Priority Queues

Reichman University

- ADT:

maintain a set of elements, where each element has a priority (key)

  - insert(x,k): insert element x with key k
  - find-min(): return element with minimum key
  - delete-min(): delete element with minimum key and return it
  - decrease-key(x,k): Decrease priority (key) of x to be k

- Many applications

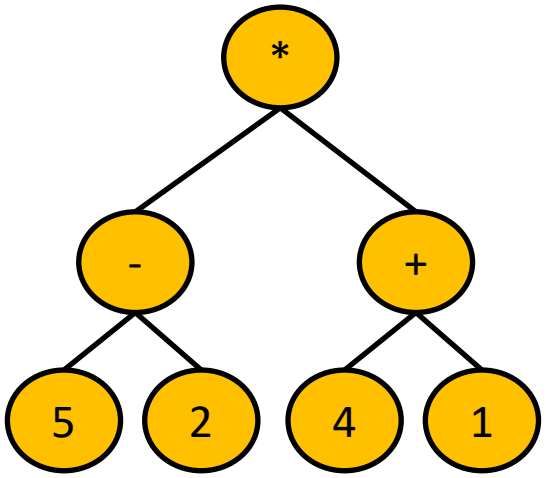
Naïve implementation using lists

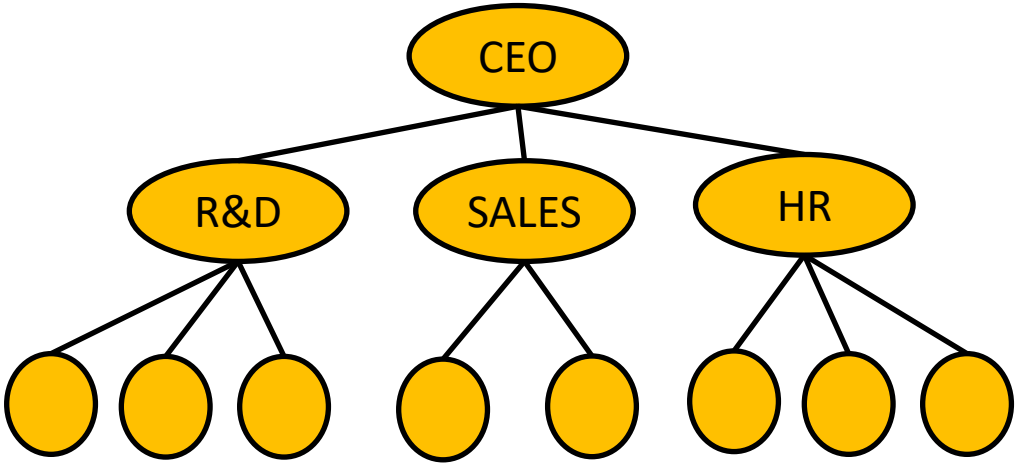
Reichman University

- Just a linked list (arbitrary order)
  - insert(x,k): O(1)
  - find-min(): O(n)
  - delete-min(): O(n)
  - decrease-key(x,k): O(1)
- Linked list sorted by key
  - insert(x,k): O(n)
  - find-min(): O(1)
  - delete-min(): O(1)
  - decrease-key(x,k): O(n)
- Can we do better?

Trees

- Used to represent hierarchical information



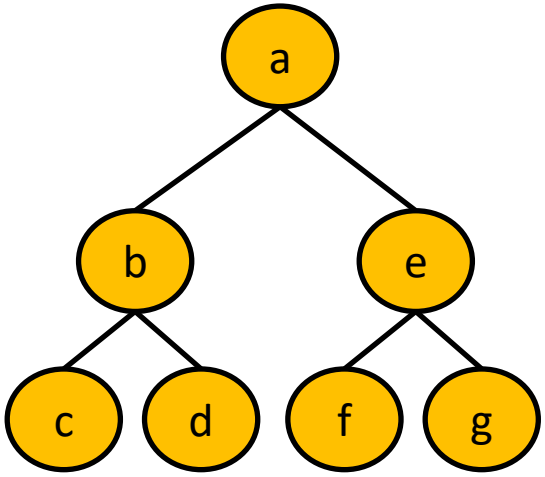


(5-2)*(4+1)

Trees

Reichman University

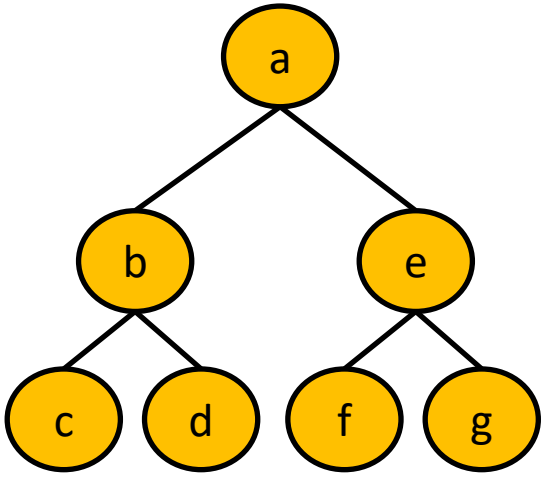
- A hierarchical set of nodes
- Recursive definition:
- Either an empty set
- Or consists of a special node called the root.
  The root can have zero or more child nodes, each of which is the root of a (sub)-tree.
  The subtrees are node-disjoint.



Trees - vocabulary

Reichman University

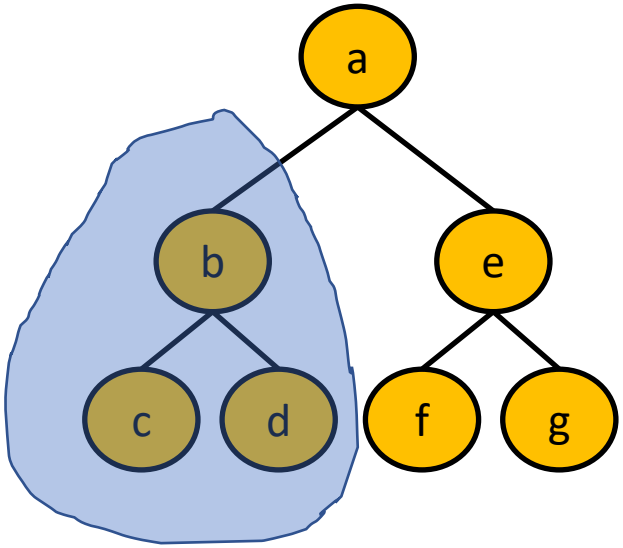
- a,b,c,d,e,f,g are nodes of the tree T
- ef is an edge of T
- b and e are the children of a
- b is the parent of c (and of d)
- a is the root of T (node with no parent)
- d is a leaf of T (node with no children)
- a node that is not a leaf is internal
- b and e are siblings in T
- a is a (strict) ancestor of f
- d is a (strict) descendent of a



Trees - vocabulary

Reichman University

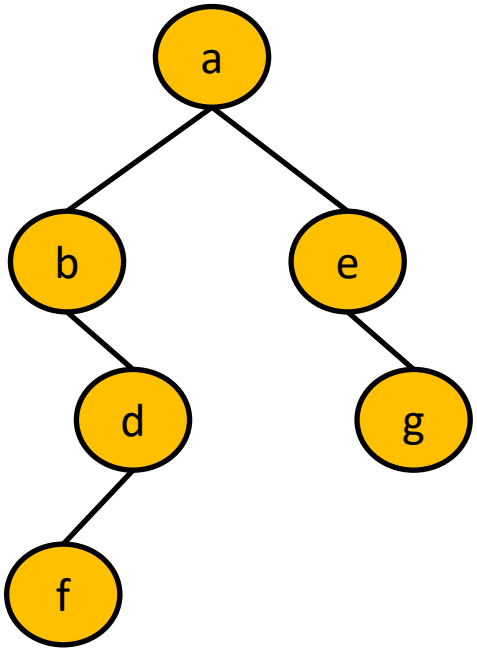
- ordered / unordered trees
- subtree rooted at b
- path between d and f
- the degree of a node is the number of children
- a binary tree has nodes of degree at most 2.



More tree vocabulary

Reichman University

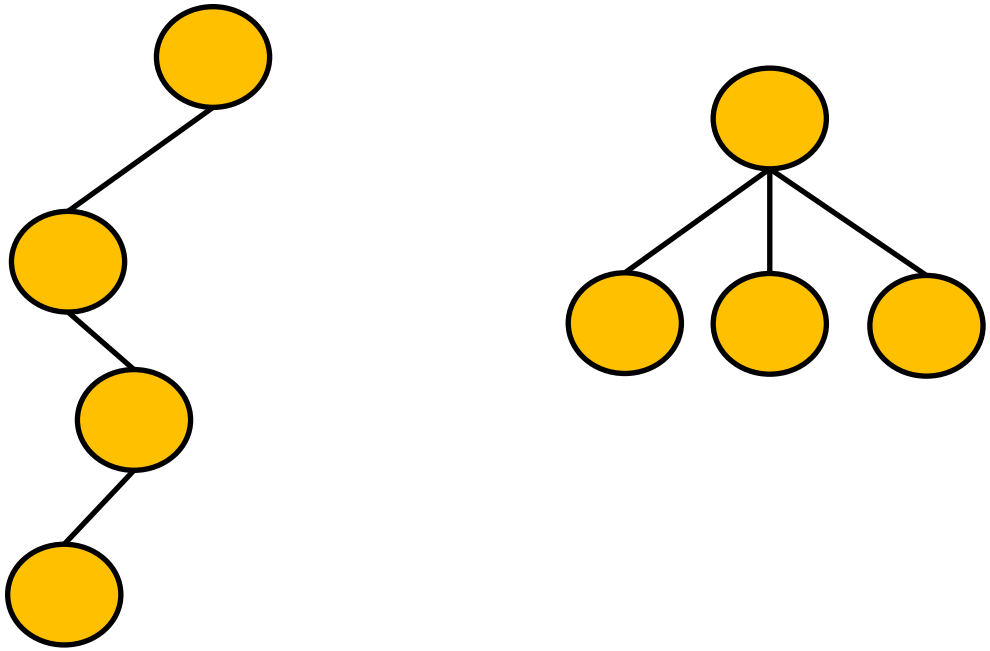
- the length of the path from d to e is 3 (number of edges)
- the depth of a node d is 2 (length of path from root)
- the height of node b is 2 (length of longest path from to a leaf)
- the depth of T is 3 (depth of deepest node)
- the height of T is 3 (height of the root)



Basic tree counting

Reichman University

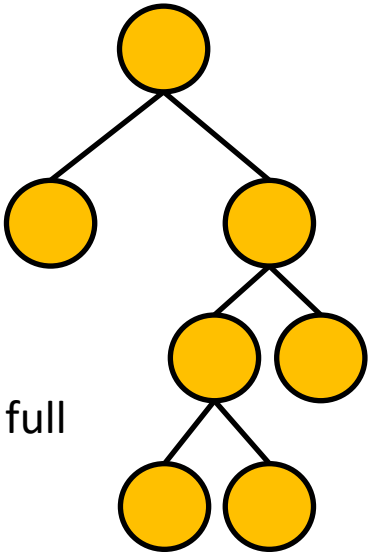
- a tree with n nodes has n-1 edges
- a tree with n≥1 nodes has maximum depth n-1
- a tree with n>1 nodes has minimum depth 1



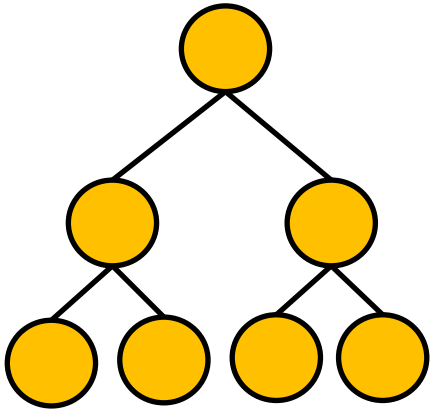
binary trees

Reichman University

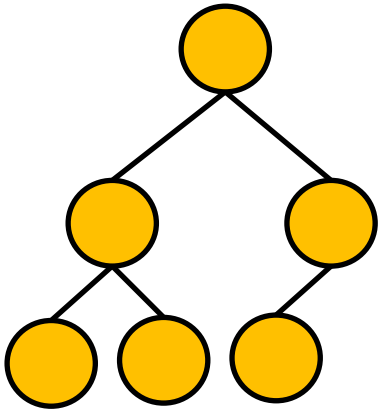
- a binary tree is full if every node has degree 2 or 0
- a binary tree is complete if all leaves are at the same depth
- an almost-complete binary tree might have a few nodes missing at the end (from the right) of the “deepest layer”.



full



complete

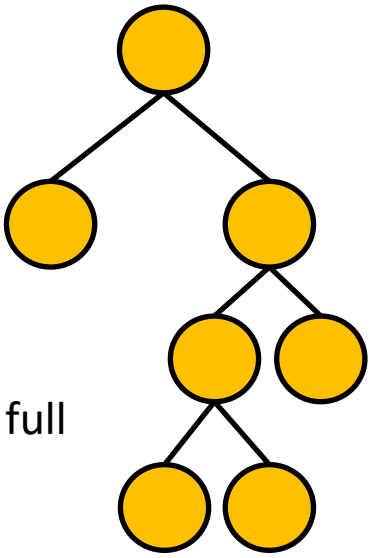


almost-complete

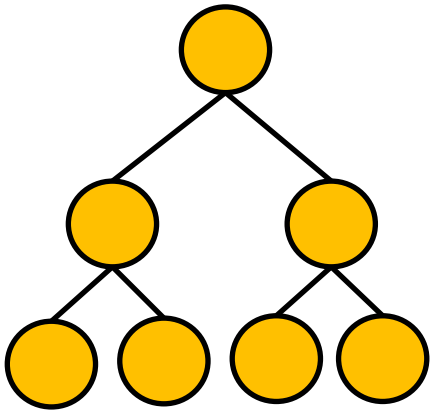
binary trees - counting

Reichman University

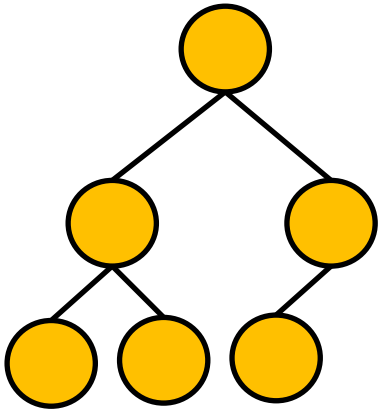
- a complete binary tree with height h has:
  - 2^i nodes at depth i (so 2^h leaves)
  - $n = 2^{h+1} - 1$ nodes
- the depth of a full binary tree with n nodes is at least $\lfloor \log_2 n \rfloor$ and at most $\lceil n/2 \rceil$



full



complete



almost-complete

Priority queue implementations

Reichman University

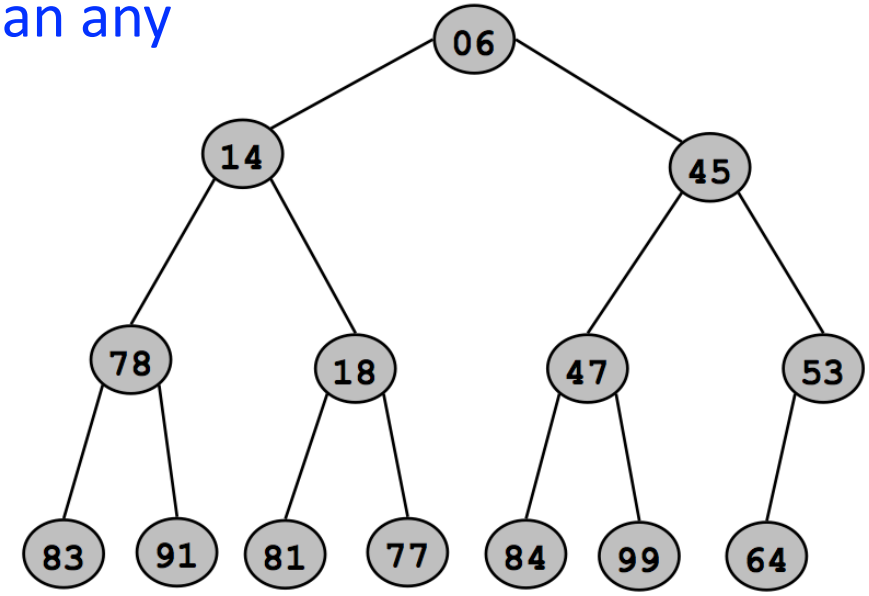
- Just a linked list (arbitrary order)
  - insert(x): O(1)
  - find-min(): O(n)
  - delete-min(): O(n)
  - decrease-key(x,k): O(1)

- Binary heaps – implemented with a binary tree
  - insert(x): O(log n)
  - find-min(): O(1)
  - delete-min(): O(log n)
  - decrease-key(x,k): O(log n)

Binary heaps

Reichman University

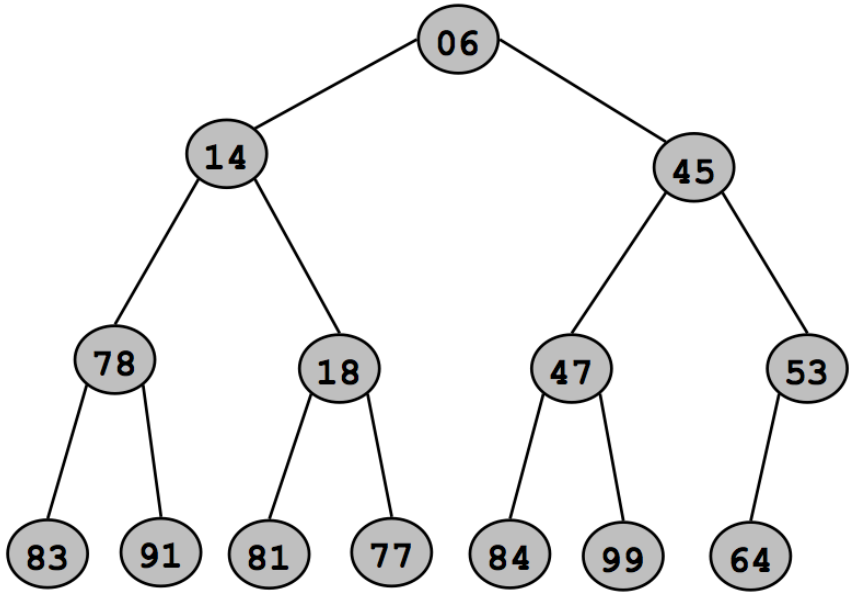
- a heap implemented using a binary tree
- maintaining the following invariants:
  - the tree is almost-complete:
    all levels are full excepts, perhaps, the deepest level
  - obeys min-heap property:
    a node is not greater than any
    of its descendants.



