Hashmap

↳ <K, V>

→ insert(K,V)

→ update(K,V)

→ get(K)

→ delete(K)

} O(1) avg case

Hashset

↳ Unique keys.

→ insert(K)

→ delete(K)

→ get(K)

} O(1) avg TC

C++ : unordered_map , unordered_set

Java : HashMap , HashSet.

Python : dict , set.

Q.1 Given an Array of size N. Count the no. of duplicate pairs i.e A[i] = A[j], i != j

Swiggy

Ex A: {1, 2, 3, 4, 1, 2, 1, 4, 6, 1}
(0 1 2 3 4 5 6 7 8 9)

(0,4) (4,6)
(0,6) (4,9) => 8 pairs.
(0,9) (6,9)
(1,5)
(3,7)

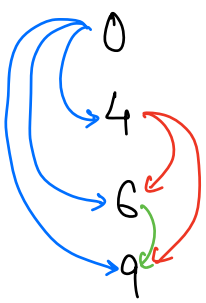
Brute force

TC: $O(N^2)$
SC: $O(1)$

#

freq(1) = 4 = freq

indexes



$3 + 2 + 1 = \frac{3(4)}{2}$

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

Combinatorics

4 indexes.

No. of (1,1) pairs = No. of ways of picking 2 indexes out of 4 indexes.

$= {}^4C_2$

No. of (a,a) pairs = No. of ways of picking 2 indexes out of x indexes.

↳ freq of (a)

$= {}^xC_2$

$= \frac{x!}{2!(x-2)!}$

$= \frac{x*(x-1)*(x-2)!}{2*(x-2)!}$

$= \frac{x(x-1)}{2}$

$^NC_r = \frac{N!}{r!(N-r)!}$

Approach :-

1) Create freq. map $\Rightarrow O(N)$

2) Iterate over the map & calculate the no. of pair. $\Rightarrow O(N)$

$\Rightarrow$ We are doing 2 traversals.

TC : $O(N)$

SC : $O(N)$

Can we do this in single iteration

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

map<ele, freq>

<1, 4>
<2, 2>
<3, 1>
<4, 2>
<6, 1>

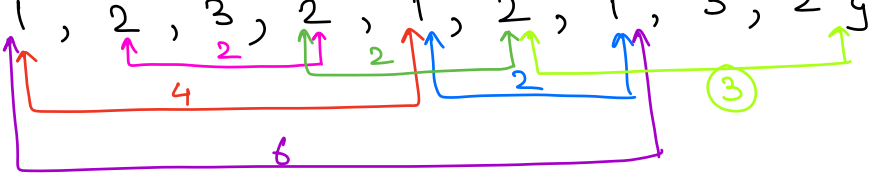
count = 0 + 1 + 1 + 2 + 1 + 3
= 8

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

Q: Google Given an Array of size N, return the minimum distance b/w any two duplicate elements.

(i, j) : A[i] = A[j] & |i-j| ⇒ MIN

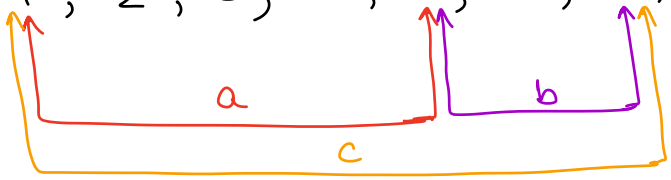
A: {1, 2, 3, 2, 1, 2, 1, 3, 2}
0 1 2 3 4 5 6 7 8



min = 2

Ex

A: { 1 , 2 , 3 , 2 , 1 , 2 , 1 , 3 , 2 }
(0) (1) (2) (3) (4) (5) (6) (7) (8)



a < c
b < c
} min distance will be either (a) OR (b)

⇒ Minimum distance will always be present b/w adjacent duplicate pair.

A: { 1 , 2 , 3 , 2 , 1 , 2 , 1 , 3 , 2 }
(0) (1) (2) (3) (4) (5) (6) (7) (8)

map<ele, index>

<1, 0> ~~4~~ 6
<2, 1> ~~3~~ ~~5~~ 8
<3, 2> 7

ans = ~~∞~~ 2

* Map will contain the latest occurrence of any element.

