l] = 14$

$r \leftarrow 5$ $\quad A[r] = 7$

if $A[l] > A[i]$

then largest $\leftarrow l$

else largest $\leftarrow i$

$\Downarrow$

largest $\leftarrow 4$

if $A[r] > A[\text{largest}]$

then largest $\leftarrow r$

$\Downarrow$

largest $= 4$

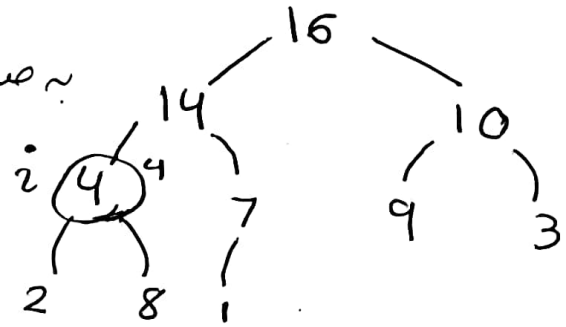
if largest $\neq i$

then exchange $A[i] \leftrightarrow A[\text{largest}]$

MAX-HEAPIFY (A, largest, n)

به صورت بازگشتی ادامه می یابد تا برگها

MAX-HEAPIFY(A, 4, 10)



در پایان اجرای recursively این پروسیجر زیر درخت ای از درخت اصلی که ریشه آن $i=2$ است، max-heap شده است

Running Time

زمان اجرا:

$O(\lg n)$

یک درخت almost-complete و دودویی $O(\lg n)$، level دارد.
(عمق یا ارتفاع آن $O(\lg n)$ است)

تمام سطر دستورات پروسیجر در زمان constant انجام می‌شود فقط بدلیل recursive بودن ممکن است به ازای یک مسیر از گره i ام تا یکی از leaf ها (تعداد node های) الگوریتم خود را call کند.

یک مسیر از node ی i تا یک leaf از ارتفاع درخت کمتر است.

$$O(\lg n) \times C = O(\lg n)$$

Building a heap:

پروسیجر زیر از یک آرایه‌ی unordered یک max-heap تولید می‌کند

```
BUILD-MAX-HEAP (A, n)
  for i ← ⌊n/2⌋ downto 1
      do MAX-HEAPIFY (A, i, n)
```

که n همان heap-size[A] یا length[A] است

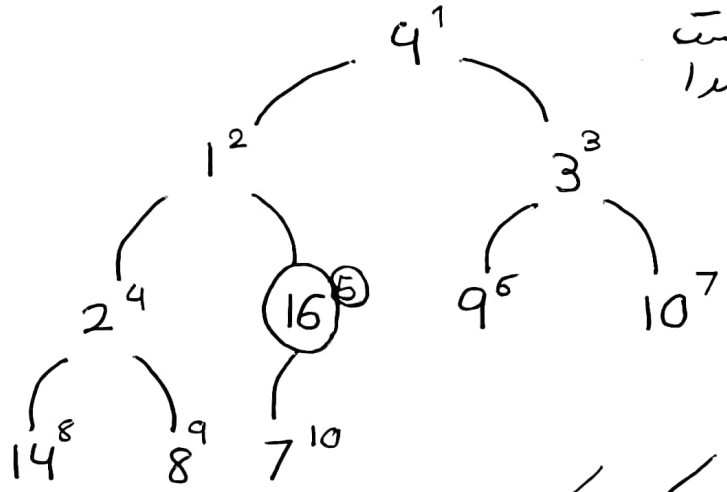
Example:

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

$\lfloor n/2 \rfloor$ که در اینجا می‌شود 5

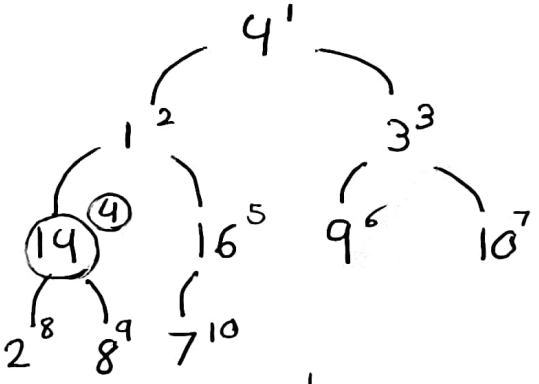
اندیس می‌باشد که آن node دارای فرزند است

