BLIND ~ 75

- ★
- ★★
- ★★★

#	Question.	Topic	Pg No.
1	Check if array contains duplicate	Arrays and hashing.	3
2	Check if 2 given strings are valid anagrams		4
3	Two Sum :- Find 2 numbers that add up		4
4	Grouping anagrams :- Find the anagrams and group them together	Array, Hash map, bucket sort, PQ	6
5	Top k frequent elements	Arrays, math	8
6	Product of array except self.	Priority queue	13
7	K^{th} Largest element	PQ	15
8	Last stone weight : Pick the 2 heaviest and combine.	PQ	17
9	Return the 'k' closest points to the origin.	Math, Hashing	18
10	Task scheduler: Return min units of time req to complete tasks	Boolean algebra, Recursion, bit masking	20
11	Reversing bits of an integer	Math, matrix 2D	22
12	Rotate an image in place	Math 2D matrix	24
13	Spiral matrix, with steps		29
14	Insert interval	Linked list	30
15	Fibonacci numbers with memoization	Recursion, dyn. memoization prog.	31
16	House robber problem	Dyn Prog, Rec. reln.	32
17	Maximum sum subarray	Brute force is $O(n^2)$ Optim Kadane $O(n)$	34
18	Best time to buy and sell stock.	Brute : $O(n)$ Optim : Kadane	37
19	Climbing stairs,	Recur : $O(2^n)$ DP: $O(n)$ Rec + memo : $O(n)$	40
20	Min cost to climb stairs	DP: Rec reln.	41
21	House robber	DP : Rec Reln	42
22	Meeting rooms II	PQ	42,43
23	Jump game II	Greedy	43,44
24	Implement a trie	Trie	44
25	Pairs of strings with same prefix	Trie $O(n)$	44.
26	Reversing a linked list	Iteration, recursion.	47
27	Merging 2 sorted linked list	Traversing a LL	48

Arrays and Hashing - Revision.

① #217 Contains Duplicate

[1, 2, 3, 4, 5, 1, 2]

Return true if contains duplicate

Approach 1 → - Sort the array
- Iterate and check.

Tc :- - Sorting is $O(n \log n)$ ★
- Searching is $O(n)$

∴ Tc is $O(n \log n)$

Sc :- $O(1)$

Approach 2 → - Have a hash map
- check in HM as you iterate

Tc → $O(n)$ → searching in HM is $O(1)$

Sc → $O(n)$

② Valid Anagram

Q: given 2 strings check if one is te anagram of the other.

"abc" "def"

M_1 M_2

M_1 M_2

a b c d e f

If deleting characters from M_1 based on char from M_2 makes M_1 0 then it is anagram.

---

③ Two sum ②

Given an array of numbers

Find the indices that add upto given value. We cant sort

[2 7 11 15] : 9

ARR T

4+5 = 7

[3, 2, 4] : 6

i j

Approach 1 $O(n^2)$ Runtime.

```
for i in range (len(nums))
    for j in range (i+1, len(nums))
        check i+j == Target.
```

Approach ② $O(n)$ Runtime

2 pass hash table

- 1st pass to store values in a table
- 2nd pass to check if compliment exists.

Approach 3 :- 1 pass hash table

- Have a hash table TC $O(n)$
- as you iterate keep checking if value exists a complimentary value

Character count pattern

Hash map & sorted string : [achal/str]

④ Grouping anagrams :

Q:- Given a array of words group the anagrams.

Anagrams :- Words with same characters.

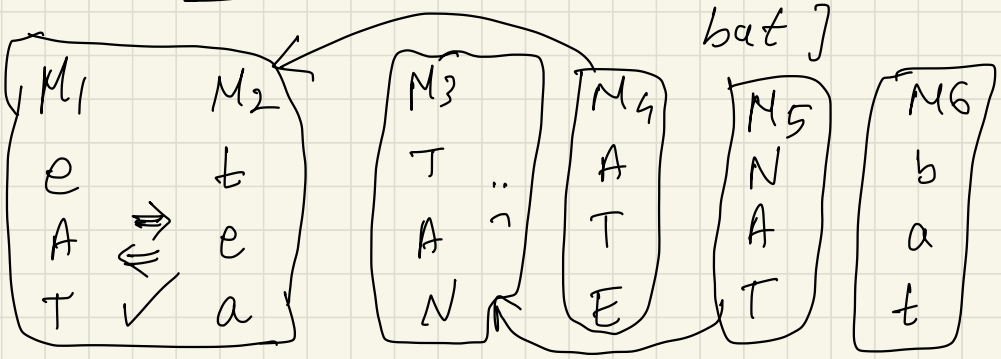
concept :- Hashing . char count

Brute force

$O(n^2)$ Each word check other words.

