Heaps

Priority Queues

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Priority Queues

- Heaps or priority queues are data structures used to store elements and manage them based on their priority
  - Each data structure must incude a field used as priority, such that all main operations are based on such a field
- Priority queues are often uses to manage limited resources and have several applications
  - Huffman coding, Best-First Search (A*), Prim's algorithm, Routing, Job scheduling, etc.

Priority Queues

- It is possible to implement
  - Min-priority queues
    - The maximum priority is given to the element with the minimum priority value
  - Max-priority queues
    - The maximum priority is given to the element with the maximum priority value

Priority Queues

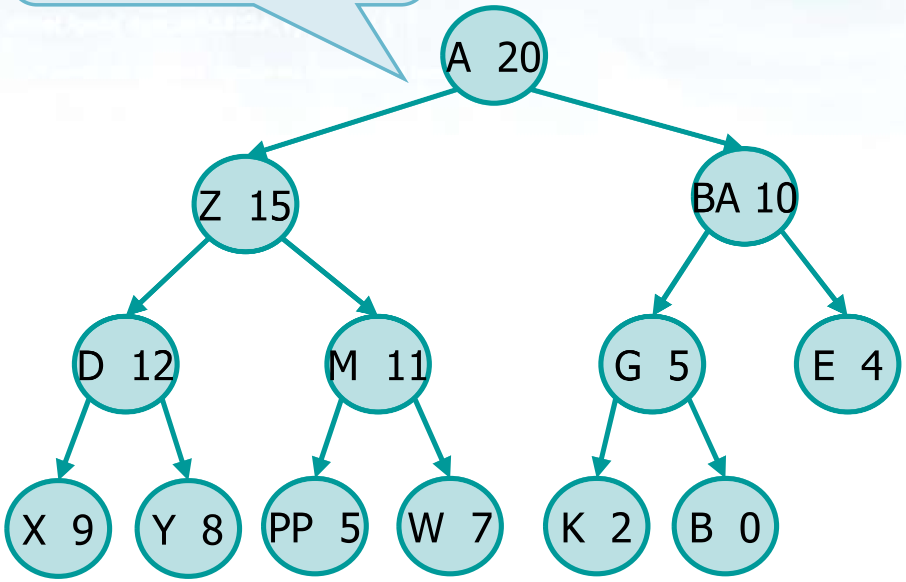
- Main operations
  - Insert, extract maximum, read maximum, change priority
- There are several possible data structure implementations
  - Unordered array/list
  - Ordered array/list

  but heaps are more efficient

As we just need to extract the element with the maximum priority, completely sorting all elements is useless

Example

PQ
(Priority Queue)




```
#define LEFT(i)    (2*i+1)
#define RIGHT(i)   (2*i+2)
#define PARENT(i)  ((int)(i-1)/2)
```

```
struct heap_s {
  Item *A;
  int heapsize;
} heap_t;
```

Array representation

heap->A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	Z	BA	D	M	G	E	X	Y	PP	W	K	B		
20	15	10	12	11	5	4	9	8	5	7	2	0		

