Write Priority Queue Using Heap

Heap →

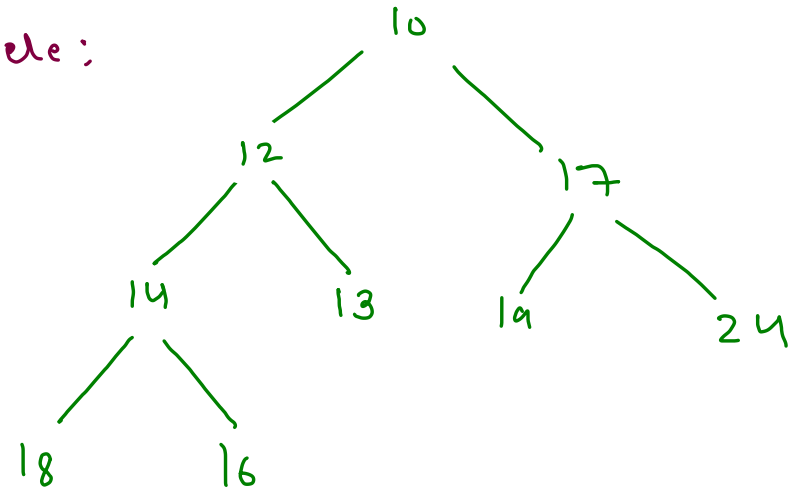
(i) complete binary tree → till second level, levels should be completely filled and last level should be filled L -> R

(ii) heap order property → priority (parent) > priority of both its child.

peek : O(1)

most prioritised ele :

