Data Structures and Algorithms

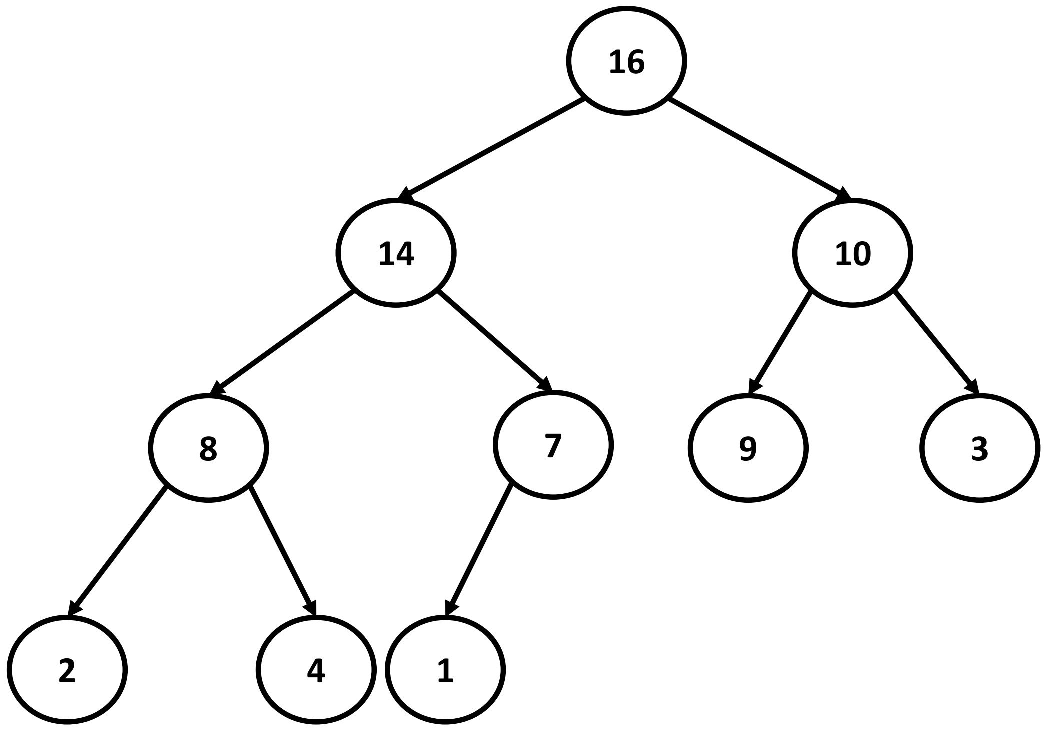
Dr. L. Rajya Lakshmi

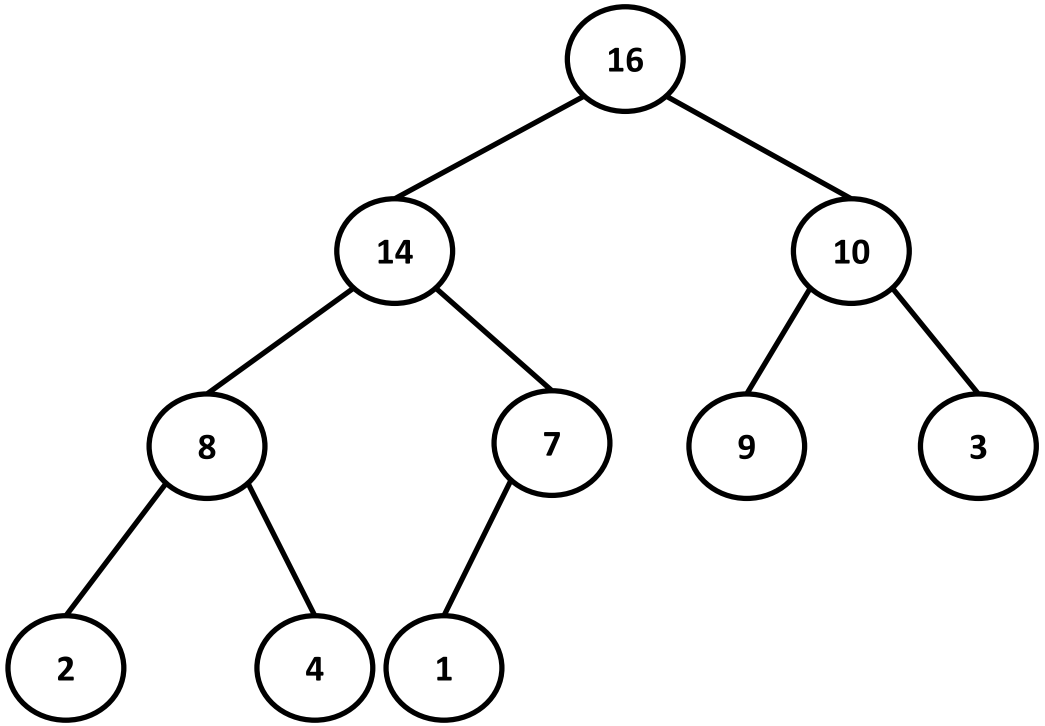
Heapsort

- Runs in O(n log n) time
- An example of in-place sorting algorithm
- A new data structure "heap" is used (different from the heap that we use for garbage collection)

Heapsort

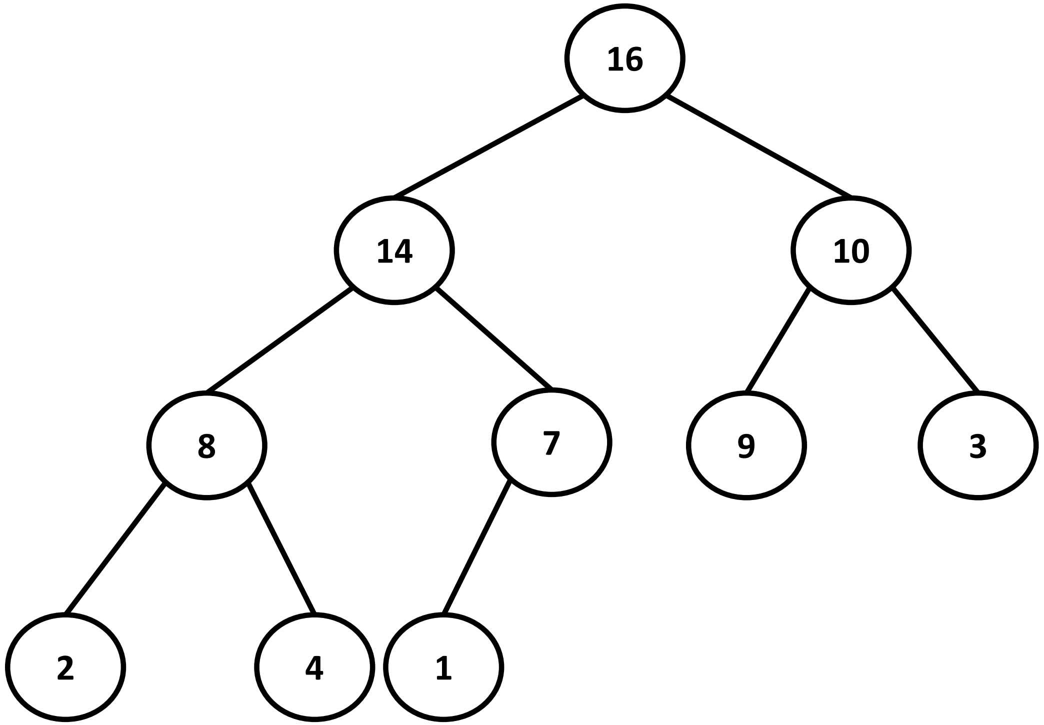
- A binary tree: An ordered rooted tree where each internal node can have at most two children (left child and right child)
- Leaf nodes, level of a node
- A nearly complete binary tree: A binary tree where except for the last level, the other levels are completely filled





Heapsort

- The binary heap data structure is an array object which can be viewed as a nearly complete binary tree



16	14	10	8	7	9	3	2	4	1

Heapsort

- An array A that represents a heap is an object with two attributes:
  - A.length (represents the size of A)
  - A.heap_size (represents the number of elements in the heap that are stored within A)
- Though A[1 . . A.length] may contain numbers, only the elements in A[1 . . A.heap_size], where 0 ≤ A.heap_size ≤ A.length are valid elements of the heap
- A[1] is the root
- Given an index “i“, can easily locate its parent, left and right child

