PQ creation

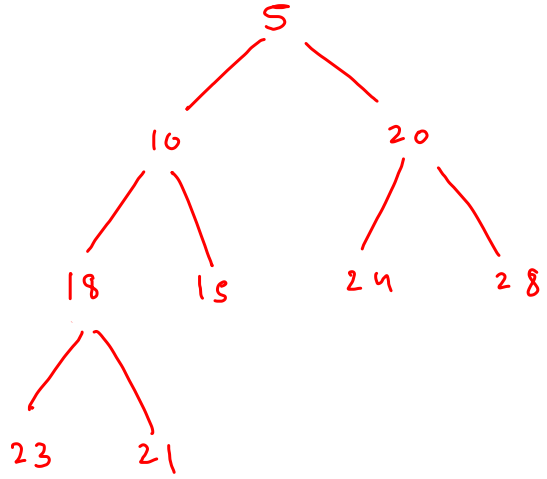
peek -> O(1)

add -> log n

remove -> log n

heap (min)

smaller value has higher priority



heap -> binary tree

(i) heap order property

priority(parent) > both child priority

(ii) complete binary tree

ht -> h

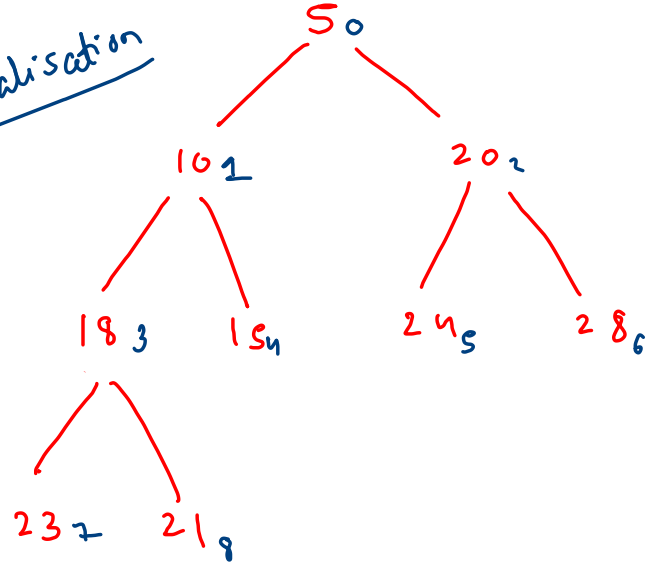
last level -> h

starting (0 to h-1) completely fill

h^{th} level should left to right

actual data : 5_0 10_1 20_2 18_3 15_4 24_5 28_6 23_7 21_8

Visualisation



pi to childs

lci = 2pi + 1

rci = 2pi + 2

heap :

