Topics

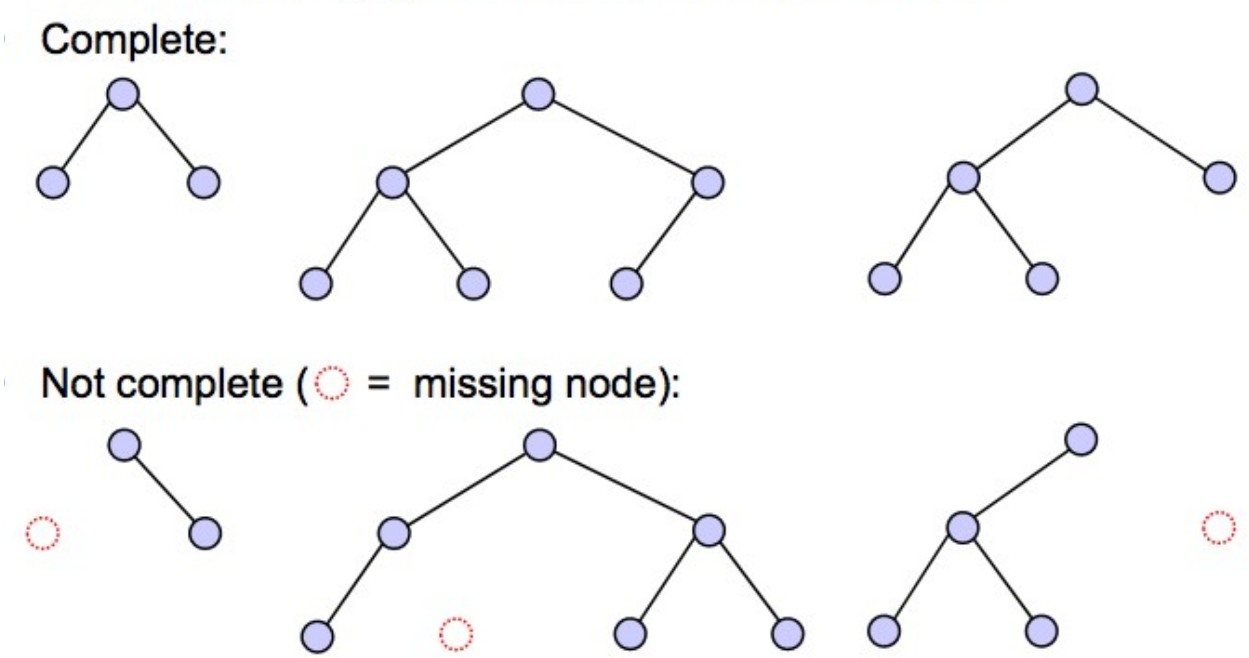
- Heap
- Heapify
- Heap Sort

Heap

- Not what it sounds like
  - i.e. not a disordered pile of items
- Partially ordered data structure
- Specifically, a heap is a complete binary tree where:
  - The element value of each parent node is greater than or equal to the element values its children (**max heap**)
  - Or, the element value of each parent node is less than or equal to the element values its children (*min heap*)

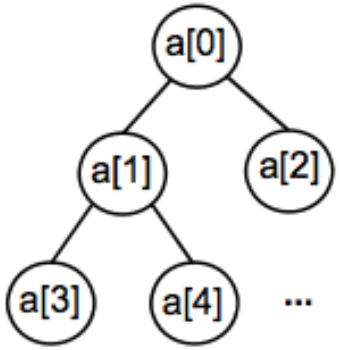
Complete Binary Tree?

- A binary tree of height ***h*** is complete if:
  - Levels ***0*** through ***h-1*** are fully occupied
  - There are no gaps to the left of a node in level ***h***

