Priority Queues

- priority queue : remove the largest (or smallest) item

API class MaxPQ<Key extends Comparable>

- void insert(Key v)
- Key deleteMax()
- boolean isEmpty()

- generalizes: stack, queue, randomized queues
- Challenge: find the largest M items in a stream of N items
  - * not enough memory to store N items
  - * use a min oriented priority queue

```
if (pq.size() > M)
    pq.delMin()
```

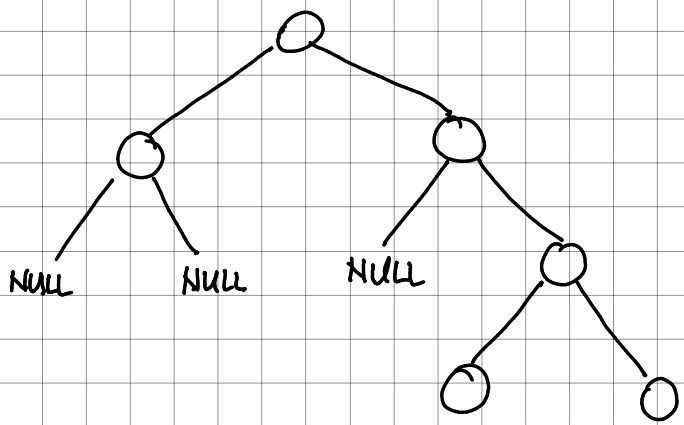
- can't sort because we can't store all N items
- elementary PQ $O(M*N)$ — too slow

- binary heap $O(N \log M)$ / space M

Binary Heap

* Complete Binary Tree

Binary Tree : empty or node with links to left and right binary trees



Complete tree : perfectly balanced, except for bottom level

Property : Height of a complete tree with N nodes is $\lfloor \lg N \rfloor$

* height only increases when N is a power of 2

Binary heap : array representation of a heap-ordered complete binary tree

Heap-ordered binary tree

- keys in nodes
- parent's key no smaller than children's key (>=)

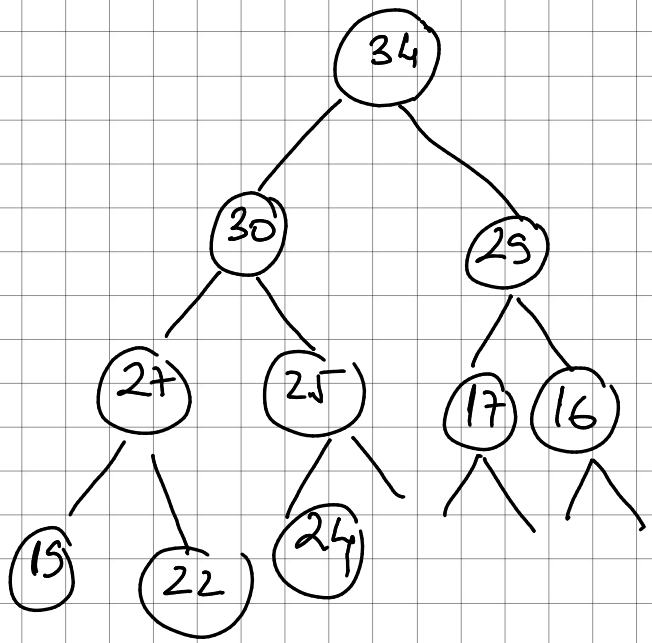
Array representation

- indices start at 1
- take nodes in level order
- no explicit links needed

Binary heap properties

- largest key is a[1] => root of the binary tree
- can use array indices to move through tree.
  - parent of node k is at ⌊k/2⌋ (int division)
  - children of node k are at 2k, 2k+1

34 30 29 27 25 17 16 15 22 24



max oriented binary heap

Promotion in a heap

Scenario - child's key becomes larger than the parent's key

Fix : • exchange key in child with key in parent

• repeat until heap is ordered

Peter principle: node promoted to level of incompetence

- Insertion in a heap
  - add a new node at the end and swim it up
  - cost: at most $1 + \lg N$ compares

Demotion in a heap

Scenario
- parent's key becomes smaller than one (or both) of its children's

Fix
- exchange key in parent with key in larger child
- repeat until order is restored

Delete the maximum in a heap

- exchange root node with node at the end, then sink the small node down
- at most $2 \lg N$ compares

* Fibonacci max PQ => insert $O(1)$
del max $O(\lg |1|)$
max $O(1)$

- immutability of keys
- underflow and overflow
  - underflow: throw exception if deleting from empty PQ
  - overflow: add no-arg constructor and use resizing of array

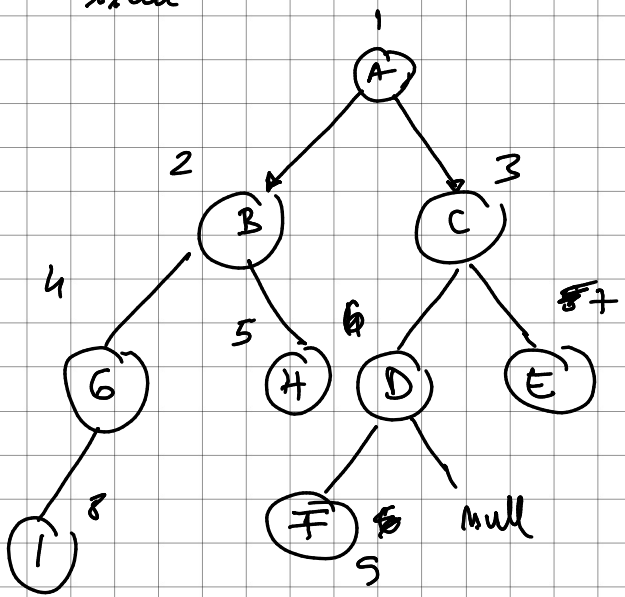
* min oriented priority queue => replace (less) with (greater)

Heap Sort

- start with an array of keys in arbitrary order
- build a max-heap in place

- repeatedly remove maximum key

* Build max heap using the bottom up method
* sort down — repeatedly delete the largest remaining item



A B C D E F
size = 6

Mathematical Analysis

- heap construction uses $\leq 2N$ compares and exchanges
- heapsort uses $\leq 2N \lg N$ compares and exchanges

- first sorting algorithm [in-place] with $N \log N$ – worst case

- in-place $N \log N$ / not used as much in practice
  - inner loop longer than QS
  - makes poor use of cache memory
  - not stable

Event-driven simulation

* Simulate the motion of N moving particles that behave according to the laws of elastic collision.

Hard disc model

- moving particles collide with each other and the walls
- each particle is a disc with position,

velocity, mass and radius.

- no other forces