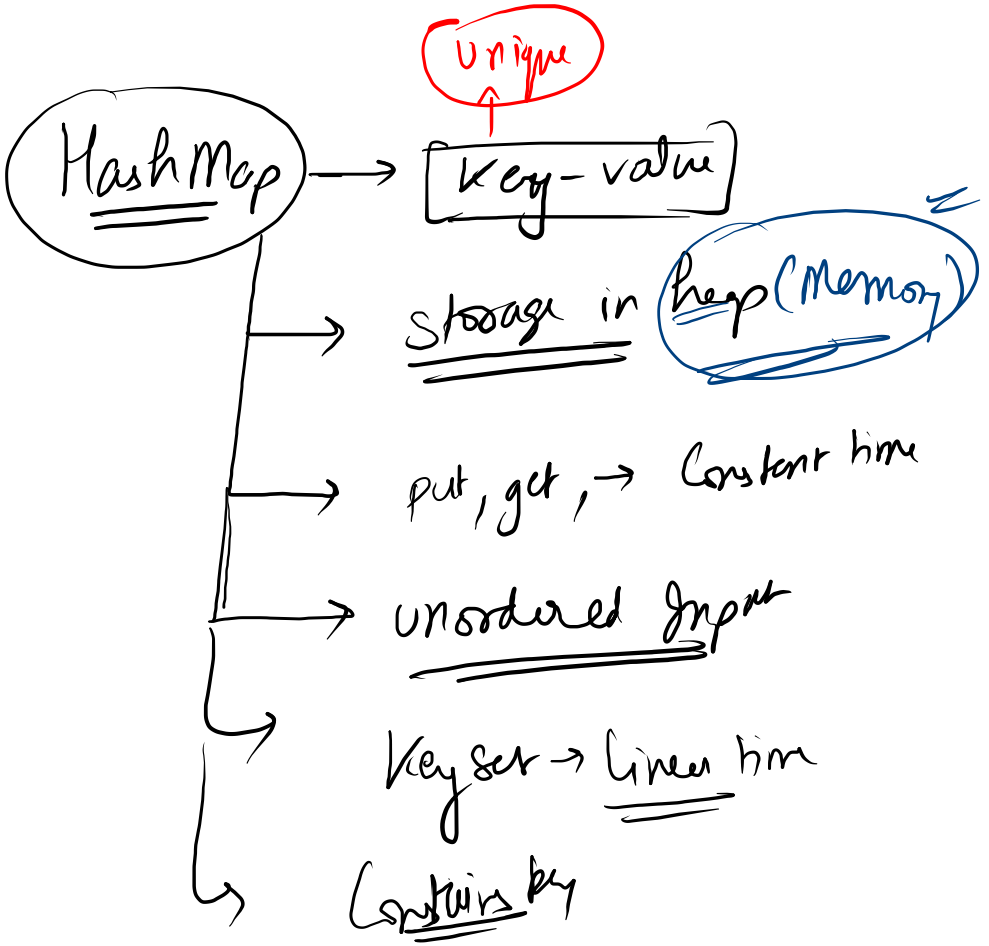
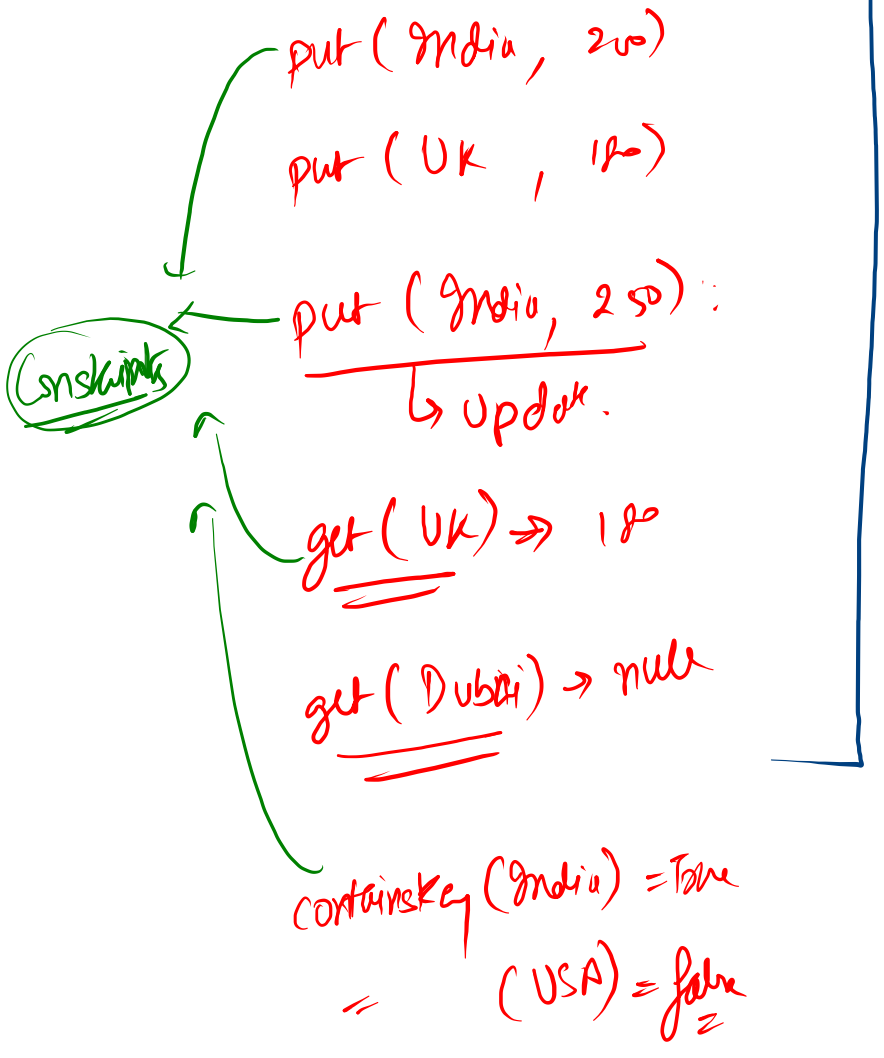
