Great minds discuss ideas;

Average minds discuss events;

Small minds discuss people.

Learn DAA : From B K Sharma

TCS-503: Design and Analysis of Algorithms

Heapsort

Unit I: Syllabus

- **Introduction:**
  - Algorithms
  - Analysis of Algorithms
  - Growth of Functions
  - Master's Theorem
  - Designing of Algorithms

Unit I: Syllabus

- **Sorting and Order Statistics**
  - Heap Sort
  - Quick Sort
  - Sorting in Linear Time
    - Counting Sort
    - Bucket Sort
    - Radix Sort
  - Medians and Order Statistics

Sorting Revisited

Bubble Sort

Design approach: incremental

Sorts in place: Yes

Running time: $\Theta(n^2)$

Selection sort

Design approach: incremental

Sorts in place: Yes

Running time: $\Theta(n^2)$

Insertion sort

Design approach: incremental

Sorts in place: Yes

Running time: $\Theta(n^2)$

Merge Sort

Design approach: Divide & Conquer

Sorts in place: No

Running time: $\Theta(n \lg n)$

Sorting Revisited

What is the advantage of merge sort?

Answer: good worst-case running time $O(n \lg n)$

Conceptually easy, Divide-and-Conquer

What is the advantage of insertion sort?

Answer: sorts in place-

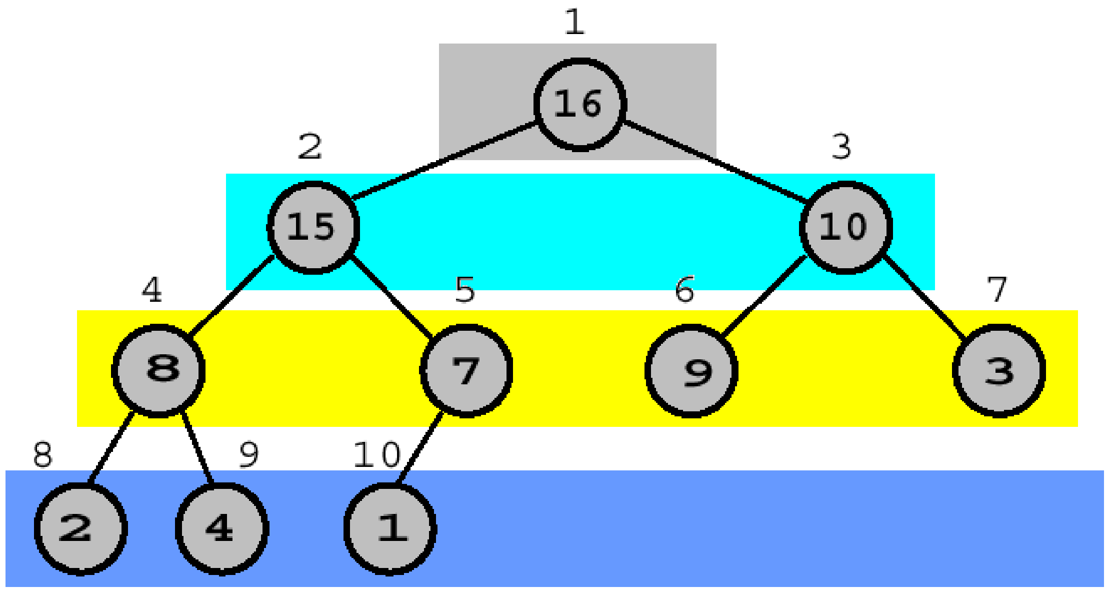
only a constant number of array elements are stored outside the input array at any time.

Next on the agenda:
Heap sort

Combines advantages of both previous algorithms

Learn DAA : From B K Sharma

Heap



1	2	3	4	5	6	7	8	9	10
16	15	10	8	7	9	3	2	4	1

Level: 0 1 2 3

Heap

A heap can be seen as a complete binary tree with the following properties:

If a Node is at position i, *then*

Its left child will be at position 2i

Its right child will be at position 2i+1

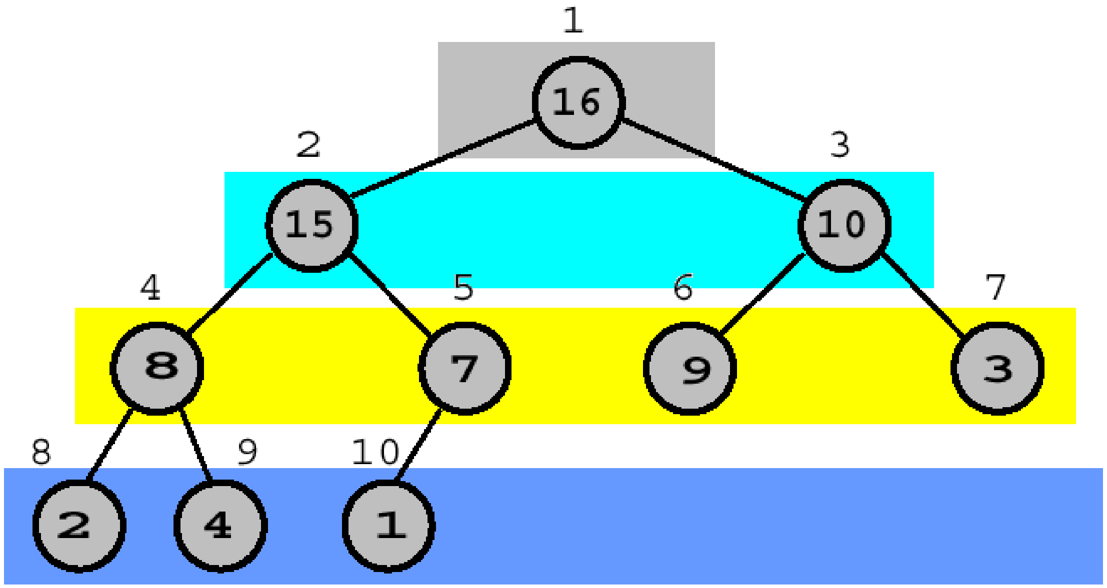
And

Its parent will be at position i/2

Heap Property

Heaps also satisfy the *heap property*:

$A[Parent(i)] \geq A[i]$ for all nodes $i > 1$



Heap Sort Algorithm

has

two supporting algorithms:

MAXHEAPIFY()

