PRIORITY QUEUES (HEAPS)

—— **delete the element with the highest \ lowest priority**

§1 ADT Model

Objects: **A finite ordered list with zero or more elements.**

Operations:

- ☞ PriorityQueue Initialize(int MaxElements);
- ☞ void ✸Insert(ElementType X, PriorityQueue H);
- ☞ ElementType ✸DeleteMin(PriorityQueue H);
- ☞ ElementType FindMin(PriorityQueue H);

§2 Simple Implementations

- **Array :**

  Insertion — add one item at the end ~ $\Theta(1)$

  Deletion — find the largest \ smallest key ~ $\Theta(n)$

  remove the item and shift array ~ $O(n)$

- **Linked List :**

  Ins tion — add to the front of the chain ~ $\Theta(1)$

  Deleti — find the largest \ smallest key ~ $\Theta(n)$

  move the item ~ $\Theta(1)$

- **Ordered Array**

  Insertion — ition ~ $O(n)$

  $n)$

Better since there are never
more deletions than insertions

- **Ordered**

  Insertion — find the proper position ~ $O(n)$

  add the item ~ $\Theta(1)$

  Deletion — remove the first \ last item ~ $\Theta(1)$

✍ Binary Search Tree :

better option?

Now you begin to know me ☺

Be

always dangerous.

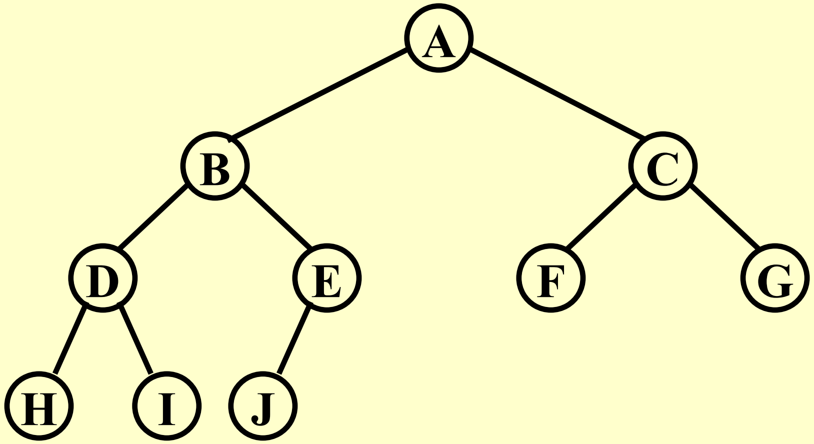
§3 Binary Heap

1. Structure Property:

【 **Definition** 】 A binary tree with n nodes and height h is complete iff its nodes correspond to the nodes numbered from 1 to n in the perfect binary tree of height h.

A complete binary tree of height h has between 2^h and $2^{h+1} - 1$ nodes. ➡ $h = \lfloor \log N \rfloor$

❖ Array Representation : BT [n + 1] (BT [0] is not used)



BT

0	1	2	3	4	5	6
	A	B	C	D	E	F

7	8	9	10	11	12	13
G	H	I	J			

【**Lemma**】 **If a complete binary tree with *n* nodes is represented sequentially, then for any node with index *i*, 1 ≤ *i* ≤ *n*, we have:**

$$\text{(1) index of } \mathit{parent}(i) = \begin{cases} \lfloor i/2 \rfloor & \text{if } i \neq 1 \\ \text{None} & \text{if } i = 1 \end{cases}$$

$$\text{(2) index of } \mathit{left\_child}(i) = \begin{cases} 2i & \text{if } 2i \leq n \\ \text{None} & \text{if } 2i > n \end{cases}$$

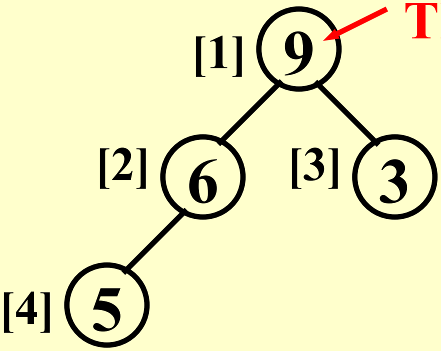
$$\text{(3) index of } \mathit{right\_child}(i) = \begin{cases} 2i+1 & \text{if } 2i+1 \leq n \\ \text{None} & \text{if } 2i+1 > n \end{cases}$$

```
PriorityQueue  Initialize( int  MaxElements )
{
     PriorityQueue  H;
     if ( MaxElements < MinPQSize )
              return  Error( "Priority queue size is too small" );
     H = malloc( sizeof ( struct HeapStruct ) );
     if ( H ==NULL )
              return  FatalError( "Out of space!!!" );
     /* Allocate the array plus one extra for sentinel */
     H->Elements = malloc(( MaxElements + 1 ) * sizeof( ElementType ));
     if ( H->Elements == NULL )
              return  FatalError( "Out of space!!!" );
     H->Capacity = MaxElements;
     H->Size = 0;
     H->Elements[ 0 ] = MinData;  /* set the sentinel */
     return  H;
}
```

2. Heap Order Property:

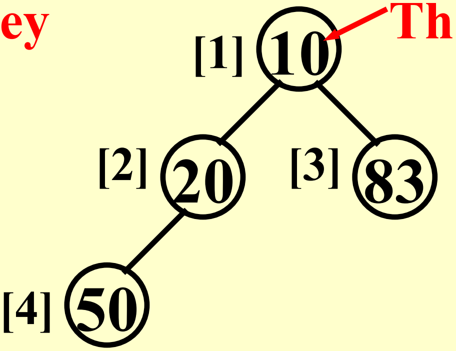
【Definition】 A **min tree** is a tree in which the key value in each node is no larger than the key values in its children (if any). A **min heap** is a **complete** binary tree that is also a min tree.

Note: Analogously, we can declare a ***max*** heap by changing the heap order property.



The largest key

A max heap



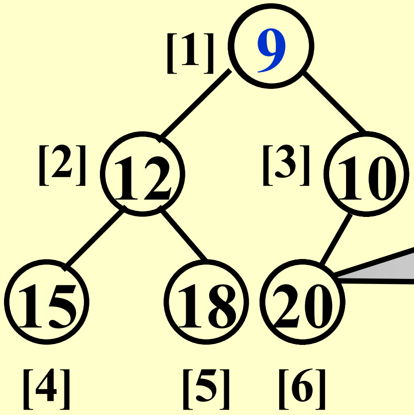
The smallest key

A min heap

3. Basic Heap Operations:

☞ **insertion**

➢ **Sketch of the idea:**



The only possible position for a new node since a heap must be a complete binary tree.

Case 1 : new_item = 21 ⑳ < ㉑ ✓

Case 2 : new_item = 17 ⑳ > ⑰ ⑩ < ⑰ ✓

Case 3 : new_item = 9 ⑳ > ⑨ ⑩ > ⑨ ✓

```
/* H->Element[ 0 ] is a sentinel */
void  Insert( ElementType  X,  PriorityQueue  H )
{
     int  i;

     if ( IsFull( H ) ) {
          Error( "Priority queue is ful
          return;
     }

     for ( i = ++H->Size; H->Elements[ i / 2 ] > X; i /= 2 )
          H->Elements[ i ] = H->Elements[ i / 2 ];

     H->Elements[ i ] = X;
}
```

H->Element[0] is a *sentinel* that is no larger than the minimum element in the heap.

Faster than *swap*

$T(N) = O(\log N)$

☞ DeleteMin

➢ **Sketch of the idea:**

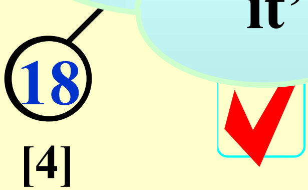
Ah! That's simple -- we only have to delete the root node ...

And re-arrange the rest of the tree so that it's still a min heap.

l to keep a

hild of 18

inary tree.



12 < 18

15 < 18

$$T(N) = O(\log N)$$

```
ElementType  DeleteMin( PriorityQueue  H )
{
    int  i, Child;
    ElementType  MinElement, LastElement;
    if ( IsEmpty( H ) ) {
        Error( "Priority queue is empty" );
        return  H->Elements[ 0 ];   }
    MinElement = H->Elements[ 1 ];  /*    e the min el    ent */
    LastElement = H->Elements[ H   Size-- ];  /* take l    and reset size */
    for ( i = 1; i * 2 <= H->Size; i = Child ) {  /* Find sm  ller child */
        Child = i * 2;
        if (Child != H->Size && H->Elements[Child+  ] < H->Elements[Child])
                Child++;
        if ( LastElement > H->Elements[ Child ] )   /* Percolate one level */
                H->Elements[ i ] = H->Elements[ Child ];
        else     break;   /* find the proper position */
    }
    H->Elements[ i ] = LastElement;
    return  MinElement;
}
```

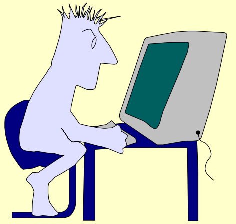
Can we remove it
by adding another
sentinel?

4. Other Heap Operations:

Note: Finding any key except the minimum one will have to take a linear scan through the entire heap.

☞ DecreaseKey (P, Δ, H)

Percolate up

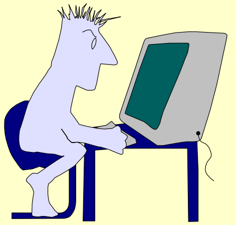


sys. admin.

Lower the value of the key in the heap H at position P by a positive amount of Δ……so my programs can run with highest priority ☺.

☞ IncreaseKey (P, Δ, H)

Percolate down

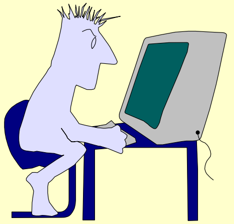


sys. admin.

Increases the value of the key in the heap H at position P by a positive amount of Δ……drop the priority of a process that is consuming excessive CPU time.

☞ Delete (P, H)

DecreaseKey(P, ∞, H); DeleteMin(H)



sys. admin.

Remove the node at position P from the heap H …… delete the process that is terminated (abnormally) by a user.

☞ BuildHeap (H)

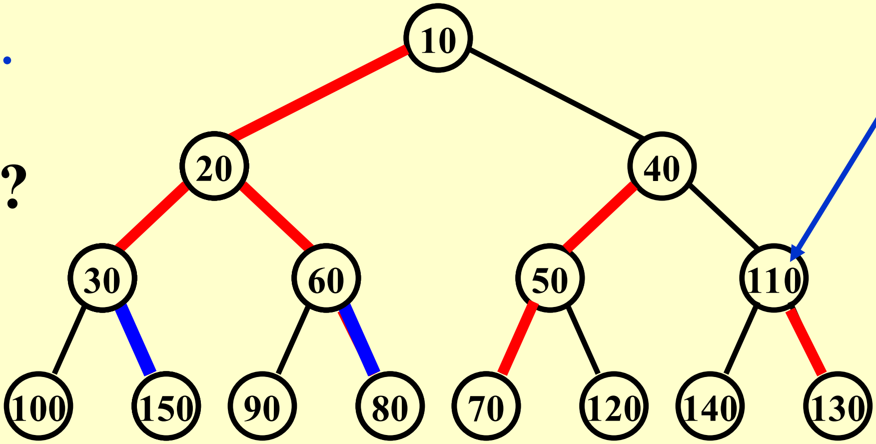
Nehhhhh that would be toooo slow !

sys. admin.

Place N input keys into an empty heap H.

150, 80, 40, 30, 10, 70, 110, 100, 20, 90, 60, 50, 120, 140, 130

$T(N) = ?$



PercolateDown (7)
PercolateDown (6)
PercolateDown (5)
PercolateDown (4)
PercolateDown (3)
PercolateDown (2)
PercolateDown (1)

【Theorem】 For the perfect binary tree of height h containing $2^{h+1} - 1$ nodes, the sum of the heights of the nodes is $2^{h+1} - 1 - (h + 1)$.

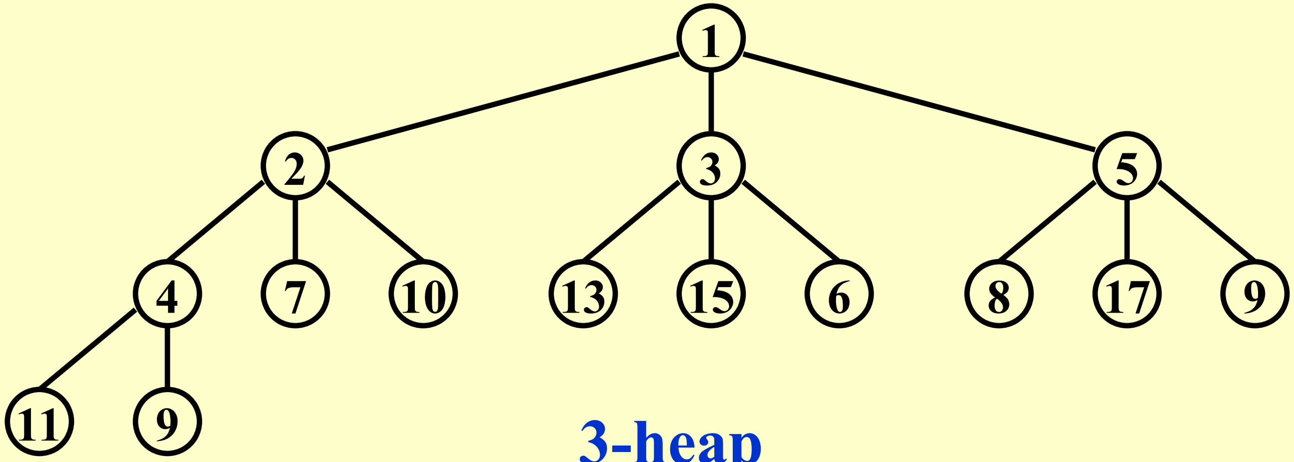
$$\longrightarrow T(N) = O(N)$$

§4 Applications of Priority Queues

〖Example〗 Given a list of N elements and an integer k. Find the kth largest element.

How many methods can you think of to solve this problem? What are their complexities?

§5 *d*-Heaps ---- All nodes have *d* children



3-heap

Question: Shall we make *d* as large as possible?

Note: ① DeleteMin will take $d - 1$ comparisons to find the smallest child. Hence the total time complexity would be $O(d \log_d N)$.

② *2 or /2 is merely a bit shift, but *d or /d is not.

③ When the priority queue is too large to fit entirely in main memory, a *d*-heap will become interesting.