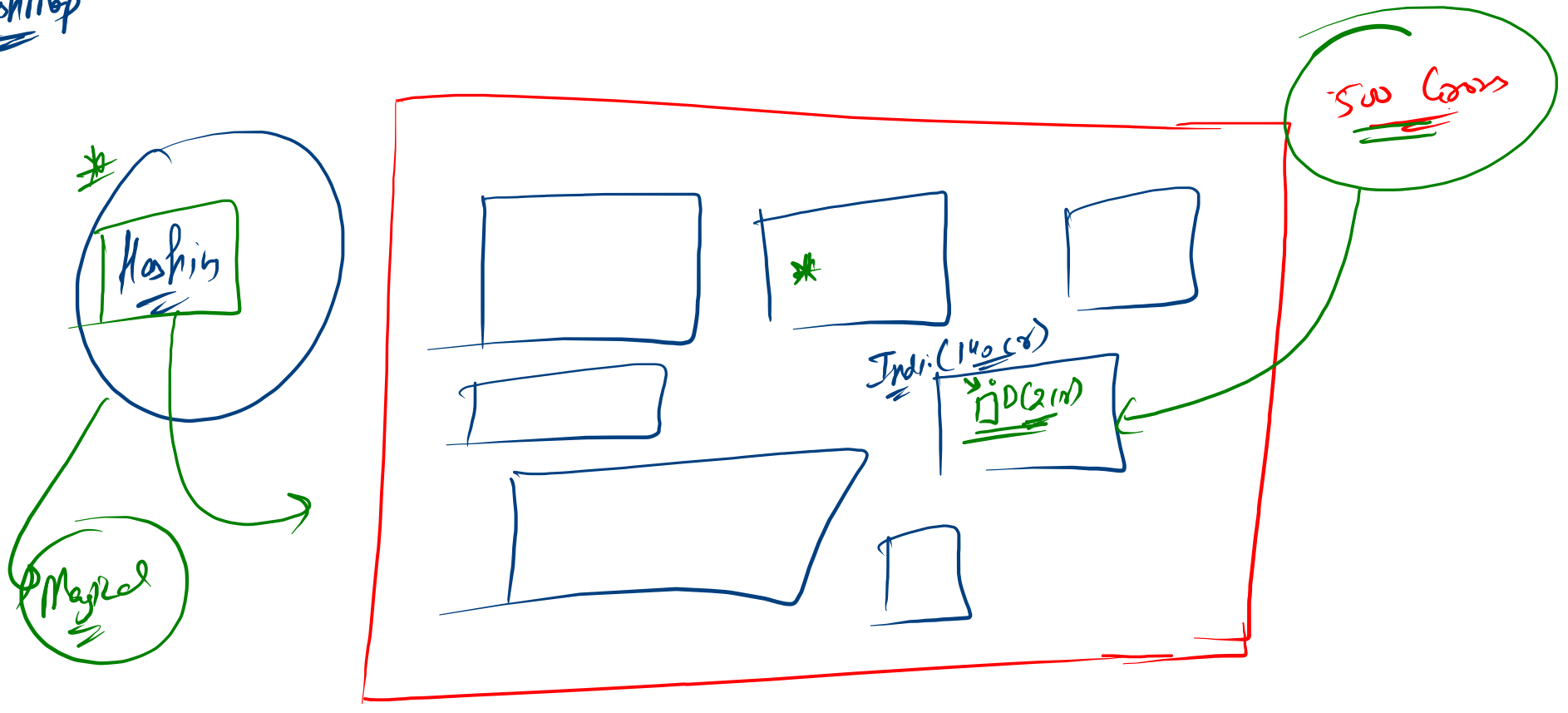