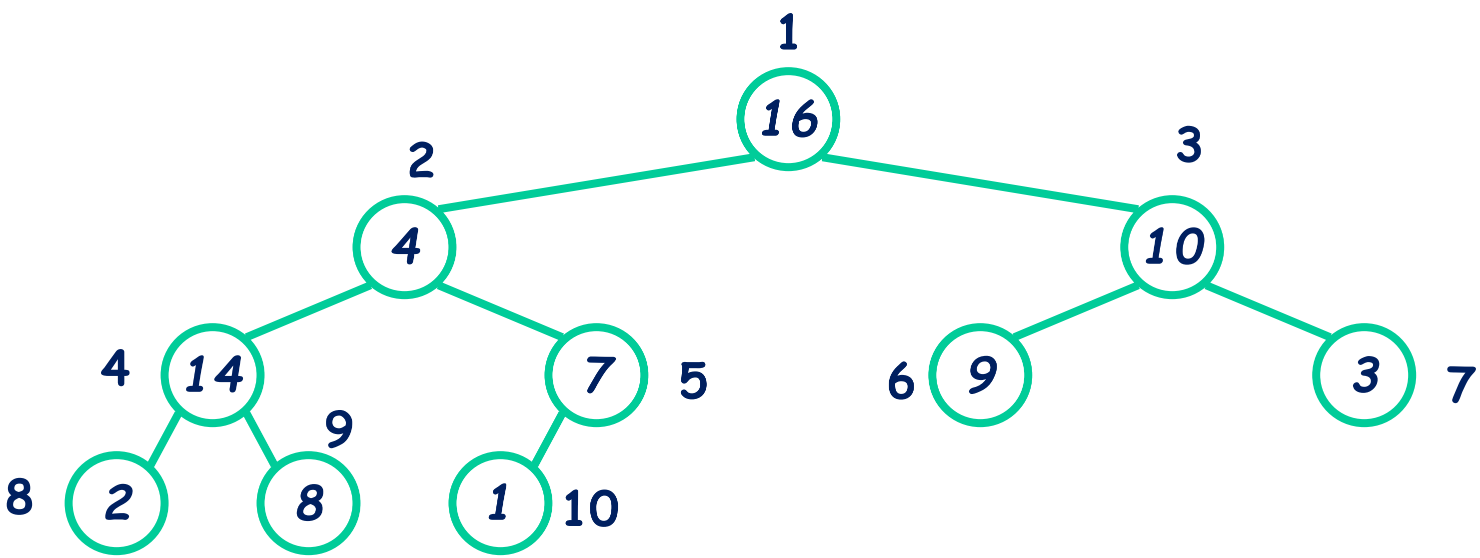
and

BUILD-MAX-HEAP()

MAX-HEAPIFY() Algorithm

```
MAX-HEAPIFY(A, i)
    l=LEFT(i)
    r=RIGHT(i)
    If l ≤ heap-size[A] and A[l] > A[i]   then
                largest ← l
    else
                largest ← r
     If r ≤ heap-size[A] and A[r] > A[largest]  then
                largest ← r
     If largest ≠ i   then
               Exchange A[i] ↔ A[largest]
                MAX-HEAPIFY(A, largest)
```

MAX-HEAPIFY(A, i) Algorithm: Example



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

MAX-HEAPIFY(A,2) Algorithm: Example

l<=Heapsize(A):
2<=10: Yes
And A[l]>A[i]:

14>4: Yes then
largest=4

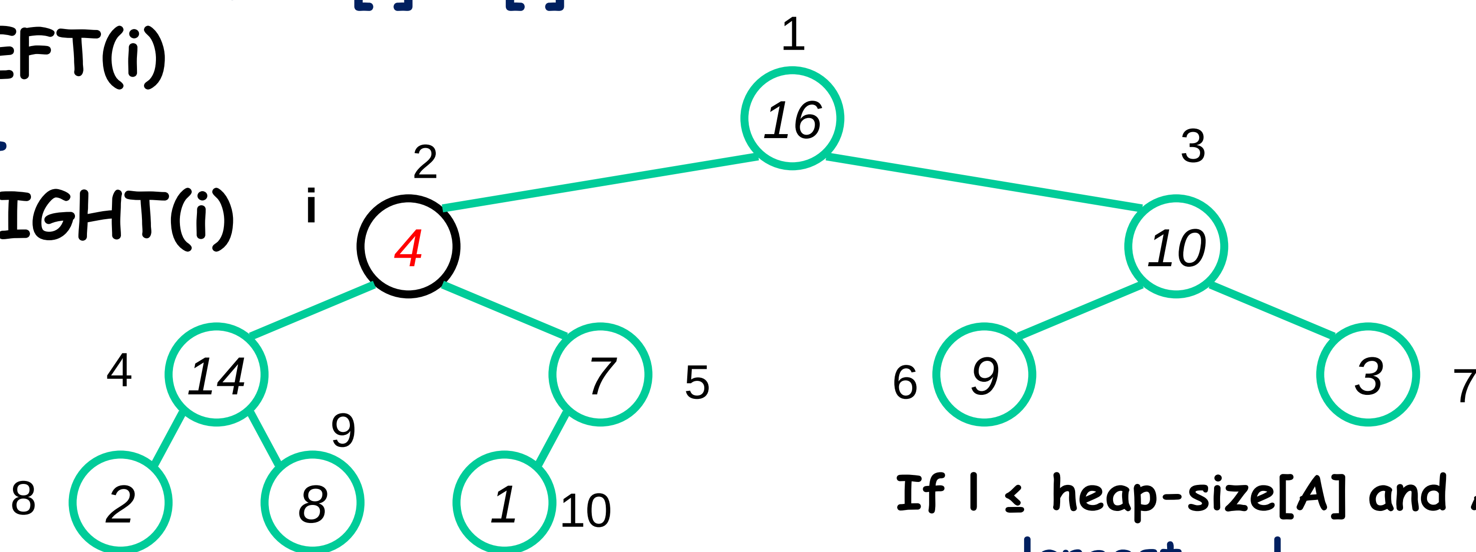
i=2

l=LEFT(i)

l=4

r=RIGHT(i)

r=5



If l ≤ heap-size[A] and A[l] > A[i]
largest ← l
else
largest ← r

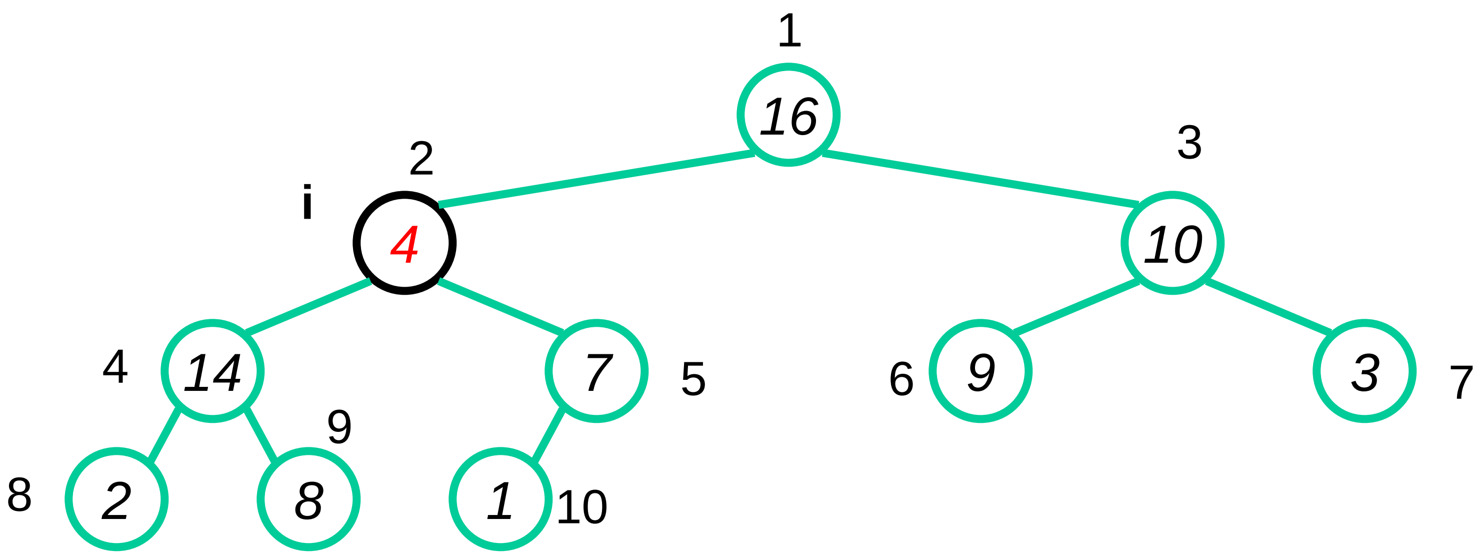
A =	16	4	10	14	7	9	3	2	8	1
	1	2	3	4	5	6	7	8	9	10

