Hashmap & Heap

DSA classes

Order of Content

1. → Hashmap & Heap → level 1 (without construction)
2. → Graphs - Lecture (1)+(2)
3. → Hashmap & Heaps + Arrays & Strings → level 2 + construction
   - # Chaining
   - # Open Addressing
   - # Good Hashfunction, Bad Hashfunction
4. → Bit Manipulation → Competitive
5. → Number Theory → Competitive
6. → Trees - level 2 of remaining lectures
   - (Revising DP)
   - DP on trees
   - Self Balancing BST (AVL)

Data Structures

	Arrays/Strings	# hh(PHR)
① add (insert) → O(1)	O(1) last, O(n) first	O(1) f, O(1) last
② remove (delete) → O(1)	O(1) last, O(n) first	O(1) f, O(1) last
③ update → O(1)	O(1) {indexing}	O(n) {indexing}
④ search (find) → O(1)	O(n) LS, O($\log_2 n$) BS	O(n) LS
⑤ display/Traverse	O(n)	O(n)
⑥ length/size	O(1)	O(1)
⑦ read (get) → O(1)	O(1) {indexing}	O(n) worst case


Team (String)
IPL Trophies count (Integer)
Team (String)
Years of winning (ArrayList<Integer>)
Java
Hashmap
key-value pairs
Read
Create
Update
Delete
Find
Class & objects
C++
Unordered_map
Python
Dictionary
Json
key: value
(String) Key → Country
(Long) Value = Population

Hashmap

```
# HashMap < key, value > hm = new HashMap<>();
```

1. insert
2. update

→ put → O(1)

(key found / key found)

3. delete → remove → O(1)
4. find/read → get → O(1)
5. search → containsKey → O(1)
6. size() → O(1) {k-v pairs}
7. display → Syso(hm); → O(n)
8. keyset → O(n)

↓

Set < key >

function, variable name

↓ ↓

Java: getOrDefault() populationOfCountry

C++: get_or_default() population_of_country

HashMap →
- Order of insertion ≠ Order of key-value pair
- ≠ Sorted order based on key/value

Unordered
no indexing

Number of Key-value pairs
= Number of keys

Unique Keys
↳ put → Insert (key ∄)
→ Update (key ∃)

1 key → 1 entry { No duplicates keys will be present }

Common Elements / Intersection

{ 2, 1, 1, 3, 4, 2, 5, 1, 2, 5 }

{ 1, 1, 4, 4, 4, 5, 5, 3 }

LC 349

{ 1, 3, 4, 5 }

LC 350

{ 1, 1, 3, 4, 5, 5 }

Brute force

Sort Both Arrays + Two pointer

$N_1 \log N_1 + N_2 \log N_2 + (N_1 + N_2)$

Optimized Hashmap

{ 2, 1, 1, 3, 4, 2, 5, 1, 2, 5 }

{ 1, 1, 4, 4, 4, 5, 5, 3 }

"1,4,5,3"

HashMap<Integer, Integer>

Element → Frequency

T, W, T → 9:45 PM - 11:45 PM (2hr)

S, S → 9 AM - 12 PM (3hr)

↓ 2-3 weeks

T, W, T → 9:15 PM - 11:15 PM (2hr)

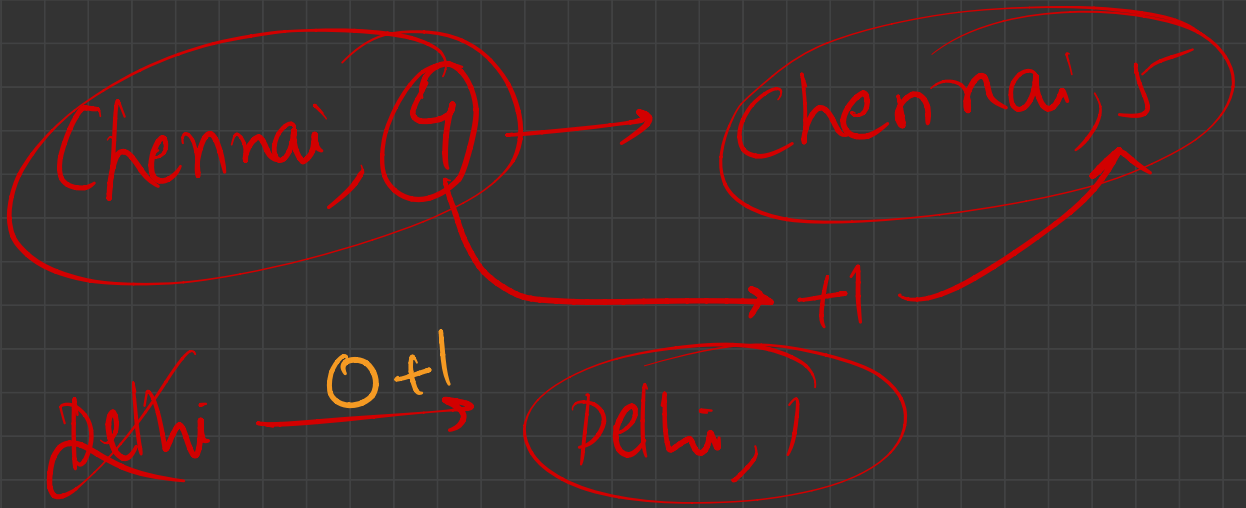
S, S → 9 AM - 12 PM (3hr)

```
if(freq.containsKey(nums1[i]) == true){
    freq.put(nums1[i], freq.get(nums1[i] + 1);
} else {
    freq.put(nums1[i], 1);
}
```

