Today's Content:

- No of duplicate pairs
- Min distance b/w any 2 distance elements
- longest Consecutive Sequence
- Number of unique 2d points

(Myntra)

Q) Given ar[N], count no of duplicate pairs i.e

$A[i] = A[j]$ and $i \neq j$

	0	1	2	3	4	5	6	7	8	9
Ex1:	1	2	2	4	1	2	1	4	4	1

// (i, j) : ⟶ 12 pairs

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

Idea: Check all pairs. TC: $O(N^2)$ SC: $O(1)$

```
i = 0; i < N; i++){
    j = i+1; j < N; j++){
        3
    }
}
```

// Idea: We can only pair similar elements

: Index = 1 : {0 4 6 9} ⇒ 6 ⇒ $^4C_2$ ⇒ 6

: Index = 2 : {1 2 5} ⇒ $^3C_2$

: freq(4) ⇒ 3 : $^3C_2$

: ans = 12

Idea 2: We need freq of all elements

a) HashMap<Int, Int> hm ✓ (arr[i] → freq of array element)

b) Insert all arr[] in hm ✓ } TC: O(N)

hm

1	4
2	3
4	3

$$nC_2 = \frac{n!}{2! * (n-2)!}$$

$$nC_2 = \frac{(n)(n-1)}{2}$$

$$1C_2 = \frac{(1)(0)}{2} = 0$$

ans = 0

c) Iterate on hashmap for every key gets its frequency say = n } TC: O(N)

ans = ans + nC_2

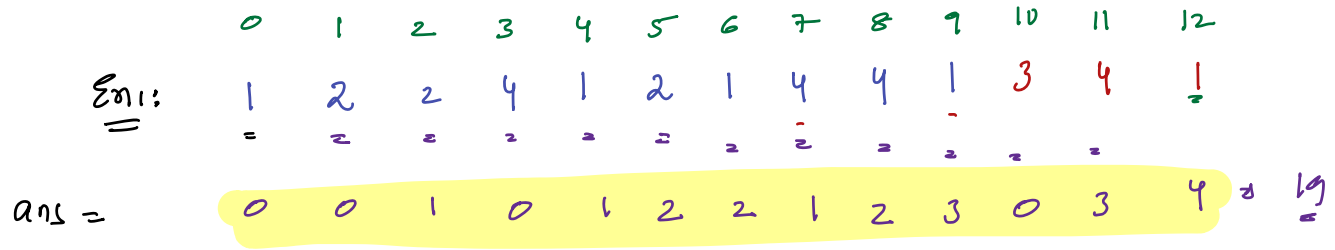
d) return ans;

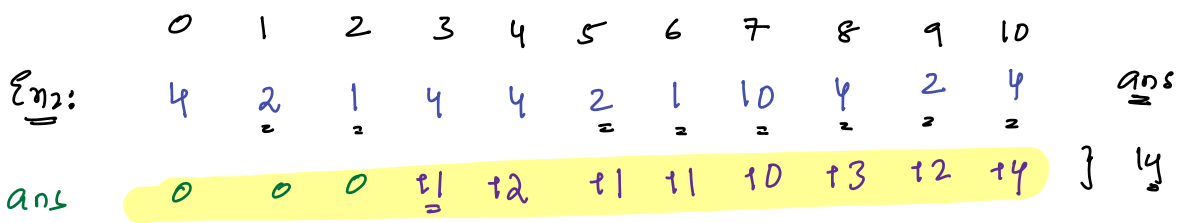
TC: O(N)

SC: O(N)

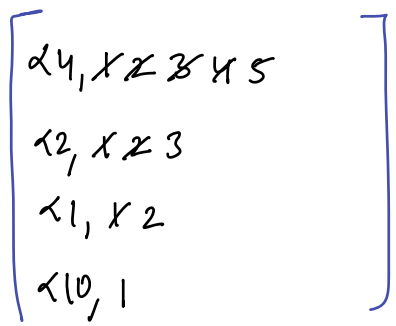
Try to do it single iteration

Idea: { for every ar[i], get its occurence from {0, i-1} }





HashMap <Int, Int> hm;



TC: O(N)

SC: O(N)

Pseudocode:

```
HashMap <Int, Int> hm
ans = 0;
i = 0; i < N; i++){
    if(ar[i] is in hm){
        ans = ans + hm[ar[i]]
        hm[ar[i]]++;
    }
    else { hm.insert(ar[i], 1) }
}
return ans
```

208) Given an array of size N, return the min distance b/w any two duplicate elements