Approach 2

[eat, tea, tan, ate, nat, bat]



Maps = $O(n)$

checking is $O(n)$ $\therefore O(n^2)$

Sort the strings

[aet, aet, ant, aet, ant, abt]

(Use map to store old values)

Map

aet :- [eat, tea, ate]

abt :- [bat]

ant :- [tan, nat]

TC :- N : length of strings arr.

k : length of longest string

log k : time for sorting the string.

$O(N \cdot k \cdot \log k)$

Approach 3 :- Using number count

- Have a map of string : arraylist
- for each string have an array of size 26 (representing 26 alphabets)
- Set the count in array based on frequency of occurance of characters
- Convert array to string representation.
  #count a #count b #count c ....
- add this and string to map
- Return values at the end.

---

⑤ # 347 Top k frequent elems.

Q:- Given array of nums, find top k frequent nums.

[1, 1, 1, 2, 2, 3]

C

- 10^5 nums
- negative nums
- There will be k unique nums

No info about sort.

TC better than $O(n \log n)$

App 1

Frequency count

$O(n)$

basically $O(n)$ or $O(\log n)$

TC

Naive approach

① num → Map → freq. $O(n)$

② Sort (Map, freq) $O(N \log N)$

③ loop (Map, k) $O(k)$

SC

$O(n)$

as the hash map is sorted when we loop, we can access top k or bottom k elements.

Optimizing this approach

For problems that talk about top k or bottom k elements

it is a good idea to use

Heap

In this problem we only care about the top k most frequent elements.

We are going to use min heap.

Min Heap - Priority queue

is the data structure for creating a heap.

1. Complete binary tree
2. Each node is ≤ value of children
   - We use min heap as we want to keep track of highest freq elems.

Eg:- [1, 1, 1, 2, 2, 3, 4], k = 2

① Make the hash map.

{1 : 3, 2 : 2, 3 : 1, 4 : 1}

Here the focus is on the frequency.

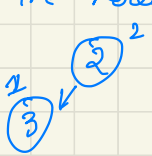
Look into how min heap works

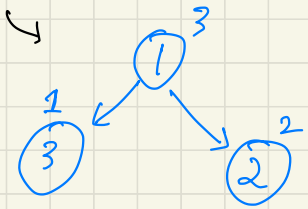
① Insert node 1 with value of 3.

② Insert 2 with value of 2

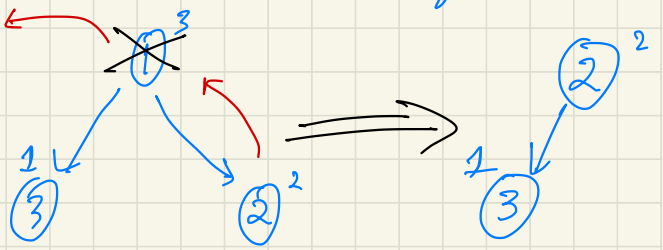
③ since min heap has property 2 the nodes flip.

④ Insert 3,

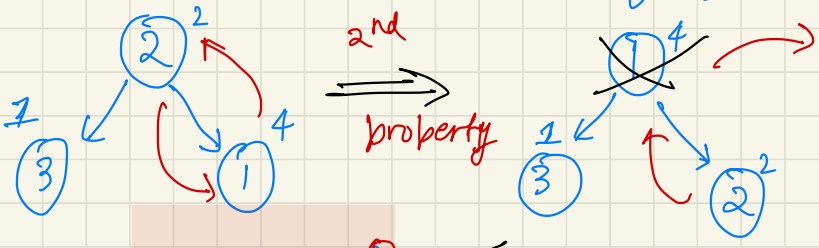


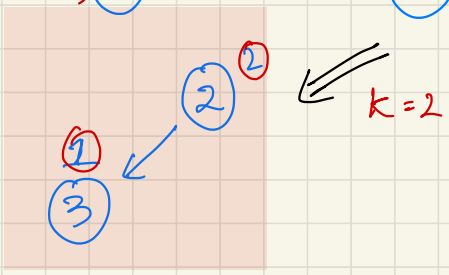


⑤ Now since we only need top 2 frequent elements, we cut down the elements in the heap and keep only top 2 elements.