Binary heaps

- min element is the root
- value parent $\leq$ value of each of its children
- heap with N elements has height $\lfloor \log N \rfloor$

N = 14
height = 3

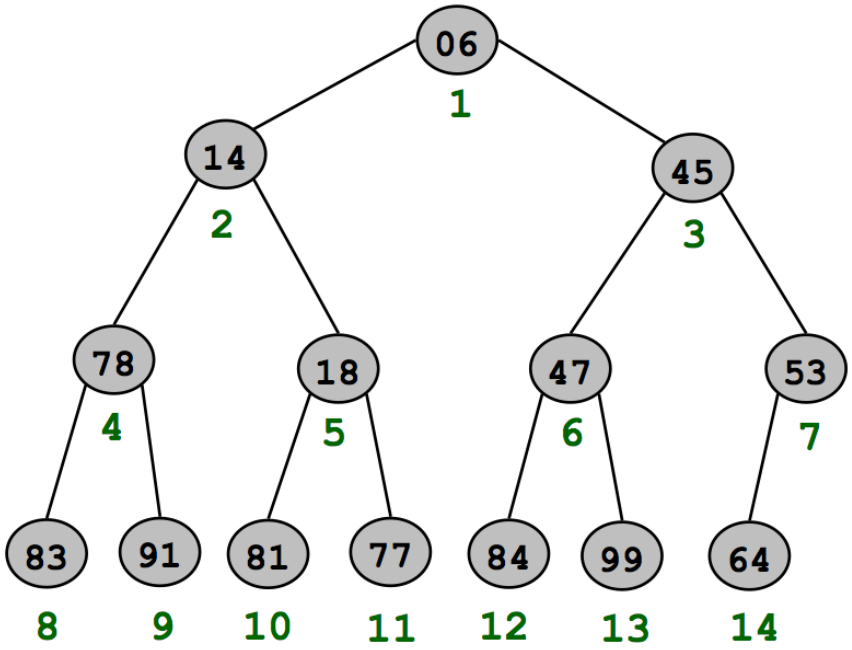


Binary heaps

Reichman University

Implementing binary heaps.

- **Use an array: no need for explicit parent or child pointers.**
  - `Parent(i) = ⌊i/2⌋`
  - `Left(i) = 2i`
  - `Right(i) = 2i + 1`

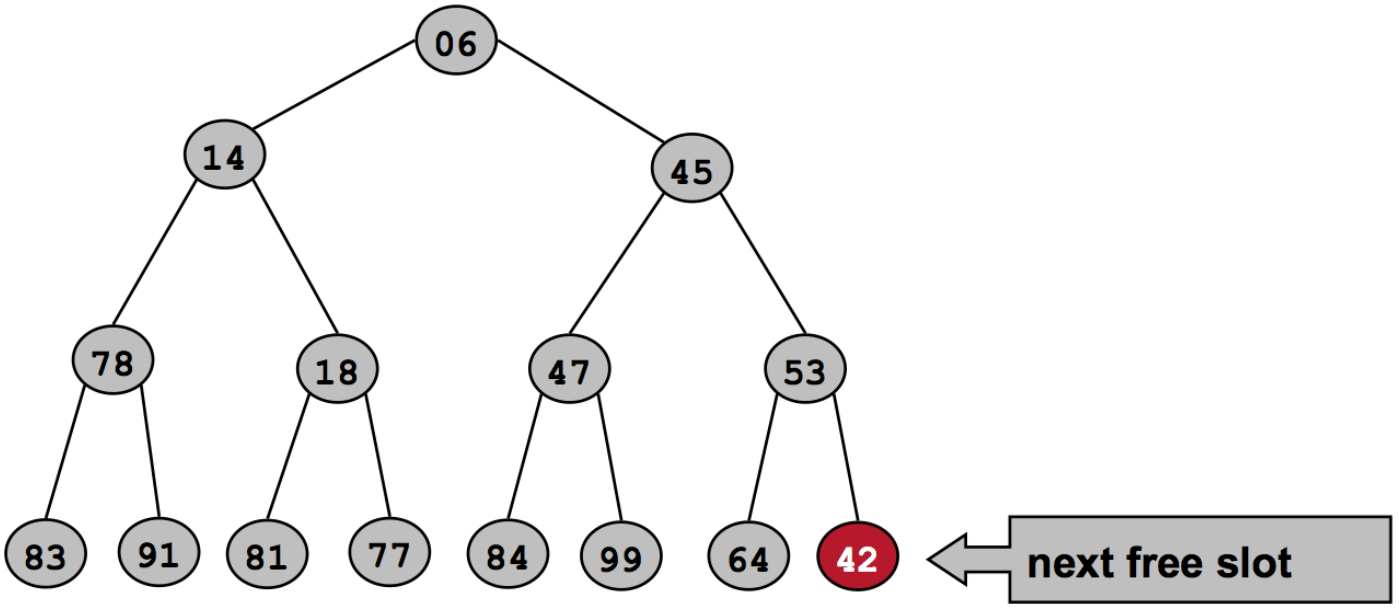


Binary heaps : insertion

Reichman University

Insert element x into heap.

- **Insert into next available slot.**
- **Bubble up until it's heap ordered.**
  - **Peter principle: nodes rise to level of incompetence**

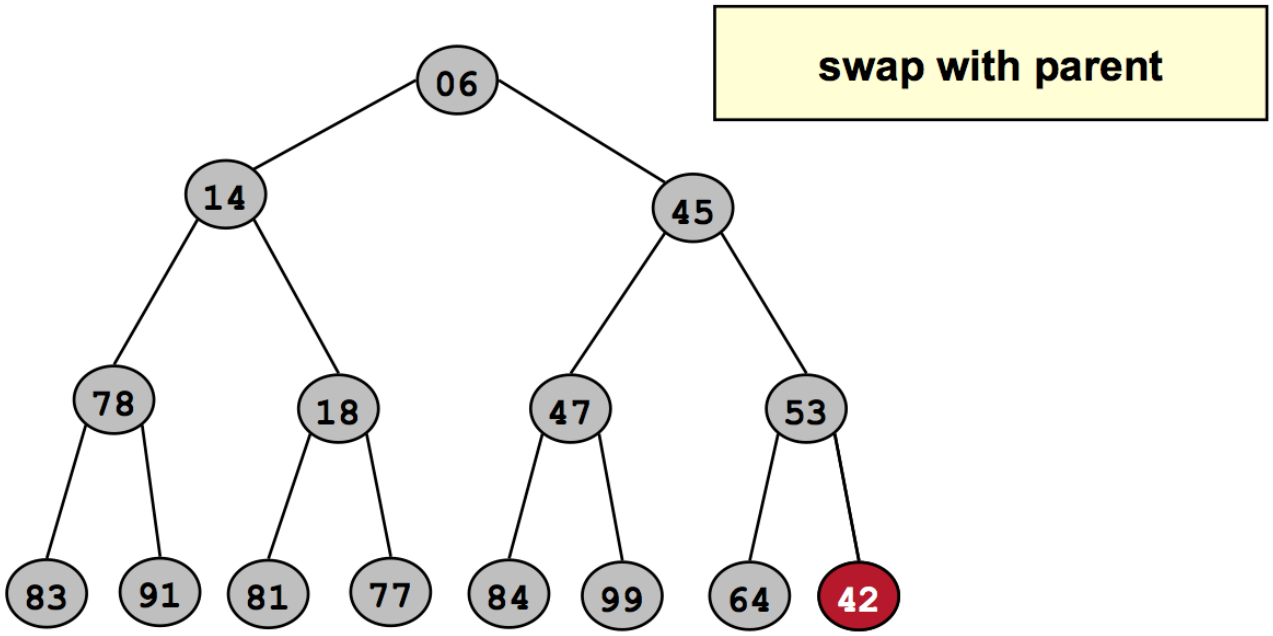


Binary heaps : insertion

Reichman University

Insert element x into heap.

- **Insert into next available slot.**
- **Bubble up until it's heap ordered.**
  - **Peter principle: nodes rise to level of incompetence**

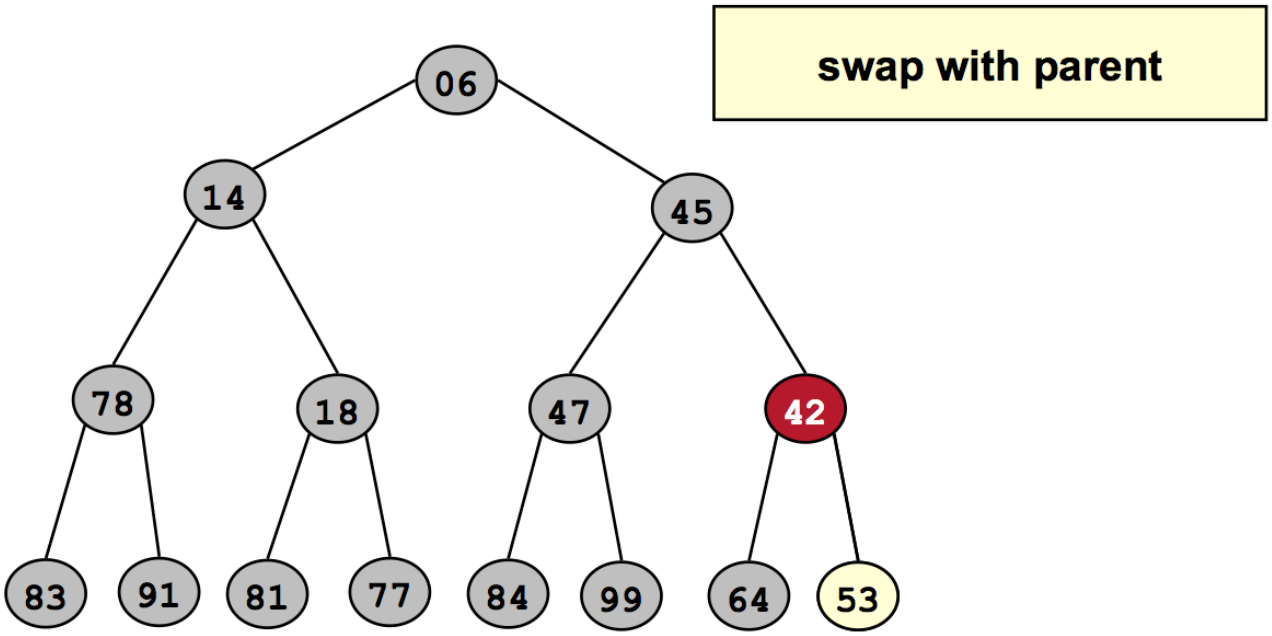


Binary heaps : insertion

Reichman University

Insert element x into heap.

- **Insert into next available slot.**
- **Bubble up until it's heap ordered.**
  - **Peter principle: nodes rise to level of incompetence**

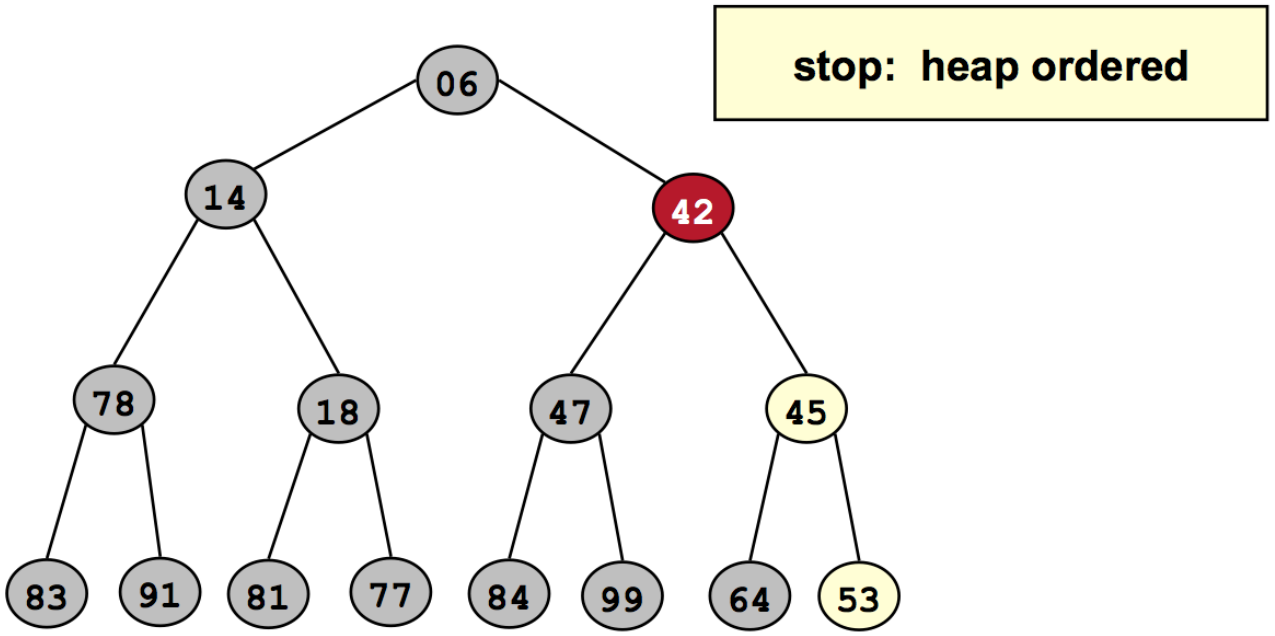


Binary heaps : insertion

Reichman University

Insert element x into heap.

- **Insert into next available slot.**
- **Bubble up until it's heap ordered.**
  - **Peter principle: nodes rise to level of incompetence**
- **O(log N) operations.**

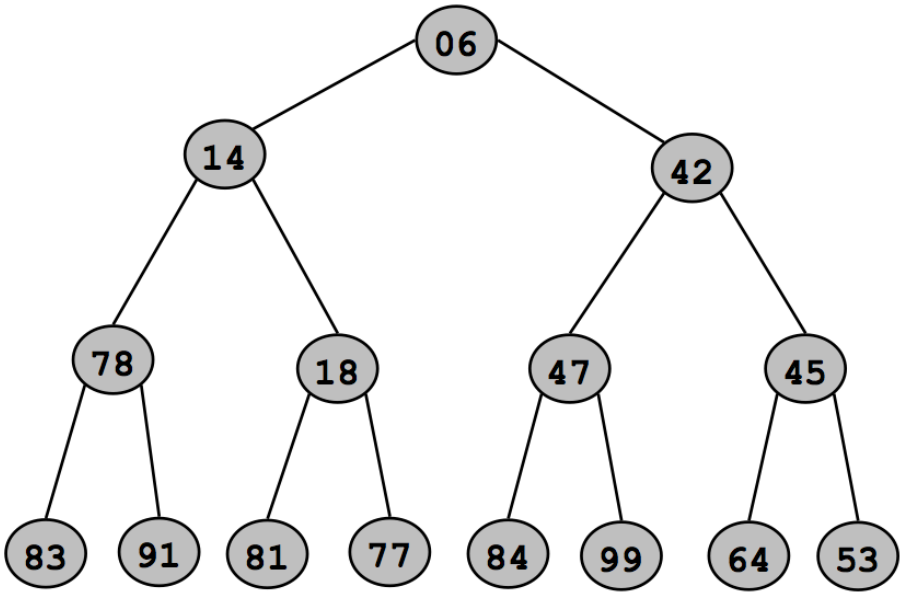


Binary heaps : decrease-key

Reichman University

Decrease key of element x to k.