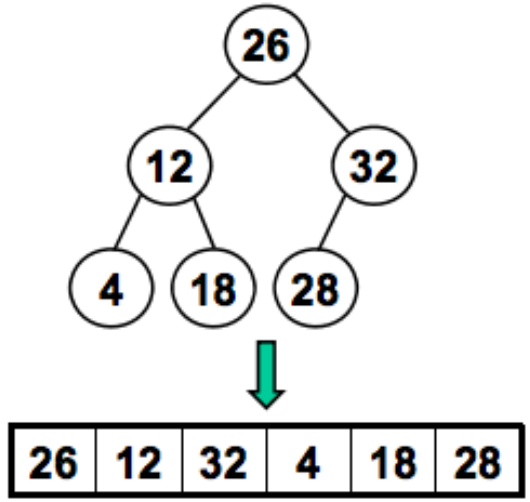


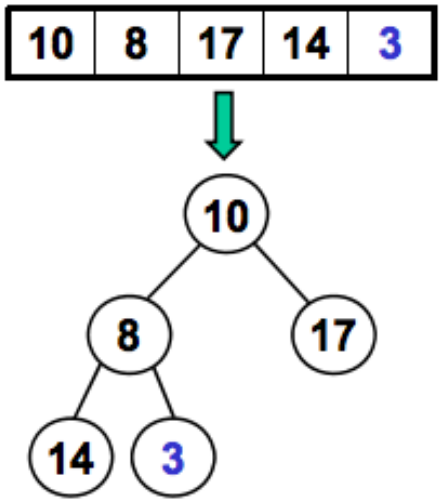
Why complete binary tree?

- Has simple array representation
- Nodes are stored in the order visited
  - Top to bottom
  - Left to right



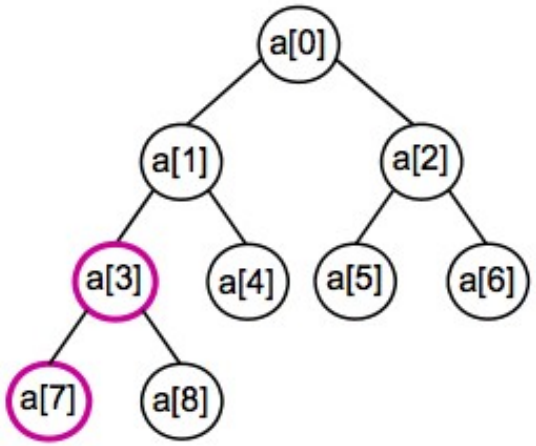
Examples:





Nodes and Index Positions

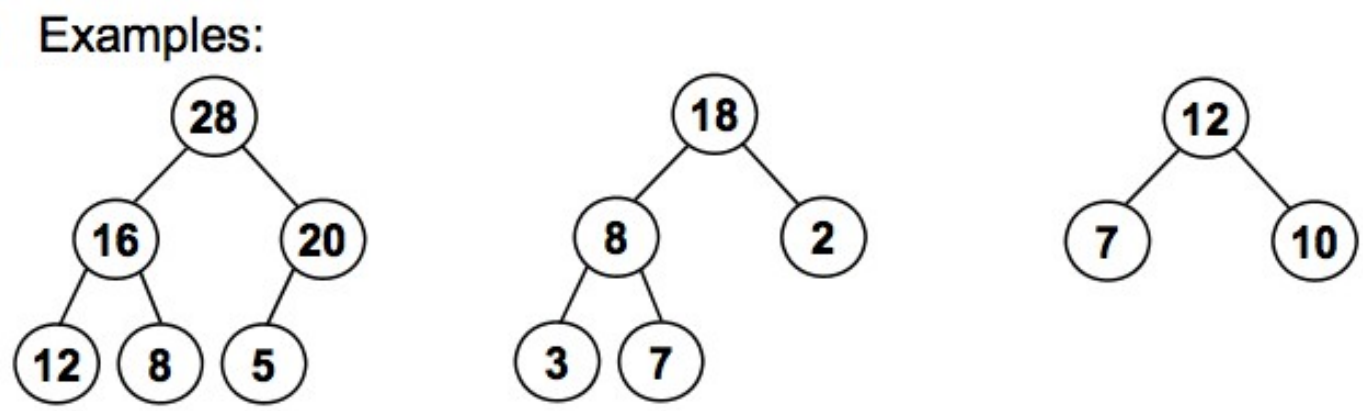
- Root node is `A[0]`
- Given node at `A[i]`
  - Left child `A[ 2*i + 1 ]`
  - Right child `A[ 2*i + 2 ]`
  - Parent `A[ floor( ( i – 1 ) / 2 ) ]`



- Examples
  - Left child of `A[1]: A[2*1+1] = A[3]`
  - Right child of `A[3]: A[2*3+2] = A[8]`
  - Parent of `A[4]: A[floor((4-1)/2)] = A[1]`

Max Heap Examples

- Largest value is always root node of tree
- Smallest value can be any leaf node
  - No guarantee which one it will be ...



