Todays Content:

- → length of longest seq
- → longest substring with all distinct characters
- → Permutations of A in B

IQ) Given ar[N] ele, find length of longest seq (↳ subseq) which can be re-arranged in a strictly increasing by 1

{Note: Index elements doesn't have to be continous}

Ex1: ar[] = {-1 8 5 3 10 2 4 9} ans=4

(0 1 2 3 4 5 6 7)

Seq: {8 10 4} = {4 8 10}*

Seq: {5 3 2 4} = {2 3 4 5} ✓ len=4, ans=4

Seq: {8 10 9} = {8 9 10} ✓ len=3

Ex2: ar[] = {3 8 2 1 9 6 5 6 7 2} ans=5

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

Seq: {3 2 1} = {1 2 3} len=3

Seq: {8 9 6 5 7} = {5 6 7 8 9} len=5

Seq: {8 9 6 5 6 7} = {5 6 6 7 8 9}*

Ideal: Sort ar[] comp adj elements, get longest sub: TC: nlogN + N, SC: O(1)

ar[] = {-1 8 5 3 10 2 4 9}

Sort

ar[] = {-1 | 2 3 4 5 | 8 9 10} ans=4

l=1, l=4, l=3

ar[] = {3 8 2 1 9 6 5 6 7 2}

Edge Case

Sort

ar[] = {1 2 2 3 | 5 6 6 7 8 9} ans=5

(+1 +1 +0 +1 | +1 +1 +0 +1 +1 +1)

l=3, l=5

Idea2: for every arr[i], check if we can start sequence from arr[i] & get length & calculate overall max : TC: O(N) SC: O(N)

Eg1: arr[] = { -1 8 2 3 7 1 4 9 } ans=4

before	start
-2 ✗	-1 → 0 ✗ len = 1
7 ✓	8 ✗
1 ✓	2 ✗
2 ✓	3 ✗
6 ✗	7 → 8 → 9 → 10 ✗ len = 3
0 ✗	1 → 2 → 3 → 4 → 5 * len = 4
3 ✗	4 ✗
8 ✗	9 ✗

y = 7 → 8 → 9 → 10 →
l = 0 +1 +1 +1 break l = 3

Eg2:
arr[] = { 9 7 8 6 10 } ans=5

before	start
8 ✓	9 ✗
6 ✓	7 ✗
7 ✓	8 ✗
5 ✗	6 → 7 → 8 → 9 → 10 → 11 ✗ len = 5
9 ✓	10 ✗

{ Iterate on each ele in arr[] } ⇒ TC: O(N²)

Edge Case: arr[] = { 6 6 6 6 8 9 7 10 }

Start Start

SX 6 → 7 → 8 → 9 → 10 → 11x len = 5

SX 6 → 7 → 8 → 9 → 10 → 11x len = 5

SX 6 → 7 → 8 → 9 → 10 → 11x len = 5

SX 6 → 7 → 8 → 9 → 10 → 11x len = 5

↳ Reason: Since arr[] can contain duplicates, same start of seq can occur multiple times, due to this we will iterate on same subseq again & again TC: O(N²)

Resolve: Iterate on HashSet, instead of arr[] so that we will only get unique elements

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

```
int longseq(int ar[n]){ TC: O(N) SC: O(N)

    hashset<int> hs
    ar[] -> hs // insert all ar[] in hashset
    int ans = 0
    for(x in hs) // x will iterate on all keys on hashset TODO
        // check if we start seq from ar[i], if x-1 is not present
        if( hs.search(x-1) == false){
            l = 0  y = x
            while( hs.search(y) == true){
                l = l+1, y = y+1
            }
            ans = max(ans, len)
        }
    }
    return ans;
}
```

2Q) length of longest substring with all distinct characters?

S_1 = a b c a b c d d ans = 4
(l = 3, l = 4)

S_2 = s i p p i e r ans = 4
(l = 3, l = 4)

S_8 = a a a a a ans = 1
(l = 1)

Idea: For every substring check if it contains all distinct characters & get max length

10:26 → 10:35

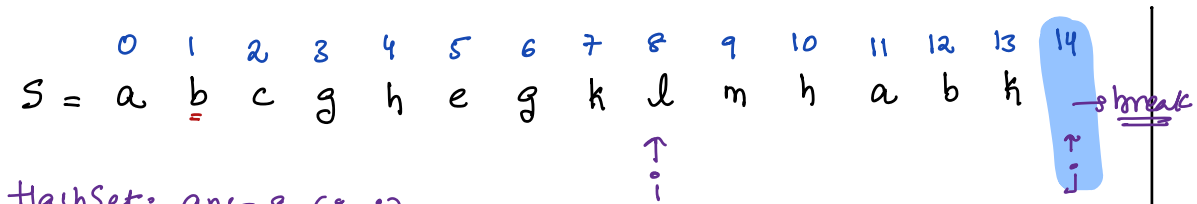
```
int longestdist(String s){ TC: O(N^3) SC: O(N)
    int n = s.length ans = 0
    for(i=0; i<n; i++){ // i is start of substring
        for(j=i; j<n; j++){ // j is end of substring
            // [i-j] is substring : len = j-i+1
            hashset<char> hs;
            for(k=i; k<=j; k++){
                hs.insert(s[k])
            }
            if(j-i+1 == hs.size()){ //substring contains all distinct characters
                ans = max(ans, j-i+1)
            }
        }
    }
}
```