MAX-HEAPIFY(A,2) Algorithm: Example

i=2
l=4,
r=5

r<=Heapsize(A):
5<=10: Yes
And A[r]>A[largest]:

7>14: No



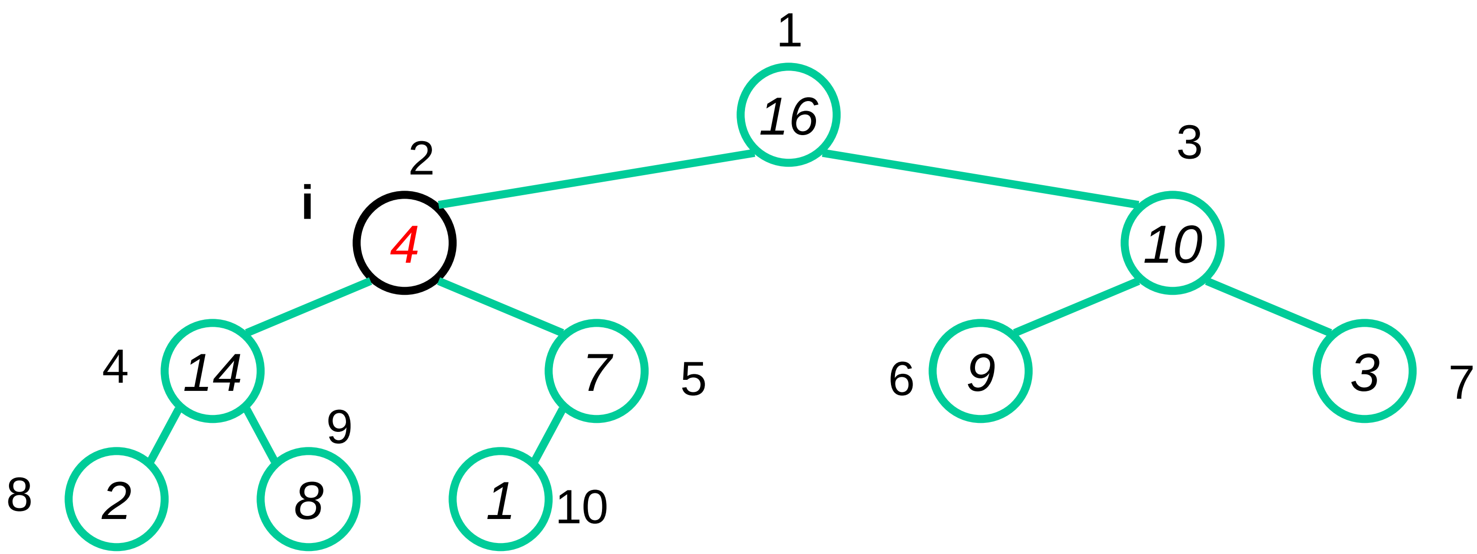
A =	16	4	10	14	7	9	3	2	8	1
	1	2	3	4	5	6	7	8	9	10

MAX-HEAPIFY(A,2) Algorithm: Example

i=2
l=4,
r=5

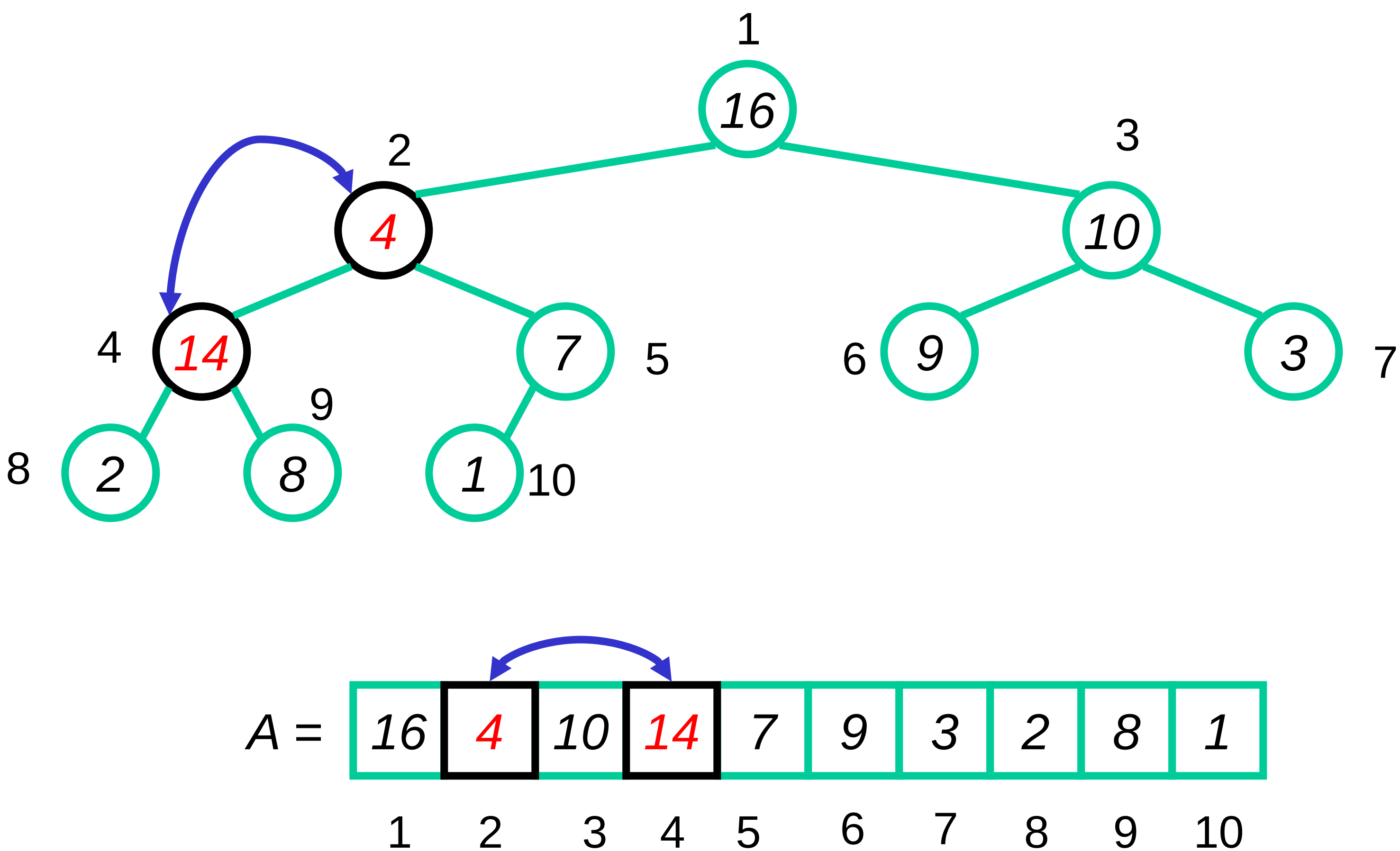
largest ≠ i:
4 ≠ 2: Yes Then
Exchange A[i] and A[largest]:

Exchange A[2] and A[4]