بزرگترین



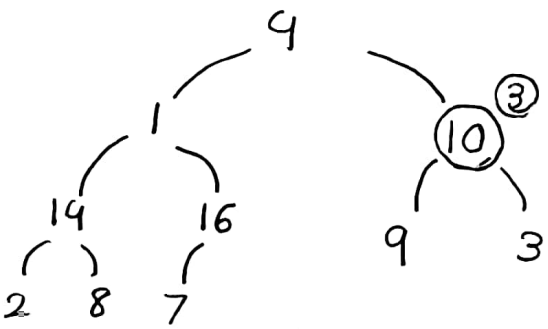
MAX-HEAPIFY(A, 5, 10)

subtree ی با ریشه 5 را یک max-heap می‌کند ← که هست



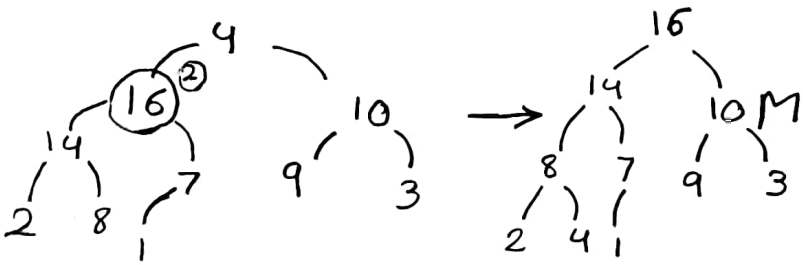
MAX-HEAPIFY(A, 4, 10)

subtree ی با ریشه 4 را یک max-heap می‌کند



MAX-HEAPIFY(A, 3, 10)

MAX-HEAPIFY(A, 2, 10)



MAX-HEAPIFY(A, 1, 10)

در ادامه correctness این پروسیجر را اثبات می‌کنیم

- یک آنالیز زمانی ارائه می‌دهیم

- و در پایان پیچیدگی زمانی آن را بیان می‌کنیم.

- الگوریتم heap sort را ارائه می‌دهیم

نکته: نحوه ساختن min-heap مشابه max-heap است.

Correctness:

درستی الگوریتم:

```
BUILD_MAX_HEAP (A, n)
    for i ← ⌊n/2⌋ downto 1
        do MAX-HEAIFY(A, i, n)
```

ثابت حلقه (Loop invariant):

در شروع هر تکرار از حلقه for، هر رأس از درخت با اندیس‌های $i+1$، $i+2$، ... تا n ریشه یک درخت max-heap است. (در شروع تکرار i ام)

Initialization

در آغاز حلقه for، مقدار اندیس حلقه $i \leftarrow \lfloor n/2 \rfloor$ است و می‌دانیم node های $\lfloor n/2 \rfloor + 1$، $\lfloor n/2 \rfloor + 2$، ... و n همگی leaf هستند پس زیردرخت‌های به ریشه آنها تک عضو هستند بنابراین هر درخت تک عضوی یک max-heap است.

Maintenance

فرض می‌کنیم در آغاز شروع تکرار i ام تمام node های $i+1$ تا n ریشه‌های زیردرخت‌های max-heap باشند.

حال در شروع تکرار $i-1$ ام هستیم:

- تمام node های $i+1$ تا n ریشه max-heap هستند پس دو فرزند node ی i ام هم که اندیس هایشان از i بزرگتر است دو زیردرخت max-heap هستند

- در سطر سوم MAX-HEAIFY(A,i,n) اجرا می‌شود چون فرزندان رأس i ام max-heap هستند این پروسیجر کل زیردرخت به ریشه node ی i ام را هم max-heap می‌کند در نتیجه

⇐ تمام node های i تا n ریشه یک زیردرخت max-heap هستند (در ابتدای تکرار i-ام)

Termionation

زمانی که $i=0$ شود وارد حلقه for نمی‌شویم و خروج می‌کنیم.

