Our Parameterized BinarySearchTree<E>

- No duplicate elements allowed
- Good for representing sets

- Inserting elements is fast: O(log *n*)
- Finding smallest / largest element is fast: O(log *n*)
- Testing for membership is fast: O(log *n*)
- Removing elements is fast: O(log *n*)

TreeSet<E>

- Java's version of BinarySearchTree<E>
- No duplicate elements allowed
- Methods:
  - **size()**
  - **add(E *element*)**
  - **contains(E *element*)**
  - **first()** returns the smallest element
  - **last()** returns the largest element
- Elements are iterated in sorted order
- TreeSetDemo

Priority Queues and Heaps

- A **priority queue** maintains a collection of elements ordered by their **priority values**
- **Smaller** values correspond to **higher** priority
- Elements must be **comparable**
- Operations on priority queues:
  - **insert** a new element into the queue
  - **get the first** element (the one with highest priority)
  - **remove** the highest priority element
- A **heap** can be used to implement a priority queue

Example: a Priority Queue of Integers

Insert 12

Example: a Priority Queue of Integers

12

Example: a Priority Queue of Integers

12

Insert 15

Example: a Priority Queue of Integers

12 15

Example: a Priority Queue of Integers

12 15

Insert 3

Example: a Priority Queue of Integers

3 12 15

Example: a Priority Queue of Integers

3 12 15

Insert 7

Example: a Priority Queue of Integers

3 7 12 15

Example: a Priority Queue of Integers

3 7 12 15

Remove Smallest

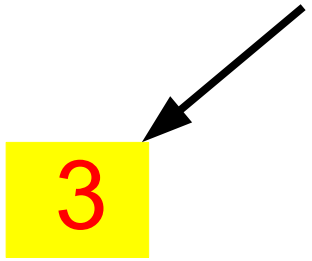
Example: a Priority Queue of Integers

3 7 12 15

Remove Smallest

Example: a Priority Queue of Integers

7 12 15



Example: a Priority Queue of Integers

7 12 15

Insert 9