Heapsort

Algorithm Parent(i)

    return$\lfloor i/2 \rfloor$

Algorithm Left(i)

    return 2i

Algorithm Right(i)

    return (2i+1)

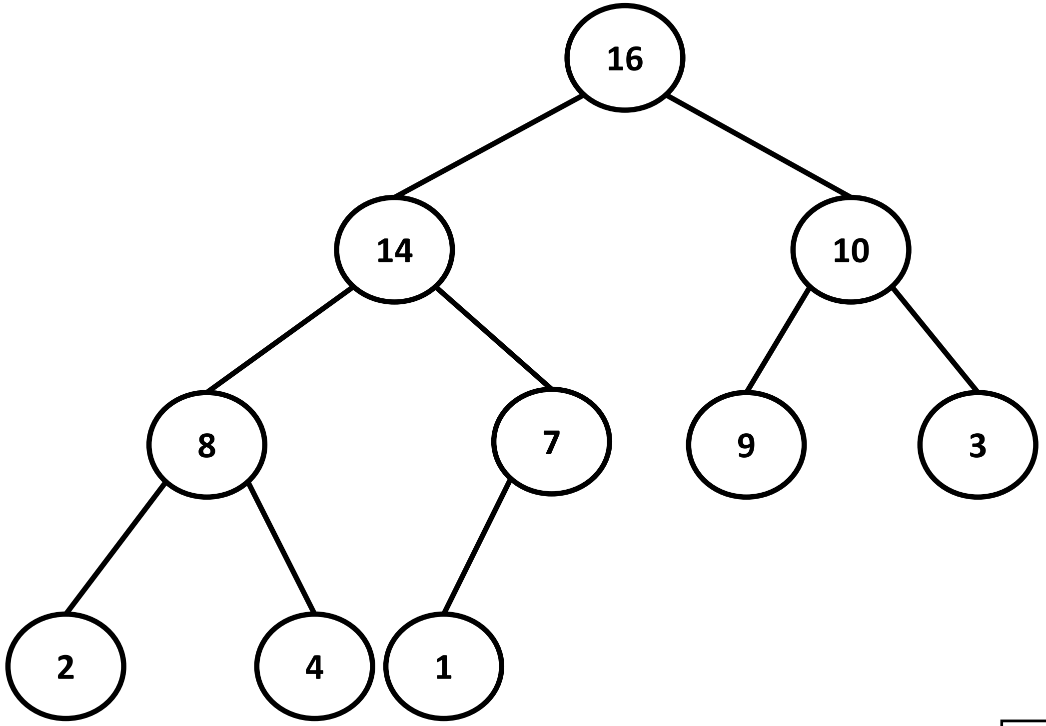
Heapsort

- There are two kinds of binary heaps: max-heaps and min-heaps
- The elements stored in nodes satisfy a heap order property
- These two types of heaps differ in terms of heap order property

Heapsort

- Max-heap-property:
  - For every node “i“ other than the root, A[Parernt(i)] ≥ A[i]
- The maximum elements is stored at the root
- The subtree rooted at a node stores elements no larger than that stored at that node