با ثابت حلقه داریم در این مرحله هر رأس از درخت با اندیس های ۱، ۲، ... تا n همه ریشه یک درخت max-heap هستند و چون رأس با اندیس ۱ همان root است کل درخت max-heap شده است

Analysis

Simple bound : در مجموع $O(n)$ بار MAX-HEAPIFY اجرا می‌شود که اگر هر بار $O(\lg n)$ زمان ببرد در مجموع $O(n\lg n)$ زمان اجرا است

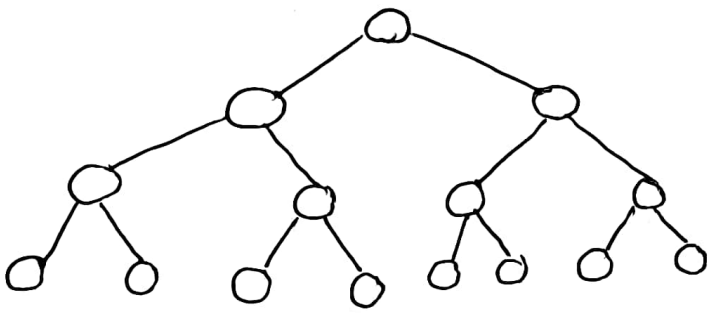
Tighter analysis :

- الگوریتم MAX-HEAPIFY در زمان linear نسبت به ارتفاع زیردرخت به راس i کار می‌کند

○ اکثر node های مورد بررسی در الگوریتم BUILD-MAX-heap دارای ارتفاع کمتر از ارتفاع درخت اصلی یعنی $\lfloor \lg n \rfloor$ هستند.

○ درخت max-heap :

$\leq \left\lceil \frac{n}{2^{h+1}} \right\rceil$ تا node دارای ارتفاع h هستند



تعداد $\leq \left\lceil \frac{n}{2} \right\rceil$ تا node دارای ارتفاع $h = 0$

تعداد $\left\lceil \frac{n}{2^2} \right\rceil$ تا node دارای ارتفاع $h = 1$

⋮

تعداد $\left\lceil \frac{n}{2^{h_0+1}} \right\rceil$ تا node دارای ارتفاع $h = h_0$

⋮

تعداد ۱ عدد node دارای ارتفاع $\lfloor \lg n \rfloor$

ارتفاع درخت heap، برابر $\lfloor \lg n \rfloor$ است که همان ارتفاع root است

پس بیایید پیچیدگی زمانی BUILD-MAX-HEAP را دقیق‌تر بررسی کنیم

$$
\begin{array}{llll}
 & 1 & \times & \lfloor \lg n \rfloor \quad // \text{root} \\
+ & 2 & \times & \lfloor \lg n \rfloor - 1 \\
+ & 4 & \times & \lfloor \lg n \rfloor - 2 \\
+ & \cdots & & \\
+ & \lceil \frac{n}{4} \rceil & \times & 1 \\
+ & \leq \lceil \frac{n}{2} \rceil & \times & 0 \quad // \text{leaves} \\
\hline
\end{array}
$$

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^{h+1}} \right)$$

در order قابل حذف است

در فرمول‌های جبری داریم که $\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}$ به ازای $x = \frac{1}{2}$

یک سری است که نتیجه می‌دهد

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \sum_{h=0}^{\infty} h \left(\frac{1}{2}\right)^h = \frac{1/2}{(1-1/2)^2} = 2$$

پس:

$$O\left(n \sum_{n=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) = O(2n) = O(n)$$

پس، running time ی الگوریتم BUILD-MAX-HEAP، Linear است.

$O(n)$

The heap sort algorithm

ورودی: یک آرایه ورودی با n عضو داده شده است

heap sort این گونه اجرا می‌شود

- از روی آرایه یک max-heap می‌سازد.
- از root شروع می‌کنیم که (maximum element) است و آن را با استفاده از swapping در جای صحیح خودش قرار می‌دهیم

یعنی این عضو ماکزیمم که root است جایش را با last element عضو می‌کنیم،

- یک عضو از heap size کم می‌کنیم که کسی با این max کاری نداشته باشد. (Discard last node)
- حال MAX-HEAPIFY را call می‌کنیم تا بر روی ریشه جدید کار کند و درخت را درست کند و تبدیل به max-heap کند.
- فرآیند انتخاب ریشه root و swap کردن و discard کردن را ادامه می‌دهیم تا زمانی که فقط یک node که همان smallest element است باقی بماند
- آرایه مرتب شده است.