Iterate on Array :-

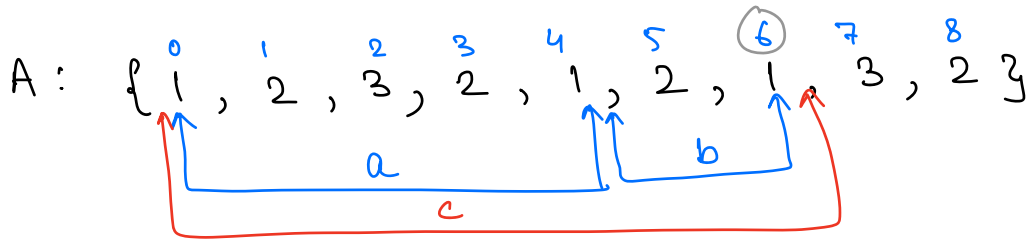
Check if A[i] is present in map :
if yes : check the distance & update ans if distance is less than ans.
Update the index of A[i] in the map
else :
make an entry of A[i], i in the map.

TC: O(N)

SC: O(N)

Q. Given an Array of size N, return the maximum distance b/w any two duplicate elements. (Google)

(i, j) : A[i] = A[j] & |i-j| ⇒ MAX.



c > a
c > b
} c is max

map < ele, first occurrence in Arr >

Ex :-

A: { 3, 2, 3, 6, 2, 9, 1, 3, 1, 2, 9, 8, 2, 2 }
0 1 2 3 4 5 6 7 8 9 10 11 12 13

map < ele, first occ. >

<3, 0>
<2, 1>
<6, 3>
<9, 5>
<1, 6>
<8, 11>

ans = -∞ ~~2~~ ~~3~~ ~~7~~ ~~8~~ 12

TC : O(N)

SC : O(N)

Q. Given an Array of size N. Find the length of largest sequence which can be rearranged to a sequence of consecutive numbers.

Google MS Amazon LinkedIn GS.....

A : 100, 4, 200, 3, 1, 2

4, 3, 1, 2 ⇒ 1, 2, 3, 4 ⇒ (4)

Quiz

A : {-1, 8, 2, 3, 7, 1, 4, 9}

2, 3, 1, 4 ⇒ 1, 2, 3, 4 ⇒ (4)

8, 7, 9 ⇒ 7, 8, 9 ⇒ (3)

Quiz

[5, 9, 100, 1, -1, 2, 3, 99, 98, 11, 101, 15, 102]

100, 99, 98, 101, 102 ⇒ (5)

1, 2, 3 ⇒ (3)

Sorting :-

{3, 100, 99, 4, 100, 3, 2, 101, 102}

↓ Sort

{2, 3, 3, 4, 99, 100, 100, 101, 102}

1 2 2 3 1 2 2 3 4

ans = ~~3~~ 4

TC : O(NlogN)

SC : Depends on sorting Algo.

Brute force :-

For every element A[i], find the length of consecutive sequence we can get starting at A[i]

