COL106 - Data Structures and Algorithms

Heaps, Maps

Priority Queues & Heaps

Data structure that allows
- arbitrary element insertion
- removal of element with first priority

assigned as a "key"

can be implemented using an array!

thanks to

min Heaps

Binary Tree storing priorities (& elements) at nodes satisfying
- Relational property: Priority(i) > Priority(parent(i))
- Structural Property: All levels except last are full. Last filled from left to right.

$$2^{h} \le n \le 2^{h+1}-1$$

$$h = \lfloor \log_2 n \rfloor$$

$$1+2+\dots+2^{h-1} = 2^{h}-1$$

Operation	Linked List (unsorted)	Linked List (Sorted)	Heap
insert (k,v)	O(1)	O(n)	O(log n)
min()	O(n)	O(1)	O(1)
remove Min()	O(n)	O(1)	O(log n)

insert at last level & move up.

min always at root

swap with last node & Heapify.

Heapify(i)

Building a heap

$O(\log \prod_{i=1}^{n} i)$

- Repeatedly insert an element in to the heap.

$$\boxed{\sum_{i=1}^{n} O(\log i) = \Theta(n \log n)}$$

Exercise

- **Bottom up constuction:** leaves are heaps; so start from there.

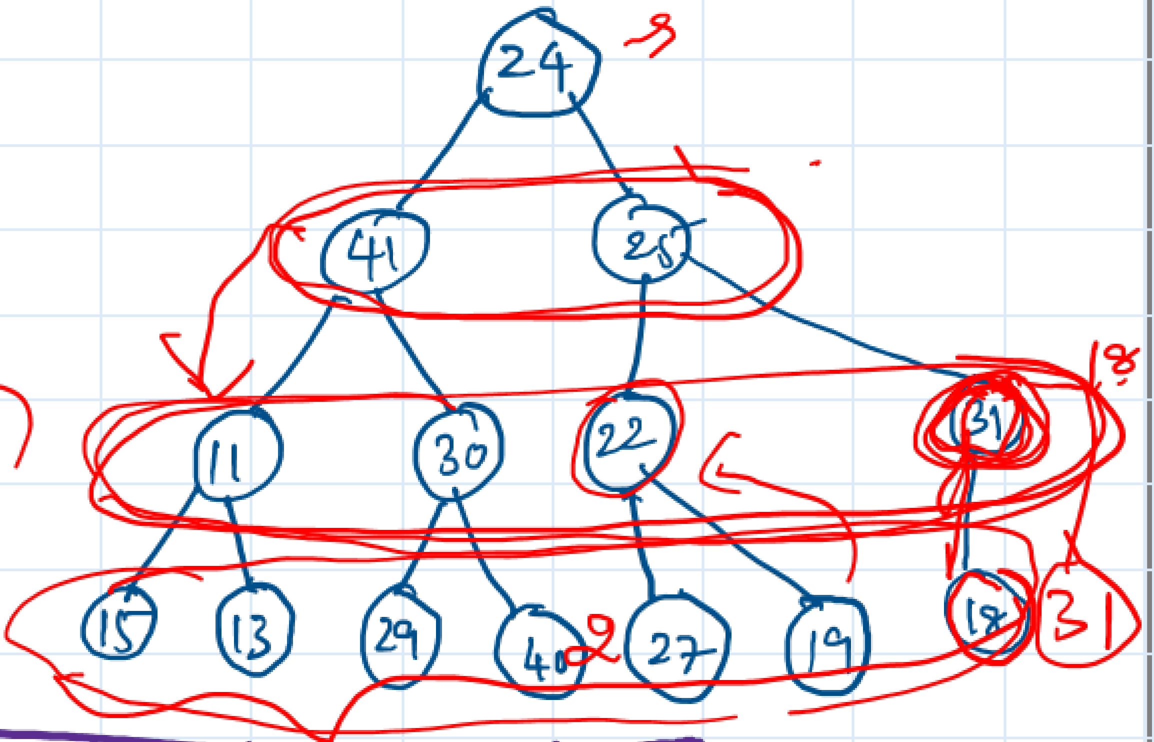
BUILD-HEAP (A)

for $i \leftarrow \lfloor n/2 \rfloor$ down to 1

  HEAPIFY(i)

$O(n)$

$n = 2$



24	41	25	11	30	22	31	15	13	29	40	27	19	18
1	2	3	4	5	6	7	8	9	10	11	12	13	

Building a Heap

- Correctness: induction on i, all trees rooted at $j > i$ are heaps.