Unique
HashMap
Key-value
Storage in
heap (Memory)
put, get, → Constant time
unordered Input
KeySet → Linear time
Contains key


<String>
<Integer>
India
250
UK
18


put (India, 250)
put (UK, 18)
put (India, 250) :
↳ update.
ConcurrentHashMap
get (UK) ⇒ 18
get (Dubai) → null
containsKey (India) = True
(USA) = false

HashMap

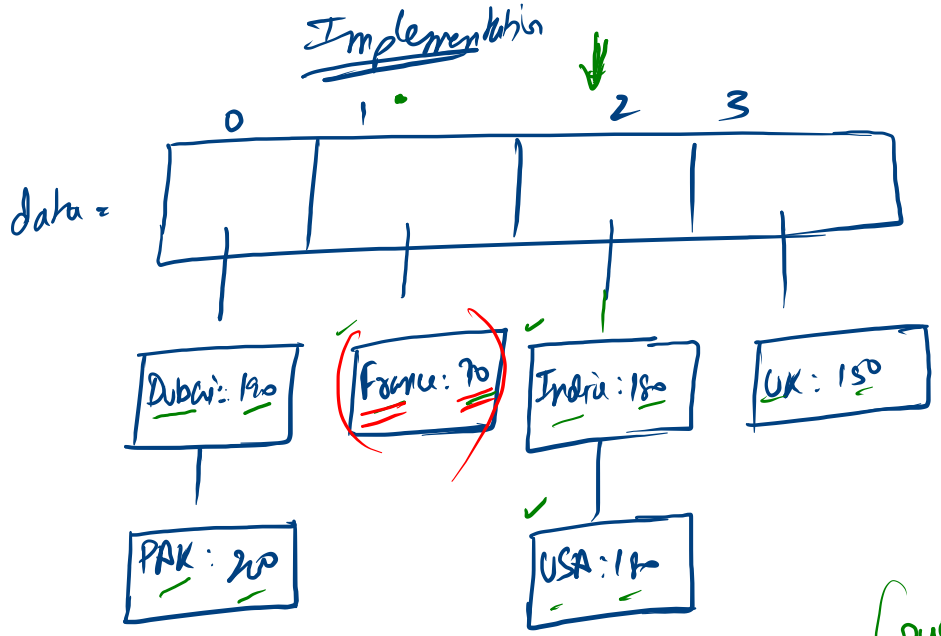


Constant

Visualize

K (String)	V (Integer)
India	200
UK	150
USA	180
France	70
PAK	20
Dubai	210

Implementation



ck("France") → 70

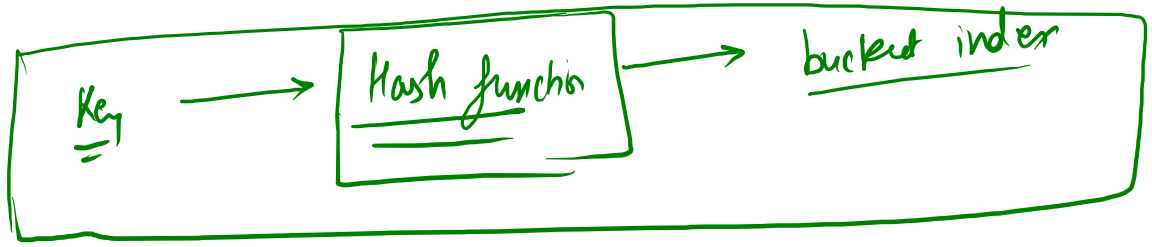
get("Iran") ⇒ null

put(India, 10);

put(key, value);

get(key);

ck(k)