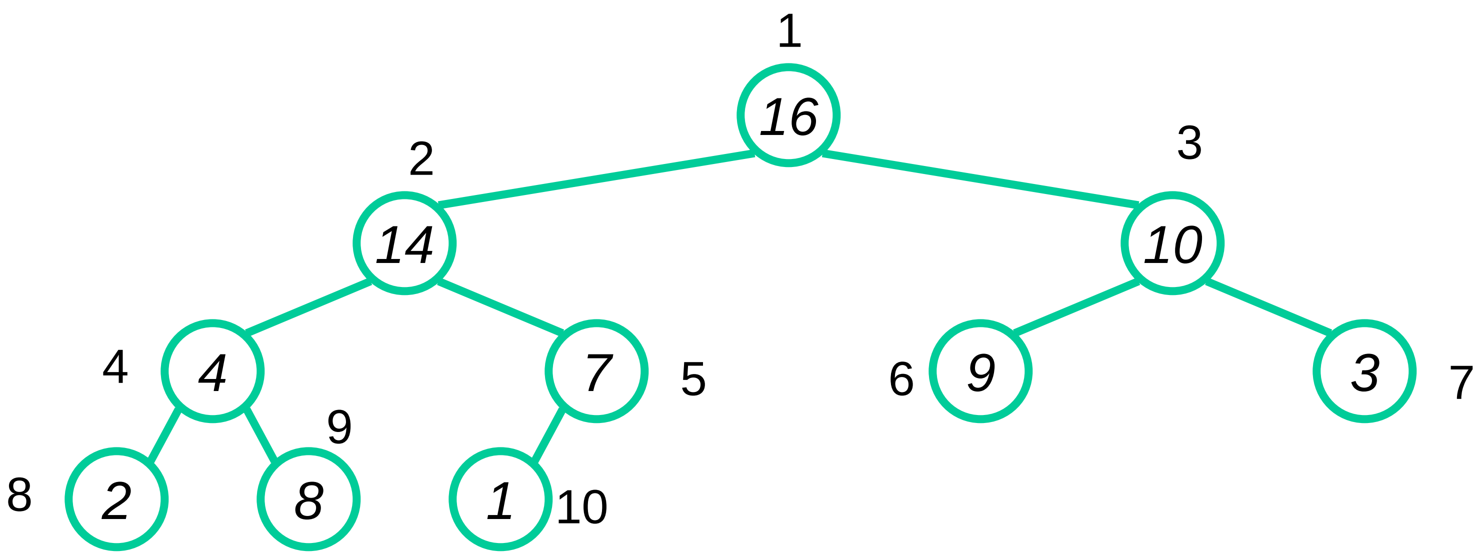
A =										
	16	4	10	14	7	9	3	2	8	1
	1	2	3	4	5	6	7	8	9	10

MAX-HEAPIFY(A,2) Algorithm: Example



MAX-HEAPIFY(A, largest) Algorithm: Example

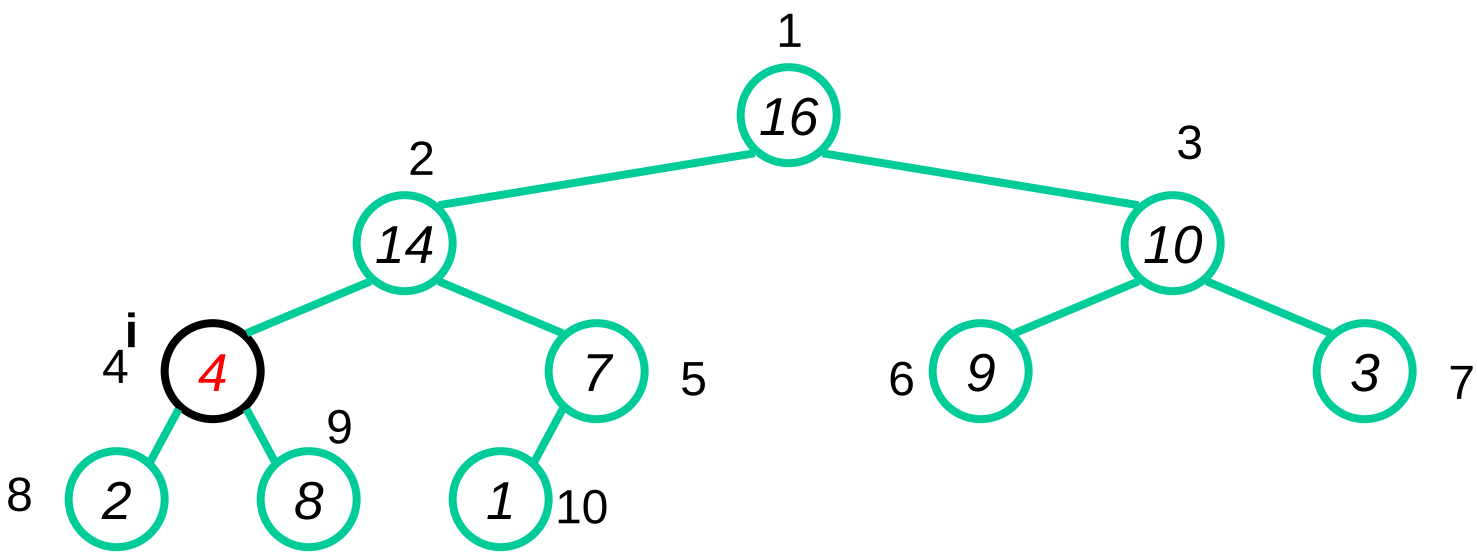
MAX-HEAPIFY(A, 4)



$A =$

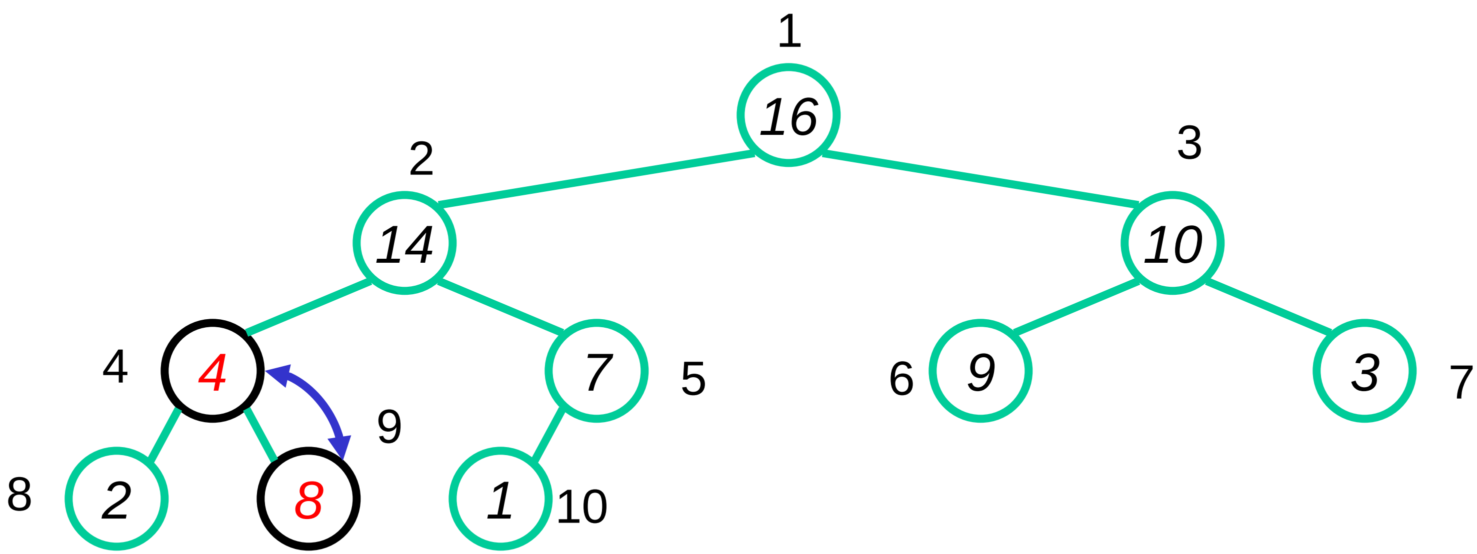
16	14	10	4	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

MAX-HEAPIFY(A, 4)