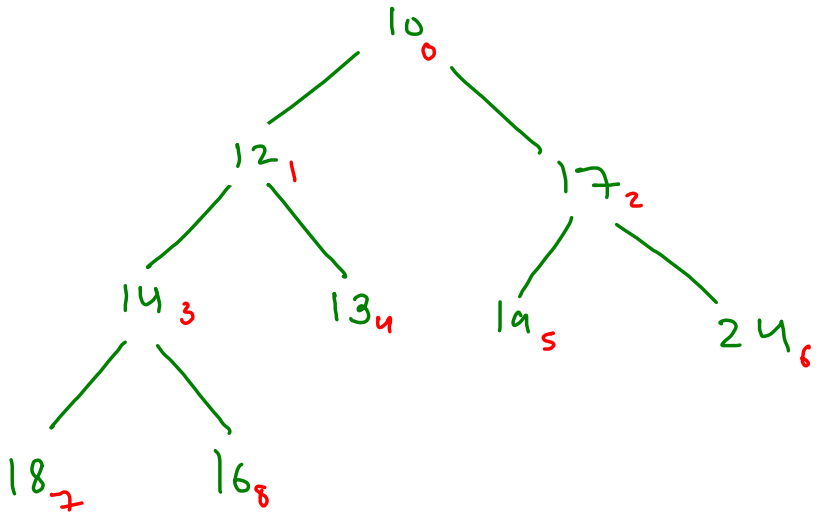
root



CBT

priority -> smaller values has higher priority

min heap



Ai: 10 12 17 14 13 19 24 18 16
0 1 2 3 4 5 6 7 8

pi → lci, rci

lci = 2*pi + 1

rci = 2*pi + 2

Ci → pi

pi = (ci - 1) / 2

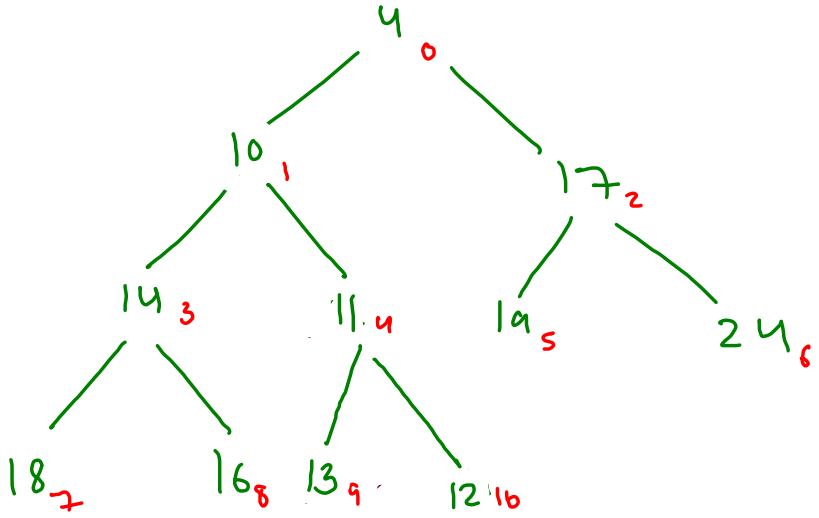
CBT

Al: 4 10 17 14 11 19 24 18 16 13 12

0 1 2 3 4 5 6 7 8 9 10

peek: return list.get(0);

add → log n



→ at last addition

→ upheapify

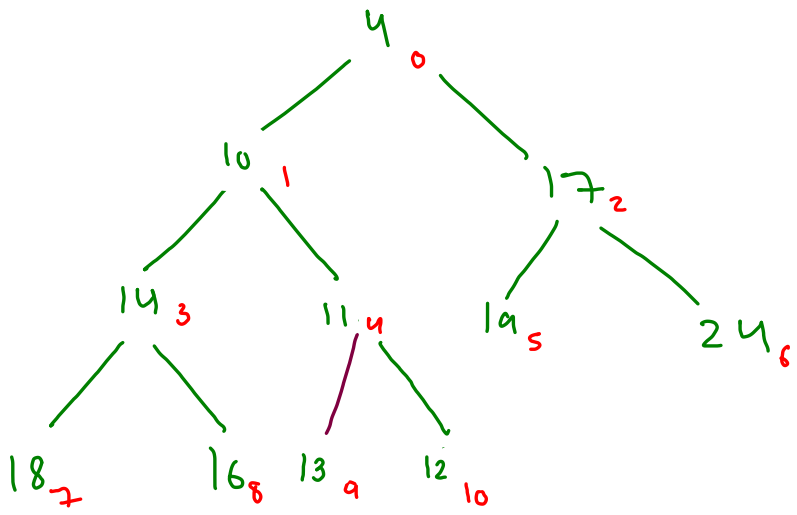
add(11)

add(4)

O(1)

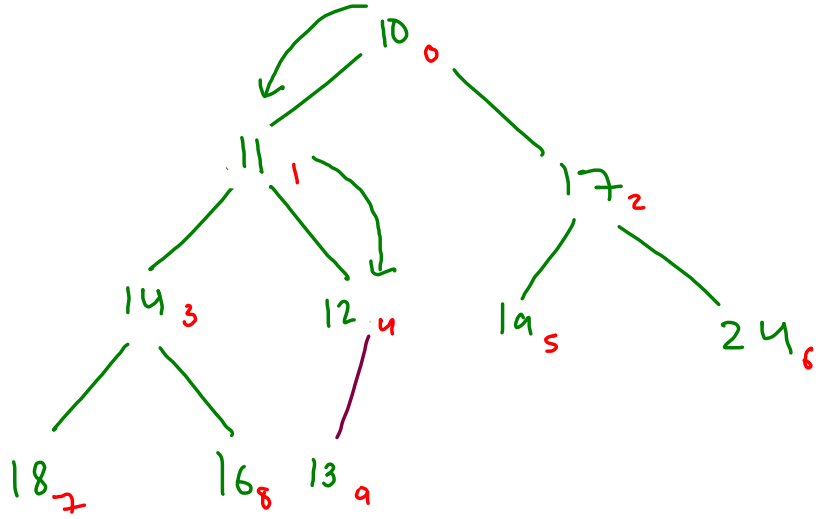
```
public void add(int val) {
    data.add(val);
    upheapify(data.size()-1);
}

private void upheapify(int ci) {
    if(ci == 0) {
        return;
    }

    int pi = (ci-1)/2;

    if(data.get(ci) < data.get(pi)) {
        swap(ci,pi);
        upheapify(pi);
    }
}
```

h : log n



add(11)

add(4)