Example: a Priority Queue of Integers

7 9 12 15

Example: a Priority Queue of Integers

7 9 12 15

Insert 20

Example: a Priority Queue of Integers

7 9 12 15 20

Example: a Priority Queue of Integers

7 9 12 15 20

Remove Smallest

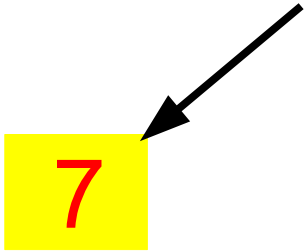
Example: a Priority Queue of Integers

7 9 12 15 20

Remove Smallest

Example: a Priority Queue of Integers

9 12 15 20



Example: a Priority Queue of Integers

9 12 15 20

Remove Smallest

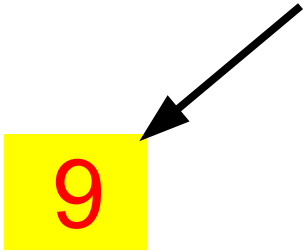
Example: a Priority Queue of Integers

9 12 15 20

Remove Smallest

Example: a Priority Queue of Integers

12 15 20



Example: a Priority Queue of Integers

12 15 20

Insert 17

Example: a Priority Queue of Integers

12 15 17 20

Example: a Priority Queue of Integers

12 15 17 20

Remove Smallest

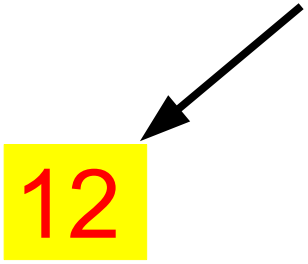
Example: a Priority Queue of Integers

12 15 17 20

Remove Smallest

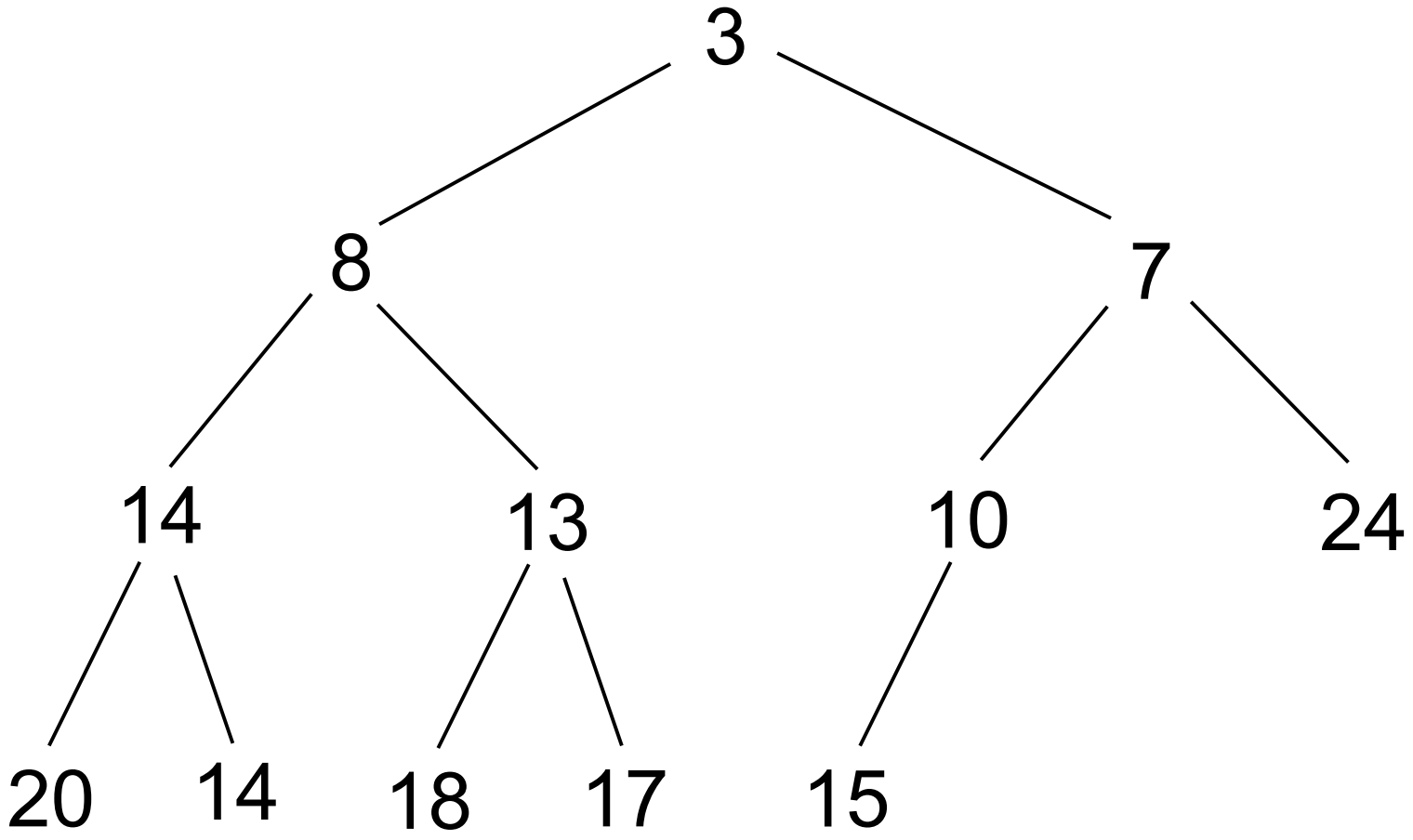
Example: a Priority Queue of Integers

15 17 20



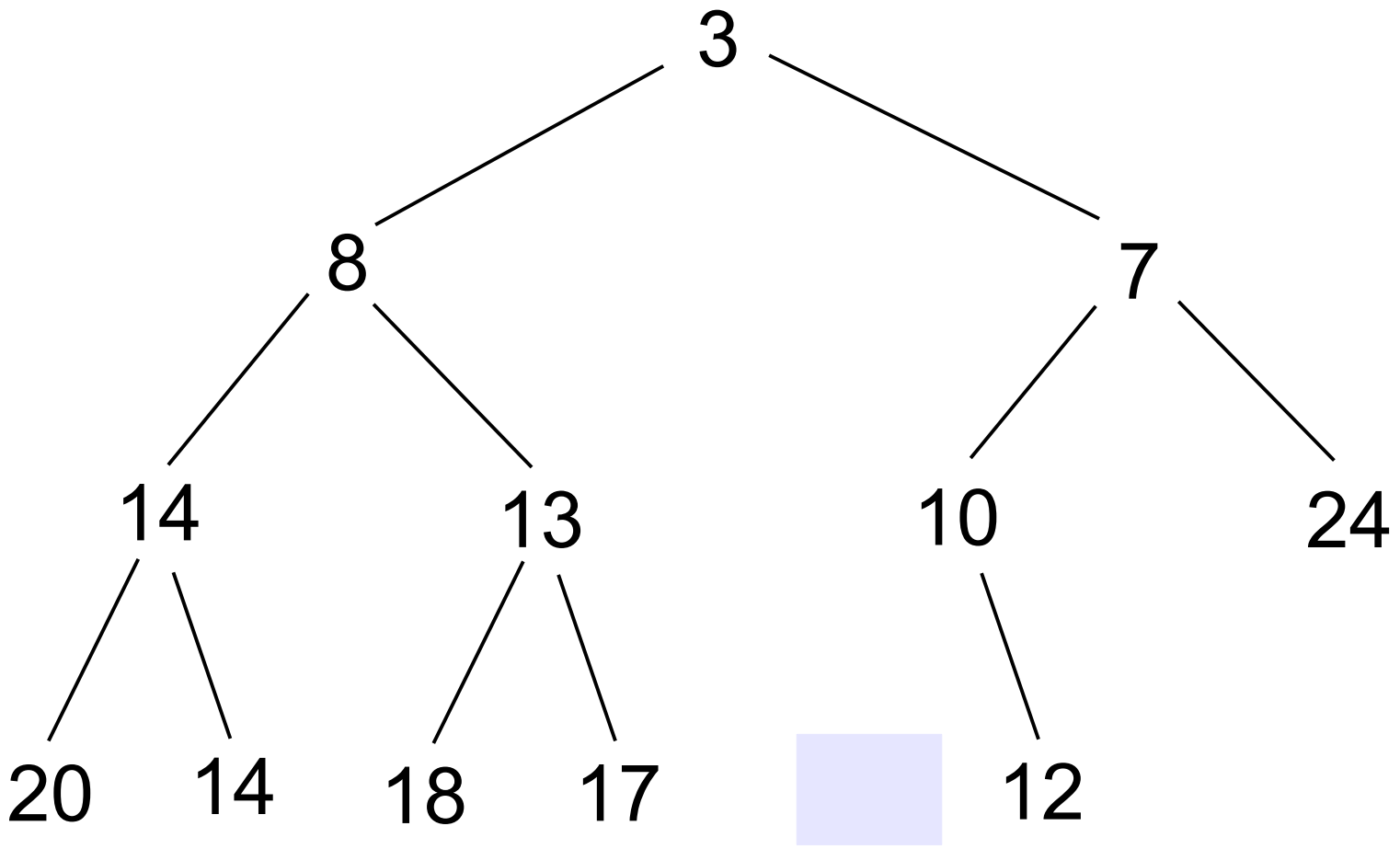
… and so on

Heaps



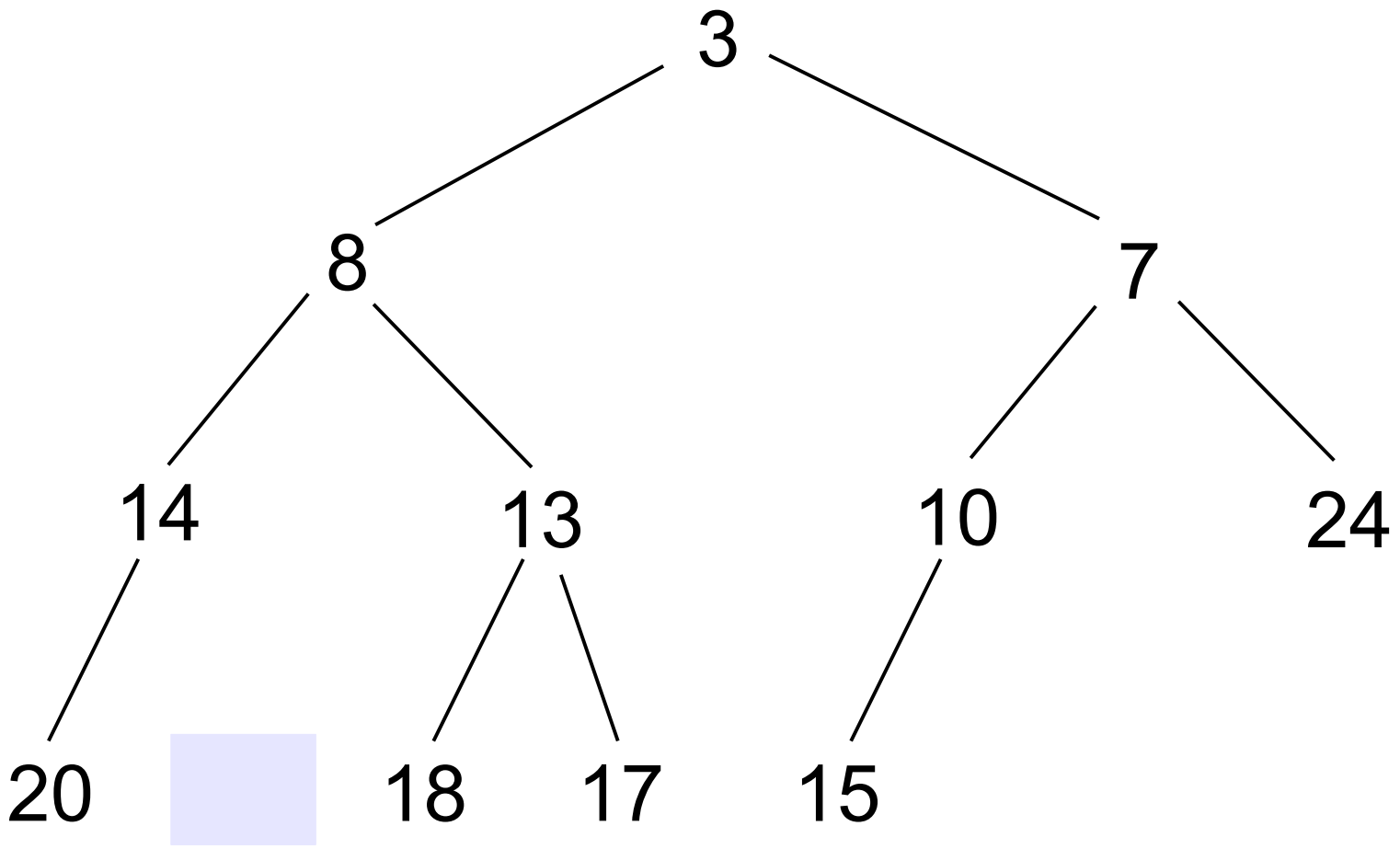
A heap is a **complete** binary tree

An Invalid Heap



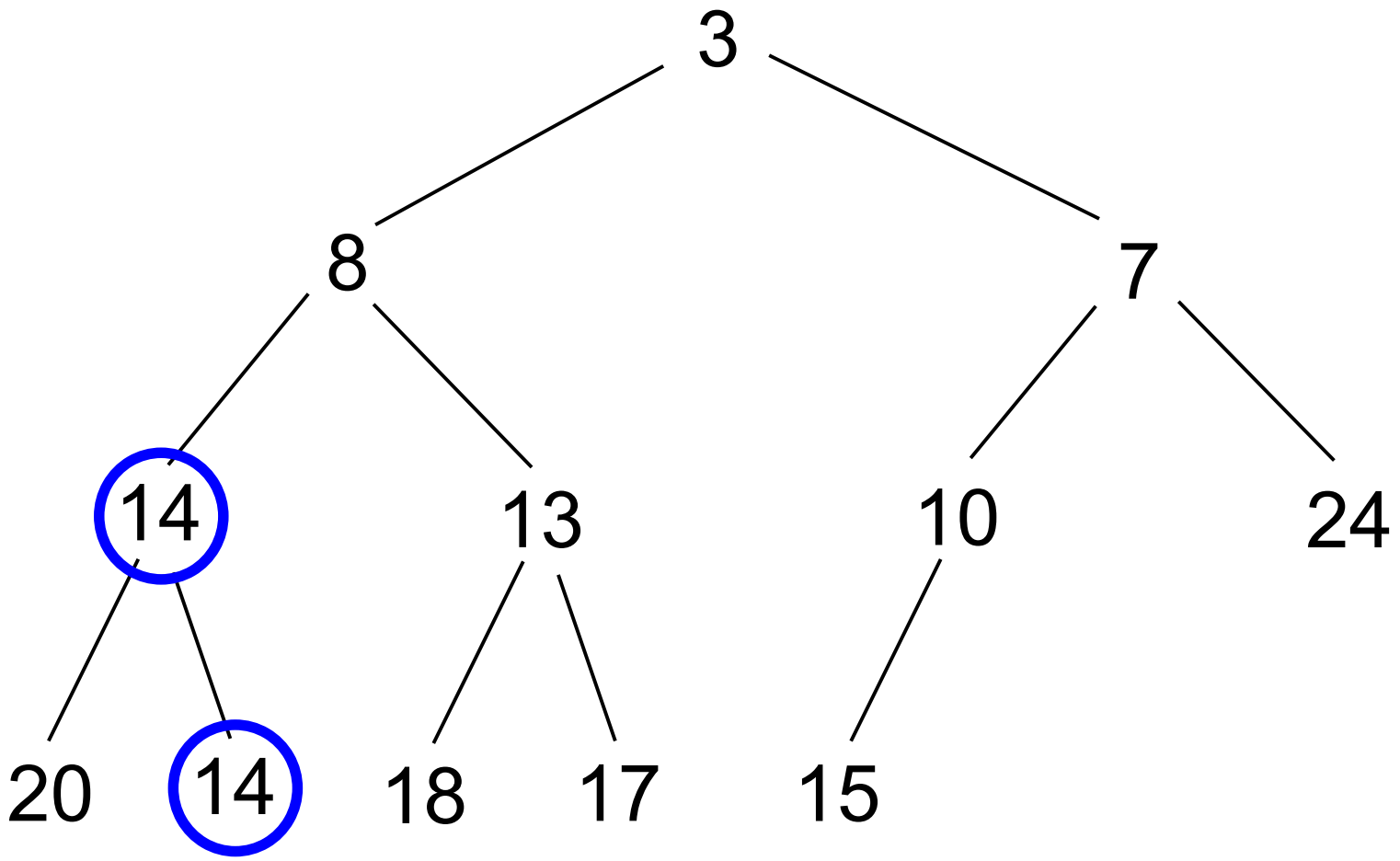
This tree is **not complete**

An Invalid Heap



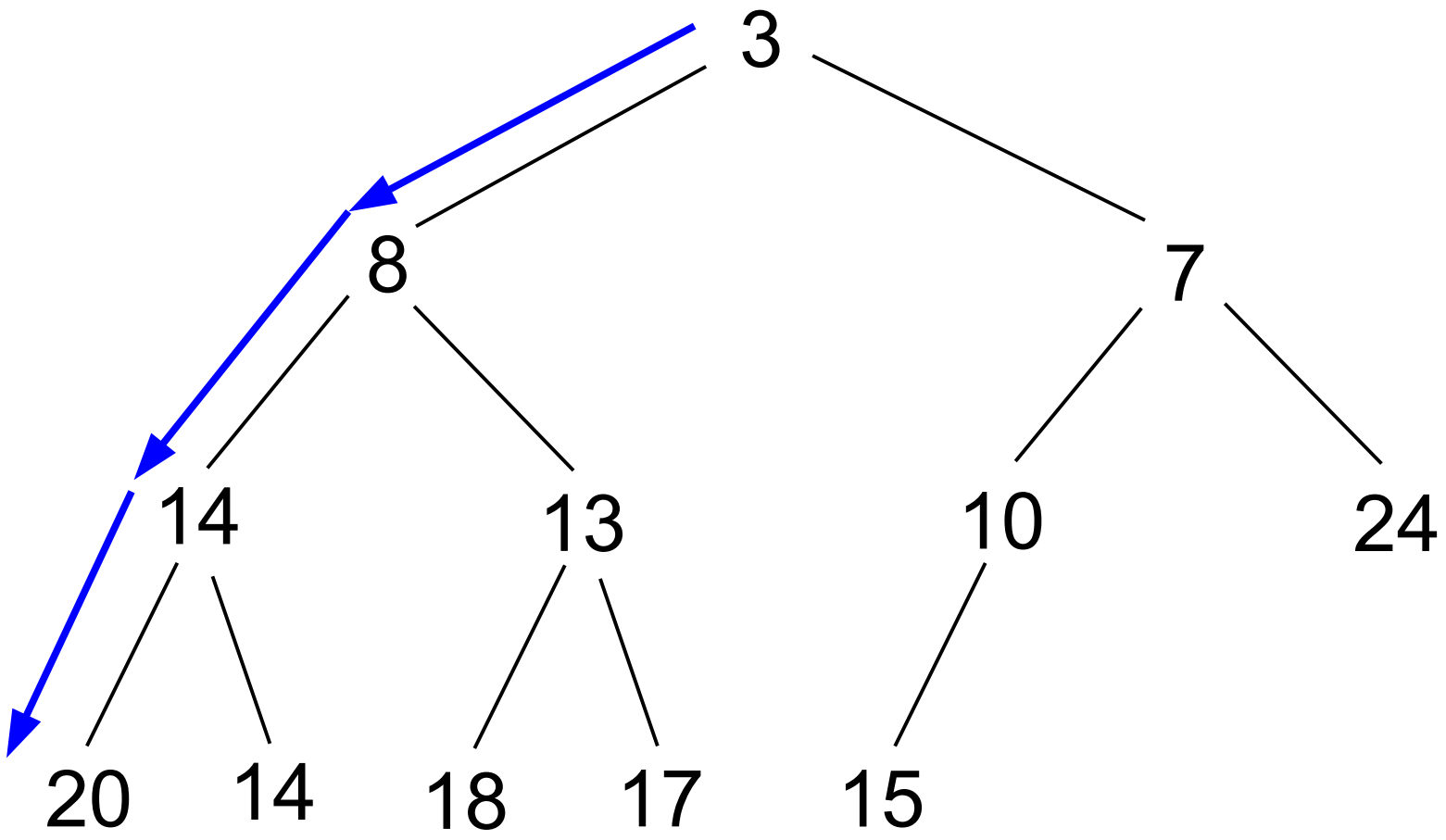
This tree is also **not complete**

Heaps



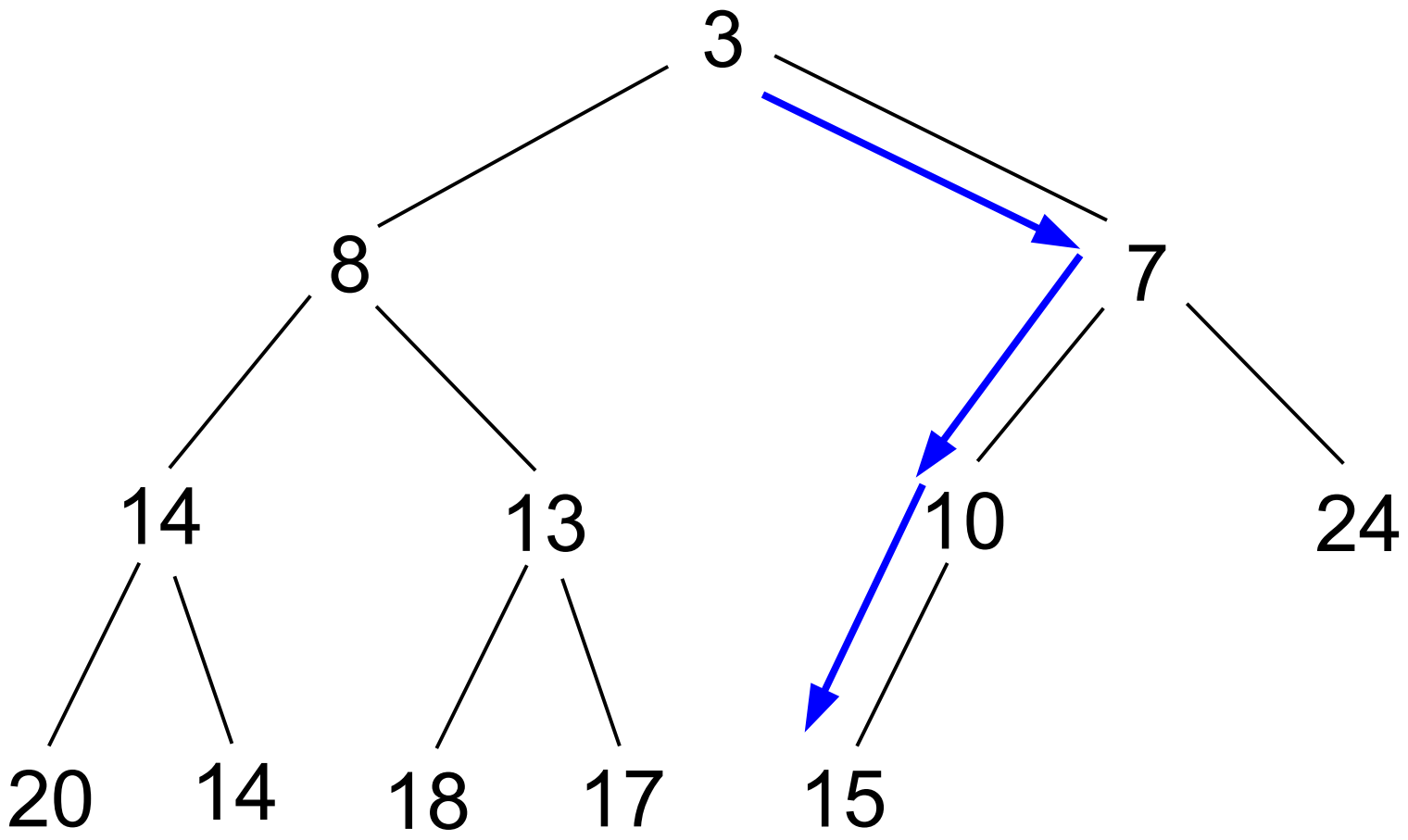
Duplicate elements **are** allowed

Heaps



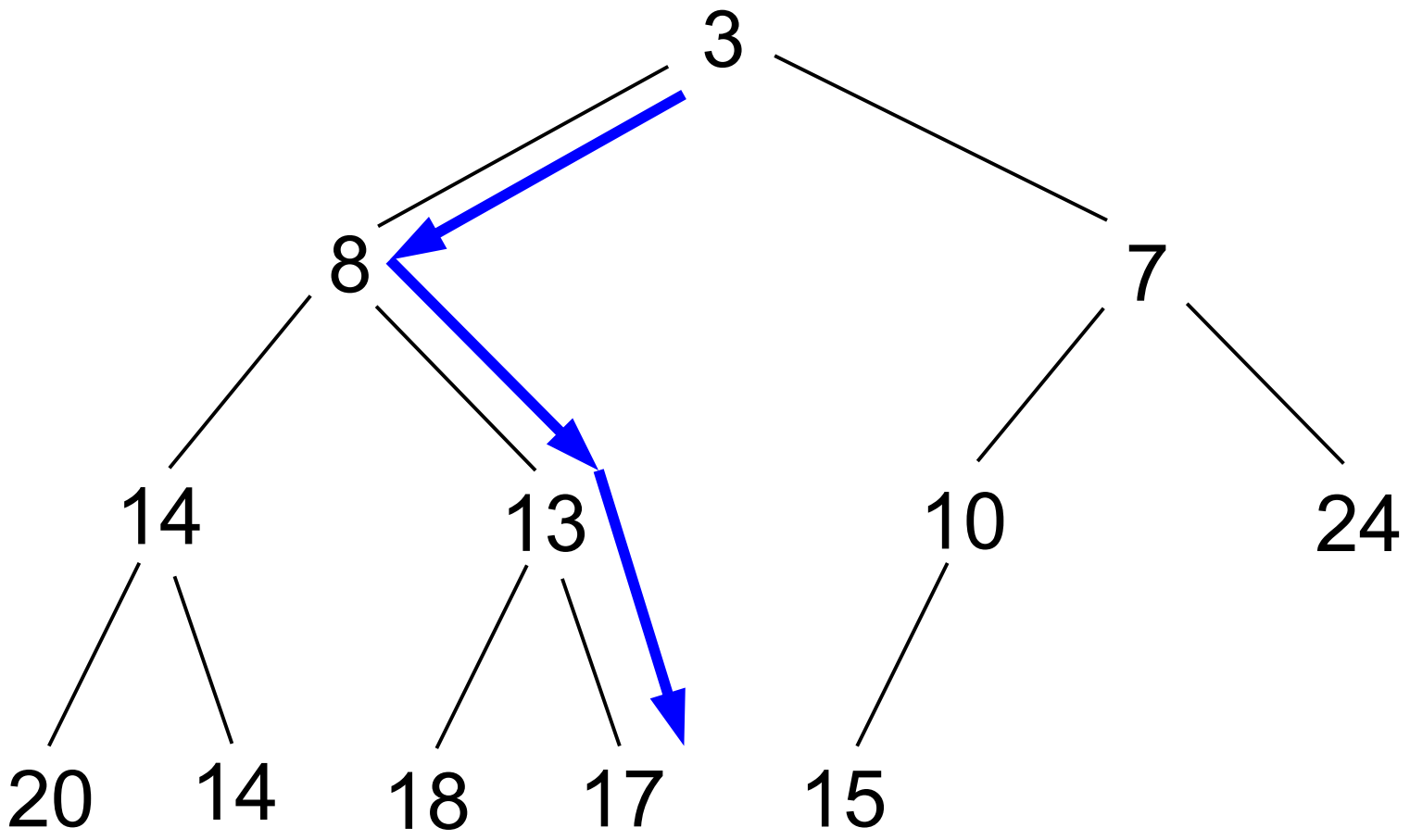
Heap property: every path from the root is non-decreasing

Heaps



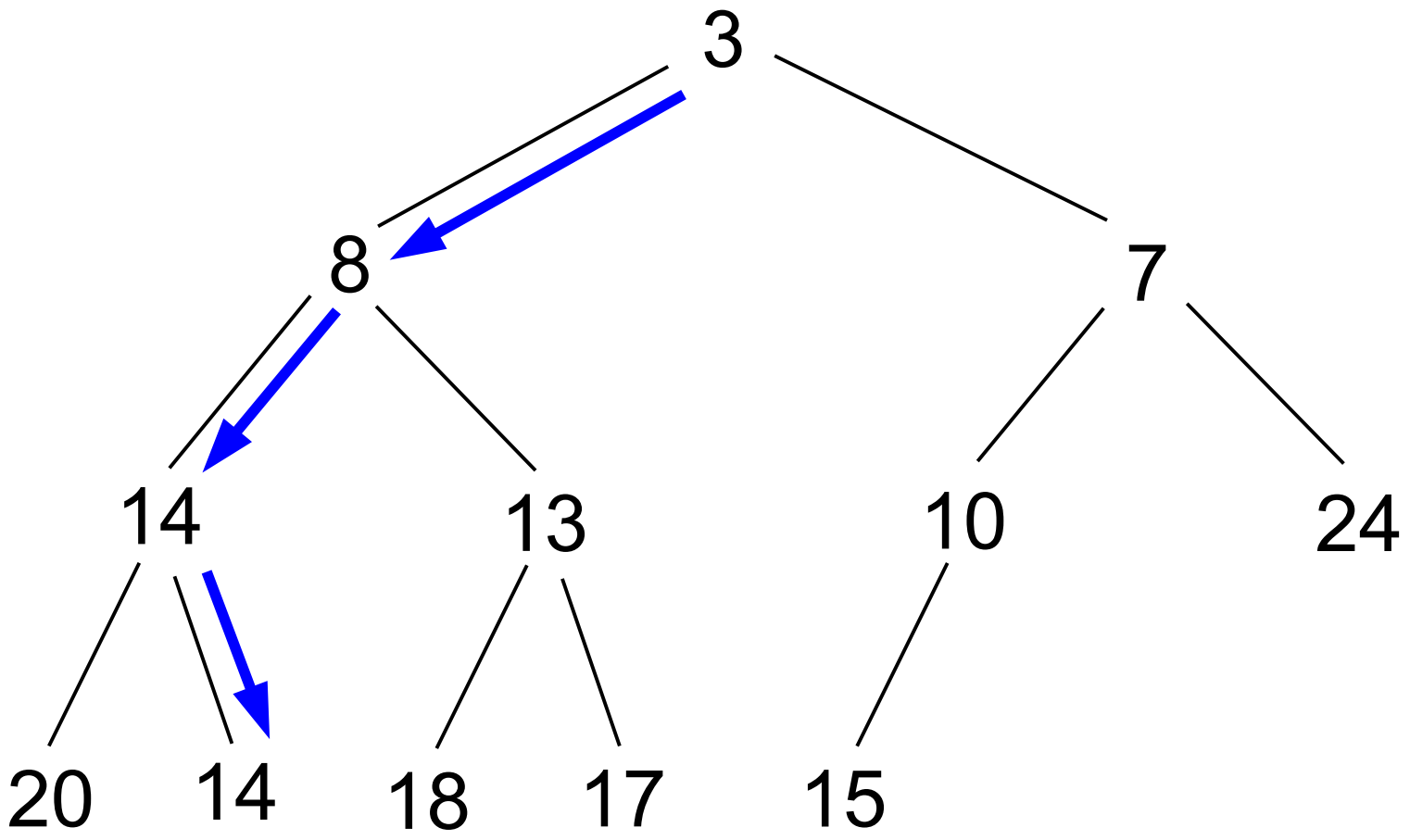
Heap property: every path from the root is non-decreasing

Heaps



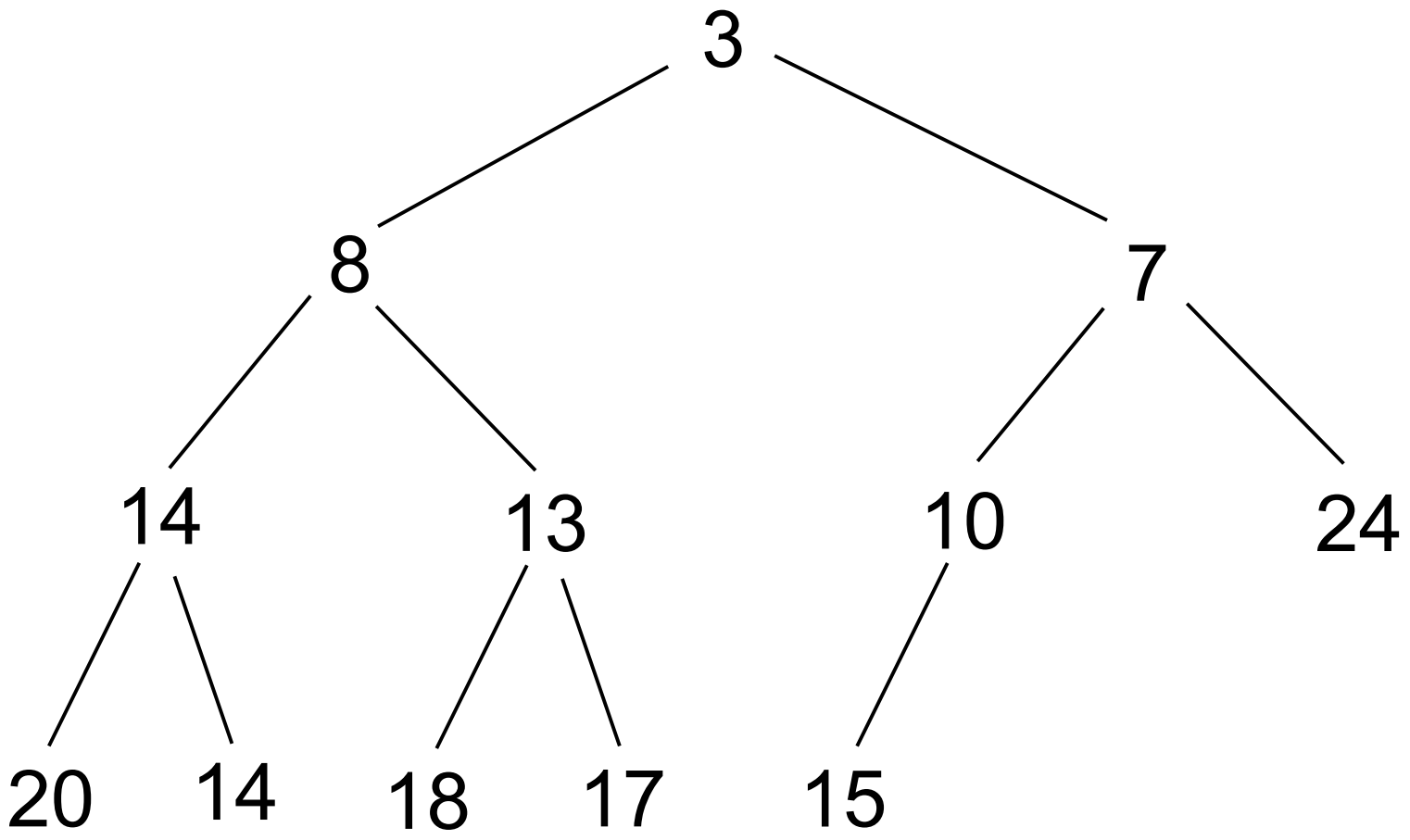
Heap property: every path from the root is non-decreasing