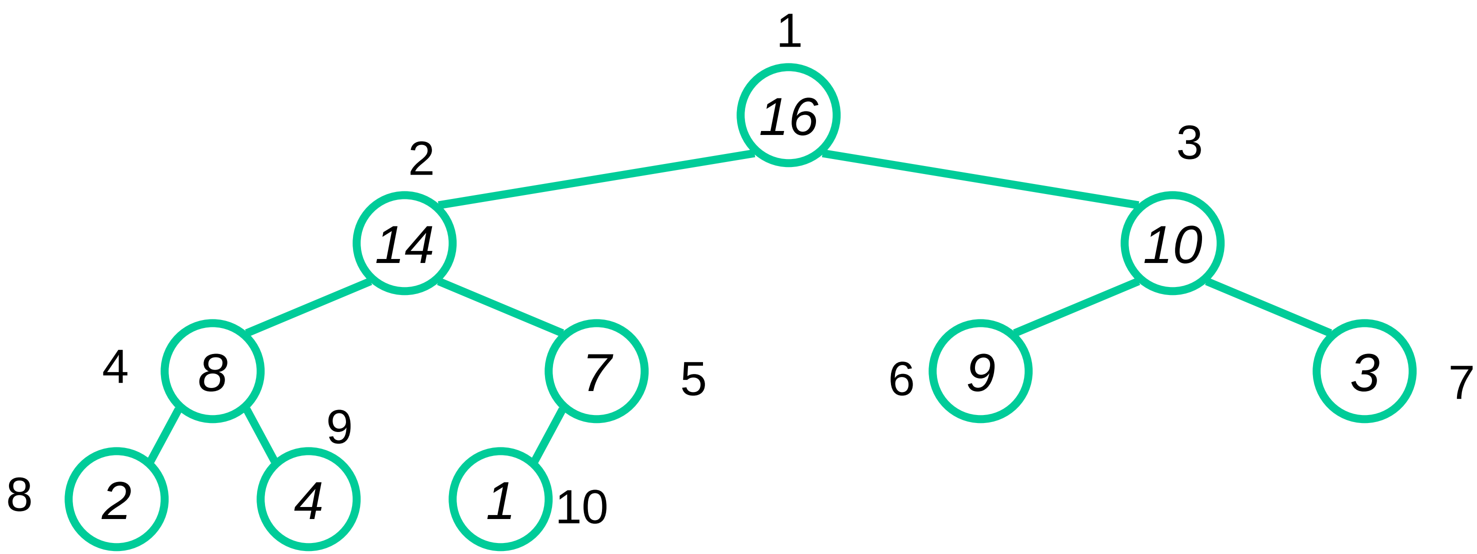
A =	16	14	10	4	7	9	3	2	8	1
	1	2	3	4	5	6	7	8	9	10

MAX-HEAPIFY(A, 4)



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

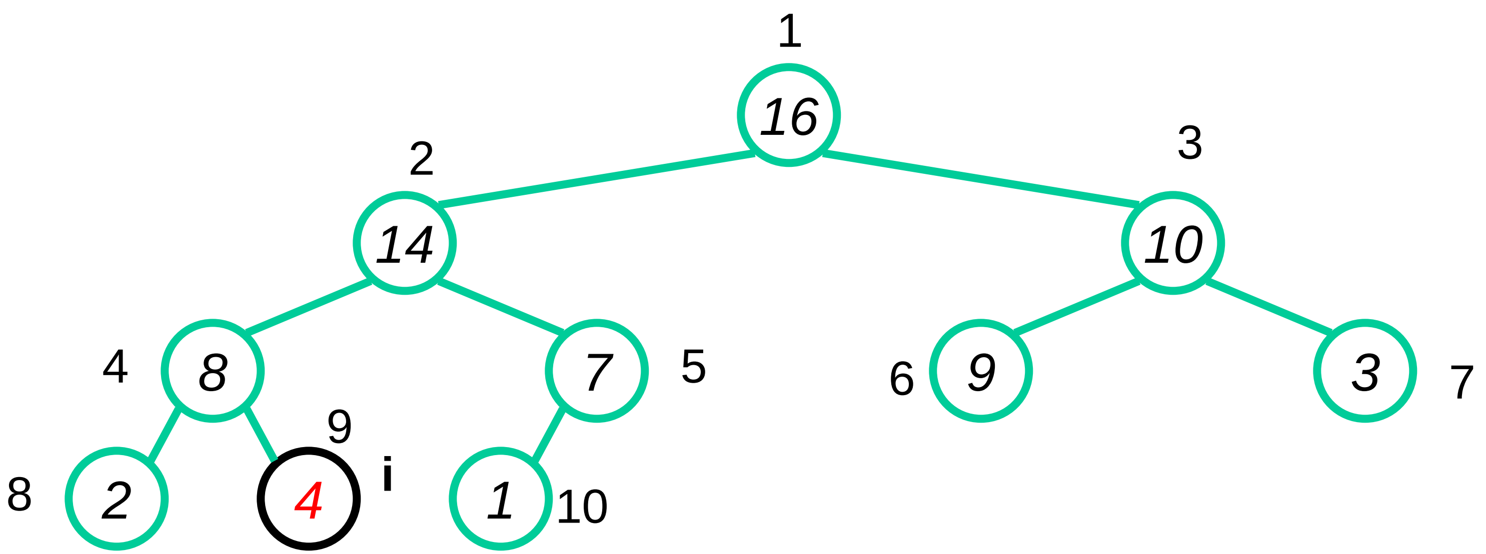
MAX-HEAPIFY(A, 4)



A =

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

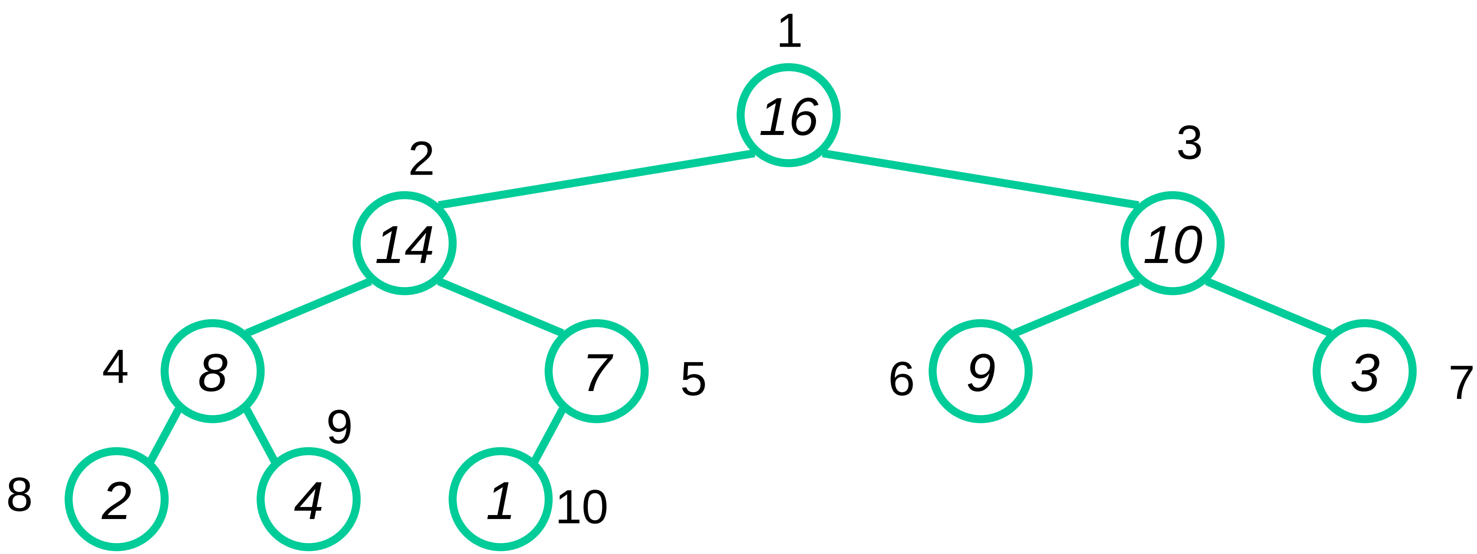
MAX-HEAPIFY(A, 9)