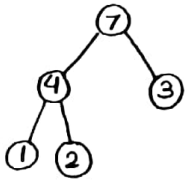
max-heap

7	4	3	1	2

↓

max

sorted array



swap (7, 2)

2	4	3	1	7

new size

↓

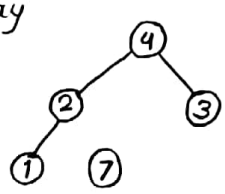
MAX-HEAPIFY(A, 1, [HEAP-SIZE] - 1)

↓

sorted array

4	2	3	1	7

new max-heap



↓

sorted array

SWAP (4, 1)

1	2	3	4	7

new size

↓

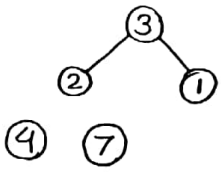
MAX-HEAPIFY(A, 1, [HEAP-SIZE] - 2)

↓

sorted

3	2	1	4	7

new max-heap



↓

sorted

SWAP (3, 1)

1	2	3	4	7

new size

↓

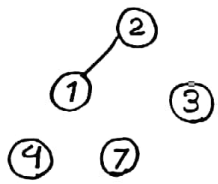
MAX-HEAPIFY(A, 1, $\overbrace{\text{[HEAP-SIZE]} - 3}^{2}$)

↓

sorted

2	1	3	4	7

new max-heap

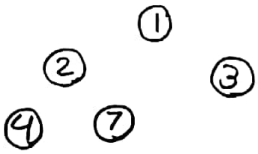


↓

SWAP (2, 1)

1	2	3	4	7

sorted



Pseudocode:

```
HEAPSORT(A, n)
BUILD-MAX-HEAP(A, n);
for i ← n downto 2
    do exchange A[1] ⟷ A[i]
       MAX-HEAPIFY(A, 1, i-1)
```

- n همان سایز آرایه length [A] است.
- i-1 در هر iteration نشان دهنده heap-size[A] است.

Analysis:

- BUILD_MAX_HEAP (A, n) : $O(n)$
- for loop : n-1 times
  - exchange elements $O(1)$
  - MAX-HEAPIFY: $O(\lg n)$

$$O(n) + (n-1)\left(O(\lg n) + O(1)\right)$$

Total time : $O(n \lg n)$

اگرچه heapsort یک الگوریتم عالی است ولی معمولا در practice
Quicksort در صورت well-implemented بودن از آن پیشی می‌گیرد

یادآوری Quicksort :

in the best case $\Theta(n \lg n)$

in the worst case $\Theta(n^2)$

heapsort :

in the worst case $O(n \lg n)$

finding the maximum element:

```
HEAP_MAXIMUM (A)
    return A[1]
```

فقط یک دستور return کافی است.

$\Theta(1)$

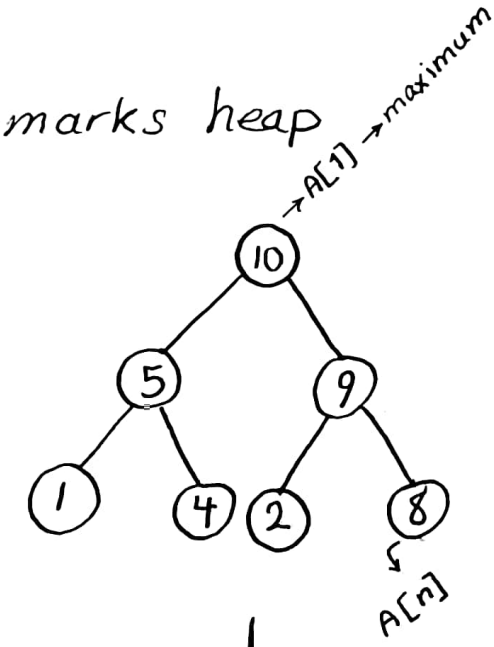
Extracting max element:

استخراج بزرگترین عنصر :

- بررسی کن که heap خالی نباشد.
- یک copy از بزرگترین عنصر که root است بگیر.
- lastnode را به عنوان root جدید قرار بده
- کل درخت را با یک عنصر کمتر max-heapify کن.
- یک copy از بزرگترین عنصر return کن