- **Bubble up until it's heap ordered.**
- **O(log N) operations.**

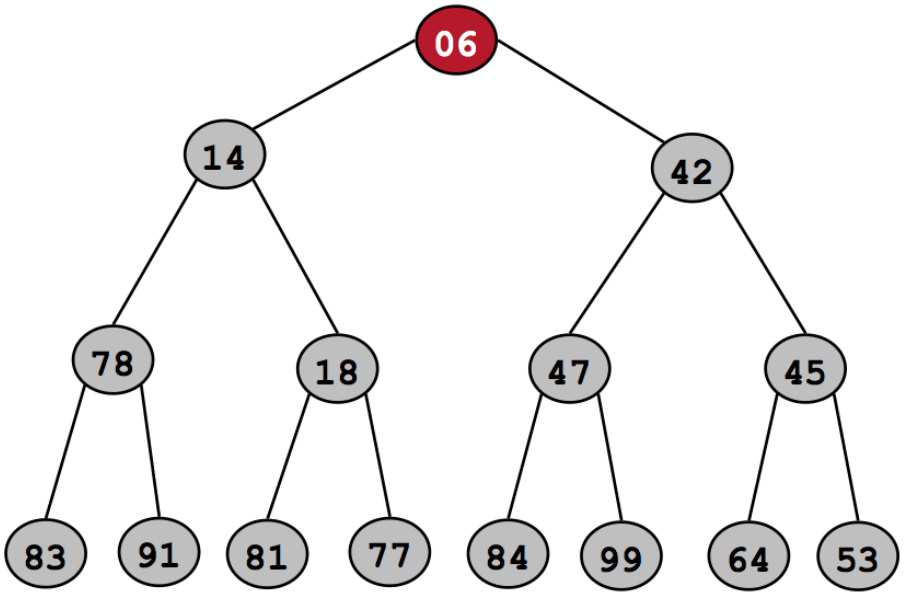


Binary heaps : delete-min

Reichman University

Delete minimum element from heap.

- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**

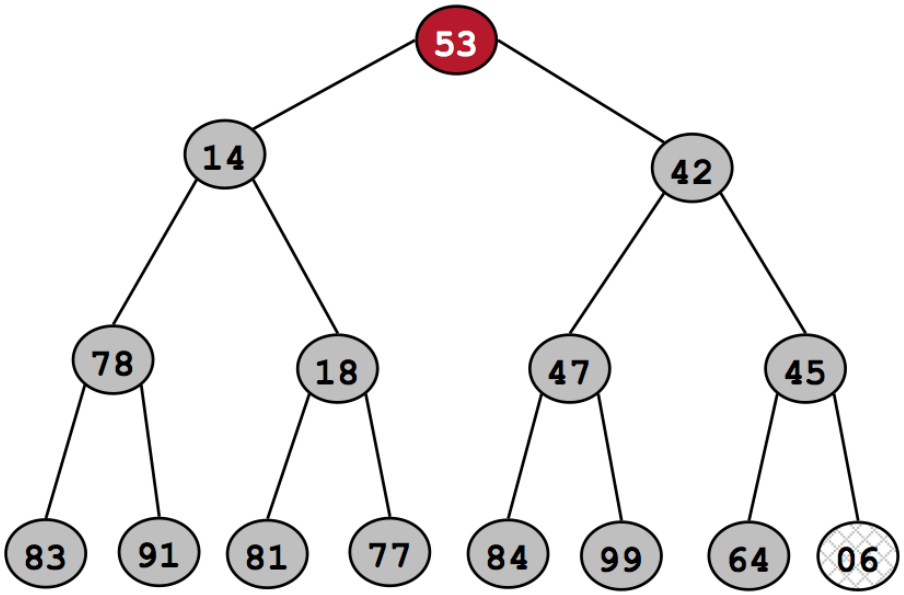


Binary heaps : delete-min

Reichman University

Delete minimum element from heap.

- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**

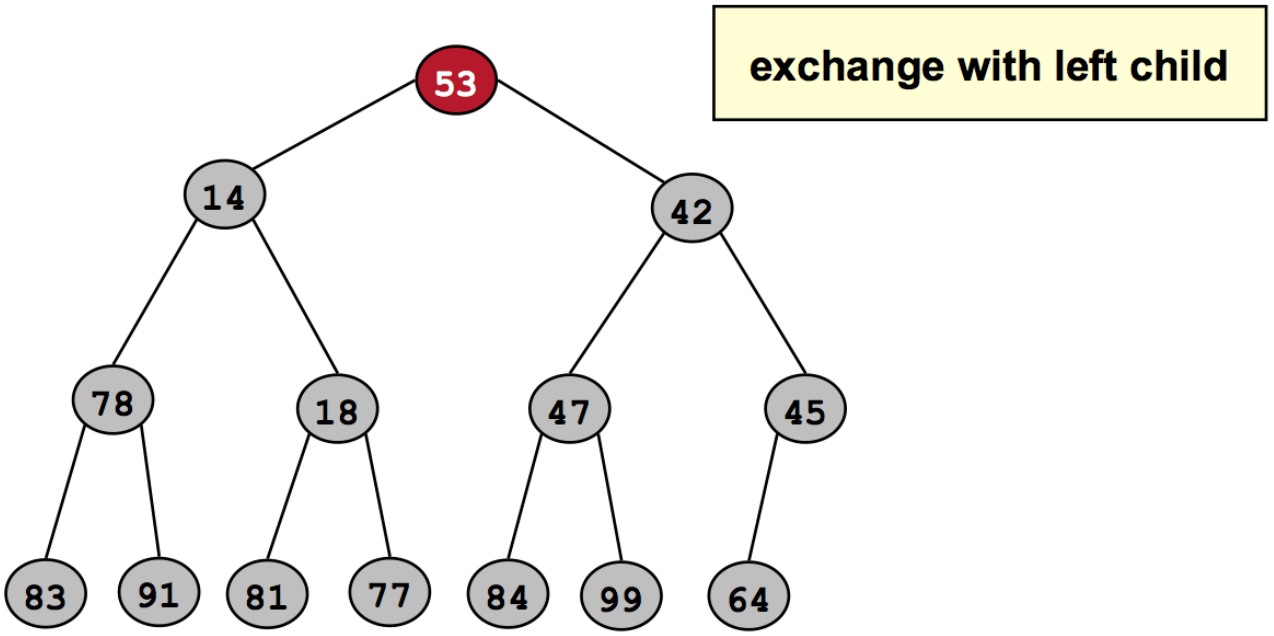


Binary heaps : delete-min

Reichman University

Delete minimum element from heap.

- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**

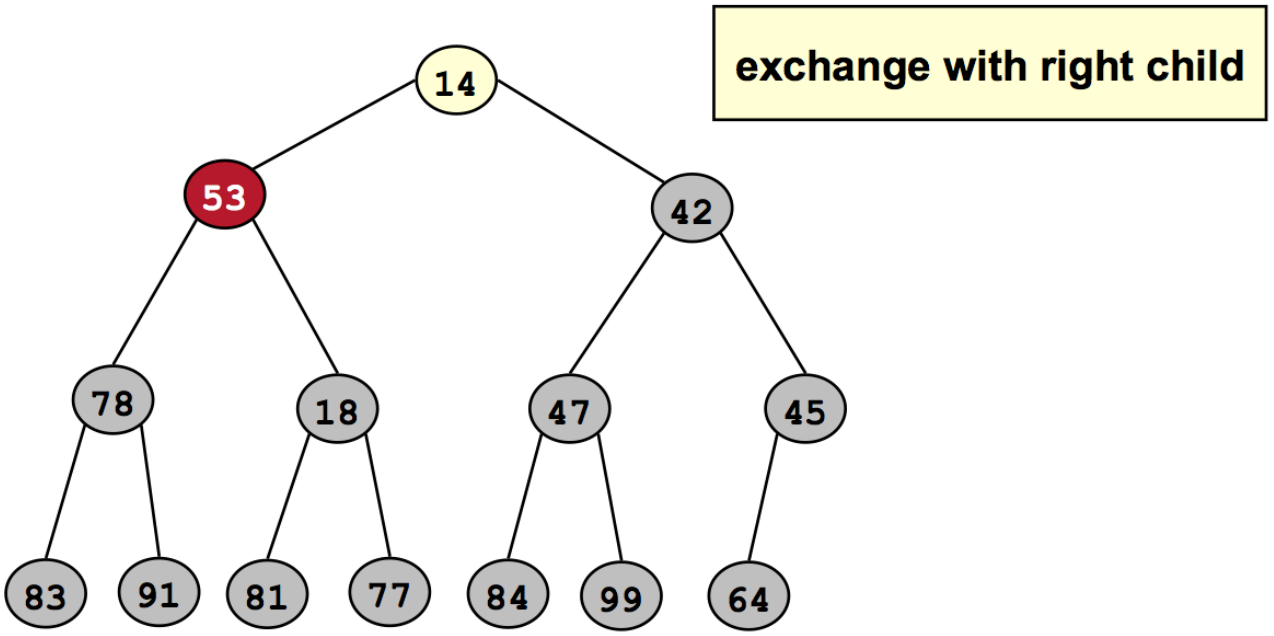


Binary heaps : delete-min

Reichman University

Delete minimum element from heap.

- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**

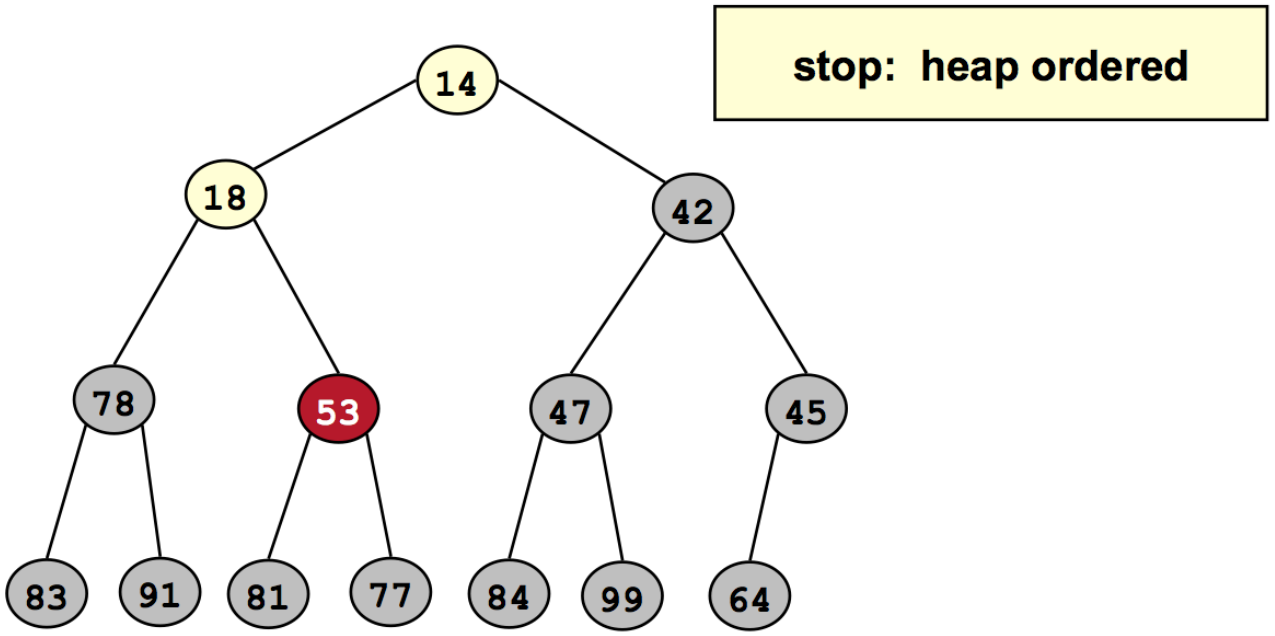


Binary heaps : delete-min

Reichman University

Delete minimum element from heap.

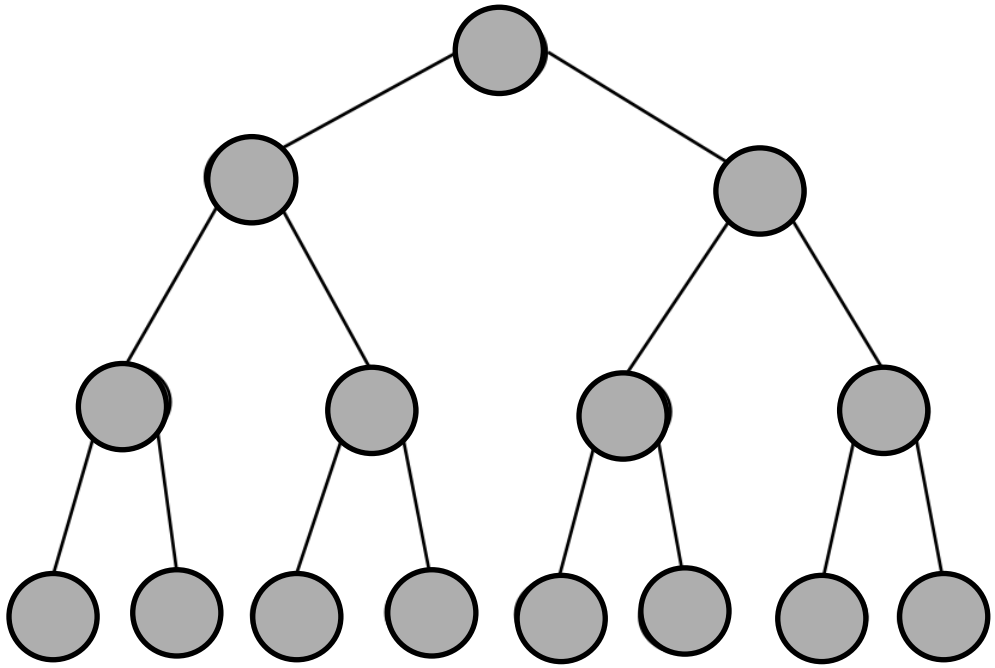
- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**
- **O(log N) operations.**



Binary heaps : build-heap with n elements

Reichman University

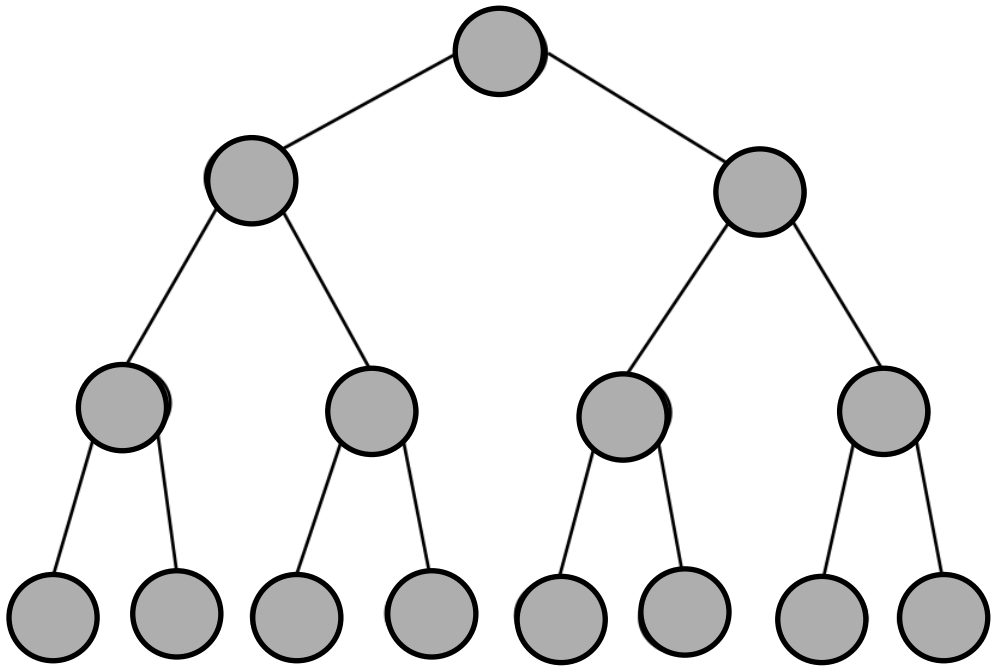
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)