heap->heapsize = 13

Heap ←→ PQ

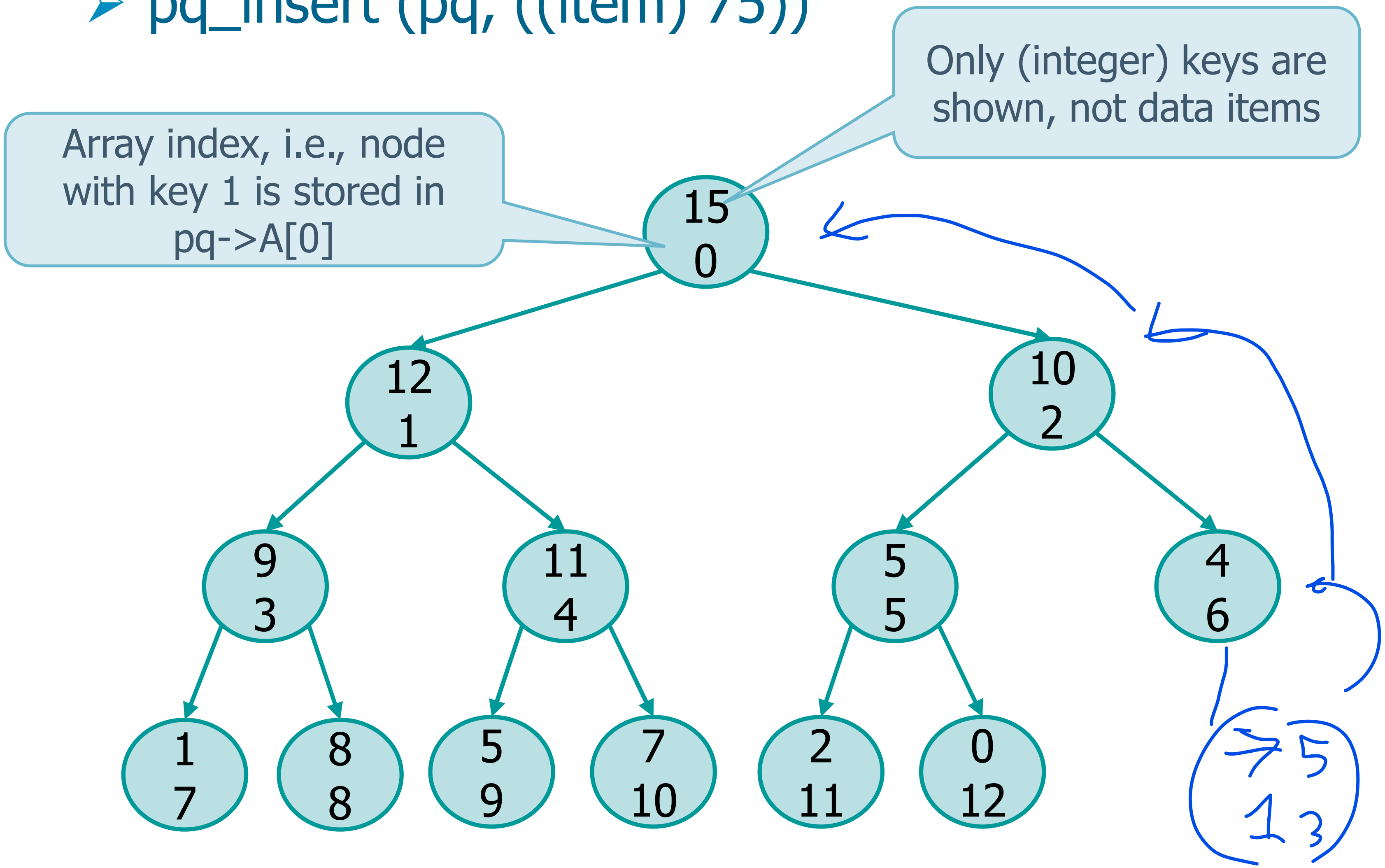
Array (maximum)
maxN = 15

Function pq_insert

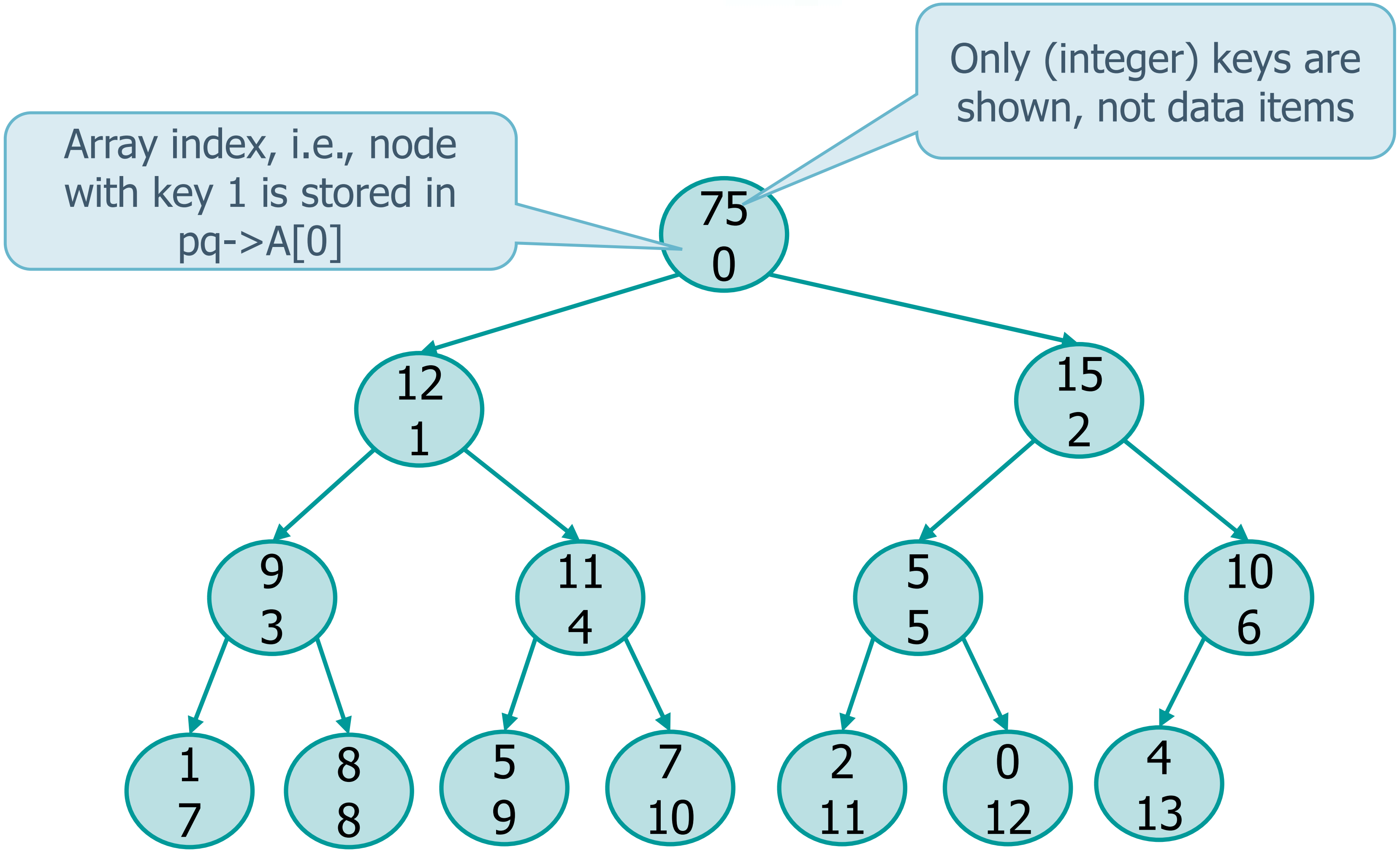
- Add a leaf to the tree
  - It grows level-by-level from left to right satisfying the structural property
- From current node up (initially the newest leaf) up to the root
  - Compare the parent's key with the new node's key, moving the parent's data from the parent to the child when the key to insert is larger
  - Otherwise insert the data into the current node
- Complexity
  - $T(n) = O(log_2 n)$