(i) hop

(ii) cbt

---

ci to parent

$$pi = \frac{ci - 1}{2}$$

data : 5_0 10_1 20_2 18_3 15_4 24_5 28_6 23_7 21_8

pi to childs

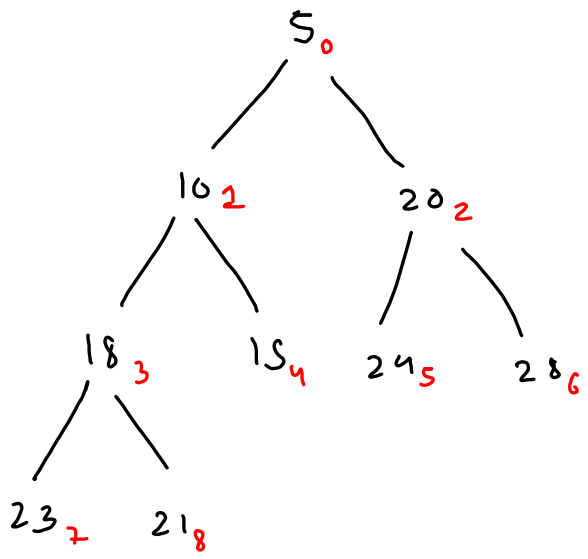
lci = 2pi + 1

rci = 2pi + 2

---

ci to parent

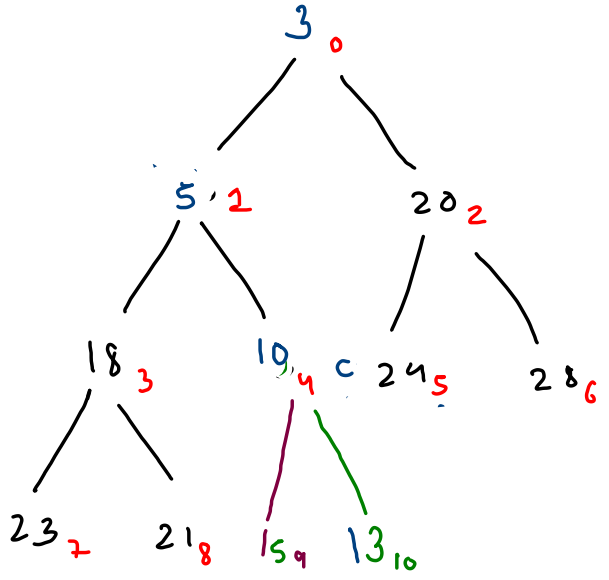
$pi = \frac{ci - 1}{2}$



peek -> return data[0]

size -> data.size()

data : 5₀ 10₁ 20₂ 18₃ 15₄ 24₅ 28₆ 23₇ 21₈ 13₉ 3₁₀

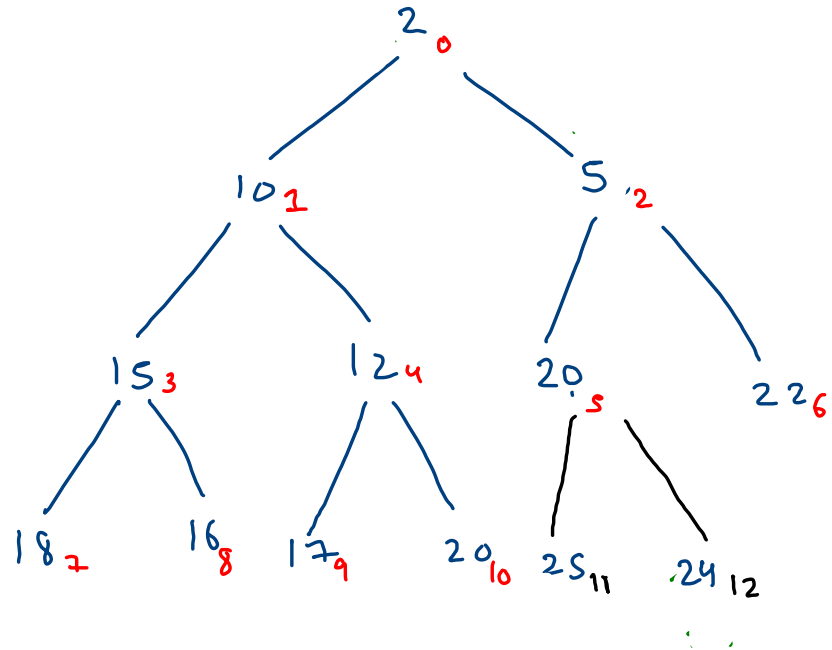


pq.add(13)

pq.add(3)

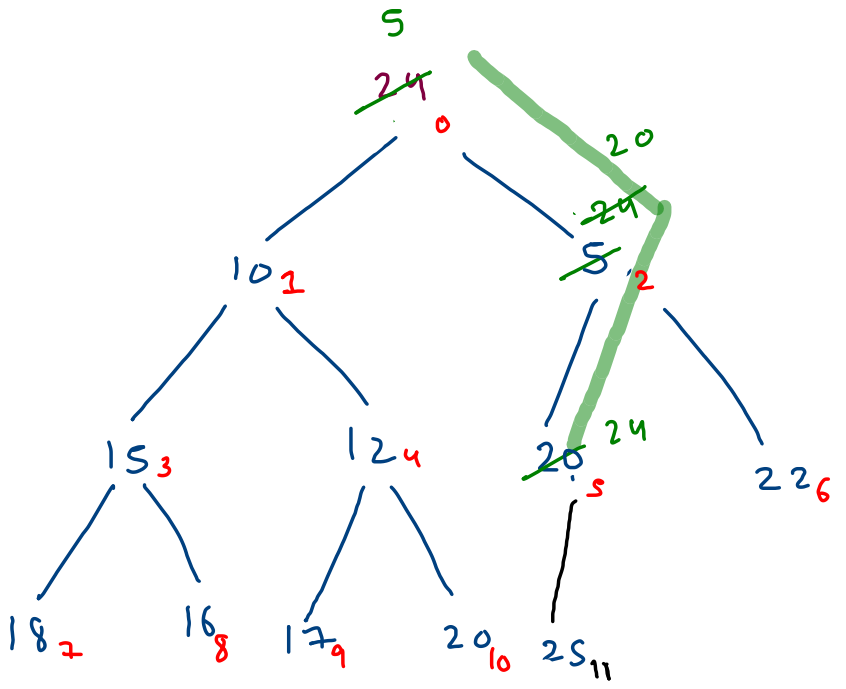
pq.peek() → 3

```java
public void add(int val) {
    data.add(val); // o(1)
    upheafipy(data.size()-1); //o(log n)
}

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

    int pi = (ci - 1)/2;

    if(data.get(ci) < data.get(pi)) {
        //child has more priority than parent
        swap(ci,pi);
        upheapify(pi);
    }

}
```