- Analysis: Idea: Time taken by Heapify(i) $= O($height of subtree rooted at $i)$

  - For $n/2$ nodes at height 1, heapify() requires ≤ 1 swap each
  - for $n/4$ " " 2, " " ≤ 2 "

$\frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 2 + \dots + \frac{n}{2^i} \cdot i$

  - $n/2^i$ . . . . i, ... $\leq i$ "

Total swaps $= O\left(\left(\sum_{i \geq 1}^{\log n} \frac{i}{2^i}\right) \cdot n\right) = O(n)$ $\quad \sum_{i \geq 1}^{\log n} \frac{i}{2^i} = c$

"Amortized" analysis

Claim: $$\sum_{i=1}^{n} \frac{i}{2^i} \le 2$$

Proof: $$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \quad \text{if } |x|<1$$

$$\sum_{i=0}^{\infty} i\,x^{i-1} = \frac{1}{(1-x)^2}$$

$$\Rightarrow \sum_{i=0}^{\infty} i\,x^i = \frac{x}{(1-x)^2}$$

plug in $x = 1/2$

Heap Sort

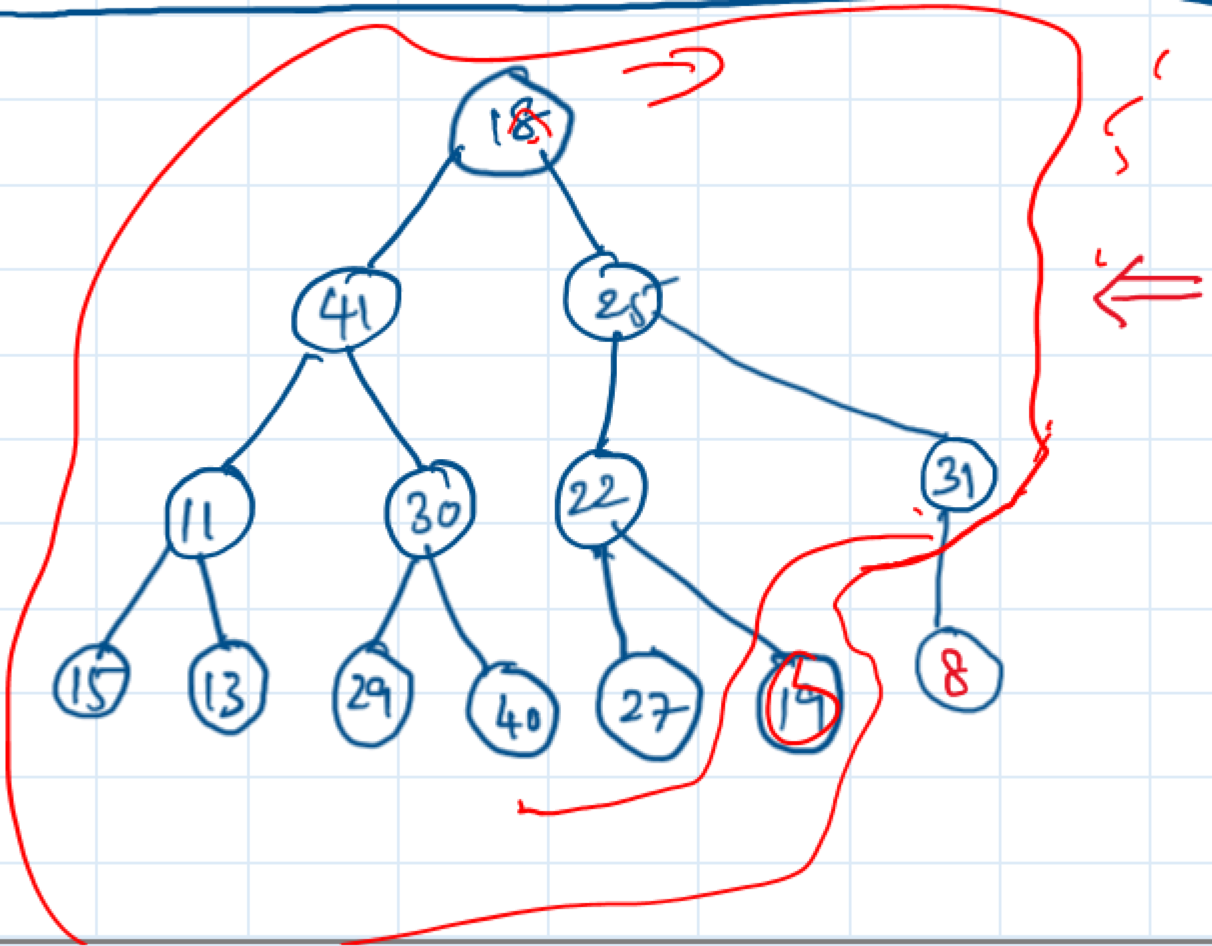
Sorting

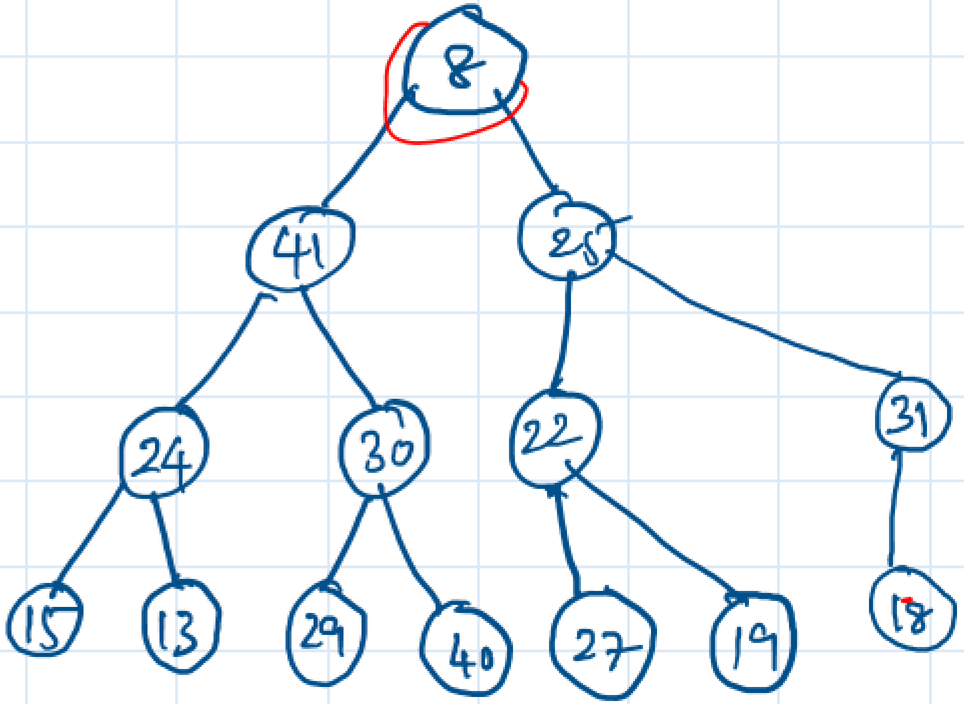
Given unsorted array $A = \langle a_1, \dots, a_n \rangle$

output $\tilde{A} = \langle \pi(a_1), \dots, \pi(a_n) \rangle$

π: a permutation

- Create a heap from A
- Remove minimum repeatedly
- Sort in place by moving deleted element to end of heap.





The Map ADT (Dictionary / Associative array)

- Model a "searchable" collection of key-value entries
- Operations: Searching, deleting and inserting items.
- Multiple entries with same key are not allowed.
- <u>Applications</u>:
  - Student record database key = entry number
    value = student name
  - web: key = URL. value = webpage content
- <u>Methods</u>: size(), isEmpty(), get(k), put(k, v), remove(k), keySet(), values(), entrySet()

Only require comparisions for equality (no order required)

Implementing Map ADT

java.util.Map
java.util.Dictionary

- Arrays, LinkedList (inefficient)
- Unordered sequence:
    - get(k), remove(k) takes O(n) time
    - put(k,v) takes O(1) time
    - Could be useful if there are not too many get(), remove() ops required.

      (e.g. log files in a computer)
- Ordered sequence (say array):
    - get(k) takes O(log n) time, put(k,v) & remove(k) take O(n) time
    - good if all you need is search.

