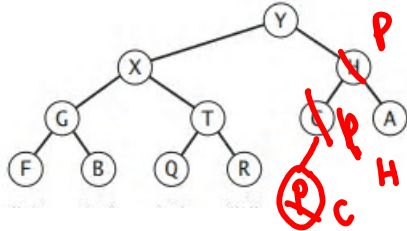


Week 4 Flipped Activities

1. Max-Heap

Consider the following binary tree representation of a max-heap.



level order
traversal of the
tree

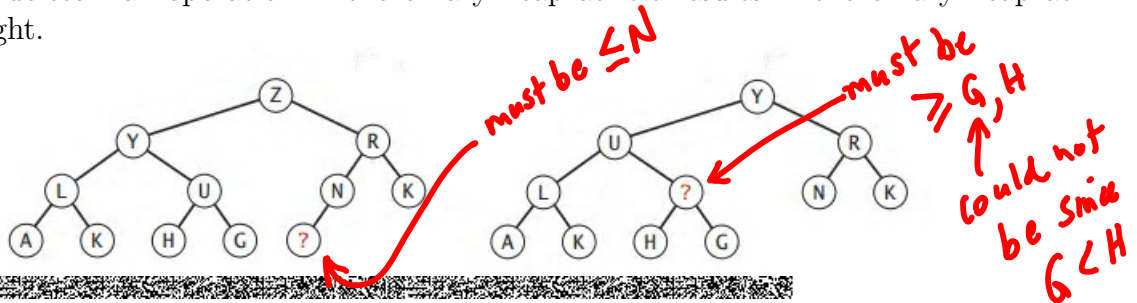
(a) Give the array representation of the heap.

[Y X H G T C A F B Q R]

(b) Insert the key **P** into the binary heap above, circling any entries that changed. Show the array representation of the heap.

the idea is to insert P as the next leaf node and swim up maintaining PQ order invariant

(c) A delete-max operation in the binary heap at left results in the binary heap at right.



Which of the keys below could be the one labeled with a question mark? Circle all possibilities.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

(d) Answer TRUE or FALSE

- It is possible to build a max-heap in linear time, given N random keys.
- It is possible to remove N items from a max-heap in linear time.
- It is possible to design a priority queue implementation that performs insert, max, and delete-max in $(1/3) * \log_2 N$ compares per operation, where N is the number of comparable keys in the data structure.

(i) Yes. Here is why. There are two ways to build a max-heap

a. insert N elements into a max-heap starting with an empty heap

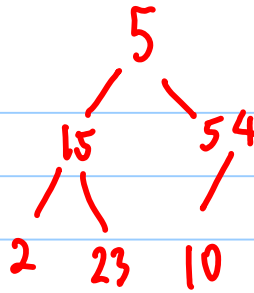
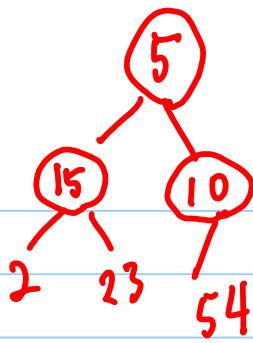
Each insert cost $\log N$ time and hence a total of $N \log N$ cost. So this method, called top-down cannot do this.

b. Bottom up method. Starting with non-leaf nodes from bottom, sink each node while maintaining max-heap invariant. See example below.

Start with some random set: [5 15 10 2 23 54]

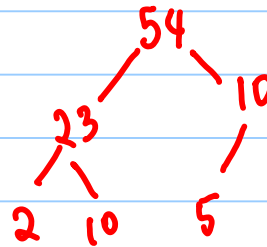
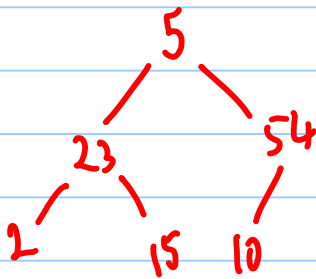
Visualize this in a binary tree

starting with 10 (first non-leaf node circled), sink it while maintaining the heap order property. So 10 would be exchanged with 54



now do the same for 15

finally for 5



Analysis: There are 2^k elements at level k . Based on our sink algorithm each one will sink at most $(h - k)$ where $h = \log N$ is the height of the tree. So to sink all of them we have to do the following work.

