

CSC 212

Data Structures and Abstractions

Spring 2020
Prof. Marco Alvarez

Building a Heap in Linear Time and Heapsort

Performance

| | Sorted Array/List | Unsorted Array/ List | Heap |
|-----------|-------------------|-------------------------|----------|
| insert | O(n) | O(1) | O(log n) |
| removeMax | O(1) | O(n) | O(log n) |
| max | O(1) | O(n) | O(1) |
| insert N | O(n^2) | O(n) | O(n)** |

(**) assuming we know the sequence in advance (**buildHeap**)

buildHeap

Problem

Build a heap by inserting a sequence of n elements

'easy': call **insert** n times

Problem

Build a heap by inserting a sequence of n elements

'easy': call **insert** n times

$O(n \log n)$

Problem

Build a heap by inserting a sequence of n elements

'easy': call **insert** n times

$O(n \log n)$

Can we do it in **linear time**?

buildHeap

Place **n** items into the tree (array) in **any order**

keeps structure property

buildHeap

Place **n** items into the tree (array) in **any order**

keeps structure property

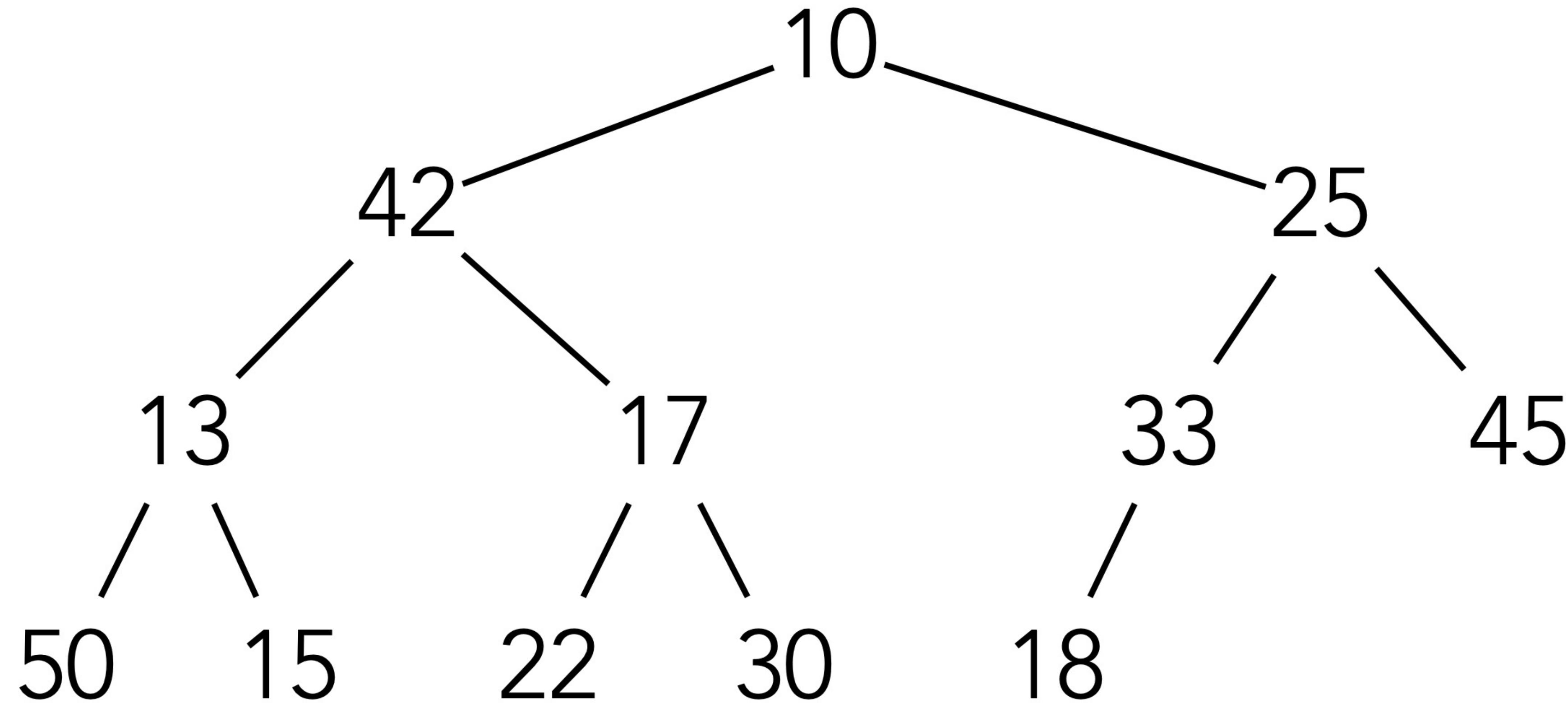
Perform **Down-Heap** on each **internal node**

from $\text{parent}(n)$ to 1

keeps heap-order property

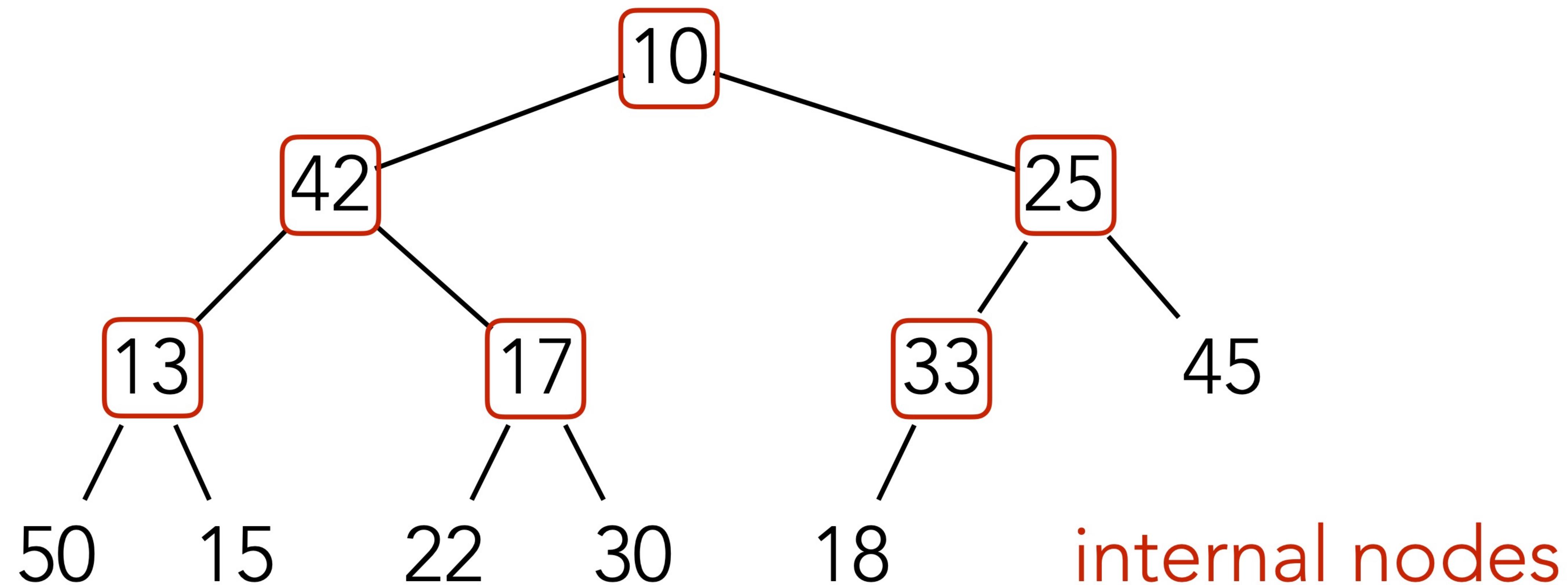
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



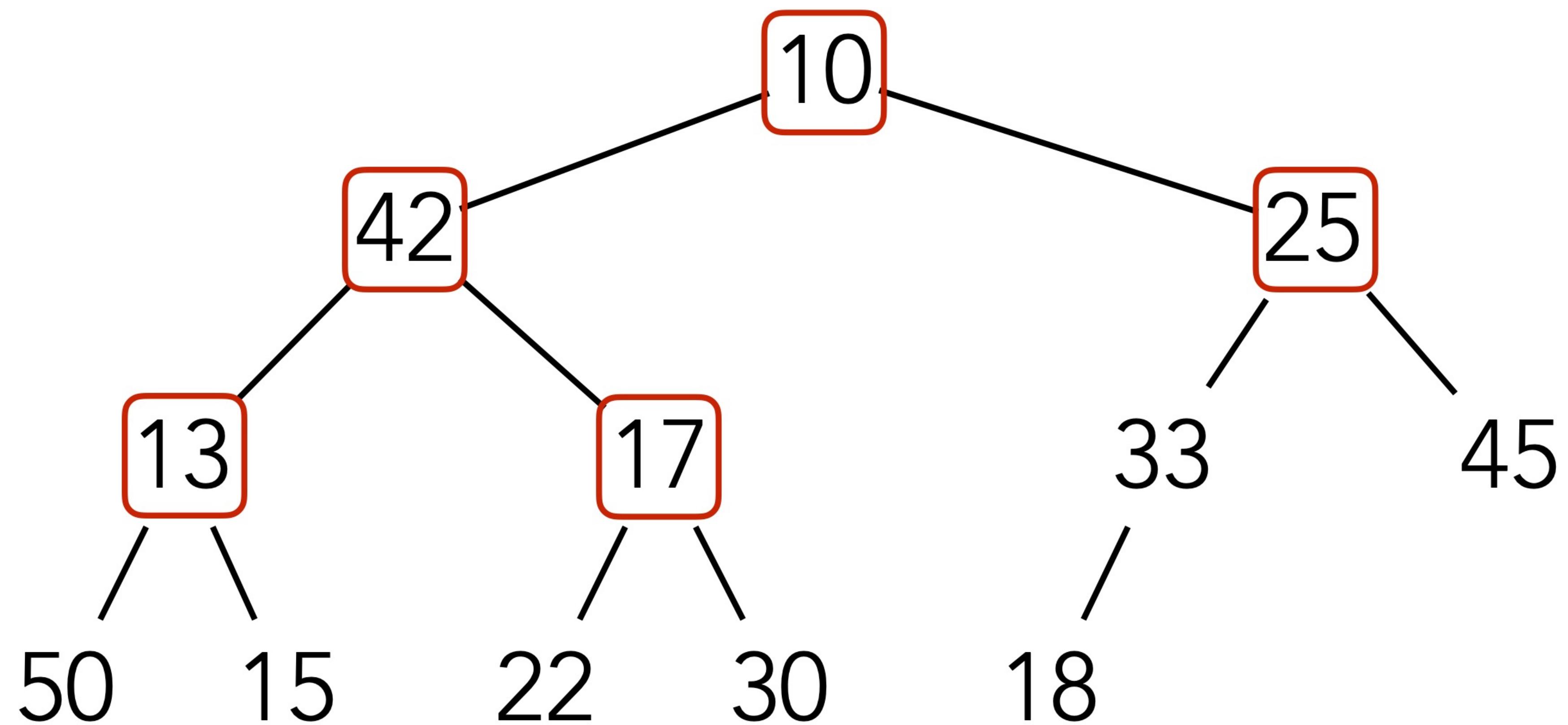
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