Hashtables

Direct addressing: Array indexed by key :- takes $O(1)$ time for all operations, but $O(r)$ space.
↳ r is range of keys.

– e.g: COL106 registry.

2021 CS

HashTable:

1. Hash function
2. Array (table)

– $O(1)$ expected time

– $O(n+m)$ space where m is table size.
n is number of entries.

Store item (k, o) at index $i = h(k)$

$h :$ → {0, ..., 999}

Example: Let keys be entry numbers of COL106 students.

{001, ..., 999}

2022 PH 10140 → 140

How to deal with alphanumeric keys?

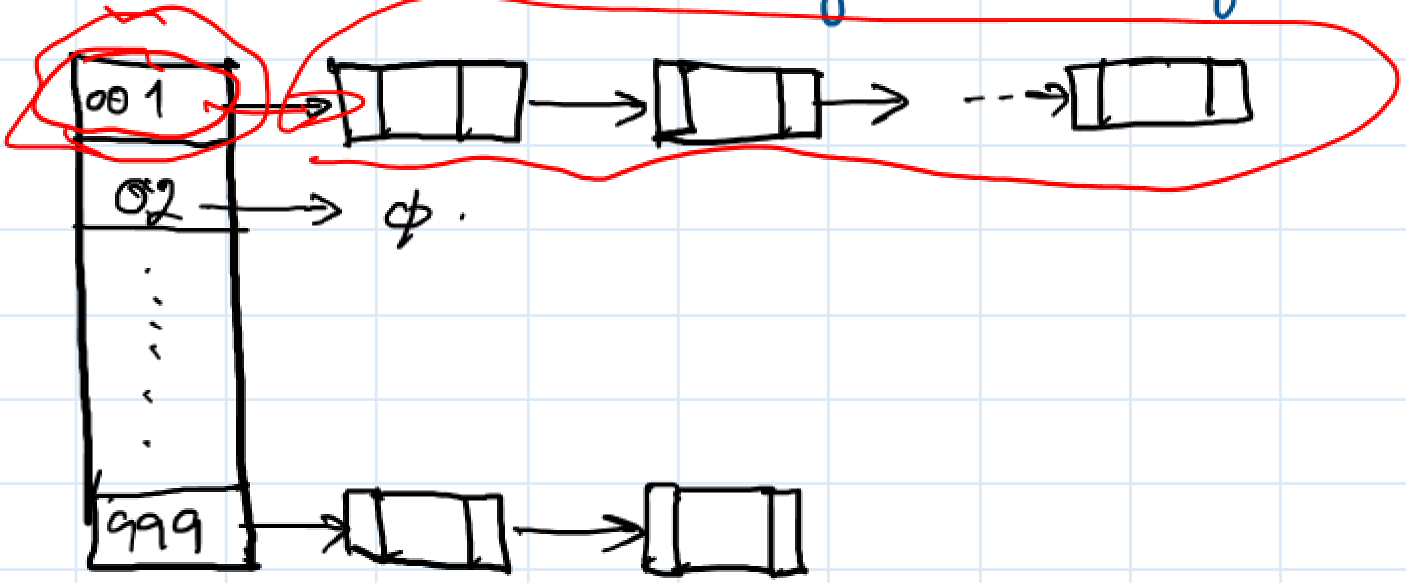
Hash function: Take last 3 digits :

"Collision"

Collision Resolution

- What if I have two keys hashing to same location?

- **Chaining:** Have an array of links indexed by keys, having list of items with same key.



To find/insert/delete an element, lookup position in table & search/insert/delete the element in the linked list of the hashed slot.

6

340

insert(k, v)

remove(k)

get(k)

Analysis of Hashing

- Element with key k stored in slot $h(k)$.

$$h: U \xrightarrow{n} \{0, 1, \cdots m-1\}$$

$|U| = n$

- Assume time taken to compute $\boxed{h(k)}$ is $\Theta(1)$.

Array indexed by

- What is a good hash function?
  - distributes keys "evenly" among slots, easy to compute, less space.
  - ideally slot is picked uniformly at random.
  - Simple uniform function (assumption)
  - Load factor: $\alpha = \frac{n}{m}$ → # elements → size of table

size of universe

not actually!
We need to know where a key got mapped to!
also not easy to store