HashMap HashSet

HashSet : LookUp → Distinct elements.

⇓

< Key >

1) Search (Key) ⇒ T/F O(1)
2) Insert (Key) ⇒ O(1)
3) Delete (Key) ⇒ O(1)
4) Size() ⇒ O(1)

Eg : Ayush, Kapil, Gaurav, Abhishek, Siddhi, Vishal, Anurag, Ayush Arpit.

HashSet <String>

Ayush, Kapil, Gaurav, Abhishek, Siddhi, Vishal, Anurag, Arpit

(Key) Name	Value Scores
Ayush	24
Kapil	26
Gaurav	21
Siddhi	30
Vishal	19
Arpit	25
Anurag	29

< Key, Value >

⇓

HashMap

1) Get (Key) ⇒ Return value for this Key

⇒ O(1)

2) Delete (Key) ⇒ O(1)

3) Insert (Key, Value) ⇒ O(1)

4) Size () ⇒ O(1)

Q1 Given an array of N integers. Return the count of distinct integers

Eg: A: 1, 3, 6, 1, 2, 3, 4, 8, 6

⇒ Use a hashset & insert all elements into it.

⇒ Return the size of the hashset.

T.C. = O(N)

S.C. = O(N)

Q2 Given an array of N elements. Count the no. of duplicate pairs.??

	0	1	2	3	4	5	6	7	8	9
A:	1	2	1	4	1	2	3	4	1	6

- (0, 2)
- (0, 4)
- (0, 8)
- (1, 5)
- (3, 7)

8

(2, 4)
(2, 8)
(4, 8)

(i, j) s.t. i != j & A[i] = A[j]

	0	1	2	3	4	5	6	7	8	9
A:	1	2	1	4	1	2	3	4	1	6

Key	Val (freq)
1	~~1~~ ~~2~~ ~~3~~ 4
2	~~1~~ 2
4	~~1~~ 2
3	1
6	1

1 2 3 4

$^4C_2$

$= \frac{(4)(3)}{2}$

Key = f ⇒ no. of pairs = $\frac{(f)(f-1)}{2}$

T.C. = O(N)

S.C. = O(N)

// 1) Create a freq map ⇒

count = 0;

Iterate over the hashmap & for

every Key:

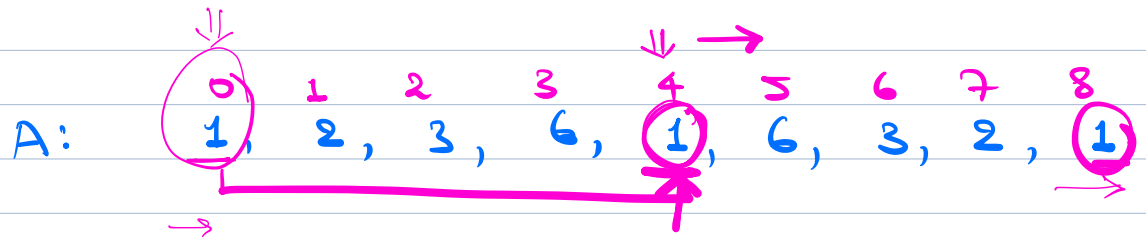
freq:

$$count += \frac{(freq)(freq-1)}{2}$$

Q) Given an array of size N.

Return the min distance b/w any 2 same elements.

$i, j \rightarrow dist = |i-j|$



ans = 2 Ans

HashMap → < Key , Value >

Key → Element

Value → Index of last occurance.

Code

```
HashMap<int, int> hm.
ans = INT_MAX
for (i=0; i<N; i++) {
    if (hm.contains(A[i])) {
        dist = |i - hm.get(A[i])|
        ans = min(dist, ans);
        hm[A[i]] = i;
    }
    else {
        hm.insert(A[i], i);
    }
}
```

Q) Given an array of size N.