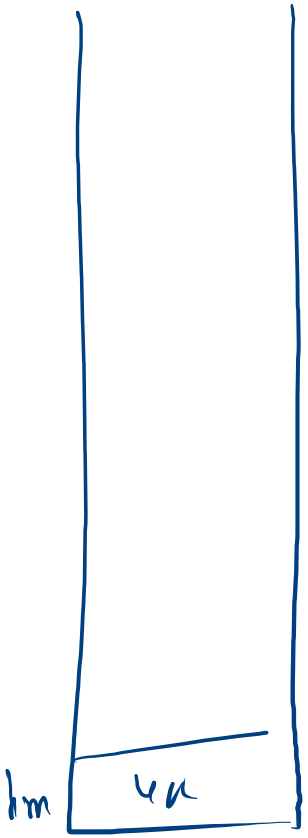


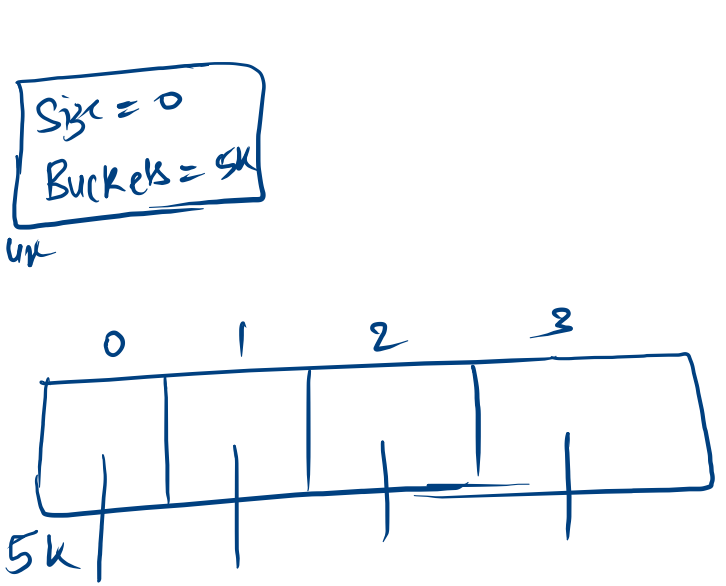
```java
public HashMap() {
    initbuckets(4);
    size = 0;
}

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

int arr[] = new int[5];

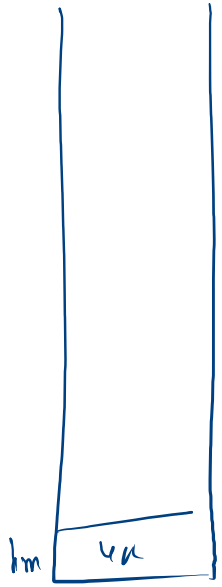
```java
public static class HashMap<K, V> {
    private class HMNode {}

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

    public HashMap() {}

    private void initbuckets(int N) {}

    public void put(K key, V value) throws Exception {}

    public V get(K key) throws Exception {}

    public boolean containsKey(K key) {}

    public V remove(K key) throws Exception {}

    public ArrayList<K> keyset() throws Exception {}

    public int size() {}

    public void display() {}
```

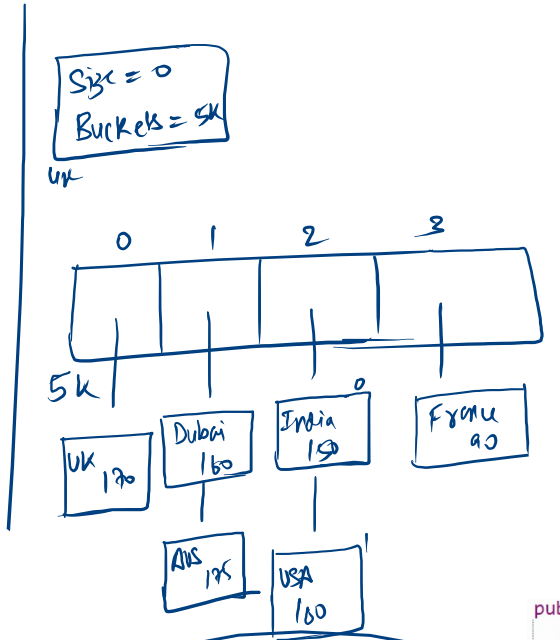
HashMap<String, Integer> hm = new HashMap<>();






