Heaps and Heap Sort

□ Priority Queue

Implements a sets of elements, each of elements associated with a key.

Insert(S, x): insert element x into S

max(S): reture element of S with the largest key

extract_max(S): do max(S) and extract it from S

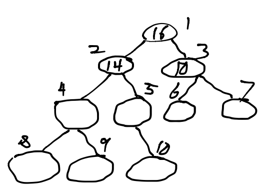
Increase key(S, x, k): Increast the value of x's key to new value k

□ Heap

An array visualized as a nearly complete binary tree. (Doesn't need to be sorted). Heighe of the tree lg(n)

e.g.

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 ~~the tree~~ : first element ($i = 1$)
- $\text{parent}(i) = i/2$
- $\text{left}(i) = 2i$
- $\text{right}(i) = 2i + 1$

∘ Max-heap Property

The key of a node is $\geq$ ~~the~~ ~~keys~~ of its children (Sorted)

- min-heap can be defined in similarly.

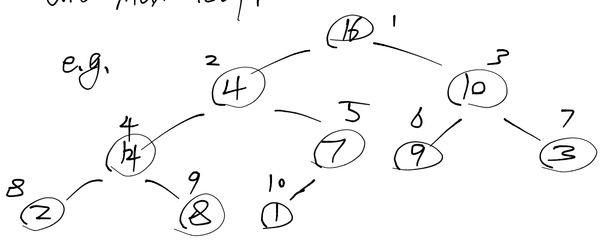
Big Question : how to maintain ~~the~~ max-heap property as we modify ~~the~~ heap.

∘ Heap Operations

- build_max_heap : produce a max-heap from a unsorted array
- max_heapify : Correct a single violation of a heap property in a sub-tree's root.

Assume that the trees rooted at left(i) and right(i) are max-heap.

e.g.



heap-size(A) = 10

max_heapify(A, 2):

- exchange A[2] with A[4]
- call max-heapify(A, 4)
- exchange A[4] with A[8]

Complexity: $\Theta(\lg n)$

- build_max_heap(A)

  Convert A[1, ... n] into a max-heap

  for i = n/2 downe to 1:

  do max_heapify(A, i)

  (explain: element A[n/2+1, ... n] are all leaves)

  Complexity: $\Theta(n)$

- observe max-heapify takes O(1) for nodes that are one level above the leaves, and in genel O(l) time that are l level above the nodes.

$l = 1$ $\quad \frac{n}{4}$ nodes

$l = 2$ $\quad \frac{n}{8}$ nodes

$\vdots \qquad \vdots$

$\lg n$ $\quad$ 1 nodes

Total amount of work in for loop:

$$\frac{n}{4}(1 \cdot c) + \frac{n}{8}(2c) + \frac{n}{16}(3c) + \cdots + 1(\lg n \, c)$$

set $\frac{n}{4} = 2^k \Rightarrow$

$$c\,2^k \underbrace{\left(\frac{1}{2^1} + \frac{2}{2^1} + \frac{3}{2^2} + \cdots \frac{k+1}{2^k}\right)}$$ bounded by const.

Heap Sort

- Build_max_heap from unordered array
- Find max element A[1]
- Swap elements A[n] with A[1]
  now max element is at the end of the array
- Discard note n from heap, decrement heap-size
- New route may violate max-heap, but children are
  max-heaps, max-heapify
  - Take $n \lg n$ time

e.g