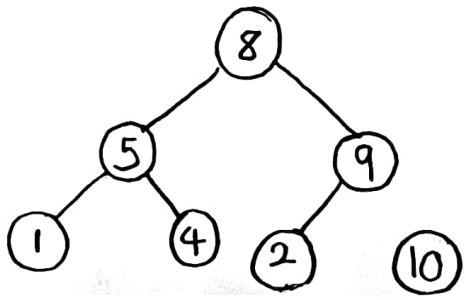
HEAP-EXTRACT-MAX (A, n)

```
1  if n < 1
2      then error "heap underflow"
3  max ← A[1]
4  A[1] ← A[n]
5  MAX-HEAPIFY(A, 1, n-1)   ▷ remarks heap
6  return max
```

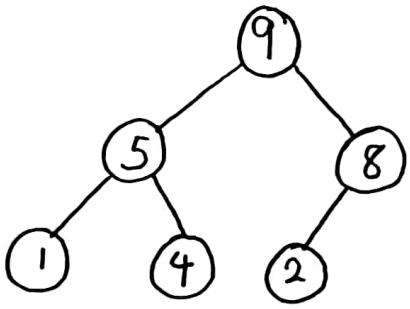


Analysis:

Lines	1 – 4	:	constant
Line	5	:	$O(\lg n)$
Line	6	:	constant
total		:	$O(\lg n)$



MAX-HEAPIFY(A, 1, 6)



output : 10

Increasing key value:

یک مجموعه A ، یک element ، x (به صورت شماره اندیس i) و یک مقدار داده جدید k به عنوان آرگومان می‌گیرد

set A

element i

new key value k

به شرطی مقدار A[i] را با k ، آپدیت می‌کند که k > A[i] باشد.

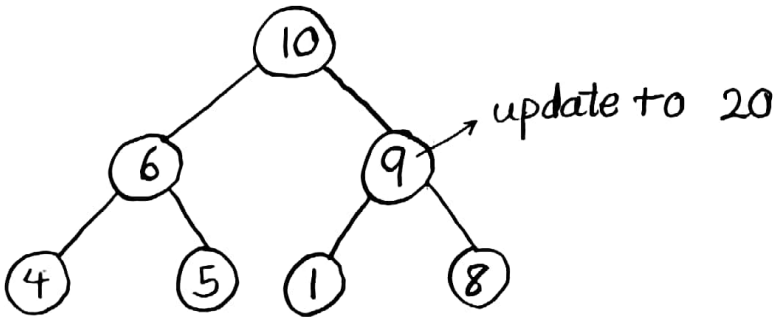
نکته: با تغییر مقدار داده یا همان key ی یکی از node های درخت، ممکن است ساختار max-heap به هم بخورد.

- چون مقدار key ی node دارای اندیس i بزرگتر از قبلش می‌شود.

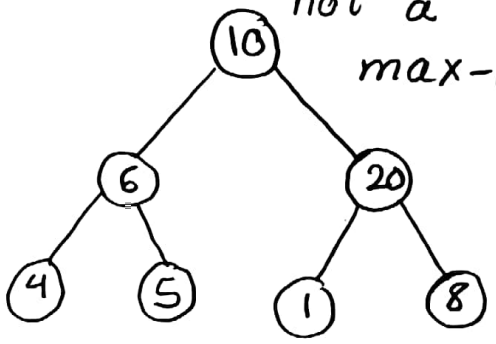
نیاز به فراخوانی MAX-HEAPIFY بر روی اندیس‌های نابزرگتر از i نداریم