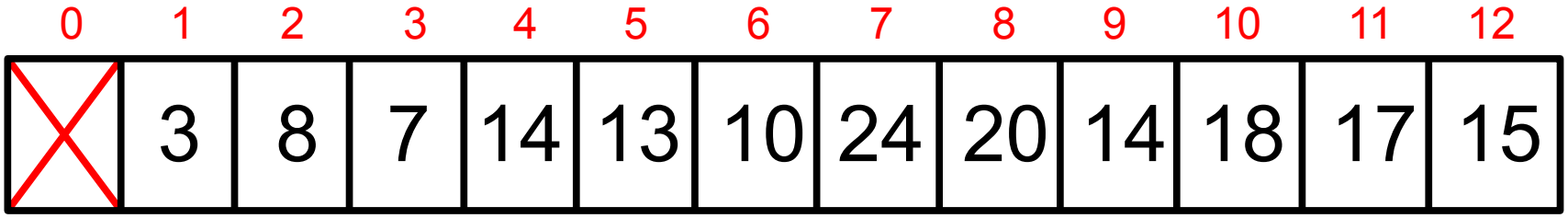
Heaps



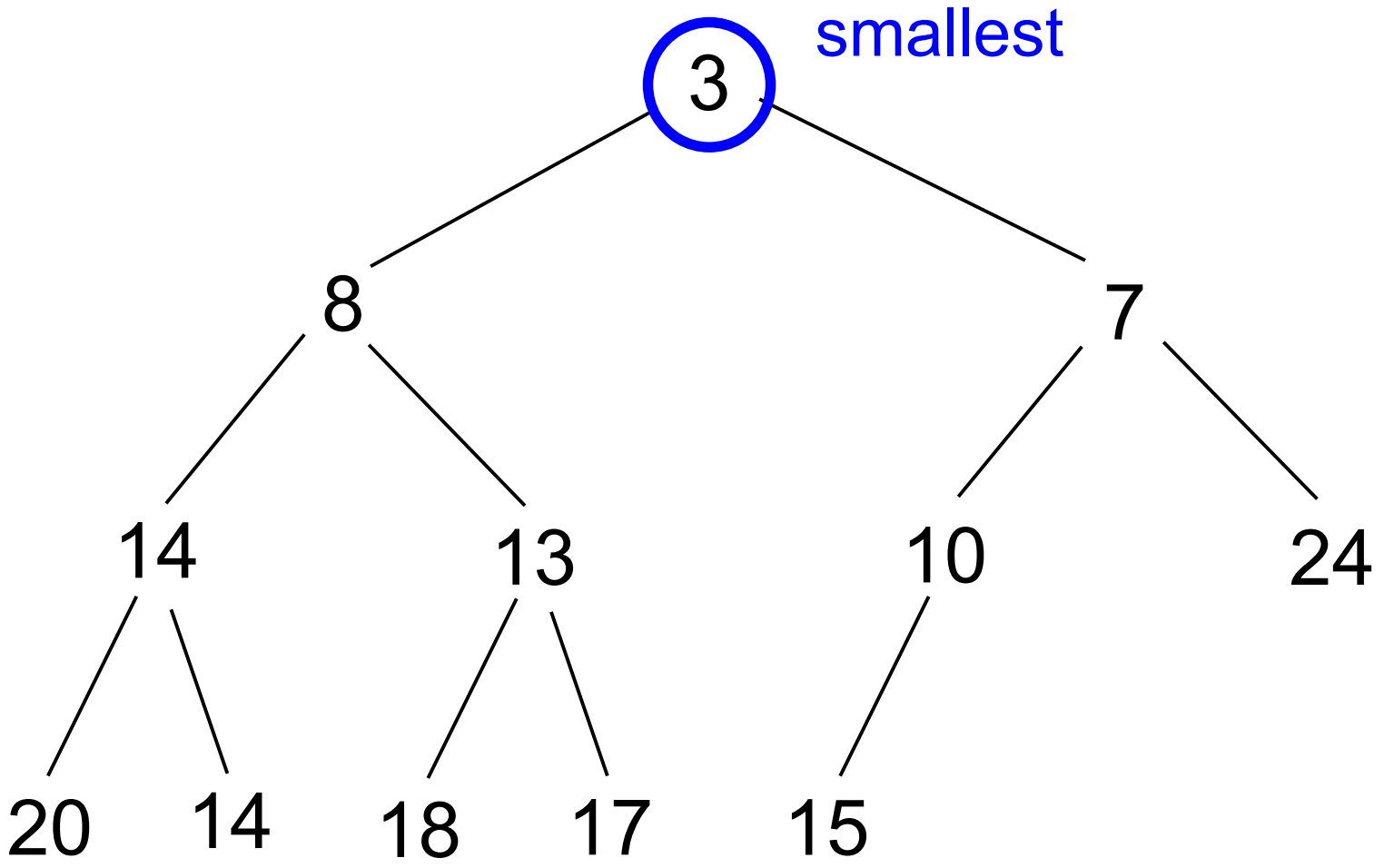
Heap property: every path from the root is non-decreasing

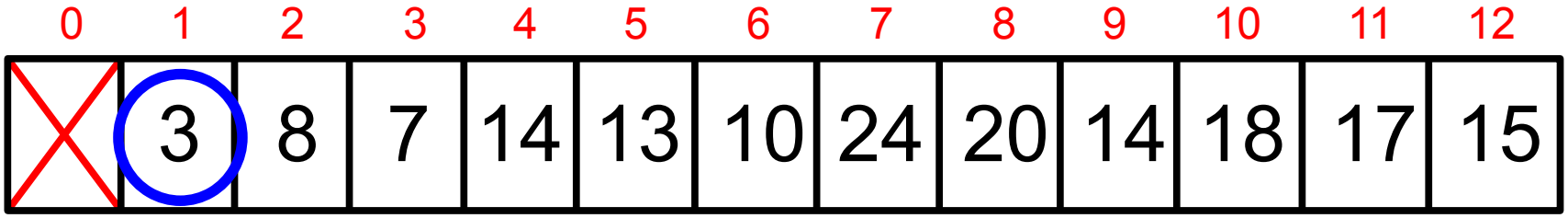
Heaps


3
8
7
14
13
10
24
20
14
18
17
15

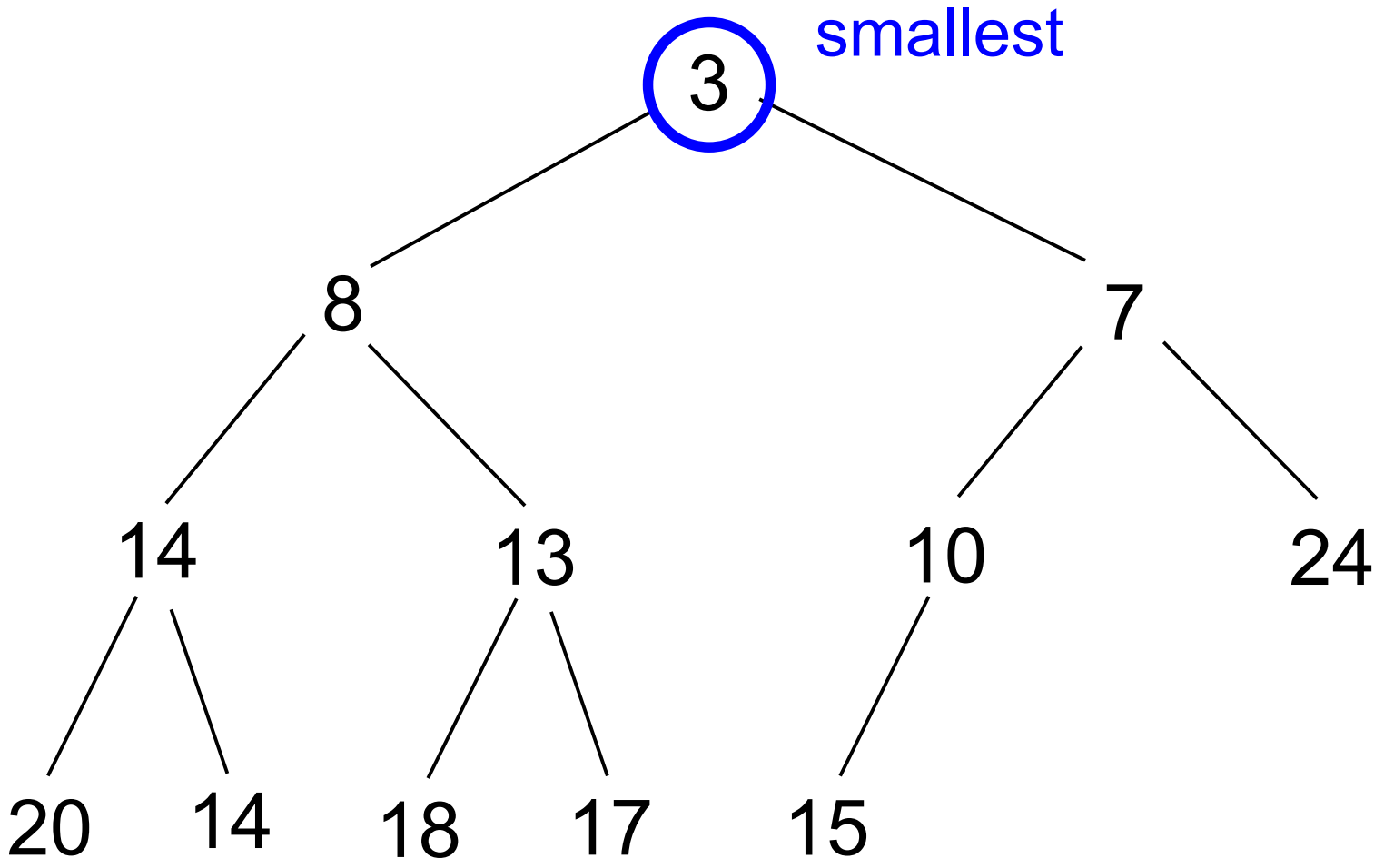

0 1 2 3 4 5 6 7 8 9 10 11 12
3 8 7 14 13 10 24 20 14 18 17 15

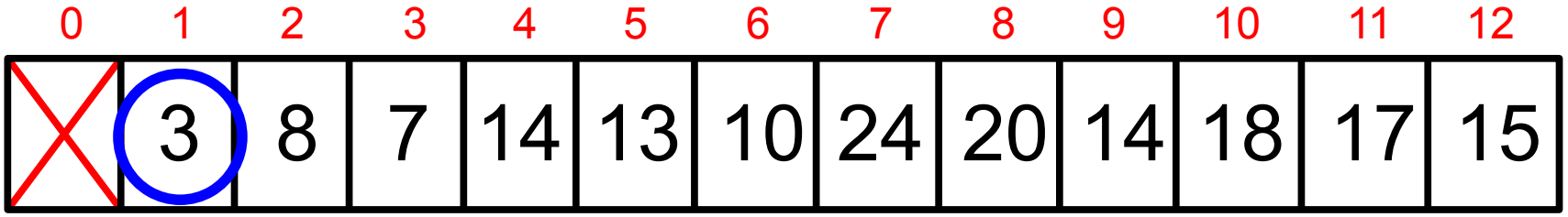
Heaps


smallest
3
8
7
14
13
10
24
20
14
18
17
15

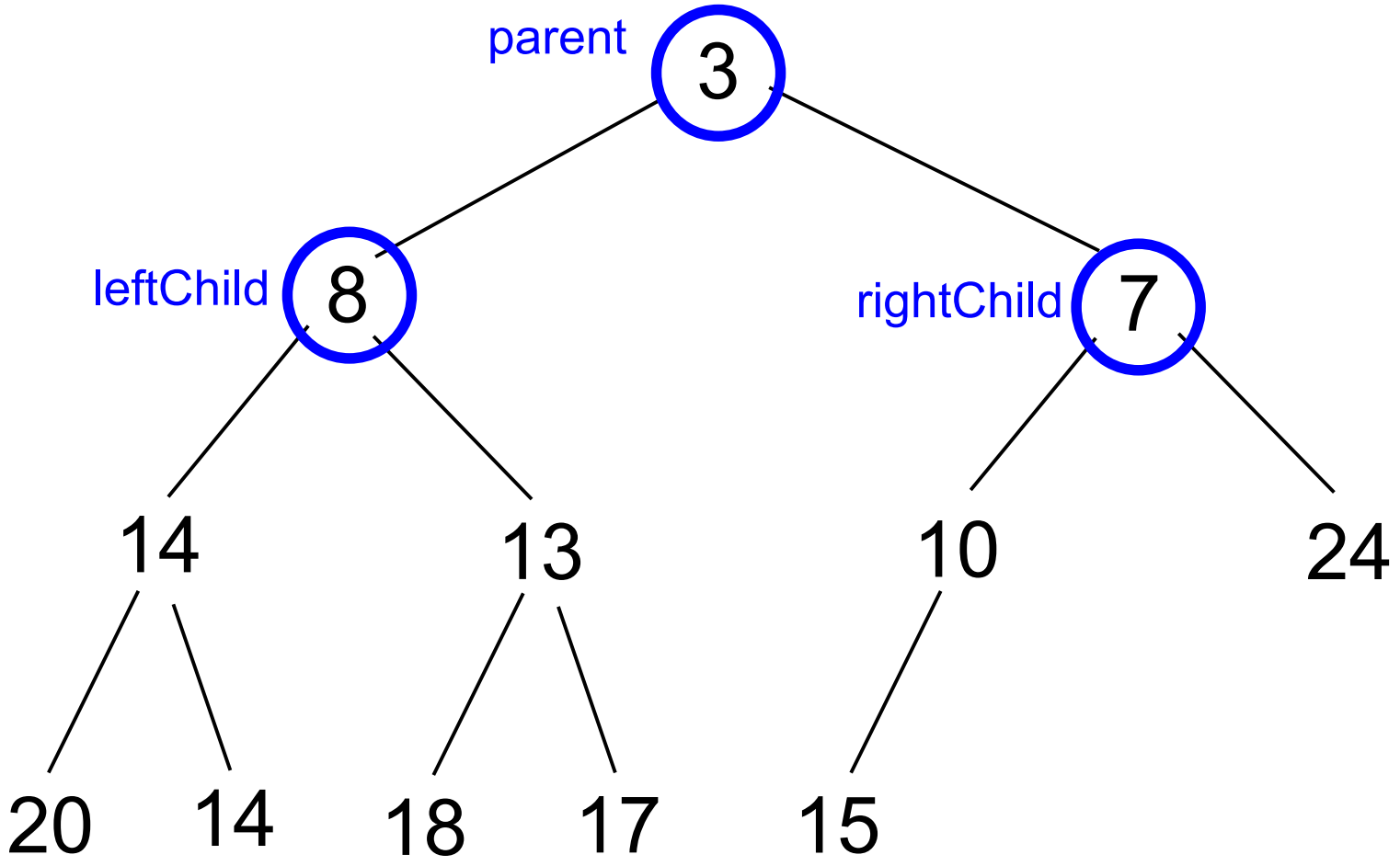

0 1 2 3 4 5 6 7 8 9 10 11 12
3 8 7 14 13 10 24 20 14 18 17 15

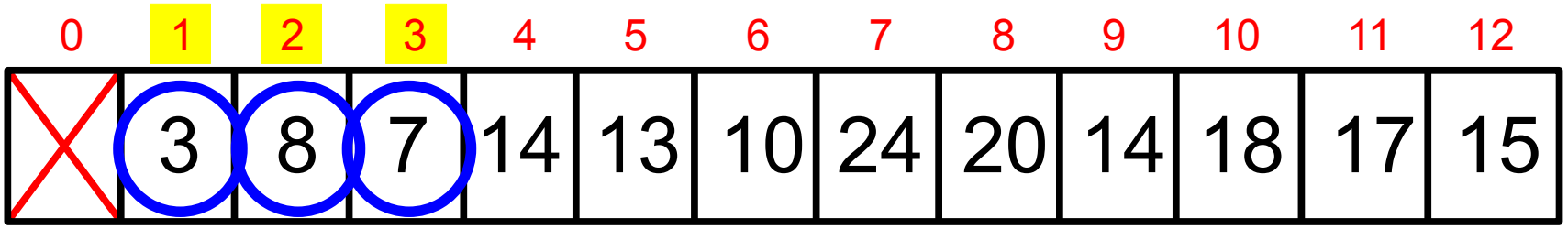
Time Complexity of GetFirst: O(1)