Heapsort

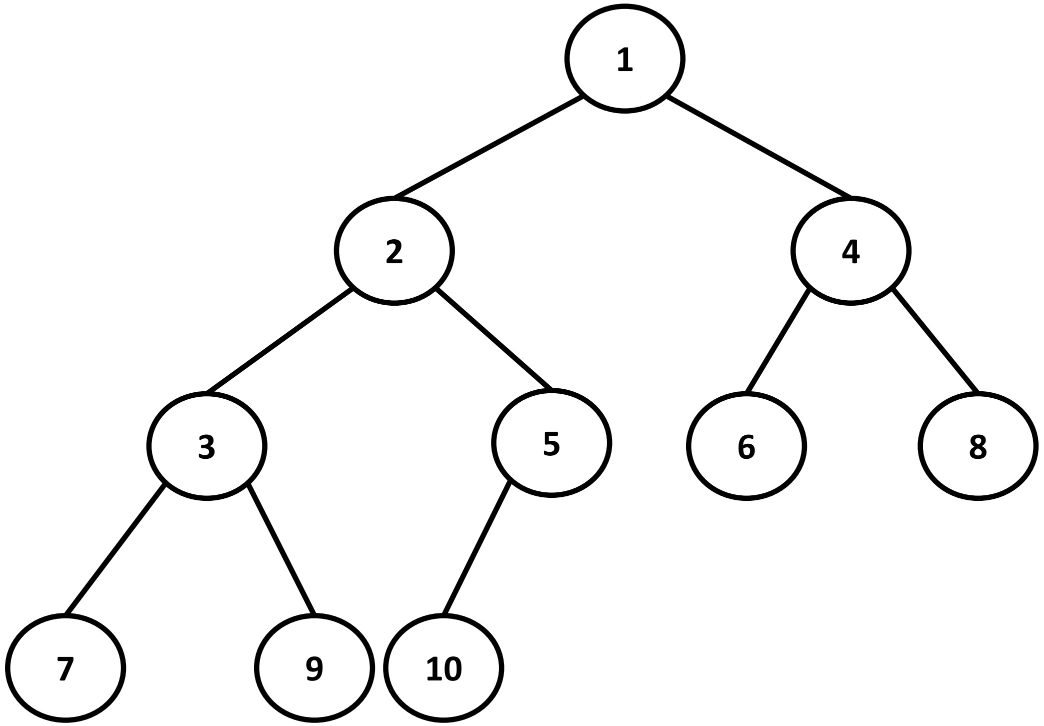


16	14	10	8	7	9	3	2	4	1

Heapsort

- Min-heap-property:
  - For every node “i“ other than the root, A[Parernt(i)] ≤ A[i]
- The minimum elements is stored at the root
- The subtree rooted at a node stores elements no smaller than that stored in that node