A : {-1, 8, 2, 3, 7, 1, 4, 9}

length

-1 ⇒ -1, ~~0~~ ⇒ 1

8 ⇒ 8, 9, ~~10~~ ⇒ 2

2 ⇒ 2, 3, 4, ~~5~~ ⇒ 3

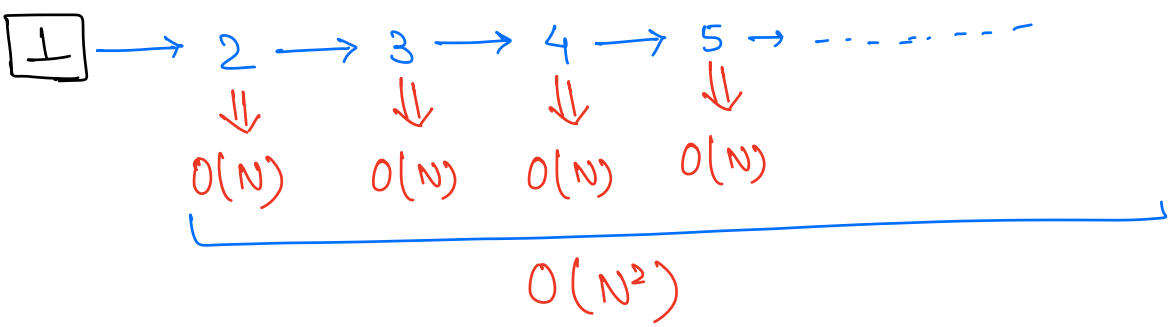
3 ⇒ 3, 4, ~~5~~ ⇒ 2

7 ⇒ 7, 8, 9, ~~10~~ ⇒ 3

1 ⇒ 1, 2, 3, 4, ~~5~~ ⇒ 4

4 ⇒ 4, ~~5~~ ⇒ 1

9 ⇒ 9, ~~10~~ ⇒ 1



for one ele $\Rightarrow O(N^2)$

N elements $\Rightarrow O(N^3)$

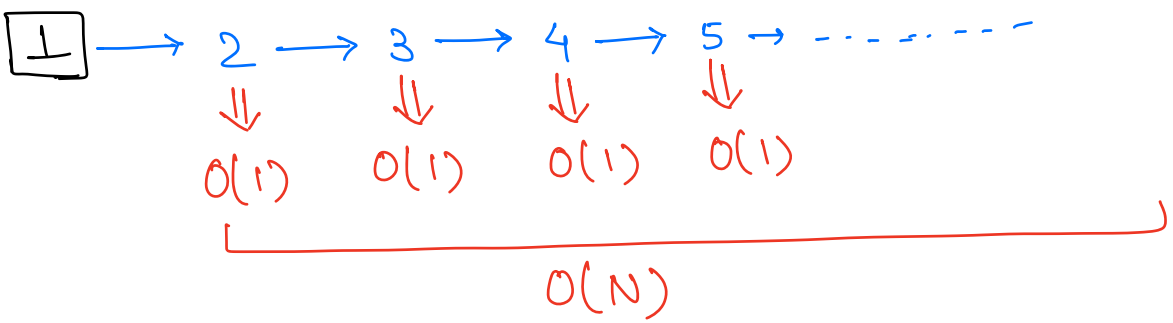
Searching $\Rightarrow$ SET

$O(1)$ TC

Code

1)
```
for ( i = 0; i < N; i++ ) {
    set.insert(a[i])
}
```

2)
```
for ( i = 0; i < N; i++ ) {
    len = 0
    x = a[i]
    while ( set.contains(x) ) {
        len++;
        x++;
    }
    ans = max(ans, len);
}
```



for one ele ⇒ $O(N)$

N elements ⇒ $O(N^2)$

Ex $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 100\}$

1 ⇒ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 $\quad l = 11$

2 ⇒ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

3 ⇒ 3, 4, 5, 6, 7, 8, 9, 10, 11

4 ⇒ 4, 5, 6, 7, 8, 9, 10, 11

5 ⇒ X

6 ⇒ X

7 ⇒ X

⋮

100 ⇒ 100 ⇒ (1)

Observation

If a[i]-1 is present in the Array | set then we don't have to check for the sequence starting at a[i].

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 100}

1 ⇒ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ⇒ (11)

2 ⇒ 1 is present in Arr, SKIP

3 ⇒ 2 is present in Arr, SKIP

4 ⇒ 3 is present in Arr, SKIP

5 ⇒ 4 is present in Arr, SKIP

⋮ ⋮

100 ⇒ 100 ⇒ (1)

Ex A: {4, 8, 10, 14, 12}

4 ⇒ 4

8 ⇒ 8,

10 ⇒ 10

14 ⇒ 14

12 ⇒ 12

TC: O(N)

* 1

Integer

{6, 6, 6, 6, 6, 7, 8, 9, 10, 11}.

6 → 6, 7, 8, 9, 10, 11 ⇒ 6

6 → 6, 7, 8, 9, 10, 11

6 → 6, 7, 8, 9, 10, 11

6 → 6, 7, 8, 9, 10, 11

6 → 6, 7, 8, 9, 10, 11

⇒ Instead of iterating over the Array iterate over the set in order to overcome the duplicate iterations.

Ex A: {6, 11, 10, 9, 8, 7}

6 → 6, 7, 8, 9, 10, 11 ⇒ 6

11 → X

10 → X

9 → X

8 → X

7 → X

```
for every element y in set :
    if ( !set.contains(y-1) ) {
        len = 0
        x = y
        while ( set.contains(x) ) {
            len++ ;
            x++ ;
        }
        ans = max(ans, len) ;
    }
}
```