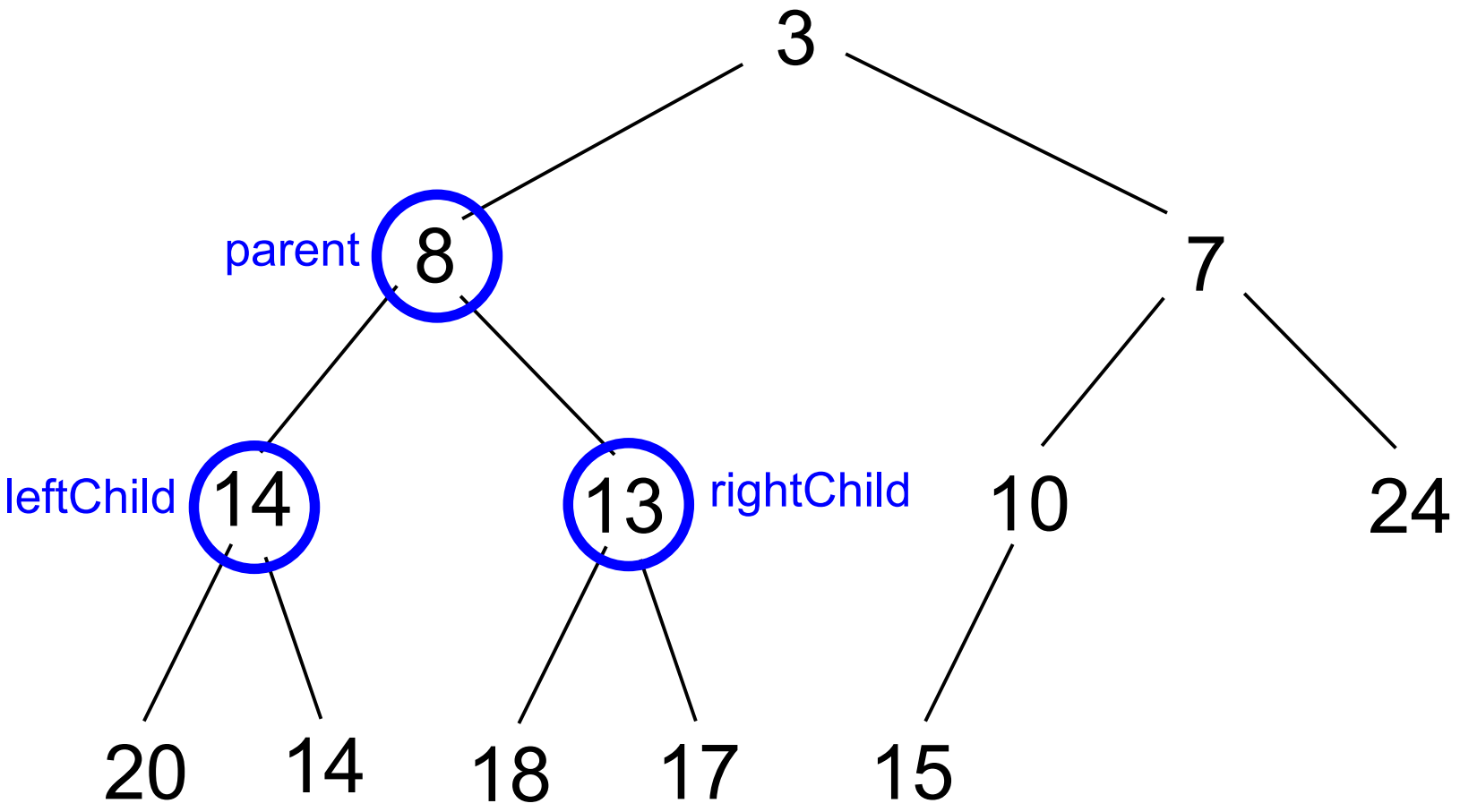


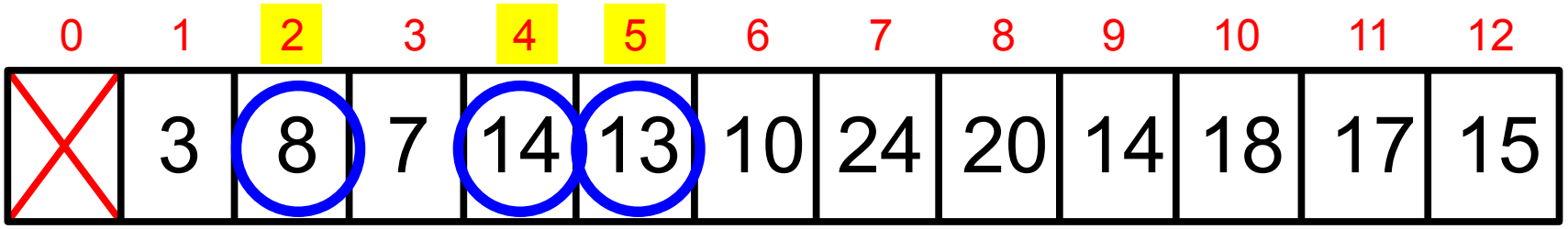
Heaps


parent
3
leftChild
8
rightChild
7
14
13
10
24
20
14
18
17
15

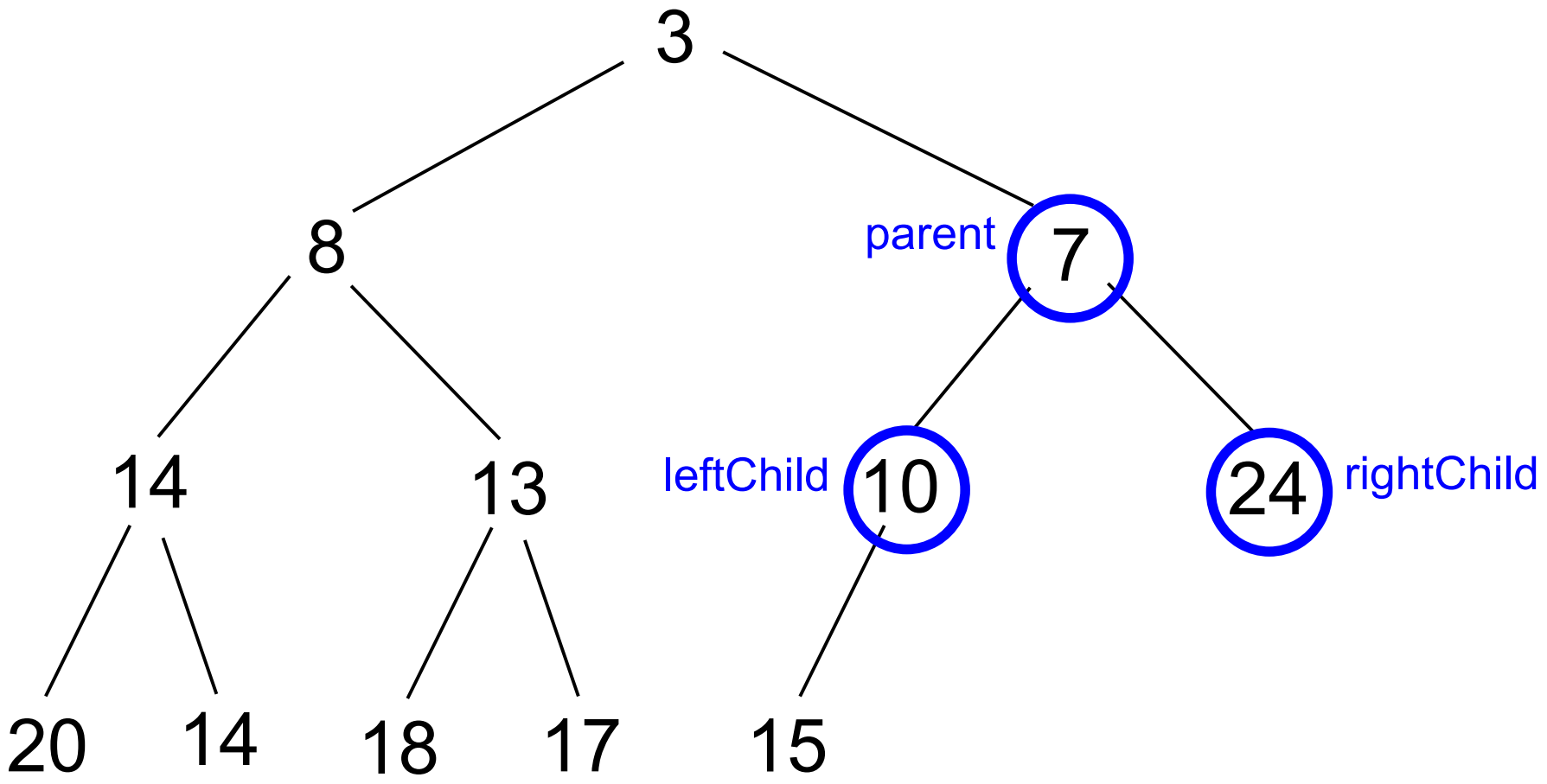

0
1
2
3
4
5
6
7
8
9
10
11
12
3
8
7
14
13
10
24
20
14
18
17
15

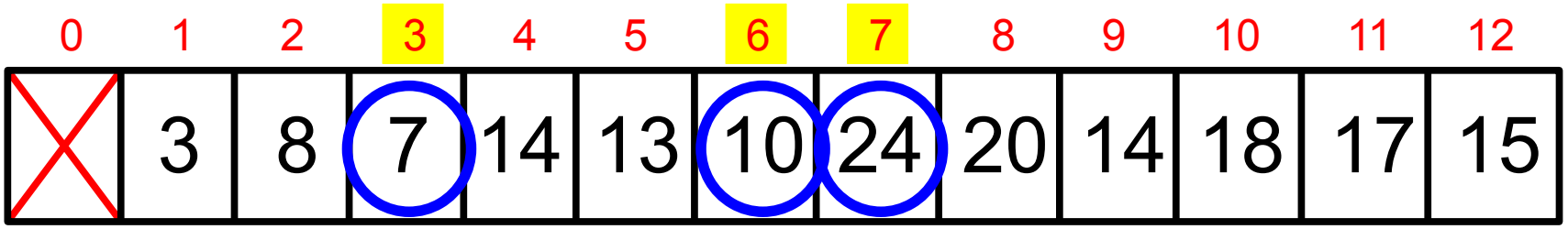
Heaps


3
parent
8
7
leftChild
14
13
rightChild
10
24
20
14
18
17
15

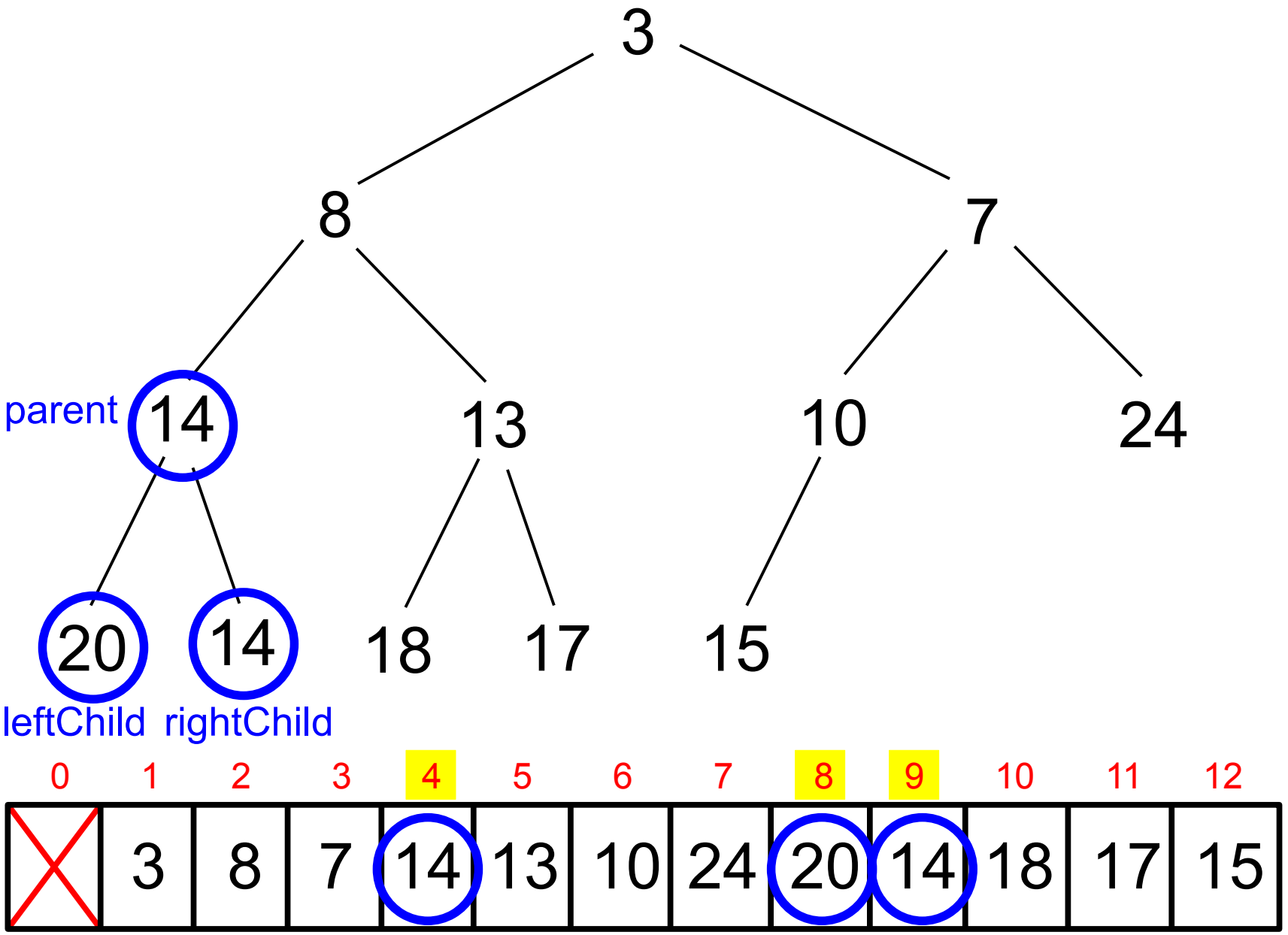

0
1
2
3
4
5
6
7
8
9
10
11
12
3
8
7
14
13
10
24
20
14
18
17
15

Heaps

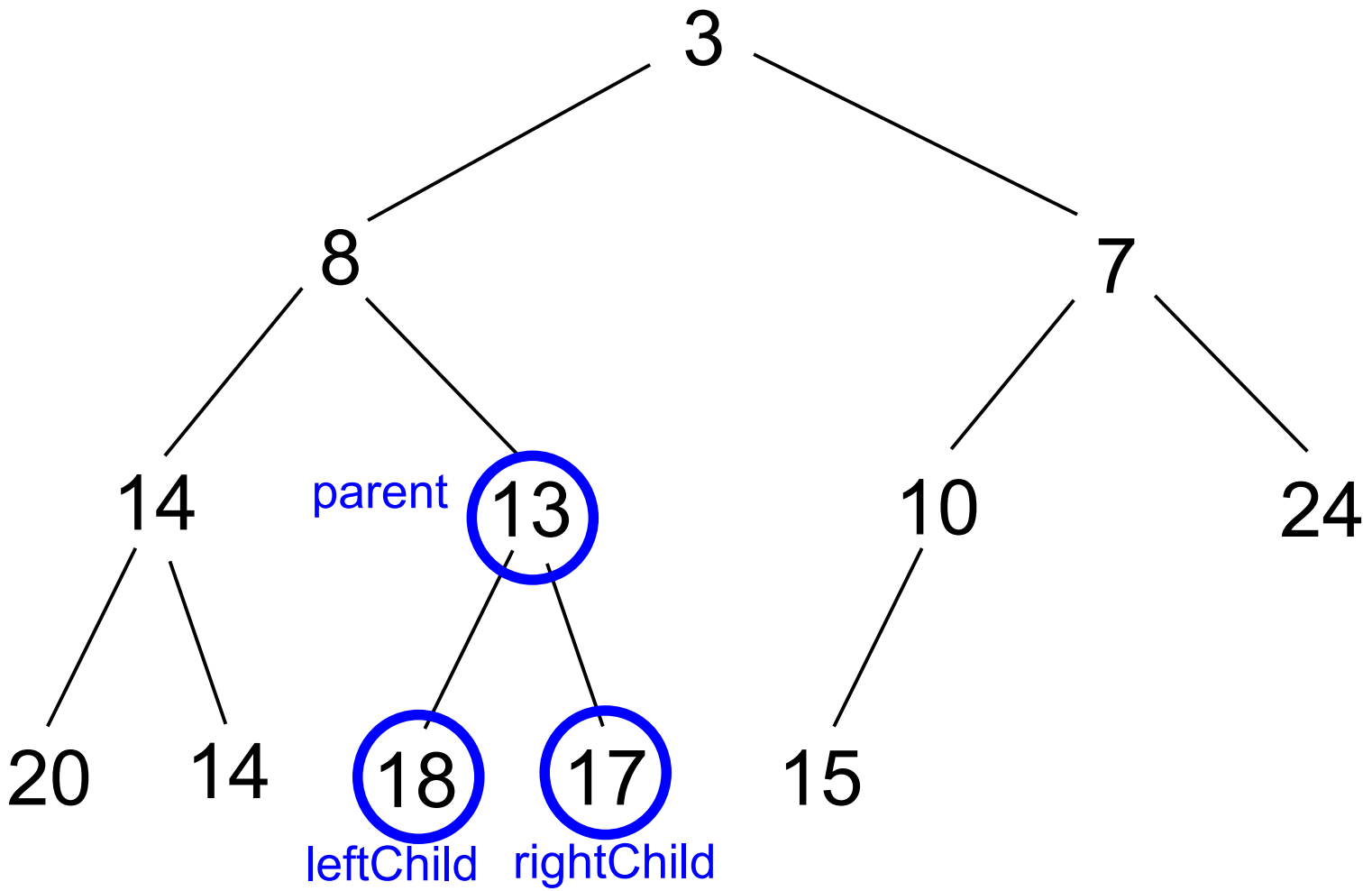

3
8
parent
7
14
13
leftChild
10
24
rightChild
20
14
18
17
15

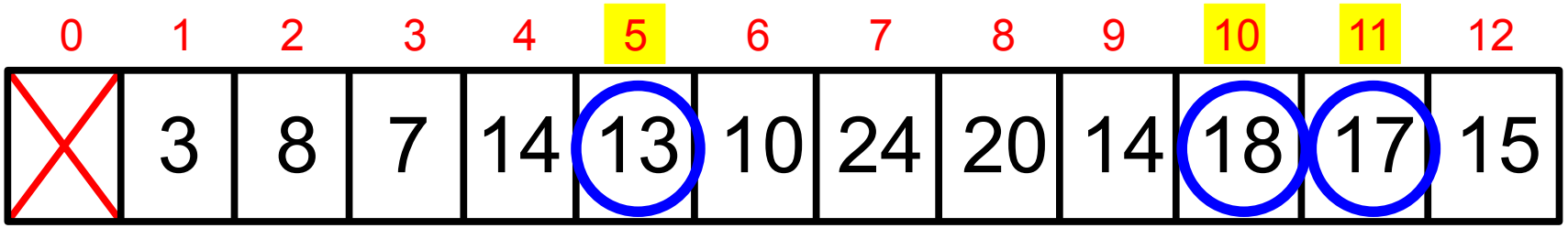

0
1
2
3
4
5
6
7
8
9
10
11
12
3
8
7
14
13
10
24
20
14
18
17
15

Heaps

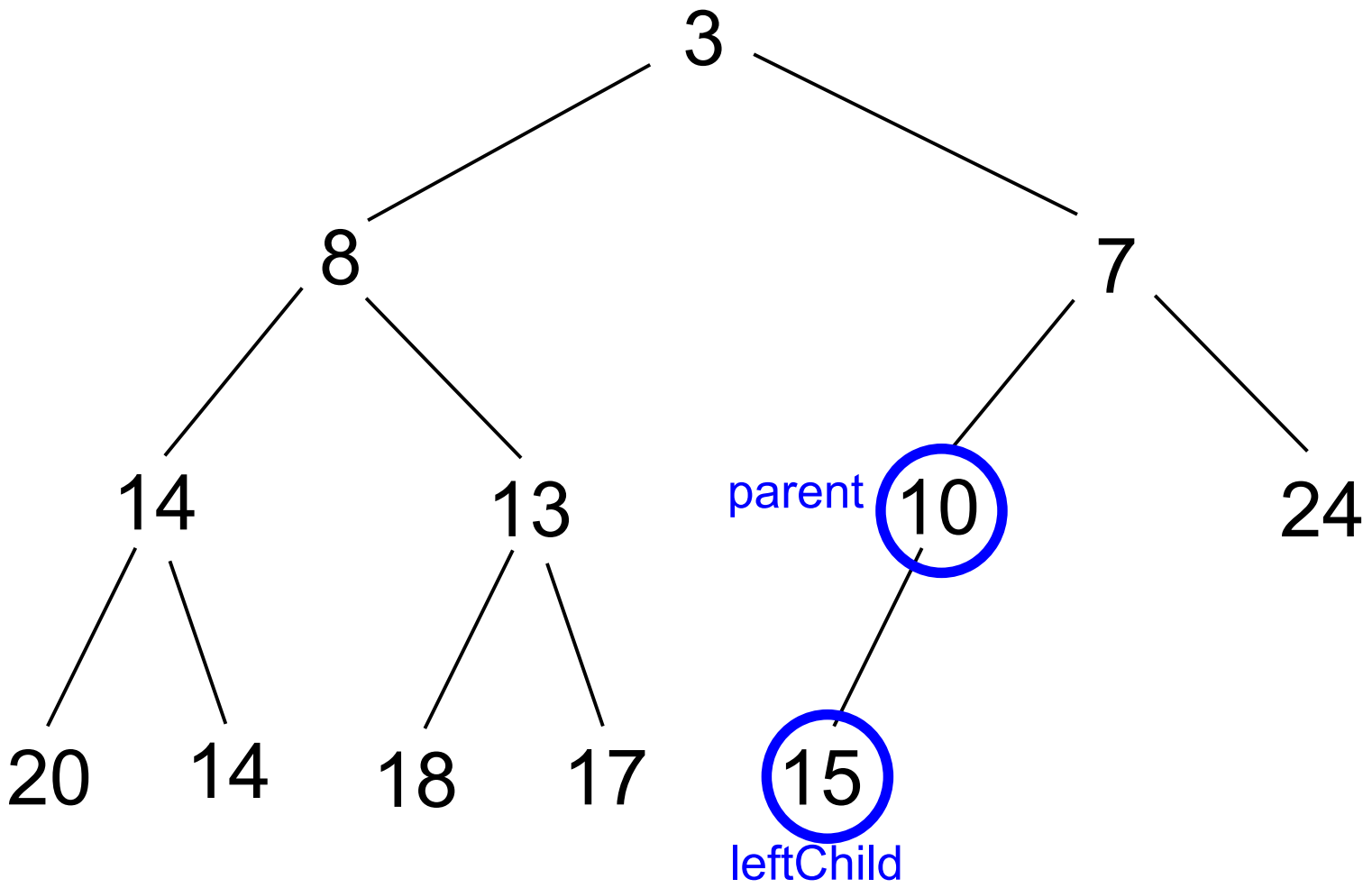

3
8
7
parent
14
13
10
24
20
14
18
17
15
leftChild
rightChild
0 1 2 3 4 5 6 7 8 9 10 11 12
3 8 7 14 13 10 24 20 14 18 17 15

