Heap sort

Based on the lecture note “**Heaps and Heap sort**”,
Introduction to Algorithms, MIT

Plan

- Priority Queue
- Heaps
- Algorithm

Priority Queue

- A data structure implementing a set S of elements, each associated with a key, supporting the following operations:
  - insert(S, x): insert element x into set S
  - max(S): return element of S with largest key
  - extractMax(S): return element of S with largest key and remove it from S
  - increaseKey(S, x, k):increase the value of element x’ s key to new value k (assumed to be as large as current value)

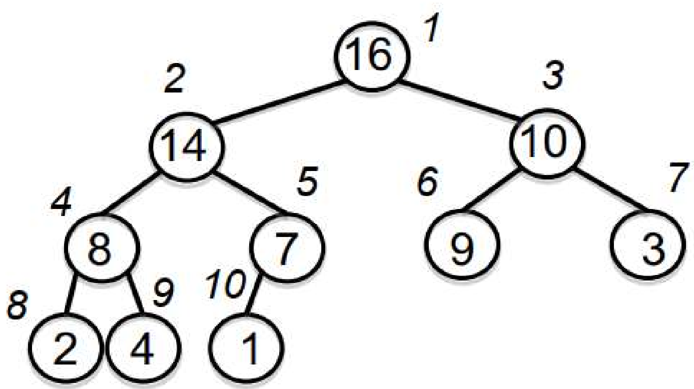
Plan

- Priority Queue
- Heaps
- Algorithm

Heap

- Implementation of a priority queue
- An array, visualized as a nearly complete binary tree
- Max Heap Property: The key of a node is ≥ than the keys of its children

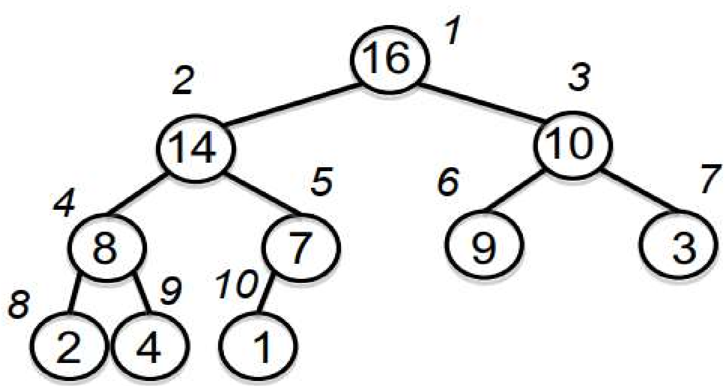
    (Min Heap defined analogously)



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Heap as a tree

- root of tree: first element in the array, corresponding to i = 1
- parent(i) = i/2: returns index of node's parent
- left(i) = 2i: returns index of node's left child
- right(i) = 2i+1: returns index of node's right child



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

→Height of a binary heap is O(logn)

Heap operations

- buildMaxHeap: produce a max-heap from an unordered array
- maxHeapify: correct a single violation of the heap property in a subtree at its root