Exercise

- Given the following max-heap, show the result of
  - pq_insert (pq, ((item) 75))



Solution


Only (integer) keys are shown, not data items
Array index, i.e., node with key 1 is stored in pq->A[0]
75
0
12
1
15
2
9
3
11
4
5
5
10
6
1
7
8
8
5
9
7
10
2
11
0
12
4
13

Implementation

Function
item_less
compares keys

```
void pq_insert (PQ pq, Item item) {
  int i;

  i  = pq->heapsize++;
  while( (i>=1) &&
         (item_less(pq->A[PARENT(i)], item)) )
    pq->A[i] = pq->A[PARENT(i)];
    i = PARENT (i);
  }
  pq->A[i] = item;

  return;
}
```

Increase the heap size

Move node down

Move up toward the root

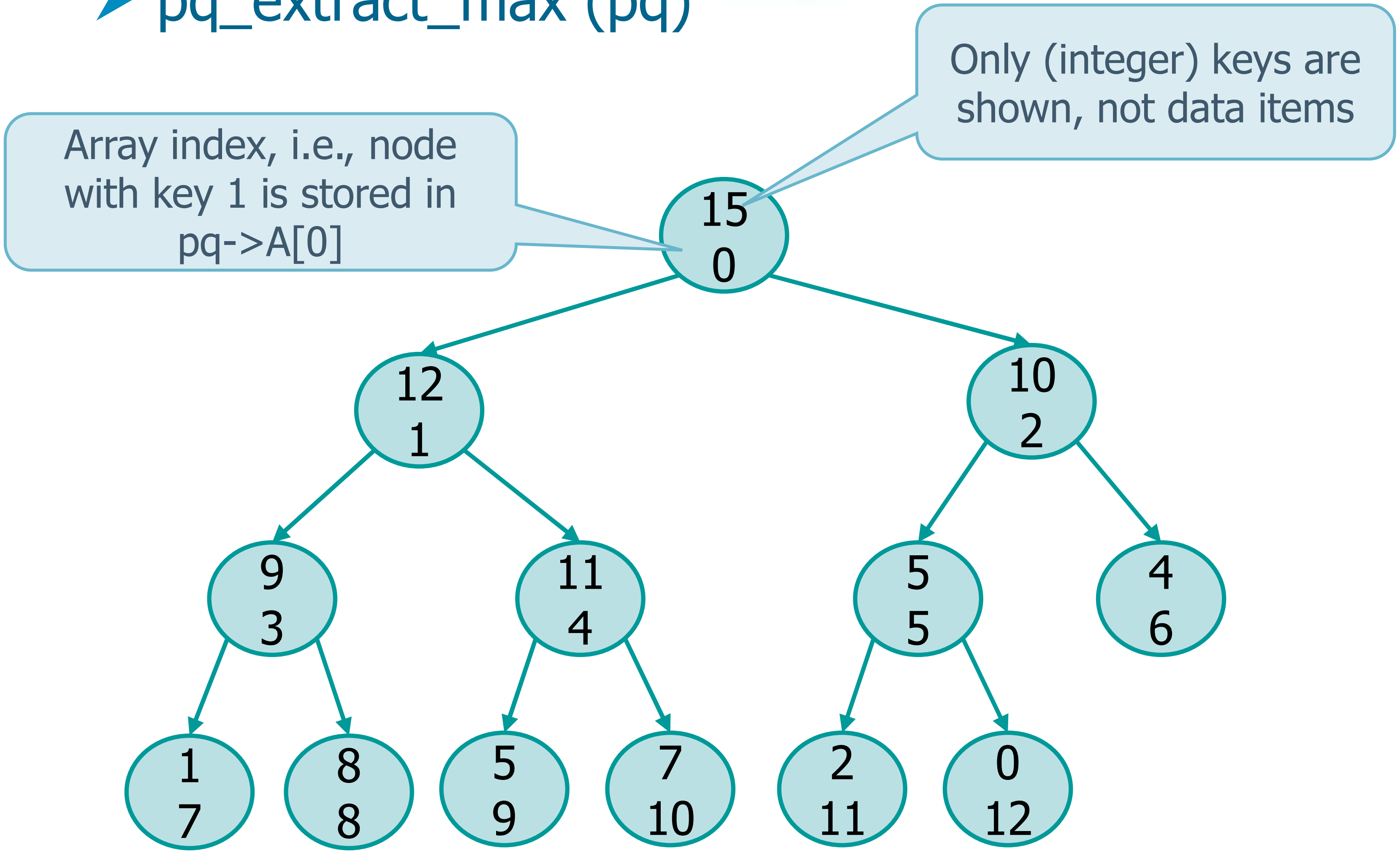
Insert new element in its final destination