(i, j) A[i] = A[j] & |i - j| is minimum

BF: All pairs:
TC: O(N²)
SC: O(1)

Ideas: { for every ar[i], get its later index from {0, i-1} }

Eg1:

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

ans = 2

Eg2:

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

ans = 2

ar[i] (later index of ar[i] element)

HashMap<int, int> hm

hm

{ <2, 0 4 8>, <4, 6>, <5, 7>, <6, 9> }

ans = min(ans, 4-0)
ans = min(ans, 5-2)
ans = min(ans, 6-1)
ans = min(ans, 7-5)
ans = min(ans, 8-4)
ans = min(ans, 9-3)

// ans = 2

HashMap<int, int> hm

ans = N

for(i=0; i<N; i++){
  if(ar[i] is in hm){
    ans = min(ans, i - hm[ar[i]])
  }
  hm[ar[i]] = i
  else { hm.insert(ar[i], i) }
}

return ans;

TC: O(N)
SC: O(N)

208) Given an array of size N, return the max distance b/w any two duplicate elements

(i, j) A[i] = A[j] & |i - j| is maximum ? } TODO

Ex2:

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

Idea: For every arr[i], we need it's first occurence index

Que: Given N array elements, find length of longest Sequence which can be re-arranged in a strictly increasing order by 1. { Consecutive }

Note: Sequence means pick any index elements, they don't have to be continous.

Ex1:

	x		*	*			*
0	1	2	3	4	5	6	7
-1	8	2	3	7	1	4	9

Seq1: 1 { 8 3 7 9 } ⟶ { 3 7 8 9 } *

Seq2: 2 { 8 7 9 } ⟶ { 7 8 9 } : len = 3

Seq3: 3 { 2 3 1 4 } ⟶ { 1 2 3 4 } : len = 4 (+1 +1 +1)

Ex2:

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

⇒ ans = 5

Seq1: { 3 2 1 3 } ⟶ { 1 2 3 3 } * (→ +1)

Seq2: { 3 2 1 } ⟶ { 1 2 3 } : len = 3

Seq3: { 8 9 6 5 7 } ⟶ { 5 6 7 8 9 } : len = 5

Idea: Sort + Iterate on array : TC: O(N log N + N)

ar[10] = { 2 6 7 10 8 14 9 12 13 15 }

ar[10] = { 2 | 6 7 8 9 10 | 12 13 14 15 } : ans = 5

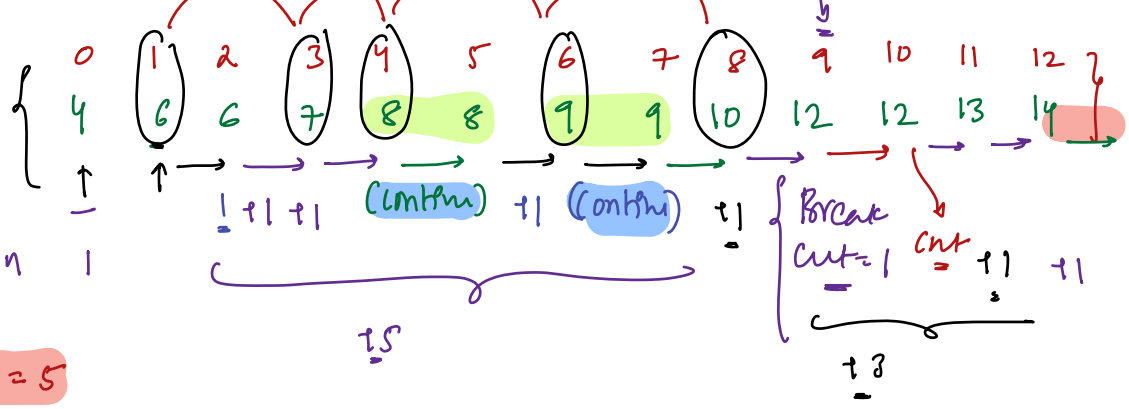
len: 1 5 4

ar[12] = { 9 4 8 8 12 14 13 6 7 6 12 10 9 }

Sort = { 4 6 6 7 8 8 9 9 10 12 12 13 14 }

len = 1 1 3 2 2 1 3

// Sorting + { Iterating when we get a duplicate, we are breaking: } { Issue }



// ans = 5

Idea: 1) Sort.