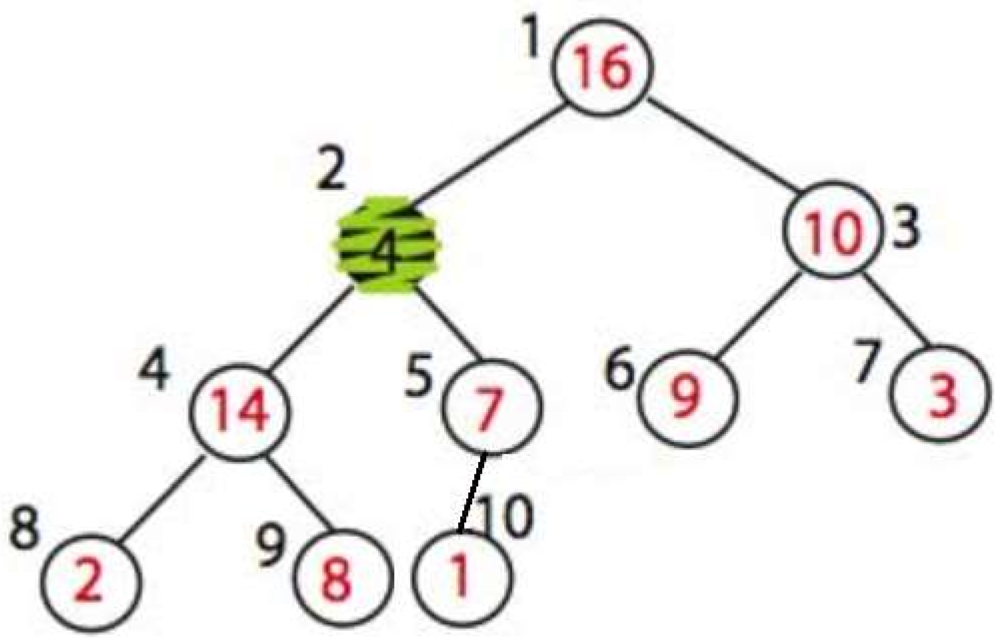
- insert, extractMax, heapSort

maxHeapify

- Assume that the trees rooted at left(i) and right(i) are max-heaps
- If element A[i] violates the max-heap property, correct violation by “trickling” element A[i] down the tree, making the subtree rooted at index i a max-heap

maxHeapify

Example

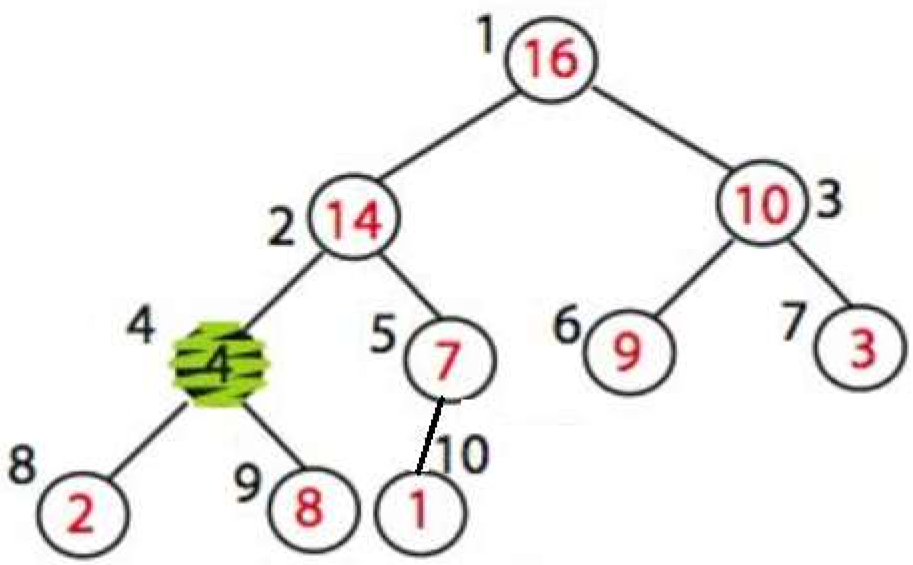


maxHeapify(A, 2)

heapSize(A) = 10

maxHeapify

Example



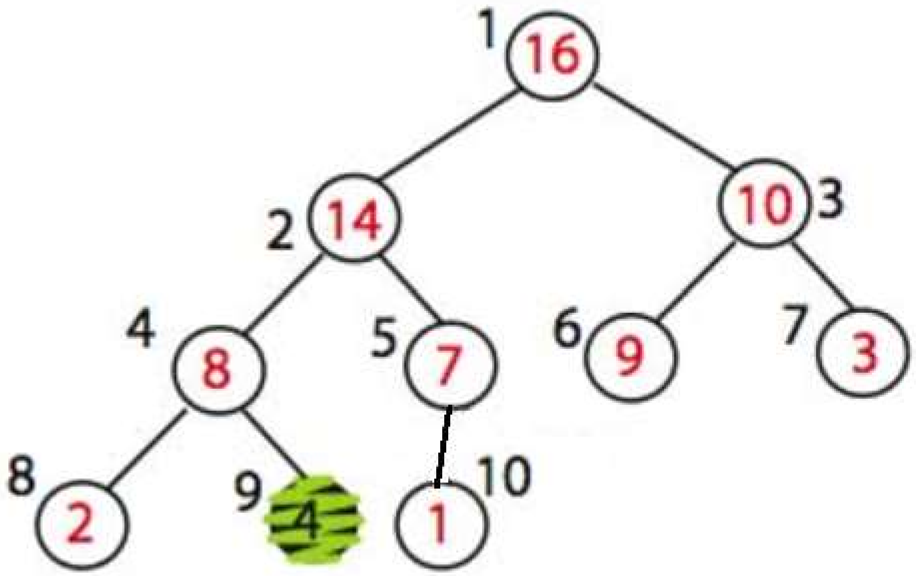
Exchange A[2] with A[4]