Min Heap?

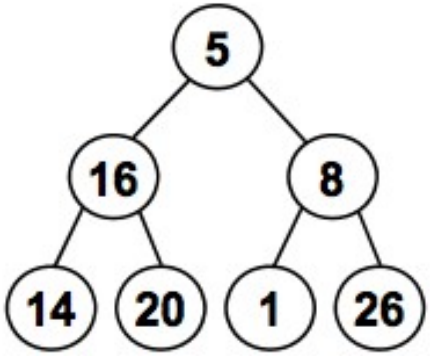
- Opposite of Max Heap
  - Smallest value is always root node of tree
  - Largest value can be any leaf node
    - No guarantee which one it will be ...

Max Heapify

- Convert an ordinary list of items to a heap
- Bottom up approach

Example:

0	1	2	3	4	5	6
5	16	8	14	20	1	26

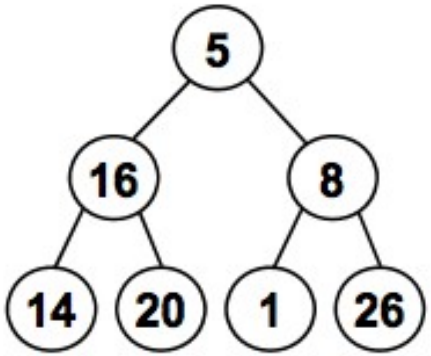


Max Heapify Pseudocode

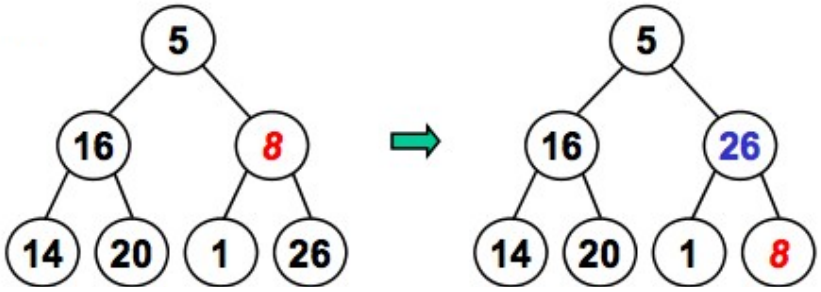
```
ALGORITHM maxHeapify( H[0 … n-1] )
// Constructs a heap from an existing list of values
// Input: list H
// Output: heap H
for i = floor( ( n-2 )/2 ) downto 0 do
    k = i, v = H[k]
    heap = false
    while not heap and 2*k+2 <= n do
        j = 2*k+1
        if j+1 < n // two children
            if H[j] < H[j+1], j = j + 1
        if v >= H[j]
            heap = true
        else
            H[k] = H[j] // swap parent and largest child
            k = j
    H[k] = v
```

Example

0	1	2	3	4	5	6
5	16	8	14	20	1	26



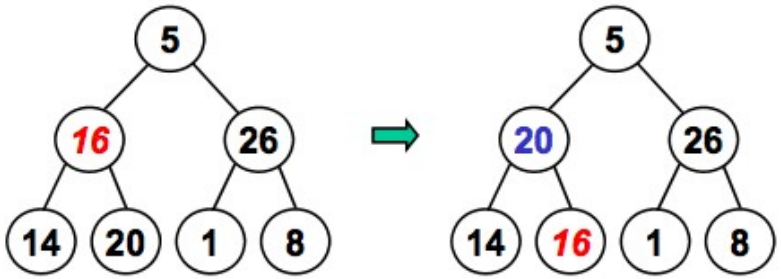

```
k = 2, v = 8, heap=false

    while 2*k+2 <= 7 and not heap

    j = 5 (5+1 < 7 –> two children)

    j = 6 ( 26 > 1 )

    H[2] < H[6] -> H[2] = H[6]

    k = 6

H[k] = 8
```