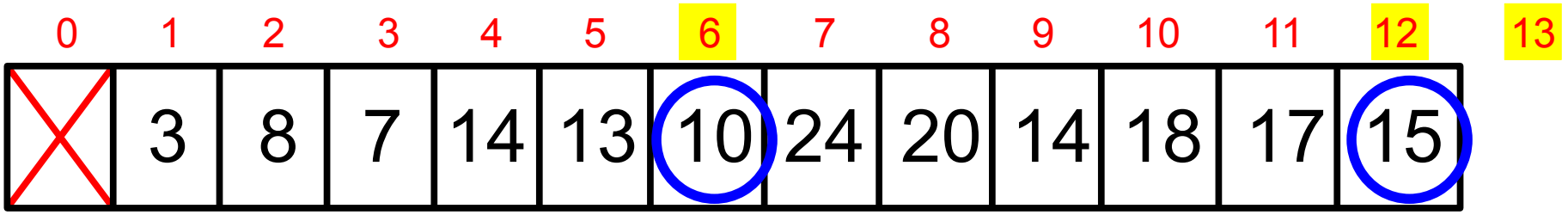
Heaps


3
8
7
14
parent
13
10
24
20
14
18
17
15
leftChild
rightChild

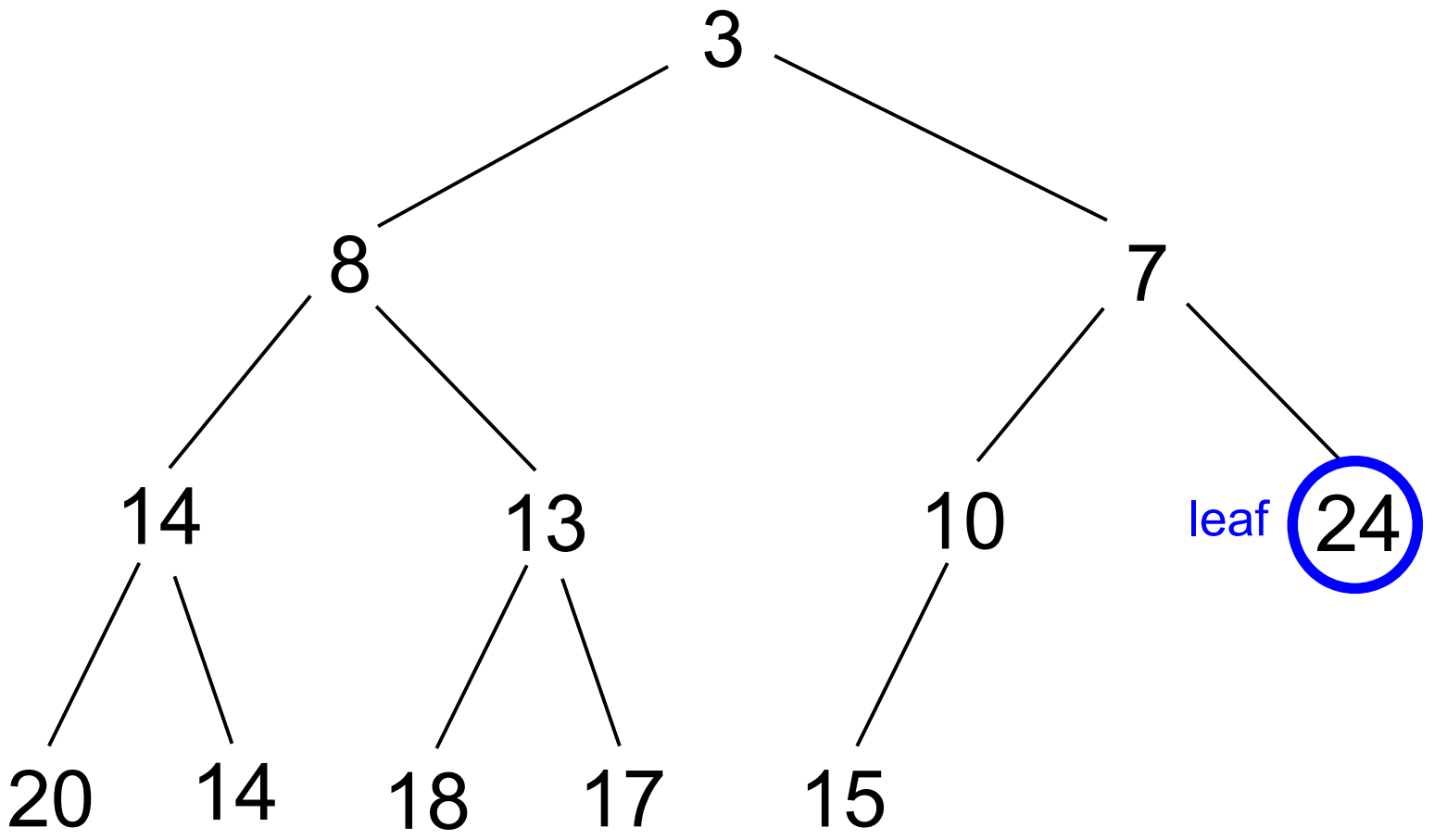

0 1 2 3 4 5 6 7 8 9 10 11 12
3 8 7 14 13 10 24 20 14 18 17 15

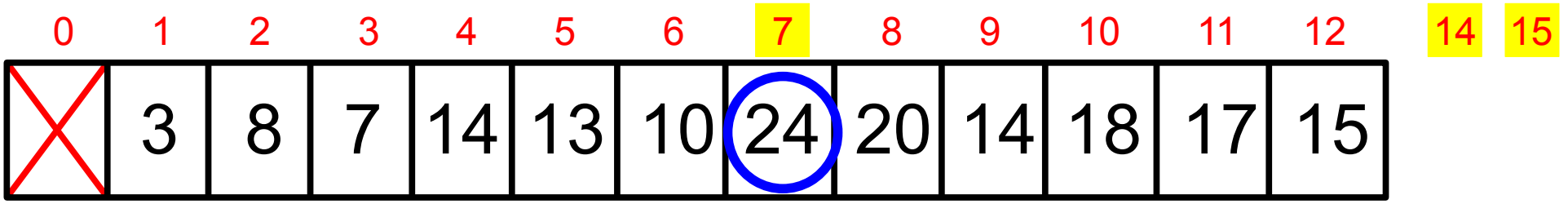
Heaps


3
8
7
14
13
parent
10
24
20
14
18
17
15
leftChild

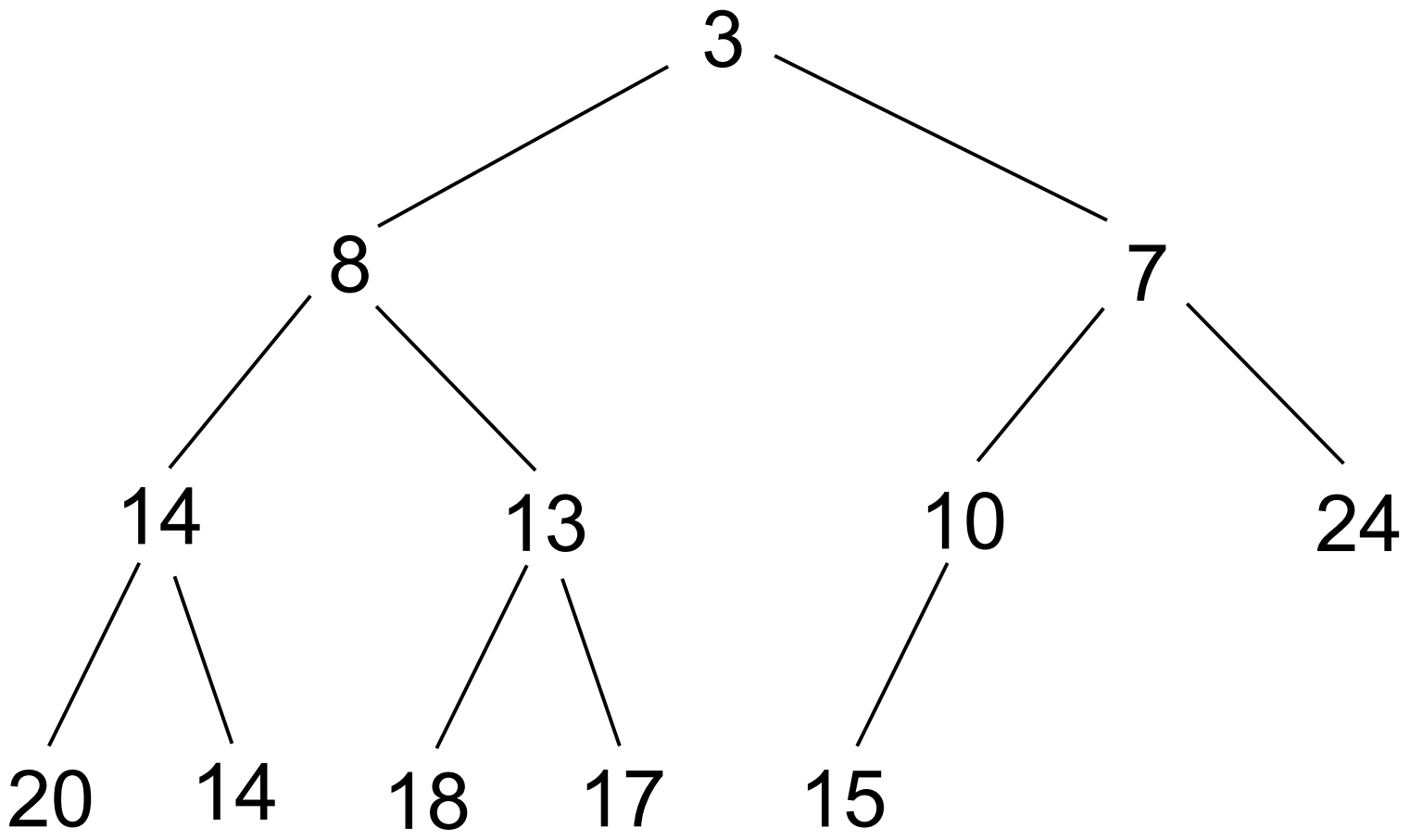

0 1 2 3 4 5 6 7 8 9 10 11 12 13
3 8 7 14 13 10 24 20 14 18 17 15

Heaps


3
8
7
14
13
10
leaf
24
20
14
18
17
15

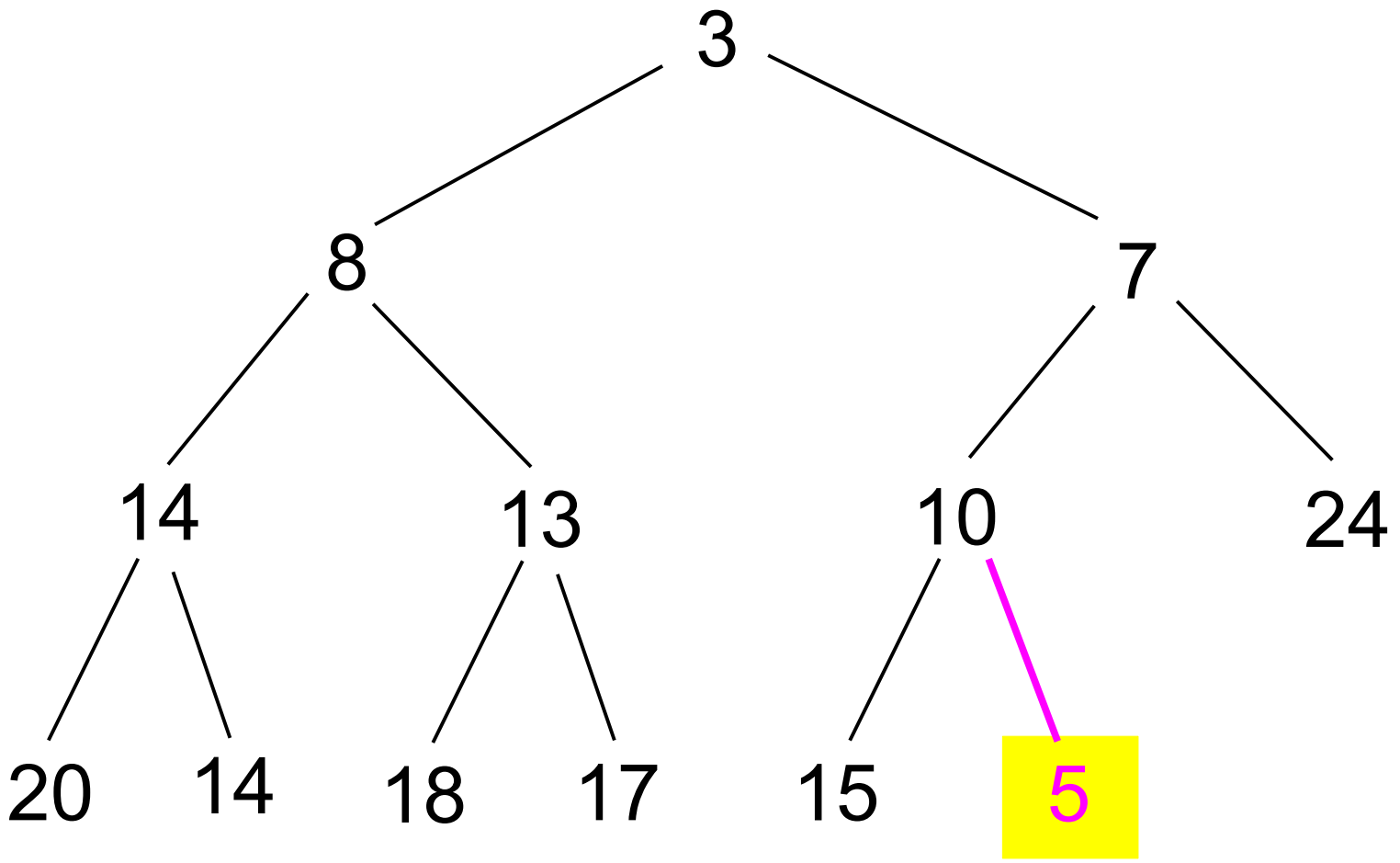

0 1 2 3 4 5 6 7 8 9 10 11 12 14 15
3 8 7 14 13 10 24 20 14 18 17 15

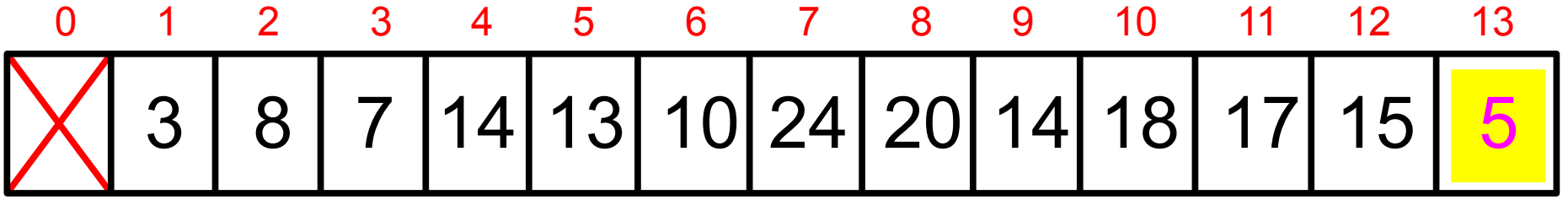
Inserting a New Element



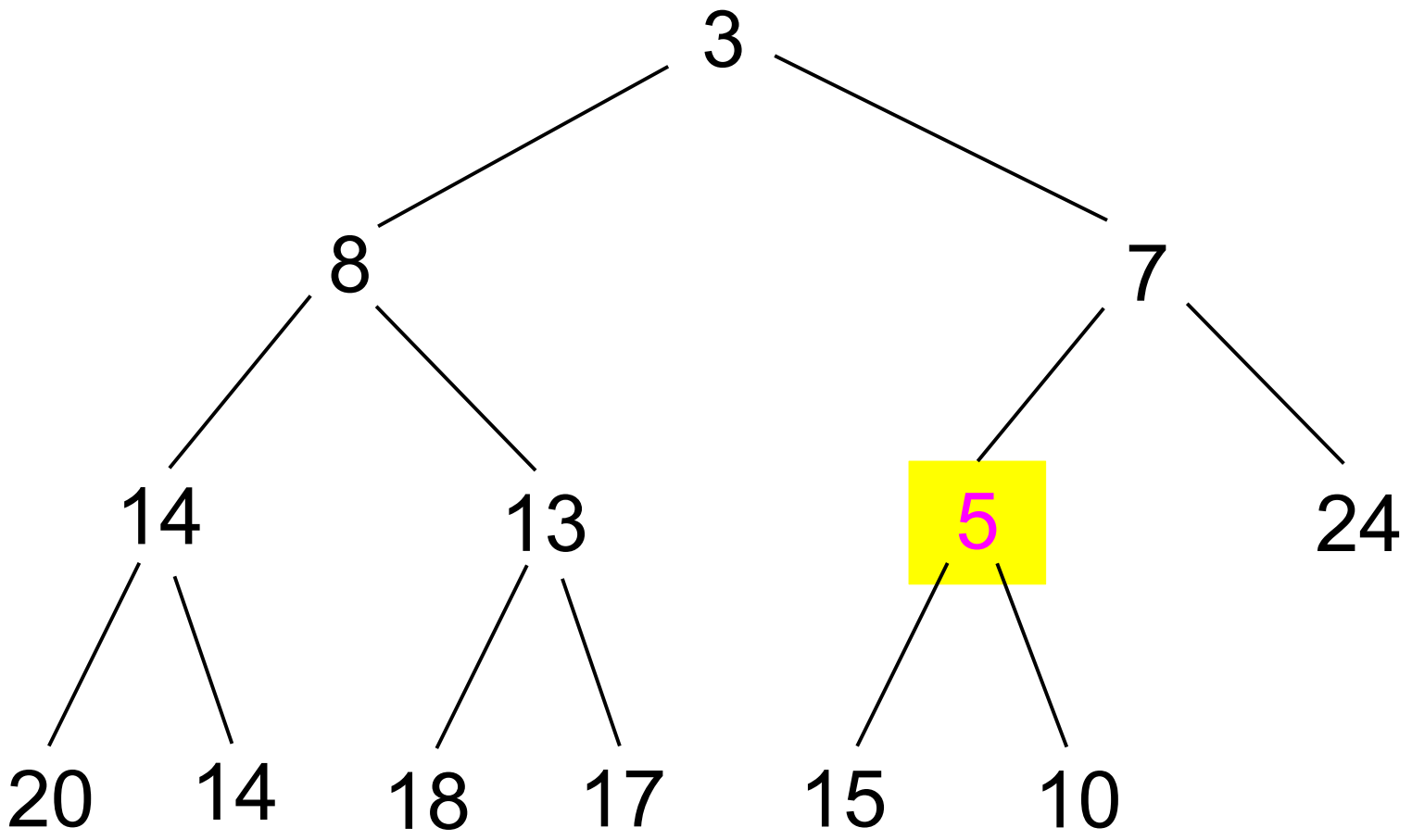
0	1	2	3	4	5	6	7	8	9	10	11	12
X	3	8	7	14	13	10	24	20	14	18	17	15