Idea2: Find length of longest substring with all distinct characters starting from i, where every index is start of substring

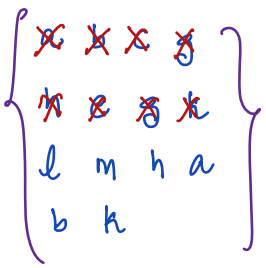
Ex:

	0	1	2	3	4	5	6	7	
S =	a	b	a	c	e	a	f	f	ans = 4
	↓	↓	↓	↓	↓	↓	↓	↓	
	2	4	3	4	3	2	1	1	

```
int longestdist(String s){ TC: O(N2) SC: O(N)
    int n = s.length ans = 0
    i=0; i<n; i++){
        // Calculate longest substring with all distict characters
           starting from i
        c=0
        hashset<char> hs
        j=i; j<n; j++){
            if(hs.search(s[j]) == false){
                c=c+1, hs.insert(s[j])
            }else break;}
        ans= max(ans, c)
    }
    return ans;
}
```



HashSet: ans = 8 (i, j)



(0, 0)
(0, 1) → (1, 1)
(0, 2) → (1, 2) → (2, 2)
(0, 3) → (1, 3) → (2, 4)
(0, 4) → (1, 4) → (2, 5)
(0, 5) → (1, 5) → (2, 6)
(0, 6) → (1, 6) → (2, 6) → (3, 6) → (4, 6)
(4, 6) → (4, 7) → (4, 8) → (4, 9) → (4, 10)
(4, 10) → (5, 10) → (5, 11) → (5, 12) → (5, 13)
(5, 13) → (6, 13) → (7, 13) → (8, 13) → (8, 14) i → ends string, j → ends string, return ans = 8

```
int longestDist(String s){ TC: O(N) SC: O(N)
    int n = s.length ans = 0
    i=0, j=0
    hashset<char> hs
    while(j < n){
        // When can we insert s[j]
        if( hs.search(s[j]) == false){
            hs.insert(s[j]) j++
            ans = max(ans, hs.size())
        }else{ // remove ith char
            hs.remove(s[i])
            i++
        }
    }
    return ans;
}
```

3Q) Given 2 Strings S_1 & S_2 of equal length, check if they are permutations of each other

Each string contains only lower case english alphabets

'a b c d .. z'

Ex1:

S_1	S_2	Match / Not
cat	tac	Yes
mata	tamt	No
anat	tana	Yes

Idea1: Sort both strings & compare

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

Sort S_1 | Sort S_2 | $S_1 == S_2$

Idea2: Freq of all characters in both strings should be same

→ Insert S_1 → hm1 → O(N), SC: O(26)

→ Insert S_2 → hm2 → O(N), SC: O(26)

→ Compare hm1 == hm2 → O(26) → O(1)

Overall TC : O(N)

SC: O(1)

↳ Check, if 2 hashmaps have same key & value pairs

↳ In above case hashmap will atmost contain keys =

We can do it in 1 hashmap as well : TODO

48) Count no: of substrings of B are permutations of A

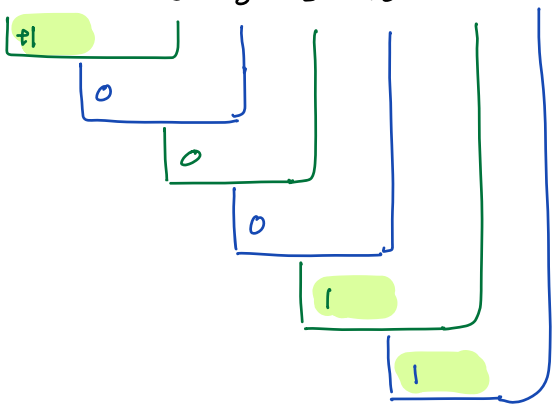
Note: len of B >= len of A

A_K : a b a

B_N :

0	1	2	3	4	5	6	7
a	b	a	b	b	a	a	b

ans = 3



Ideal: For all substrings of len = k in B, check if its permutation to A

Tc: (N - k + 1) * O(k)

↳ Worst Case k = N/2 ≈ (N/2 + 1)(N/2) ≈ O(N^2)

Sc: O(26) ≈ O(1)

Idea:2 Optimization using sliding window

```
A:  d  a  b  a   ⟶  HM1 : { a:2, b:1, d:1, }
    0  1  2  3

B:  a  a  d  b  a  b  a
    0  1  2  3  4  5  6
```

Substring:

				count
[0 3]	rem	add	HM2 { a:2, d:1, b:1 } == HM1	c=c+1
[1 4]	B[0]	B[4]	HM2 { a:2, d:1, b:1 } == HM1	c=c+1
[2 5]	B[1]	B[5]	HM2 { a:1, d:1, b:2 } ≠ HM1	
[3 6]	B[2]	B[6]	HM2 { a:2, d:0 b:2 } ≠ HM2	return c

Ideal:

- To insert all characters A → HM1 : O(k)
- For every substring of len = k in B, using sliding window get HM2 == HM1

  TC : 1 * (N-k+1) * O(26)

  ↳ 2 compare 2 hashmaps

↳ TC: O(k) + O(N-k+1) ≈ O(N) SC : O(26) ⇒ O(1)