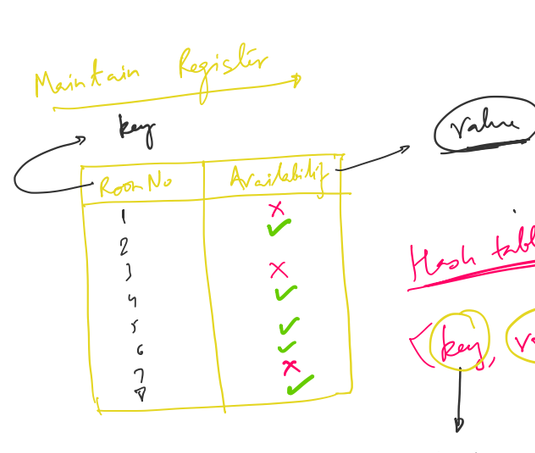
Hashing 1 October

Monday, 25 October 2021 9:04 PM

Hashing 1

⊙ Maintain Register


key
Room No
Availability
1
2
3
4
5
6
7
8
Value

Hash table

⟨ key, value ⟩

key → unique identifiers

value → useful info associated with the keys

< room no, { Availability, type of the room, ⋮ } >

Q Given an array. Find the frequency of an element x in the array.

A : 1, 2, 3, 5, 3, 5, 1, 5

Q { x = 5, 3, 18, 10 }

1 Q: TC: O(N), SC: O(1)

Q q's: TC: O(Q.N), SC: O(1)

Sort the array: N log N

A : 1, 1, 2, 3, 3, 5, 5, 5

N, Q <= 10^5

1 Q → TC: O(log N)

Q q's → TC: O(Q. log N + N log N) $10^5 \times 17 + 10^5 \times 17$ 34 x 10^5 3 x 10^6

SC: O(1)

10^8

Q If the elements are small   0 < A <= 100.

⊙ Maintain a frequency array!

A : [1, 2, 1, 5, 3, 1, 3]

freq: | 0 | 2 | 1 | | 3 | 0 | 1 | (index 0 1 2 3 4 5)

```
f (i: 0 → n-1)
    freq [A[i]] ++;
```

TC: O(N)   freq: [x]

Q q's = O(Q)   1 Q → O(1)

5   ret freq [5];

3   ret freq [3];

TC : O(N + Q)

SC : O(Max A_i)

Key : Array value (x)
Value : freq of x.

Q   -100 <= A_i <= 100

0 <= A_i+100 <= 200

|freq| → 201

∀ i   freq [A_i + 100] ++;

A_i = 3

Q : 3   ret freq [x + 100]

Key : x+100
value : freq of x

TC : O(N)

SC : O(max A_i - min A_i)

-x <= A_i <= x   range : x - (-x) = 2x

⊙ Space ✗ -10^9 <= A_i <= 10^9 key 10^7

⊙ Hash function

It converts a data item into a key such that it can fit into the space available to us.

x → h(x)

⊙ h(x) → x

⊙ h(x) → x + 100

⊙ Space ≈ O(M)

Key	Value

Size <= M

h(x) = x % M

10^8

0 <= key <= M-1

M = 17

A : [10, 20, 10, 27]

A % M : [10, 3, 13, 10]

idx	freq
3	freq of 20
10	freq of 10
13	freq of 20

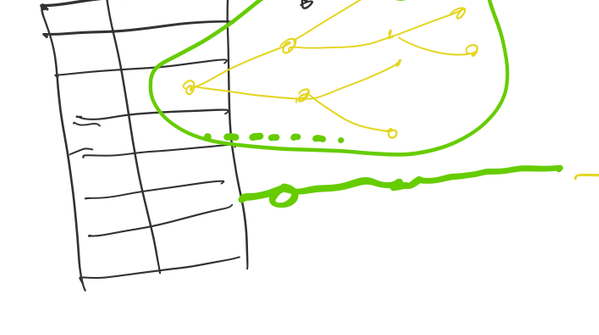
⊙ Collision

⊙ chaining   h(x) = x % 5   M = 5

A : [2, 4, 5, 7, 10, 20, 2, 5, 5]

h(x) : [2, 4, 0, 2, 0, 0, 2, 0, 0]