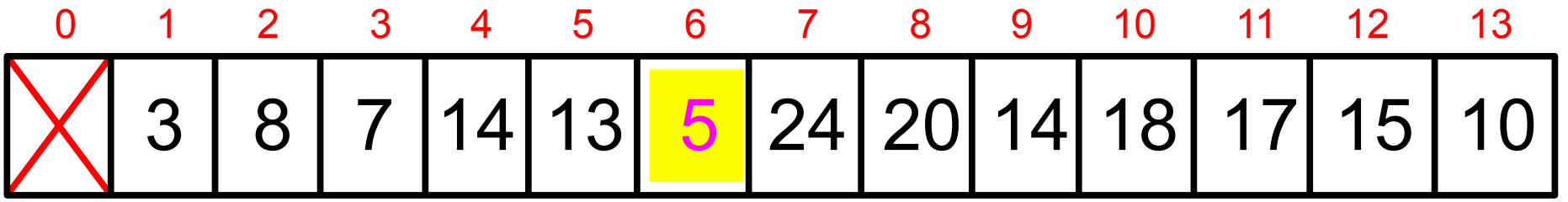
Inserting a New Element



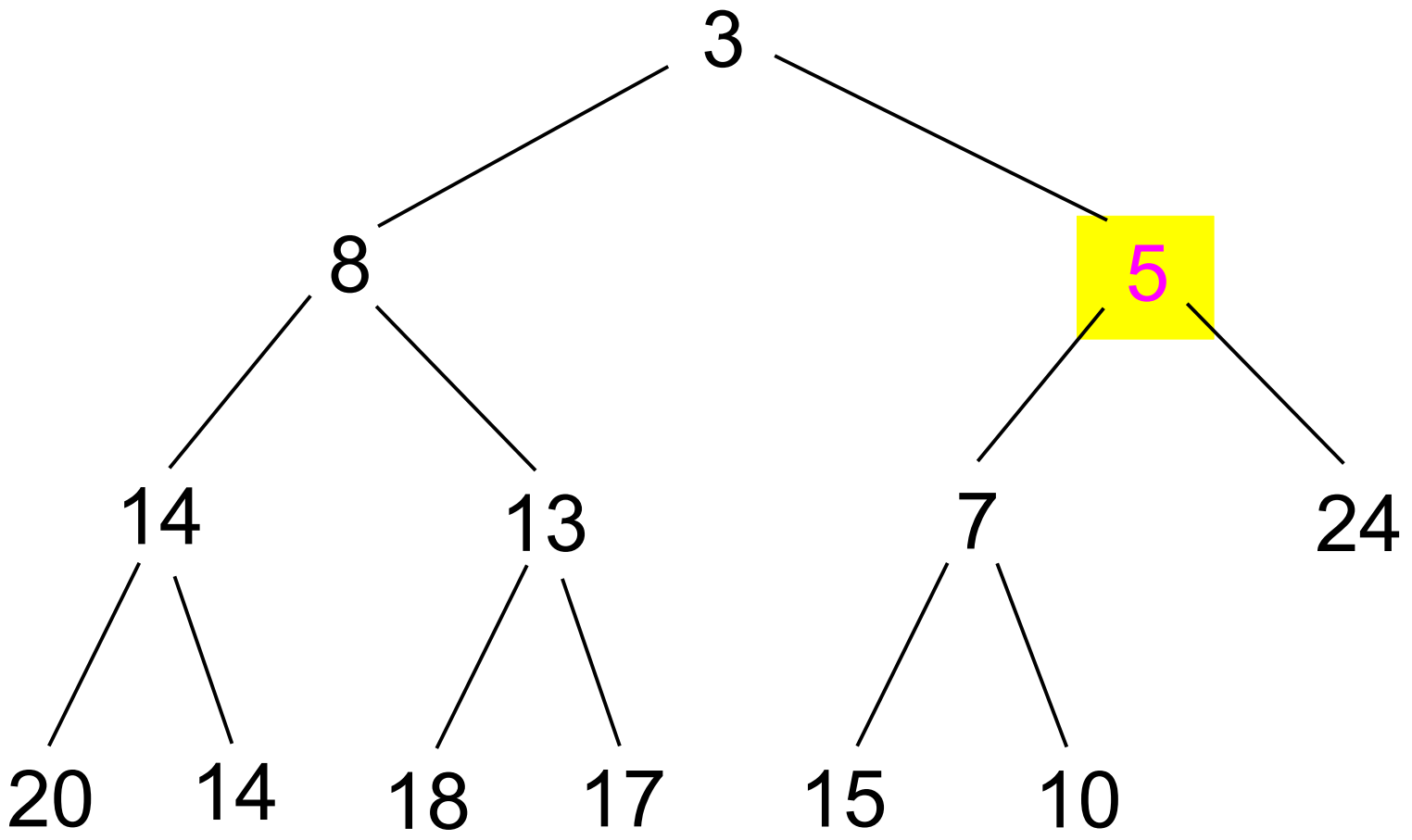


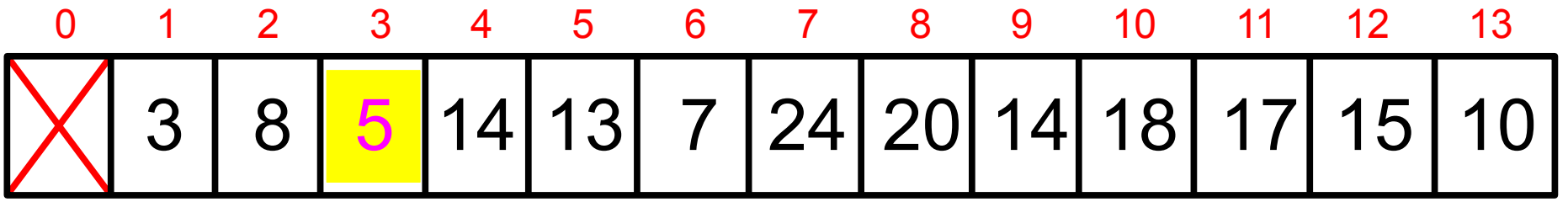
Inserting a New Element



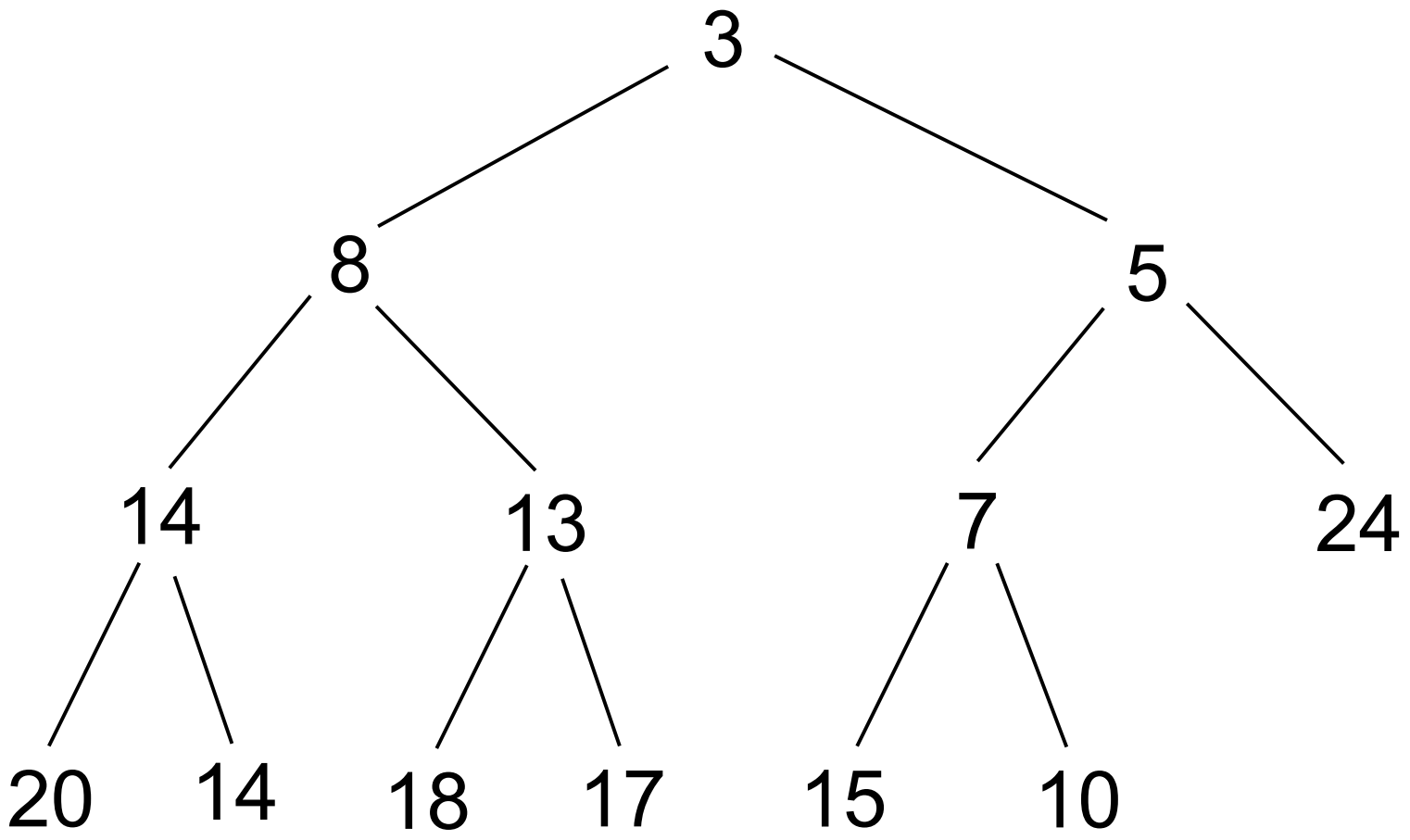


Inserting a New Element



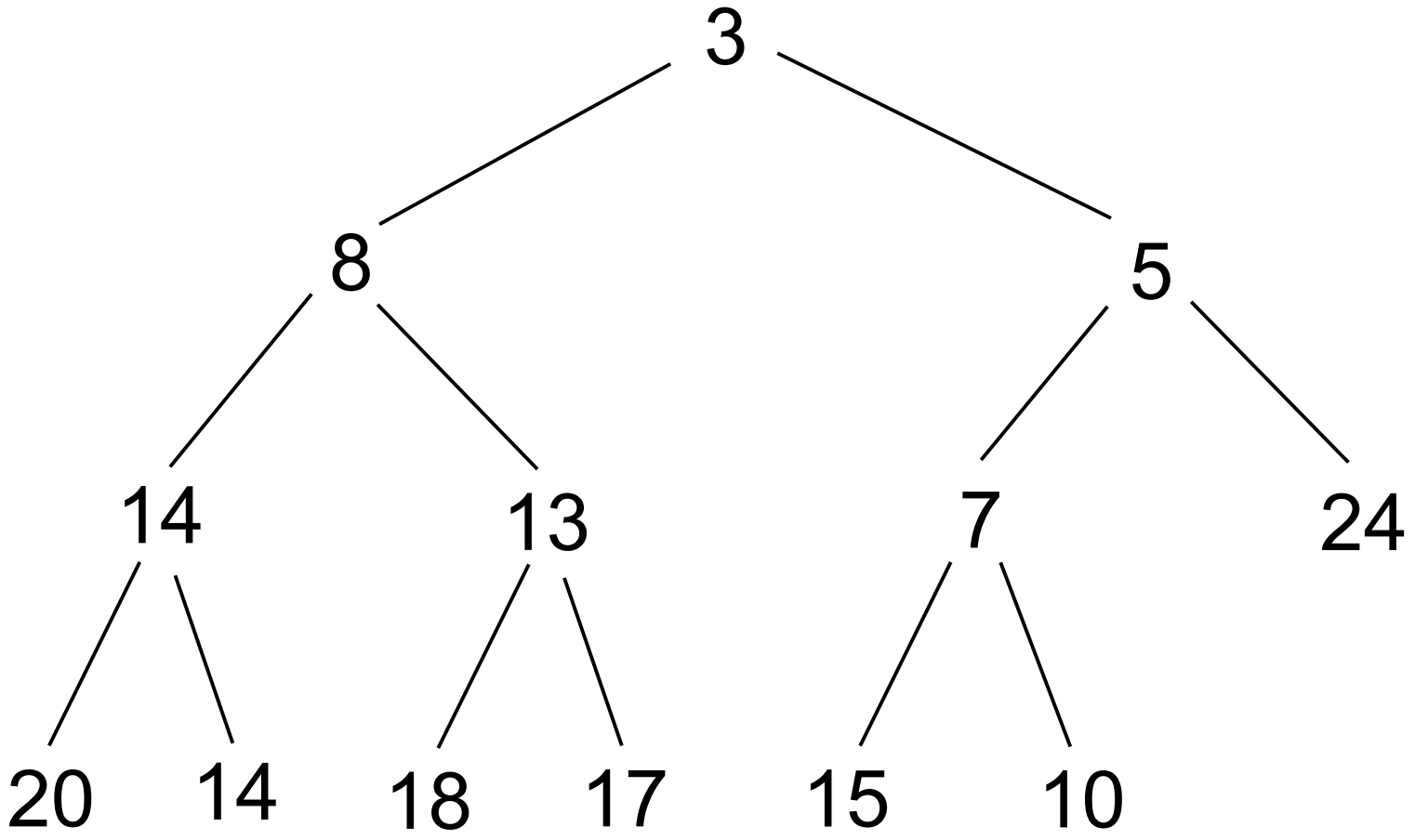


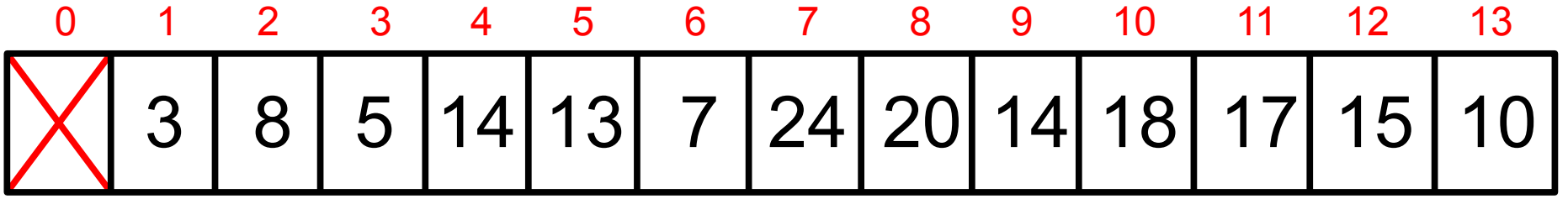
Inserting a New Element



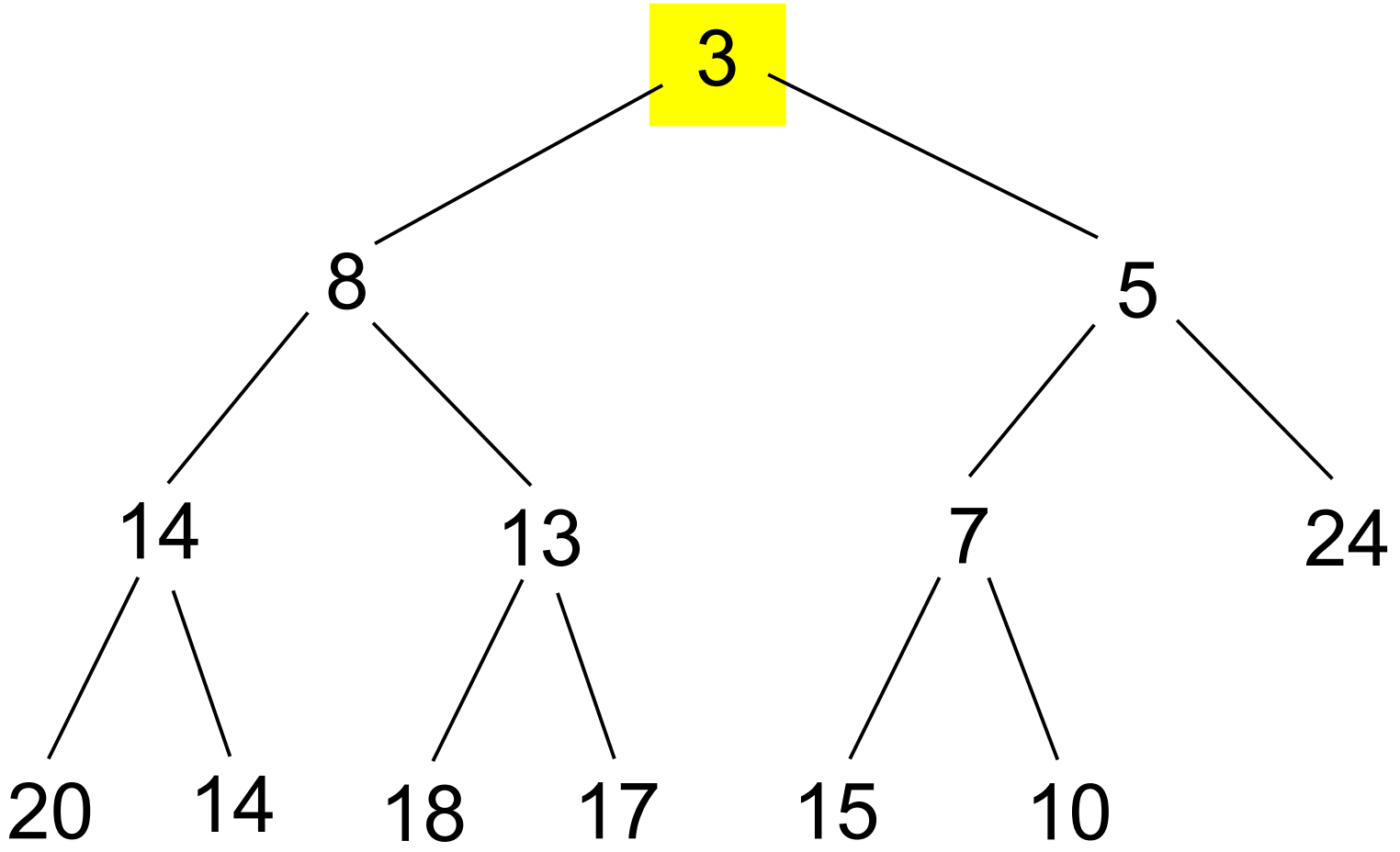
0	1	2	3	4	5	6	7	8	9	10	11	12	13
X	3	8	5	14	13	7	24	20	14	18	17	15	10

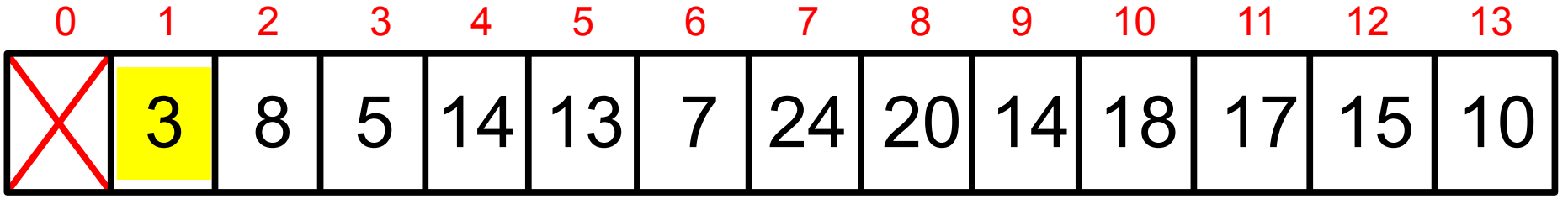
Time Complexity of Insert: O(log *n*)



