Strings

Group of characters
Array of characters
Sequence of characters

'A' - 65 | 'a' - 97 | '0' - 48
'Z' - 90 | 'z' - 122 | '9' - 57
'z' - 122 | '2' - 50 | '7' - 55

2^5

$2^5 = 32$

70: 1 0 1 1 0 1 0
4 5 3 2 1 0

122: 1 1 1 1 0 1 0

16) String contains Small & Capital toggle them

a b C B ⇒ A B c b

```
string toggle(string s){
    int n = s.size();
    for(int i=0; i<n; i++){
        if( s[i] >= 65 && s[i] <= 90){
            s[i] += (32)
        }
        else{
            s[i] -= (32)
        }
        s[i] = s[i] ^ 32 // Better than if/else
    }
}
```

1

28) Str: a b a c d e a b [lower case]

Sort the string:

Solutions:

1) Merge Sort/Quick Sort

TC: N log N SC: O(N): MS
O(1): QS

2) How many different = 26

⇒ Construct count array
cnt[26] = 0
'a' - 0
'b' - 1
'c' - 2

⇒ Using cnt arr[] do count sort

TC: O(N) SC: O(1)
No of characters cnt[] ⇒ 26

30) Given a string find first repeating character [lower case]

Str1: a b c g c b
b: a_1
c: a_2
Ans: d f g e d
e f

Str2: c d e f e d
d
e

Q1 Ans: a a c e

Traverse from back won't work

First character from start of string which is repeating

Sol:
1) Construct cnt[] array
2) Iterate on string & check if a character has cnt[] > 1 Then itself is your ans.

Q2 First character which is repeating:
While construct cnt[] first character whose cnt[] value is updated to 2, That itself is ans.

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

Q4: Given a string reverse a part of string.

Str: a b a c d e f g → [2 5]
0 1 2 3 4 5 6 7
a b e d c a f g

```
string reverse(string str, int i, int j){
    while(i<j){
        swap(str[i], str[j])
        i++, j--;
    }
}
```

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

Q5: Given a string with space seven words by word

Str: love_hate_data_structures

Note: No space at start & end only 1 space between 2 words

Str: structures_data_hate_love

Solutions:

→ Take a vector push word by word using that reverse the string

love
hate
data
structures

Iterate in reverse order

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

Because we are storing all strings in a vector

Solution:
1) Reverse entire string
structures_data_hate_love
2) Reverse each word
structures_data_hate_love
Can be done using simple 2 pointer logic.
Idea: Between P_1 & P_2 we trap a single word & reverse that string.

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

60) Given a string, find length of longest palindromic substring

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Str: x b d y z z y d b d y z z y d x
P_1 P_2
P_1 P_1 P_1 P_2 P_2 P_2 P_2
8 [All characters between Them are palindrome] 14

length: $[P_2 - P_1 - 1] = (14 - 8 - 1)$
= 5

Solutions:

1) For every substring check palindrome or not & update ans.

Substrings ⇒ $(N(N+1))/2$
⇒ #1 : N
#2 : N-1
#3 : N-2
N : 1

$\frac{N(N+1)}{2}$

check palin ⇒ O(N)

TC: O(N³) SC: O(1)

Sol2:

Take every character as a center & expand in center to get palindrome length & update with ans.

TC: # N characters (Centre)
For each character to expand on left & right
TC: O(N) C

Overall TC: O(N²) SC: O(1)

Only odd length Palindromes.

Combine them to get the final solution

Take every adjacent character as centers, & expand left & right

TC: # N-1 (Centres)
O(N)

Overall TC: O(N²) SC: O(1)

even length Palindromes

Note: Max of both is your final ans.

Final TC: O(N²) SC: O(1)

Pseudocode

```
int expand(string str, int P1, int P2){
    while( str[P1] == str[P2] && P1 >= 0 && P2 < N){
        P1--;
        P2++;
    }
    return P2 - P1 - 1;
}
```

T/F

→ We have from check boundary condition & Then str[P1] == str[P2]

// It is possible P1 == -1, || P2 == N || In this case break

// Note: Only when P1 & P2 are exceeding boundary we break it.

```
int ans = 0;
// finding centre & getting max length;
int n = str.size();
// odd centres
for(int i=0; i<n; i++){
    ans = max(ans, expand(str, i, i))
}
// even centres
first even centre = (0,1)
last even centre = (N-2, N-1)
```

0 → N-2
(0,1) (1,2) (2,3) (N-2, N-1)

```
for(int i=0; i<N-1; i++){
    ans = max(ans, expand(str, i, i+1))
}
return ans;
```

→ TC: O(N log N) SC: O(N) : Rabin Karp + Rolling Hash + BS

→ Manchar's Algo: O(N) SC: O(N)

→ Dp : O(N²) SC: O(N²)

78) Given a string, find length of longest substring which contains all distinct characters. [String contains any ASCII characters]
↳ 256

0 1 2 3 4 5 6 7 8 9 10 11 12 13
Str[]: a b c e a c d a f g h d e a
P_1

Solutions:

1) For every substring check if it contains All distinct characters.

TC: O(N²) ⇒ O(N)

To check if a string contains all distinct or not using hashset

SC: O(N)

2) Carry forward your previous hashset.
O(N²) ⇒ O(1)

3) ⇒ BS + Sliding Window
[In Lab Session]

4)

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

*

//

P_1	P_2
1	1 ✓
1	2 ✓
1	3 ✓
1	4 ✓
1	5 *
2	5 *
3	5 ✓
3	6 ✓
3	7 *
4	7

Since 0-3 is already answer Directly check 1-4

$P_1 - P_2$
If valid: P_1 P_2+1
If Inveld: P_1+1, P_2

Pseudocode:

```
int i = 0, j = 0;
set<char> data;
while( i<N && j<N)
    if( data.find(s[j]) == data.end()){
        // s[j] is not there
        data.insert(s[j])
        ans = max(ans, data.size())
        j++;
    }
    else{
        // s[j] is there
        data.erase(s[i])
        i++
    }
return ans;
```

TC: O(N)
SC: O(N) *
O(1)
Because hashset can never exceed 256
O(1)

Sunday: Lab Session

→ Distinct elem
→ No of max prefixes
→ Quick Select
→ Above logic using BS
→ Overlapping
→ Count dist
→ 10am - 1pm

unordered set / set

Hash Map Implement

→ key: [Hash function]

Primitive
[int / long / float / double / string / char]

we have Inbuilt Function

Key: pair<int, int>

[Explicitly implement a hash function]

[overload hash function & implement this]