Call maxHeapify(A, 4)

because heap max property is violated

maxHeapify

Example



Exchange A[4] with A[9]
No more calls

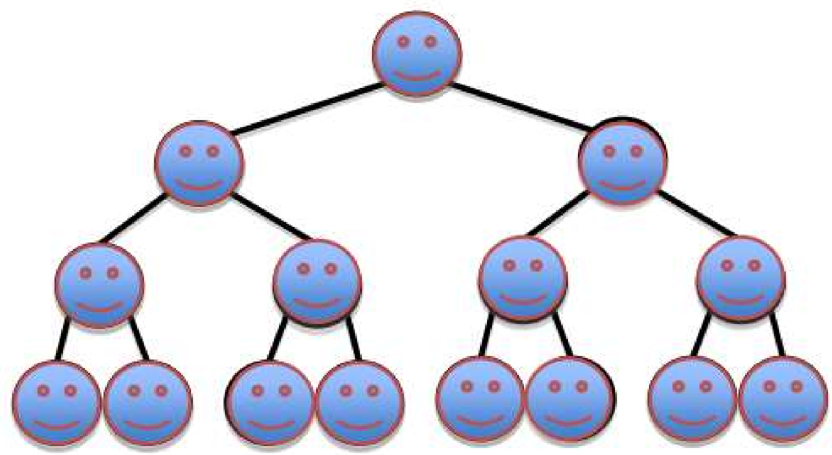
maxHeapify Pseudocode

```
l = left(i)
r = right(i)
if (l <= heap-size(A) and A[l] > A[i]) then
  largest = l
else largest = i

if (r <= heap-size(A) and A[r] > A[largest] then
  largest = r

if largest != i then
  exchange A[i] and A[largest]

maxHeapify(A, largest)
```

buildMaxHeap(A)

Convert A[1..n] to a max heap

```
buildMaxHeap(A)

   for i=n/2 downto 1 do
        maxHeapify(A, i)
```

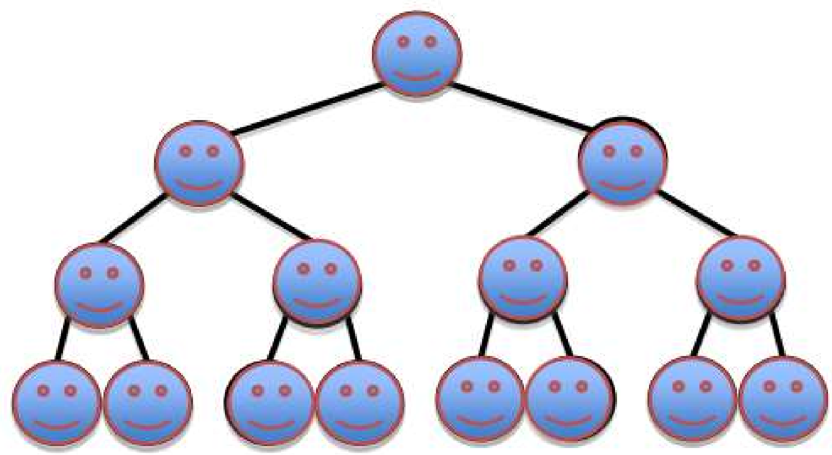


Why start at n/2?

buildMaxHeap(A)

Convert A[1..n] to a max heap

```
buildMaxHeap(A)

   for i=n/2 downto 1 do
        maxHeapify(A, i)
```

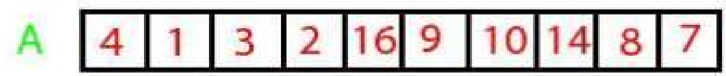


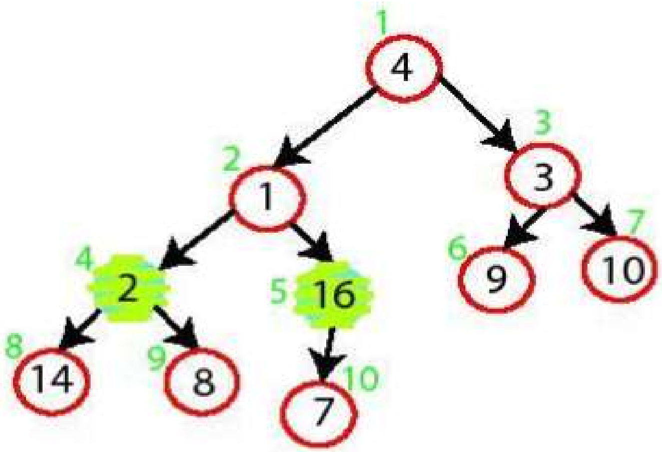
Why start at n/2?

