LEC8 Binary Heaps

3.25 https://github.com/GUMI-21/MIT6.006_note

Priority queue interface (Subset of Set)

-build(x): init to items in x
-insert(x): add item x
-delete_max(): delete & return max-key item
-find-max(): return max_key item

Set AVL

add augmentation O(1) find_max()

Today: Heaps

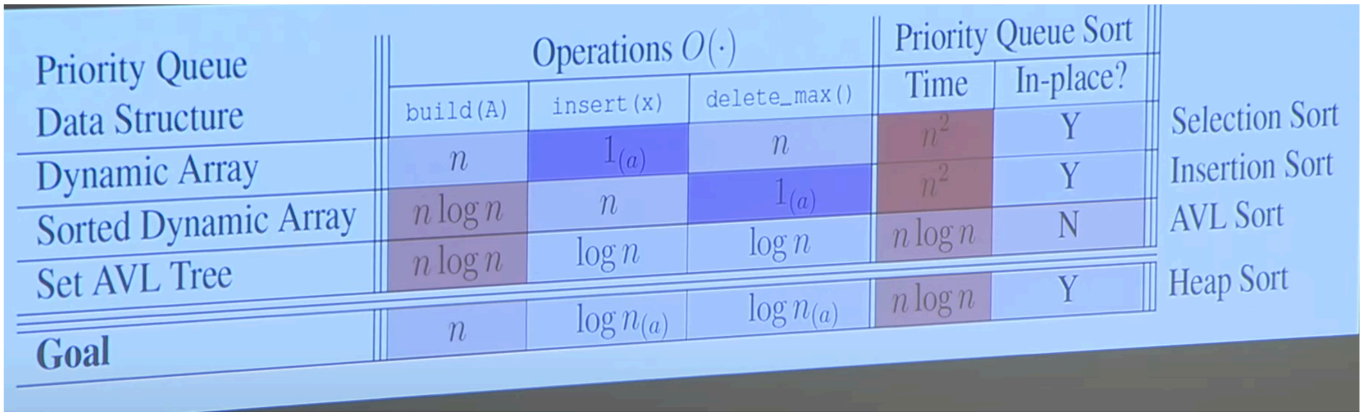
-priority queue interface & sorting Alg
-set AVL tree ->Avl Sort
-array -> selection/insertion sort
-binary heap -> heap sort ~ *inplace*

- *Array*
  insert O(1)
  delete_max() O(n)
  find_max() O(n)
- *sorted Array*
  delete_max(): O(1) am.
  insert: O(n)
  find_max: O(1)

Priority queue sort:

-insert(x) for x in A (build A)
-repeatedly delete*max()*

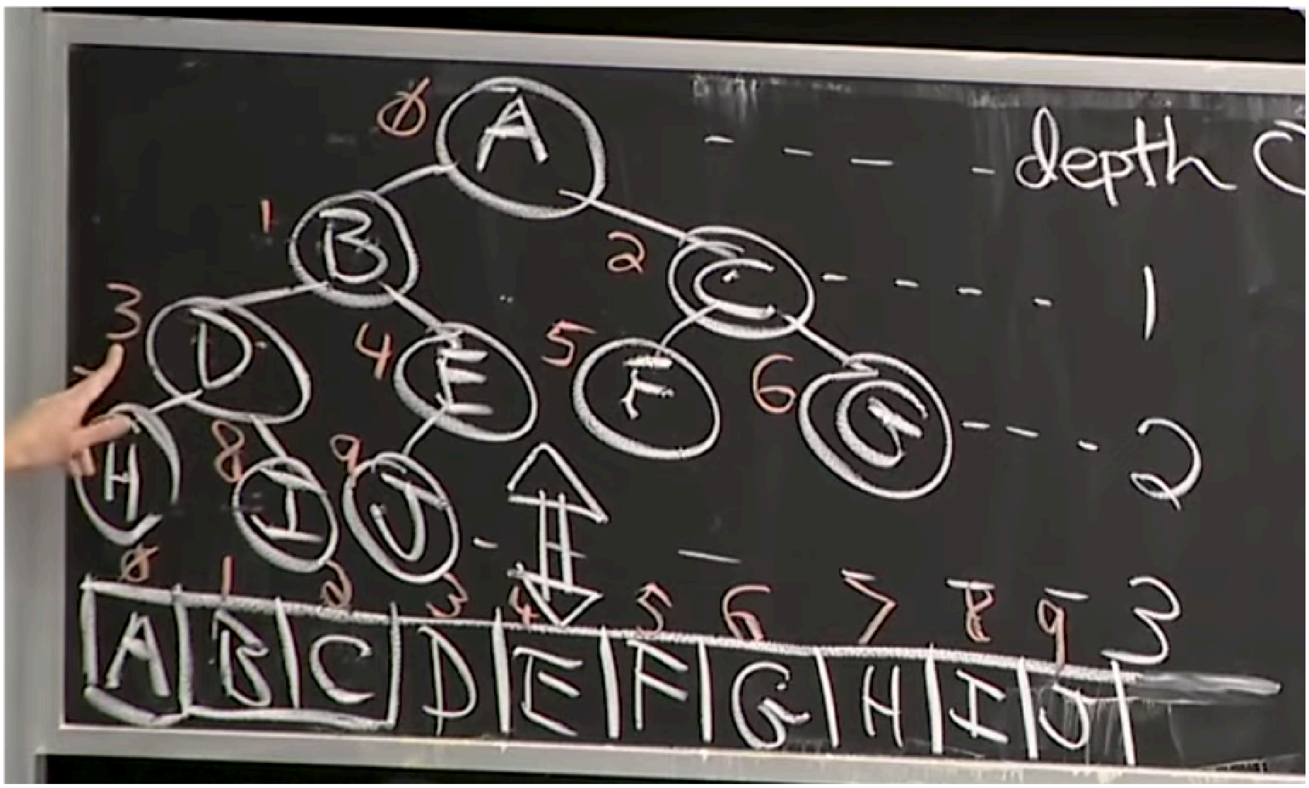
$T{build}(n) + n\cdot T{delete_max} <= n(T{insert} + T_(dete_max))$