Return the **max** distance b/w any 2 same elements.

i, j → dist = |i - j|

	0	1	2	3	4	5	6	7	8
A:	1,	2,	3,	6,	1,	6,	3,	2,	1

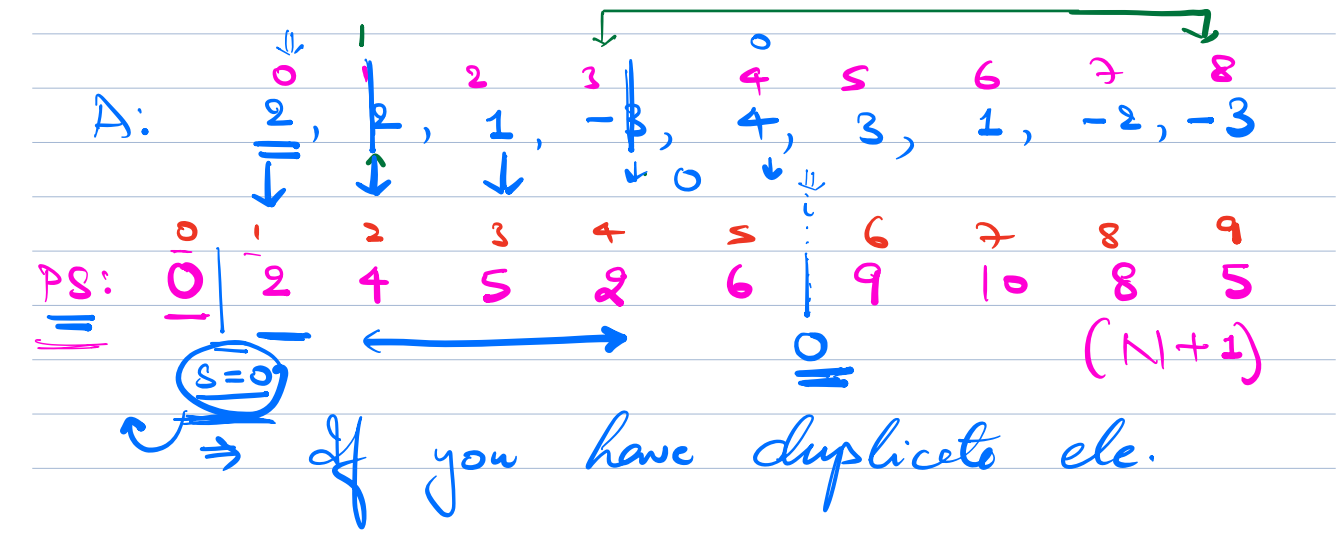
⇒ 8

Code

```
HashMap<int, int> hm.
ans = INT_MIN
for (i=0; i<N; i++) {
    if (hm.contains(A[i])) {
        dist = |i - hm.get(A[i])|
        ans = max(dist, ans);
    }
    else {
        hm.insert(A[i], i);
    }
}
```

Q. Given an array of size N.

Check if there exists a subarray of sum 0



ans is True

Hashset

$$A[s, e] = P[e] - P[s-1]$$

($s = 0 \Rightarrow s-1 = -1$)

$$P[e+1] - P[s]$$

Q Given an array of size N.
Return the count of subarrays of sum 0

	0	1	2	3	4	5	6	7	8	9	10
A:	2	2	1	−3	4	3	1	−2	−3	0	
PS:	0	2	4	5	2	6	9	10	8	5	5

(4)

\# subarray = \# duplicate pairs.

T.C. = O(N)
S.C. = O(N)

A: [6, −6, 0, −6, 6, 0]

P: 0 6 0 0 -6 0 0

0 → 5
6 → 1
-6 → 1

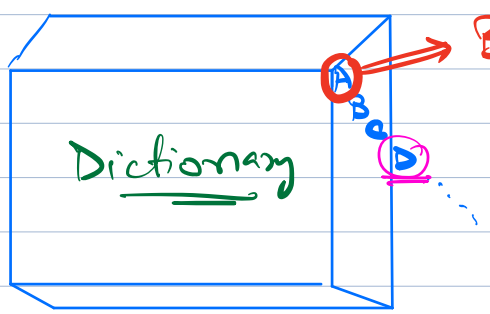
$\frac{5 \times 4}{2} = 10$

Class starts at 10:46

Google FB

Implement our own HashMap

Set/Map → Add()
get(key)



(Binary Search)
AYUSH

Medical Store

Paracetamol → Bucket (search)

Set/Map : 1) We need to bucketise the data
2) Getting to the bucket of every element should be

O(1) ⇒ array

Dictionary

ASCII(char) − ASCII(A)