```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 {}

  public V get(K key) throws Exception {}

  public boolean containsKey(K key) {}

  public V remove(K key) throws Exception {}

  public ArrayList<K> keyset() throws Exception {}

  public int size() {}

 public void display() {}
```






```
put("India",200);
put("France",90)
put("USA",100)
put("Dubai",160)
put("UK",170)
put("AUS",175)
put("India",150);
```

n → total elements = 6

N → no. of linked list = 4

$\lambda = \frac{n}{N} = (1.5)$

$\lambda < 2$

```
public void put(K key, V value) throws Exception {
    int bi = hashFunc(key);
    int di = findNodeByKey(bi,key);

    if(di == -1){
        // insert
        buckets[bi].addLast(new HMNode(key,value));
    }else{
        // update
        HMNode node = buckets[bi].getAt(di);
        node.value = value;
    }
}
```

put
get
Uk
remove
→ LL → avrage (λ)
≈ ($\lambda < 2$)

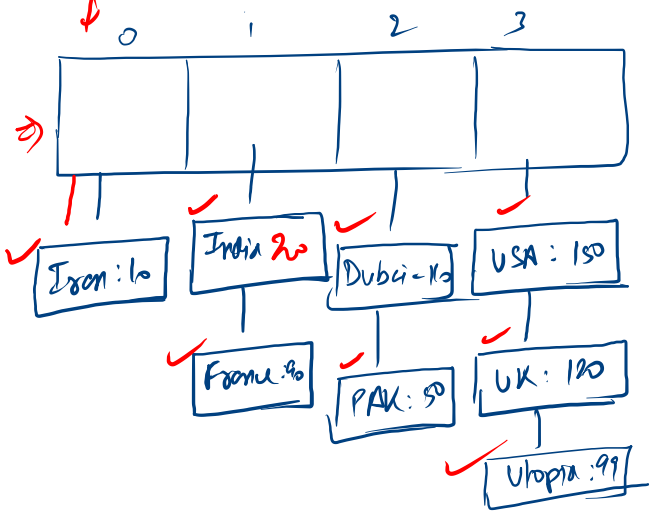