Priority Queue Data Structure	Operations $O(\cdot)$ build(A)	insert(x)	delete_max()	Priority Queue Sort Time	In-place?	
Dynamic Array	n	$1_{(a)}$	n	n^2	Y	Selection Sort
Sorted Dynamic Array	$n \log n$	n	$1_{(a)}$	n^2	Y	Insertion Sort
Set AVL Tree	$n \log n$	$\log n$	$\log n$	$n \log n$	N	AVL Sort
Goal	n	$\log n_{(a)}$	$\log n_{(a)}$	$n \log n$	Y	Heap Sort

Heap

- *complete binary tree*
  -2^i nodes at depth i
  -except at max depth where nodes are left-justified
  =>height $\lceil \log n \rceil$

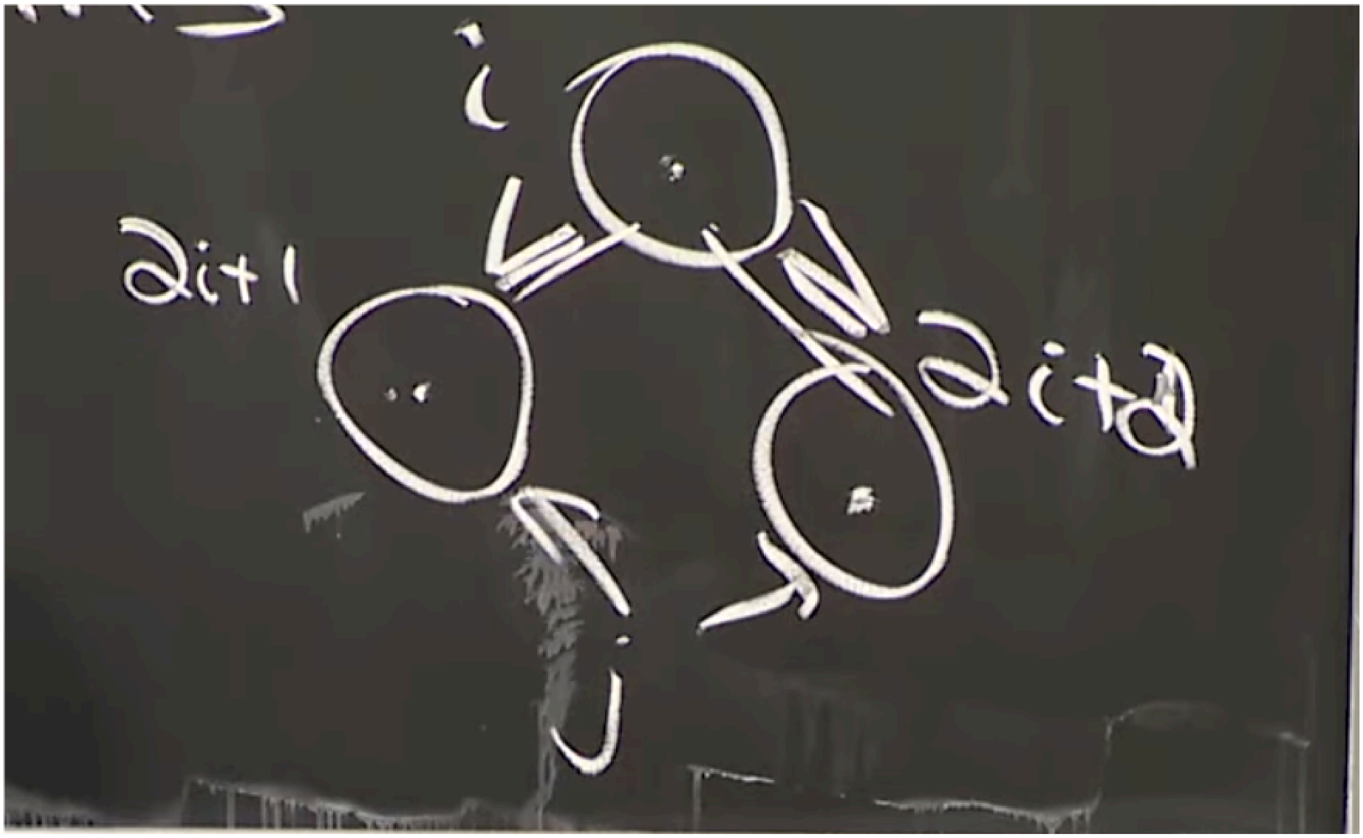


- *the depth order of complete binary tree*
  for every complete binary tree, there is only one projection unique array, and for any array there is on unique projection complete binary tree too.
- *Implicit data structure*
  -no pointers, just store array of n items.
  -left_child(i)=2i+1 see in tree

-right_child(i)=2i+2