Function pq_extract_max

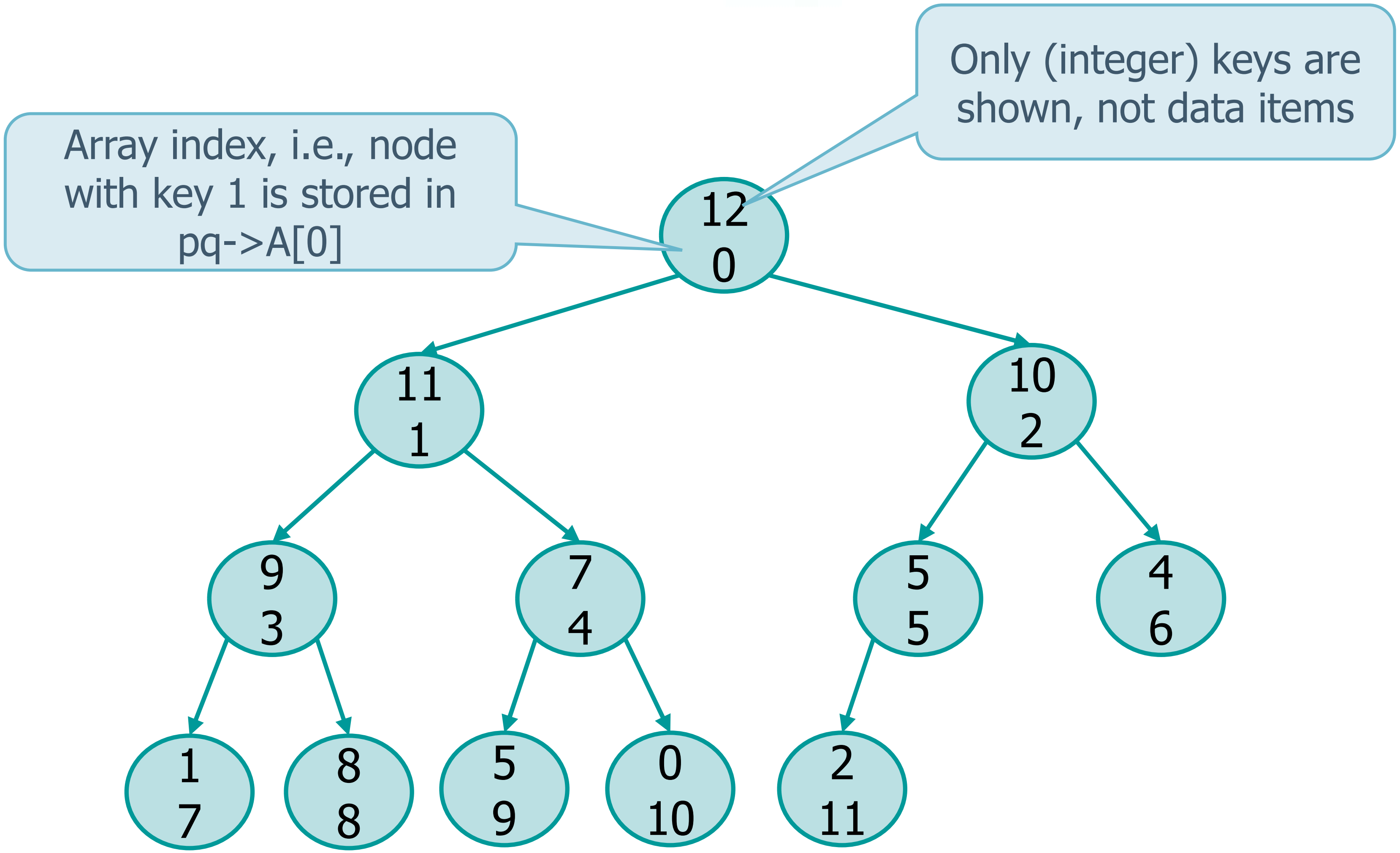
- Modify the head, by extracting the largest value, stored into the root
  - Swap root with the last leaf (the rightmost onto the last level)
  - Reduce by 1 the heap size
  - Restore the heap property by applying heapify
- Complexity
  - $T(n) = O(log_2 n)$