1

2

```
freq.put(nums1[i], freq.getOrDefault(nums1[i], 0) + 1);
```



349 LC

```
HashMap<Integer, Integer> freq = new HashMap<>();

for(int i=0; i<nums1.length; i++){
    // if(freq.containsKey(nums1[i]) == true){
    //     freq.put(nums1[i], freq.get(nums1[i] + 1);
    // } else {
    //     freq.put(nums1[i], 1);
    // }

    freq.put(nums1[i], freq.getOrDefault(nums1[i], 0) + 1);
}

ArrayList<Integer> intersection = new ArrayList<>();

for(int j=0; j<nums2.length; j++){
    if(freq.containsKey(nums2[j]) == true){
        intersection.add(nums2[j]);
        freq.remove(nums2[j]);
    }
}

int[] res = new int[intersection.size()];
for(int i=0; i<res.length; i++)
    res[i] = intersection.get(i);
return res;
```

N1

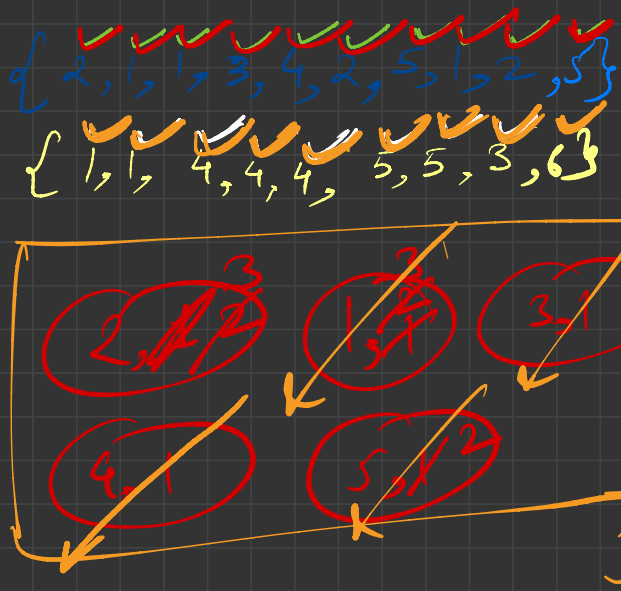
O(1)

O(1)

N2

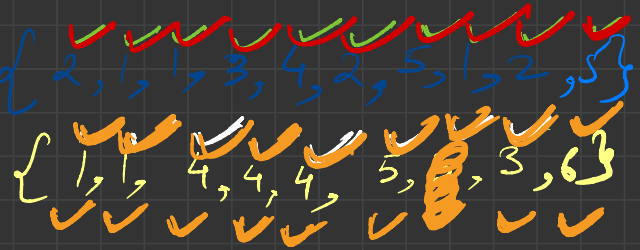
O(1)

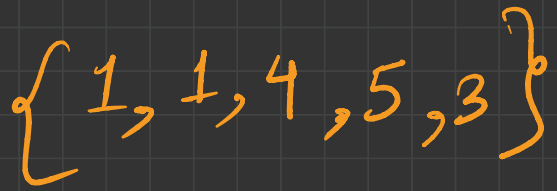
O(1)