TC: $O(N \log N + N)$

// c = 1, ans = 0

2) i = 0; i < N-1; i++) {

```
    if (ar[i+1] - ar[i] == 1) {
        c = c+1;
    }
    else if (ar[i+1] == ar[i]) {
        continue
    }
    else {
        ans = max(ans, c)
        c = 1;
    }
}
```

return max(ans, c) { last sequence is left out }

Ex: ar[5] :

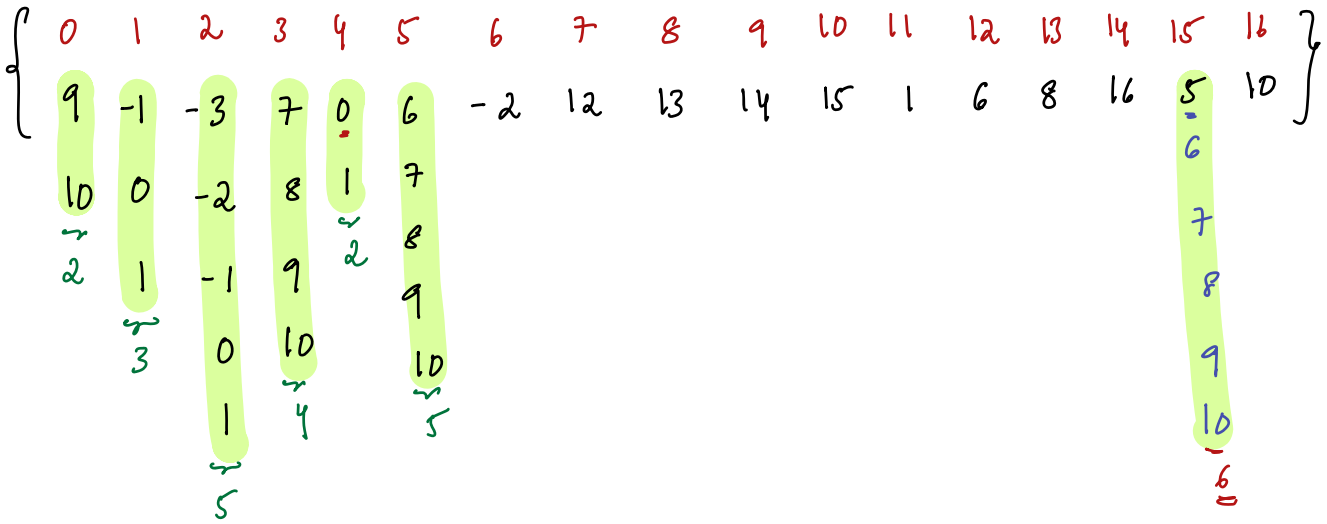
0	1	2	3	4	⇒ { consecutive }
1	2	3	4	5	

{ c = 1+1+1+1+1
  ans = 0

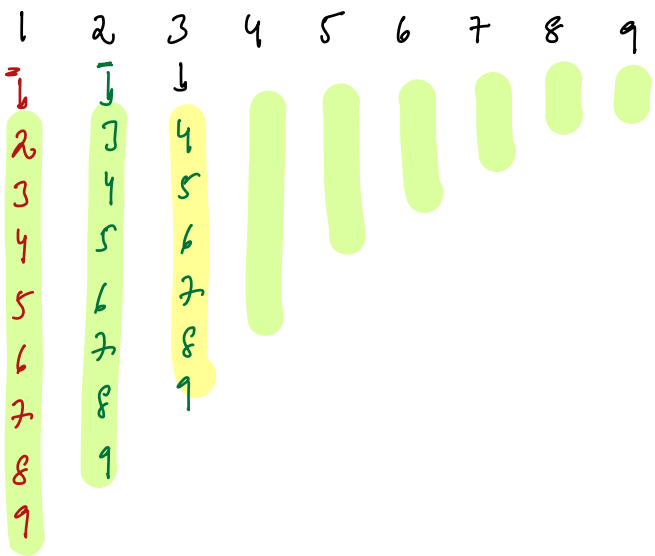
Idea2: a) Hashset<int> hs, insert all Elements in hs → O(N)

b) Take every element as start of Sequence, keep iterating until next element is not present in hs } TC: O(N) * {N}

c) TC: O(N + N²) SC: O(N)



Worst Can: TC: O(N²) SC: O(N)



Ideas:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
9	-1	-3	7	0	6	-2	12	13	14	15	1	6	8	16	5	10

