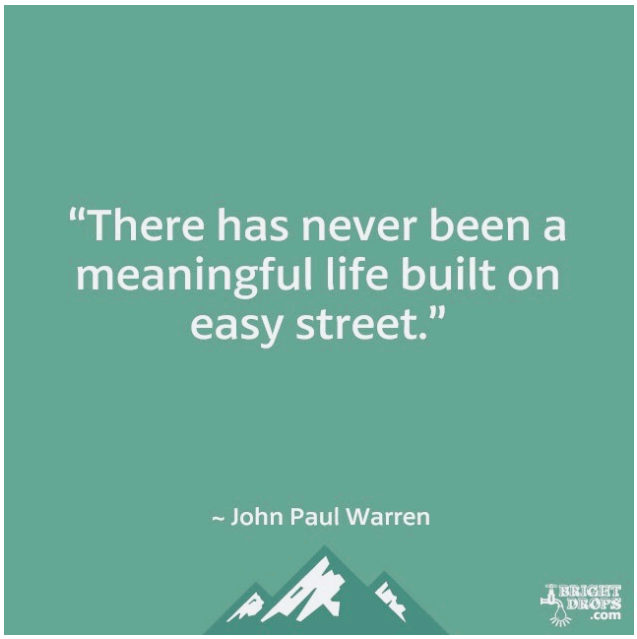
STRINGS



Good Evening

Welcome to easiest class of Intermediate class

Today's content → Introduction

- → Flip
- → Sort ch[]
- → check if string is palindrome
- → Longest Palindromic substring

String - → Array of characters

→ Series of characters

→ Group of characters

"abc" "acb"

Character → Assigned with a decimal value

{ASCII TABLE}

'A' → 65

'B' → 66

'C' → 67

.

.

.

'Z' → 90

'a' → 97

'b' → 98

'c' → 99

.

.

.

'z' → 122

'0' → 48

'1' → 49

'2' → 50

'3' → 51

.

.

.

'9' → 57

"a" + 1 → "a1"

'10' → Combination of char

```
Print ((char)('a' + 1)
Print (char) (97 + 1)
```

→

String → Array of character

String s = "abcd"
0 1 2 3

s[] = | a | b | c | d |
0 1 2 3

print(s[0]) → a

print(s[0]) → a

01. Given a char ch[], toggle every character of array

↓

uppercase to lowercase

lowercase to uppercase

Note :- char[] only contains lowercase or uppercase characters.

Eg :- scaLeR ⟶ SCALEr

Eg :- aBcdE ⟶ AbCDe

Idea → Check if a character is in uppercase

// ch ≥ 65 && ch ≤ 90
// ch ≥ 'A' && ch ≤ 'Z'
} check

Convert to lowercase

ch + 32

ch − 65 + 97

ch − 'A' + 'a'

01. 'A' − 'A' + 'a' ⇒ 'a'

02 'c' − 'A' + 'a'

$$\underbrace{67 - 65}_{2} + 97$$

$$2 + 97 = \boxed{99} \quad 'c'$$

→ Check if a character is in lowercase

// ch ≥ 'a' && ch ≤ 'z'

// ch ≥ 97 && ch ≤ 122

} check

conversion

ch - 32

ch - 97 + 65

ch - 'a' + 'A'

```
Toggle (char [] str)
    for (i=0; i<n; i++)
        char ch = str[i];
        if (ch ≥ 'A' && ch ≤ 'Z'){
            str[i] = (char) (str[i] + 32);
        }
        else {
            str[i] = (char) (str[i] - 32);
        }
}
```

TC: O(n)

SC: O(1)

Constraint → You can't use if & else or ternary op

Hint → Think in direction of bit manipulation {TODO}

02. Given a char ch[], which contains only lower case alphabets. Sort given ch[], in alphabetical order.

Constraints: $1 \le N \le 10^5$

$'a' \le ch[i] \le 'z'$

Expected SC: O(1)

Eg:- a a b c a z d ⟶ a a a b c d z

b c a e d ⟶ a b c d e

Brute force → Use inbuit sorting function

TC: O(n log n)