Because elements A[n/2 + 1 … n] are all leaves of the tree
2i > n, for i > n/2 + 1

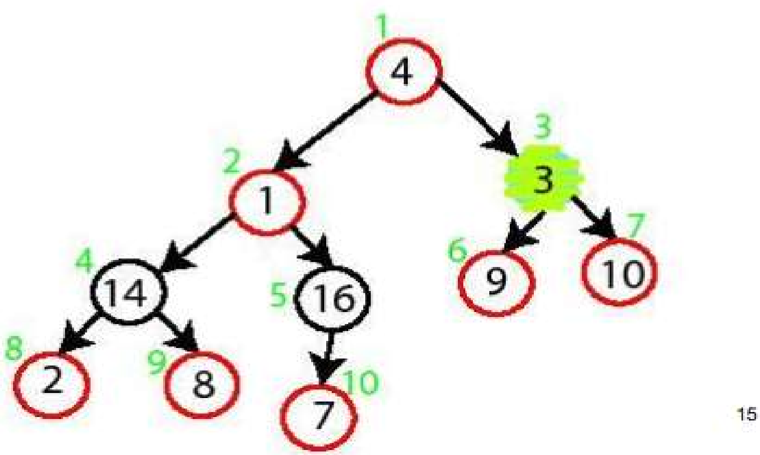
Time=O(n log n)

buildMaxHeap(A) demo



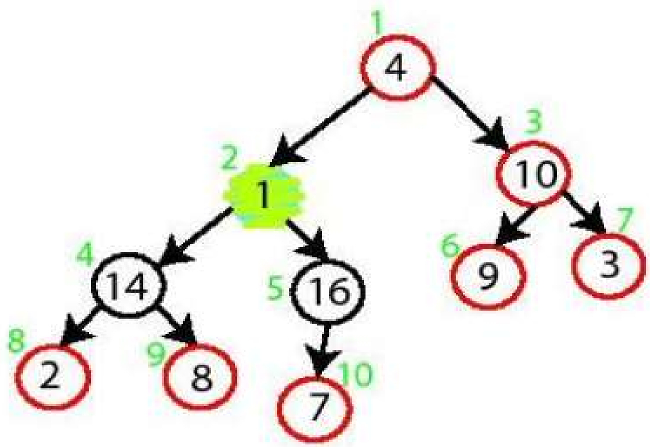


maxHeapify(A, 5)
No change
maxHeapify(A, 4)
Swap A[4] and A[8]

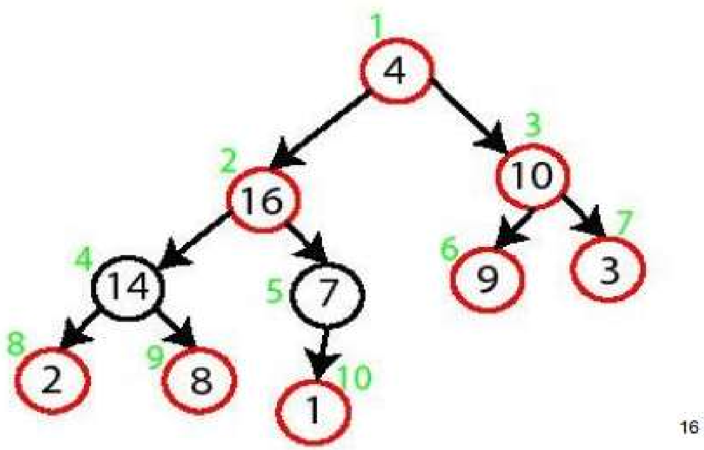


maxHeapify(A, 3)
Swap A[3] and A[7]