pq.add(24)

pq.add(2)



remove -> log n

swap (fi, li)

data.remove(li);

downHeapify(0);

↓

logn

```java
public int remove() {
    if (data.size() == 0) {
        System.out.println("Underflow");
        return -1;
    } else {
        int li = data.size() - 1;
        swap(0, li);

        int val = data.remove(li); //o(1)

        downheapify(0); //o(log n)
        return val;
    }
}
```

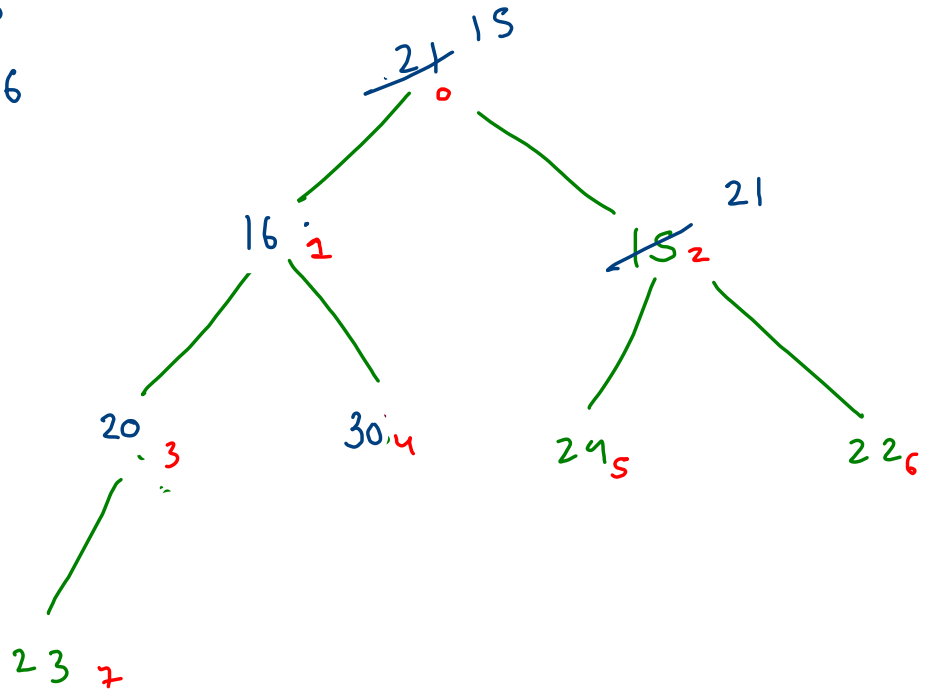
```java
private void downheapify(int pi) {
    int hpi = pi;

    int lci = 2 * pi + 1;
    int rci = 2 * pi + 2;

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

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


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


}
```

