Welcome ☺

Agenda : Strings
- Intro.
→ Flip
→ Sort ch[]
→ Reverse string
→ Palindromic

String : → array of characters
→ bunch of ''
→ sequence of ''

{a c b} → not same
{a b c} → order matters.

Character : ASCII values.

'A' → 65 (+32 → 'a' → 97, -32 ← 'a')
'B' → 66 (+32 → 'b' → 98, -32 ← 'b')
'C' → 67 ('c' → 99)
.
.
.
'Z' → 90 ('z' → 122)

'0' → 48
'1' → 49
'2' → 50
'3' → 51
.
.
.
'9' → 57
'10' → ~~58~~ X
it is not a single char

char ch = 'g'
char → 1 byte
'g' → ASCII → 57

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

→
```
ch = ch + 8
print(ch)
```
⇒ 'g' + 8
⇒ 57 + 8 ⇒ 65 ⇒ A

String → array of characters.

String s = "abda"

→ s[0]

Q1 Given a char[], toggle each and every character.

↳ Capital ⟷ small

Note: Input contains only small & capital letters.

eg: ch[] → AnaConDa

ans = aNAcONdA

Code

```
Toggle ( char s[] )
{
    int n = s.length();
    for( int i=0 ; i<n ; i++)
    {
        if( s[i] >= 65 && s[i] <= 90) // capital letter.
        {
            // s[i] is capital.
            s[i] += 32;
        }
        else
        {
            // s[i] is small
            s[i] -= 32;
        }
    }
    return s;
}
```

T.C → O(N)

S.C → O(1)

Idea 2 → Toggle s^{th} bit

s[i] = s[i] ^ 32

Q2 Given a char [], which contains only lowercase alphabets. Sort the given array. in alphabetical order.

eg: S = d a b a c d b

after sorting ↓

S = a a b b c d d

constraints

→ $1 \le N \le 10^5$

→ $'a' \le ch[i] \le 'z'$

Bruteforce

① Sort all characters using bubble sort.

T.C $O(N^2)$ S.C → $O(1)$

② Sort using in built func^{n}s., using comparator (if needed)

T.C $O(N \log N)$ S.C → $O(1)$

Idea

S → d a b a c d b

int cnt [26]

'a' → 2 — 'a' $\xrightarrow[-'a']{-97}$ cnt [0]

'b' → 2 — 'b' $\xrightarrow[-'a']{-97}$ cnt [1]

'c' → 1 — 'c' $\xrightarrow[-'a']{-97}$ cnt [2]

'd' → 2

'z' $\xrightarrow[-'a']{-97}$ cnt [25]

Ans = a a b b c d d

Approach: Iterate on string and get count of each and every char.

Pseudocode

```
sortString ( char s[] )
{
    int n = s.length();
    int cnt[26] = {0};

① // Count freq. of all char in string.
    for( int i=0 ; i<n ; i++)
    {
        int index = s[i] - 'a'
        cnt[index] += 1 ;   // incrementing count.
    }

②   int K = 0;
    for( int i = 0 ; i<26 ; i++)
    {
        char ch = 'a' + i ;
        for( int j =1 ; j<= cnt[i] ; j++)
        {
            S[K] = ch ;      K = K+1 ;
        }
    }
return S;
}
```

O(N)

~~26 * N~~

N iterations

O(N)

T.C → O(N)

S.C → O(26) → O(1)

Table

i	j : [1, cnt[i])	#iteration
0	[1, cnt[0]]	cnt[0]
1	[1, cnt[1]]	cnt[1]
⋮		⋮
25	[1, cnt[25]]	cnt[25]
		N

cnt[0] + cnt[1] + cnt[2] ... cnt[25]

=

= freq of all char.

= N → string length.

Substrings → concept is same as subarray.

↳ contiguous part of string

→ Full string can be substring

→ a single char can also be a substring.

$$\#\text{ substrings} = \frac{N*(N+1)}{2}$$

Q Check if a given substring is Palindrome or not?

eg: madam

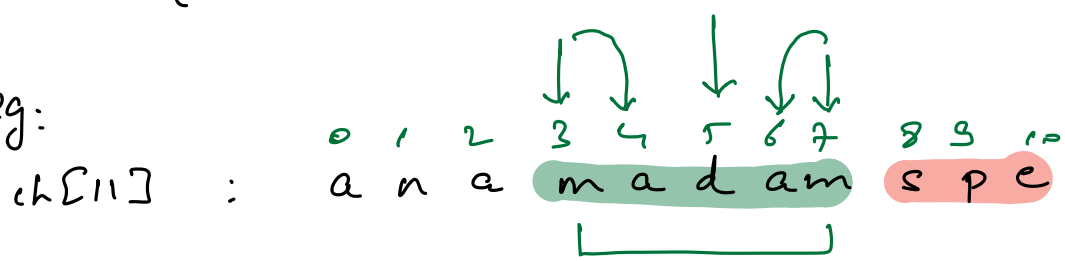
mam

dad

L → R

R → L

eg:



Code

```
bool isPalin( char ch[], int s, int e )
{
    while ( s<e )
    {
        if ( ch[s] != ch[e] ) {
            return false
        }
        s+=1;      e-=1;
    }
    return true;
}
```

T.C → O(N)

S.C → O(1)

Q Given a string, calculate length of longest palindromic substring

eg: a b a c a b

Ans = 5

eg: a b c d e

Ans = 1 single char is palindrome.