A =

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

MAX-HEAPIFY(A, 9)



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

BUILD-MAX-HEAP()

BUILD-MAX-HEAP(A)

1 heap-size[A]←length[A]

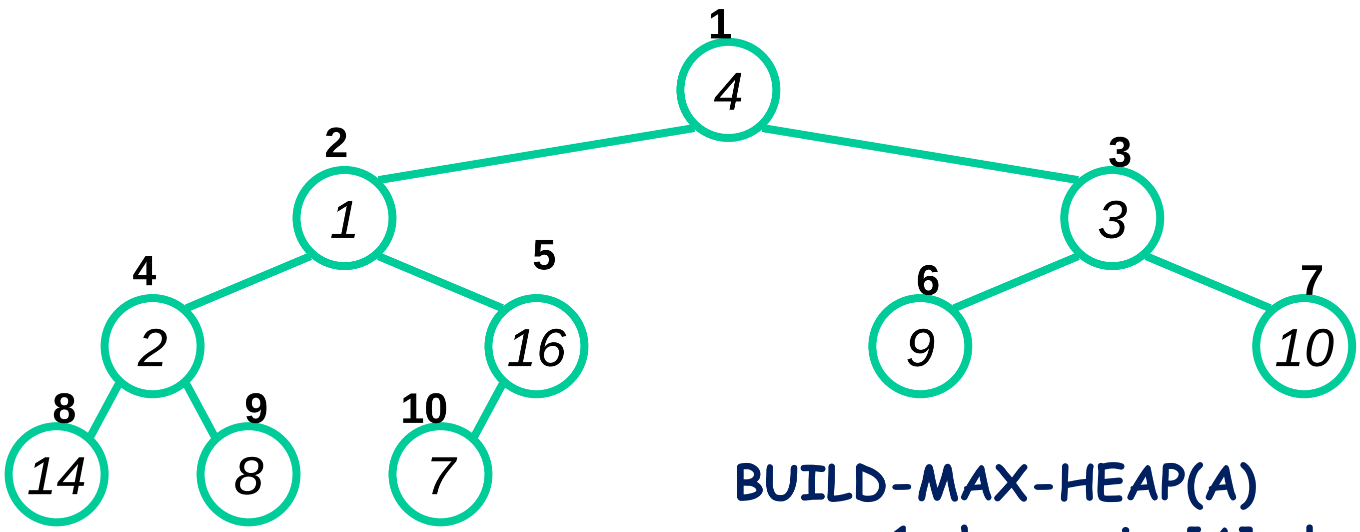
2 For i←⌊length[A]/2⌋ downto 1

3 do MAX-HEAPIFY(A,i)

BUILD-MAX-HEAP() Example

A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7}

n=10, n/2=5



BUILD-MAX-HEAP(A)

1 heap-size[A]←length[A]

2 For i←⌊length[A]/2⌋ downto 1

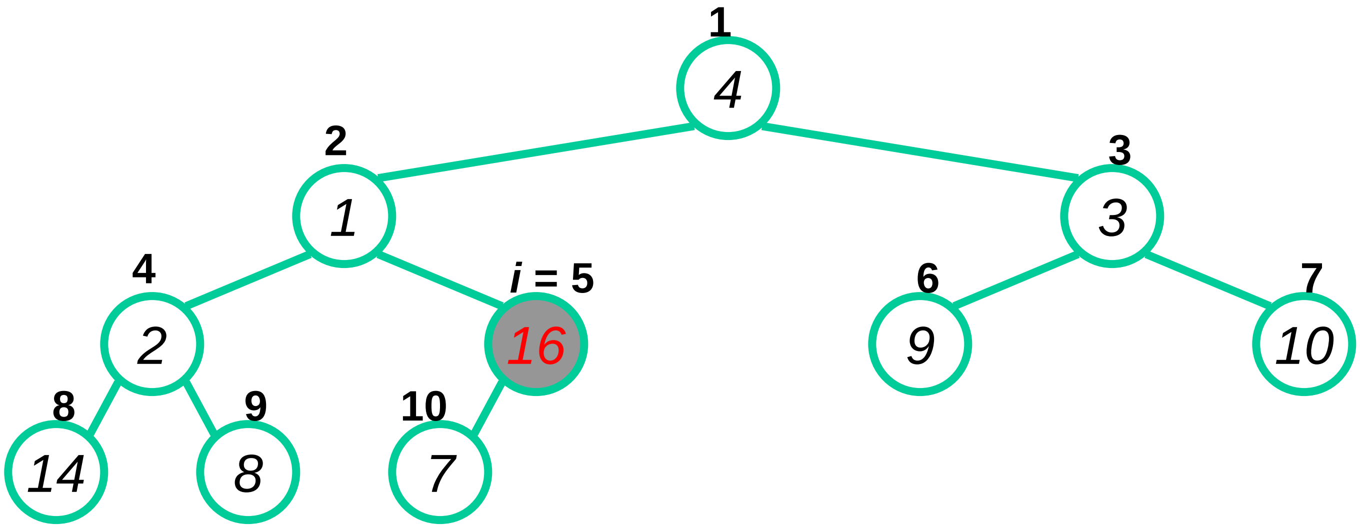
3 do MAX-HEAPIFY(A,i)

BUILD-MAX-HEAP() Example

A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7}

for i=5 to 1

do MAX-HEAPIFY(A,i)