Heapsort

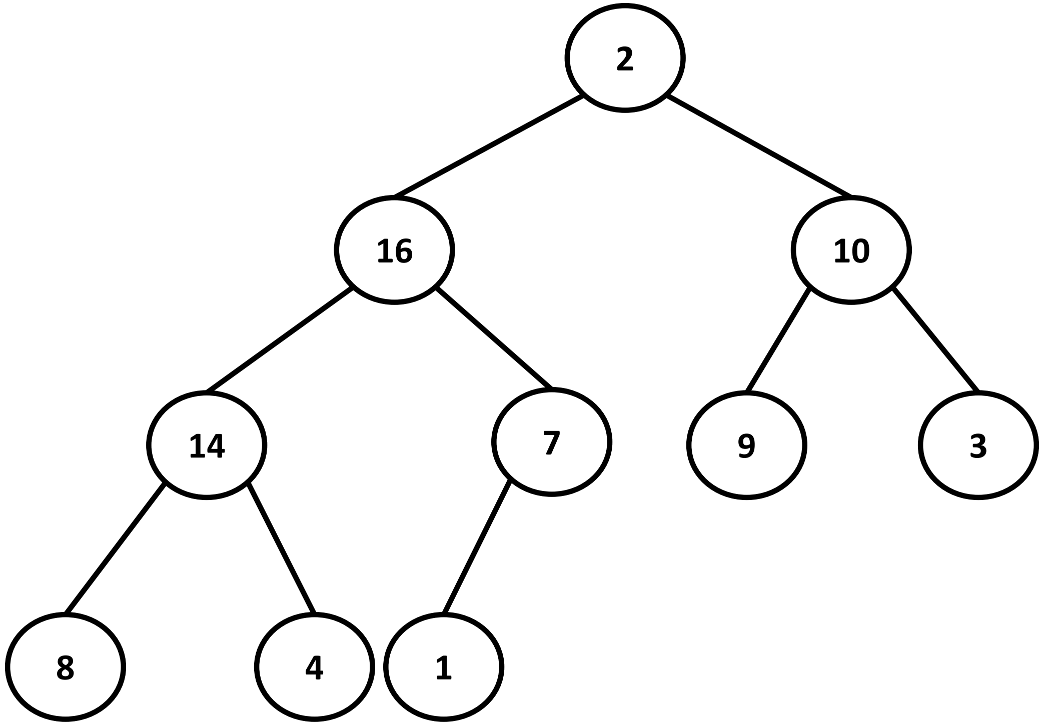


1	2	4	3	5	6	8	7	9	10

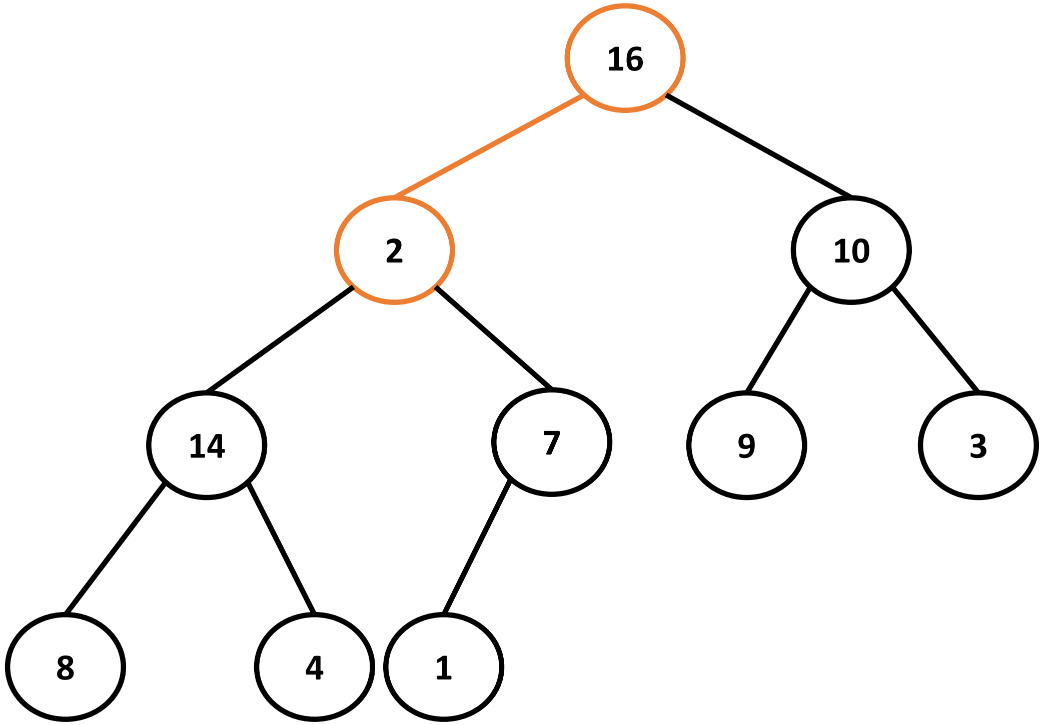
Heapsort

- Height of a node: the length (number of edges) of the longest downward path from that node to a leaf
- Height of the tree: height of the root
- A heap of “n” elements is a nearly complete binary tree, its height is $\theta(\log n)$
- Basic operations on heap run in time proportional to the height of the tree, thus the time complexity of these operations is $O(\log n)$