buildMaxHeap(A) demo



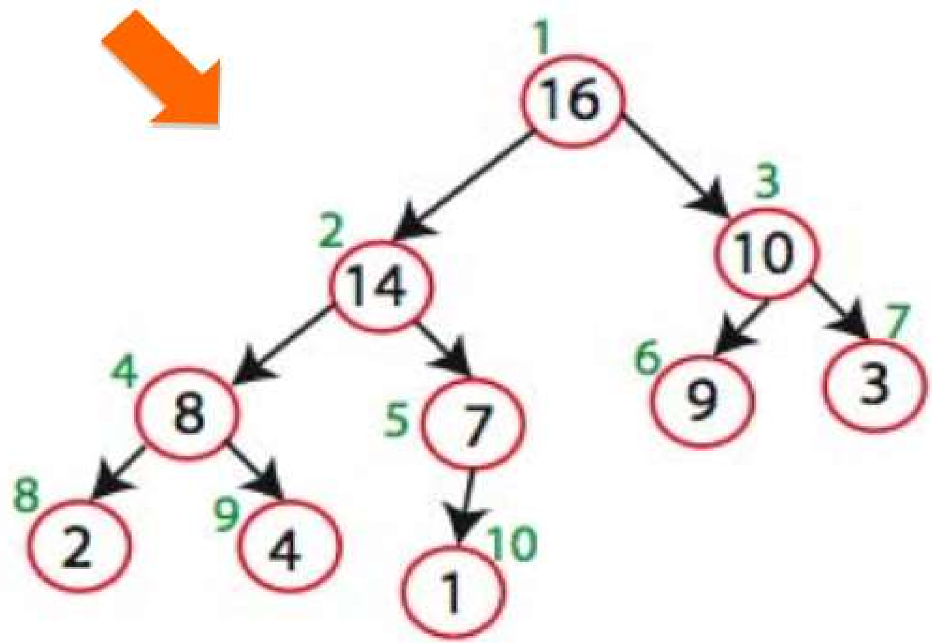
maxHeapify(A, 2)
Swap A[2] and A[5]
Swap A[5] and A[10]



maxHeapify(A, 1)
Swap A[1] and A[2]
Swap A[2] and A[4]
Swap A[4] and A[9]

buildMaxHeap(A)