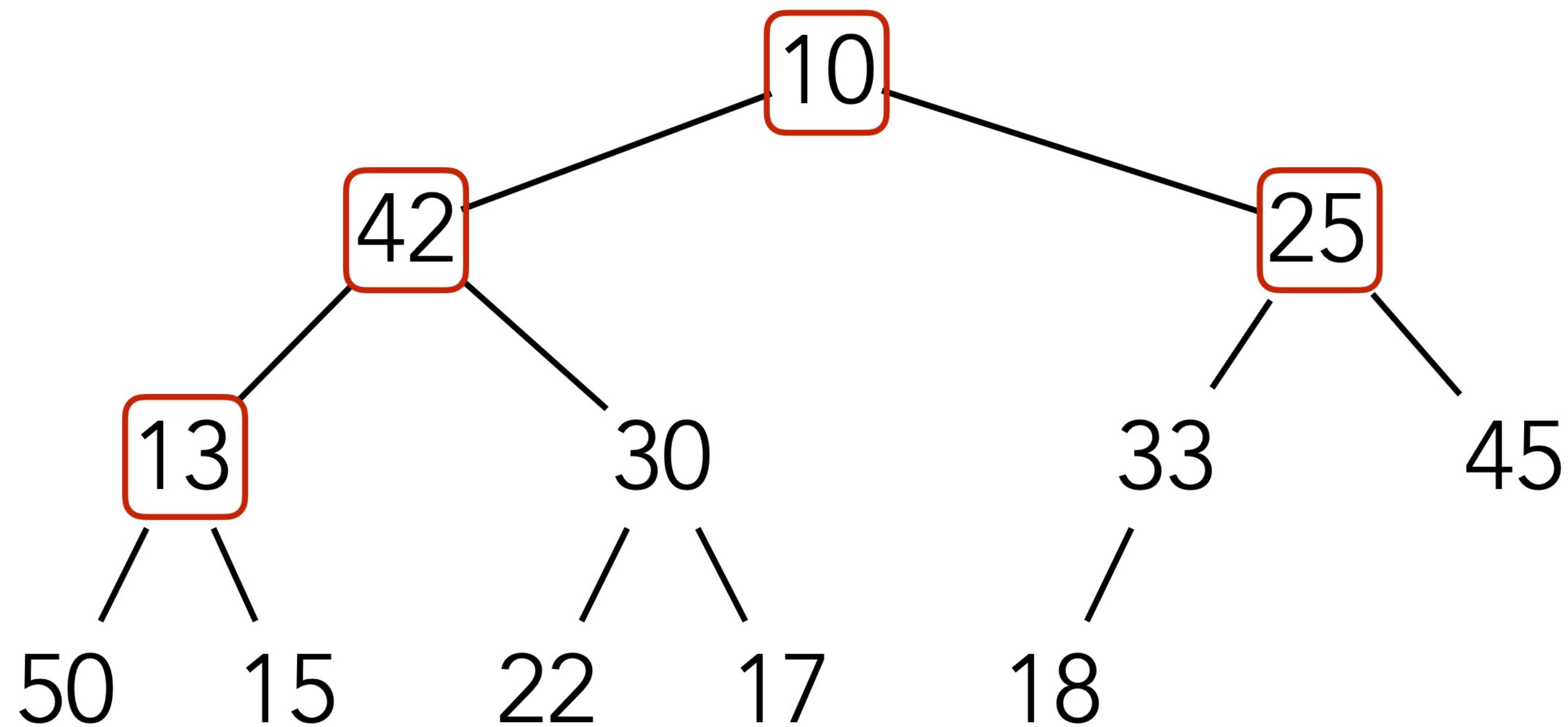
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



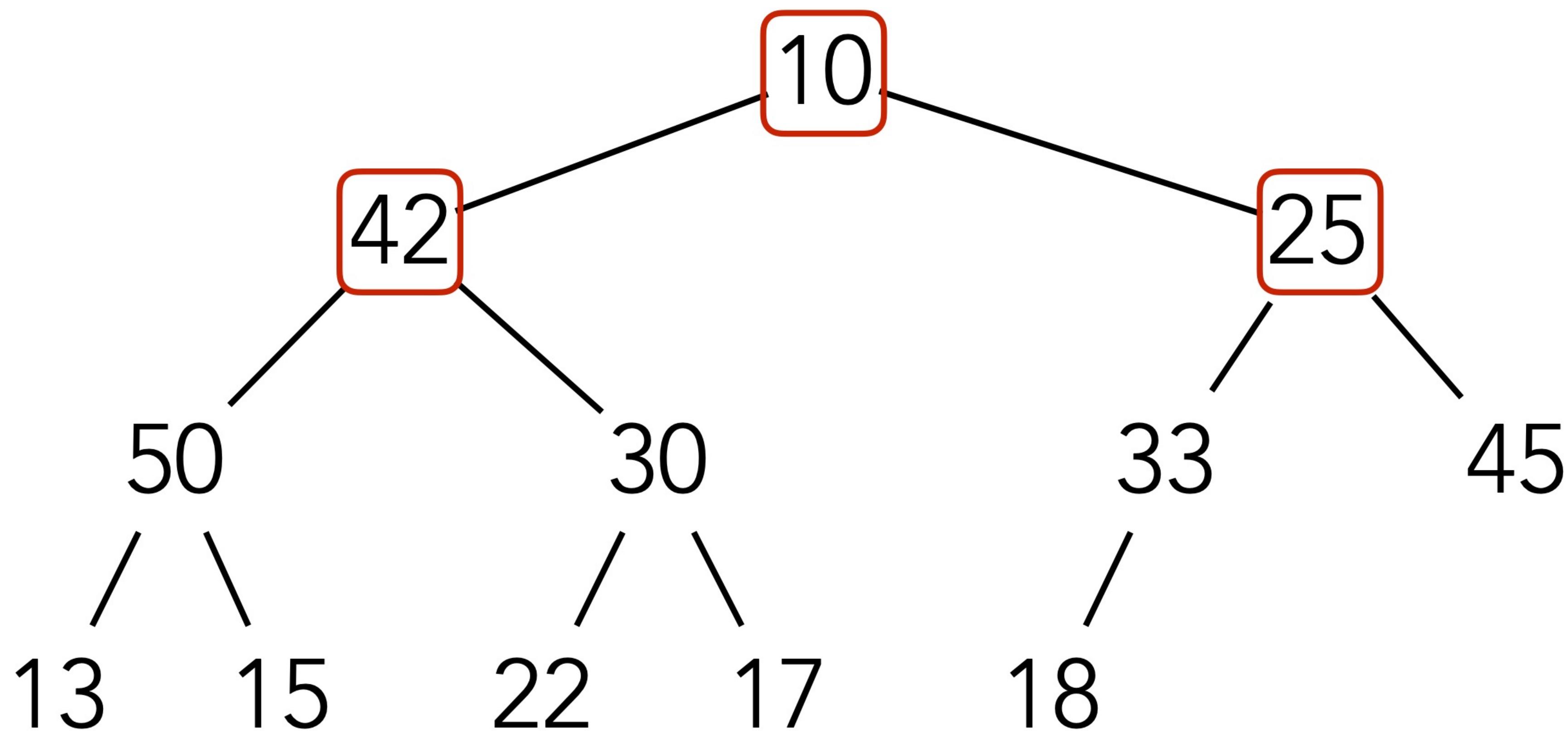
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



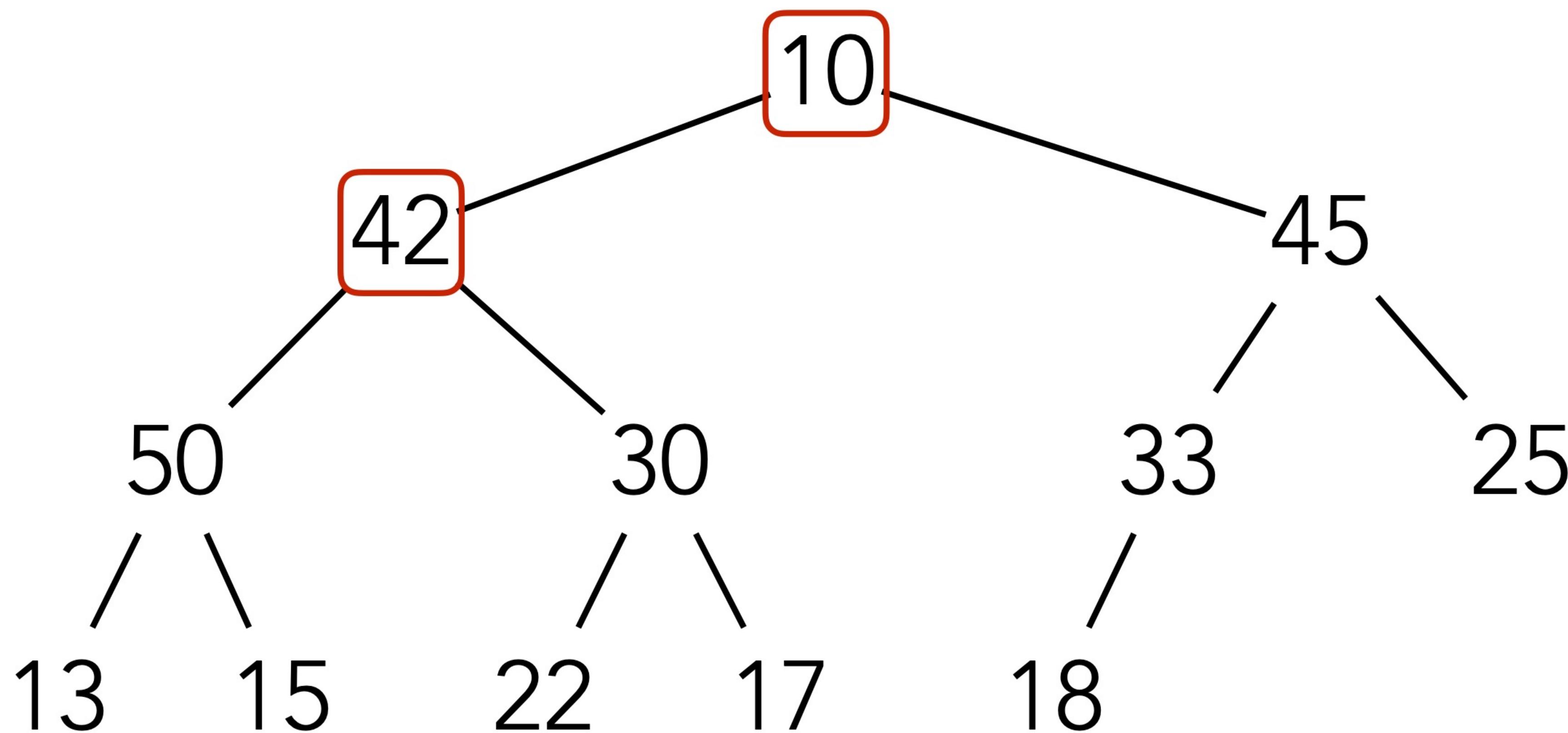
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



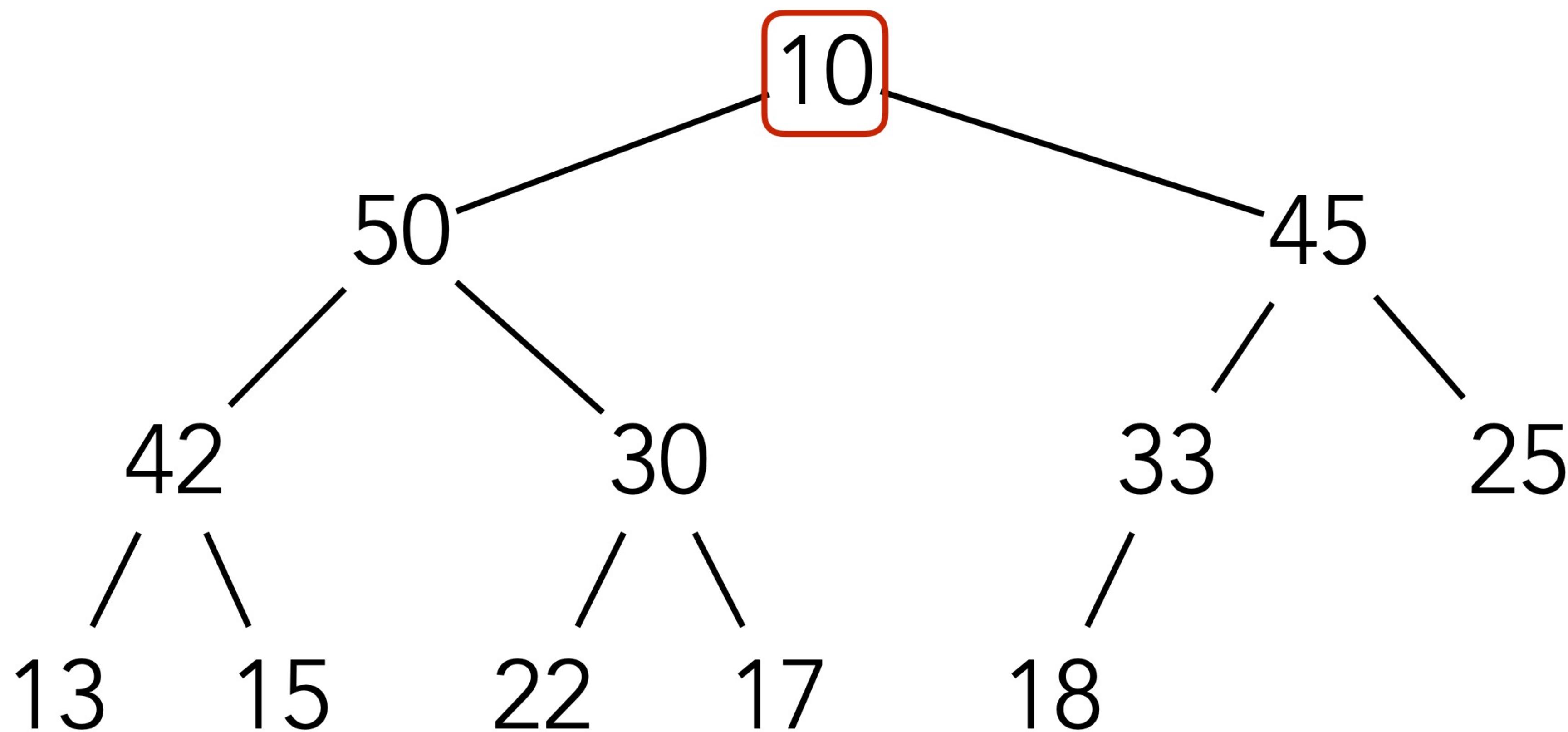
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



