درخت دودویی جست‌وجو

از فصل ۱۲ کتاب CLRS

درخت دودویی جست‌وجو (تعریف)

- **درخت دودویی**

درخت دودویی جست‌وجو (تعریف)

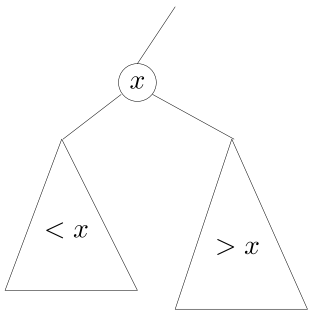
- **درخت دودویی**
- **برای هر گره n با برچسب x**

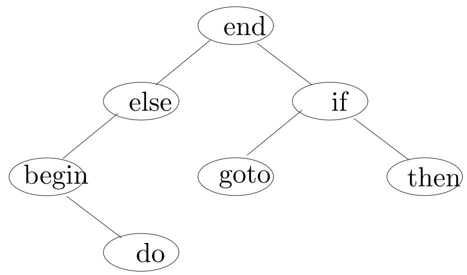
درخت دودویی جست‌وجو (تعریف)

- **درخت دودویی**
- **برای هر گره n با برچسب x**
  - **برچسب کلیه‌ی گره‌های زیردرخت چپ n کم‌تر از x و**

درخت دودویی جست‌وجو (تعریف)

- درخت دودویی
- برای هر گره n با برچسب x
  - برچسب کلیه‌ی گره‌های زیردرخت چپ n کم‌تر از x و
  - برچسب کلیه‌ی گره‌های زیردرخت راست آن بیش‌تر از x است

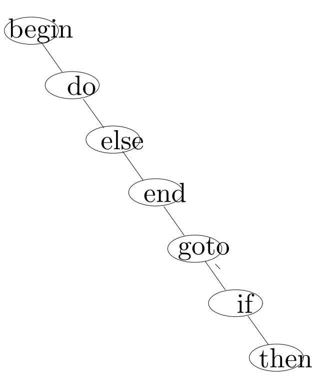


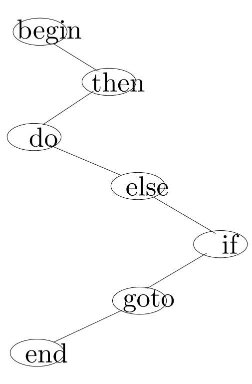


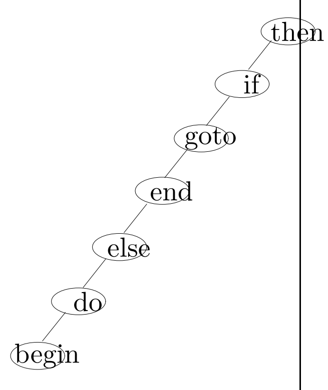
یک درخت دودویی جست‌وجو

پیمایش بین‌ترتیب $\longrightarrow$ مرتب‌شده‌ی عناصر

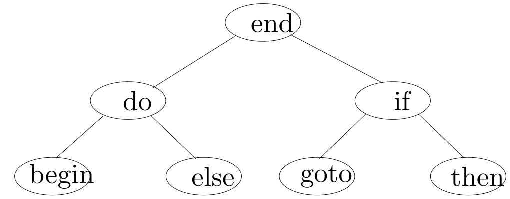
ترتیب درج در ایجاد درخت:







با بیش‌ترین ارتفاع.



با کم‌ترین ارتفاع.

$\lfloor \lg n \rfloor \le$ **ارتفاع** $\le n - 1$ **(چرا؟)**

متوسط ارتفاع: $O(\lg n)$

چه تعداد درخت دودویی جست‌وجو با ارتفاع $n - 1$؟

چه تعداد درخت دودویی جست‌وجو با ارتفاع $n - 1$؟

2^{n-1}

تعداد درخت‌های دودویی جست‌وجو

با $a_1 < a_2 < \cdots < a_n$ چند تا درخت متفاوت می‌توان ساخت؟

تعداد درخت‌های دودویی جست‌وجو

با $a_1 < a_2 < \cdots < a_n$ چند تا درخت متفاوت می‌توان ساخت؟

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} T(i-1)T(n-i), \\ T(0) &= 1 \end{aligned}$$

تعداد درخت‌های دودویی جست‌وجو

با $a_1 < a_2 < \cdots < a_n$ چند تا درخت متفاوت می‌توان ساخت؟

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} T(i-1)T(n-i), \\ T(0) &= 1 \end{aligned}$$

$$T(n) = \frac{1}{n+1}\binom{2n}{n}$$

عدد کاتالان

همین‌جا حل کنید!