N : 1000

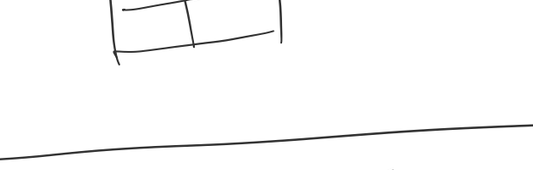


SC: O(M + N)

fn 1 (7) ?   7 % 5 → 2

Worst Case

TC : O(N)



BBST   height = O(log N)

Updating / adding / deleting / lookup   O(log N)

STL

	TC.	Java	C++
Sorted Set	O(log N)	Tree Set	Set
Unsorted set	O(1)	Hashset	unordered_set
Sorted Map	O(log N)	Tree Map	map
Unsorted Map	O(1)	Hashmap	unordered_map

< key >   O(N)

< key, value >

BBST

```
unordered_map < int, int > mp;
f ( i: 0 → n-1)
    mp[ Ai]++;

cout << mp [x] ;
```

A_i <= 10^9

Q Given an array of integers.
Find a pair (i, j) : A_i == A_j & (i != j)
| i - j | is minimized!

A : [1, 2, 3, 2, 1, 6, 3, 2, 1]   (index 0 1 2 3 4 5 6 7 8)

(1, 3)

B.F   ∀ (i, j)   if(A_i == A_j && i != j)   ans = min(ans, |i-j|)

TC: O(N^2)

SC : O(1)

⊙ Sort   [1, 1, 1, 2, 2, 2, 3, 3, 6]   (0 4 8 | 1 3 7 | 2 6 | 5)   4 4 2 4 4

TC: O(N log N)

SC: O(N)

⊙ HM   [1, 2, 3, 2, 1, 6, 3, 2, 1]   (0 1 2 3 4 5 6 7 8)

(i, j)   (3, 1)   ans = 2

HM < int, int >   < val, last seen idx >

(1, 8)
(2, 7)
(3, 6)
(6, 5)

```
hm < int, int > mp;                    ans = ∞
f ( i:0 → n-1) {
    if( mp. get (Ai)) {
        pos ans =  i - mp[Ai];
    }
    mp[A[i]] = i;
}
```

Q Count pairs (i, j) s.t. A_i == A_j & (i != j)

A : [1, 2, 3, 2, 1, 6, 3, 2, 1]

→ 7

(1, 3) → $^3C_2$ → 3
(2, 3) → $^3C_2$ → 3
(3, 2) → $^2C_2$ → 1
(6, 1) →

7

```
f (i: 0 → n-1)
    mp [ A(i) ]++;
```

Sum = 0
Iterate over the map
Sum += $^{freq}C_2$

```
sum = 0;
f(i: 0 → n-1) {
    sum += mp[Ai];
    mp[Ai]++;
}
```

TC: O(N)

SC: O(N)

Q Given an array of integers.
Find the longest subarray with sum = 0.

A : [1, -2, 2, 4, 0, -4]

5

B.F ∀ subarray → O(N^2)
find the sum → O(N)   O(N^3)
PS : O(1)   O(N^2)

L — R

P[R] - P[L-1] = 0

P[R] == P[L-1]

A : 1, -2, 2, 4, 0, -4

P : [0, 1, -1, 1, 5, 5, 1]   (sum; 0 1 2 3 4 5 6)   L-1 R

0 : 0
1 : 1
-1 : 2
5 : 4

(i, j)   6 - 1 = 5   3 - 1 = 5

P_i == P_j
| i - j | is Maximized!

PS + HM

TC: O(N)

SC: O(N)

Q Given an array of n integers (distinct)
Find longest chain of consecutive elements.
(Ordering of indexes do not matter).