و به جای آن داریم:

max-heap



not a max-heap

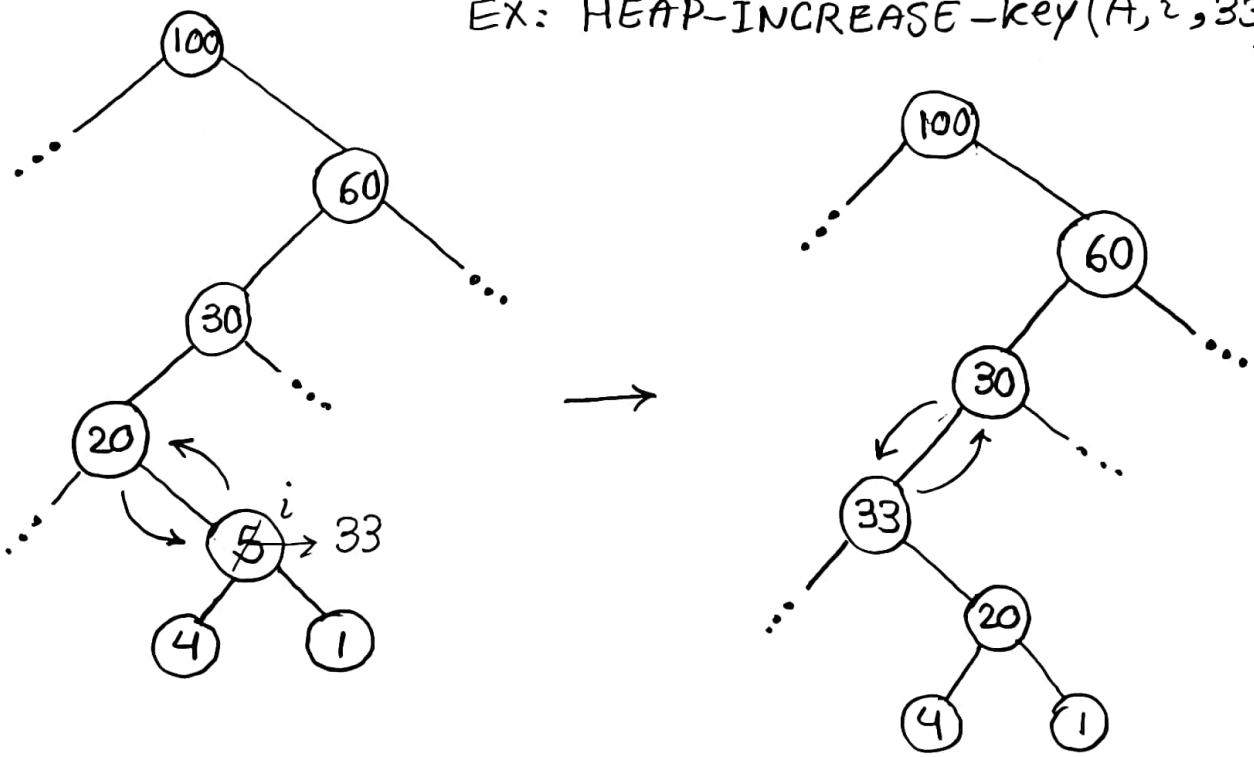


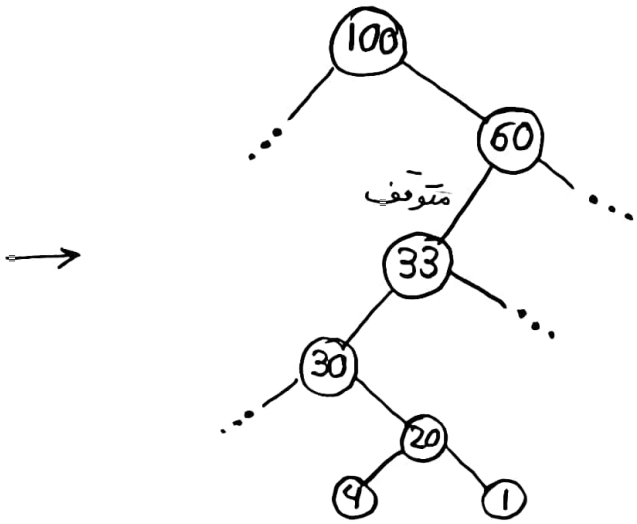
- ساختار زیر درختی که ریشه آن 20 است درست است

- کافی است که 20 با پدر، پدر پدر، پدر پدر پدر تا ریشه مقایسه شود در صورت لزوم تغییر و جابه جا شود.

```
HEAP-INCREASE-KEY (A, i, key)
if key < A[i]
    then error "new key is smaller than current key"
A[i] ← key
while i > 1 and A[parent(i)] < A[i]
    do exchange A[i] ↔ A[parent(i)]
        i ← parent(i)
```