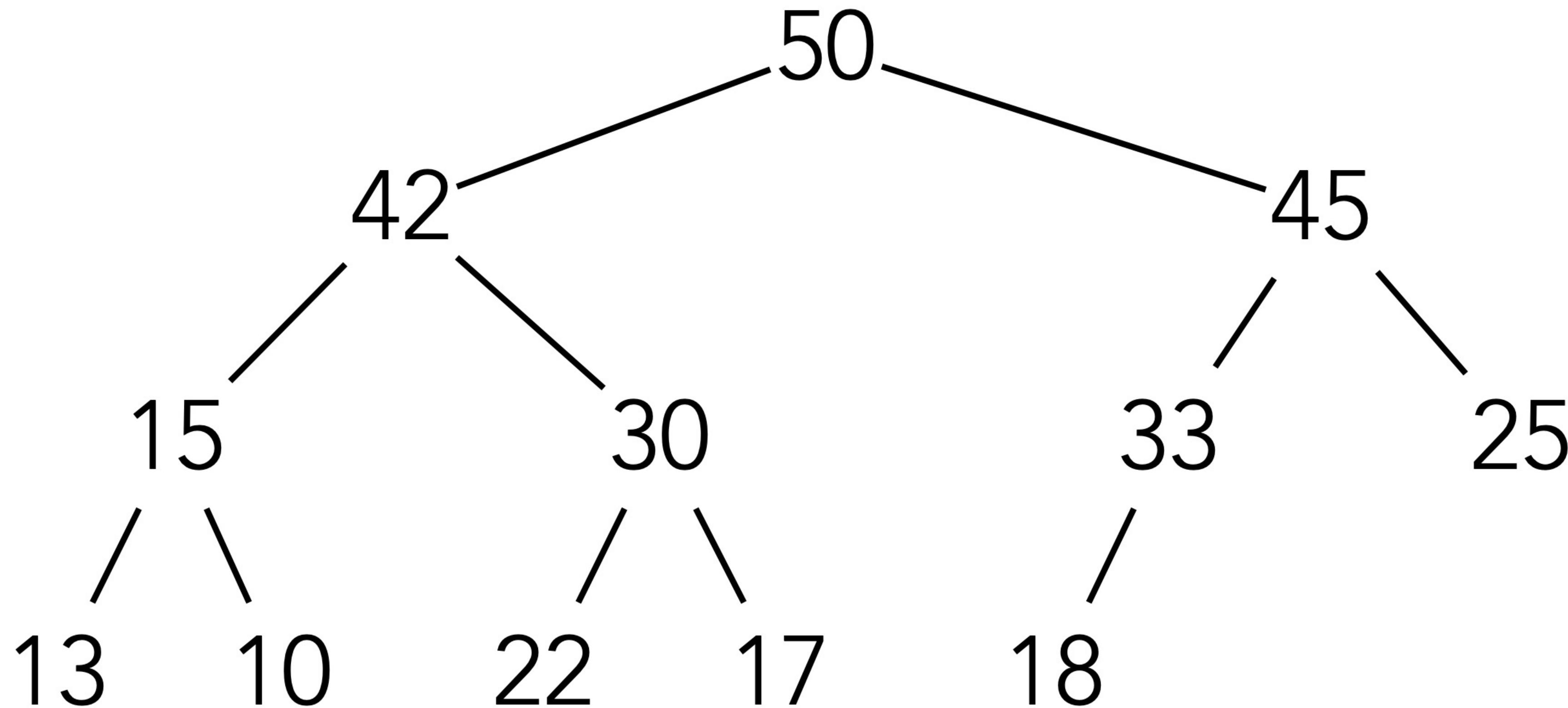
buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



buildHeap example

input: 10 42 25 13 17 33 45 50 15 22 30 18



Analysis

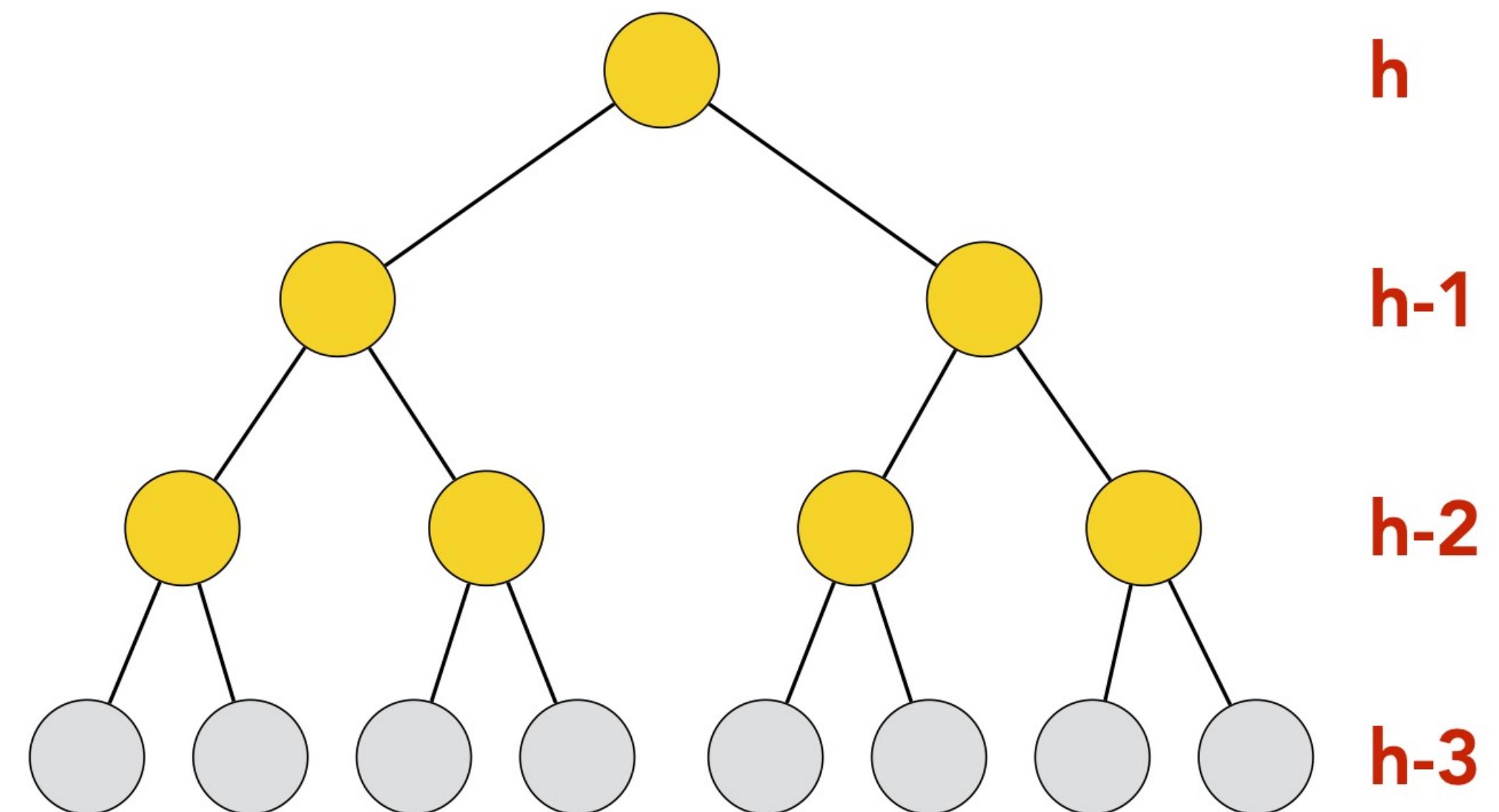
Cost is **sum of the heights** of all internal nodes

assume tree is full and complete, thus **$n=2^{h+1}-1$**

Analysis

Cost is **sum of the heights** of all internal nodes

assume tree is full and complete, thus $n=2^{h+1}-1$



Analysis

Cost is **sum of the heights** of all internal nodes

assume tree is full and complete, thus $n=2^{h+1}-1$

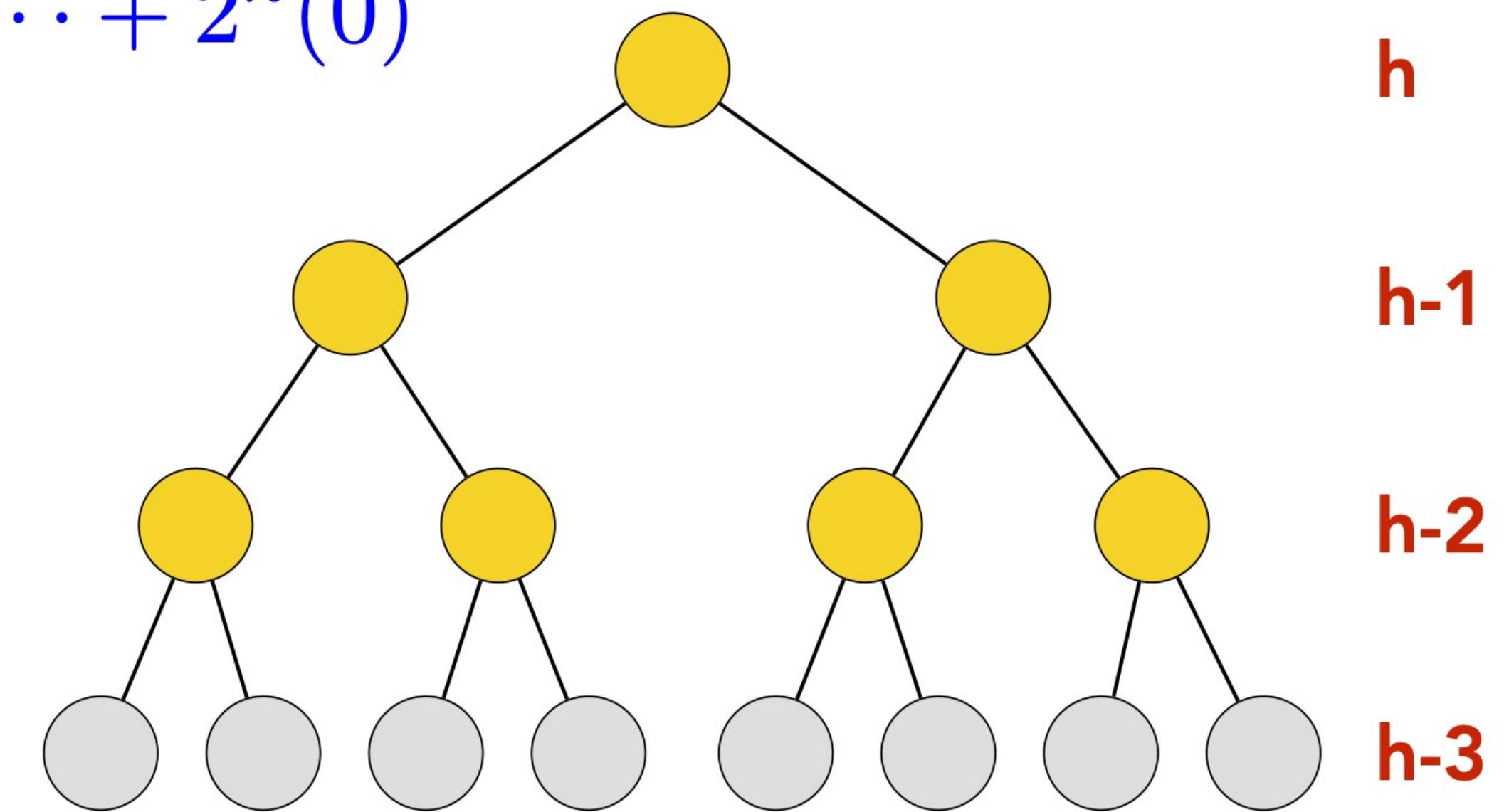
$$T(n) = h + 2(h - 1) + 4(h - 2) + 8(h - 3) + \dots + 2^h(0)$$