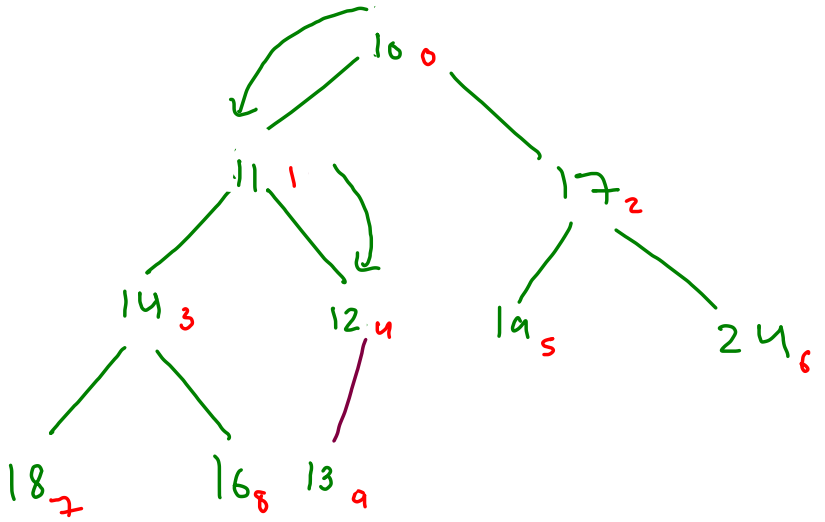
remove

(i) swap first & last and remove from last.

(ii) downheapify (0)