Example

0	1	2	3	4	5	6
5	16	26	14	20	1	8

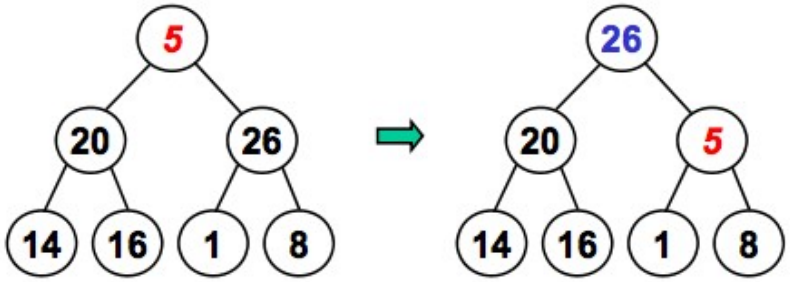
```
k = 1, v = 16, heap=false

    while 2*k+2 <= 7 and not heap

    j = 3 (3+1 < 7 –> two children)

    j = 4 ( 20 > 14 )

    H[1] < H[4] -> H[1] = H[4]

    k = 4

H[k] = 16
```



Example

0	1	2	3	4	5	6
5	20	26	14	16	1	8

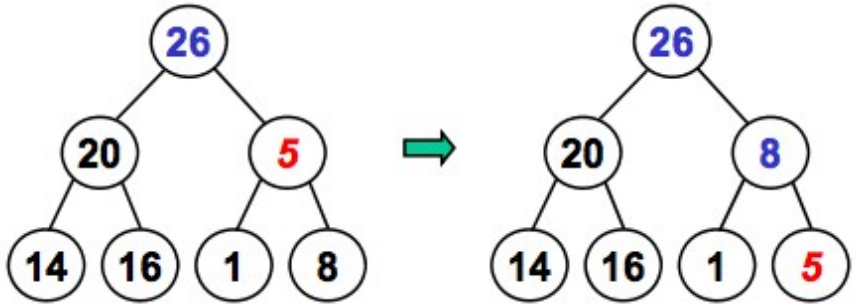
```
k = 0, v = 5, heap=false

    while 2*k+2 <= 7 and not heap

    j = 1 (1+1 < 7 –> two children)

    j = 2 ( 26 > 20 )

    H[0] < H[2] -> H[0] = H[2]

    k = 2

H[k] = 5
```



Example

0	1	2	3	4	5	6
26	20	5	14	16	1	8

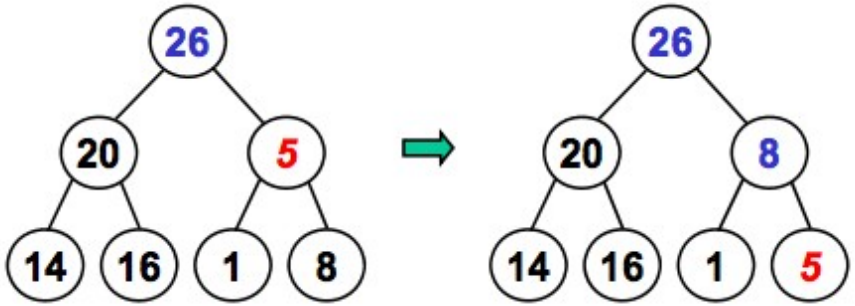
```
k = 0, v = 5, heap=false

    while 2*k+2 <= 7 and not heap

    j = 5 (5+1 < 7 –> two children)

    j = 6 ( 8 > 1 )

    H[2] < H[6] -> H[2] = H[6]

    k = 6

H[k] = 5
```



Example

0	1	2	3	4	5	6
26	20	8	14	16	1	5

Done



Heap Sort

- Use heap to sort a list of values
- Neat little algorithm
  - Bonus - very simple!

Heap Sort

- Algorithm
  1. Heapify the array (see below).
  2. Swap the first element in the heap with the last element in the heap.
  3. Reduce the size of heap by 1.
  4. Repeat Step 1 while size of heap is greater than 1.
- Step 1 in algorithm
  - Max heapify: values are sorted in increasing order
  - Min heapify: values are sorted in decreasing order