pi = 2

lci = 5

rci = 6

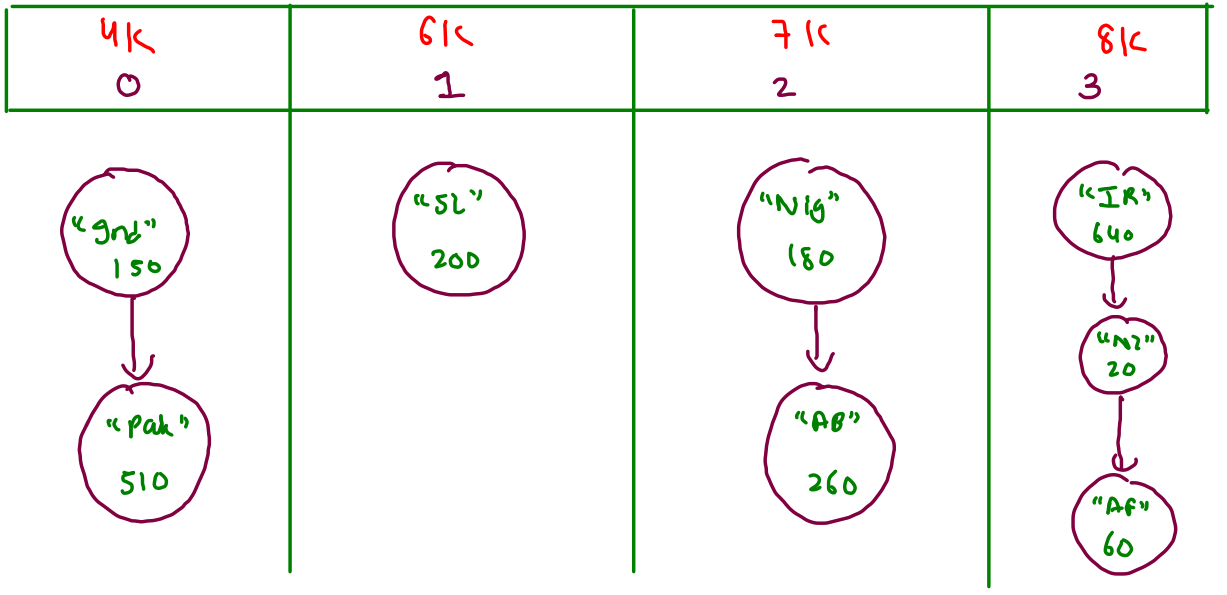
hpi = 2



```java
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<>();
    }
  }

  public void put(K key, V value) throws Exception {
    // write your code here
  }

  public V get(K key) throws Exception {
    // write your code here
  }

  public boolean containsKey(K key) {
    // write your code here
  }

  public V remove(K key) throws Exception {
    // write your code here
  }

  public ArrayList<K> keyset() throws Exception {
    // write your code here
  }

  public int size() {
    // write your code here
```

LinkedList < HMNode > [] bucket;

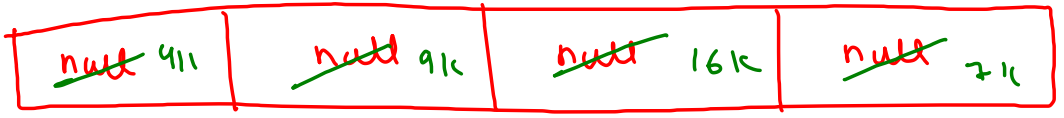
bucket

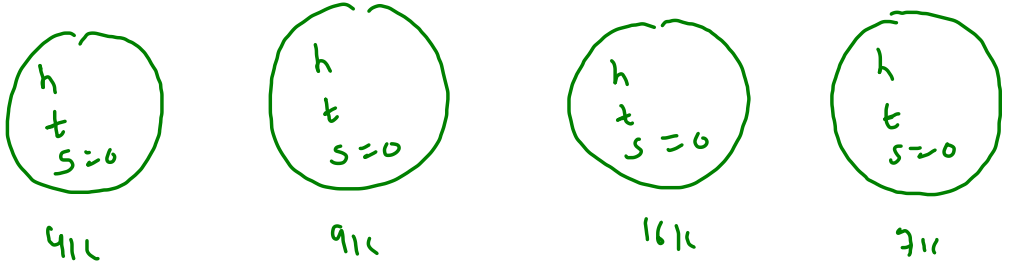


```java
private int size; // n
private LinkedList<HMNode>[] buckets; // N = buckets.length

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<>();
  }
}
```