A	4	1	3	2	16	9	10	14	8	7



Plan

- Priority Queue
- Heaps
- Algorithm

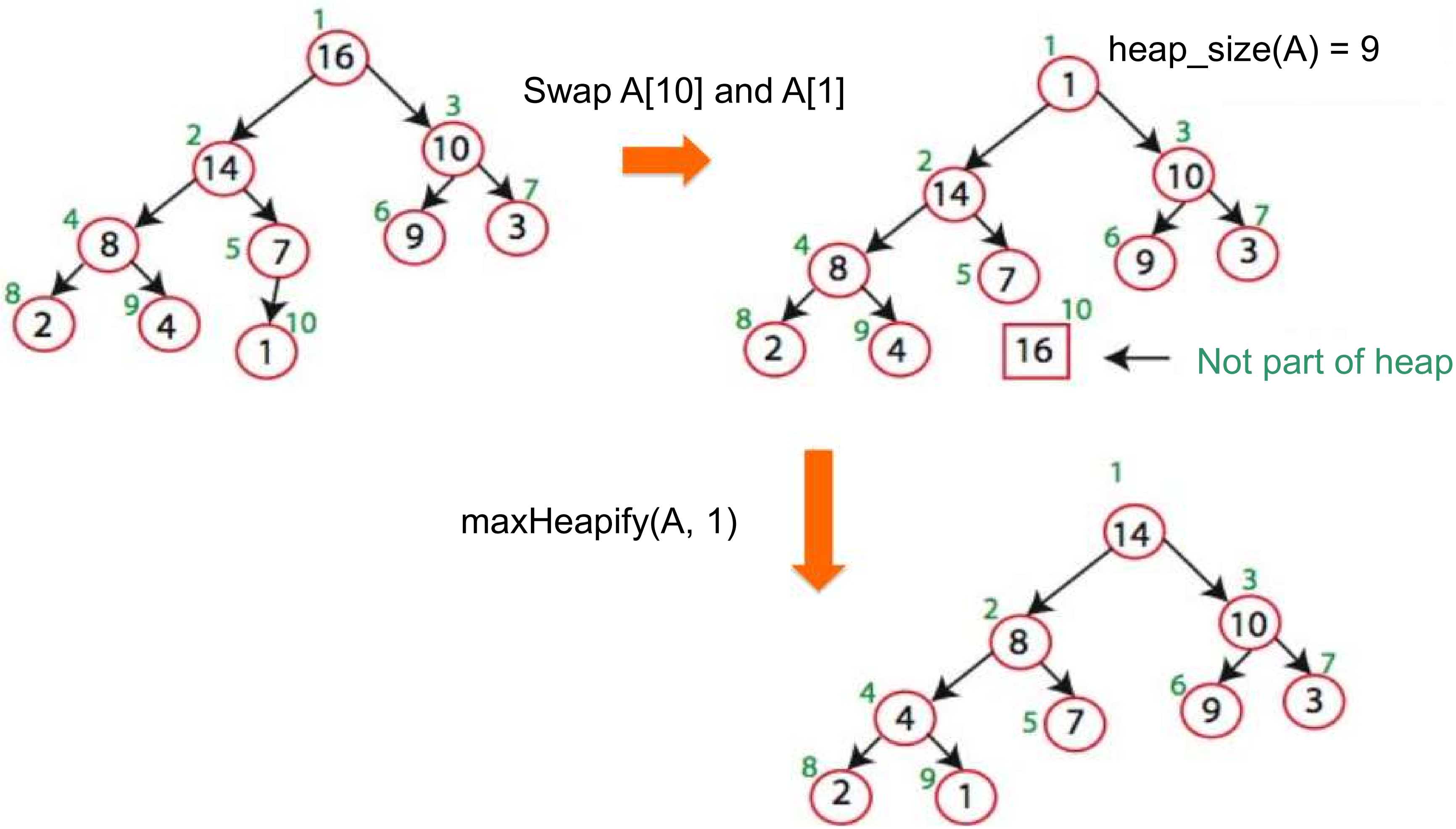
Heap sort

- Build Max Heap from unordered array;
- Find maximum element A[1];
- Swap elements A[n] and A[1]:

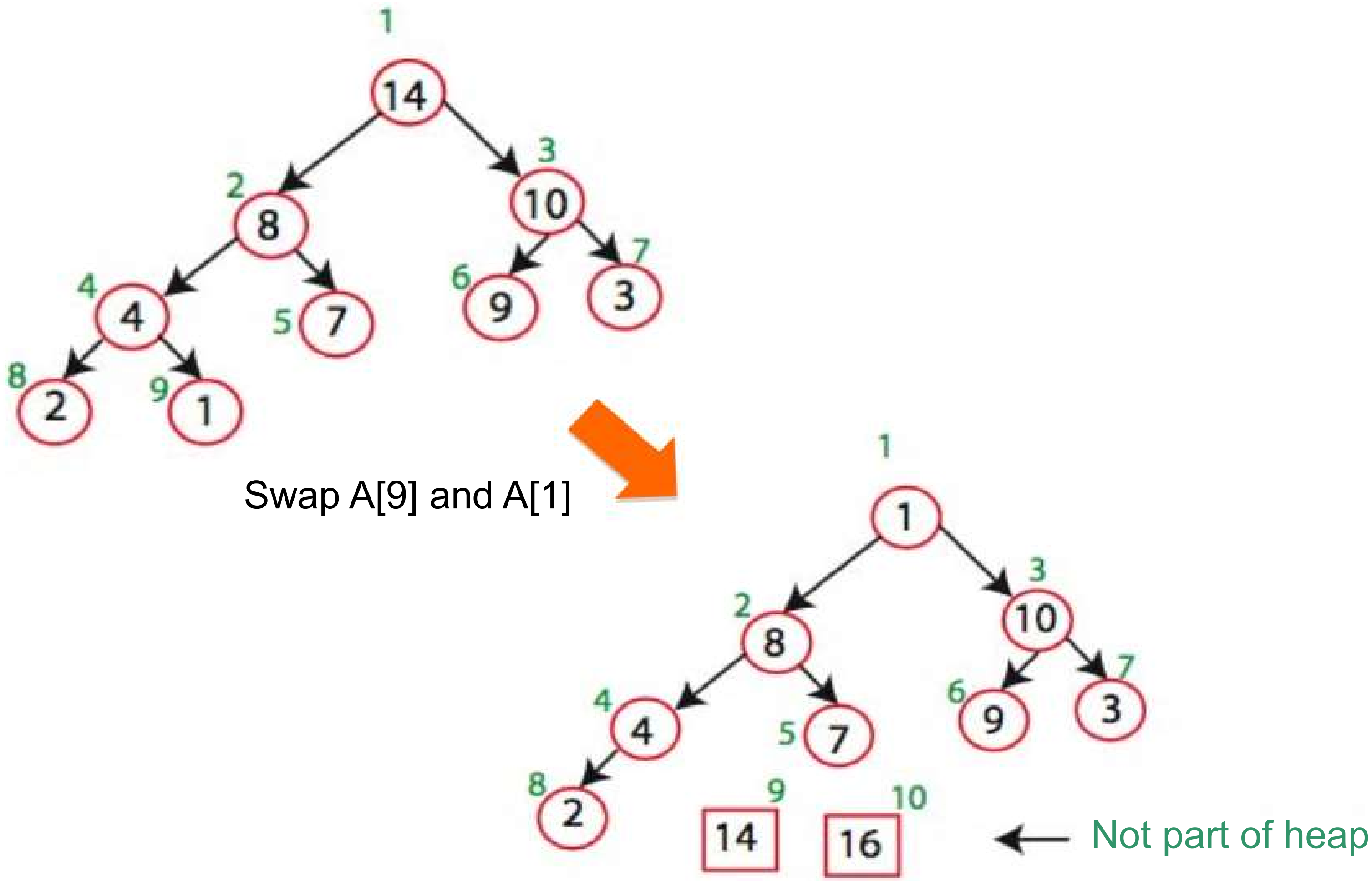
  now max element is at the end of the array!
- Discard node n from heap

