Data structures and algorithms

Tutorial 10

Amr Keleg

Faculty of Engineering, Ain Shams University

April 29, 2020

Contact: `amr_mohamed@live.com`

Outline

1 Heap

- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array

Outline

1 Heap

- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array

What is a heap?

- A Heap is used to implement a priority queue
- A heap stores data in left-justified balanced binary tree

Outline

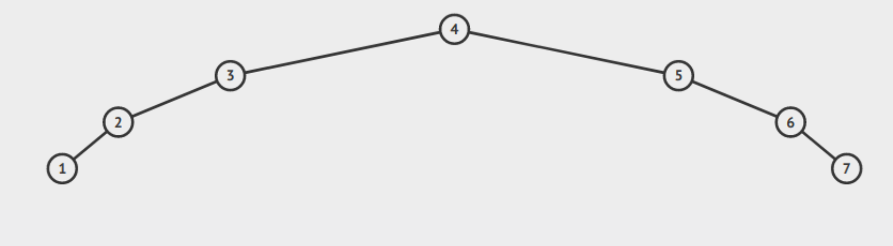
1 Heap

- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array

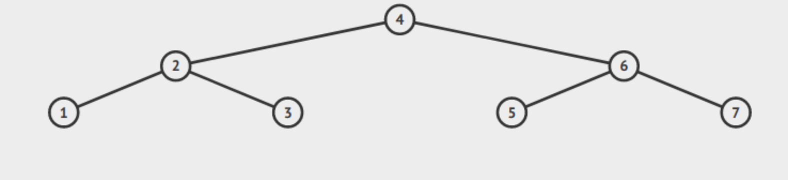
A binary tree is balanced if:

- Both sub-trees are balanced and the height of the two sub-trees differ by at most one.
  (Equivalently)
- All the nodes at depths 0 through n-2 have two children.

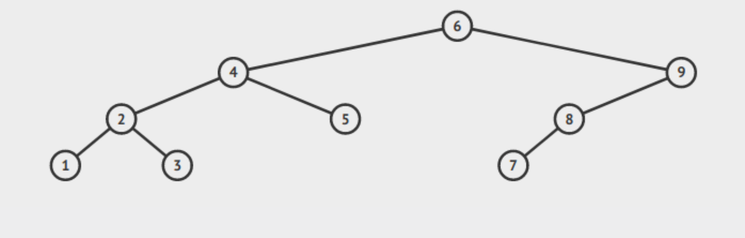
Is this a balanced tree?



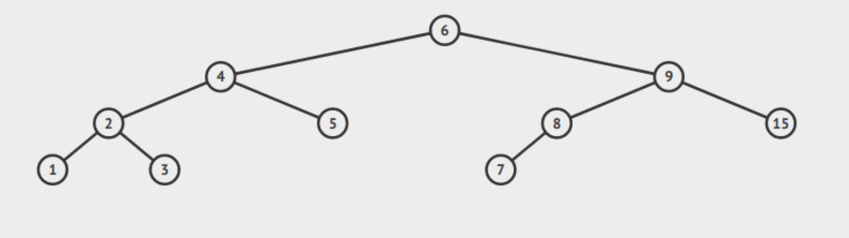
Is this a balanced tree?



Is this a balanced tree?



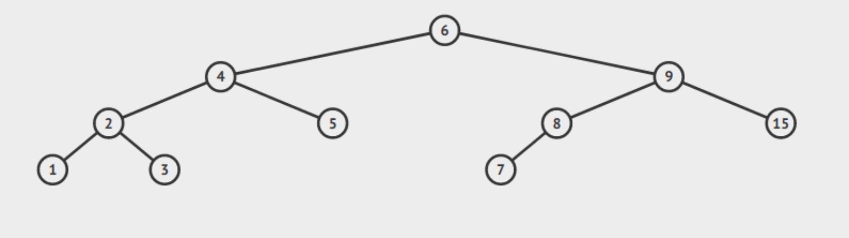
Is this a balanced tree?