4

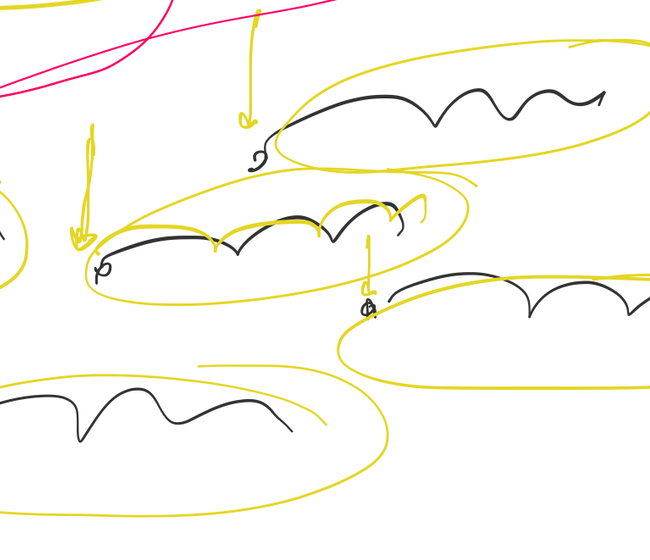
A : [100, 4, 3, 6, 10, 20, 11, 5, 101]

1) Sort   [3, 4, 5, 6, 10, 11, 20, 100, 101]   4 2 1 2

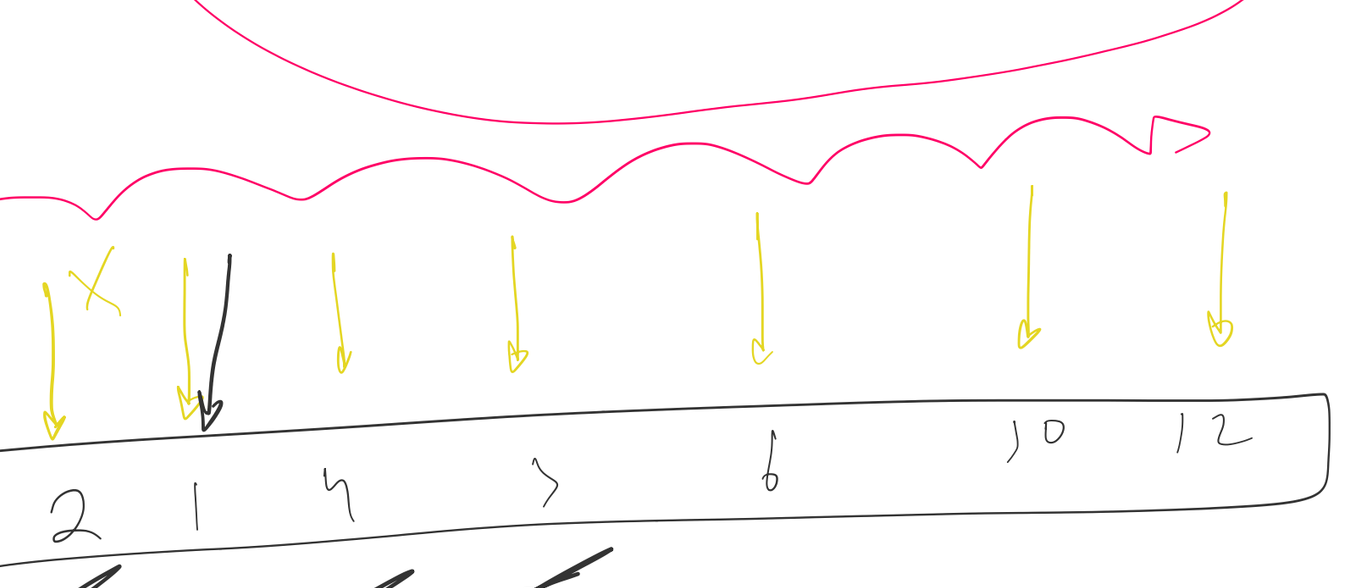
TC: O(N log N)

SC: O(1)