$$\sum_{k=1}^h 2^k \cdot (h - k) \sim \int_{k=1}^h 2^k (h - k) \sim N$$

$h \sim \log N$

Not expected to do this but know that linear construct is possible

(ii) The answer is NO. Because if it is possible, we can build a PQ in linear time and we can do del-max in linear time and hence we have a sorting algorithm, that can sort in linear time. But we showed that sorting has a lower bound of $N \log N$

(iii) Similar to the argument in (ii), if we can do this, we can insert in $1/3 N \log N$ and delete in $1/3 N \log N$ and we have a sorting algorithm in $2/3 N \log N$, a violation of $N \log N$ lower bound

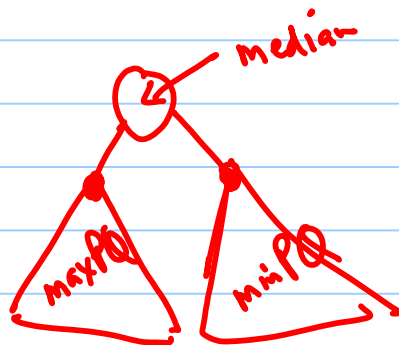
Randomized PQ solution

- `sample()`: Pick a random array index r (between 1 and N) and return the key $a[r]$.
- `delRandom()`:
 - *Select*: pick a random array index r (between 1 and N) and save away the key $a[r]$, to be returned.
 - *Delete*: exchange $a[r]$ and $a[N]$ and decrement N .
 - *Restore heap order invariants*: call `sink(r)` and `swim(r)` to fix up any heap order violation at r . Note that $a[N]$ in the original heap need not be the largest key, so the call to `swim(r)` is necessary.

```
public Key sample() {
    int r = 1 + StdRandom.uniform(N); // between 1 and N
    return a[r];
}

public Key delRandom() {
    int r = 1 + StdRandom.uniform(N); // between 1 and N
    Key key = a[r];                  // save away
    exch(r, N--);                    // to make deleting easy
    sink(r);                         // if a[N] was too big
    swim(r);                         // if a[N] was too small
    a[N+1] = null;                   // avoid loitering
    return key;
}
```

3. Dynamic median - we can design a solution (note that there can be other ways to do this) as follows. We combine a minPQ, maxPQ and a single element as follows



In this construction, when we insert, we insert to either maxPQ or minPQ ($\log N$ time). Note that median can tell us which way to go. Also note that we want to maintain maxPQ and minPQ almost the same size, so we have the median at the top or closer to it. All the elements in maxPQ are less or equal to median and all elements in minPQ are greater than or equal to median. So this structure allows all operations as expected

1. find median in constant time (middle or max of max-heap or min of min-heap)
2. insert in $\log N$, since we are inserting to one of the PQ's
3. remove in $\log N$, since we are removing middle or max of max-heap or min of min-heap and then adjusting the structure

2. Randomized priority queue

Describe how to add the methods `sample()` and `delRandom()` to our binary heap implementation of the MinPQ API. The two methods return a key that is chosen uniformly at random among the remaining keys, with the latter method also removing that key.

<code>public class MinPQ<Key> extends Comparable<Key>></code>		
	<code>MinPQ()</code>	<i>create an empty priority queue</i>
$\log N \leftarrow$	<code>void insert(Key key)</code>	<i>insert a key into the priority queue</i>
$1 \leftarrow$	<code>Key min()</code>	<i>return the smallest key</i>
$\log N \leftarrow$	<code>Key delMin()</code>	<i>return and remove the smallest key</i>
{		
	<code>Key sample()</code>	<i>return a key that is chosen uniformly at random</i>
	<code>Key delRandom()</code>	<i>return and remove a key that is chosen uniformly at random</i>

- (a) Implement the `sample()` method in constant time and . For simplicity, do not worry about resizing the underlying array.
- (b) Implement the `delRandom()` method in time proportional to $\log N$, where N is the number of keys in the data structure. For simplicity, do not worry about resizing the underlying array.

See solution above

3. Dynamic Median Design a data type that supports the following operations and performance requirements.

- (a) insert in logarithmic time
- (b) find-the-median in constant time
- (c) remove-the-median in logarithmic time.

See above