Heapsort

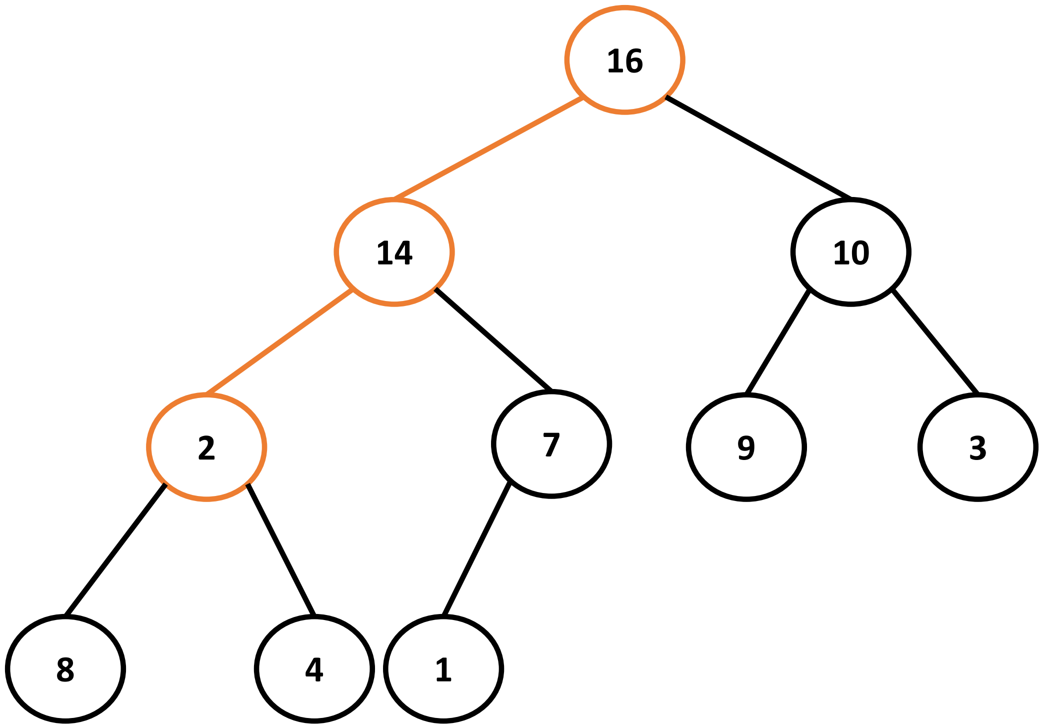


2	16	10	14	7	9	3	8	4	1

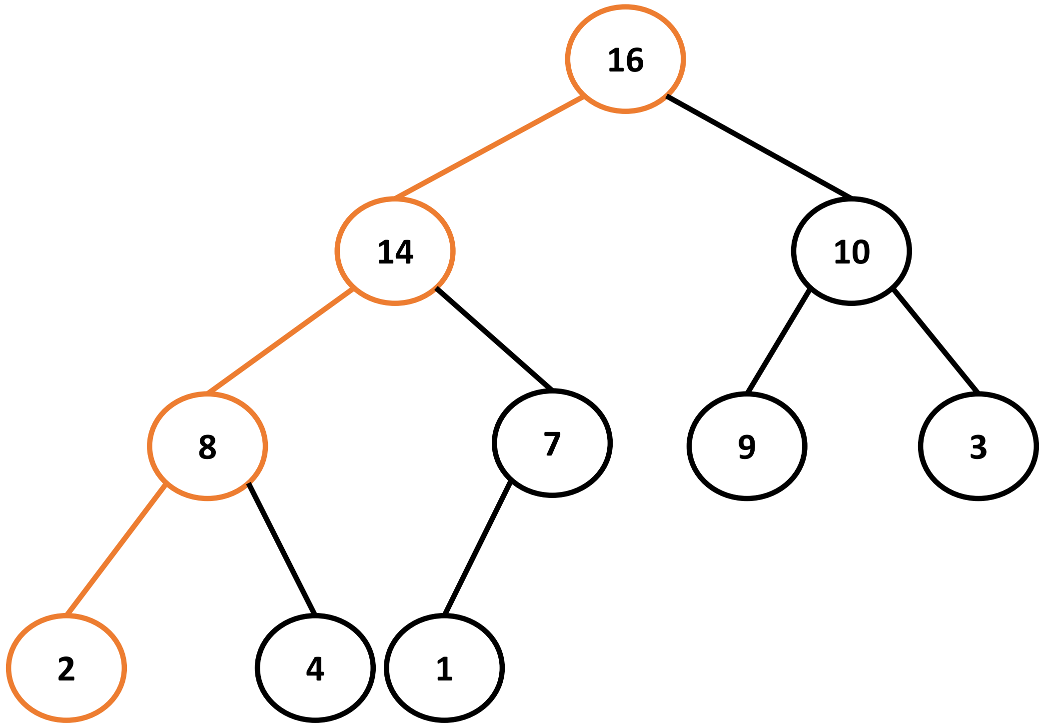


16	2	10	14	7	9	3	8	4	1

Heapsort



16	14	10	2	7	9	3	8	4	1



16	14	10	8	7	9	3	2	4	1

Heapsort

Algorithm Max_Heapify(A, i)

```
l ← Left(i)
r ← Right(i)
if l ≤ A.heap_size and A[l] > A[i]
    largest ← l
else
    largest ← i
if r ≤ A.heap_size and A[r] > A[largest]
    largest ← r
if largest ≠ i
    swap A[i], A[largest]
    Max_Heapify(A, largest)
```

Heapsort

- Max_Heapify assumes that the binary trees rooted at Left(i) and Right(i) are max-heaps
- The running time on a subtree of size n rooted at node i is:
  - Time required to fix up the relationships among the elements A[i], A[Left(i)], A[Right(i)]
  - Time to rum Max_Heapify on a subtree rooted at one of children of node i
- The children's subtree size is at most 2n/3 (the bottom level of the tree is exactly half full)
- $T(n) \le T(2n/3) + \theta(1)$
- The solution to this recurrence is $O(\log n)$