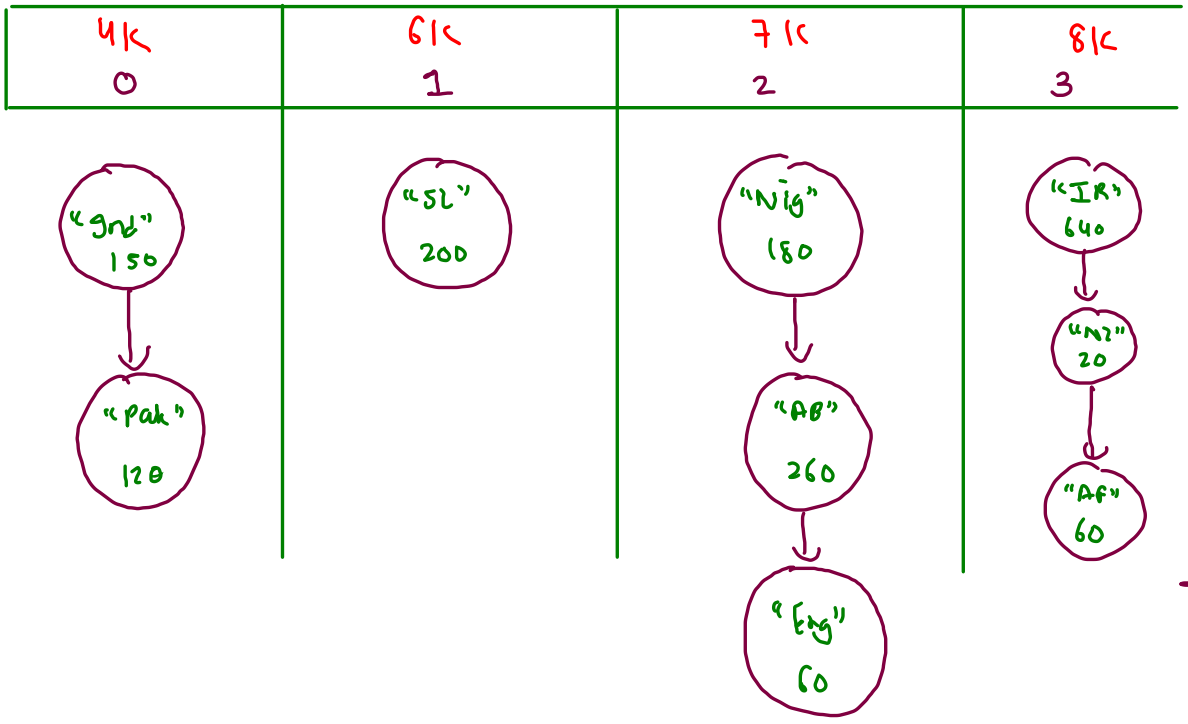
buckets = 191c

191c



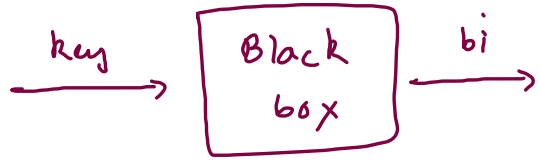


bucket



hm.put("Eng", 60);

hm.put("Pak", 120);



```java
public void put(K key, V value) throws Exception {
    int bi = getBucketIndex(key);
    int di = findWithinBucket(key,bi);

    if(di == -1) {
        //addition
        HMnode node = new HMnode(key,value);
        bucket[bi].add(node);
        size++;
    }
    else {
        //updation
        HMNode node = buckets[bi].get(di);
        node.value = value;
    }
}
```

```java
private int findWithinBucket(K key, int bi) {
    LinkedList<HMnode>list = buckets[bi];

    for(int i=0; i < list.size();i++) {
        HMNode node = list.get(i);
        if((node.key).equals(key) == true) {
            return i;x
        }
    }

    return -1;
}

private int getBucketIndex(K key) {
    int hc = key.hashCode();

    int bi = (Math.abs(hc)) % buckets.length;

    return bi;
}
```