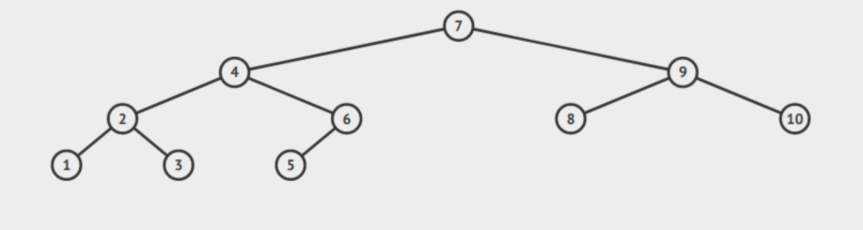
A binary tree is balanced and left-justified if:

- The tree is balanced.
- Leaves are filled in a left to right fashion.

Is this a left-justified balanced binary tree?



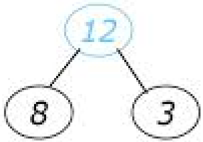
Is this a left-justified balanced binary tree?



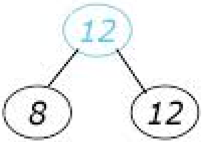
Outline

1 Heap
- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array

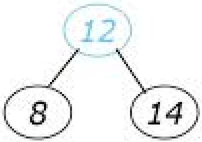
Each and every node in the heap should satisfy the heap property: The value in the node is as large as or larger than the values in its children.



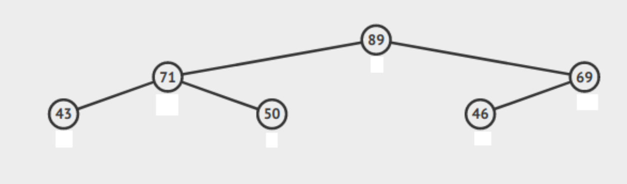
Blue node has heap property



Blue node has heap property



Blue node does not have heap property



Important note: The tree satisfying the heap property isn't a Binary Search Tree!

Outline

1 Heap

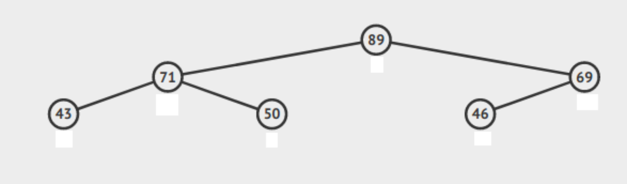
- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array

Operations that a heap should support are:

- Insert a new element to the heap.
- Get the maximum value.
- Delete the maximum value.

How to do the following operations:

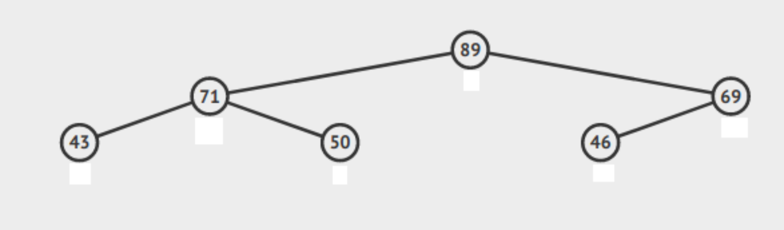
- Insert a new value 100
- Insert a new value 75



How to do the insertion?

- Add the new value as the last leaf.
- Compare it to its parent.
    - If the new value is larger than the parent, swap them and compare it to the new parent (Perform sift-up recursively).
    - else, DONE.

How to delete the top of the tree (the root/ the maximum value)?



Outline

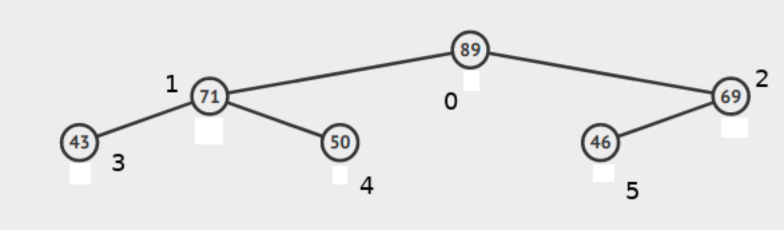
1 Heap
- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array

- Insert a new element to the heap: O(log(n))
- Get the maximum value: O(1)
- Delete the maximum value: O(log(n))