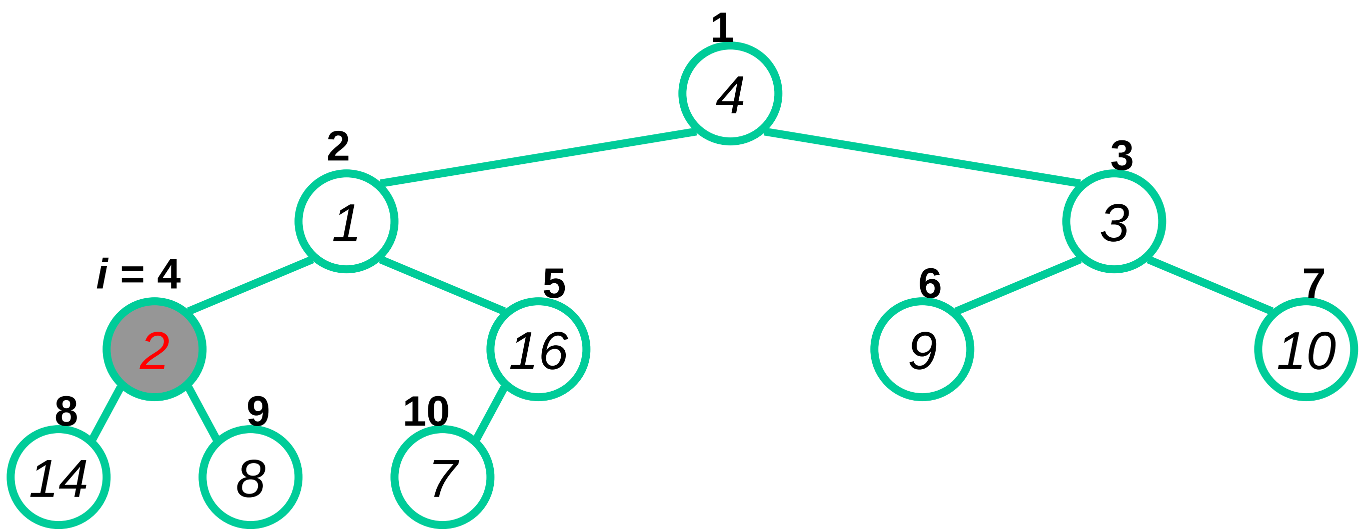


BUILD-MAX-HEAP() Example

A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7}

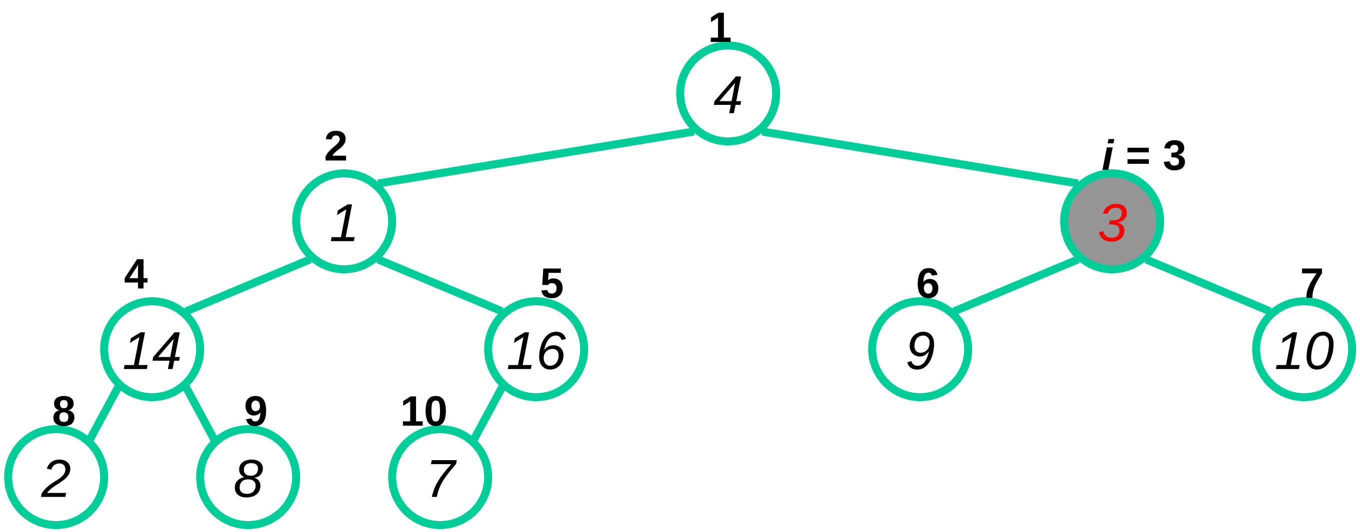
for i=5 to 1

do MAX-HEAPIFY(A,i)