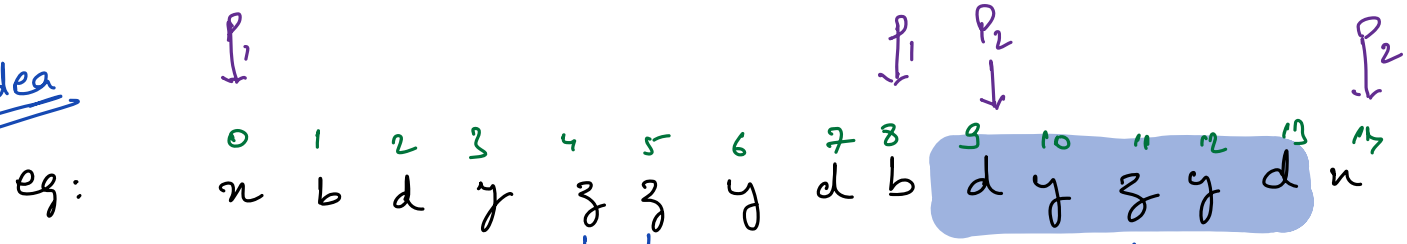
Brute force for every substring, check if palindrom or not and get max. length.

T.C → $O(N^3)$

→ [# of substring] * T.C to find palindrome

→ $\frac{N*(N+1)}{2} * N \Rightarrow O(N^3)$

Idea



→ Take every character as centre and expand the centre and get max. palindromic substring.

T.C → $O(N) * O(N) = O(N^2)$

S.C

$(P_1, P_2) = P_2 - P_1 + 1 - 2$

$= P_2 - P_1 - 1$

→ length of odd palindromes.

→ Take every adjacent characters as centre and expand the centre and get max len palindrome → even length palindrome.

T.C → $O(N) * O(N) = O(N^2)$

S.C

Pseudo code

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

// P1 = -1 P2 = 4

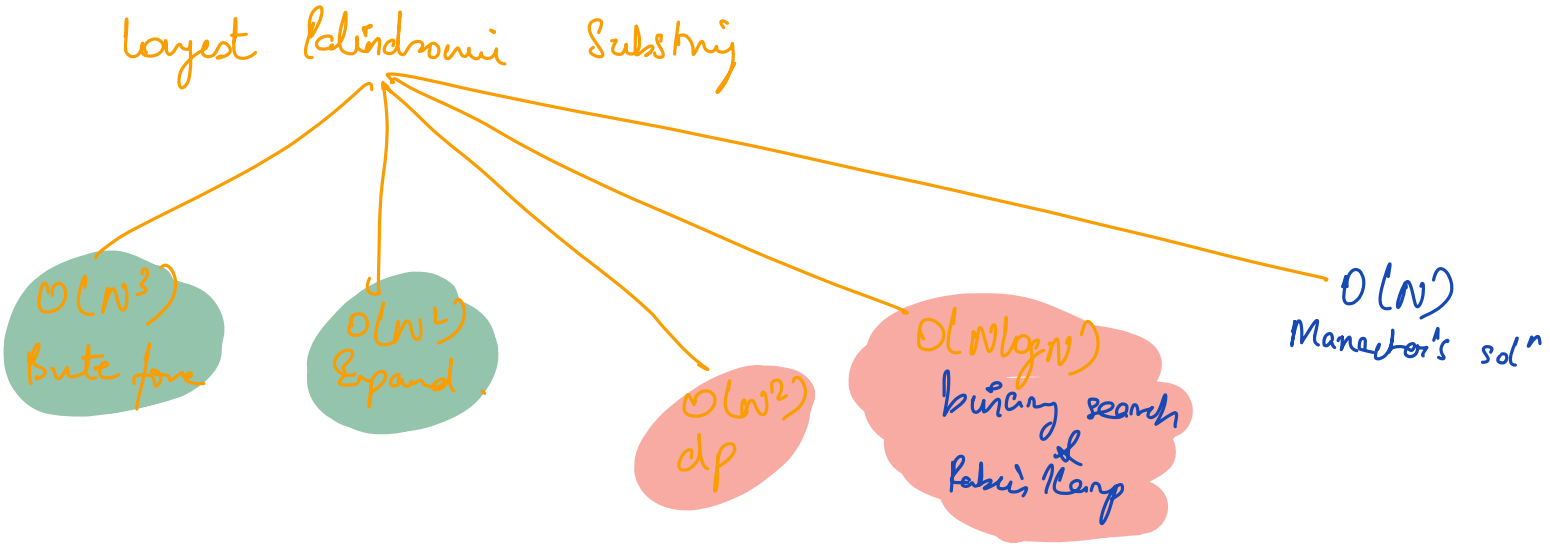
s[-1] → out of bound.

```
int lenPal ( char s[] )
{
    int n = s.length();
    int ans = 0/1 ;

    for ( int i= 0 ; i<n ; i++) → odd length palindrome
    {
        // centre = s[i]
        P1 = i   P2 = i
        ans = max ( ans, expand ( s, P1, P2));
    }

    for ( int i= 0 ; i<n-1; i++) → even length palindrome
    {
        // centre = s[i] , s[i+1]
        P1 = i   P2 = i+1
        ans = max ( ans, expand ( s, P1, P2));
    }

    return ans
}
```


Longest Palindromic Substring
O(N^3)
Brute force
O(N^2)
Expand
O(N^2)
dp
O(NlogN)
binary search
&
Rabin Karp
O(N)
Manacher's soln