# Todays class

→ Intro

→ Flipping

→ Sort ch[]

→ Reverse string

→ Longest Palindromic Substring.

String : → array of characters → {acb}
→ bunch of characters → {abc} ] Order matters
↘ sequence of characters

# Characters

How is a char stored ? ⇒ ASCII value

ASCII val

'A' → 65 $\xrightarrow{+32}$ 'a' - 97
65 $\xleftarrow{-32}$ 'a'

'B' → 66

'C' → 67

⋮

'Z' → 90

'a' - 97

'b' - 98

'c' - 99

⋮

'z' - 122

'0' - 48

'1' - 49

'2' - 50

⋮

'9' - 57

'10' -
↳ Not a single char

Size of char = 1 byte ⇒ 8 bits

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

⇒ 57

ch = '9' (57)

ch = ch + 8 (57+8)

print (ch) (65)

↳ A

Strings: array of characters

```
String s = "abcd"
print ( s[0])
      ↓
     'a'
```

```
char ch[] = "abcd"
print ( ch[0])
        ↳ 'a'
```

Q1: Given a char[], Toggle everything.

↳ Small ⟺ Capital

Note: char[] contains only lower case or uppercase alphabets.

Eg: ch[] = AnaConDa

Toggle: aNAcONdA

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

Solve w/o if-else

s[i] = s[i] ^ 32

or

s[i] = s[i] ^ (1<<5)

TC: O(N)

SC: O(1)

(4) HINT: 32 is a special number

| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|
| 'A'(65): | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 'B' (66): | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 'C' (67): | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| ⋮ | | | | | | | | |
| 'Z' (90): | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

| | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|
| 'a': | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 'b': | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 'c': | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| ⋮ | | | | | | | | |
| 'z': | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

Convert

Toggle the 5th bit

Upper case

5th bit is unset

Small case

5th bit is set

## Q2: Given a char [], which contains only lower case alphabets.

**Sort** the given char [] in alphabetical order

Eg: s = dabacdb

↓ Sort

aabbcdd

Constraint:

$1 <= N <= 10^5$

'a' <= ch[i] <= 'z'

ideas: ① Sort ch[] using bubble sort

TC: $O(N^2)$

↳ $10^{10}$ → TLE

② Use in-built library + custom comparator (if required):

TC: $O(N \log N)$ ✓

# idea:

s = dabacdb

'a' – 2

'b' – 2

'c' – 1

'd' – 2

⇒ ans ⇒ aa bb c dd

k

↓

a a b b b c c c e e = aabbbccccee

s = ~~a~~ ~~b~~ ~~c~~ ~~e~~ ~~b~~ ~~c~~ ~~b~~ ~~a~~ ~~c~~ ~~e~~

↑

'a' – 2

'b' – 3

'c' – 3

'd' – 0

'e' – 2

ans = aa bbb ccc ee

cnt [26] =

| 0 | 1 | 2 | 3 | 4 | 5 | 25 |
|---|---|---|---|---|---|---|
| 2 | 3 | 3 | 0 | 2 | . . . . | 0 |

idx = s[i] – 97

cnt [idx] = cnt [idx] +1

How many distinct chars: 26

```
int cnt[26] = {0}

cnt[0]  → freq of 'a' (97)
cnt[1]  → freq of 'b' (98)
cnt[2]  → freq of 'c' (99)
   :
cnt[25] → freq of 'z' (122)
```

c - 97

or

c - 'a'

or

c = c%97

```
Sort String(char s[]) {
    n = s.length
    int cnt[26] = {0}
    for(i=0; i<n; i++) { // iterate string
        idx = s[i] - 97
        cnt[idx]++
    }
    // modify original string.

    K=0
    for(i=0; i<26; i++) { // iterate cnt array.
        // cnt[i] = freq of('a'+i)
        char ch = 'a'+i
        for(j=1; j<=cnt[i]; j++) {
            s[K] = ch
            K++
        }
    }
    return s
}
```

TC: O(N)

SC: O(26) = O(1)

N x 26

TC: O(N)

SC O(1)

TC: O(N)

SC: O(1)

Table:

| i | j | #iter |
|---|---|---|
| 0 | [1 cnt[0]] | cnt[0] |
| 1 | [1 cnt[1]] | cnt[1] |
| 2 | [1 cnt[2]] | cnt[2] |
| ⋮ |  | ⋮ |
| 25 | [1 cnt[25]] | cnt[25] |

This sum = No of iter.

freq of 'a' + freq of 'b' + freq of 'c' + ....+ freq of 'z'

---

Length of string = N

Break : 8:30 am

Substring : concept is same as subarray

1. Continuous part of string
2. Full string is a substring
3. A single char is also a substring.

Qn: Check if a given substring is palindrome or not.