max $n = 10^5$

$= 10^5 * \log_{10}(10^5)$

$= 10^5 * 5$

$= 5 * 10^5$ Work

02. Optimised Approach

→ Count the freq of each & every character

→ use the freq arr or freqmap to fill the given arr

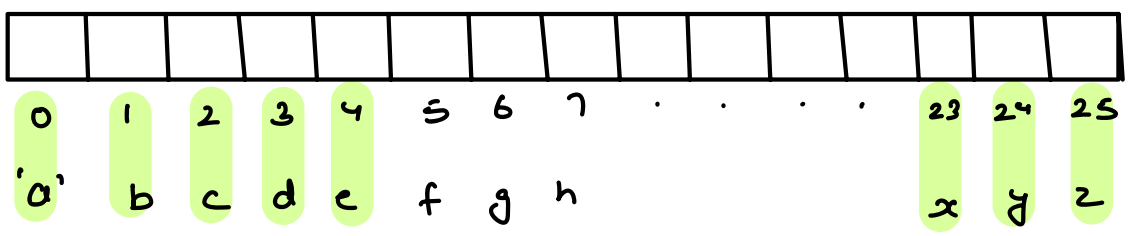
Eg:- ch[] = {z d a b c a z}

a → 2
b → 1
c → 1
d → 1
e → 0
f → 0
⋮
z → 2

ch = {a a b c d z z}

↳ Array of size 26 which is going to store freq of each & every distinct char

```
int [] count = new int [26]
```



'a' (97) —(-97)→ 0

'b' (98) —(-97)→ 1

'c' (99) $\xrightarrow{-97}$ 2

Eg:- ch[] = { a c b a d }

a a b c d ⟶ sorted

count[26] =

2	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	.	.	.	.	23	24	25	
'a'	b	c	d	e	f	g	h					x	y	z	

```
char[] sortstr (char[] str)
    int[] count = new int[26];          # count sort
    for (i=0; i<n; i++){
        int idx = str[i] - 'a'
        count[idx]++;
    }
    // iterate on count arr & fill given arr with
       the frequencies of each & every character
        idx = 0
    for (i=0; i<26; i++)
        // count[i] → frequency of (97 + i) character
        int freq = count[i];
        for (j=1; j<=freq; j++){ → run for freq times
            str[idx] = (char) (97 + i)
            idx++;
        }
}
```

122

i	j = [1 → freq of i]	no. of iterations
0	j : [1 – c[0]]	c[0]
1	j : [1 – c[1]]	c[1]
2	j : [1 – c[2]	c[2]
.	.	.
.	.	.
.	i	.
.		.
25	j : [1 – c[25]]	c[25]

length of ch[]
$= O(n)$

Total TC : $O(n)$

Total SC : $O(26) = O(1)$

With the change in string size, will our 26 sized count array change?

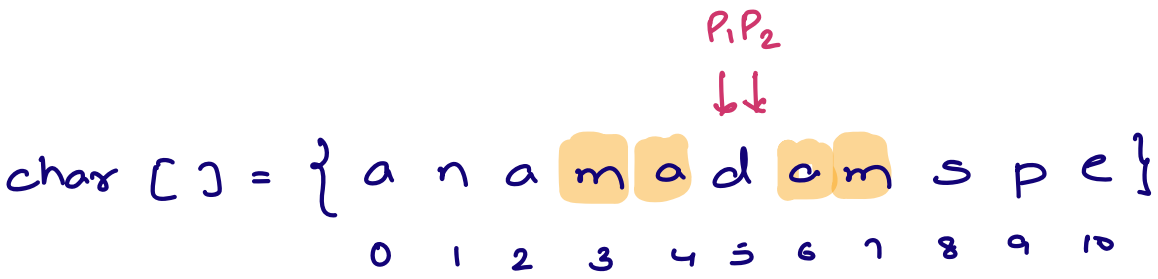
Substring → Concept is same as subarray

- ↳ continous part of string
- ↳ Full string is a substring
- ↳ single char of string is a substring

Q → check if given substring is palindrome or not.

Eg:- madam
nayan
racecar
a

} Palindrome



st = 3

end = 7

Idea → Use two pointers at st & end & check characters

```
boolean ispali (char[] str, int st, int e)
    while (st < e)
        if (str[st] != str[e]) return false;
        st++;
        e--;
    }
    return true;
}
```

TC: O(n)
SC: O(1)

Q Given a string, calculate length of longest palindromic substring

Constraint :- $1 \leq N \leq 3 * 10^3$

Eg :- a b a c a b d (0 1 2 3 4 5 6) Ans = 5

Eg :- a b c d Ans = 1

char[] = { a n a m a d a m s p e } Ans = 5
(0 1 2 3 4 5 6 7 8 9 10)

BF Approach → Generate all the substrings & check if the substring is palindrome & then update the max length