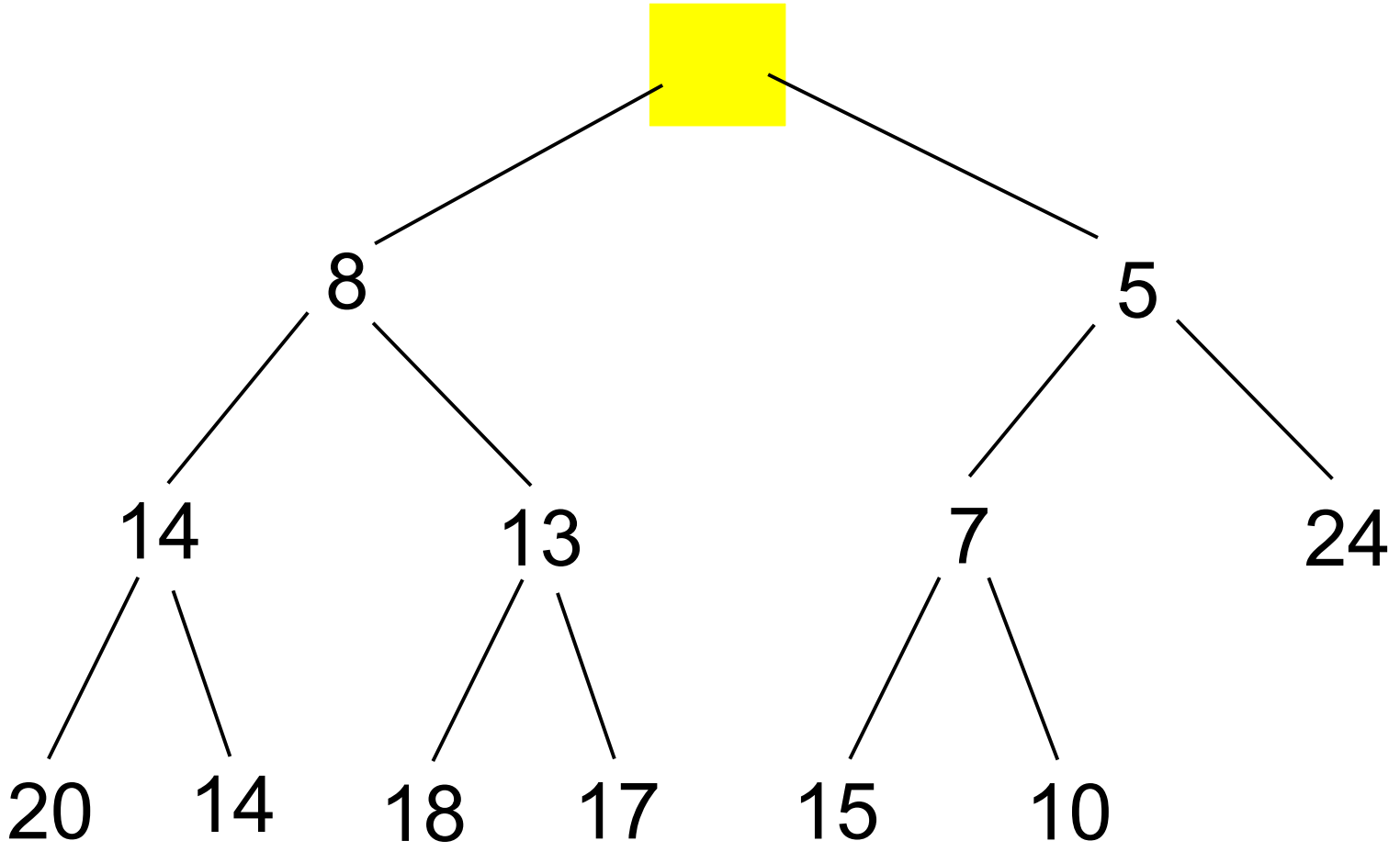


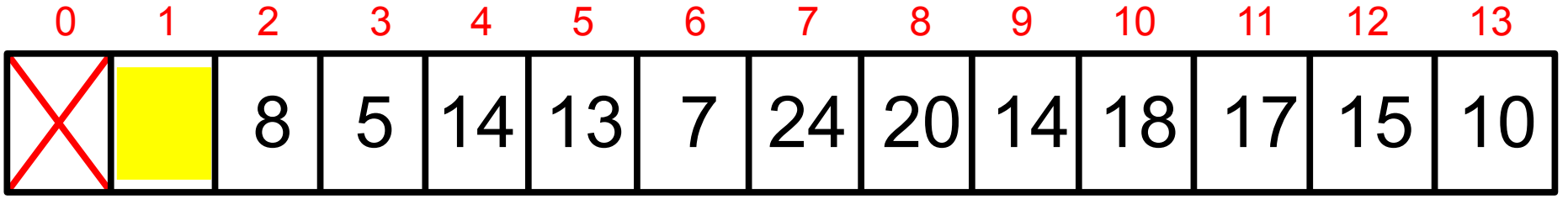
Removing the Smallest Element



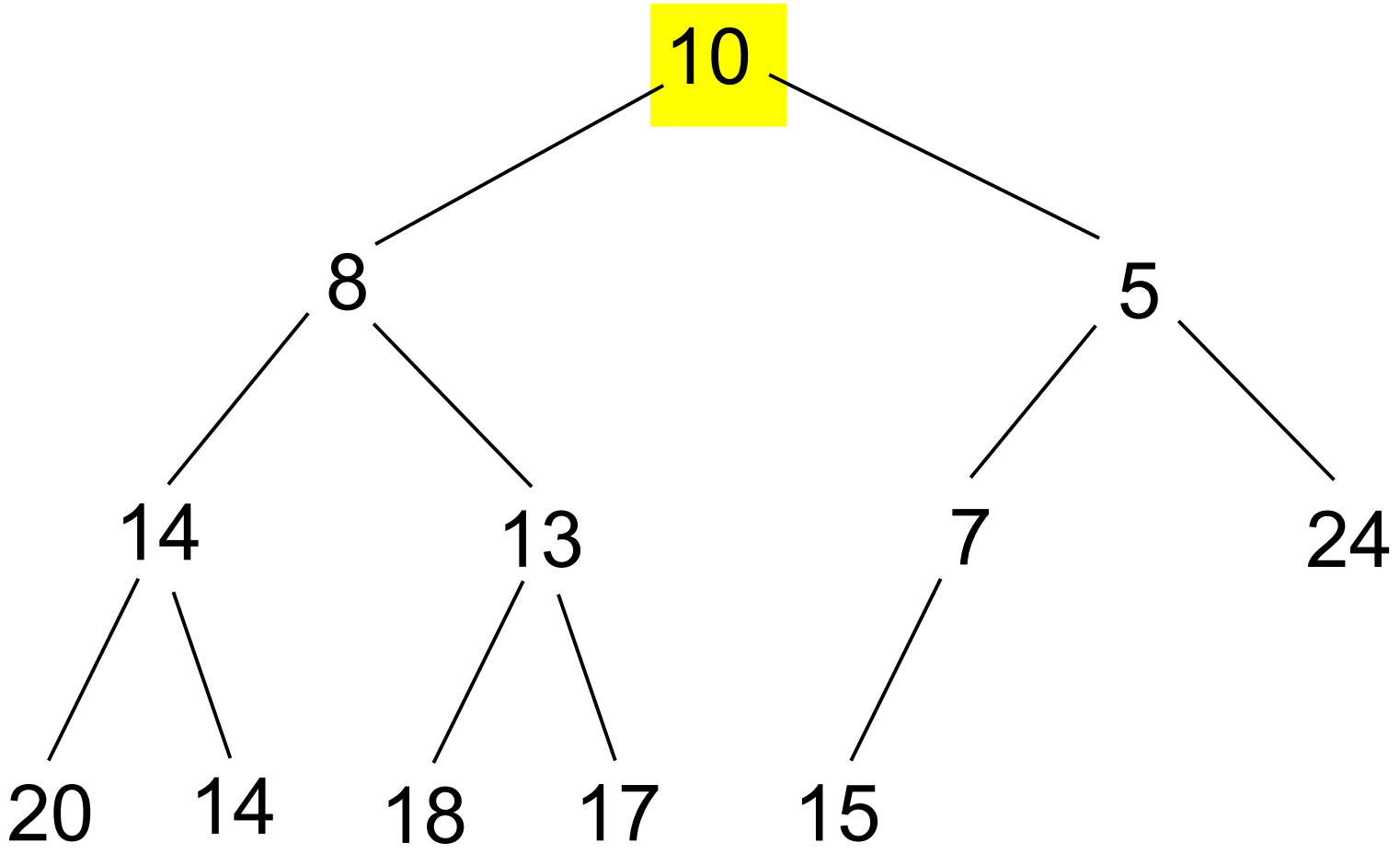


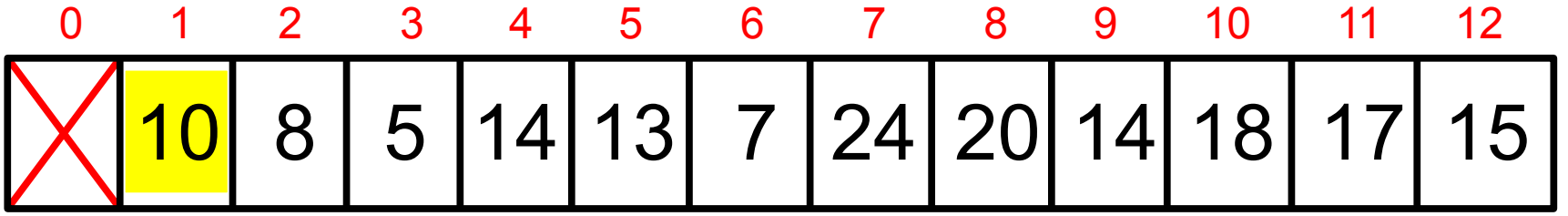
Removing the Smallest Element



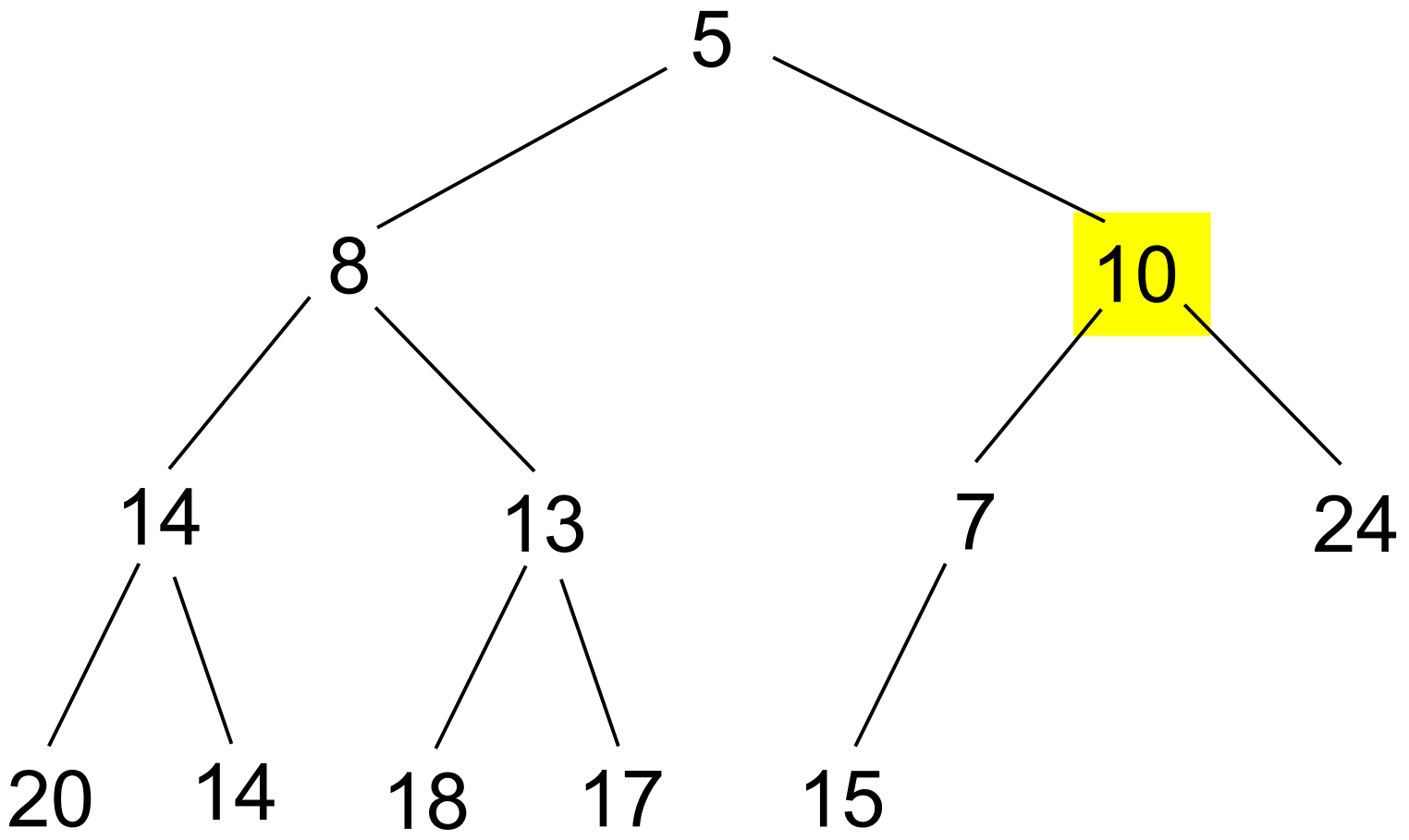


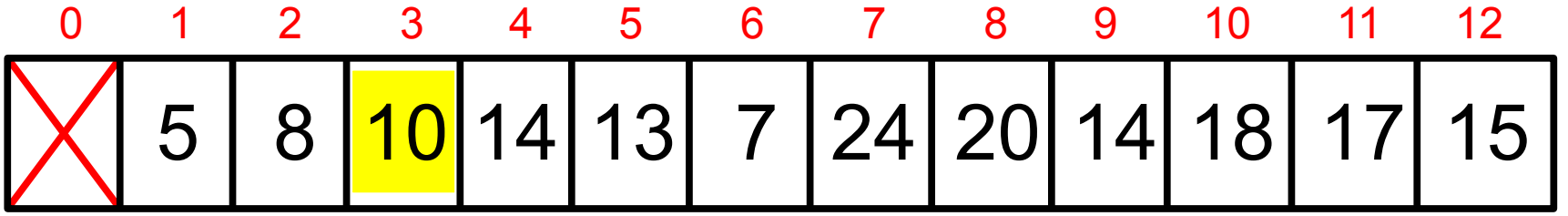
Removing the Smallest Element



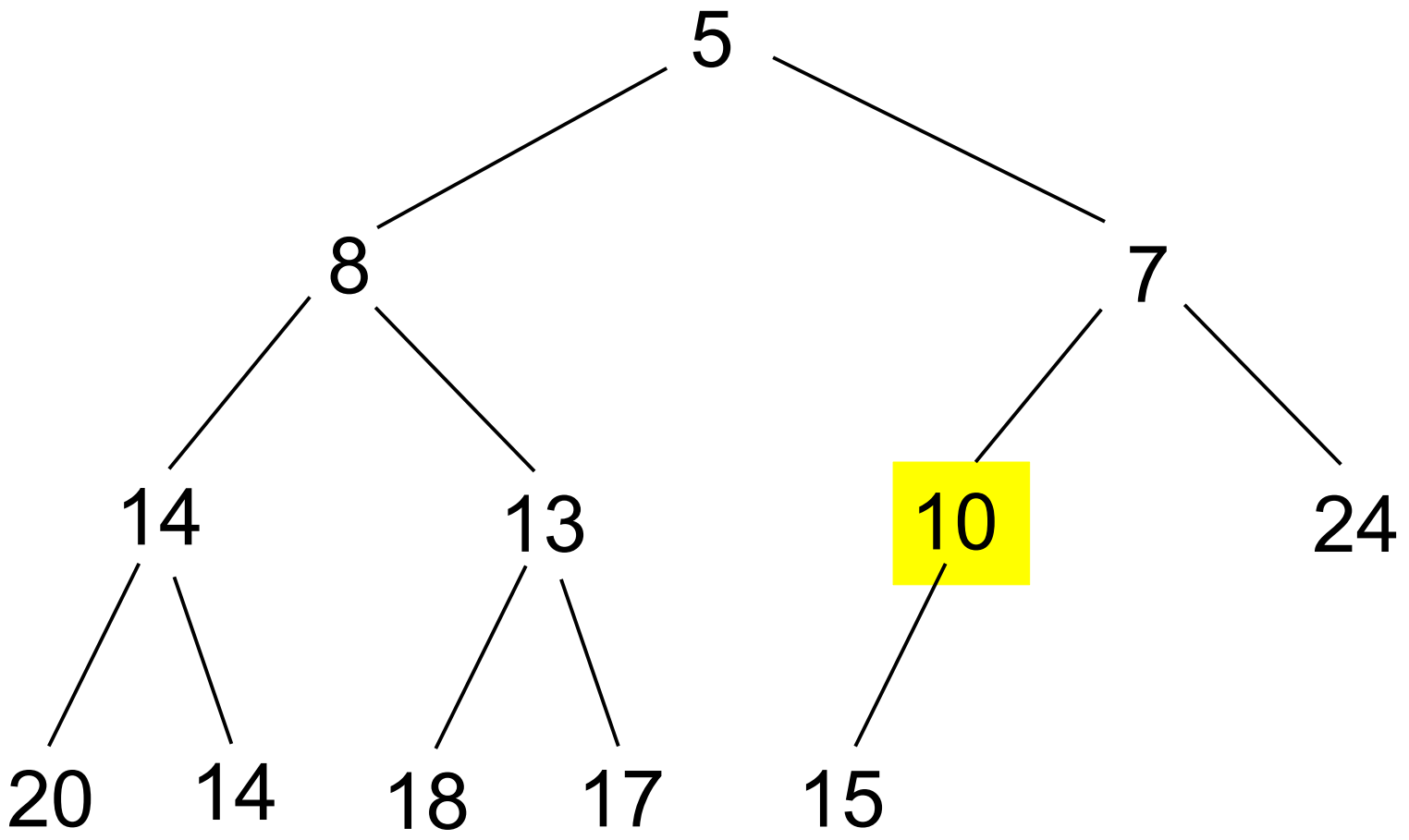


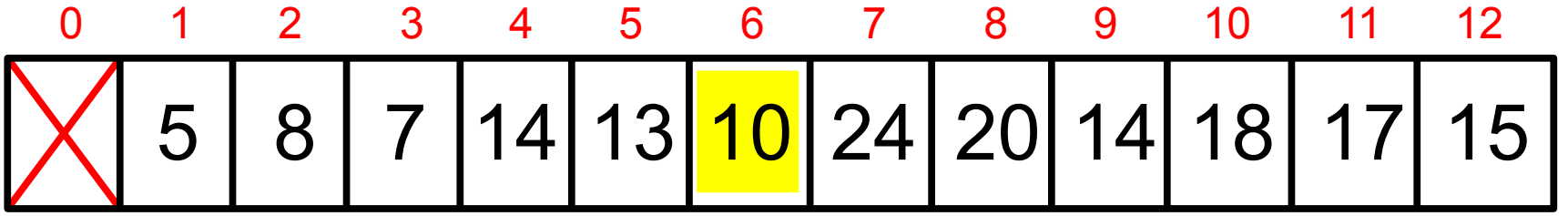
Removing the Smallest Element



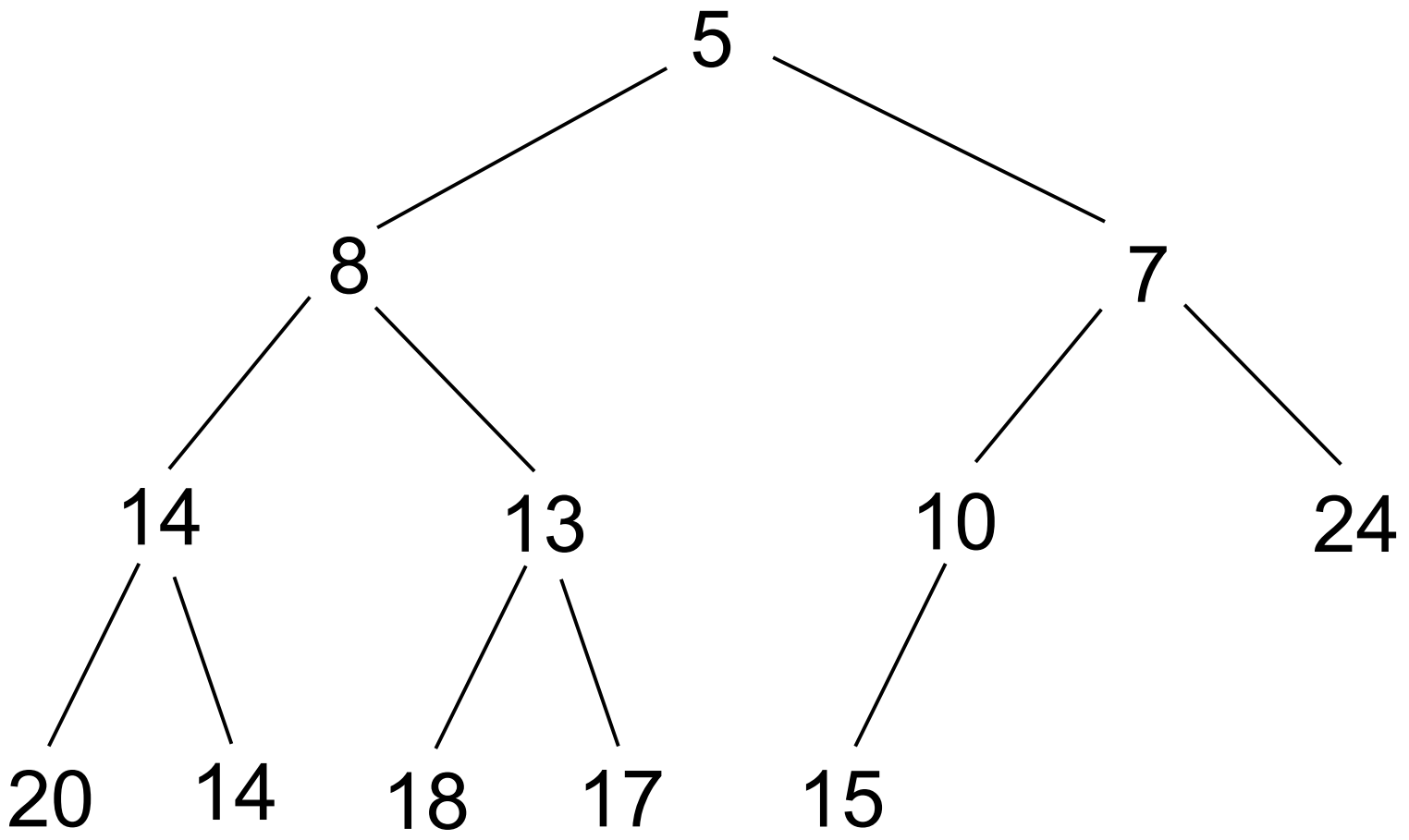