9 : { 8 present : no need * }

-1 : { -2 present : no need * }

-3 : { -4 not present : -3 -2 -1 0 1

7 : { 6 present : no need * }

0 : { -1 present : no need * }

6 : { 5 present : no need * }

-2 : { -3 present : no need * }

12 : { 11 not present : 12 13 14 15 16

13 : { 12 present : no need * }

14 : { 13 present : no need * }

15 : { 14 present : no need * }

1 : { 0 present : no need * }

6 : { 5 present : no need * }

8 : { 7 present : no need * }

16 : { 15 present : no need * }

5 : { 4 is not present : 5 6 7 8 9 10 }

10 : { 9 present : no need * }

TC : O(2N) ≈ O(N)

SC : O(N)

Pseudo code:

a) HashSet<int> hs

b) Insert all ar[] elements in hs

c) ans = 0

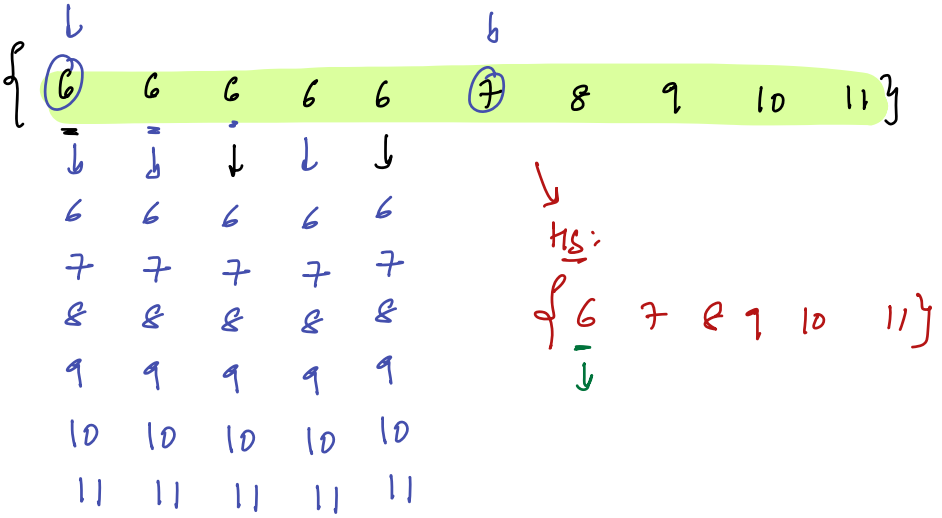
```
for (i = 0; i < N; i++) {
    // When can start a sequence from ar[i] ?
    if (ar[i] - 1 is not present in hs) {
        // we can start our sequence at ar[i]
        ele = ar[i] + 1; c = 1,
        while (ele is in hs) {
            ele++, c++;
        }
        ans = max(ans, c)
    }
}
return ans
```

TC: O(N) → O(N²)

SC: O(N) → O(N)

Worst Can :

A[] = { 6 6 6 6 6 7 8 9 10 11 }



a) HashSet<int> hs

b) Insert all ar[] elements in hs

c) ans = 0

{ Iterate in hashset, So that we can get only unique elements }

```
// When can start a sequence from ar[i] ?
if ( ar[i] - 1 is not present in hs ) {
    // we can start our sequence at ar[i]
    ele = ar[i] + 1 ; c = 1,
    while ( ele is in hs ) {
        ele++, c++;
    }
    ans = max( ans, c)
}
}
return ans
```

TC: O(N)

SC: O(N)

40) Given 2 arrays x[N], y[N], i^{th} → (x[i], y[i]) is point in 2d plane, how many different points are there?

N=5

A	B	
[3]	[5]	→ 1
[13]	[6]	→ 2
[1]	[16]	→ 3
[3]	[5]	✗
[6]	[13]	→ 4

ans = 4

Idea: Hashset

// Idea: Insert all points in hashset < pair< int, int >>

every point is a pair

key: pair< int, int >

In hashset / hashmap key can be following datatypes

- : int
- : long
- : float
- : string

Idea2:

Concate x[i], y[i] as string & insert

[3]	[5]	→	[35]
[13]	[6]	→	[136]
[1]	[36]	→	[136]
[3]	[5]	→	[35]
[6]	[13]	→	[613]

to different points can have same concatenation

Idea:

Concate x[i], y[i] as string with a separator = @, & insert all of them in HashSet<String> hs

[3] [5] = [3 @5]

[13] [6] = [13 @6]

[1] [36] = [1 @36]

[3] [5] = [3 @5]

[6] [13] = [6 @13]

// ans = hs.size()

TC: O(N)

SC: O(N)

// Doubts:

In HashSet / HashMap if we want key to be our required datatype, { override hashfunction }