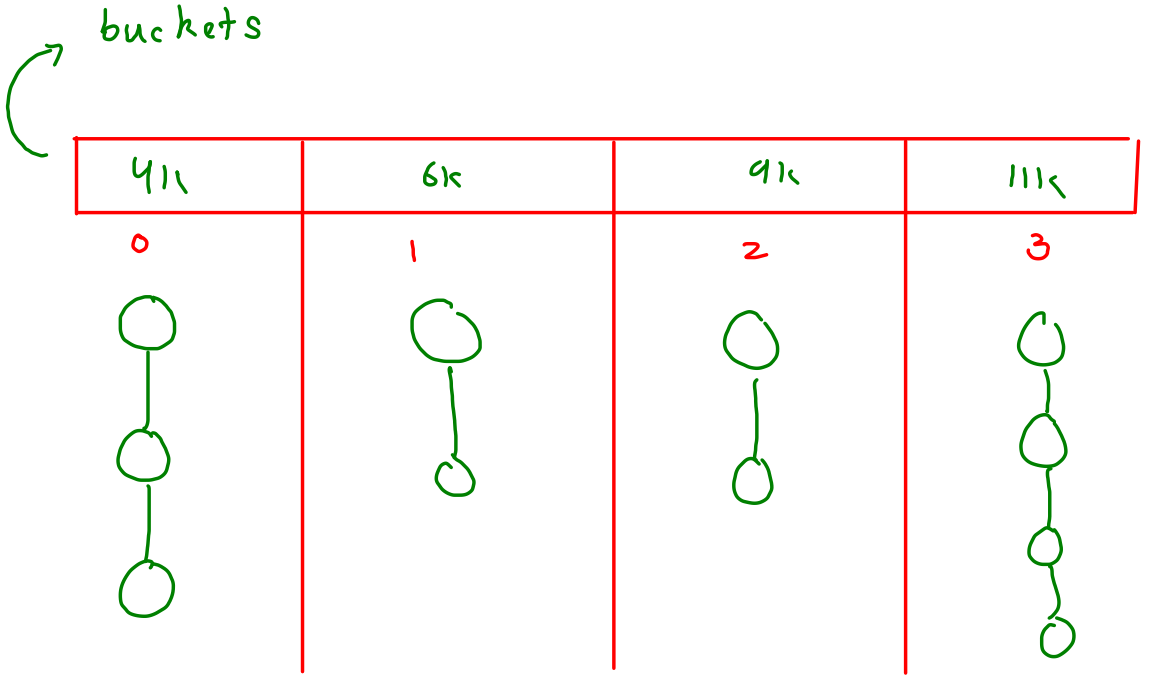
end removal

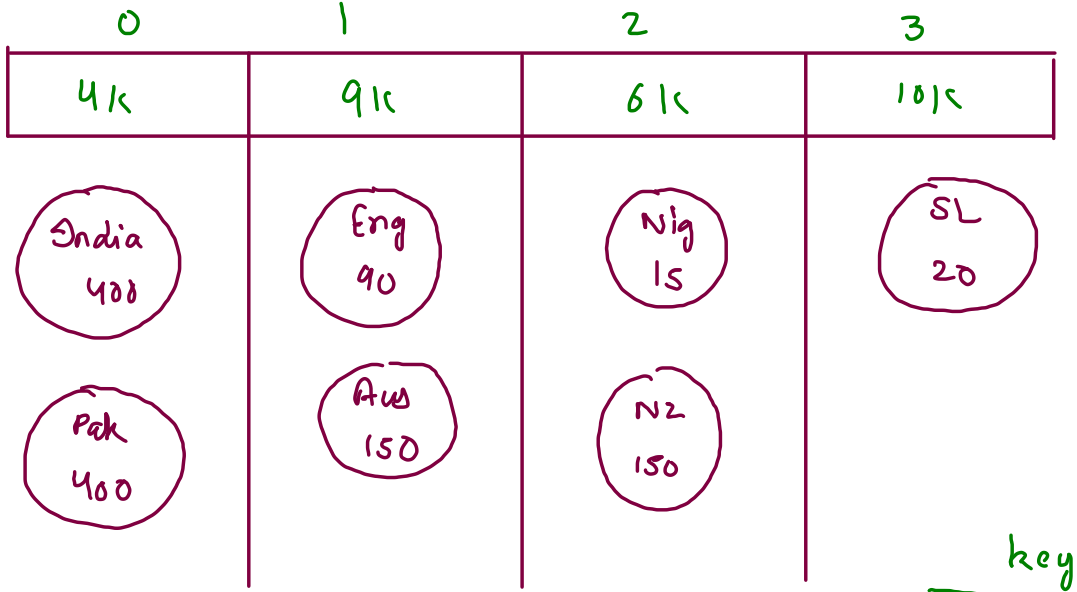
```java
public int remove() {
    if(size() == 0) {
        System.out.println("Underflow");
        return -1;
    }
    else {
        swap(0,data.size()-1);
        int ele = data.remove(data.size()-1); -> O(1)
        downheapify(0); -> h: logn

        return ele;
    }
}

private void downheapify(int pi) {
    int lci = 2*pi + 1;
    int rci = lci + 1;

    int mpi = pi; //most priority index

    if(lci < data.size() && data.get(lci) < data.get(mpi)) {
        mpi = lci;
    }
    if(rci < data.size() && data.get(rci) < data.get(mpi)) {
        mpi = rci;
    }


    if(pi != mpi) {
        swap(pi,mpi);
        downheapify(mpi);
    }
}
```



HM creation

```
public static class HashMap<K, V> {
  private class HMNode {
    K key;
    V value;

    HMNode(K key, V value) {
      this.key = key;
      this.value = value;
    }
  }

  private int size; // n
  private LinkedList<HMNode>[] buckets; // N = buckets.length

  public HashMap() {
    initbuckets(4);
    size = 0;
  }

  private void initbuckets(int N) {
    buckets = new LinkedList[N];
    for (int bi = 0; bi < buckets.length; bi++) {
      buckets[bi] = new LinkedList<>();
    }
  }
```





put (N2, 150)

put (India, 400)