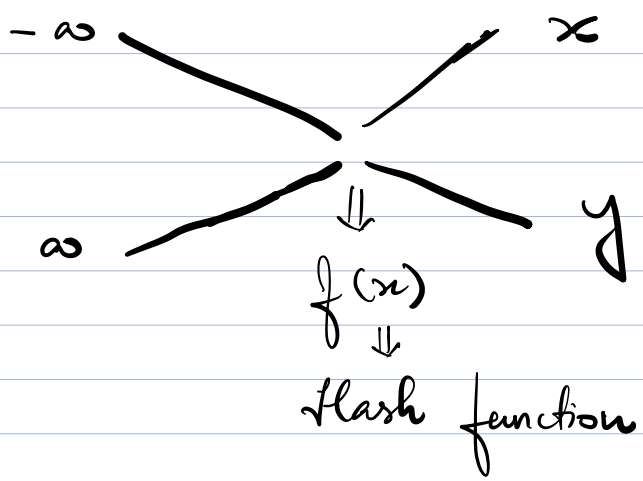
A − A = 0

B − A = 1

C − A = 2

0	A	→ 1, 2, 3.	30
1	B	→ 31, 32	75
2	C		
3	D		
.	.		
.	.		
.	.		
25	Z	→ 931, 932	1021

get ("cat")



1) Mod operation

[−∞, ∞]

⇓

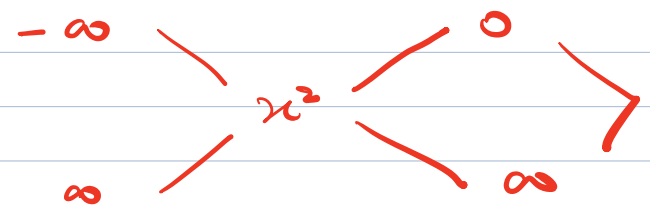
$x \,\%\, M = [0, M-1]$

2) $f(x) = \begin{cases} -1, & \text{if } x < 0 \\ 0, & \text{if } x = 0 \\ 1, & \text{if } x > 0 \end{cases}$ $[-1, 1]$

3) $f(x) =$ last digit of x

$x \,\%\, 10 \Rightarrow [0, 9]$

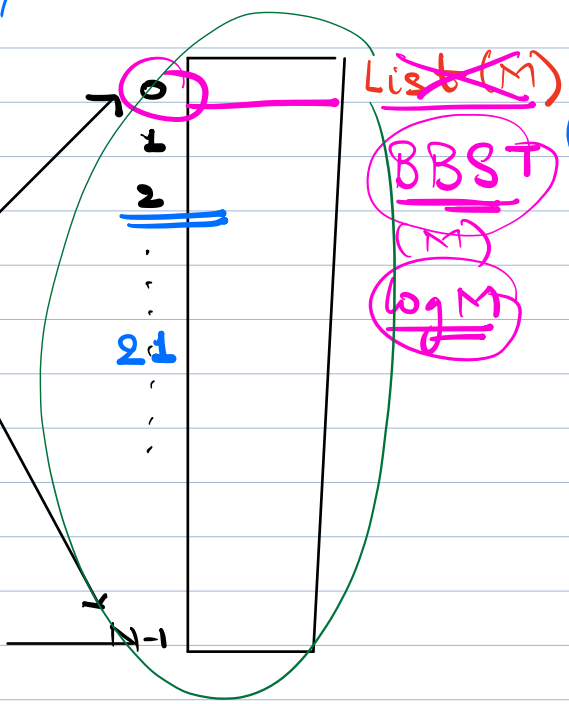
~~4)~~ $f(x) = x^2$



Look Up Table

101	YES , 3
102	YES , 4
⋮	⋮
601	⋮
⋮	
610	No , 2

Look Up Table ⇒ Array



O(M) for searching

121 %. 100 = 21

221 %. 100 = 21

2021 %. 100 = 21

I/P ⇒ Hf

x %. 100

(N) ⇒ HashMap ⇒ log (N)

Good Hash function (SHA1, SHA2, MD5,)

1) Should be able to use all possible buckets

2) for any data, all slots should be equally likely available.

3) Should be able to distribute the data uniformly across buckets

Elements = N

Buckets = B

Elements in each bucket = (N/B)

List / Array T.C. = $O(N/B)$

Self Balancing BST → T.C. = $O(\log_2 (N/B))$

$N/B = 10^9$

$\log_2 (10^9) \approx 32$

Amortized T.C. = $O(1)$

N → $(N-1) = O(1)$

$1 \rightarrow O(N)$

$(N-1) \times 1 + N$

$= 2N - 1$

$= 2N = O(N)$

$1 \rightarrow O(1)$