اعداد $\{8, 1, 13, 14, 9, 22, 6\}$ را به درخت زیر طوری نسبت دهید که درخت دودوی
جست‌وجو شود.

```
          []
         /  \
      []      []
     /  \       \
   []    []      []
        /
      []
```

```
            [13]
           /    \
        [6]      [14]
       /   \         \
    [1]     [9]       [22]
           /
        [8]
```

این دنباله‌های درج در یک درخت تهی درخت فوق را می‌سازد

13, 6, 9, 1, 14, 8, 22

13, 14, 6, 22, 1, 9, 8

13, 6, 1, 9, 8, 14, 22

به‌چند حالت می‌توان اعداد فوق را وارد یک درخت تهی کرد تا در انتها درخت فوق حاصل شود؟ این مقدار را دقیقاً محاسبه کنید.

```
13,     6,   9,   1,    8
     ^            ^

     14           22
```


$$r \ \underbrace{\cup\cup\ldots\cup\cup}_{n_2} \ l_1 \ \underbrace{\cup\cup\ldots\cup\cup}_{n_2} \ l_2 \ldots \underbrace{\cup\cup\ldots\cup\cup}_{n_2} \ l_{n_1} \ \underbrace{\cup\cup\ldots\cup\cup}_{n_2}$$

اما واقعاً

$$r \ \cup \ l_1 \ \cup \ l_2 \ldots \cup \ l_{n_1} \ \cup$$

$$T(n) = T(n_1)T(n_2)\frac{(n_1+n_2)!}{n_1!n_2!}$$

n_1 is the number of nodes in the left subtree

n_2 is the number of nodes in the right subtree

جست‌وجو

BST-Search(r, x)

▷ r is a node (or the root) of a BST

if $r =$ **null** **or** $x = key[r]$

  then return r

if $x < key[r]$

  then return BST-Search$(left[r], x)$

  else **return** BST-Search$(right[r], x)$

جست‌وجو (غیر بازگشتی)

```
BST-Search(r, x)
    ▷ r is a node (or the root) of a BST
while r ≠ null  and x ≠ key[r]
        do


return r
```


جست‌وجو (غیر بازگشتی)

```
BST-Search(r, x)
    ▷ r is a node (or the root) of a BST
while r ≠ null  and x ≠ key[r]
        do if x < key[r]
              then r ← left[r]
              else  r ← right[r]
return r
```

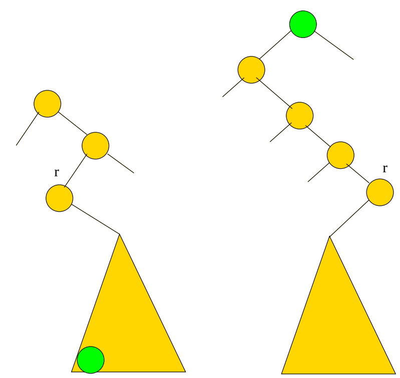

پیدا کردن عنصر کمینه

```
BST-Minimum(r)
if left[r] = null
   then return r BST-Minimum(left[r])
```


پیدا کردن کمینه (غیر بازگشتی)

```
BST-Minimum(r)
while left[r] ≠ null
        do r ← left[r]
return r
```

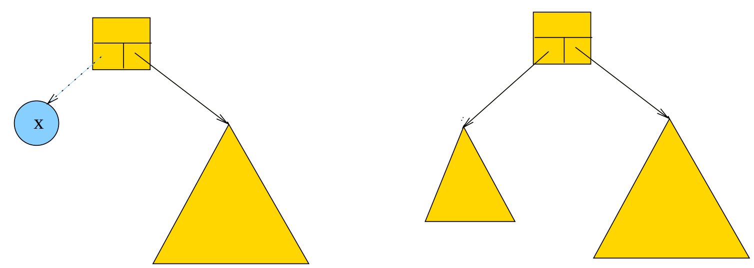
عنصر بعدی



BST-SUCCESSOR(r)

if $right[r] \neq$ **null**
  then return BST-MINIMUM($right[r]$)
$y \leftarrow parent[r]$
while $y \neq$ **null** **and** $r = right[y]$
    do $r \leftarrow y$
      $y \leftarrow parent[y]$
return y

درج



BST-INSERT(r, x)

  ▷ r is a node (or the root) of a BST
  ▷ important: r should be a reference parameter
  ▷ no $parent$ is assumed
if $r =$ **null**
  then $r \leftarrow$ ALLOCATE-NODE(x, **null** , **null**)
if $x < key[r]$
  then BST-INSERT($left[r], x$)
  else **if** $x > key[r]$
      then BST-INSERT($right[r], x$)

درج بازگشتی با مولفه‌ی پدر

درج بازگشتی با مولفه‌ی پدر

BST-Insert(r, p, x)

▷ r is a node (or the root) of a BST
▷ p is the parent of r
▷ important: r should be a reference parameter
▷ BST-Insert$(Root[T]$, **null** $, x)$:فراخوانی اولیه
if $r =$ **null**
  then $r \leftarrow$ Allocate-Node(x, **null** , **null** , p)
if $x < key[r]$
  then BST-Insert$(left[r], r, x)$
  else **if** $x > key[r]$
    then BST-Insert$(right[r], r, x)$

درج غیر بازگشتی

درج غیر بازگشتی

BST-Insert(T, x)

$n \leftarrow$ Allocate-Node(x, **null** , **null** , **null**)
$prevp \leftarrow$ **null**
$p \leftarrow root[T]$
while $p \neq$ **null**
  do $prevp \leftarrow p$
    if $x < key[p]$
      then $p \leftarrow left[p]$
      else $p \leftarrow right[p]$
$parent[n] \leftarrow prevp$

```
if prevp = null
  then root[T] ← n
  else  if x < key[prevp]
           then left[prevp] ← n
           else  right[prevp] ← n
```


حذف کوچک‌ترین عنصر

حذف کوچک‌ترین عنصر

```
BST-DELETEMIN(r)
    ▷ r is a node (or the root) of a BST
    ▷ r is assumed to be a reference variable
if r = null
  then  error ("Tree is Empty")
if left[r] = null
  then x ← label[r]
       t ← r
       r ← right[r]
       parent[r] ← parent[t]
       FREE-NODE (t)
       return x
  else  return BST-DELETEMIN(left[r])
```


حذف یک عنصر

```
BST-DELETE(r, x)
    ▷ r is a node (or the root) of a BST
    ▷ r is a reference variable
if r = null
  then  error ("Tree is Empty")
if x < label[r]
  then BST-DELETE(left[r], x)
if x > label[r]
  then BST-DELETE(right[r], x)
if x = label[r]
  then
```



```
BST-DELETE(r, x)
    ⋮
if x = label[r]
  then temp ← r
       if left[r] = null
         then r ← right[r]
              parent[r] ← parent[temp]
              FREEE-NODE(temp)
         else if right[r] = null
                then r ← left[r]
                     parent[r] ← parent[temp]
                     FREEE-NODE(temp)
                else label[r] ← BST-DELETEMIN (right[r])
```


همین‌جا حل کنید!

آیا با داشتن دنباله‌ی بین‌ترتیب از عناصر یک ددج می‌توان آن‌را به‌صورت تک ساخت؟

با پس‌ترتیب چه‌طور؟

با پیش‌ترتیب چه‌طور؟

متوسط ارتفاع درخت دودویی جست‌وجوی

$a_1 < a_2 \cdots < a_n$ به‌صورت تصادفی و با احتمال یکسان وارد یک درخت دودویی جست‌وجوی تهی T می‌شوند.

اثبات می‌کنیم که متوسط ارتفاع T برابر $O(\lg n)$ است.

- فرض کنید a_i اولین عنصری است که درج می‌شود.
- a_i ریشه‌ی درخت خواهد بود.
- a_1 تا a_{i-1} به‌هر ترتیبی که درج شوند در زیردرخت چپ قرار خواهند گرفت.
- a_{i+1} تا a_n در زیر درخت راست خواهند بود.

- اگر $h(n)$ متوسط ارتفاع T باشد، داریم:

$$h(n) = 1 + \sum_{i=1}^{n} \frac{1}{n} \max\{h(i-1), h(n-i)\}$$

- فرض می‌کنیم $h(n) \leq c \lg n$، برای یک $c > 0$

- با توجه به این‌که $h(i)$ یک تابع غیر نزولی است،

$$h(n) = 1 + \frac{1}{n}\left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} h(n-i) + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1)\right)$$

- با توجه به این‌که $h(i)$ یک تابع غیر نزولی است،

$$\begin{aligned} h(n) &= 1 + \frac{1}{n}\left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} h(n-i) + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1)\right) \\ &\leq 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1) \end{aligned}$$

- با توجه به این‌که $h(i)$ یک تابع غیر نزولی است،

$$\begin{aligned} h(n) &= 1 + \frac{1}{n}\left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} h(n-i) + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1)\right) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n-1} h(i) \end{aligned}$$

- با توجه به این‌که $h(i)$ یک تابع غیر نزولی است،

$$\begin{aligned} h(n) &= 1 + \frac{1}{n}\left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} h(n-i) + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1)\right) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n-1} h(i) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n} c \lg i \end{aligned}$$

- با توجه به این‌که $h(i)$ یک تابع غیر نزولی است،

$$\begin{aligned} h(n) &= 1 + \frac{1}{n}\left(\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} h(n-i) + \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1)\right) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} h(i-1) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n-1} h(i) \\ &\le 1 + \frac{2}{n} \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n} c \lg i \\ &\le 1 + \frac{2c}{n} \lg \frac{(n)!}{(\frac{n}{2})!} \\ &\le 1 + \frac{2c}{n} [\lg n! - \lg(\frac{n}{2})!] \end{aligned}$$

- با توجه به این که $\lg n! = \Theta(n \lg n)$ یعنی $k_1 n \lg n \le \lg n! \le k_2 n \lg n$،

$$\begin{aligned} h(n) &\le 1 + \frac{2c}{n}\left(k_2 n \lg n - k_1 \frac{n}{2} \lg \frac{n}{2}\right) \\ &\le 1 + 2c(k_2 - \frac{k_1}{2}) \lg n - 2ck_1 \end{aligned}$$

می‌توان $0 < k_1 < k_2$ را پیدا کرد که $k_2 - \frac{k_1}{2} < 1/2$ و $1 - 2ck_1 \le 0$

پس

$$h(n) \le c \lg n$$

صف اولویت (Priority Queue)

از بخش ۶/۵ کتاب CLRS

داده‌ساختاری برای اعمال

- درج
- حذف بزرگ‌ترین (کوچک‌ترین) عنصر
- افزایش (کاهش) مقدار کلید یک عنصر

همه در $O(\lg n)$

صف اولویت (تعریف)

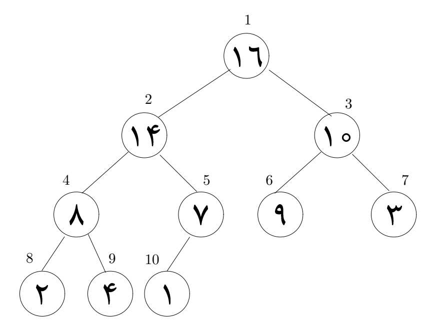
یک درخت دودویی کامل (complete binary tree) (به‌جز حداکثر یک عنصر با یک
فرزند)

برگ‌های سطح آخر آن از سمت چپ چیده شده‌اند.

کلید هر عنصر از کلیدهای فرزندانش کوچک‌تر نیست.

به این داده‌ساختار درخت نیمه‌مرتب (Partially Ordered Tree)، max-heap، ی
max-priority queue نیز می‌گویند

متناظر با آن min-heap است.



1	2	3	4	5	6	7	8	9	10
۱۶	۱۴	۱۰	۸	۷	۹	۳	۲	۴	۱

یک صف اولویت

max-heap (ویژگی‌ها)

- ریشه بزرگ‌ترین عنصر است.
- ارتفاع یک درخت max-heap با n عنصر $\lfloor \lg n \rfloor$ می‌باشد.
- اعمال درج، حذف بزرگ‌ترین و افزایش کلید از مرتبه‌ی $\lg n$ انجام می‌شود

min-heap (ویژگی‌ها)

- ریشه کوچک‌ترین عنصر است.
- ارتفاع یک درخت min-heap با n عنصر $\lfloor \lg n \rfloor$ می‌باشد.
- اعمال درج، حذف کوچک‌ترین و کاهش کلید از مرتبه‌ی $\lg n$ انجام می‌شود

پیاده‌سازی max-heap

- آرایه‌ی `A[1..n]`
- ریشه در `A[1]`
- فرزند چپ عنصر iام در `A[2i]` (اگر $۲i \leq n$)
- فرزند راست آن در `A[2i+1]` (اگر $۲i + ۱ \leq n$)
- پدرش در `A`$[\lfloor \frac{i}{2} \rfloor]$

Parent(i)
  return $\lfloor \frac{i}{2} \rfloor$

LeftChild(i)
  return $2i$

RightChild(i)
  return $2i + 1$

افزایش کلید

Max-Heap-Increase-Key(A, i, key)
- **if** $key < A[i]$
-   **then** **error** “new key is smaller than current key”
- $A[i] \leftarrow key$
- **while** $i > 1$ **and** $A[i] > A[\text{Parent}(i)]$
-     **do** swap($A[i], A[\text{Parent}(i)]$)
-       $i \leftarrow$ Parent(i)]

درج

MAX-HEAP-INSERT(A, key)

$length[A] \leftarrow length[A] + 1$

$A[length[A]] \leftarrow -\infty$

MAX-HEAP-INCREASE-KEY$(A, length[A], key)$

به‌صورت max-heap در آوردن $A[i..length[A]]$

به‌صورت max-heap در آوردن $A[i..length[A]]$

MAX-HEAPIFY(A, i)

$l \leftarrow$ LEFTCHILD (i)

$r \leftarrow$ RIGHTCHILD (i)

if $l \leq length[A]$ **and** $A[l] > A[i]$

  then $bigchild \leftarrow l$

  else $bigchild \leftarrow i$

if $r \leq length[A]$ **and** $A[r] > A[bigchild]$

  then $bigchild \leftarrow r$

if $bigchild \neq i$

  then swap$(A[i], A[bigchild])$

    MAX-HEAPIFY $(A, bigchild)$

تبدیل یک آرایه به max-heap

تبدیل یک آرایه به max-heap

BUILD-HEAP(A)
  for $i \leftarrow \lfloor \frac{length[A]}{2} \rfloor$ downto 1
      do HEAPIFY(A, i)

حذف بزرگ‌ترین عنصر در max-heap

حذف بزرگ‌ترین عنصر در max-heap

HEAP-DELETE-MAX(A)
  if $length[A] < 1$
    then **error** "heap underflow"
  $max \leftarrow A[1]$
  $A[1] \leftarrow A[length[A]]$
  $A[length] \leftarrow A[length] - 1$
  MAX-HEAPIFY($A, 1$)
  return max

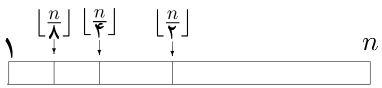
مثال

```
 1  2  3  4  5  6  7  8  9 10
--+--+--+--+--+--+--+--+--+--+
18 18 16  9  7  1  9  3  7  5  initial heap

5  18 16  9  7  1  9  3  7     deletemax

18  5 16  9  7  1  9  3  7     after heapify (A,1)

18  9 16  5  7  1  9  3  7     after heapify (A,2)

18  9 16  7  7  1  9  3  5     after heapify (A,4)

1  2  3  4  5  6  7  8  9  10  11
--+--+--+--+--+--+--+--+--+---+---+
18 18 16  9  7  1  9  3  7   5  13  after Insert 13
18 18 16  9 13  1  9  3  7   5   7
```

تحلیل

همه‌ی اعمال به‌جز Build-Heap متناسب با ارتفاع heap و از $O(\lg n)$

تحلیل Build-Heap



$$S(i)=\begin{cases} ۱ & i=\lfloor n/۲^۲\rfloor+۱\ldots\lfloor n/۲\rfloor \quad \text{تعداد} = \lfloor n/۲^۲\rfloor \\ ۲ & i=\lfloor n/۲^۳\rfloor+۱\ldots\lfloor n/۲^۲\rfloor \quad \text{تعداد} = \lfloor n/۲^۳\rfloor \\ \ldots & \ldots \\ k & i=\lfloor n/۲^{k+۱}\rfloor+۱\ldots\lfloor n/۲^k\rfloor \quad \text{تعداد} = \lfloor n/۲^{k+۱}\rfloor \\ \ldots & \ldots \\ \lfloor \lg n\rfloor & i=۱ \quad \text{تعداد} = ۱ \end{cases}$$

$$T(n)\leq \sum_{k=۱}^{\lfloor \lg n\rfloor} k\frac{n}{۲^{k+۱}}$$

$$\sum_{i=۱}^{\infty} i/۲^i=(۱/۲)+(۱/۴+۱/۴)+(۱/۸+۱/۸+۱/۸)+...=۲$$

$$\begin{aligned} ۱/۲+۱/۴+۱/۸+۱/۱۶+\cdots &= ۱ \\ ۱/۴+۱/۸+۱/۱۶+\cdots &= ۱/۲ \\ ۱/۸+۱/۱۶+\cdots &= ۱/۸ \\ \cdots &= .. \end{aligned}$$

پس ساخت heap از $O(n)$ است.

همین‌جا حل کنید!

k عدد کوچک‌ترین عناصر n عنصر را می‌خواهیم به‌ترتیب به‌دست آوریم.

همین‌جا حل کنید!

برای یک آرایه به‌صورت heap کارهای زیر را با چه مرتبه‌ای به‌صورت کارا می‌توا
انجام داد؟

- جمع همه‌ی اعداد:
- جمع تعداد $\lg n$ بزرگ‌ترین عدد:
- جمع ۱۰ عدد بزرگ:

همین‌جا حل کنید!

برای یک آرایه به‌صورت heap کارهای زیر را با چه مرتبه‌ای به‌صورت کارا می‌توان انجام داد؟

- جمع همه‌ی اعداد: n
- جمع تعداد $\lg n$ بزرگ‌ترین عدد: در $\lg^{۲} n$
- جمع ۱۰ عدد بزرگ: $O(۱)$

heapsort

```
BUILD-HEAP(A)
for i ← ⌊length[A]/2⌋ downto 1
    do HEAPIFY(A, i)

HEAPSORT(A)
BUILD-HEAP(A)
for i ← length[A] downto 2
    do swap(A[1], A[length[A]])
       length[A] ← length[A] − 1
       HEAPIFY (A, 1)
```

```
heapify (A,i)

  1   2   3   4   5   6   7   8   9  10
+---+---+---+---+---+---+---+---+---+---+
  3   6   1   9   7  18   9  18  5   7   initial array
  3   6   1   9   7  18   9  18  5   7   i:=5

  3   6   1  18   7  18   9   9   5   7   i:=4
             --                  --
  3   6  18  18   7   1   9   9   5   7   i:=3
         --          --
  3  18  18   6   7   1   9   9   5   7   i:=2
     --      --
  3  18  18   9   7   1   6   9   5   7
             --          --
 18   3  18   9   7   1   6   9   5   7   i:=1
 --  --
 18   9  18   3   7   1   6   9   5   7
     --      --
 18   9  18   9   7   1   6   3   5   7   heap
             --                  --
```



```
  1   2   3   4   5   6   7   8   9  10
--+---+---+---+---+---+---+---+---+---+
18   9  18   9   7   1   6   3   5   7          heap

 7   9  18   9   7   1   6   3   5|  18         swap (1,10)
18   .   7   .   .   .   .   .   .|

 5   9   7   9   7   1   6   3  |18  18         swap (1,9)
 9   5   .   .   .   .   .   .
 .   9   .   5   .   .   .   .

 3   9   7   5   7   1   6  |9   18  18         swap(1,8)
 9   3   .   .   .   .   .
 .   7   3   .   .   .   .
 .   .   6   .   .   .   3

 3   7   6   5   7   1  |9   9   18  18         swap(1,7)
 7   3   .   .   .   .
 7   6   3   .   .   .
```



```
  1   6   3   5   7  |7   9   9   18  18          swap(1,6)
  6   1   .   .   .
  .   7   .   .   1

  1   7   3   5  |6   7   9   9   18  18          swap(1,5)
  7   1   .   .
  .   5   .   1

  1   5   3  |7   6   7   9   9   18  18          swap(1,4)
  5   1   .

  3   1  |5   7   6   7   9   9   18  18          swap(1,3)
  1  |3   5   7   6   7   9   9   18  18          swap(1,2)

 |1   3   5   7   6   7   9   9   18  18          sorted
```