oper → Time ∝ λ

$\lambda = \frac{n}{4} \Rightarrow n = 8$

λ → avrage no. of elements in a linked list (bucket)

Hash Map → average Time ~~Complexity~~ → (Constant time).

put("India", 200)
= (USA, 150)
= (France, 90)
= (UK, 120)
= (Iran, 10)
= (Dubai, 110)
= (Utopia, 99)
= (PAK, 50)



value → hashcode → bi (unique)

India	65	1
USA	115	3
France	201	1
UK	167	7
Iran	32	0
Dubai	34	2
Utopia	55	7
PAK	6	6

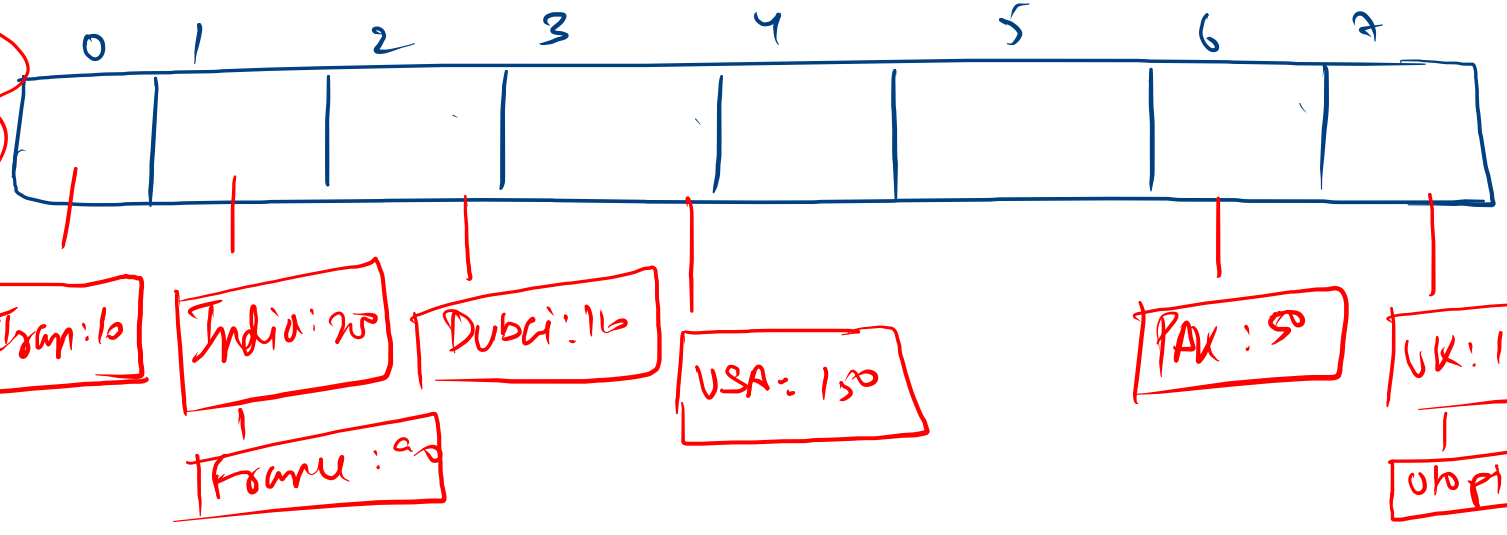
```
private int hashFunc(K  key){
    int hc = key.hashCode();
    int bi = Math.abs(hc%buckets.length);
    return bi;
}
```

HashMap
↳ efficient
↳ unavoidable

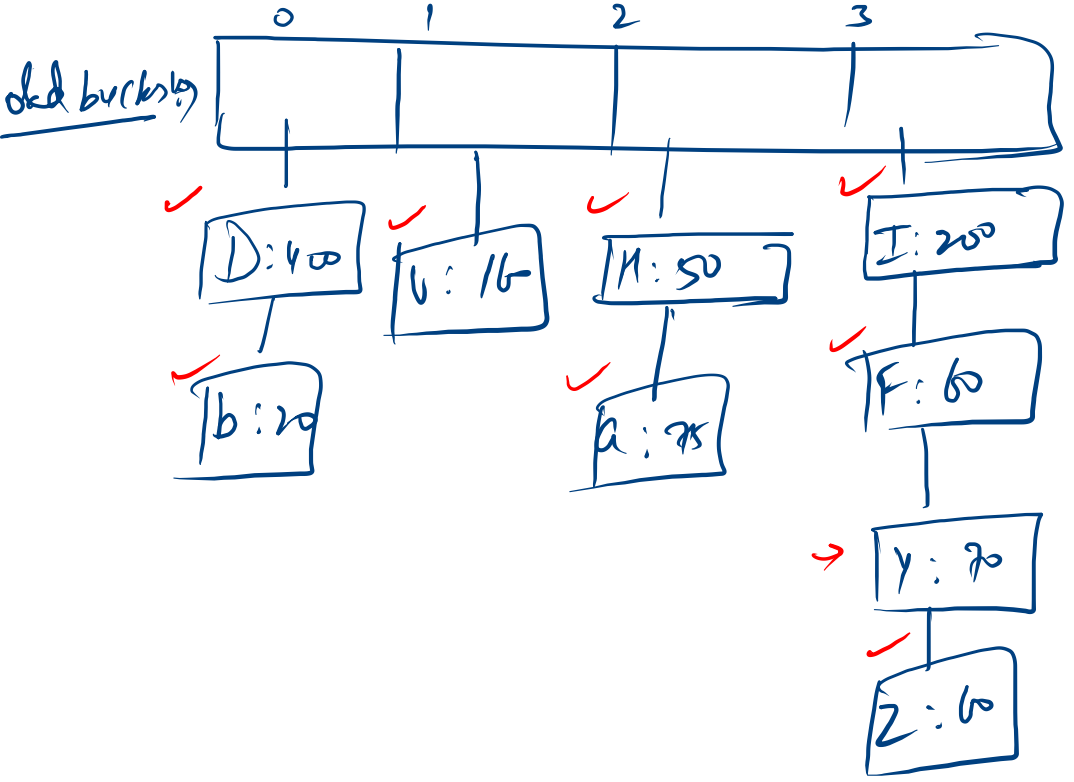
n: 8
N: 8
λ: 8/8

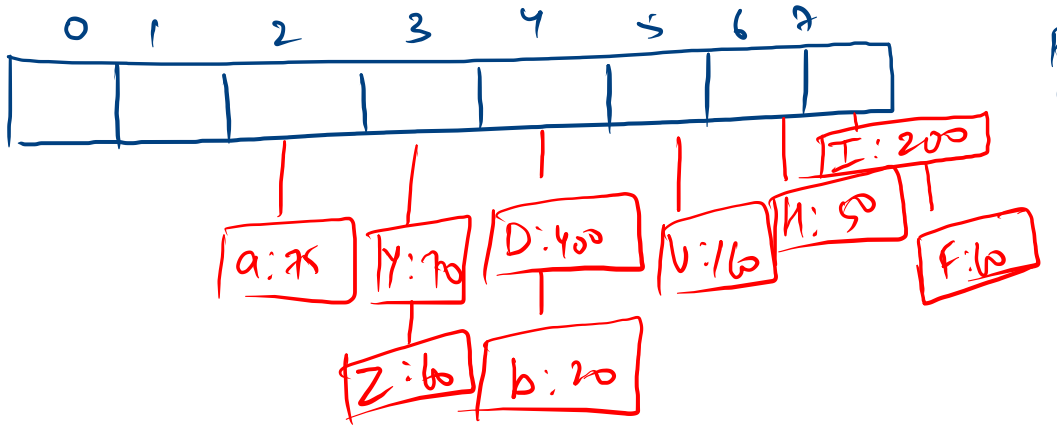
λ = 1
buckets = 8

λ < 2



n: 2
N: 4
λ = 0.5
λ = 0
2 → 2
2 → 2
2 → 2





I → 7
D → 12
H → 6
F → 15

U → 21
Y → 3
Z → 29
a → 2
b → 20

put(I, 200) → λ

put(D, 400) → λ

put(H, 50) → λ

put(F, 60) → λ

put(U, 16) → λ