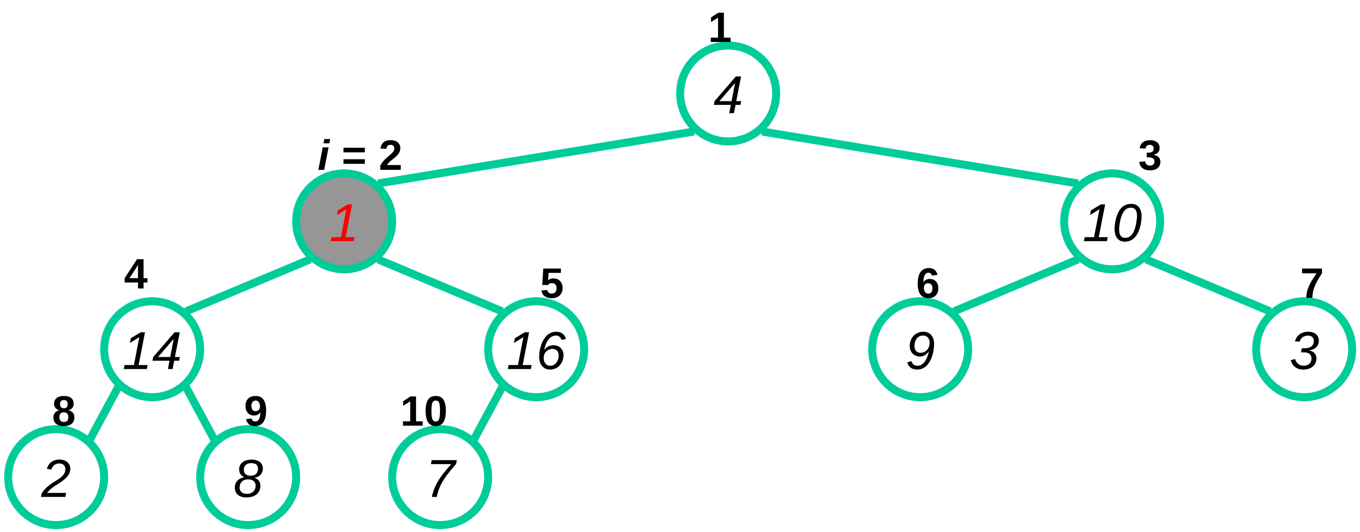
BUILD-MAX-HEAP() Example

A = {4, 1, 3, 14, 16, 9, 10, 2, 8, 7}



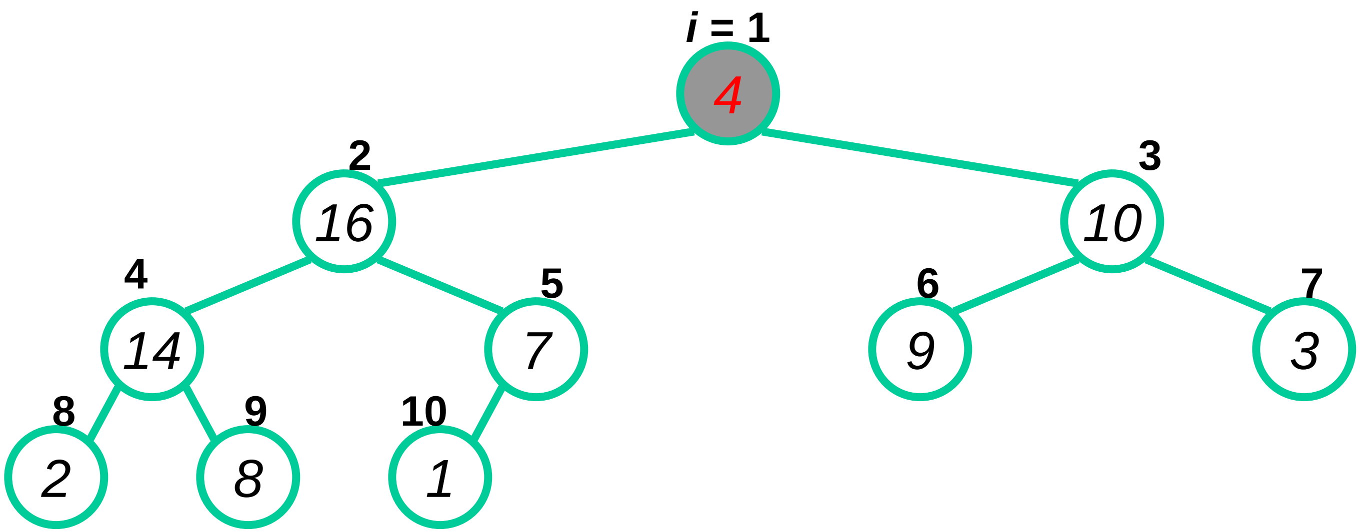
BUILD-MAX-HEAP() Example

A = {4, 1, 10, 14, 16, 9, 3, 2, 8, 7}



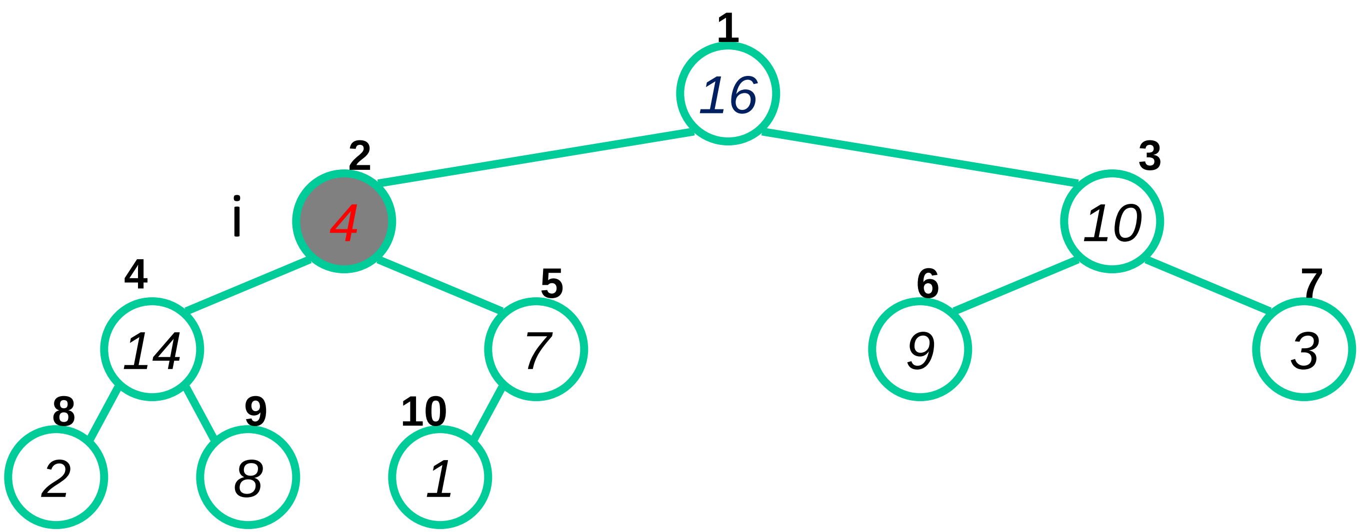
BUILD-MAX-HEAP() Example

A = {4, 16, 10, 14, 7, 9, 3, 2, 8, 1}



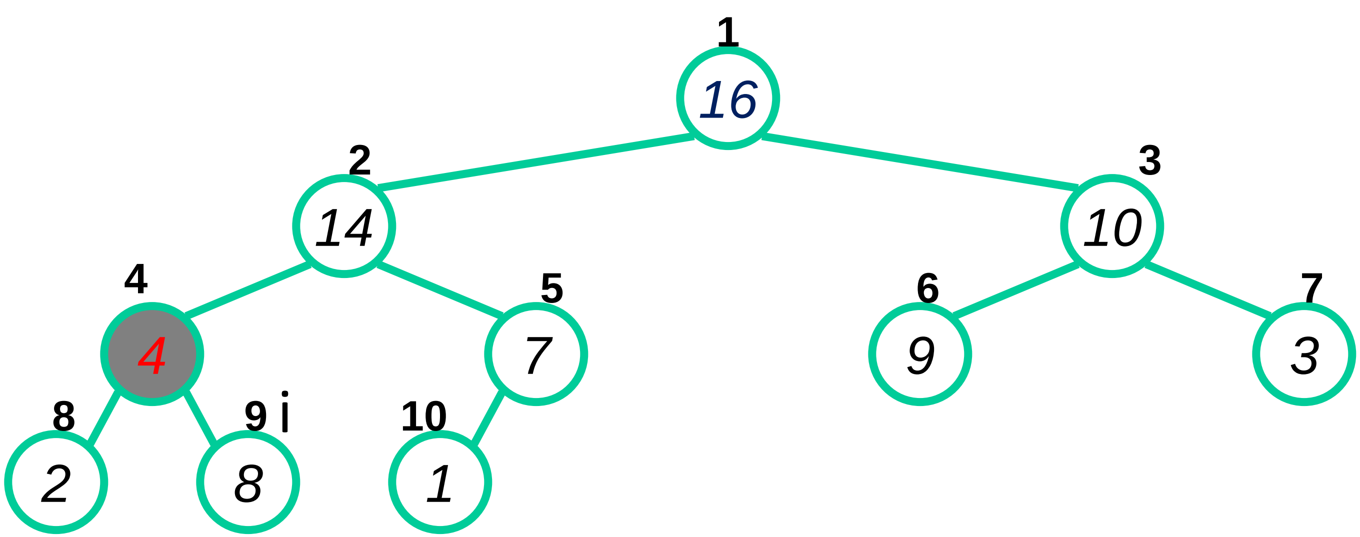
BUILD-MAX-HEAP() Example

A = {16, 4, 10, 14, 7, 9, 3, 2, 8, 1}



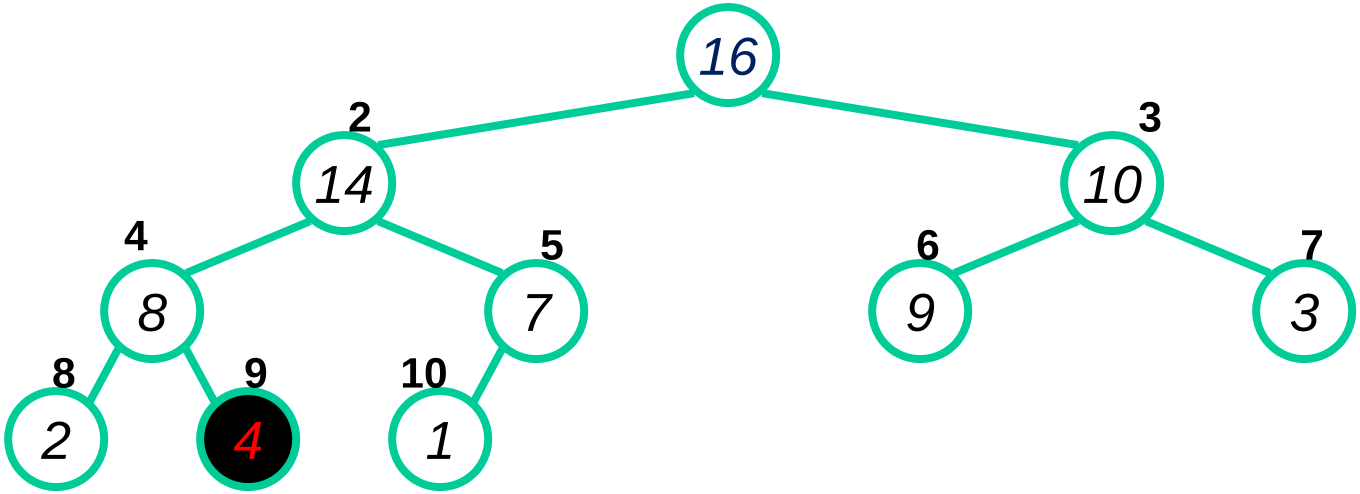
BUILD-MAX-HEAP() Example

A = {16, 14, 10, 4, 7, 9, 3, 2, 8, 1}



BUILD-MAX-HEAP() Example

A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}



BUILD-MAX-HEAP() Example

A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1}