Example

- Given the following max-heap, show the result of
  - pq_extract_max (pq)



Solution


Only (integer) keys are shown, not data items
Array index, i.e., node with key 1 is stored in pq->A[0]
12
0
11
1
10
2
9
3
7
4
5
5
4
6
1
7
8
8
5
9
0
10
2
11

Implementation

```
Item pq_extract_max(PQ pq) {
  Item item;

  swap (pq, 0, pq->heapsize-1);
  item = pq->A[pq->heapsize-1];
  pq->heapsize--;
  heapify (pq, 0);

  return item;
}
```

Extract max and move last element into the root node

Reduce heap size

Heapify from root

Function pq_change

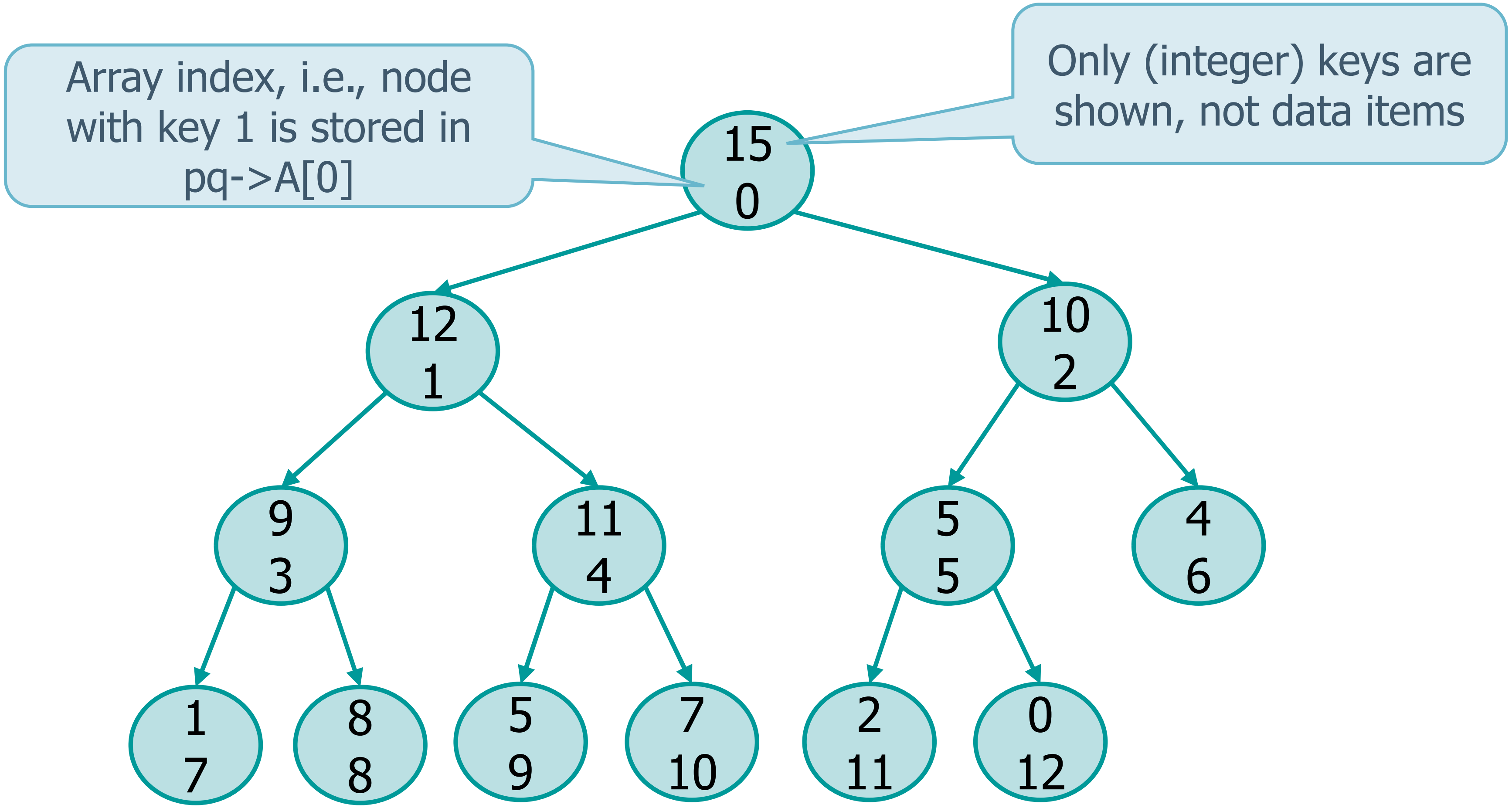
- Modify the key of an element **in a given position** given its index
- Can be implemented as two separate operations
  - Decrease key
    - When a key is decreased, we may need to move it downward
    - To move a key downward, we can adopt the same process analyze in **heapify**
      - Heapify keeps moving the key from the parent to the child with the largest key until the key is inserted into the current node