Outline

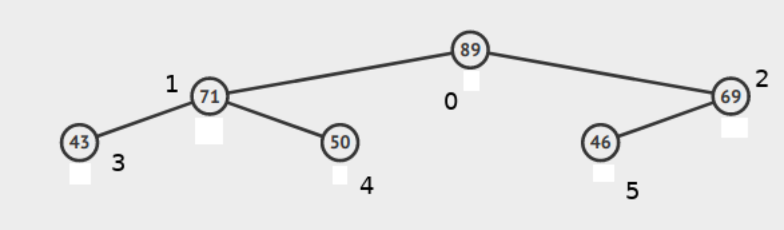
1 Heap

- Definition
- Back to tree definitions
- Heap property
- Heap operation
- Complexity of the heap operations
- Mapping a heap into an array



For node of index i:

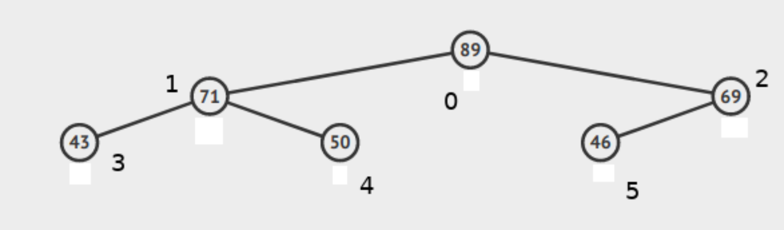
The left node is at index



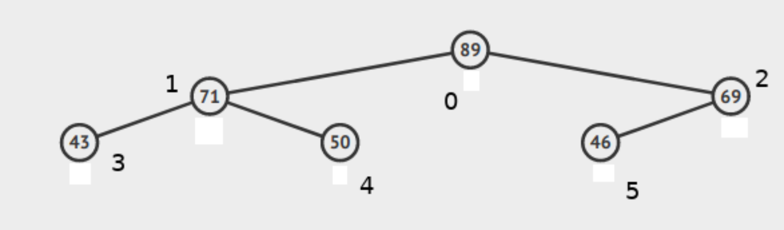
For node of index i:

The left node is at index (2*i) + 1

The right node is at index



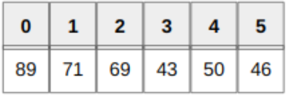
For node of index i:
The left node is at index (2*i) + 1
The right node is at index (2*i) + 2



For node of index i:

The left node is at index (2*i) + 1

The right node is at index (2*i) + 2



0	1	2	3	4	5
89	71	69	43	50	46

- Heap visualization: `https://visualgo.net/en/heap`
- How to do heap sorting?

- Heap visualization: `https://visualgo.net/en/heap`
- How to do heap sorting?

- Insert all the values in a heap
- Delete the max values one by one and insert them in reverse order

Q1. Write down a C++ code to check whether a given array is heap or not. Hence, Determine whether each of the following arrays heap or not.

```
bool is_heap(int arr[], int arr_size);
bool is_heap(int arr[], int arr_size){
  for(int i=0; i < arr_size; i++){
    if(2*i+1 < arr_size && arr[i] < arr[2*i+1])
      return false;
    if(2*i+2 < arr_size && arr[i] < arr[2*i+2])
      return false;
  }
  return true;
}
```

10	15	8	20	14	17	12	9	13

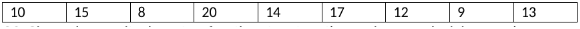
100	65	8	40	34	7	2	9	13

Q3. Given a heap write down two functions to return the maximum and minimum values.

```
class maxHeap{
private:
  int inner_array[1000];
  int cur_size;
public:
  ....
  int get_max();
  int get_min();
};
```

```
int maxHeap::get_max(){
  return inner_array[0];
}
int maxHeap::get_min(){
  int min_val = inner_array[0];
  for(int i=1; i< cur_size; i++)
    min_val = min(min_val, inner_array[1]);
  return min_val;
}
```

Q2. The following integers are stored inside an array, show the array after heapifying it element by element.



10	15	8	20	14	17	12	9	13