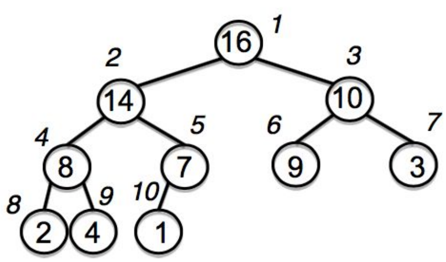
Lec4_Heaps and heap sort

Priority Queues

- A data structure implementing a set S of elements, each associated with a key, supporting the following operations
    1. insert (S,x) : insert element x into set S
    2. max(S) : return element of S with largest key
    3. extract_max(S) : return element of S with largest key and remove it from S
    4. increas_key(S,x,k) : increase the value of element x's key to new value k

Heap

- What is 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
      ex)



- Heap As a tree
    - Root of tree : first element in the array, corresponding to i = 1
    - parent(i) = i/2
    - left(i) = 2i
    - right(i) = 2i+1
- Operations
    - build_max_heap : produce a max heap from an unordered array
    - max_heapify : correct a single violation of the heap property in a subtree at its root
- Build_Max_Heap
    - Max_heapify 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]
        Max_Heapify(A,largest)
```

- Overall process (converts A[1...n] to a max heap)

```
Build_Max_Heap(A) :
    for i=n/2 downto 1
        do Max_Heapify(A,i)

Why start at n/2 ?
  A[n/2+1...n] : all leaves
```

- Running time
  n/4(1*c) + n/8(2*c) + … + 1(lg(n) * C)   (Set n/4 = 2^k)
  ⇒ c * (2^k * (1/(2^0)) + (2/2) + … + (k+1)/2^k)
  ⇒ 4c (if k → ∞)
  ⇒ O(n)

Heap-Sort

1. Build Max heap from unordered array;
2. Find maximum element A[1]
3. Swap element A[n] and A[1]
4. Discard node n from heap
5. Max heapify the new root

- Running time
  n : iteration
  max _heapify : O(lg(n))
  ⇒ O(nlg(n))