⑥ Insert 4 that has freqency of 1

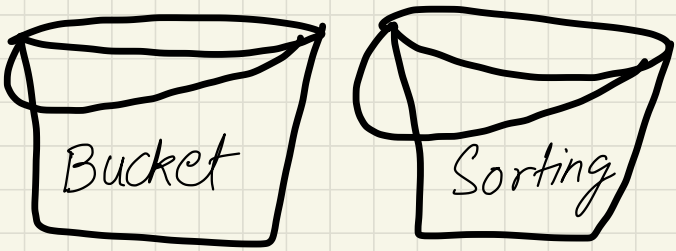




⑦ Extract the elements from the heap.

[2, 1]

Approach 3



In an array of elements of size N.

Each element can have a frequency of 1 or N.

∴ We create N buckets to hold the various elements.

Step 1 :- Create N buckets (an array of arrays)

Space complexity O(n)

TC :- O(1)

Step 2 :- Create a hash map and make note of all frequencies.

TC :- O(n)

SC :- O(n)

Step 3 :- Iterate through HM and add nums to bucket matching the frequency.

TC :- O(n)

No SC for this step.

Step 4 :- Reverse iterate the bucket array and pour all elements into a single array.

TC

O(n)

SC(O(n))

Step 5 :- Iterate through array list / array and return top k elems.

TC :- O(k)

Total TC is O(N)

(6) Product of array except self

#238

→ given an array of numbers, return an array that has product of all numbers except itself.

→ Do not use division.

→ Write alg in O(n) TC.

Eg:- [-1, 1, 0, -3, 3] [1, 2, 3, 4]

out [0, 0, 9, 0, 0] [24, 12, 8, 6]

Brute force without using division

Approach 1

① create an array → O(n) SC
keep filling with left product

② Iterate from other end.
create a right product array

③ Combine both arrays.

O(1) Space complexity

Approach 2

Last problem, we had O(n) SC as we created 2 separate arrays and then merged them at the end.

Here we do it with 1 single output array.

Eg:- [-1, 1, 0, -3, 3]

lp 1 rp 1

L [1, -1, -1, 0, 0]

R [0, 0, -9, 3, 1]

out [0, 0, 9, 0, 0]

7) K^{th} Largest element #703

703. Kth Largest Element in a Stream

Easy 3.6K 2.1K

Amazon Facebook Google ...

Design a class to find the k^{th} largest element in a stream. Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

Implement `KthLargest` class:

- `KthLargest(int k, int[] nums)` Initializes the object with the integer `k` and the stream of integers `nums`.
- `int add(int val)` Appends the integer `val` to the stream and returns the element representing the k^{th} largest element in the stream.

Example 1:

Input
```
["KthLargest", "add", "add", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
```
Output
```
[null, 4, 5, 5, 8, 8]
```

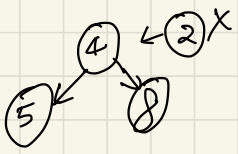
Explanation
```
KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);
kthLargest.add(3);   // return 4
kthLargest.add(5);   // return 5
kthLargest.add(10);  // return 5
kthLargest.add(9);   // return 8
kthLargest.add(4);   // return 8
```

```java
class KthLargest {

    public KthLargest(int k, int[] nums) {

    }

    public int add(int val) {

    }
}

/**
 * Your KthLargest object will be instantiated and called as such:
 * KthLargest obj = new KthLargest(k, nums);
 * int param_1 = obj.add(val);
 */
```

Solution Approach 1

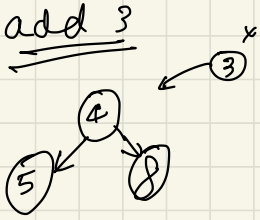
- We maintain a heap that only holds the top k elements.
- This must be a min heap - the smaller element is tossed out.
- [[3, [4, 5, 8, 2]], [3], [5], [10], [9] [5]]

k = (3). We can hold 3 elements.

(1) 

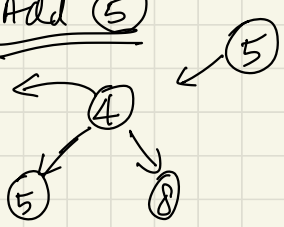
2 is added and also popped out.

(2) add 3



3 is popped out
So 4 is the 3rd largest element in the heap.

(3) Add (5)



When 5 is added 4 is popped out
∴ 5 is 3rd largest element present.

Likewise
keep adding, popping and Return the kth largest elem present.

(8) #1046 Last Stone Weight

- Int array of stone weights.
- In each turn choose 2 heaviest stones and combine them together.

based on some rule.

- At the end there is only 1 left

Return its weight.

- If none remains, ret = 0.

Eg:- (2, 7, 4, 1, 8, 1)

(1) Construct a max priority queue.

The highest element will be polled first.

(2) Poll 2 elements and add the combined result.