bi = 0

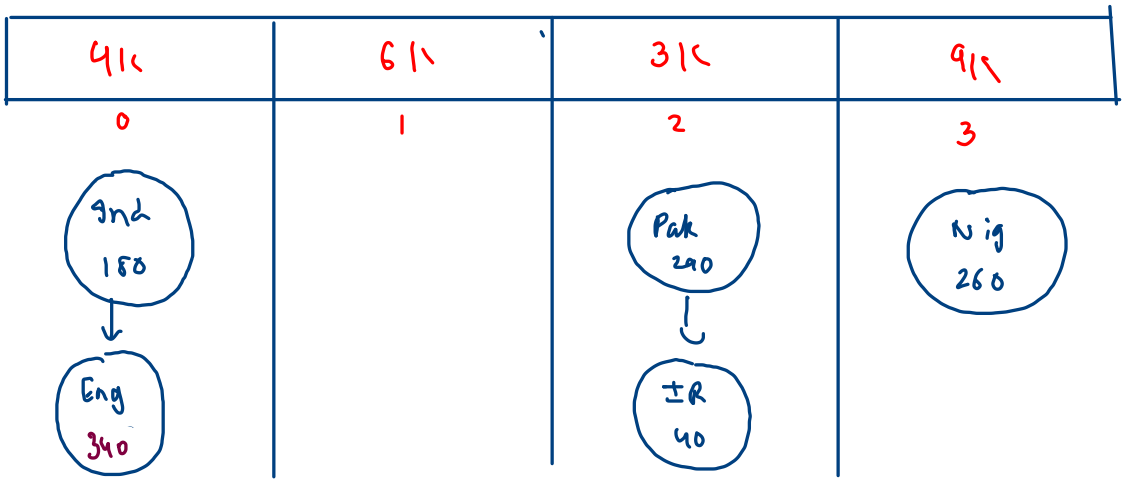
di = 1 s = ~~1~~ 2

hm.put ("Ind", 180);

hm.put ("Eng", 260);

⋮

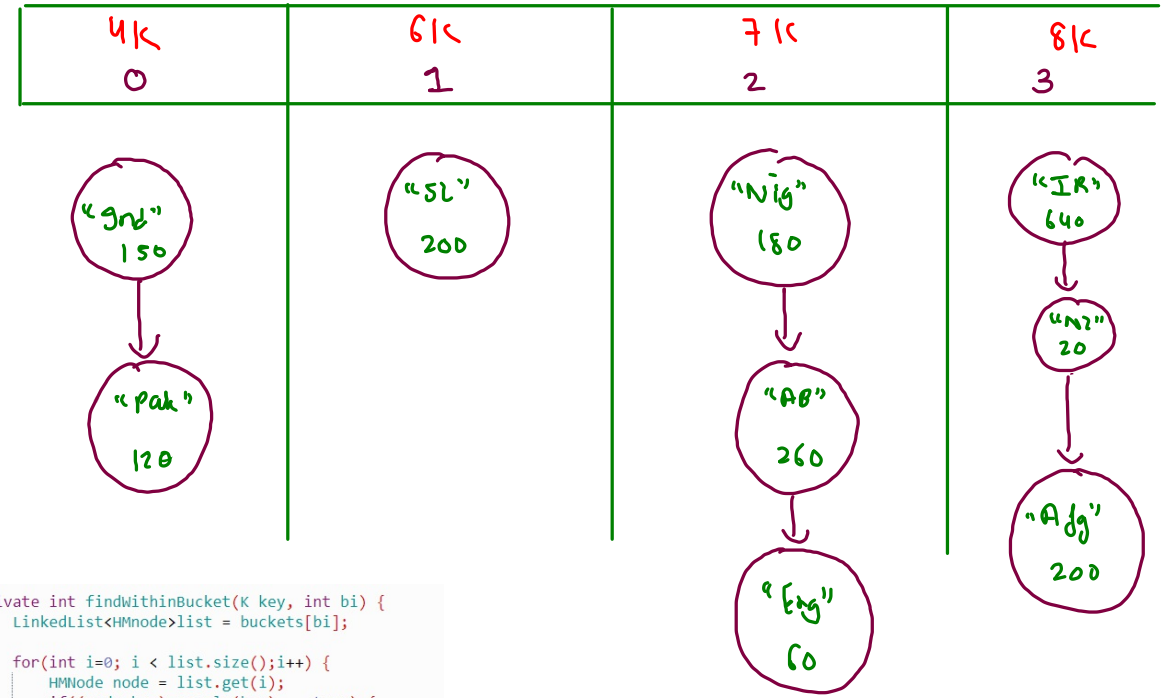
hm.put ("Eng", 340);