$$= \sum_{i=0}^h 2^i(h - i) = h \sum_{i=0}^h 2^i - \sum_{i=0}^h i2^i$$

$$= h [2^{h+1} - 1] - [2 + (h + 1 - 2)2^{h+1}]$$

$$= 2^{h+1} - h - 2 = n - (h + 1)$$

$$= O(n)$$



Analysis

Cost is **sum of the heights** of all internal nodes

assume tree is full and complete, thus $n=2^{h+1}-1$

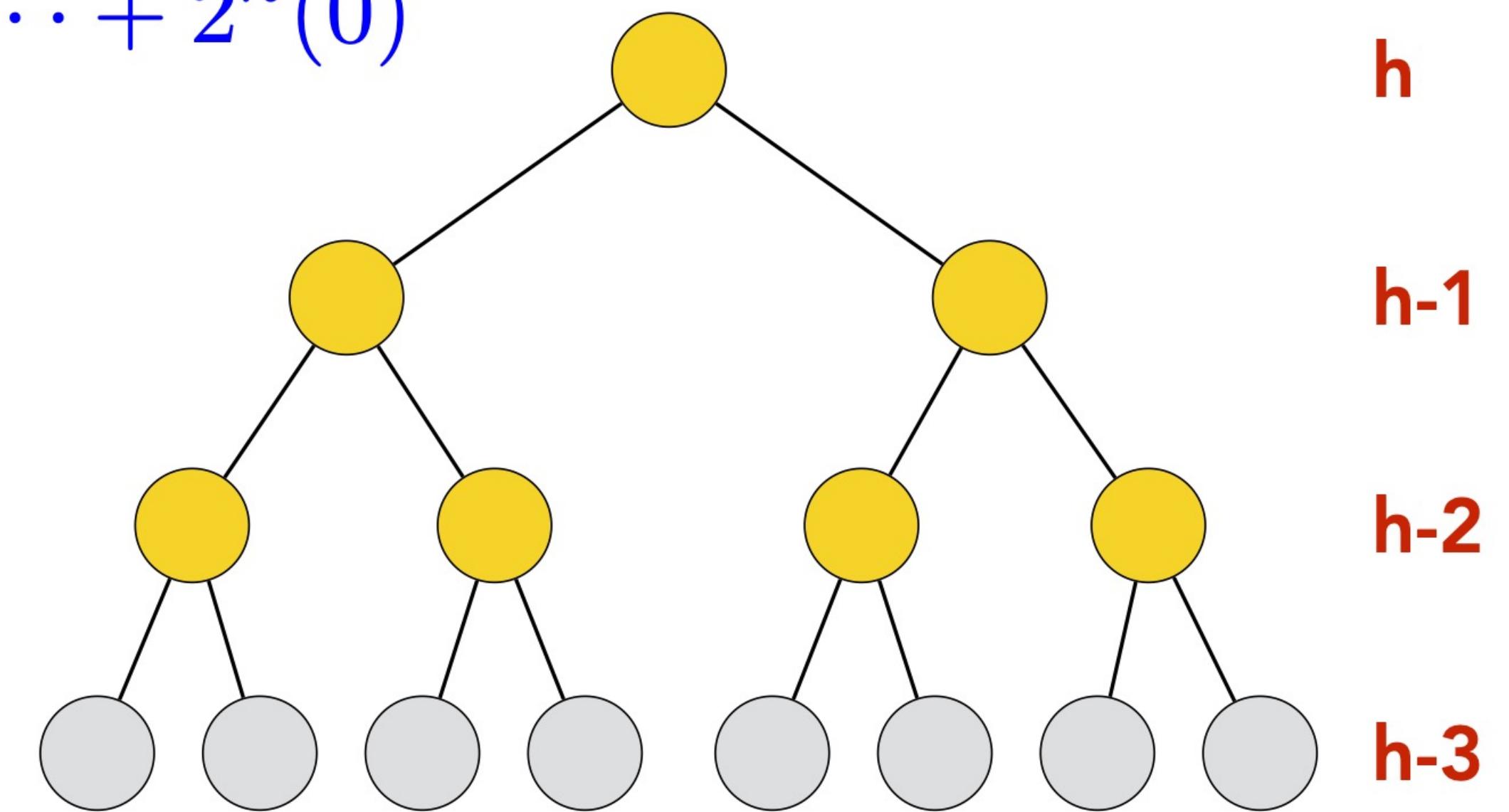
$$T(n) = h + 2(h - 1) + 4(h - 2) + 8(h - 3) + \dots + 2^h(0)$$

$$= \sum_{i=0}^h 2^i(h - i) = h \sum_{i=0}^h 2^i - \sum_{i=0}^h i2^i$$

$$= h [2^{h+1} - 1] - [2 + (h + 1 - 2)2^{h+1}]$$

$$= 2^{h+1} - h - 2 = n - (h + 1)$$

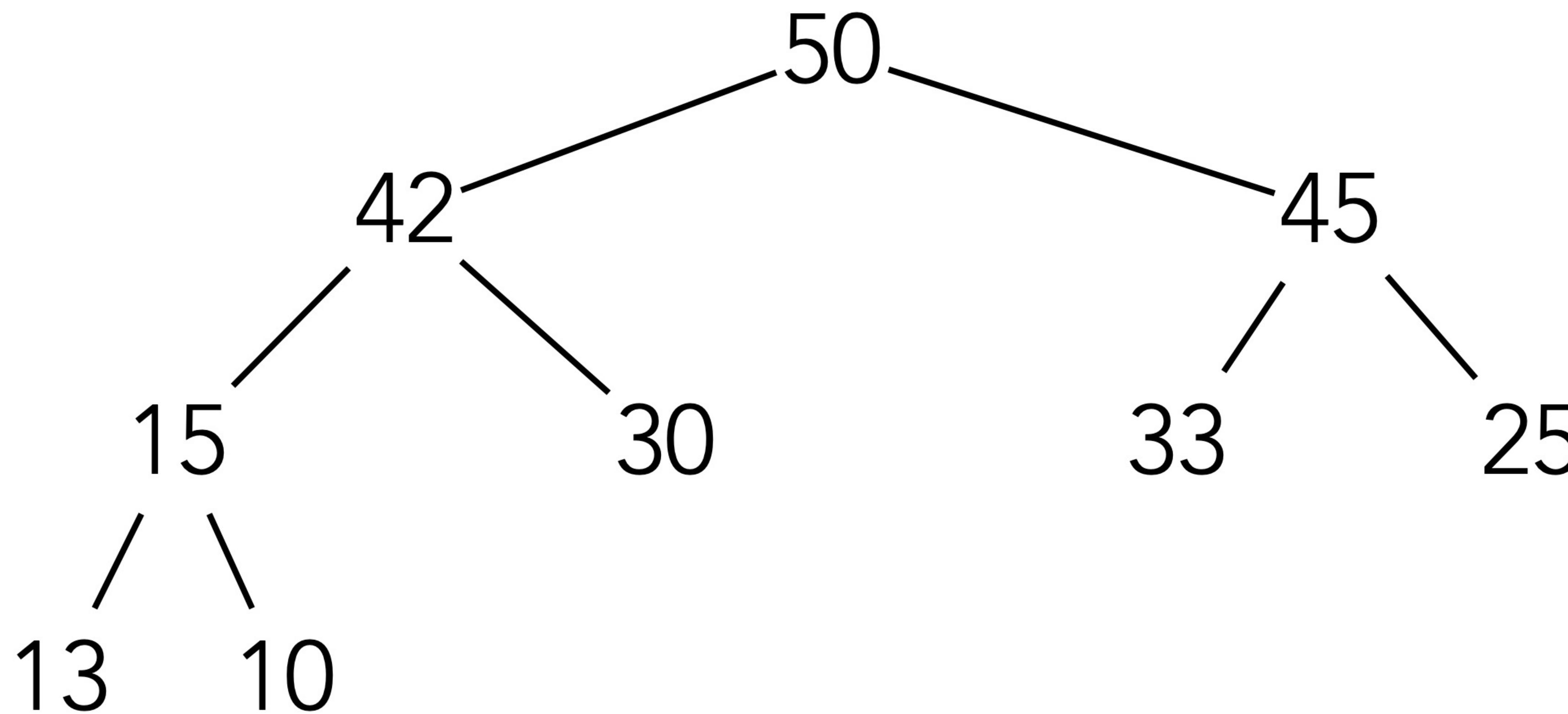
$$= O(n)$$



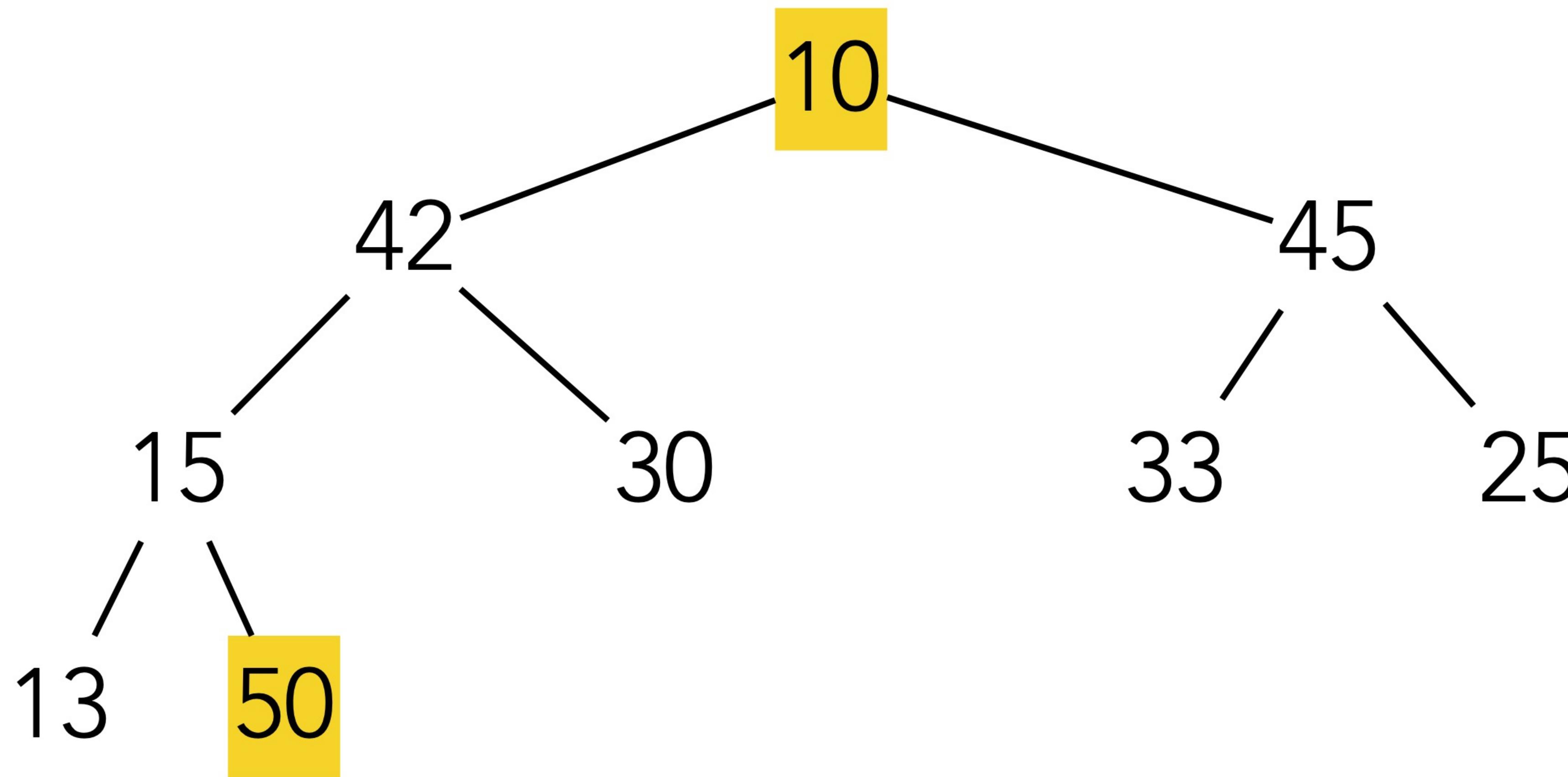
HeapSort Algorithm

1. apply **buildHeap** to input array **A**
2. swap **first** with **last** element
now the max element is at the end of **A**
3. decrement heap-size
4. apply **downHeap** to new **first** element
5. if heap is not empty go to step 2