Function pq_change

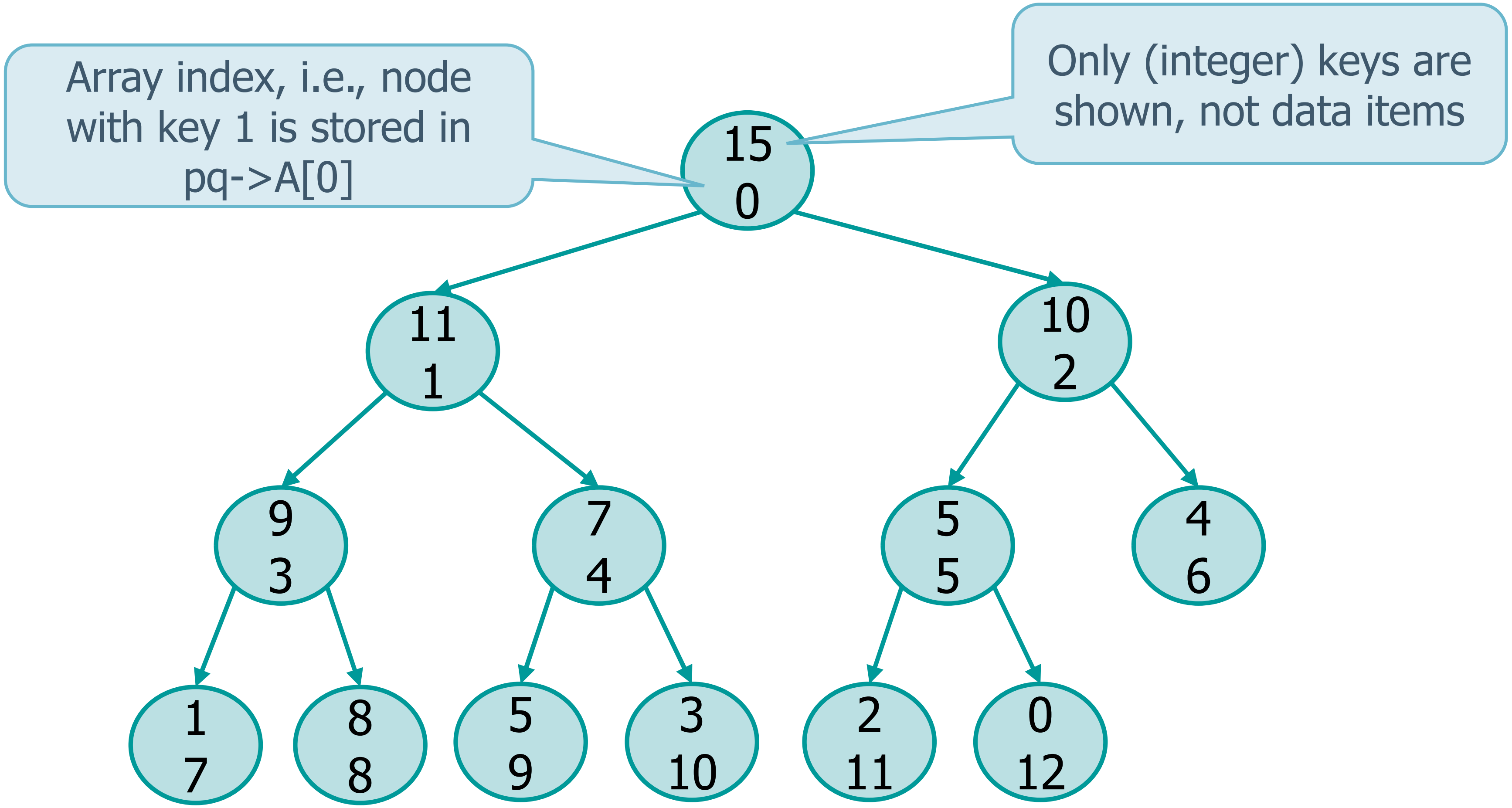
- Increase key
  - When a key is increased, we may need to move it upward
  - To move a key upward, we can adopt the same process analyze in **pq_insert**
    - We move the key up into the parent until the key is inserted into the current node
- Complexity
  - Dependent on the tree height
  - $T(n) = O(log_2 n)$

Example: Decrease key

- Given the following max heap, show the result of
  - pq_change (pq, 1, ((item) 3))