Binary heaps : build-heap with n elements

Reichman University

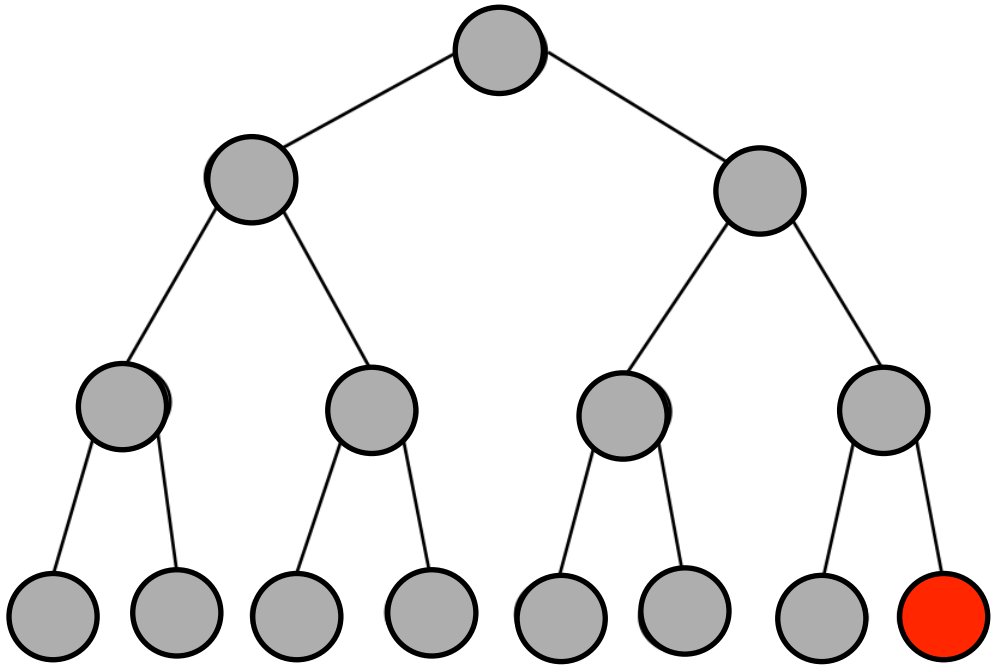
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

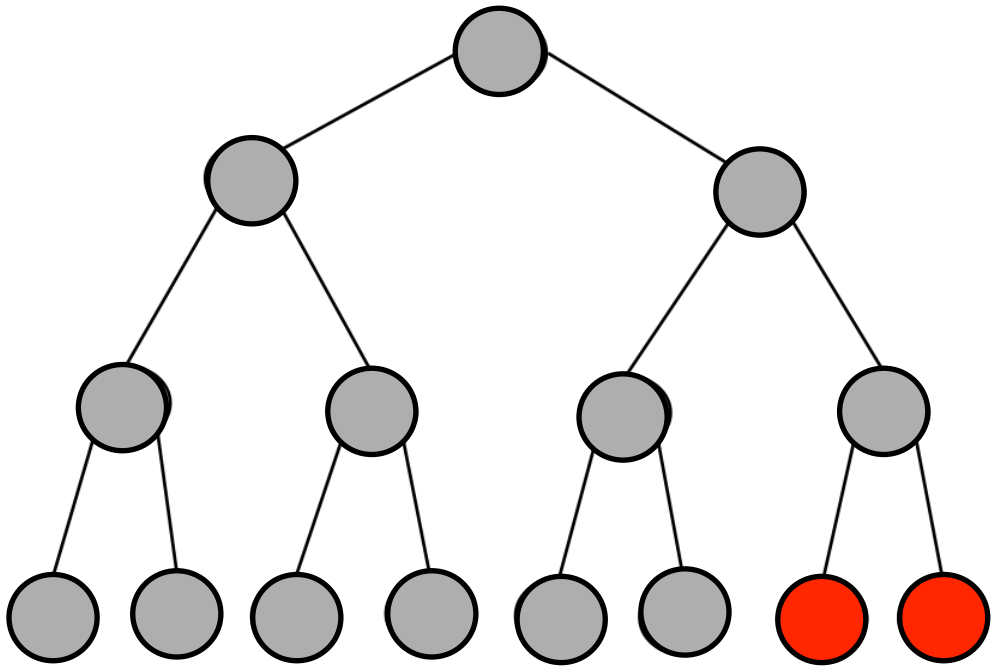
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

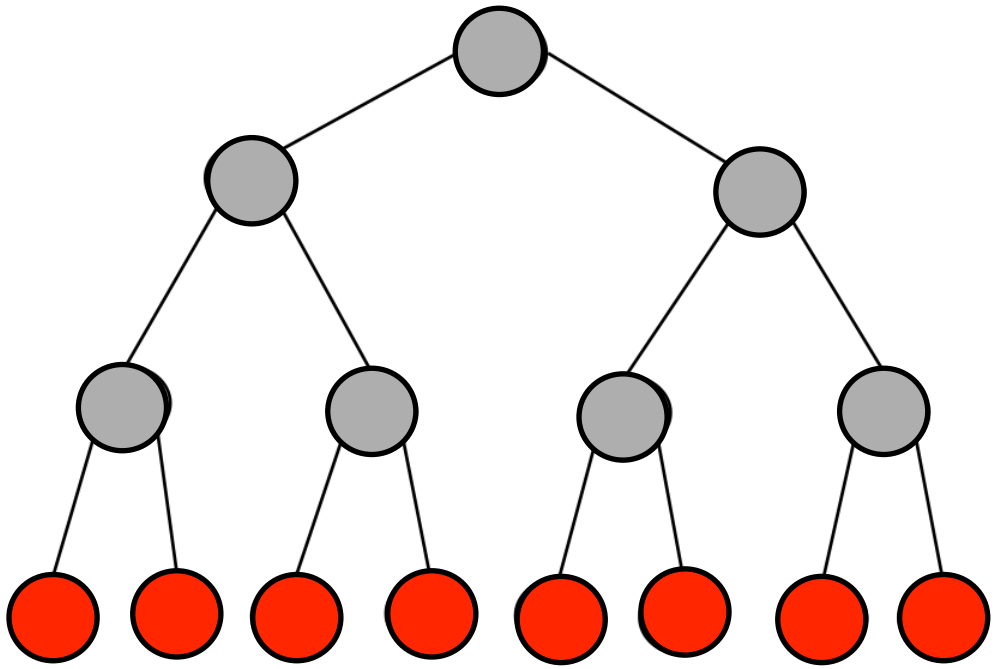
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

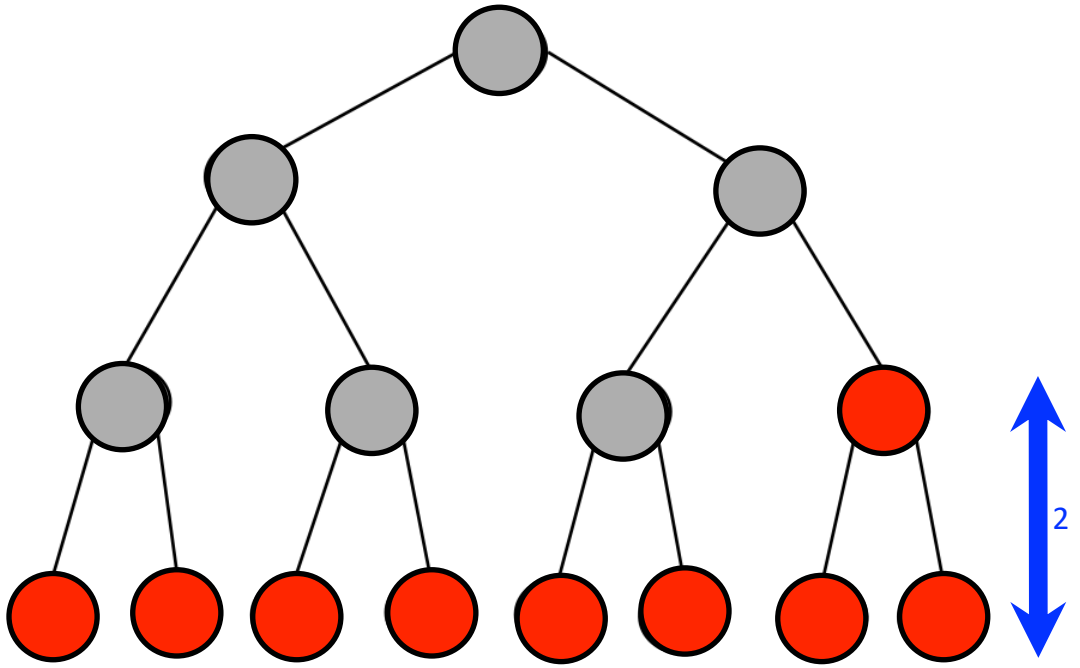
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

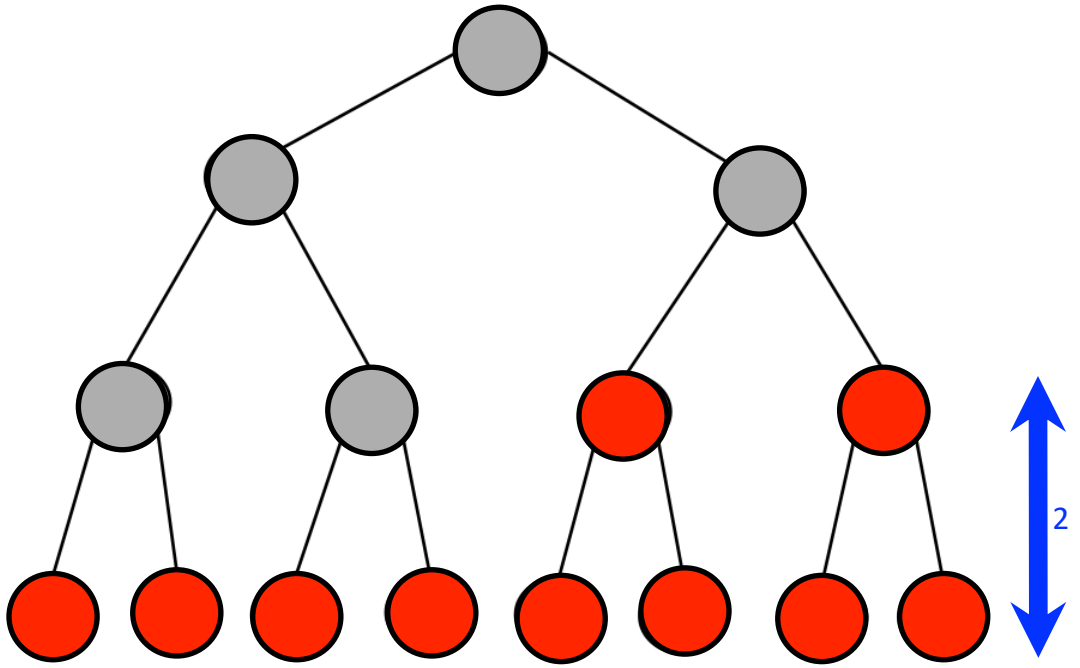
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

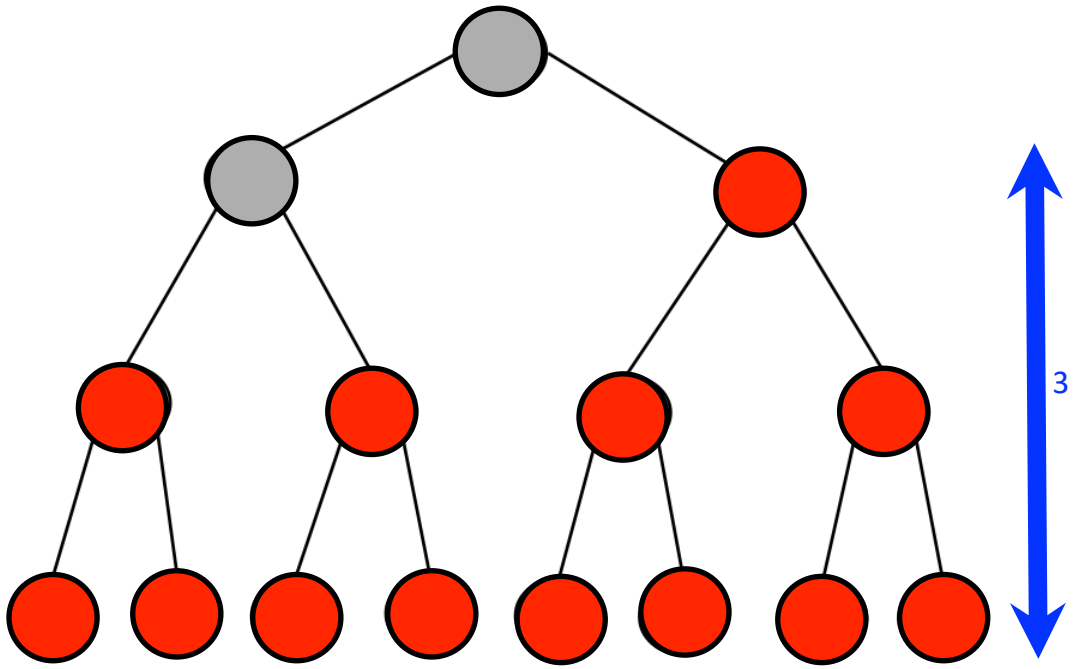
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

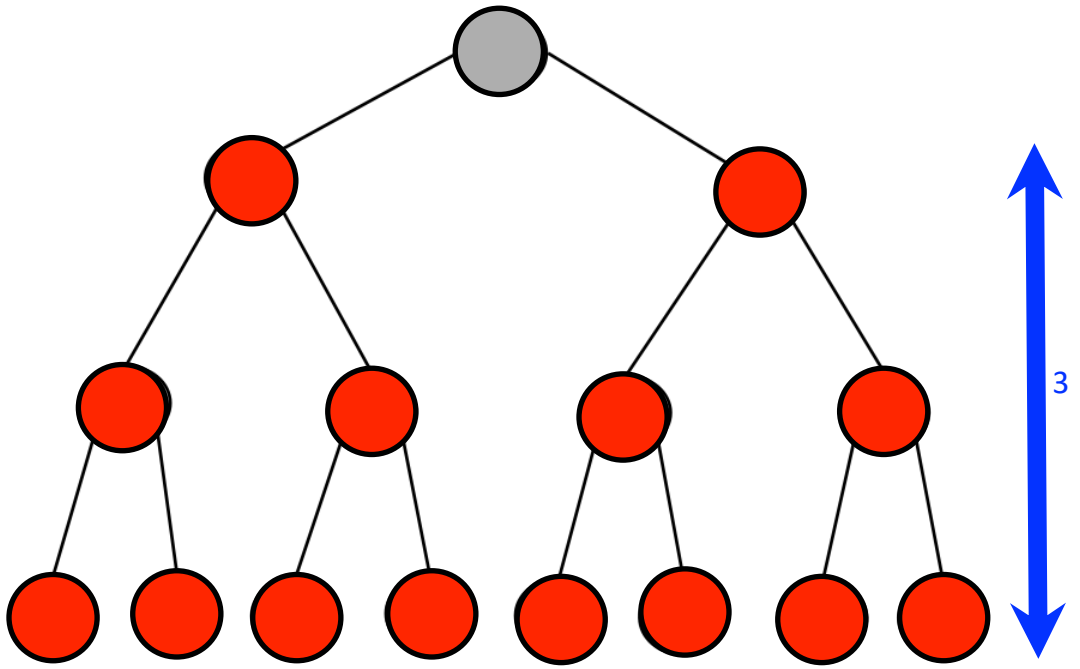
- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Binary heaps : build-heap with n elements

Reichman University

- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

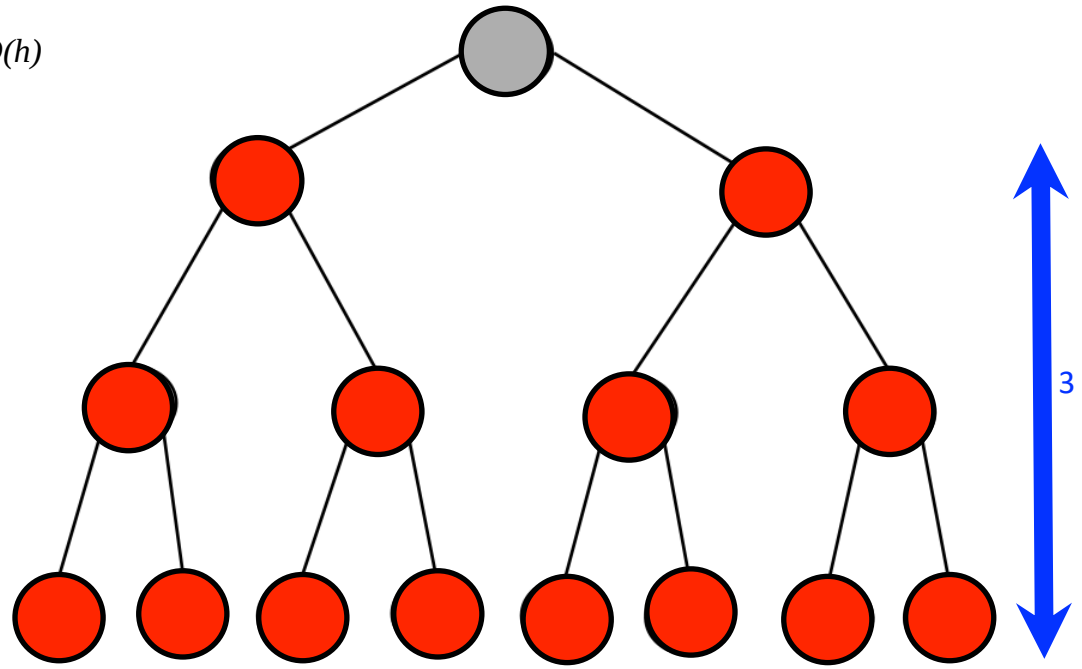


Binary heaps : build-heap with n elements

Reichman University

- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

$$\leq \sum_{h=0}^{\log n} (n/2^{h+1}) \cdot O(h)$$



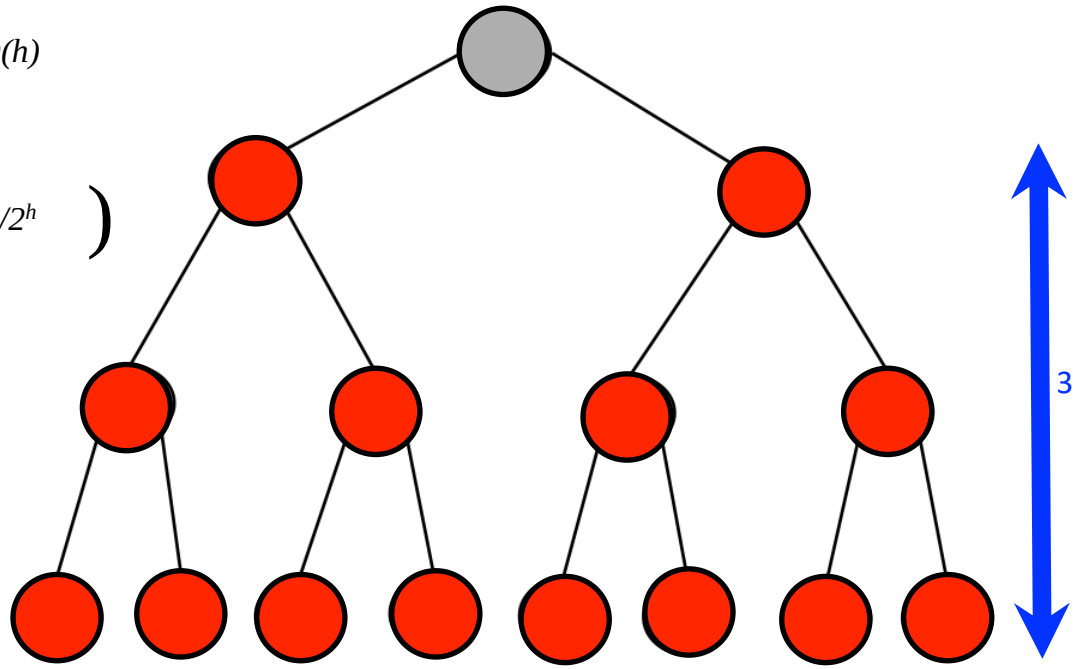
Binary heaps : build-heap with n elements

Reichman University

- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

$$\leq \sum_{h=0}^{\log n} (n/2^{h+1}) \cdot O(h)$$

$$\leq O\left(n \cdot \sum_{h=0}^{\infty} h/2^{h}\right)$$



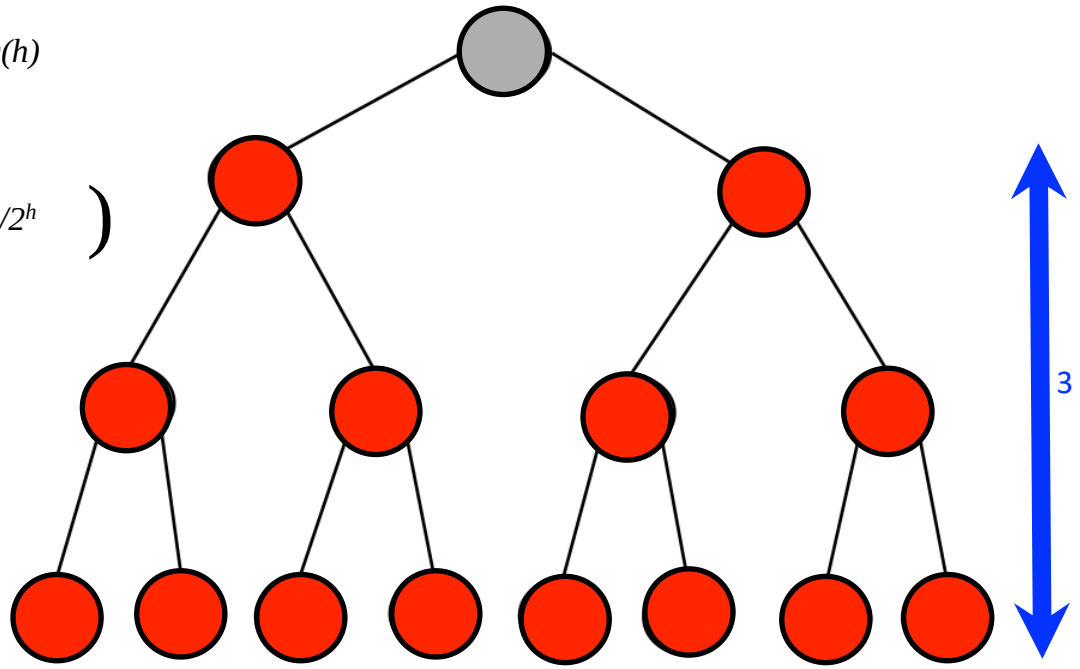
Binary heaps : build-heap with n elements

- Start with arbitrary array
- Think of it as a binary heap (without the heap property)
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

$$\leq \sum_{h=0}^{\log n} (n/2^{h+1}) \cdot O(h)$$

$$\leq O\left(n \cdot \sum_{h=0}^{\infty} h/2^{h}\right)$$

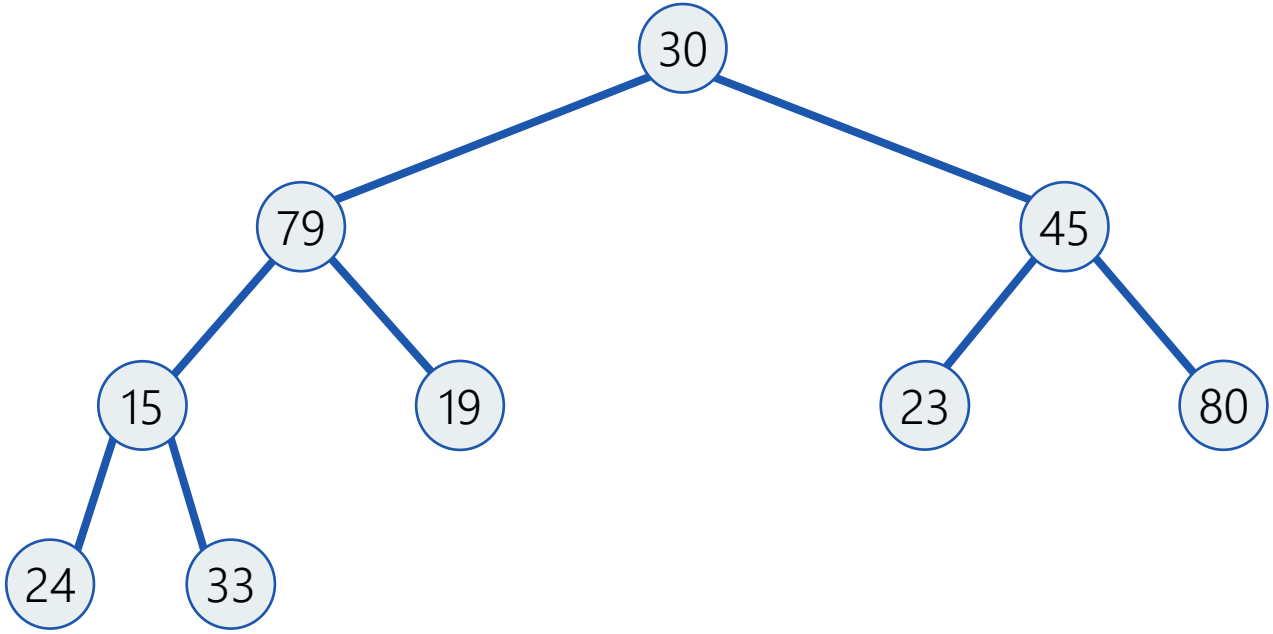
$$= O(n)$$



Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



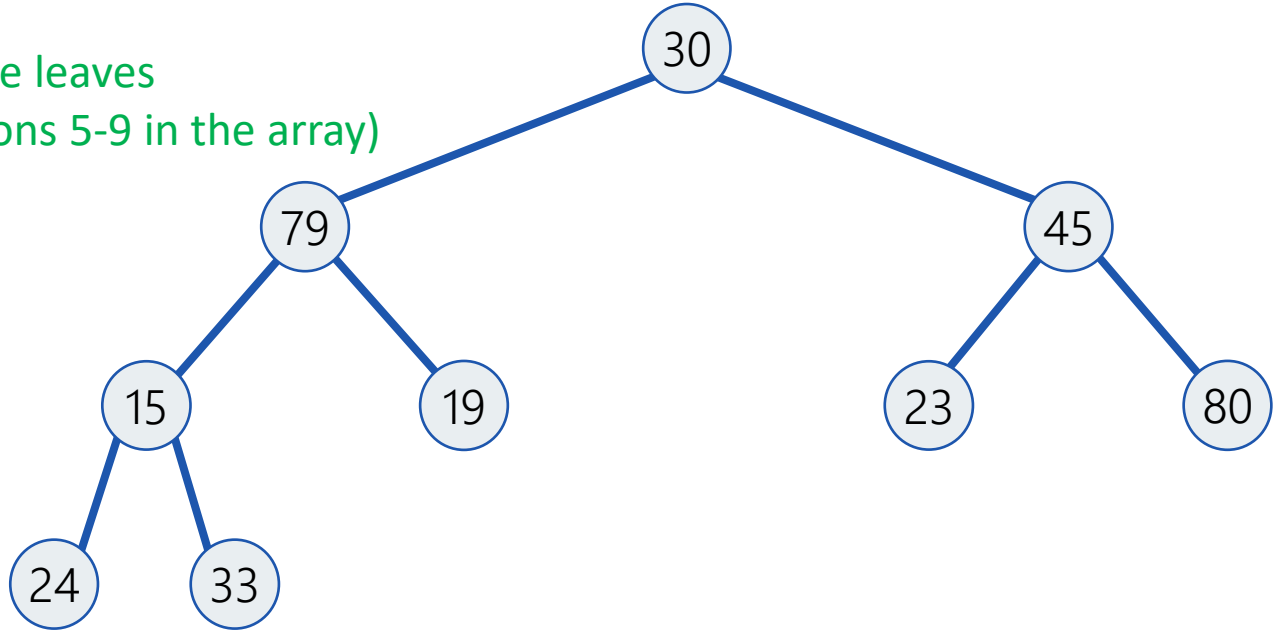
1	2	3	4	5	6	7	8	9
30	79	45	15	19	23	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

Skip the leaves
(positions 5-9 in the array)

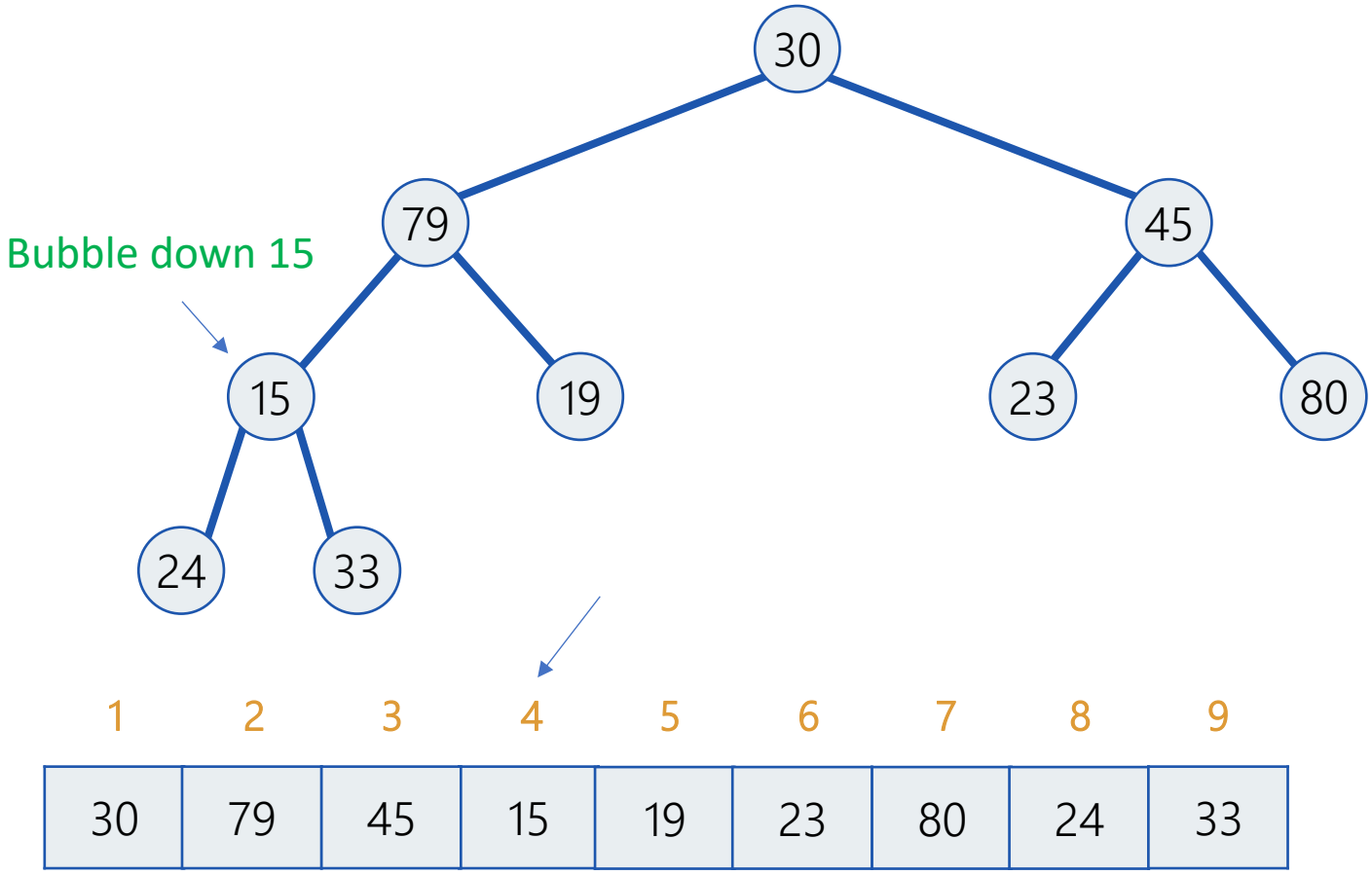


1	2	3	4	5	6	7	8	9
30	79	45	15	19	23	80	24	33

Example: build-heap with n elements

Reichman University

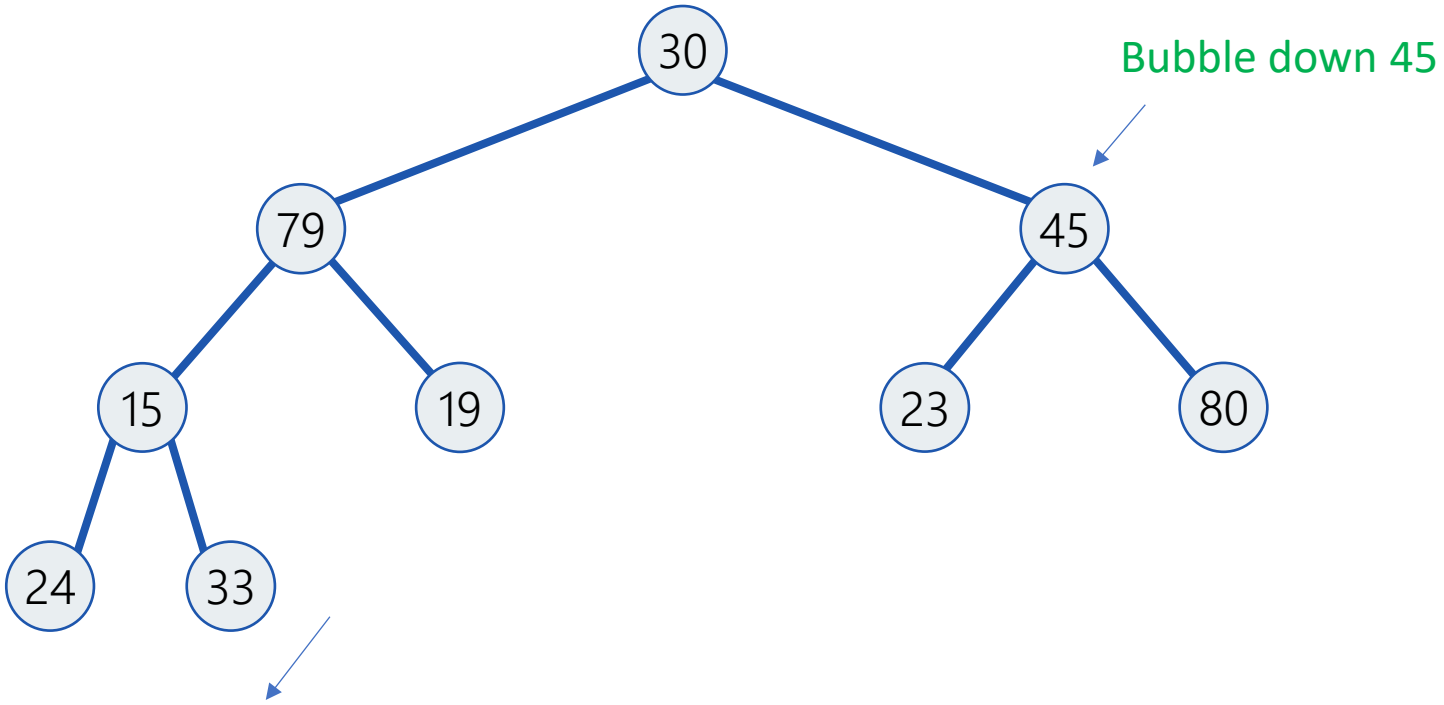
- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

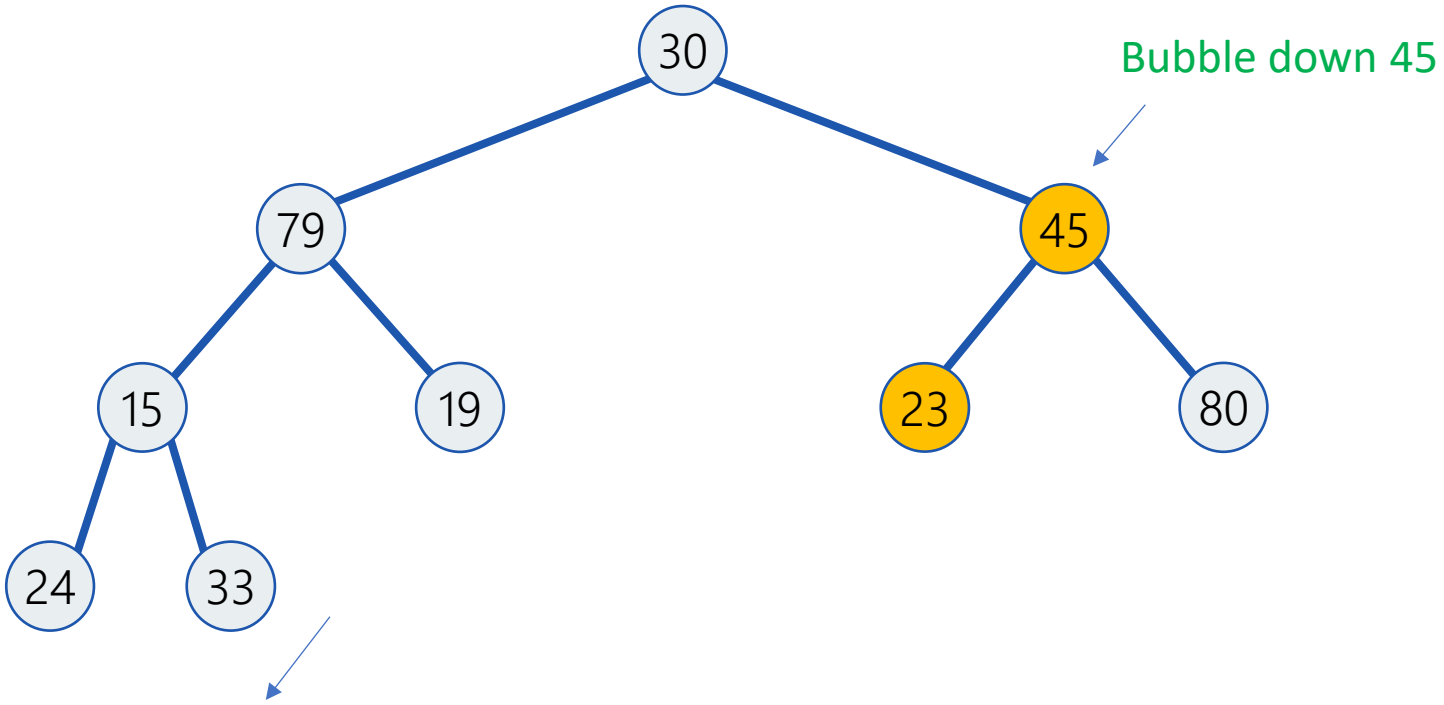


1	2	3	4	5	6	7	8	9
30	79	45	15	19	23	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

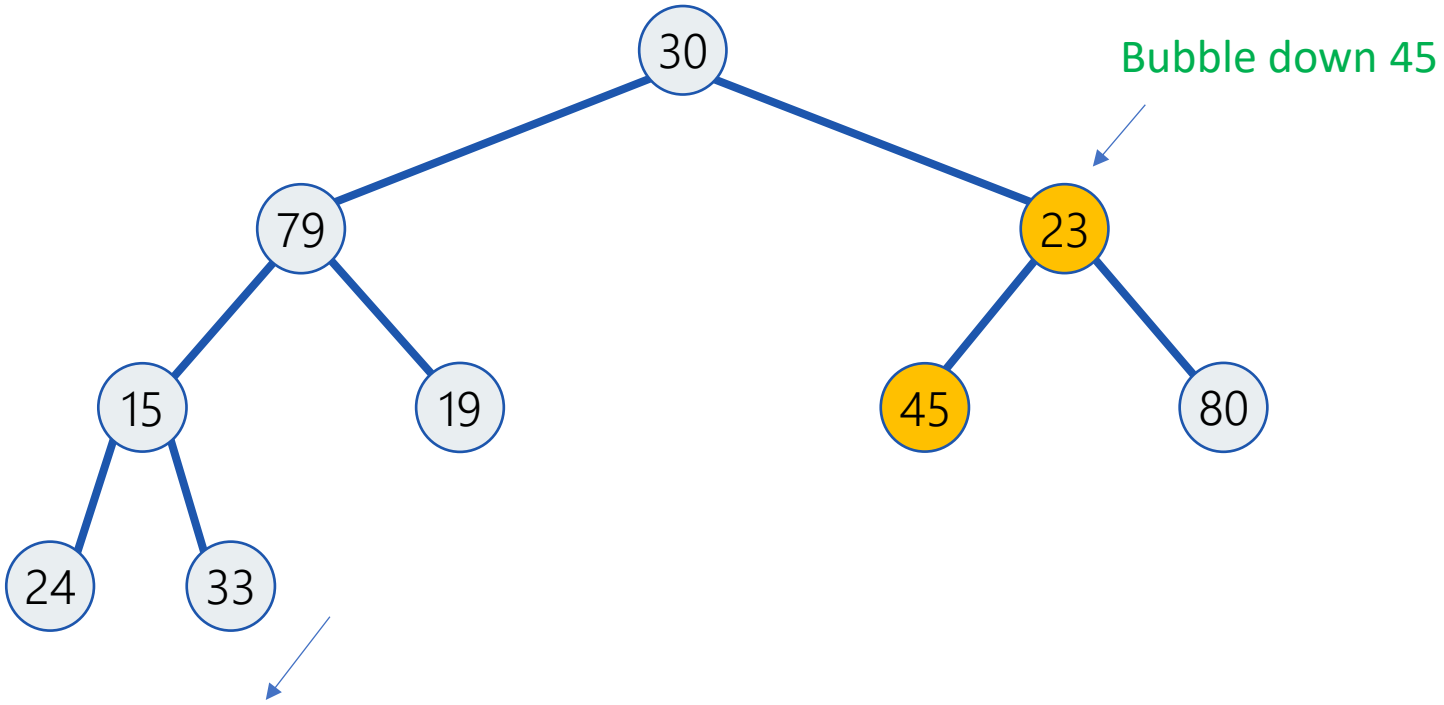


1	2	3	4	5	6	7	8	9
30	79	45	15	19	23	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

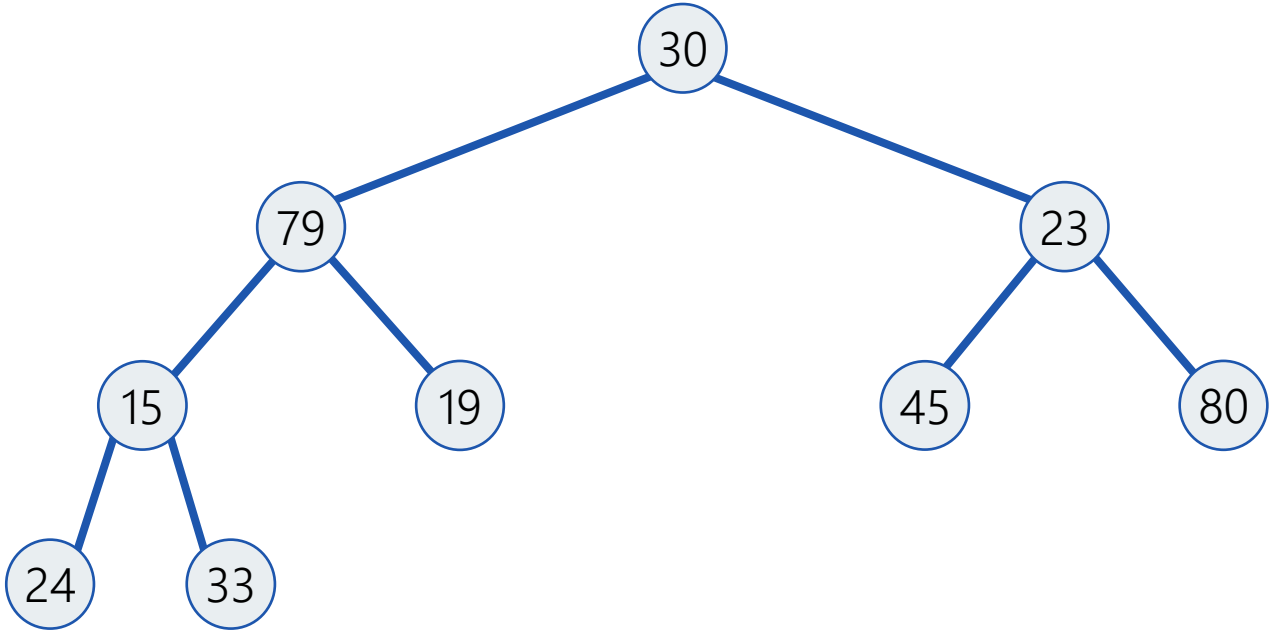


1	2	3	4	5	6	7	8	9
30	79	23	15	19	45	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

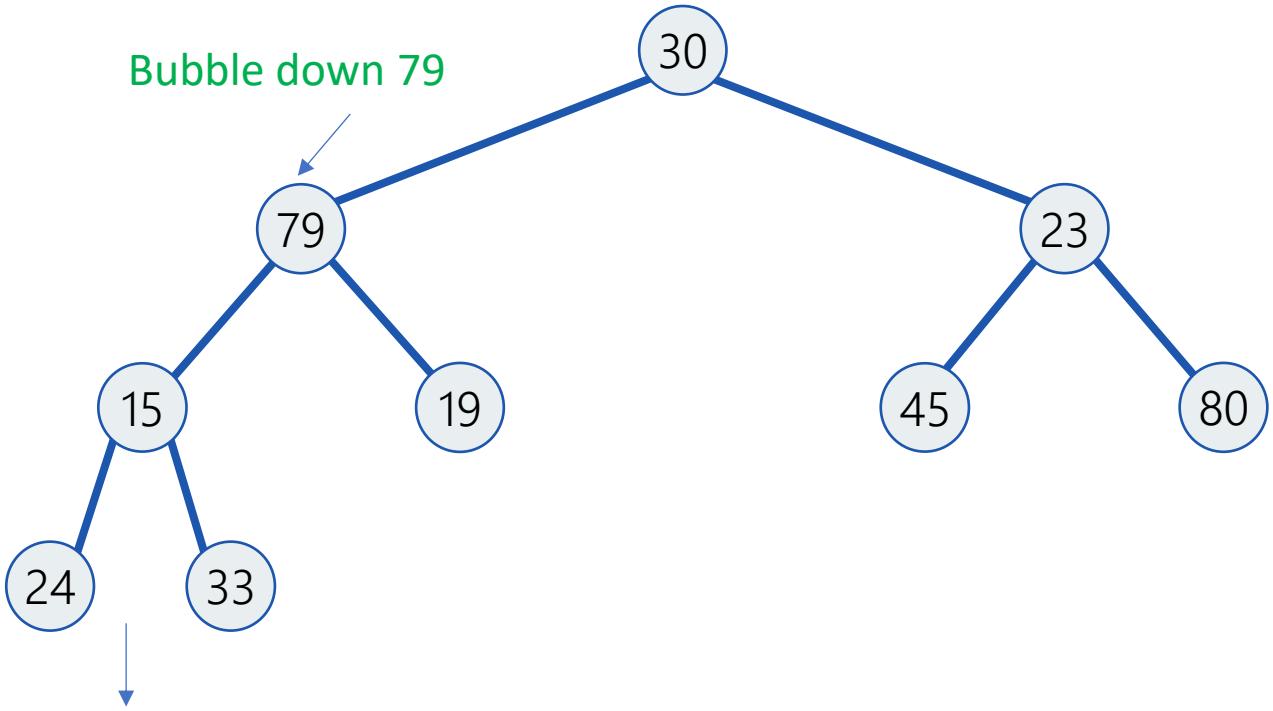


1	2	3	4	5	6	7	8	9
30	79	23	15	19	45	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

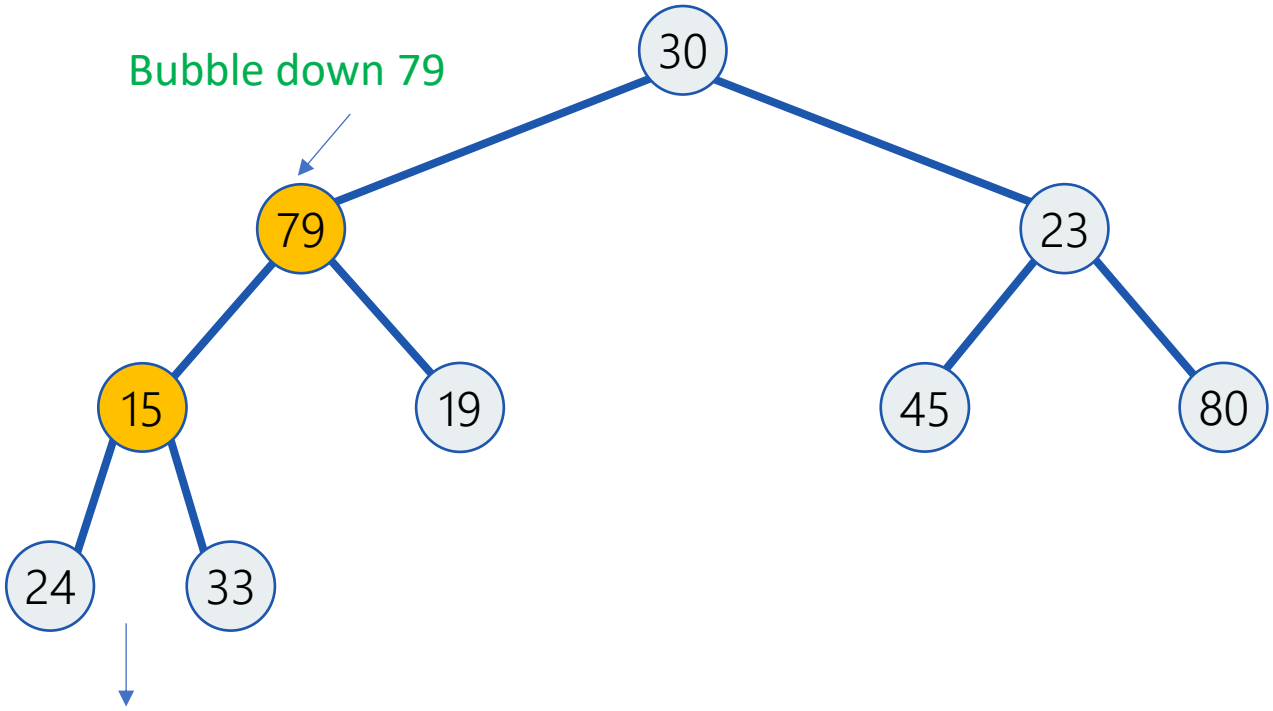


1	2	3	4	5	6	7	8	9
30	79	23	15	19	45	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

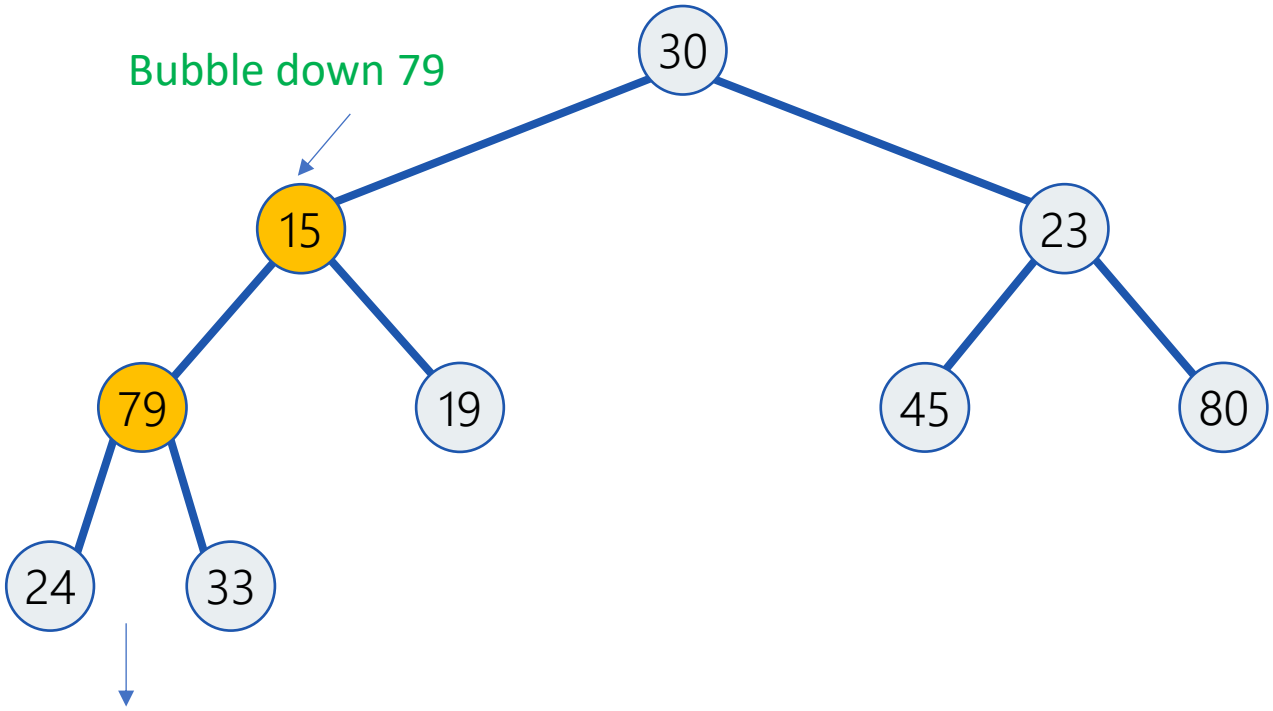


1	2	3	4	5	6	7	8	9
30	79	23	15	19	45	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

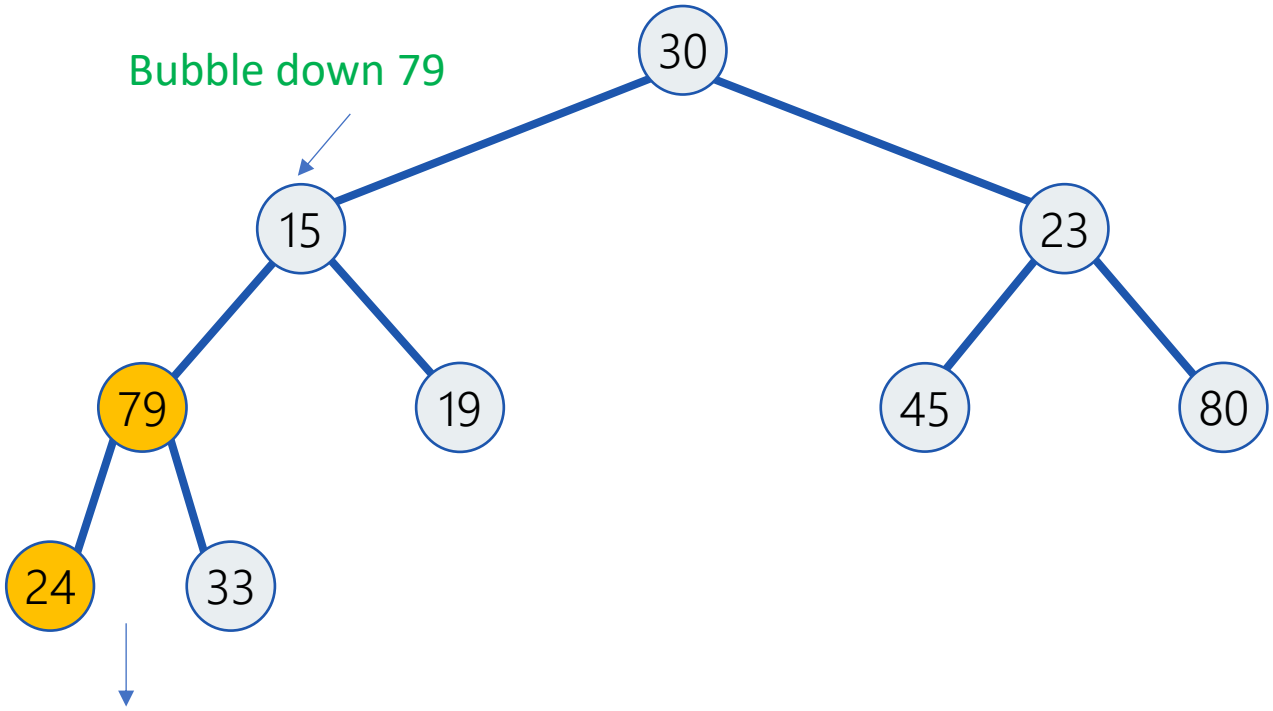


1	2	3	4	5	6	7	8	9
30	15	23	79	19	45	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

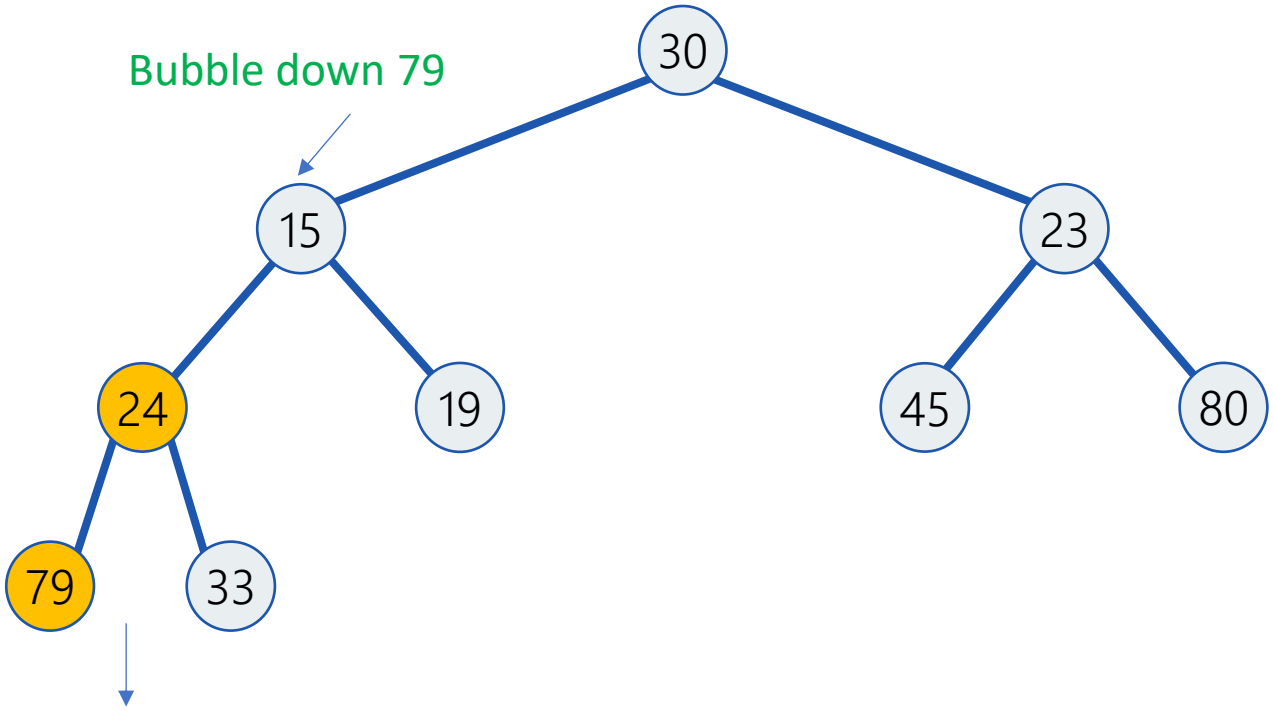


1	2	3	4	5	6	7	8	9
30	15	23	79	19	45	80	24	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

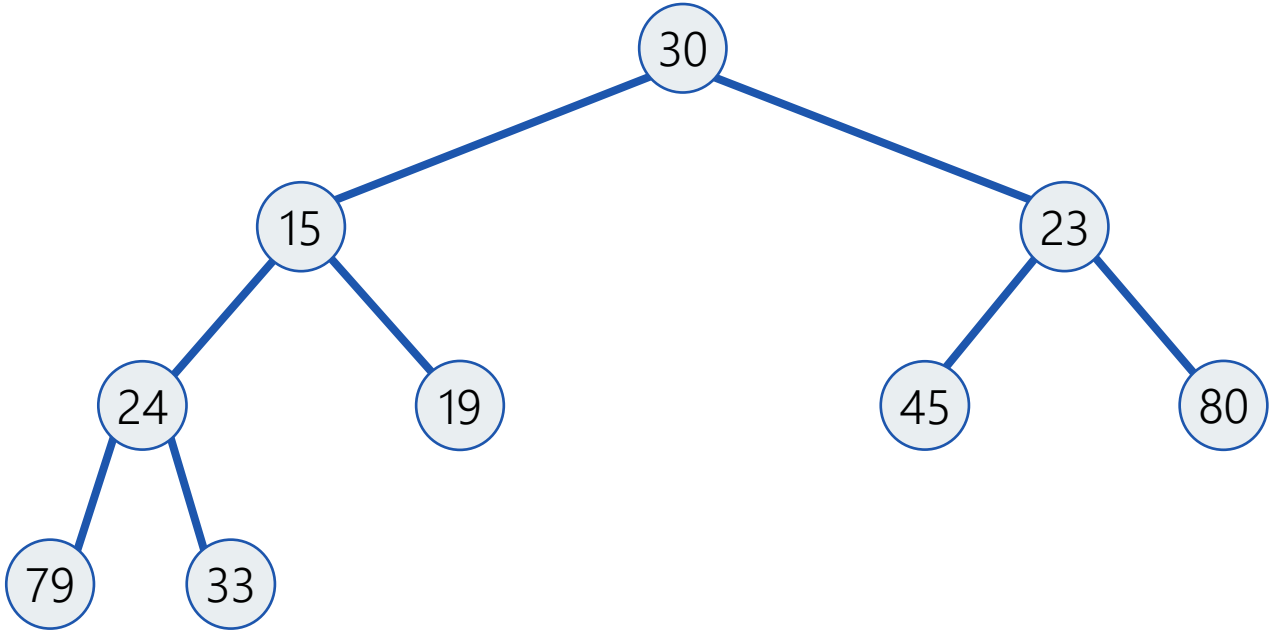


1	2	3	4	5	6	7	8	9
30	15	23	24	19	45	80	79	33

Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on

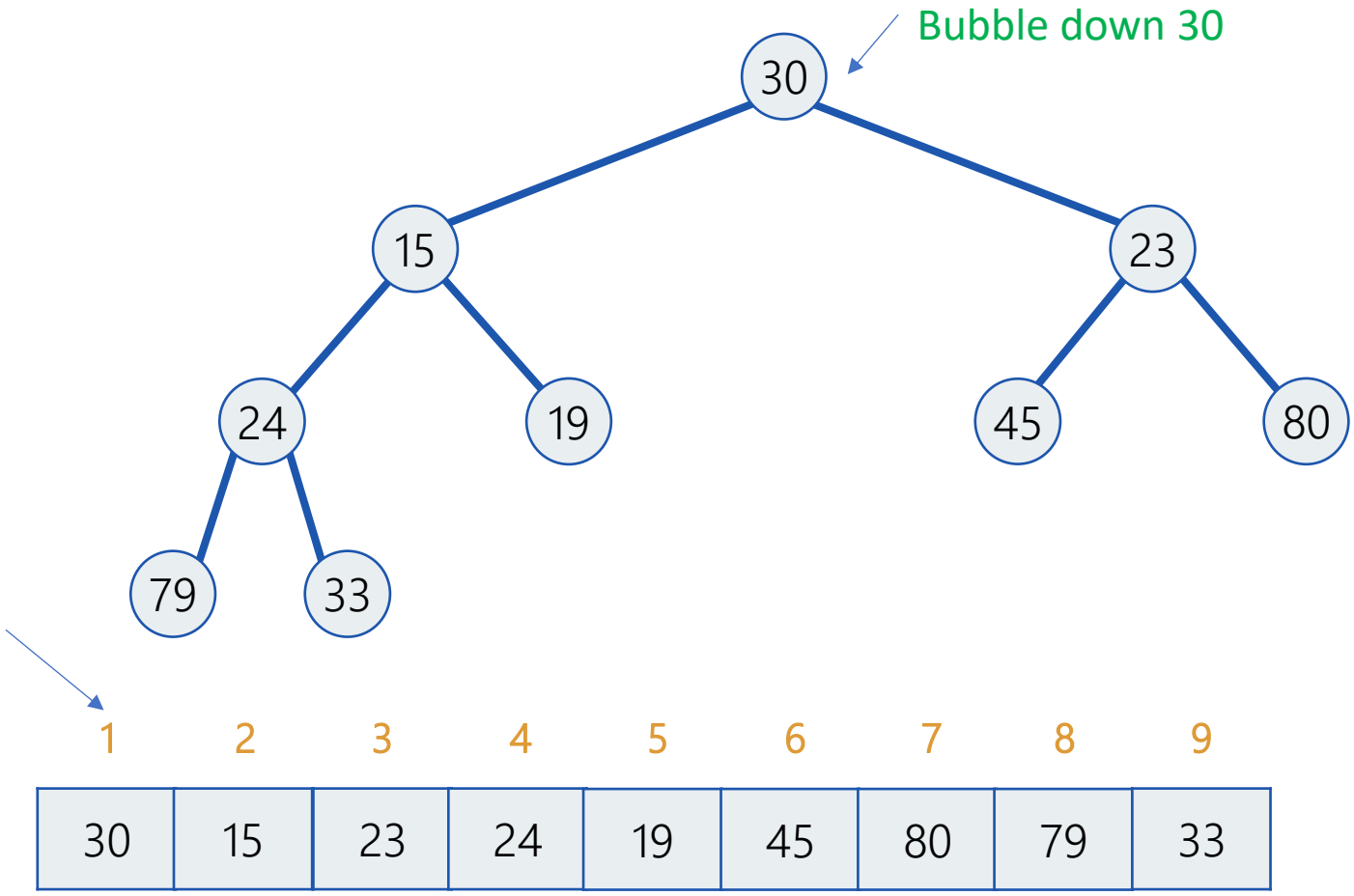


1	2	3	4	5	6	7	8	9
30	15	23	24	19	45	80	79	33

Example: build-heap with n elements

Reichman University

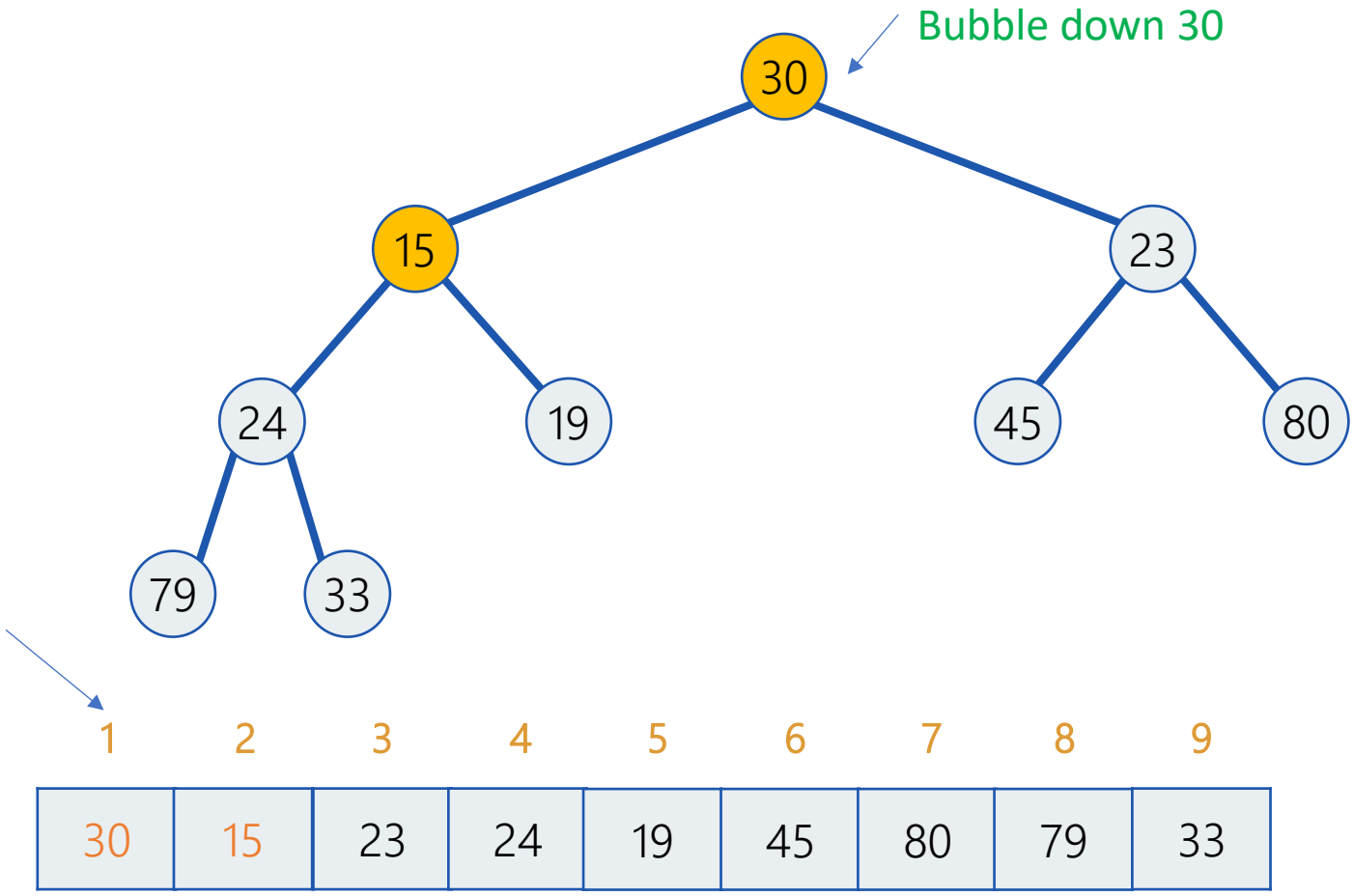
- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Example: build-heap with n elements