-parent =(i-1) / 2

Binary heap Q

array representing a complete binary tree where every node i satisfy *Max-Heap Property* at i: Q[i] >= Q[j] for $j \in \{left(i), right(i)\}$



- *Lemma*
  =>Q[i] >= Q[j] for node j in subtree(i)
  the property queue just need to delete max

Alg

- **insert(x)**
  -Q insert_last(x)
  -max_heapify_up(|Q| - 1)
  max_heapify_up(i): if Q[parent(i)].key < Q[i].key: swapQ[parent(i)] & Q[I], recurse on parent.
  nad if i = 0: return.
  -runtime: O(logn)
- **delete_max()**:
  what we need to do: delete root item.
  -swap Q[0] with Q[|Q| -1]
  -Q.delete_last()

-max_heapify_down(0)

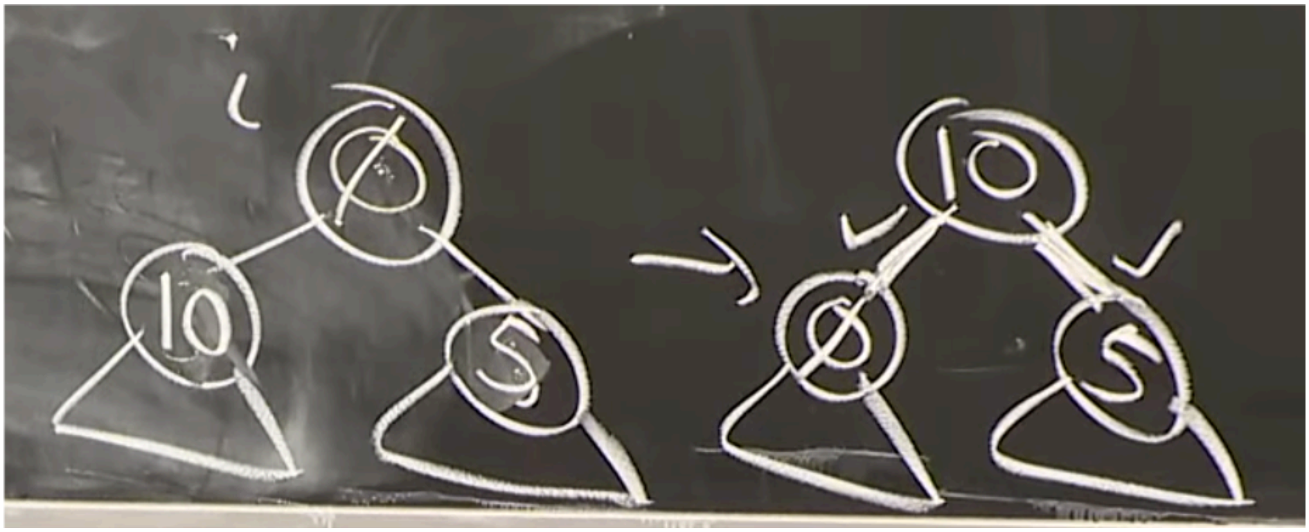
max-heapify_down(i):

-if i leaf: done

-let j $\in \{left(i), right(i)\}$,maximizing Q[j].key

-if Q[i] < Q [j]: swap Q[i] - Q[j]

-recurse on j



-runtime: O(logn)

In place

-insert: increament |Q|

-delete-max: decreamtn |Q|