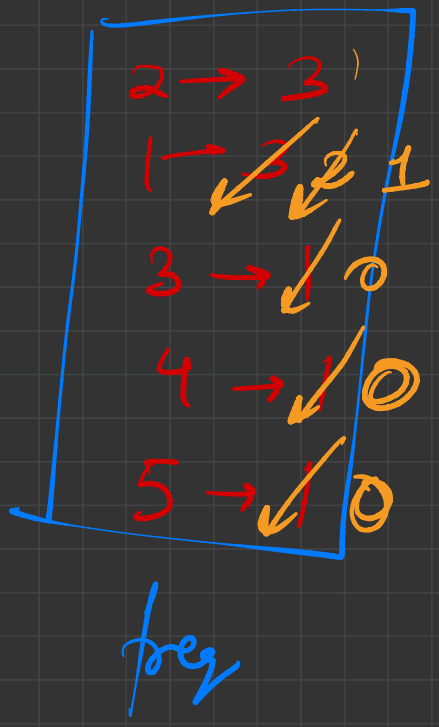


{1, 4, 5, 3}

O(N1+N2) Time , O(N1) extra Space (Hashmap)






freq

```java
HashMap<Integer, Integer> freq = new HashMap<>();

for(int i=0; i<nums1.length; i++)
    freq.put(nums1[i], freq.getOrDefault(nums1[i], 0) + 1);

ArrayList<Integer> intersection = new ArrayList<>();

for(int j=0; j<nums2.length; j++){
    if(freq.containsKey(nums2[j]) == true && freq.get(nums2[j]) > 0) {
        intersection.add(nums2[j]);
        freq.put(nums2[j], freq.getOrDefault(nums2[j], 0) - 1);
    }
}

int[] res = new int[intersection.size()];
for(int i=0; i<res.length; i++)
    res[i] = intersection.get(i);
return res;
```

Frequency Sort

Sort String by Frequency

a b c a b d d c d b e

0 1 2 3 4 5 6 7 8 9 10

N = 11

HashMap<Character, Integer> freq = new HashMap<>();

key, value

get-value

a → 1 2

b → 1 2 3

c → 1 2

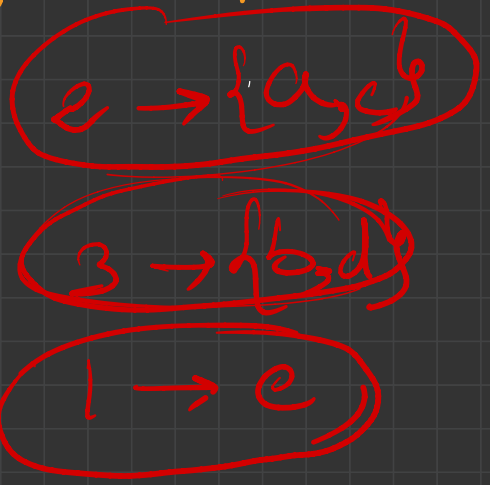
d → 1 2 3

e → 1

Expected Output: bbb ddd aa cc e

3 2 1

Hashmap <Integer, Arraylist <Character>>



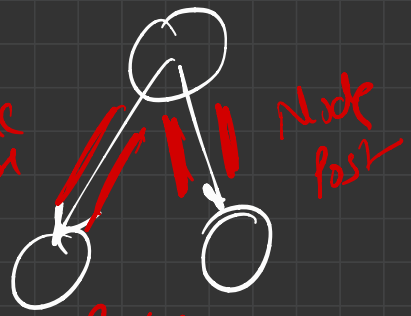
get key

bbb ddd aac e

max-N

11 → { }
10 → { }
9 → { }
8 → { }
7 → { }
6 → { }
5 → { }
4 → { }
3 → {b, d}
2 → {a, c}
min-1 → 1 → {e}

Recursion
↳ DFS
↳ Trees



Graph
① Hashmap (Use eg)
② Trees → DFS { Inorder & Post }
③ Arraylist / OOPS

```java
HashMap<Character, Integer> freq = new HashMap<>();
for(int i=0; i<s.length(); i++){
    char ch = s.charAt(i);
    freq.put(ch, freq.getOrDefault(ch, 0) + 1);
}

HashMap<Integer, ArrayList<Character>> rev = new HashMap<>();
for(Character ch: freq.keySet()){
    int f = freq.get(ch);
    if(rev.containsKey(f) == false)
        rev.put(f, new ArrayList<>());
    rev.get(f).add(ch);
}

StringBuilder res = new StringBuilder("");
for(int f=s.length(); f>=1; f--){
    if(rev.containsKey(f) == true){
        for(Character ch: rev.get(f)){
            for(int i=0; i<f; i++)
                res.append(ch);
        }
    }
}

return res.toString();
```