EX: HEAP-INCREASE-KEY(A, i, 33)





آنچه برای insertion می‌خواستیم مهیا شده است

S نام tree

MAXIMUM (S)

EXTRACT_MAX (S)

INCREASE-KEY (S, x, k)

↓

INSERT (S, x)

Inserting into the heap:

یک key ی k برای insert به heap داده شده است

- یک node جدید در آخرین مکان heap در درخت با key برابر $-\infty$ درج کن.
- مقدار $-\infty$ را به k با استفاده از پروسیجر HEAP-INCREASE-KEY افزایش بده.

MAX-HEAP-INSERT (A, key, n)

1. $A[n+1] \leftarrow -\infty$
2. HEAP-INCREASE (A, n+1, key)

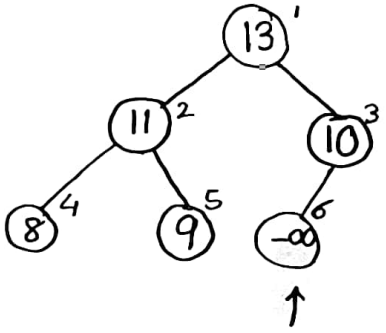
heap-size[A] در قبل از اجرای پروسیجر n است و پس از پایان به مقدار n+1 آپدیت می‌شود.

Analysis:

	cost	times
Line 1	constant	1
Line 2	$O(\lg n)$	1

total Time : $O(\lg n)$

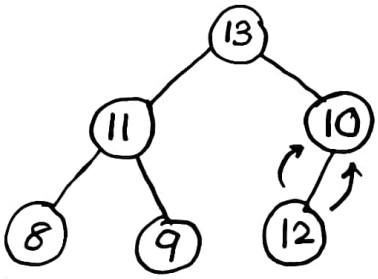
EX: MAX-HEAP-INSERT (A, 12)

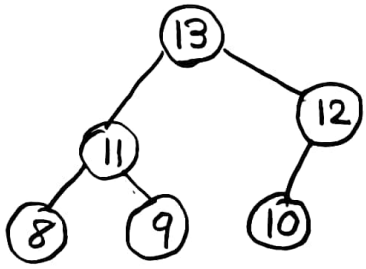


very last position

(almost complete, حفظ ساختار)

→ HEAP-INCREASE-KEY (A, 6, 12)





Heap implementation of priority queue:

priority queue یا صف اولویت:

- یک Data Structure پرکاربرد است operating Systems ها از آن برای مدیریت پردازش‌هایشان استفاده می‌کنند.

- در صف عادی (queue) از روش FIFO استفاده می‌شود، اولین داده‌ای که وارد شده است، ابتدا خارج می‌شود.

- در priority queue برای هر داده‌ی ورودی اولویت در نظر گرفته می‌شود، داده‌ای که اولویت آن از همه بیشتر باشد خارج می‌شود.

	9	1	7	5
priority :	3	2	2	1

output : 9 →
7
1
5

- صف معمولی، صف اولویت داری است که در آن اولویت عناصر به ترتیب اضافه شدنشان کاهش می‌یابد، پس عنصری که در اول لیست است، بالاترین اولویت را دارد.

ساختن یک درخت heap وارد کردن متوالی node ها در آن با عمل MAX-HEAP-INSERT (یا MIN-HEAP-INSERT) است. که برای وارد کردن یک node جدید به روش زیر عمل می کنیم:

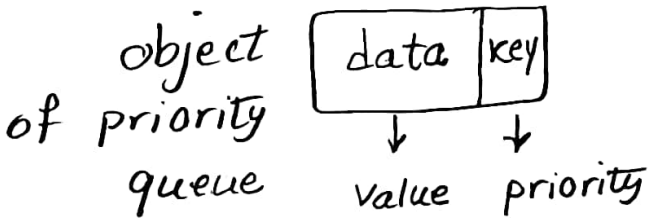
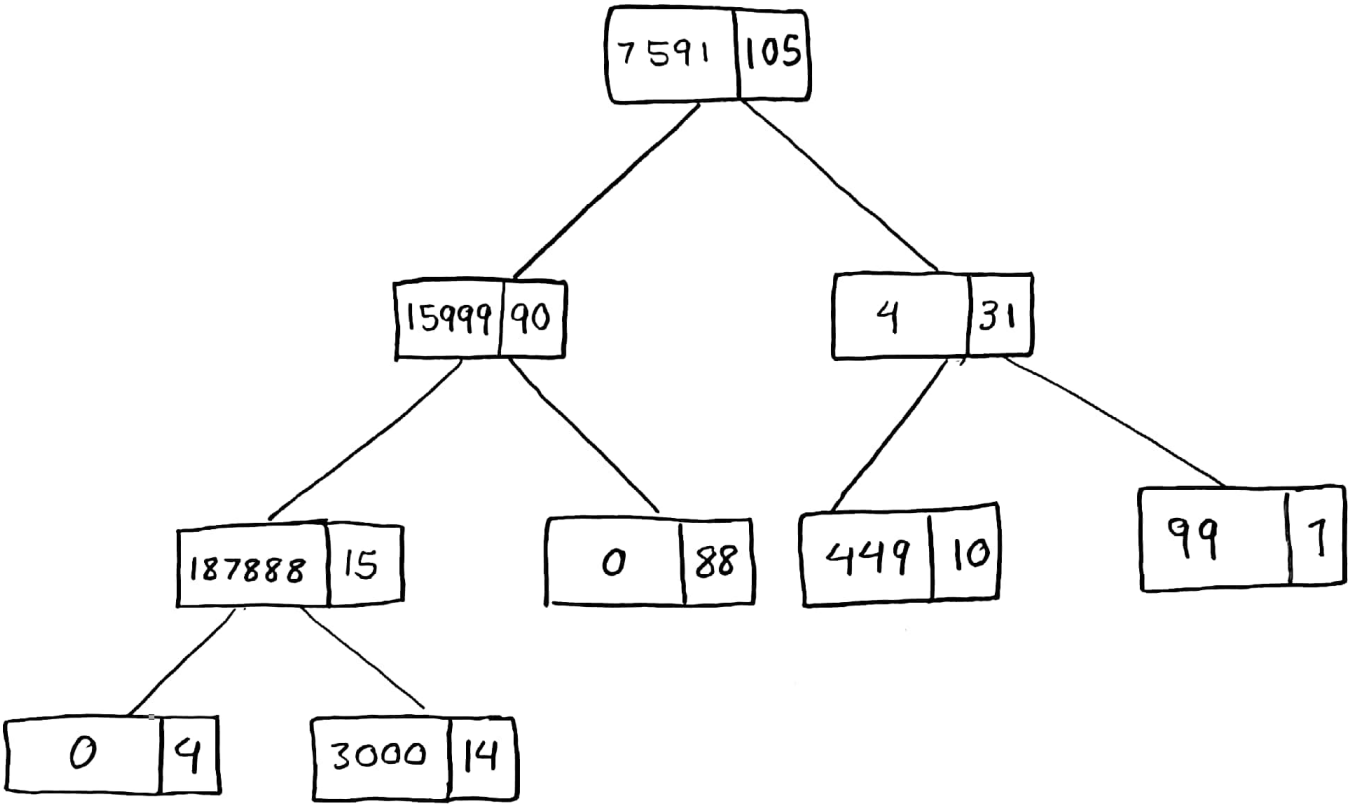
۱) node جدید را در محلی که در آخر درخت باشد و almost-complete بودن را بهم نزند درج می کنیم

۲) HEAP-INCREASE-KEY (و یا HEAP-DECREASE-KEY)، با توجه به اولویت (که در اینجا همان key ی node است) در روندی که max-heap (و یا min-heap) بودن بهم نخورد آن را در درخت بالاتر می برد و به آن اولویت می دهد.

۳) Extract-MAX (و یا Extract-MIN) همیشه عنصر با بیشترین اولویت را از صف dequeue می کند.

* از درخت MIN-HEAP هم به عنوان priority queue استفاده می شود، برای آن کافی است به جای اصطلاحات مربوط به MAX HEAP داخل پرانتزها را بخوانید.

نکته: key در HEAP نقش priority را دارد پس باید کنار هر node در درخت HEAP یک data واقعی هم باشد.



Example: