Priority Queues

Rupesh Nasre.
rupesh@iitm.ac.in

July 2019

Applications

- Ambulance in traffic
- Customer buying one chocolate amid others buying loads of grocery
- Processes in OS
- Physically disabled person in lift / bus / aeroplane
- JEE admissions
- ...

If all priorities are the same, a queue suffices.
When priorities differ, we need a priority queue.

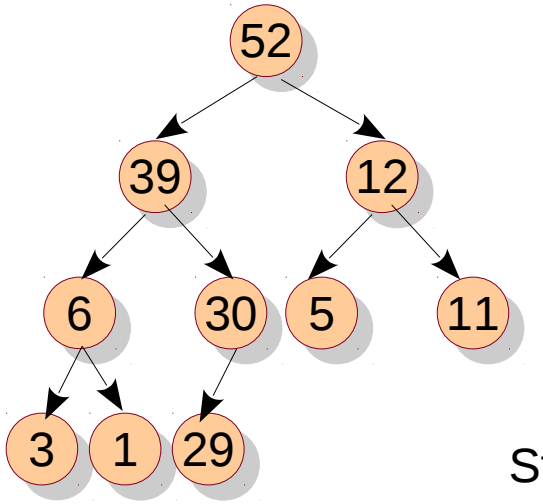
Priority Queue

- Data structure contains <key, value> pairs.
- Key is the priority, value is the associated data.
- Sometimes, only <key> is inserted.
- At any point, an application should be able to retrieve the element with the maximum priority.
- Multiple ways:
  - Linked list (queue): insert = O(1), findMax = O(N)
  - Sorted list: insert = O(N), findMax = O(1)
  - BST: insert = O(N), findMax = O(1)
  - AVL: insert = O(log N), findMax = O(1)
  - **Heap**: insert = O(log N), findMax = O(1), efficient

Heap

- By default, a binary heap.
- Structural Property: Complete tree
  - Only the last level may not be full
  - The last level must have elements pushed to the left
  - Has O(log N) height for N elements
- Heap Order Property: Child-key < Parent-key
  - Assumes unique elements
  - Opposite property in MinHeap
- NOT implemented using pointers!
- Implemented using arrays

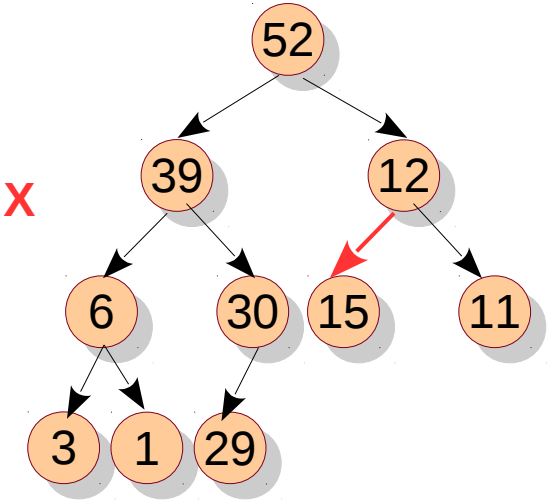
MaxHeap Example



	52	39	12	6	30	5	11	3	1	29
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

Stores elements level-wise, left-to-right.
For element at index i

- Parent is present at i / 2 (floor)
- Children are present at 2i and 2i + 1



Elements can be ordered only if one is reachable from the other.

- Otherwise, they cannot be ordered (e.g., 30 and 5).

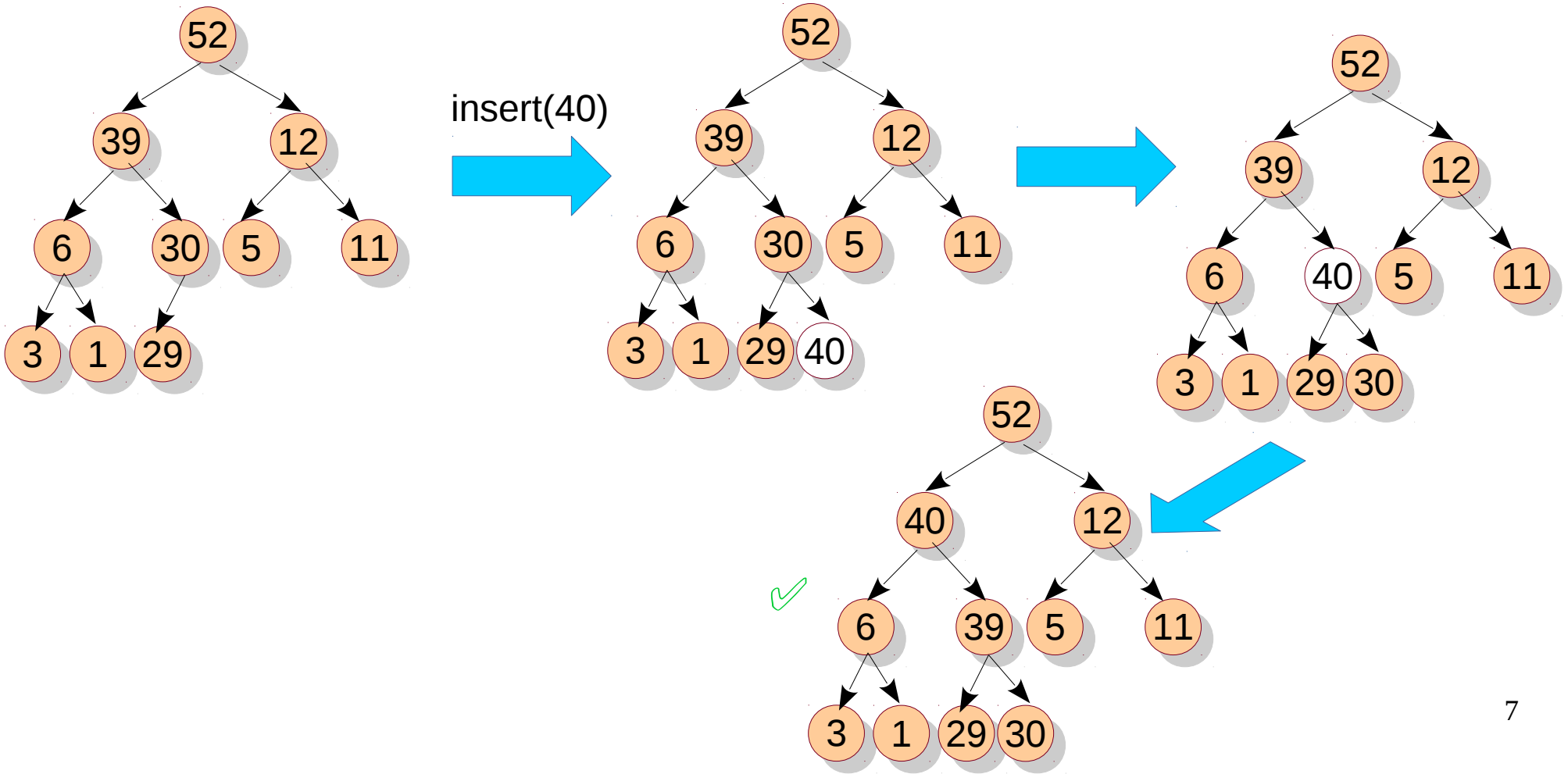
Need to depend upon a fixed array length.

MaxHeap ADT

```
class MaxHeap {
public:
    insert(e);
    deleteMax();

    increaseKey(e);
    decreaseKey(e);
    remove(e);
    buildHeap();
};
```

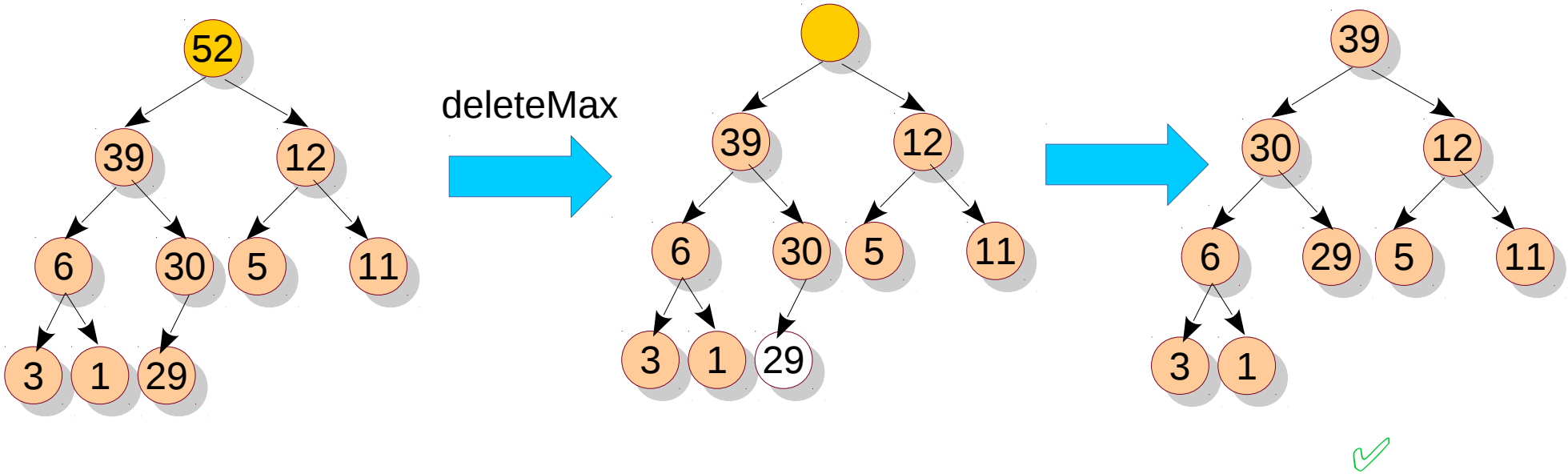
insert

- Insert at the first empty leaf.
- Percolate the element upward



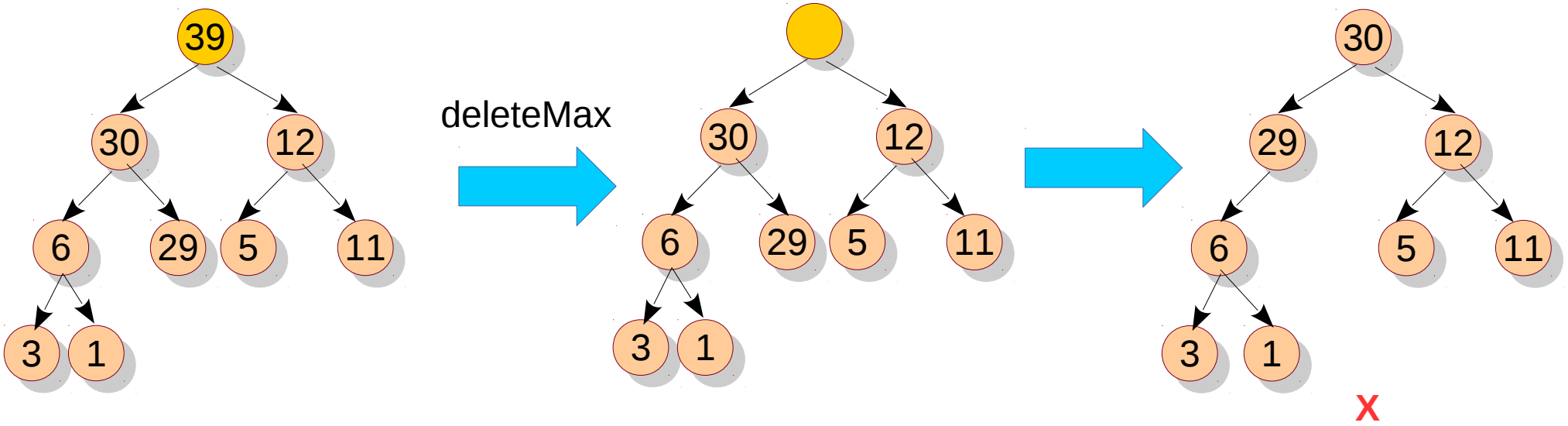
deleteMax

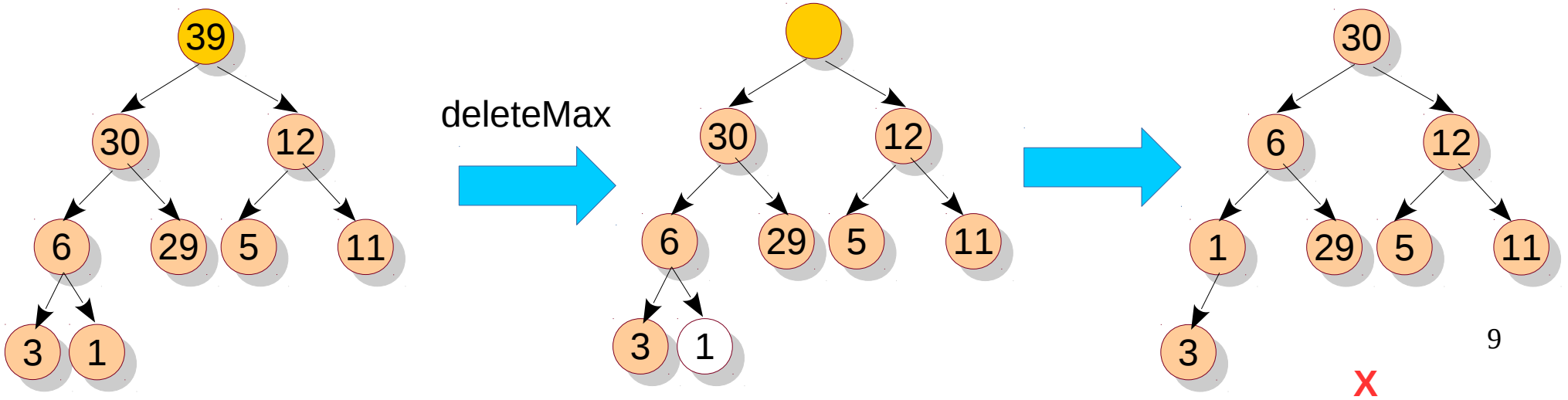
- Remove the root.
- Percolate the hole down.



Classwork

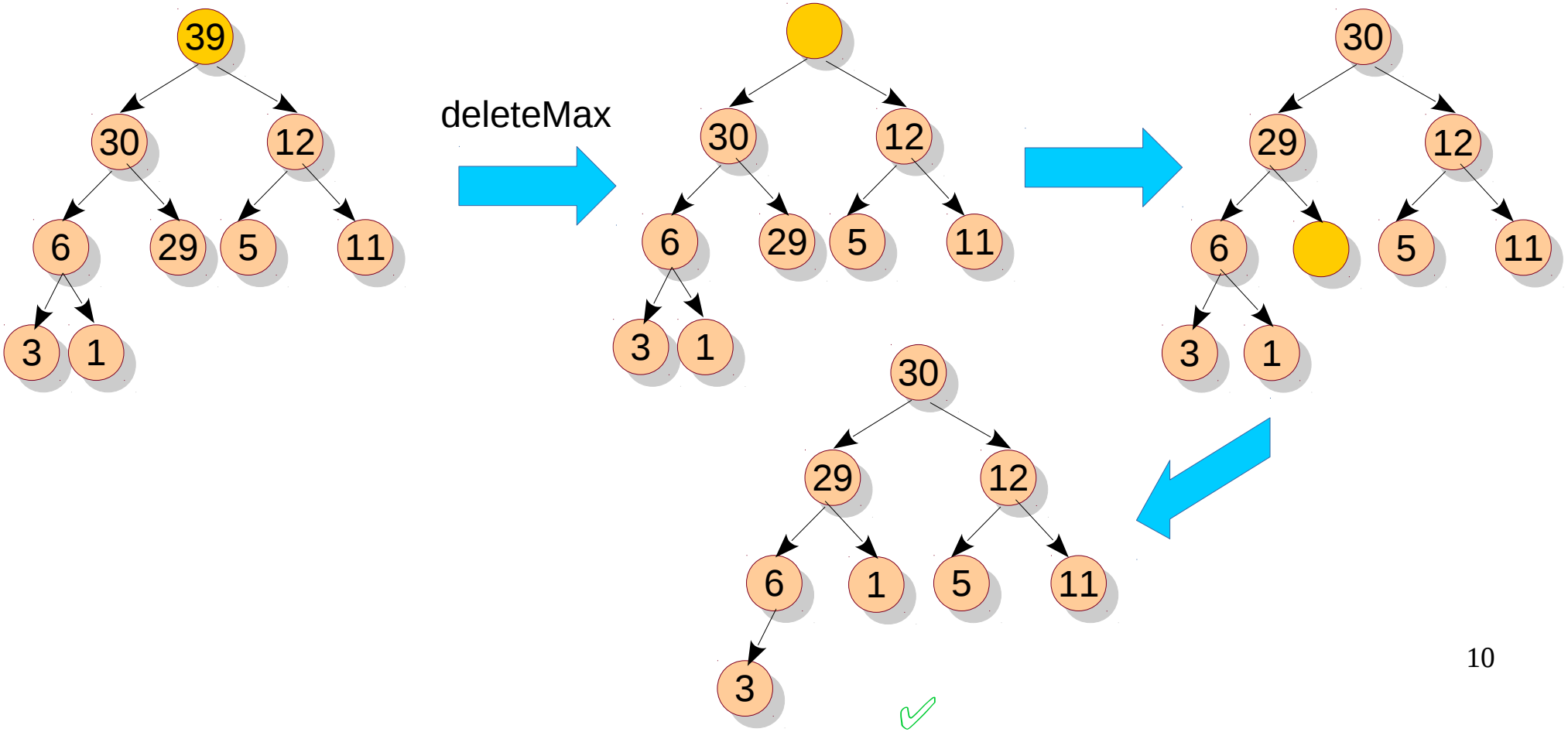
- deleteMax





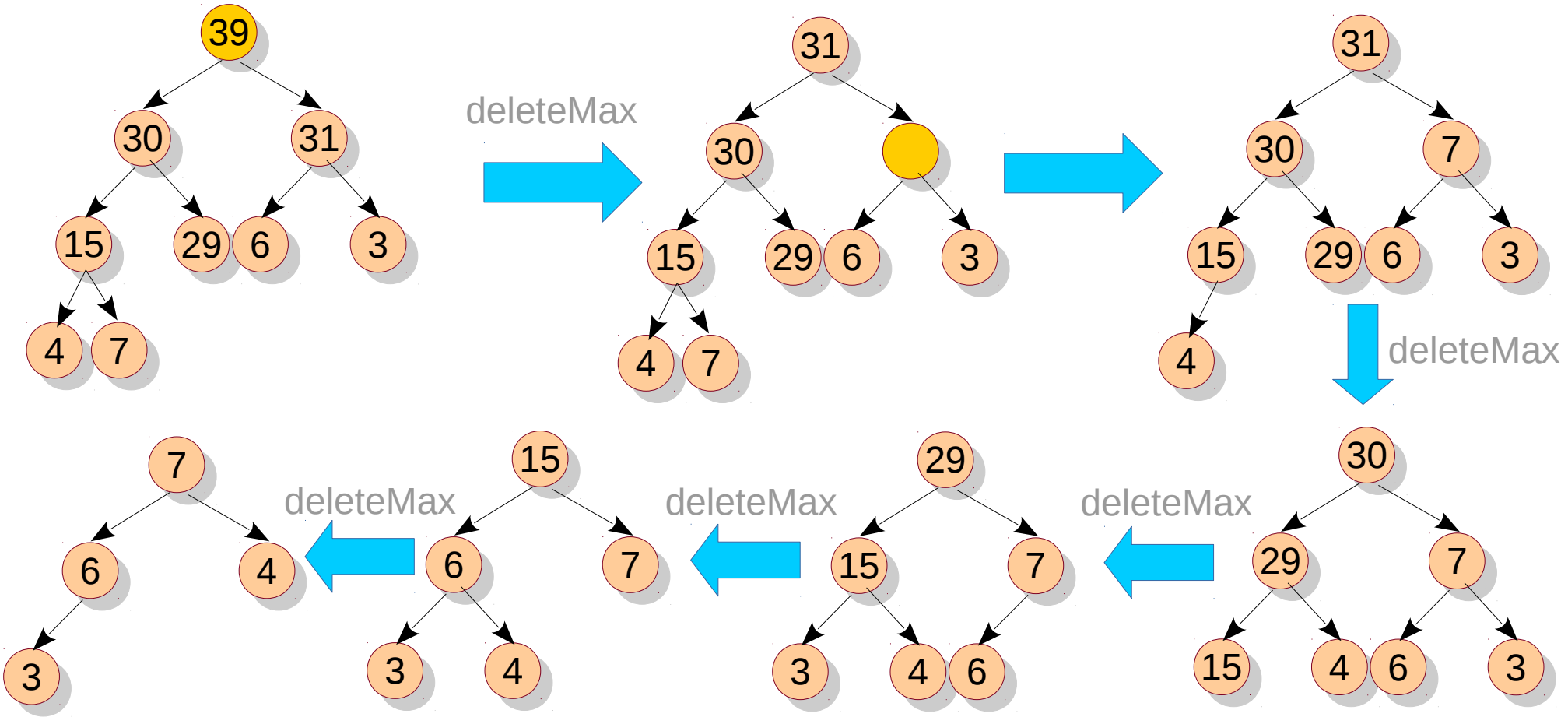
Classwork

- DeleteMax
  - Remove the root.
  - Percolate the hole down, and *finally* replace it with the last element.



Classwork

- DeleteMax
  - Remove the root.
  - Percolate the hole down, and *finally* replace it with the last element.



Other Operations

- increaseKey, decreaseKey
- remove
- build
  - N elements with max-height as log N leads to the complexity of O(N log N)
  - A tighter analysis reflects this is O(N).

Source: heap.cpp

Heap contains max. 1 node at height h (root), 2 nodes at height (h – 1), 2^2 nodes at height (h – 2), 2^i nodes at height (h – i).

Sum of the heights $S = \sum_{i=0}^{h} 2^i (h - i)$

$S = h + 2(h-1) + 4(h-2) + 8(h-3) + \ldots + 2^{h-1}(1)$

$2S = 2h + 4(h-1) + 8(h-2) + \ldots 2^h$

$2S - S = -h + 2 + 4 + 8 + \ldots + 2^{h-1} + 2^h$

$S = (2^{h+1} - 1) - (h + 1)$. Since h = log N, build takes time O(N).

Applications

- Process scheduling: Next lab assignment
- Finding k^{th} maximum
  - Sort and return k^{th} element: O(N log N)
  - Build heap and k deleteMax: O(N + k log N)

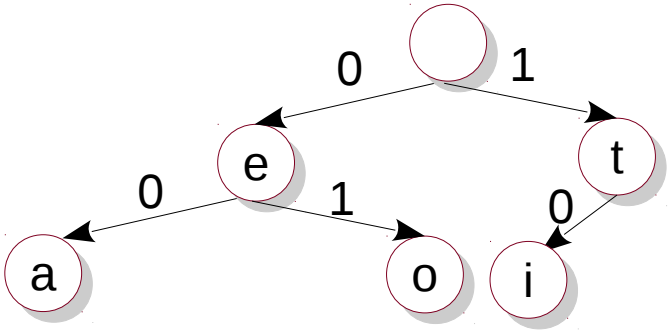
```
h.build(elements, nelements);

for (int ii = 0; ii < k - 1; ++ii)
        h.deleteMax();

cout << k << "'th largest element is " << h.deleteMax() << endl;
```

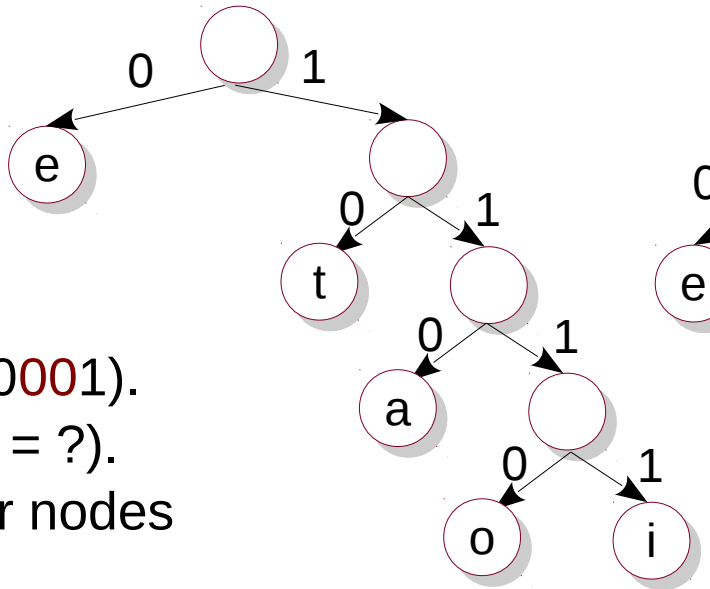
Source: heap-kmax.cpp

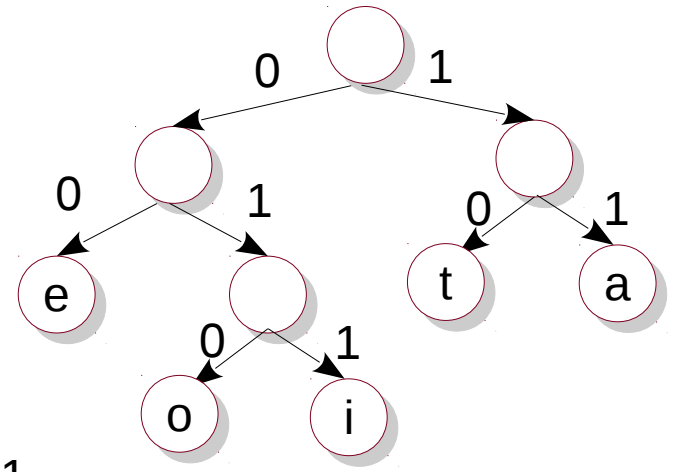
How would you find 75th largest element in a 100-element array?

Application: Huffman Coding

Character	Frequency	Code	Code 2	Code 3
e	12.02	0	0	00
t	9.10	1	10	10
a	8.12	00	110	11
o	7.68	01	1110	010
i	7.31	10	1111	011







Encoding is easy (e.g., eat = 0001).

Decoding is tough (e.g., 0001 = ?).

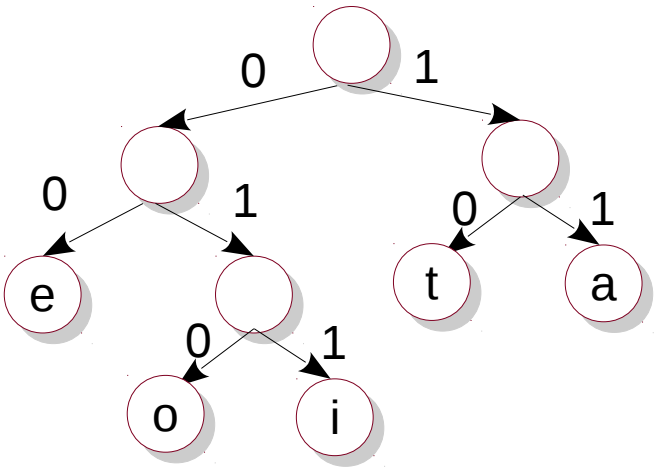
This happens because interior nodes also represent data.

We need data only at the leaves.

Called prefix codes.
We can 001110 easily now.

Application: Huffman Coding

Character	Frequency	Code	Code 2	Code 3
e	12.02	0	0	00
t	9.10	1	10	10
a	8.12	00	110	11
o	7.68	01	1110	010
i	7.31	10	1111	011



- Characters with smaller depths have shorter codes.
- Characters with higher frequencies should be near the root.
- Common subcode for characters indicates joint frequency (e.g., 01 indicates both o and i)

The above observations give us the algorithm.

Huffman Coding Algorithm

(1) Consider the lowest two frequencies.

(2) Combine to form a new frequency equal to the sum of the two.

(3) Remove the two frequencies, add the summed frequency.

(4) Repeat step (1) until a single frequency remains.

- The processing leads to a binary tree (called *Huffman Tree*).
- Finding the lowest frequency can be nicely done using Heaps.
- Huffman coding is used in compression algorithms (such as gzip and winzip) and stream formats (such as jpeg and mp3).

Source: heap-huffman.cpp

Heapsort

Given N elements,

build a heap and

then perform N deleteMax,

store each element into an array.

N storage

O(N) time

O(N log N) time

O(N) time and N space

O(N log N) time and 2N space

Can we avoid the second array?

```
for (int ii = 0; ii < nelements; ++ii) {
        h.hide_back(h.deleteMax());
}
h.printArray(nelements);
```

Source: heap-sort.cpp