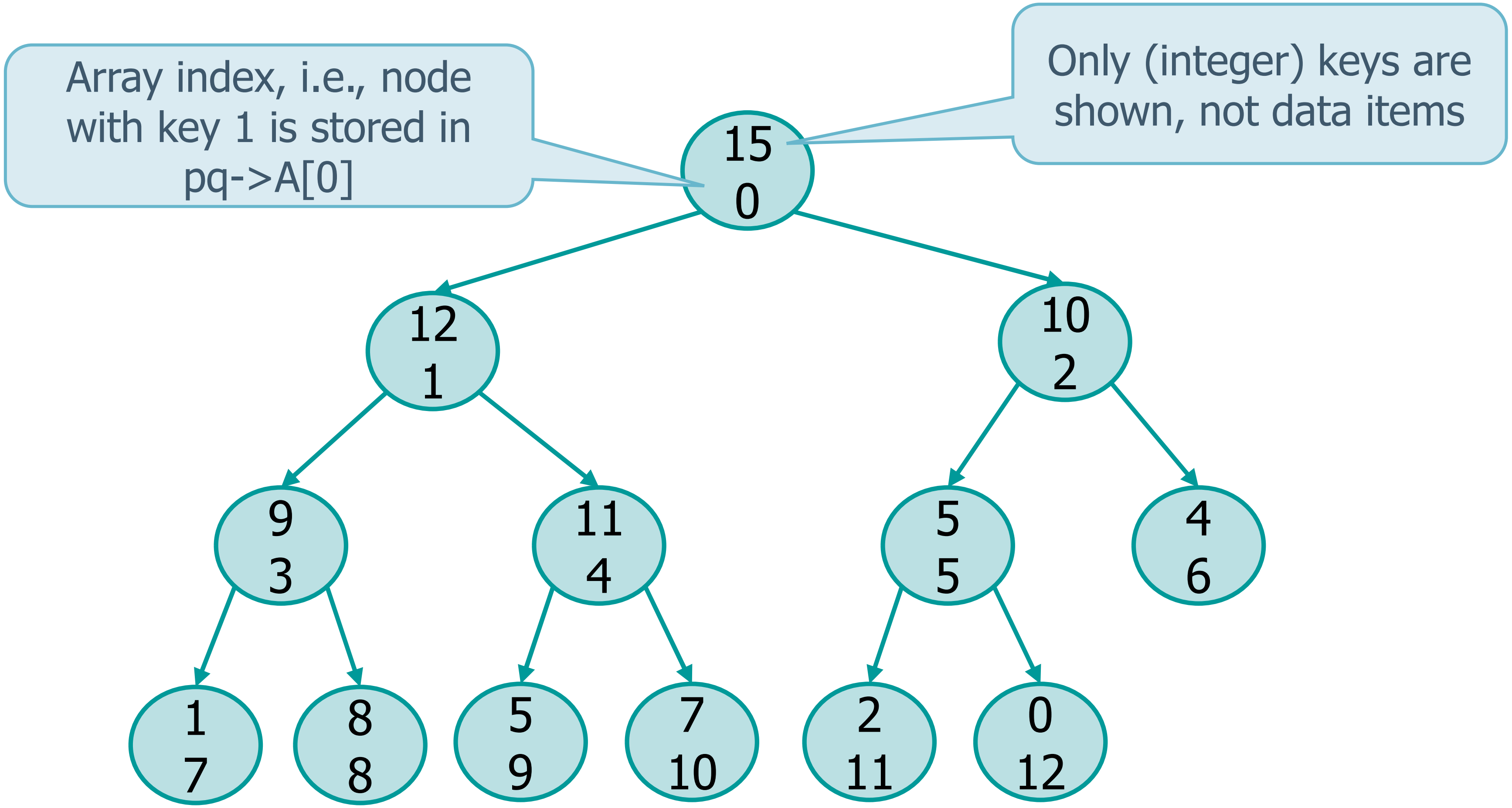


Solution

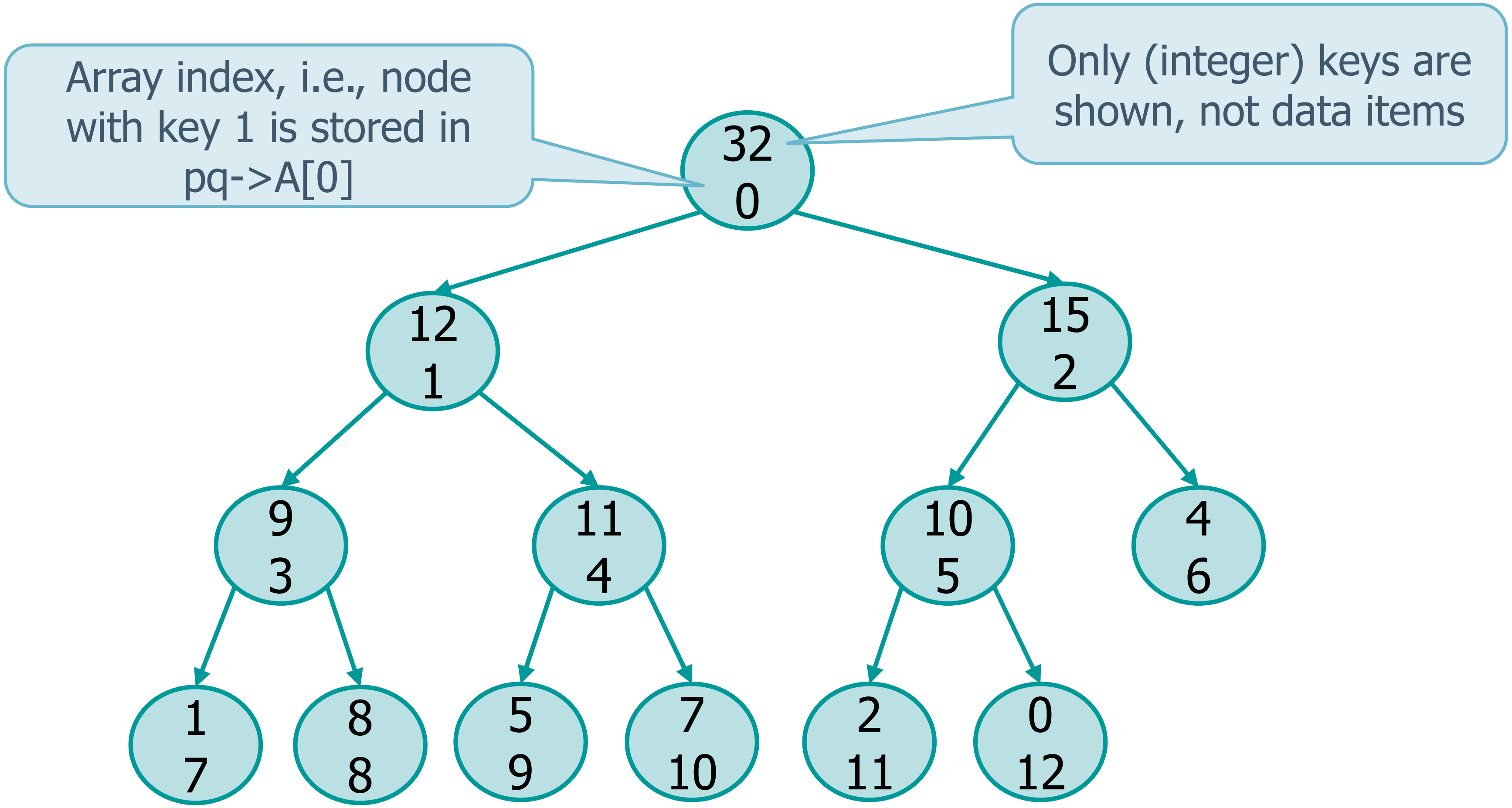

Array index, i.e., node with key 1 is stored in pq->A[0]
Only (integer) keys are shown, not data items
15
0
11
1
10
2
9
3
7
4
5
5
4
6
1
7
8
8
5
9
3
10
2
11
0
12

Example: Increase key

- Given the following max-heap, show the result of
  - pq_change (pq, 5, ((item) 32))



Solution


Array index, i.e., node with key 1 is stored in pq->A[0]
Only (integer) keys are shown, not data items
32
0
12
1
15
2
9
3
11
4
10
5
4
6
1
7
8
8
5
9
7
10
2
11
0
12

Implementation

```
void pq_change (PQ pq, int i, Item item) {
  if (item_less (item, pq->A[i]) {
    decrease_key (pq, i);
  } else {
    increase_key (pq, i, item);
  }
}
void decrease_key (PQ pq, int i) {
  pq->A[i] = item;
  heapify (pq, i);
}
void increase_key (PQ pq, int i) {
  while( (i>=1) &&
         (item_less(pq->A[PARENT(i)], item)) ) {
    pq->A[i] = pq->A[PARENT(i)];
    i = PARENT(i);
  }
  pq->A[i] = item;
}
```

Exercise

This is not an application of **heapsort** but of **pq_insert** and **pq_extract_max**

- Consider the following sequence of positive integers and -1 values