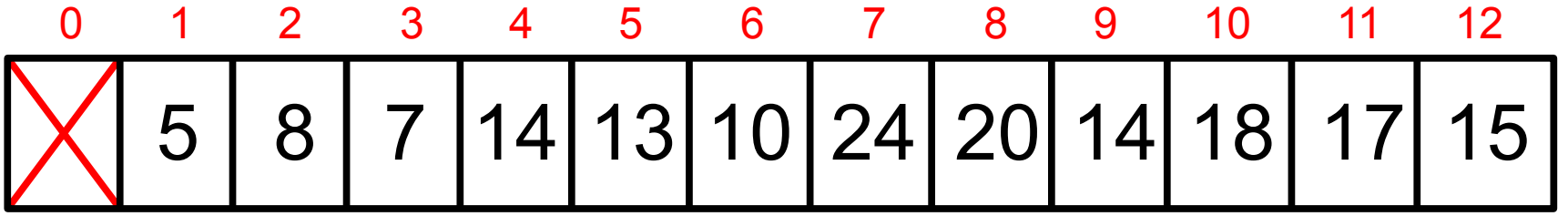
Removing the Smallest Element



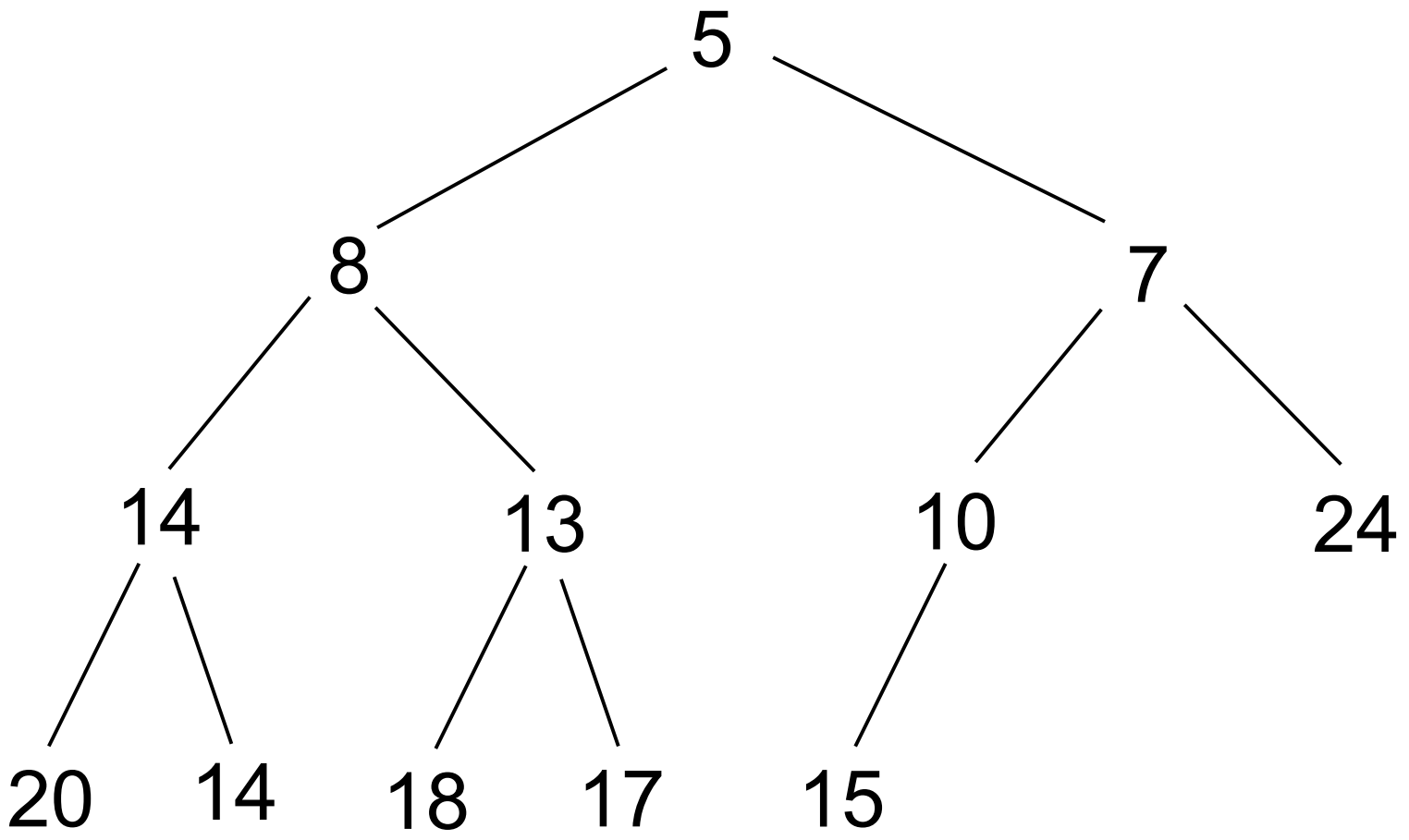


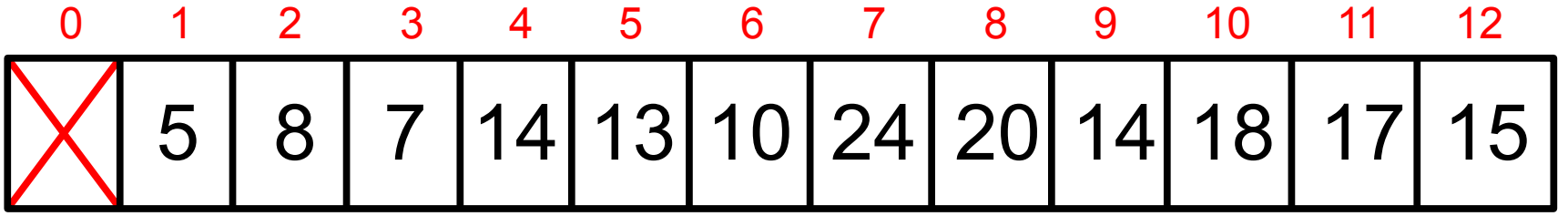
Removing the Smallest Element





Time Complexity of Remove: O(log *n*)





Heaps: Summary

- Duplicate elements allowed
- Good for representing priority queues

- Inserting elements is fast: O(log *n*)
- Retrieving smallest element is very fast: O(1)
- Removing smallest element is fast: O(log *n*)

- Testing for membership is inefficient
- Removing other elements is inefficient