  (by decrementing heap-size variable)
- New root may violate max heap property, but its children are max heaps. Run maxHeapify to fix this.
- Go to Step 2 unless heap is empty

Heap sort demo

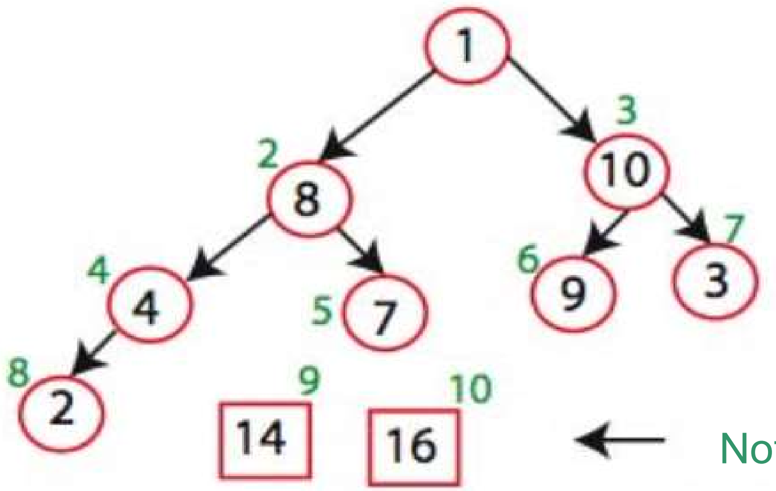


Heap sort demo



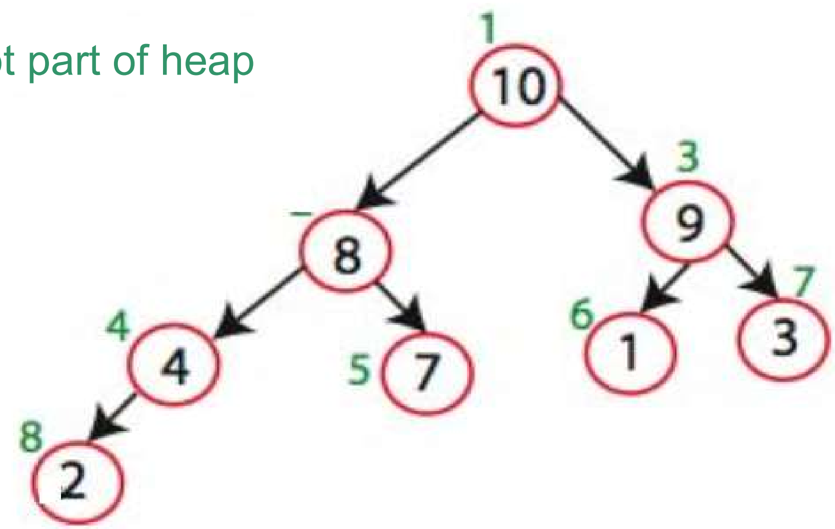
maxHeapify(A, 1)