  - Each integer corresponds to one insertion
  - Each -1 corresponds to one extraction
- Report the sequence of values as they are stored in the array representing the priority queue (min-heap and max-heap, respectively) at the end of the entire process

$$5 \quad 21 \quad 4 \quad 19 \quad 13 \quad 9 \quad 11 \quad -1 \quad -1 \quad -1$$

$$31 \quad 3 \quad 7 \quad 2 \quad 5 \quad 21 \quad 23 \quad -1 \quad -1 \quad -1$$

Solution

$5 \quad 21 \quad 4 \quad 19 \quad 13 \quad 9 \quad 11 \quad -1 \quad -1 \quad -1$

Min-heap

```
Insert     5:   5
Insert    21:   5 21
Insert     4:   4 21  5
Insert    19:   4 19  5 21
Insert    13:   4 13  5 21 19
Insert     9:   4 13  5 21 19  9
Insert    11:   4 13  5 21 19  9 11
Extract    4:   5 13  9 21 19 11
Extract    5:   9 13 11 21 19
Extract    9:  11 13 19 21
```

$31 \quad 3 \quad 7 \quad 2 \quad 5 \quad 21 \quad 23 \quad -1 \quad -1 \quad -1$

Max-heap

```
Insert    31: 31
Insert     3: 31  3
Insert     7: 31  3  7
Insert     2: 31  3  7  2
Insert     5: 31  5  7  2  3
Insert    21: 31  5 21  2  3  7
Insert    23: 31  5 23  2  3  7 21
Extract   31: 23  5 21  2  3  7
Extract   23: 21  5  7  2  3
Extract   21:  7  5  3  2
```