put(Y, 70) → λ

put(Z, 60) → λ

put(a, 75) → λ

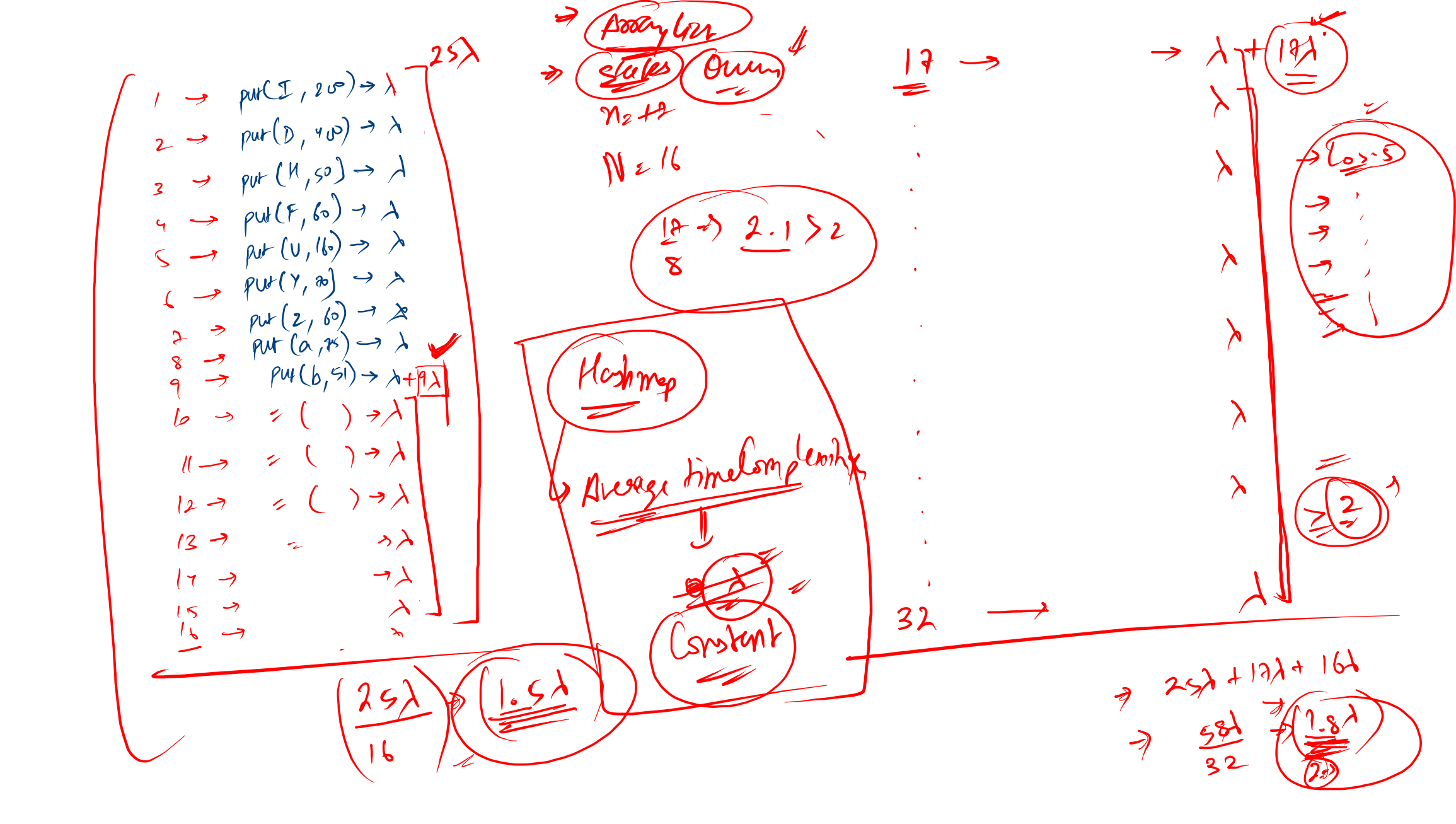
put(b, 51) → λ+9λ

n = 8 9

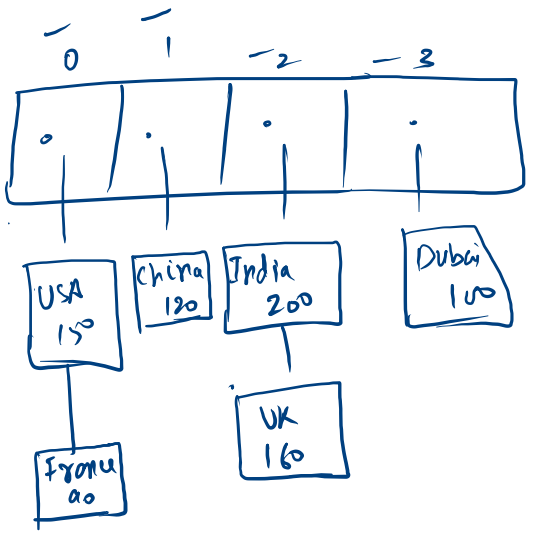
N = 8

λ = (1.1)

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

    if(di == -1){
        // insert
        buckets[bi].addLast(new HMNode(key,value));
        size++;
    }else{
        // update
        HMNode node = buckets[bi].get(di);
        node.value = value;
    }

    double lambda = (size*1.0)/buckets.length;

    if(lambda > 2.0){
        rehash();
    }
}
private void rehash() throws Exception {
    LinkedList<HMNode>[] oldBuckets = buckets;

    initbuckets(oldBuckets.length*2);
    size = 0;
    for(LinkedList<HMNode> ll : oldBuckets){
        for(HMNode node : ll){
            put(node.key,node.value);
        }
    }
}
private int hashFunc(K  key){
    int hc = key.hashCode();
    int bi = Math.abs(hc%buckets.length);
    return bi;
}
```