hc = 60, bi = 60%4 = 0

hc = 64, bi = 64%4 = 0

hc = 64, bi = 0




```java
public void put(K key, V value) throws Exception {
    int bi = getBucketIndex(key);
    int di = findWithinBucket(key,bi);

    if(di == -1) {
        //addition
        HMnode node = new HMnode(key,value);
        bucket[bi].add(node);
        size++;
    }
    else {
        //updation
        HMNode node = buckets[bi].get(di);
        node.value = value;
    }
}
```

O(1)

O(d)

O(d)

$d \leq 2$

size = 8

```java
private int findWithinBucket(K key, int bi) {
    LinkedList<HMnode>list = buckets[bi];

    for(int i=0; i < list.size();i++) {
        HMNode node = list.get(i);
        if((node.key).equals(key) == true) {
            return i;x
        }
    }

    return -1;
}

private int getBucketIndex(K key) {
    int hc = key.hashCode();

    int bi = (Math.abs(hc)) % buckets.length;

    return bi;
}
```

$d = \frac{8}{4} = 2$

↓ after addn

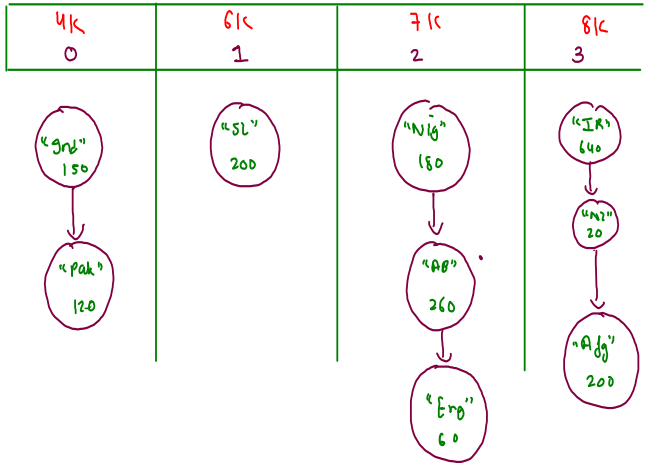
$d = \frac{9}{4} = 2.25$

$d = \frac{Size}{N}$

N -> buckets.length

d -> avg. no. of nodes per bucket

old buckets

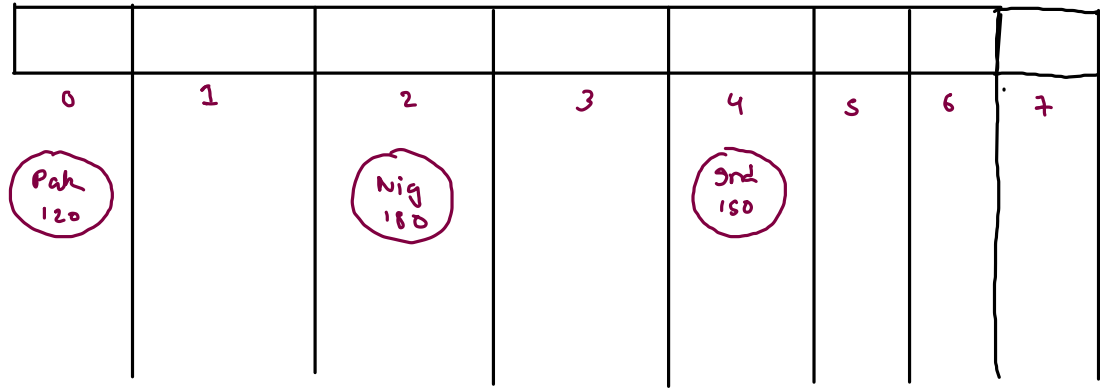


Ind → hc → 60 → bi → 60 %. 4 = 0

Pak → hc → 64 → bi → 64 %. 4 = 0

Nig → hc → 82 → bi → 82 %. 4 = 2

buckets



Ind → hc → 60 → bi → 60 %. 8 = 4

Pak → hc → 64 → bi → 64 %. 8 = 0

Nig → hc → 82 → bi → 82 %. 8 = 2