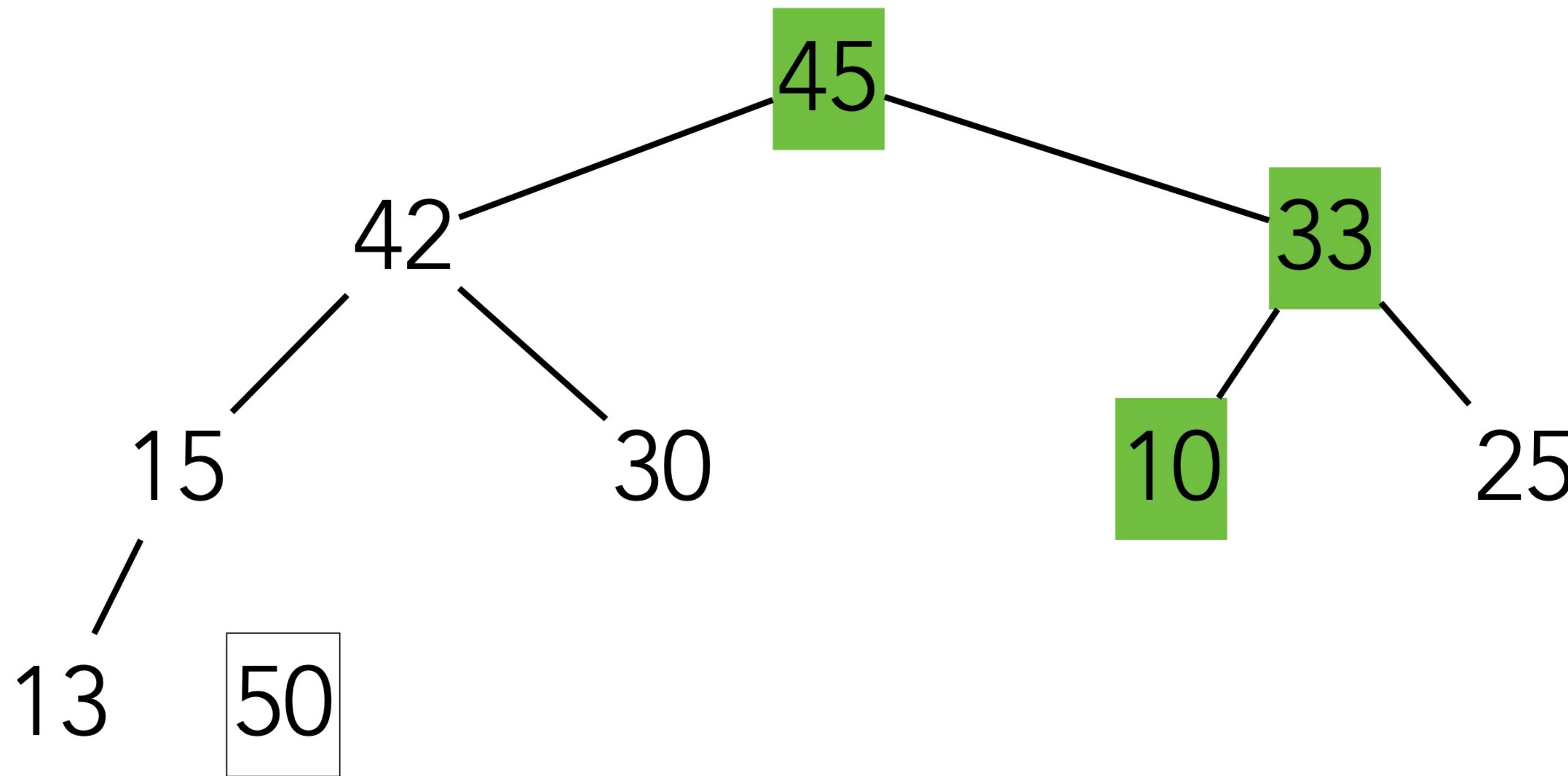
heapSort



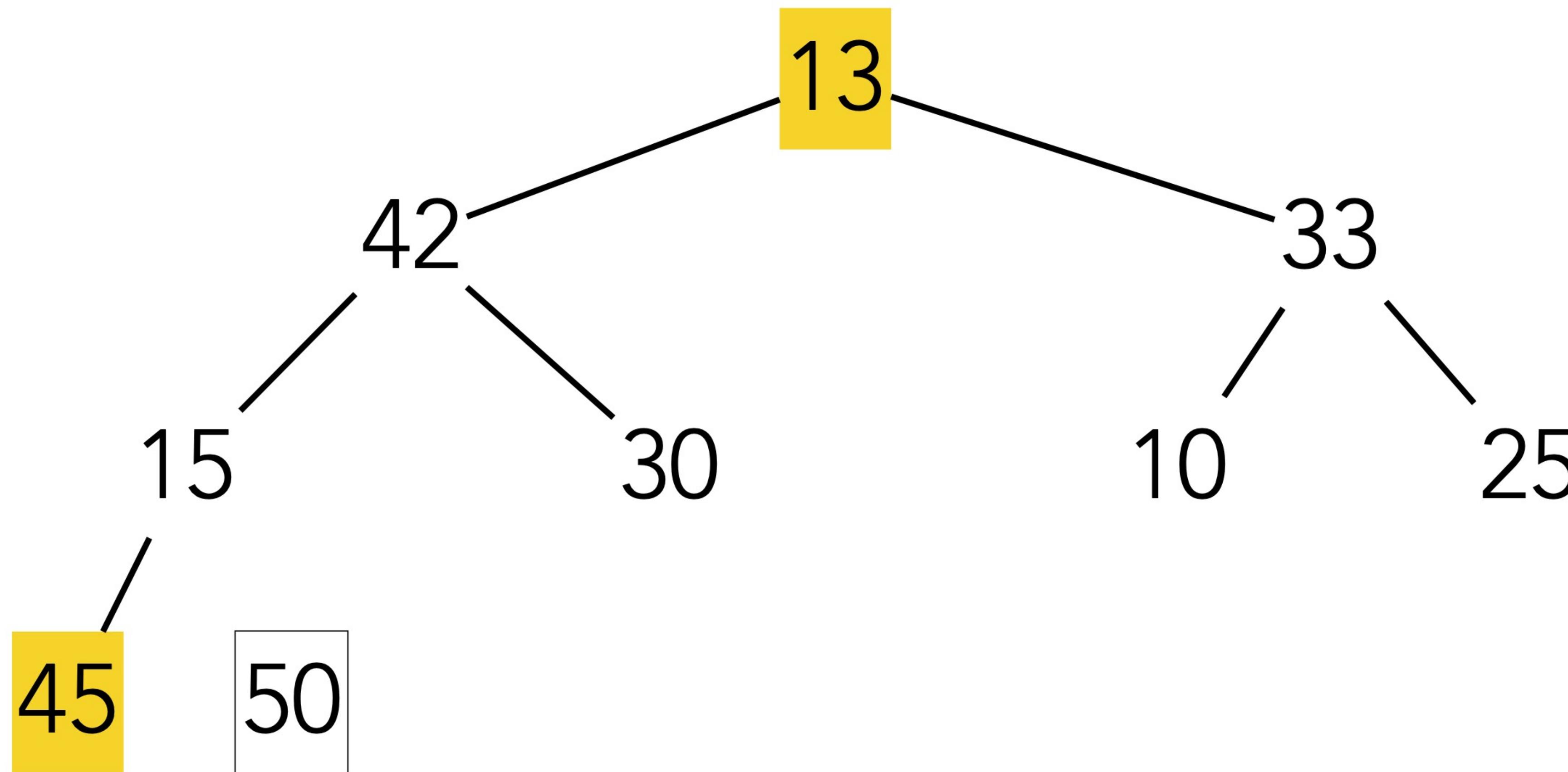
heapSort



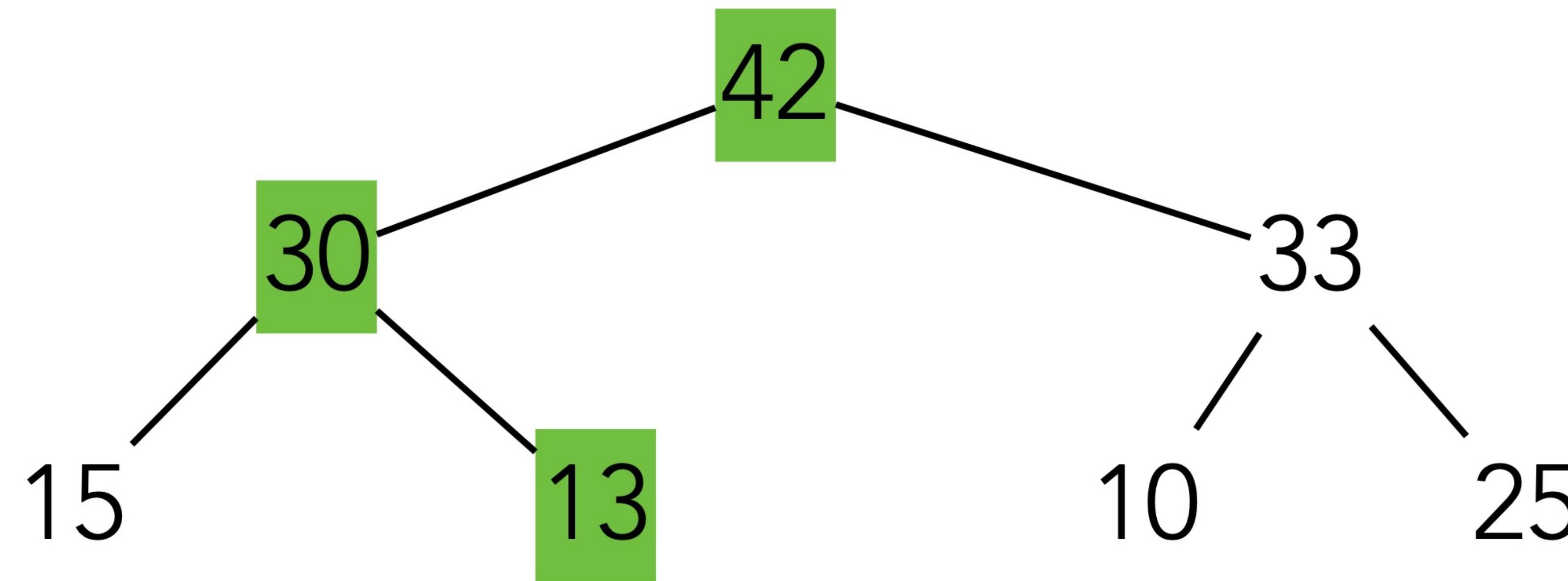
heapSort



heapSort



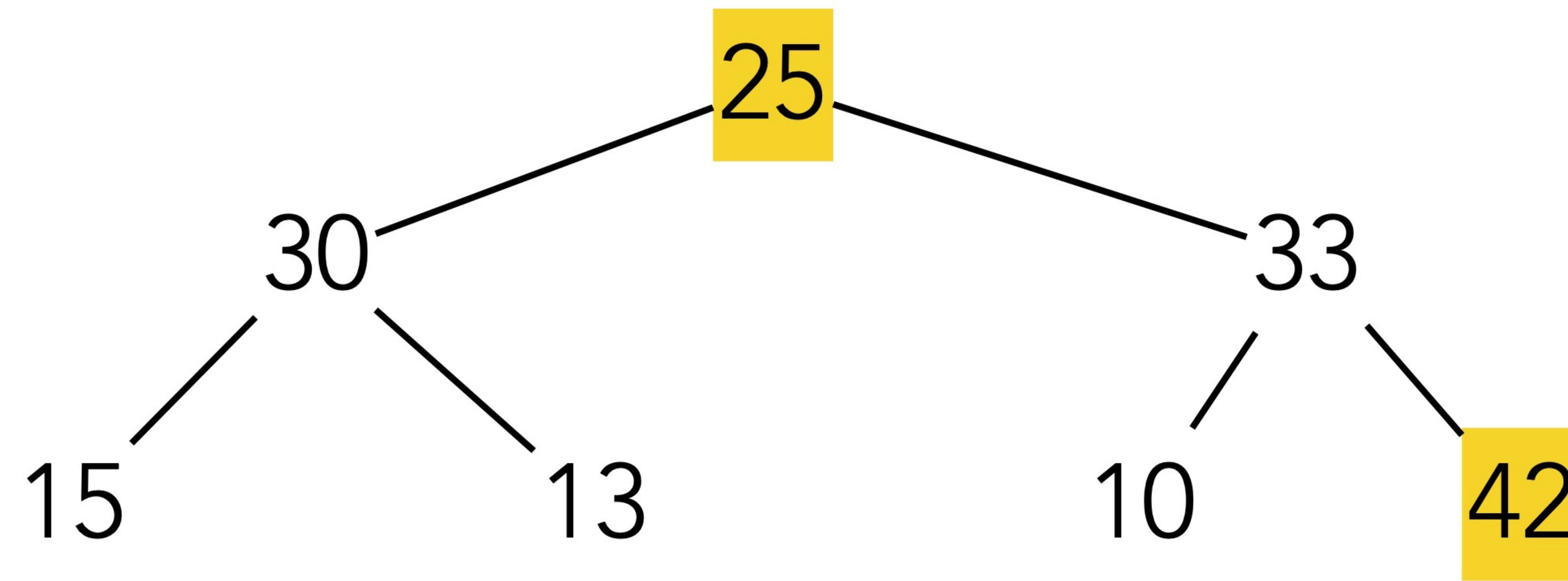
heapSort



45

50

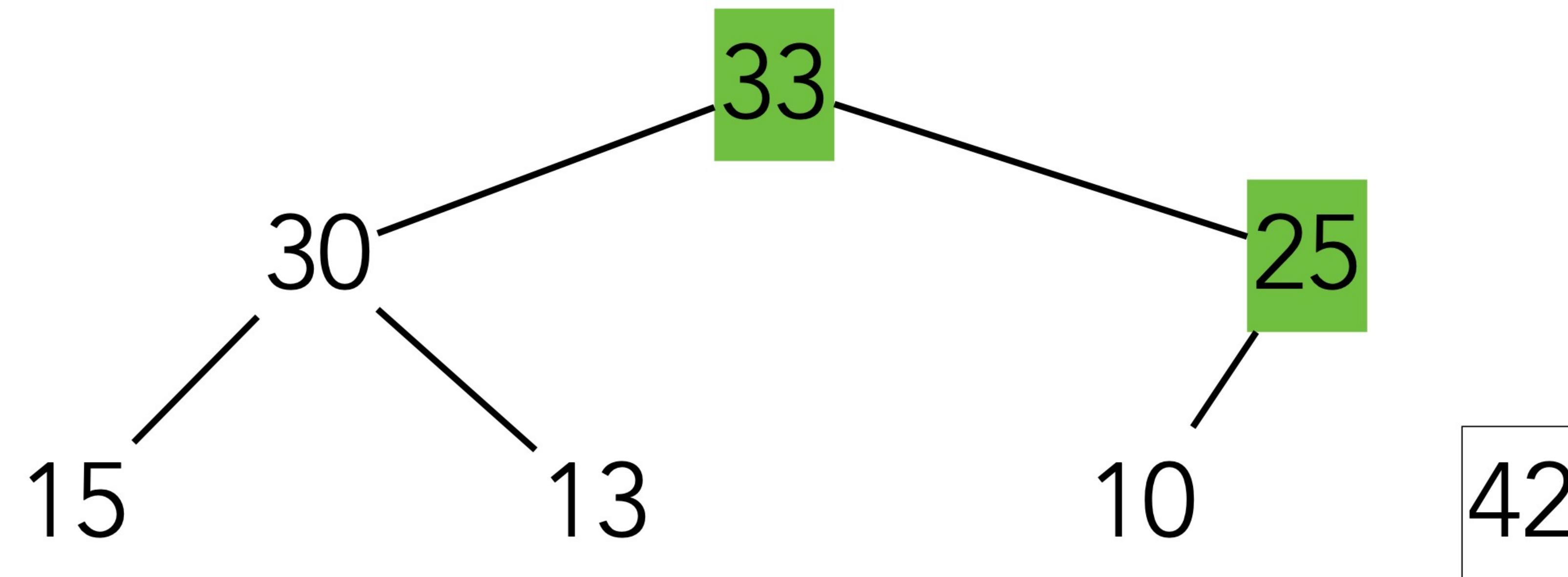
heapSort



45

50

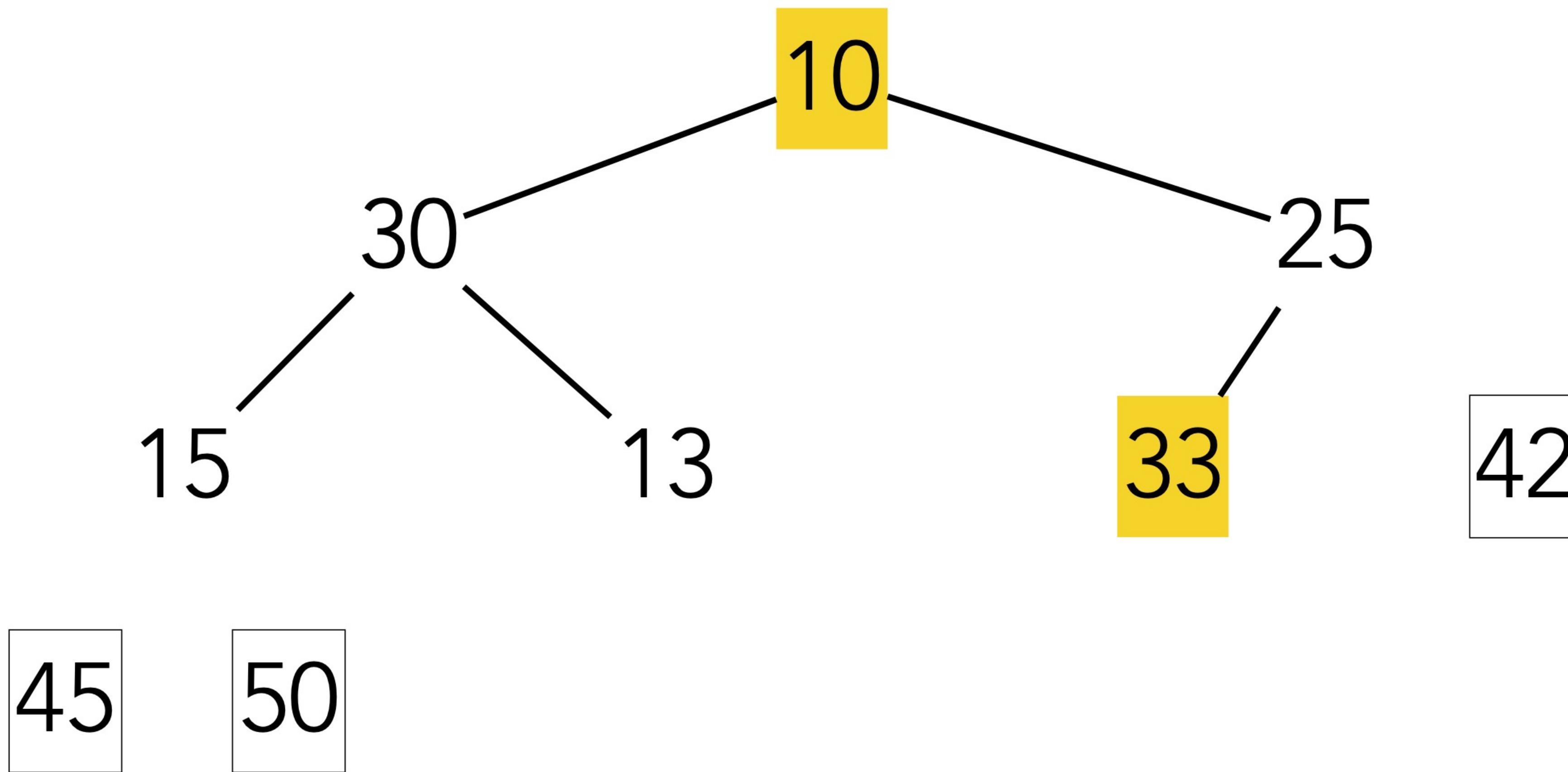
heapSort



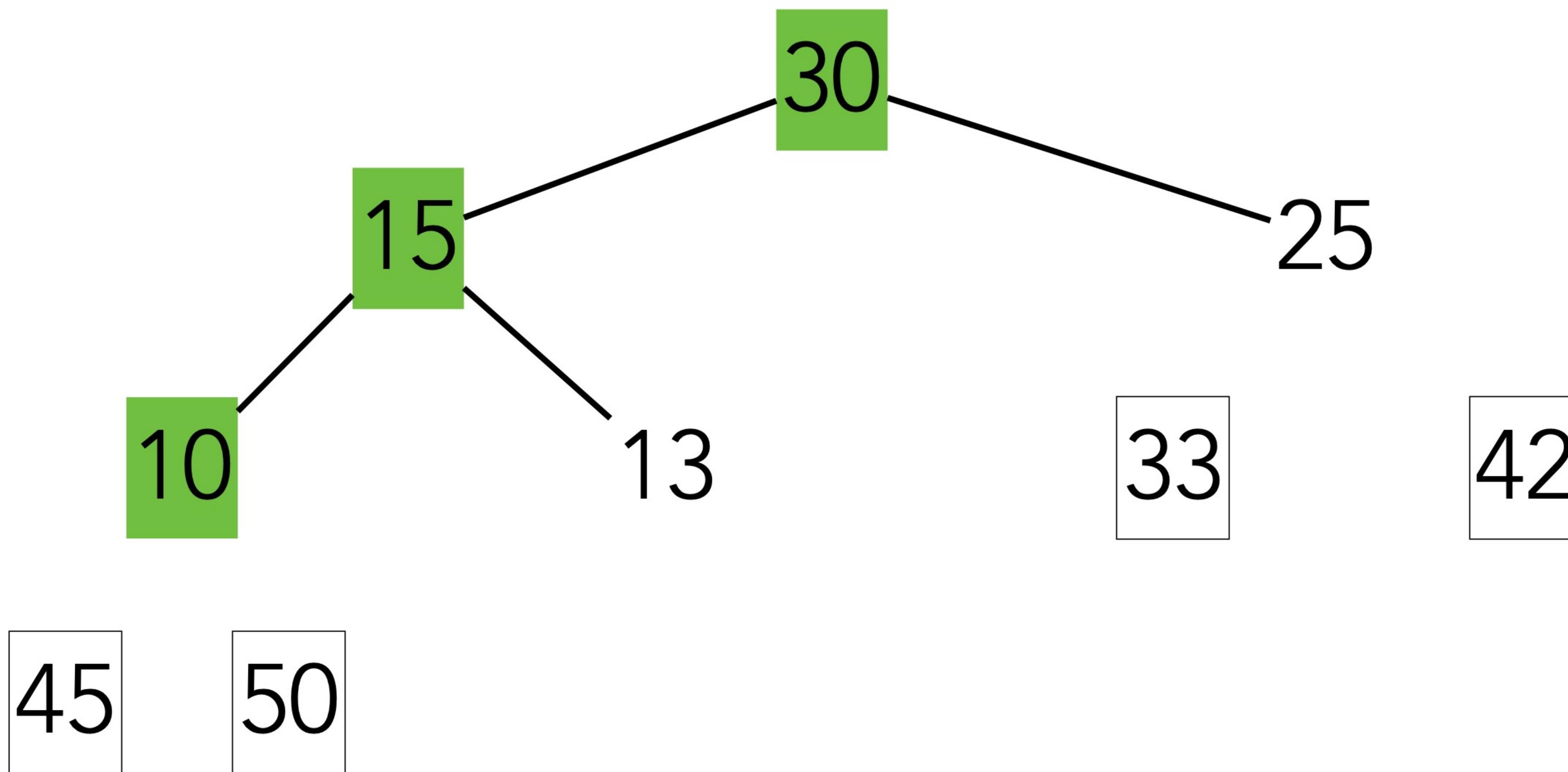
45

50

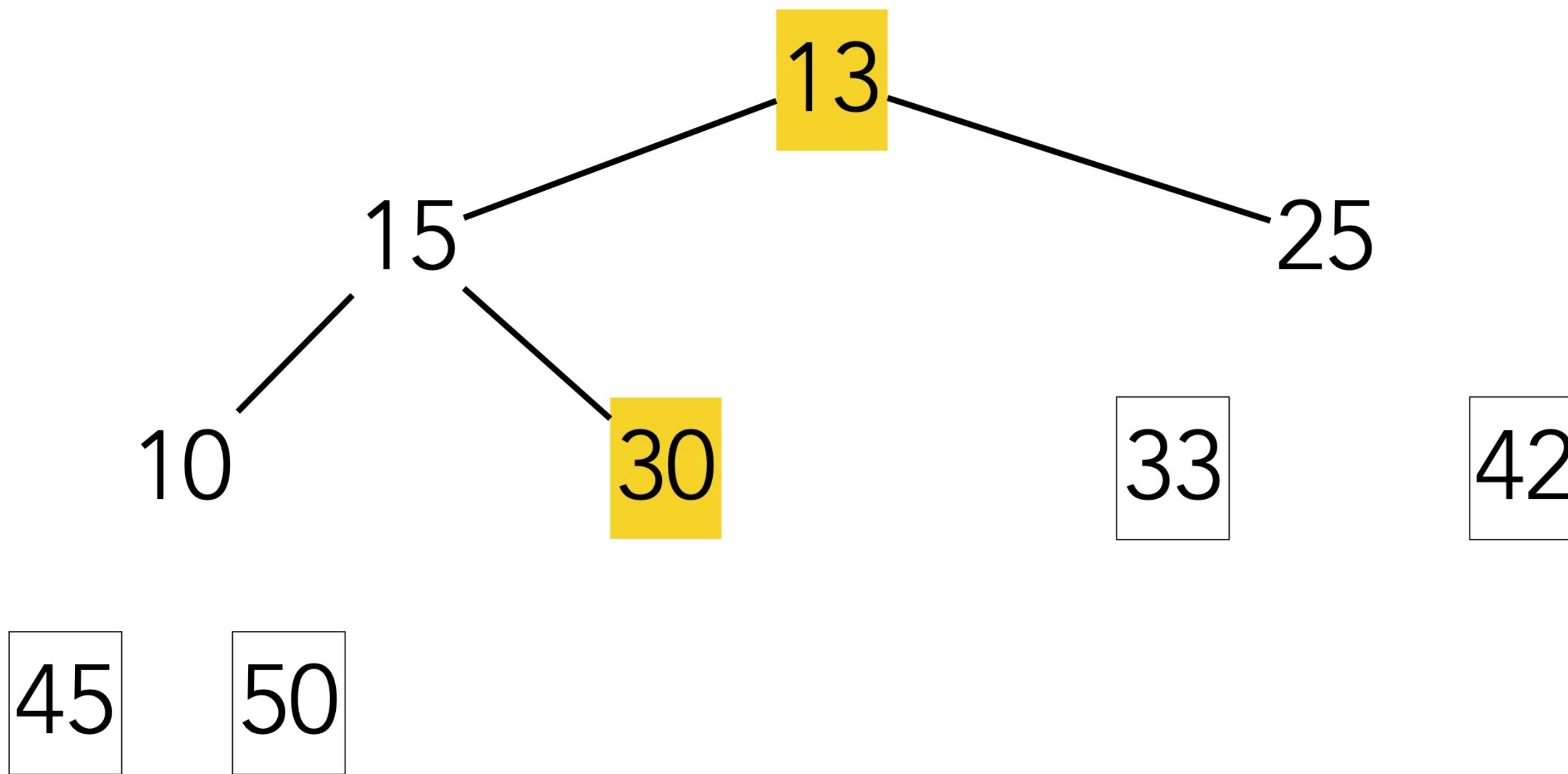
heapSort



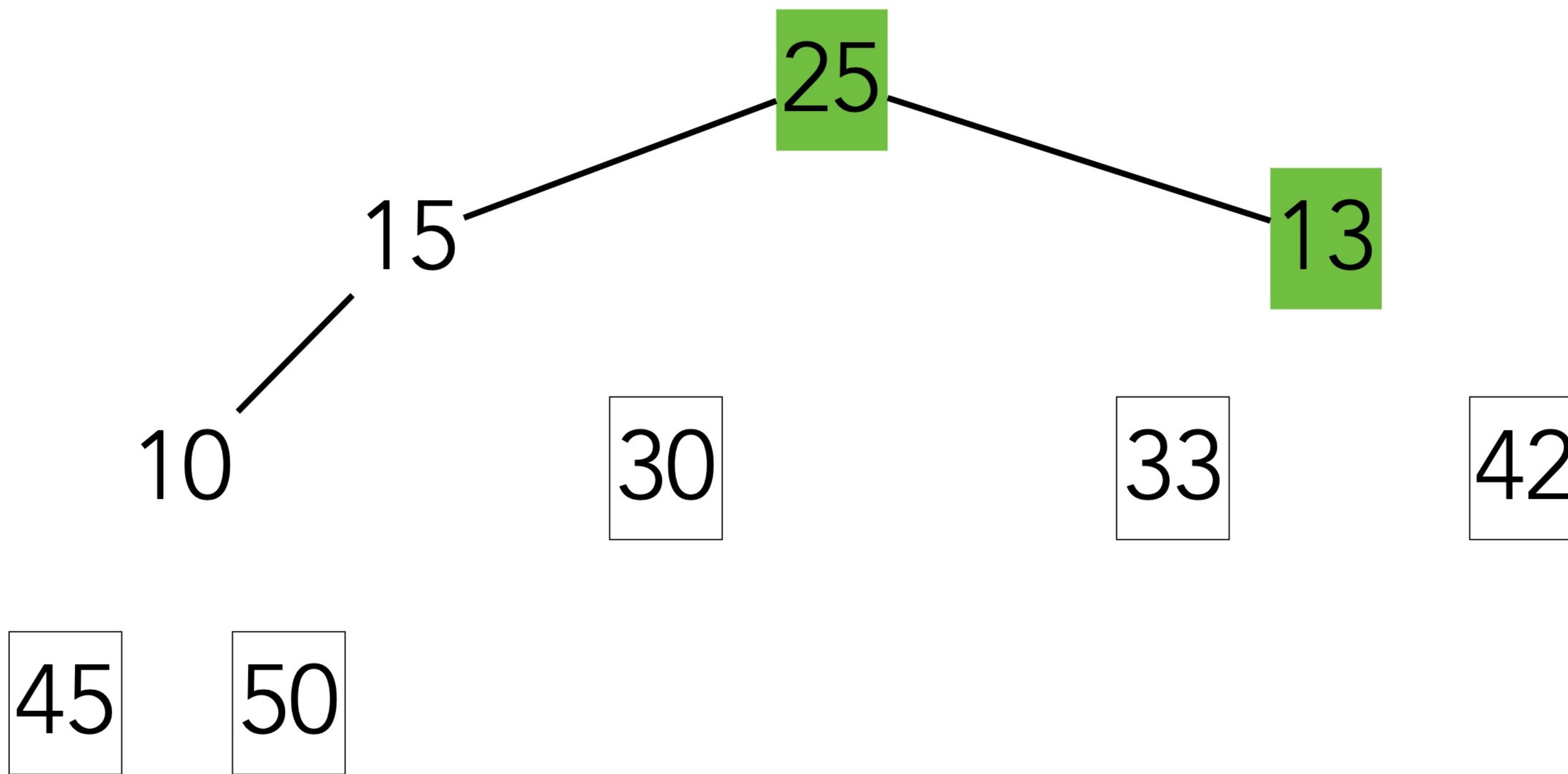
heapSort



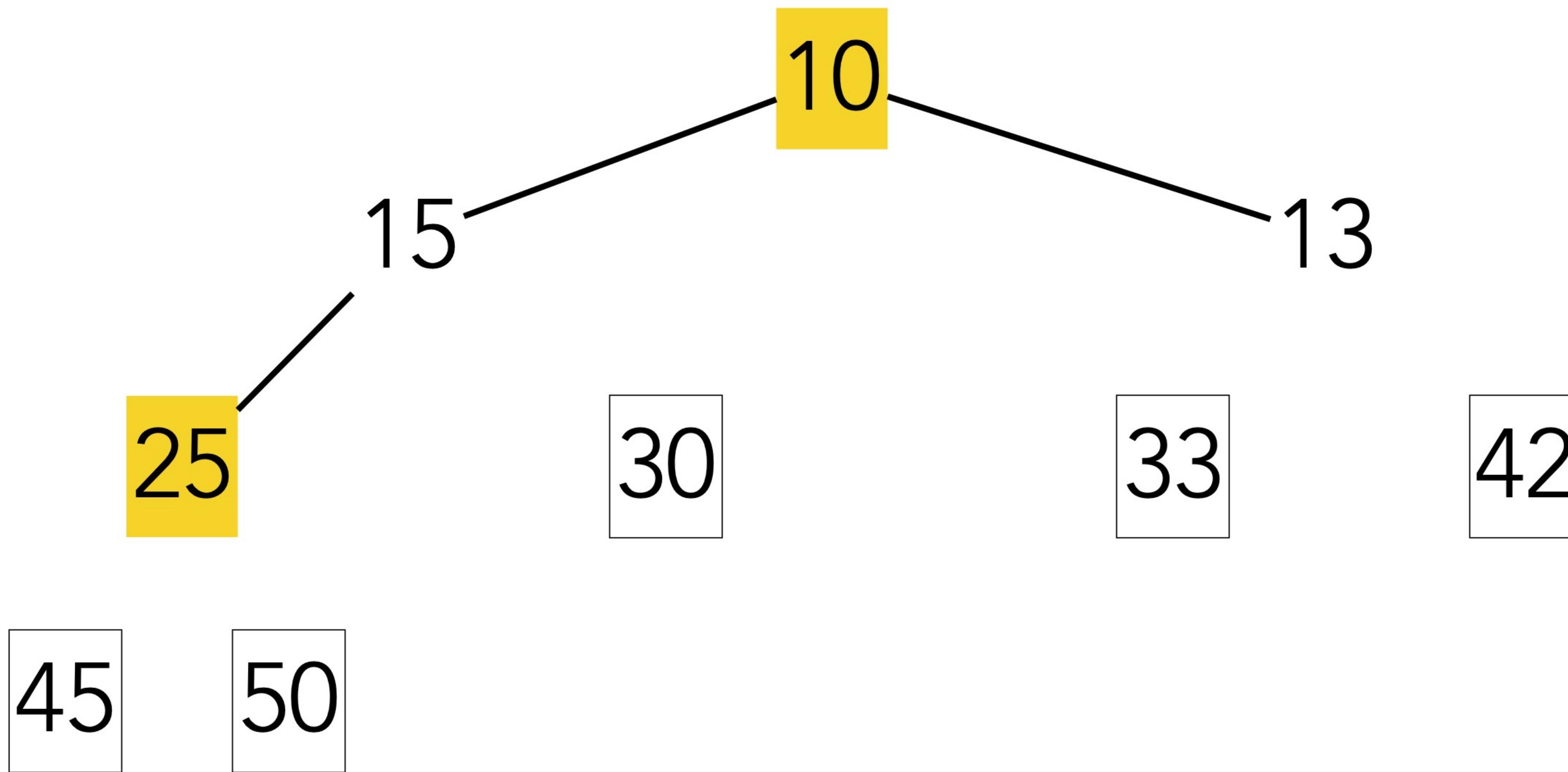
heapSort



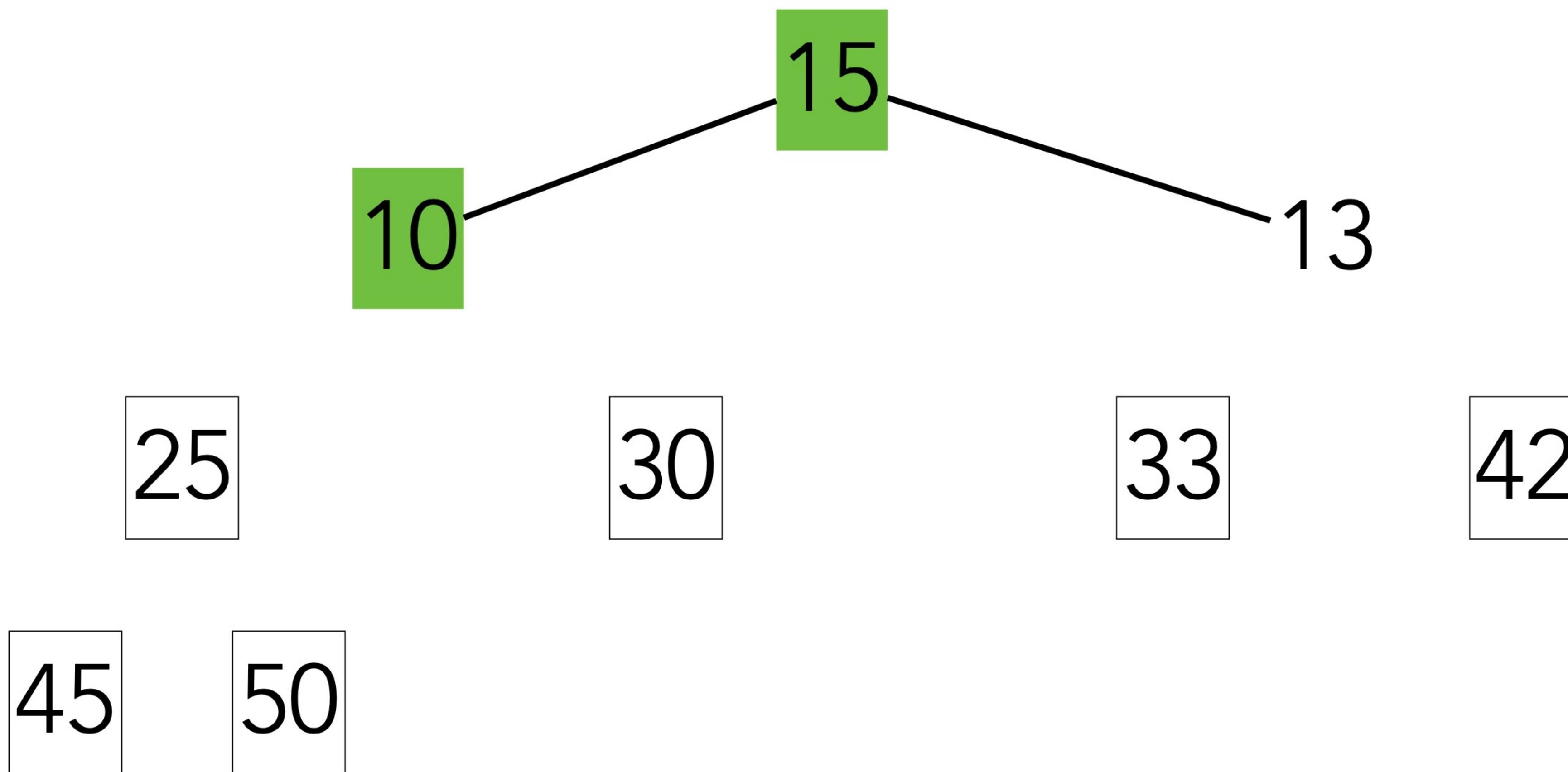
heapSort



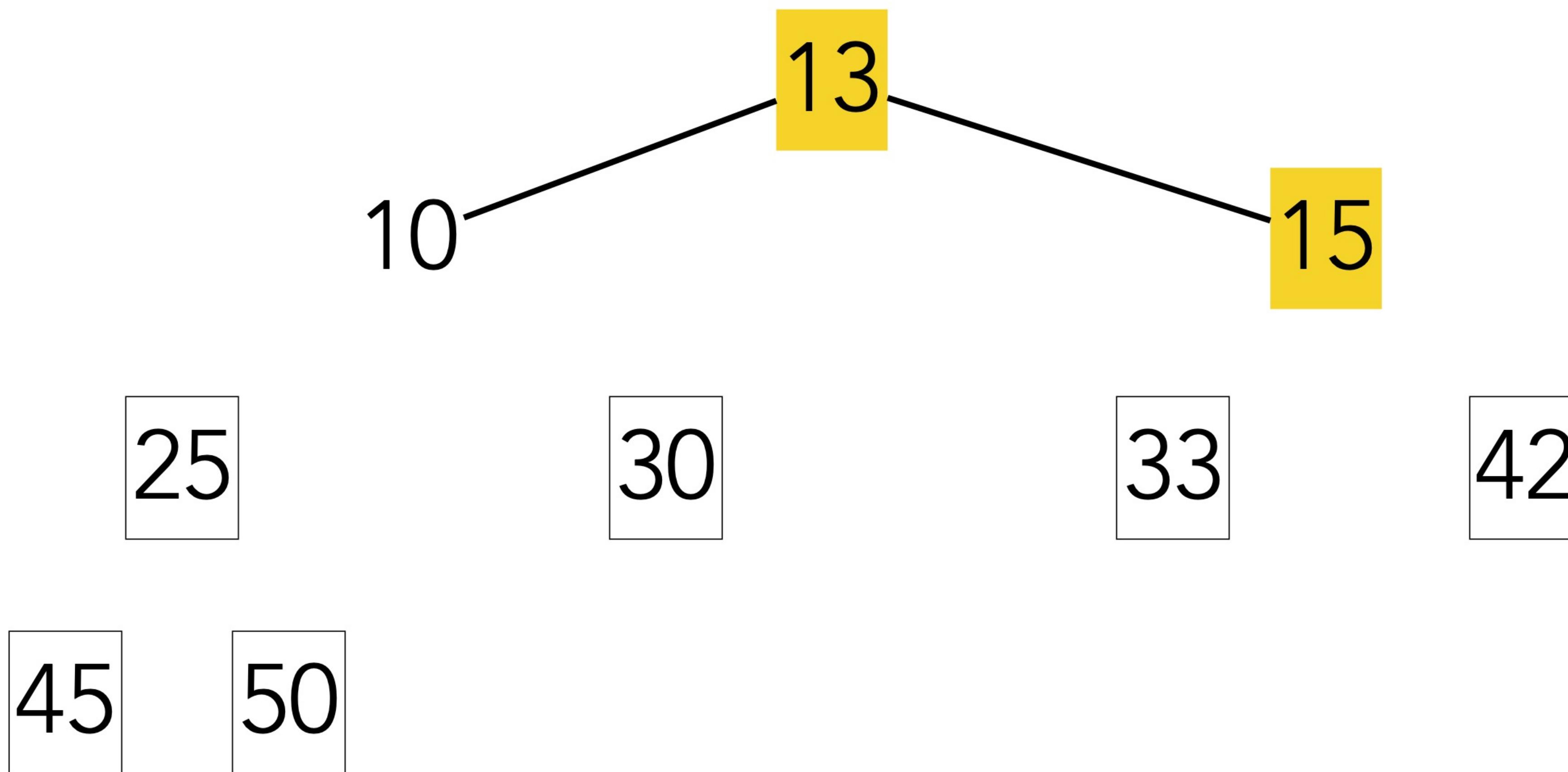
heapSort



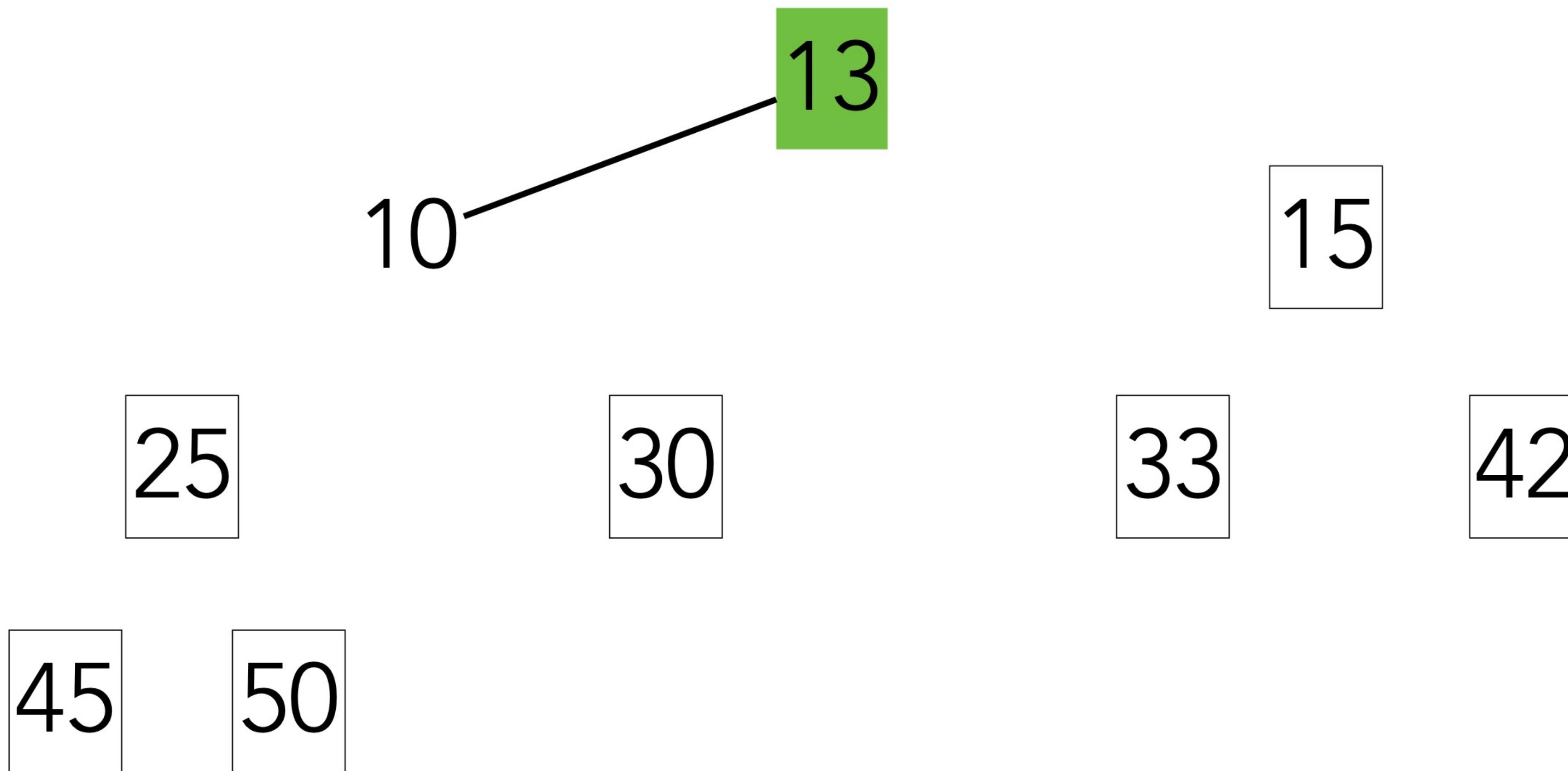
heapSort



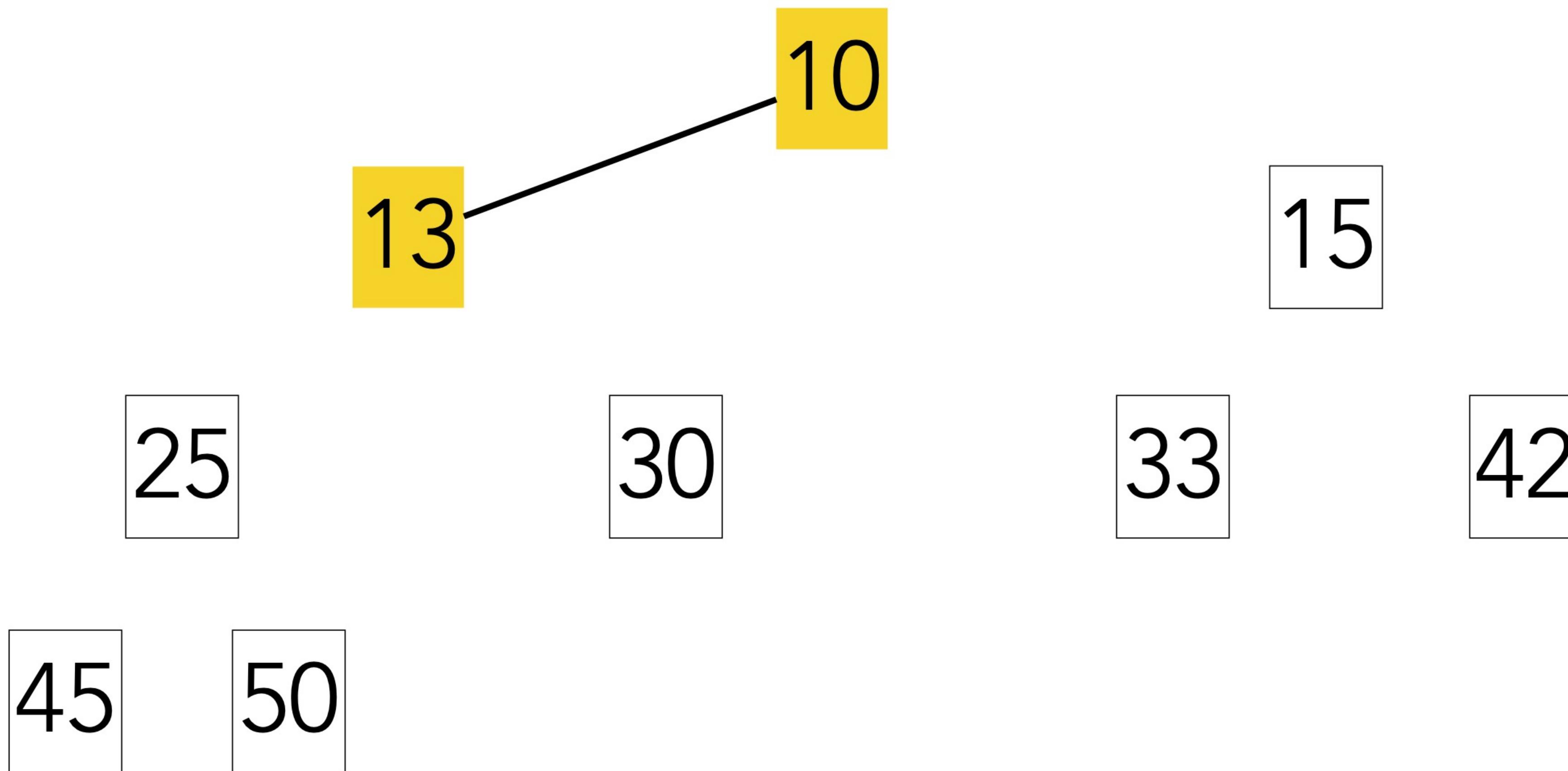
heapSort



heapSort



heapSort



heapSort

10

13

15

25

30

33

42

45

50

heapSort

10

13

15

25

30

33

42

45

50

Analysis?

buildHeap?

step 1

Analysis?

buildHeap?

step 1

$\Theta(n)$

Analysis?

buildHeap?

$\Theta(n)$

step 1

swap and downHeap?

steps 2-5

Analysis?

buildHeap?

$\Theta(n)$

step 1

swap and downHeap?

$\Theta(n \log n)$

steps 2-5