Array List
States
Queue
n_2+2
$N = 16$
$\frac{17}{8} \rightarrow 2.1 > 2$
HashMap
Average time complexity
Constant
$\frac{25\lambda}{16} \rightarrow 1.5\lambda$
25λ
$17 \rightarrow$
18λ
$32 \rightarrow$
$25\lambda + 17\lambda + 16\lambda$
$\frac{58\lambda}{32} \rightarrow 1.8\lambda$

Size = 1




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

    if(di == -1){
        // insert
        buckets[bi].addLast(new HMNode(key,value));
        size++;
    }else{
        // update
        HMNode node = buckets[bi].getAt(di);
        node.value = value;
    }
}
```

```java
public V remove(K key) throws Exception {
    int bi = hashFunc(key);
    int di = findNodeByKey(bi,key);

    if(di == -1){
        return null;
    }else{
        HMNode node = buckets[bi].removeAt(di);
        size--;
        return node.value;
    }
}
```

```java
public V get(K key) throws Exception {
    int bi = hashFunc(key);
    int di = findNodeByKey(bi,key);

    if(di == -1){
        return null;
    }else{
        HMNode node = buckets[bi].getAt(di);
        return node.value;
    }
}
```

```java
public ArrayList<K> keyset() throws Exception {
    ArrayList<K> list = new ArrayList<>();

    for(LinkedList<HMNode> ll : buckets){
        for(HMNode node : ll){
            list.add(node.key);
        }
    }

    return list;
}
```

```java
public boolean containsKey(K key) {
    int bi = hashFunc(key);
    int di = findNodeByKey(bi,key);

    if(di == -1){
        return false;
    }else{
        return true;
    }
}
```

```java
private int hashFunc(K  key){

}
```

```java
private int findNodeByKey(int bi , K key){
    LinkedList<HMNode> ll = buckets[bi];

    for(int idx = 0 ; idx < ll.size() ; idx++){
        HMNode node = ll.getAt(idx);

        if(key.equals(node.key)){
            return idx;
        }
    }

    return -1;
}
```