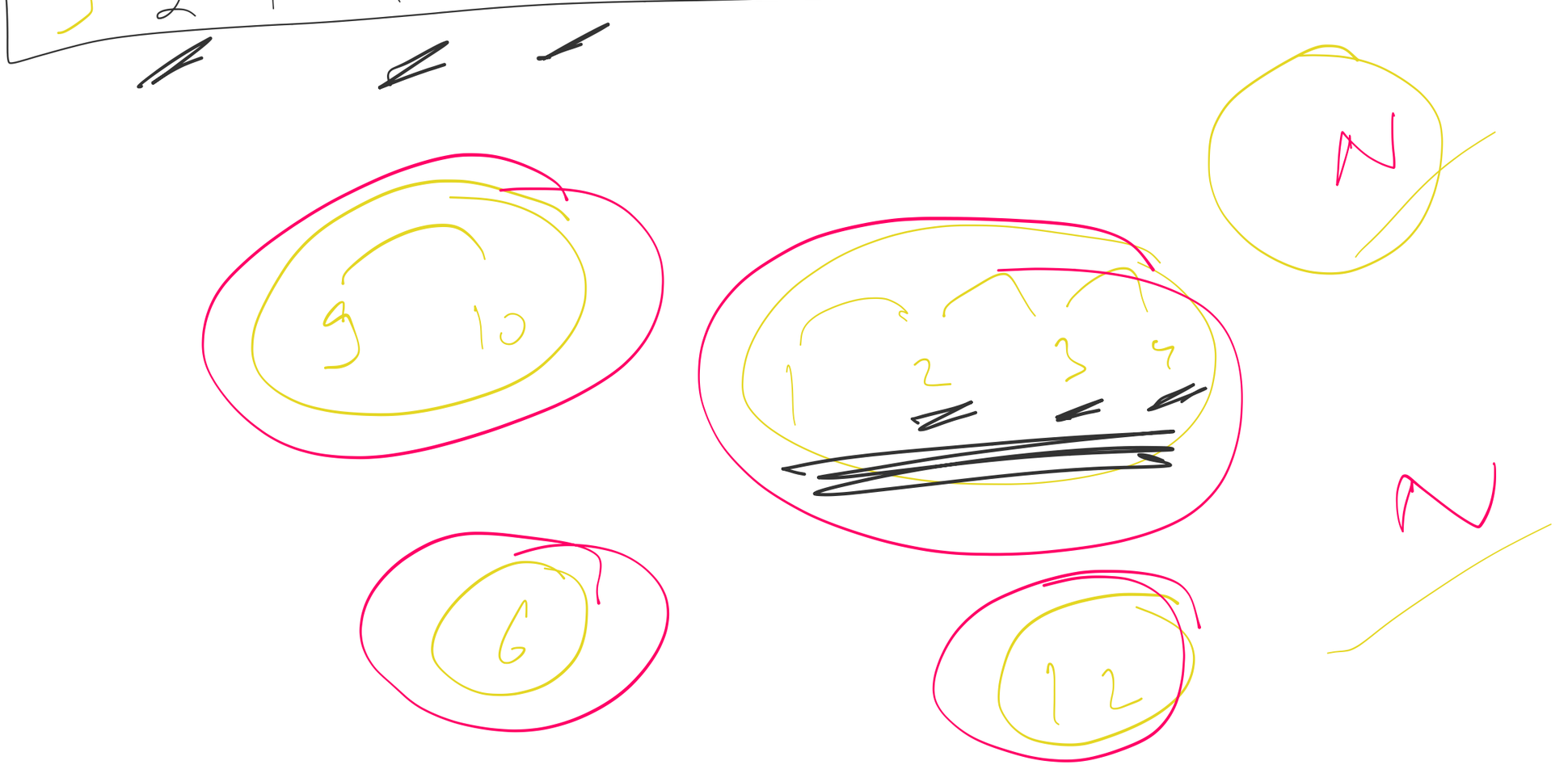
Ans = ~~2~~ 4



TC: O(N) + O(N) + O(N)



TC: O(N)

SC: O(N)