MIT 6006 · Lecture #4. 02/15/2019

Heap – data structure.

Priority Queue:

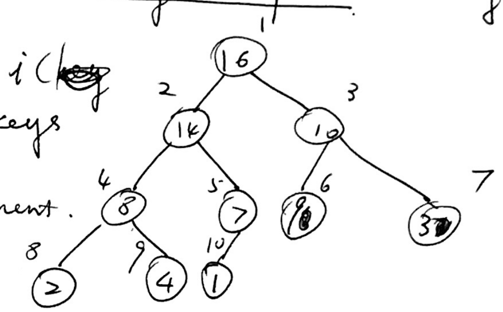
implement a set of elements, each of elements associated with a key (what is ADT?)

- insert (S, x) : insert element x into set S.
- $\max(S)$: return element of S with the largest key.
- extract-max(S) : . . . . . . and remove it from S.
- increase key(S, x, k): increase ~~key~~ the value of x's key to new value k.

Heap (as a tree).

- An array visualized as a nearly complete binary tree.

index :	1	2	3	4	5	6	7	8	9	10	→ i (~~key~~
value :	16	14	10	8	7	9	3	2	4	1	→ keys



the root of the tree : first element. $(i = 1)$.

parent $(i) = i/2$.

leaves left$(i) = 2i$ right$(i) = 2i+1$

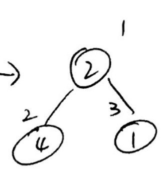
Max-heap property:

The key of a node is $\geq$ the keys of the children.

i.e.

1	2	3
2	4	1

→



→ NOT a max-heap!

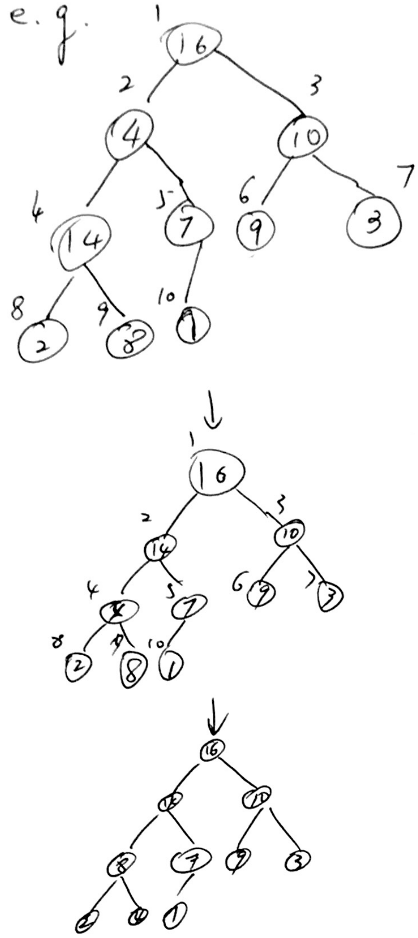
Heap operations:

build-max-heap : produce a max

↳ max-heapify : correct a single violation of the heap property in a subtree's root.

Max-heapify Assume the trees rooted at left(i) and right(i) are max heaps.

e.g.



MAX-HEAPIFY (A, 2)

(work from bottom up)

heap-size (A) = 10

EXCHANGE A[2] with A[4]

what is the complexity of max-heapify:

- the operation complexity is bounded by the levels of the tree, which is $O(\log n)$

- this (max_heapify) is the basic building block for the rest of this lecture!

Convert A[1 ... n] into a max_heap

Build_max_heap (A):

for $i = n/2$ downto 1 → because leaves are good!

do max_heapify (A, i)

$A[n/2 + 1 \ldots n]$ are all leaves.

$\boxed{O(n \lg n) \text{ simple answer.}}$

Observe Max-heapify takes $O(1)$ for nodes that are one-level above the leaves and in general $O(l)$ time for nodes that are l levels

$n/4$ nodes with level 1, $n/8$ with level 2.

Total amount of work in the for loop:

$$n/4(1\ c) + n/8(2c) + n/16(3c) + \dots + 1(\lg n\ c)$$

Set $n/4 = 2^k$

$$\Rightarrow c2^k\left(\frac{1}{2^0} + \frac{2}{2^1} + \dots + \frac{(k+1)}{2^k}\right) \leftarrow \text{convergent bounded by a constant}$$

(arithmetic series)

$\Rightarrow$ its $O(n)$.

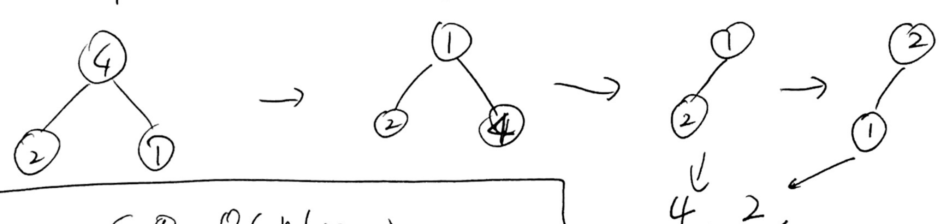
Heap sort (堆排序)

1. Build_max_heap from unordered array $O(n)$
2. Find max element $A[1]$ $O(1)$
3. Swap element $A[n]$ with $A[1]$ $O(1)$

~~4.~~ now max element is at the ~~end~~ end of array

4. Discard node n from heap - decrementing heap-size
5. New root may violate max heap ~~but~~ children are max heaps, do max_heapify

e.g.



heapsort: { ① $O(n\log n)$ ② in-place

merge-sort { ① $O(n\log n)$ ② not in-place

insertion-sort { ① $O(n^2)$ ② in-place

Readings 6.1 ~ 6.4.

What is a heap?

The term "heap" is coined in the context of heap-sort but it has since come to refer to "garbage-collected storage".

堆 < 数据结构. / 存储类型.

☆