Reichman University

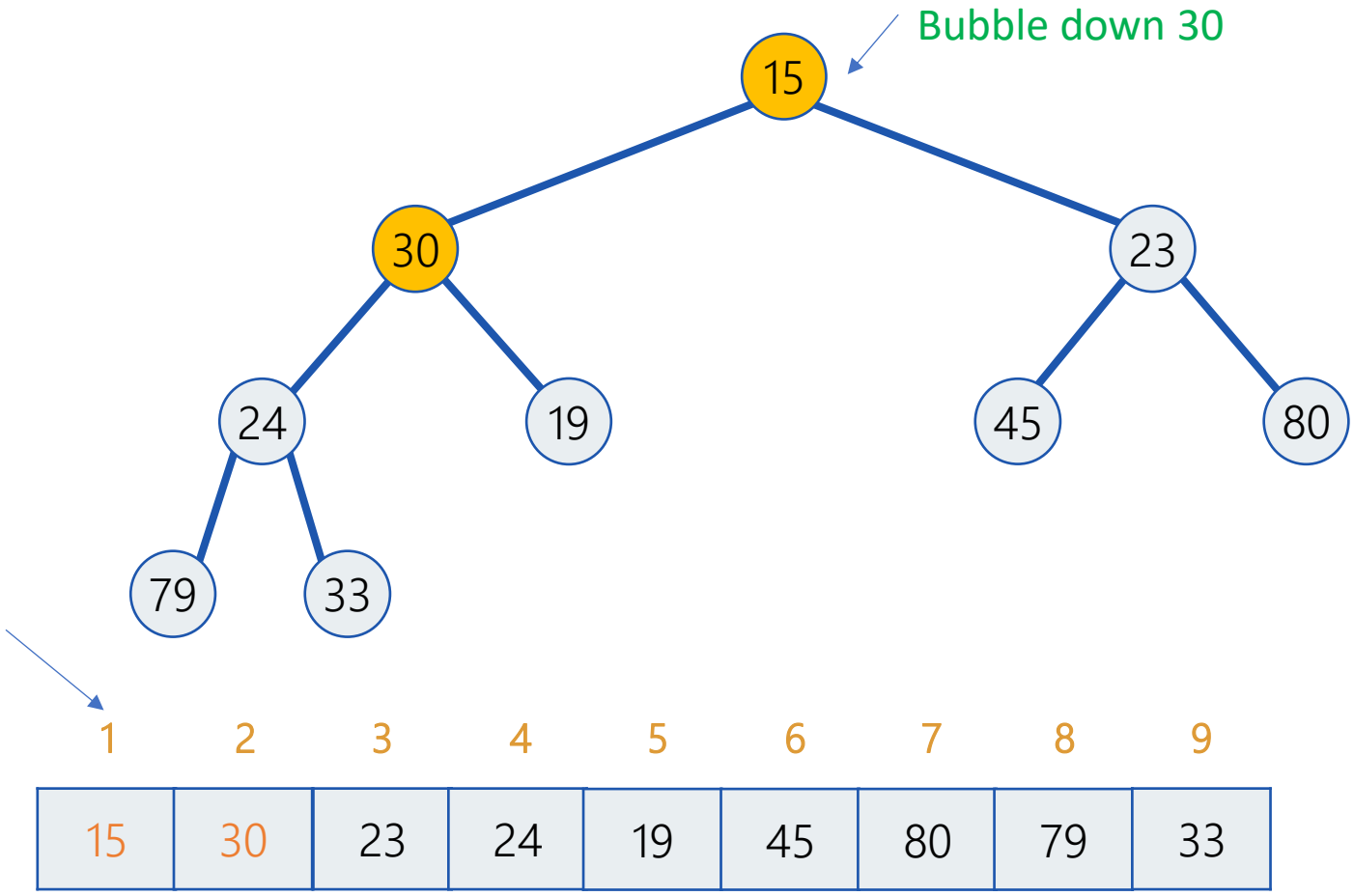
- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Example: build-heap with n elements

Reichman University

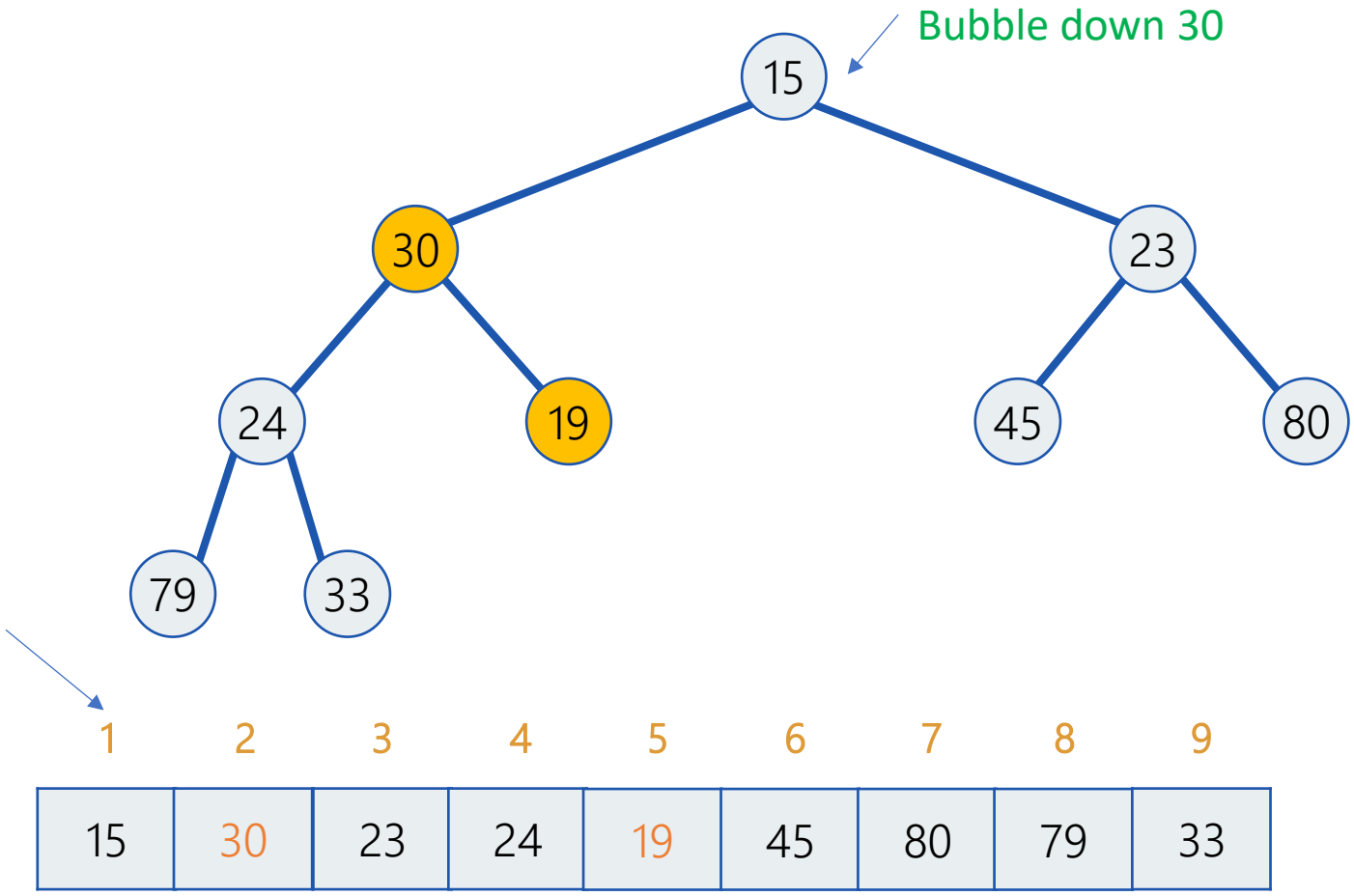
- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Example: build-heap with n elements

Reichman University

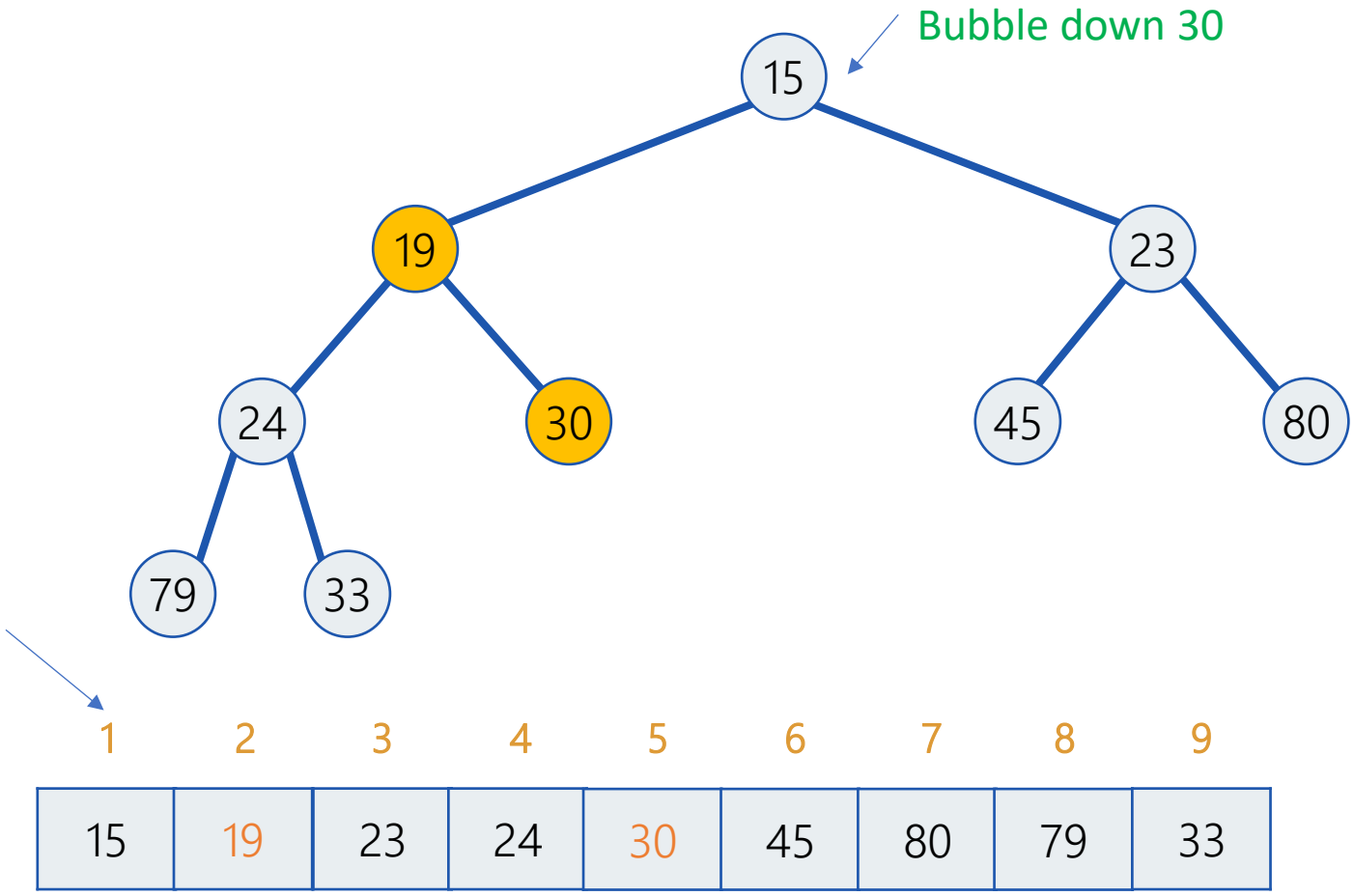
- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Example: build-heap with n elements

Reichman University

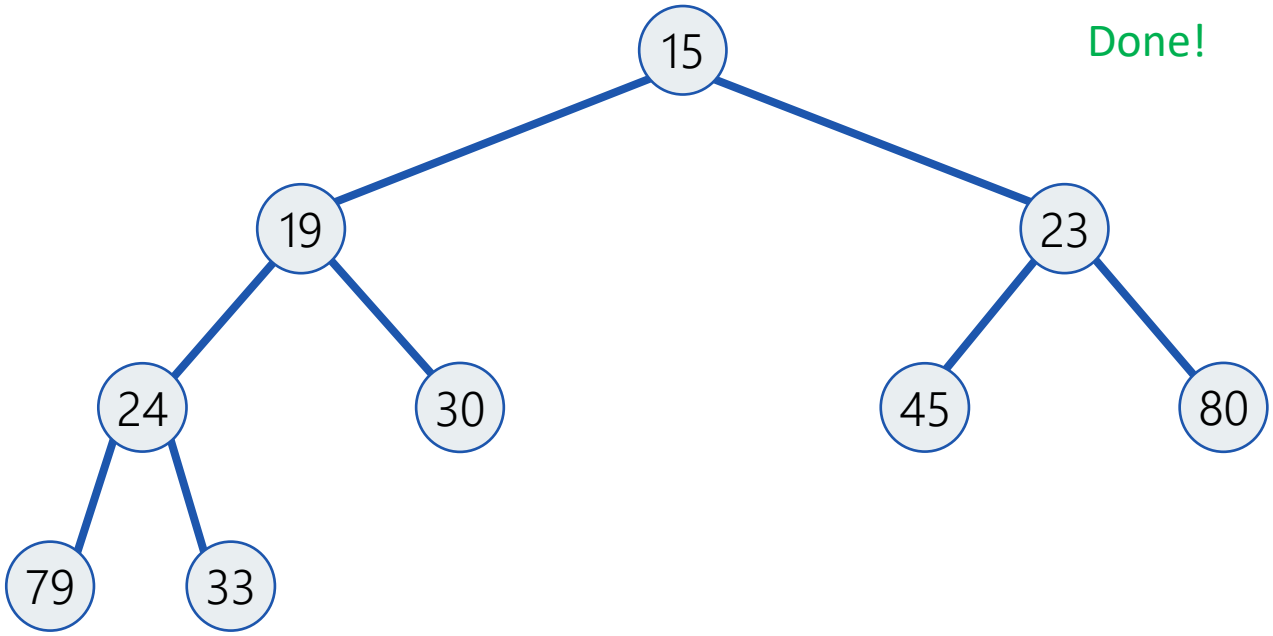
- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Example: build-heap with n elements

Reichman University

- Start with arbitrary array
- Fix heap property: Go over array from right to left bubbling down each node:
- starting with all leaves,
- then all nodes of height 1 then all of height 2 and so on



Done!

1	2	3	4	5	6	7	8	9
15	19	23	24	30	45	80	79	33

Heap sort

Reichman University

- sorting an array using a binary heap:
  - build-heap in O(n) time
  - do n delete-min() operations to get the elements in increasing order
- total time $O(n \log n)$
- no extra storage needed
  - Let *k* be the number of elements in the heap
  - Let *m* be the min value
  - After delete-min() we place *m* in position *k* in the array
  - Reverse the array after *all* delete-min() ops

Binary Heap: Union

Union.

- Combine two binary heaps H_1 and H_2 into a single heap.
- No easy solution.
  - $\Omega(N)$ operations apparently required
- Can support fast union with fancier heaps.

Binomial heaps, Fibonacci heaps