Heap sort demo

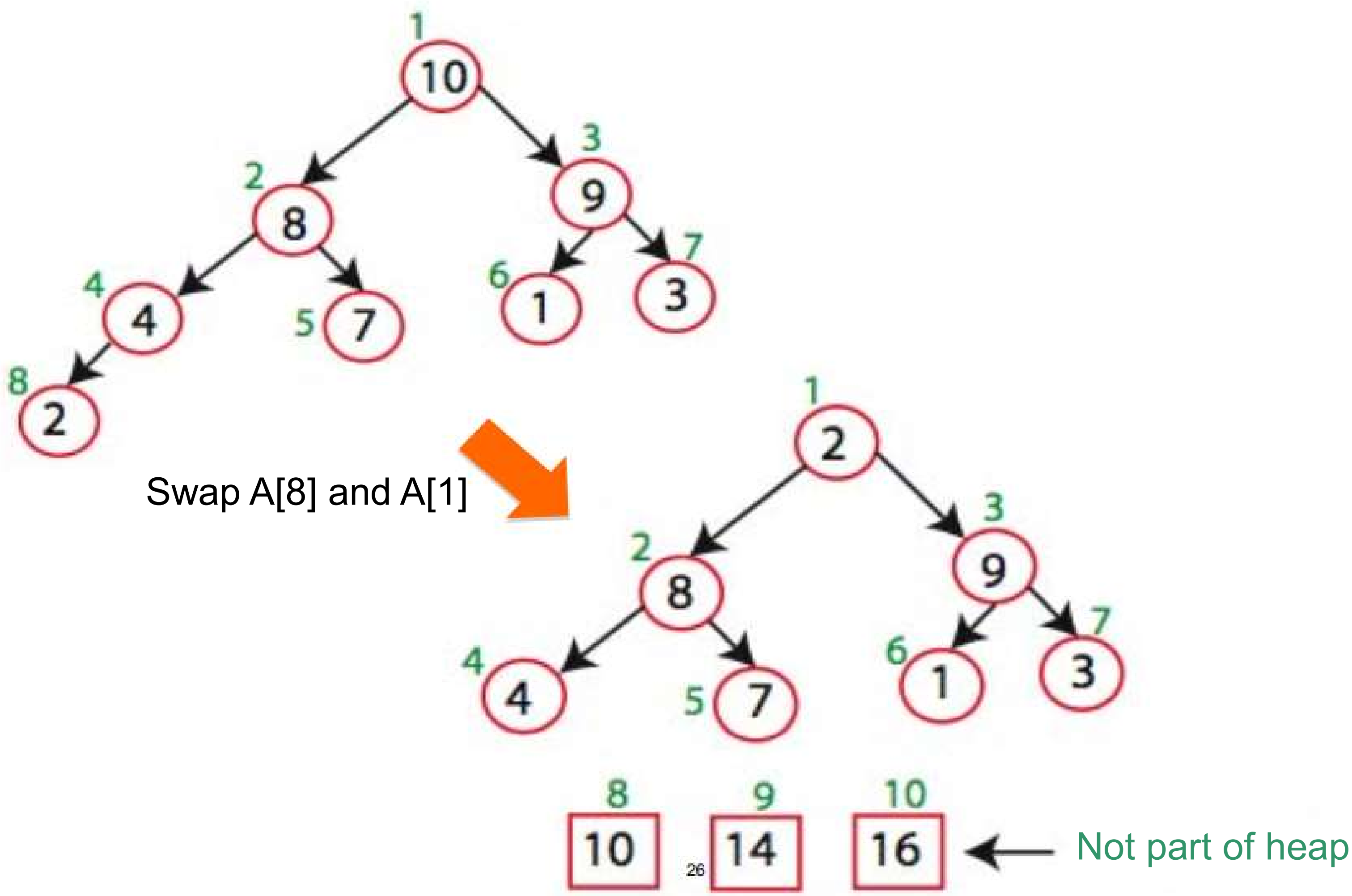


Not part of heap

maxHeapify(A, 1)



Heap sort demo



Heapsort analysis

- Running time:
  - after n iterations the Heap is empty
  - every iteration involves a swap and a maxHeapify operation; hence it takes O(l)
- Overall T(n) = O (nlogn)