⇒ # no. of substring * check if the substring is palindrome

$\frac{n(n+1)}{2} * n$

$\approx n^2 * n$

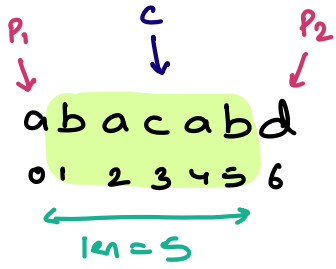
$TC = O(n^3)$

Bruteforce

```
int longpal (char[] str)
    int ans=0
    for (i=0; i<n; i++)          i = st
        for(j=i ; j<n ; j++){    j = end
            // substring [i j]
            if (ispali (str, i, j) == true){
                ans = Math.max(ans, j-i+1);
            }
        }
    return ans
}
```

TC: $O(n^3)$
SC: $O(1)$

Optimised Idea

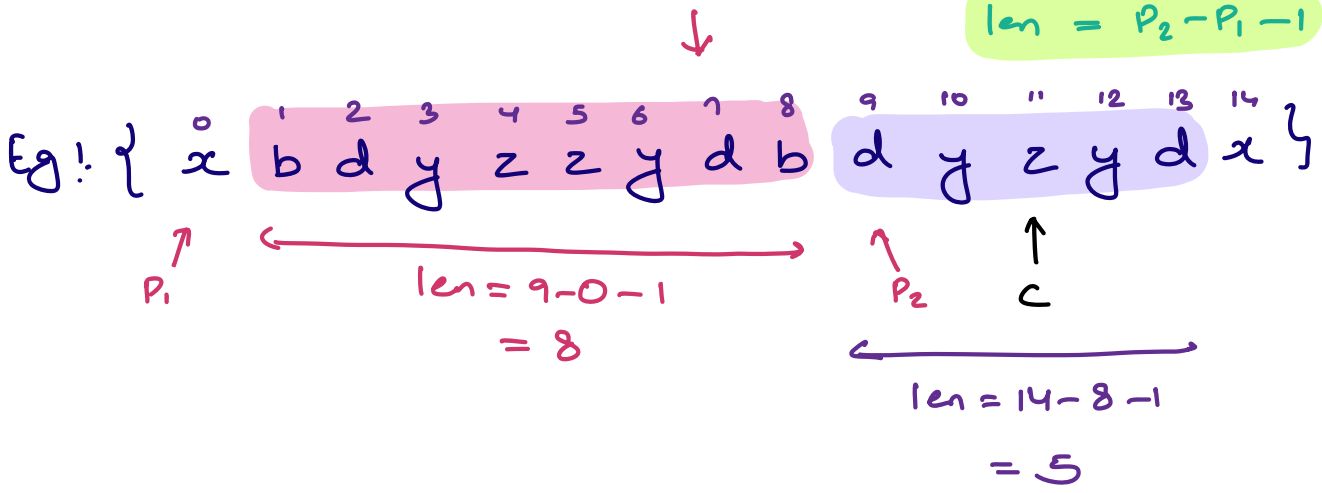


$(P_1\ P_2)$ = both included

$len = P_2 - P_1 + 1$

$(P_1\ P_2)$ = both excluded

$len = P_2 - P_1 - 1$



TC for expansion = O(n)

Each character as possible center = n

Idea 2 → Take every character as center & expand in both the direction & update our max length;

Odd len palindrome

TC ≈ O(n²)

Even len palindrome

Take every adjacent pair as centers & expand around centers in both direction & update your max palindromic length:

TC for expansion = O(n)

center = (n-1)

For even len palindrome TC ≃ $O(n^2)$

* Overall TC = $O(n^2 + n^2) \simeq O(n^2)$

```
int longpal ( char [] str )
    int ans = 0

    for ( i=0 ; i<n ; i++ ) {        // odd length palindrome
        // center = i
        P1 = i , P2 = i
        int len = expand ( str , P1 , P2 )
        ans = Math.max(ans, len);
    }

    for ( i=0 ; i<n-1 ; i++ ) {      // even length palindrome
        // center = i & i+1
        P1 = i     P2 = i+1
        int len = expand ( str , P1 , P2 )
        ans = Math.max(ans, len);
    }
}
```

TC : $O(n^2)$

SC : $O(1)$

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

TC: O(n)

Longest Palindromic substring

- Brute force — $O(n^3)$
- Expand on center — $O(n^2)$
- DP — $O(n^2)$
- Manacher's Algo — $O(n)$

Doubts

Eg :- n = 1 0 1

n>>1 = 0 1 0

& n = 1 0 1

0 0 0 → n is containing alternating zeros

n = 1 0 1 0 1 0

n>>1 = 0 1 0 1 0 1

0 0 0 0 0 0

n = 1 0 0 0 0

n>>1 = 0 1 0 0 0

1 0 0 0 0

0 0 0 0 0 → X

Sachin = if (n & 1 == (n>>1) & 1))

Sailet = val = 1

ans & val == 1

return 1

else return 0