Eg: mam madam
dad radar
civic tenet

malayalam

```
         0 1 2 3 4 5 6 7 8 9 10
Ch[11] : a n a m a d a m s p e

[3 7]  : madam  ✓
[8 10] : spe    ✗
[0 2]  : ana    ✓
```

```
bool isPalindrome(char c[], int s, int e) {
    // Check if substring [s e] is a palindrome.

    while(s < e) {
        if(c[s] != c[e]) {
            return false
        }
        s++, e--
    }
    return true
}
```

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

Qn: Given a string, calc length of longest palindromic substring.

Eg1: a $\boxed{\text{bacab}}$

len = 5

Eg:
```
0 1 2 3 4
a b c d e
```
len = 1

#ideas:

For every substring, check if its a palindrome or not. get max length.

# of substrings × isPalindrome check

$\frac{N(N+1)}{2} \times N = O(N^3)$

$\boxed{1 <= N <= 3 \times 10^3}$

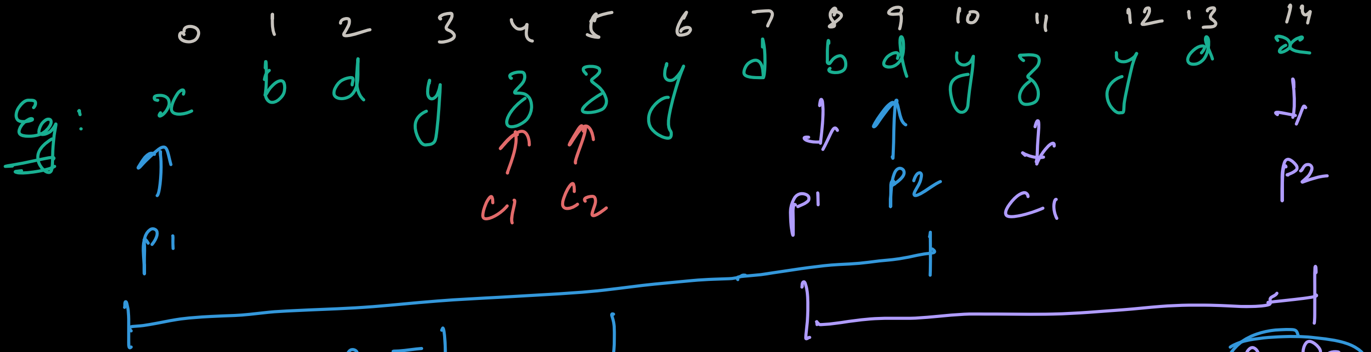
$27 \times 10^9$ ✗

```
int longPalindrome(char s[]) {
    int n = s.length
    int ans = 0
    for(i = 0; i < n ; i++) { //start
        for(j = i ; j < n ; j++) { //end
            // substring [i j]
            if(isPalindrome(s, i, j)) { //len = j-i+1
                ans = max(ans, j-i+1)
            }
        }
    }
    return ans
}
```

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

→ won't work

→ TLE



length: $p_2 - p_1 - 1$

even length:

* Take every adjacent char as centre & expand.

# Adj centre: N-1 × Expansion

TC: $O(N) \times O(N)$

$= O(N^2)$

odd length

length excluding $(p_1, p_2)$

$[p_1 \; p_2] : p_2 - p_1 + 1$

$- 2$

$\boxed{(p_1 \; p_2) \quad = p_2 - p_1 - 1}$

* Take every char as centre & expand left & right around it, get max palindromic substring

$p_2 - p_1 - 1$

TC: #No of centre * expansion

N × N

$= O(N^2)$

```
int expand(char s[], int p1, int p2) {
    while(p1>=0 && p2<N && s[p1] == s[p2] ) {
        p1 --
        p2 ++
    }
    return p2-p1 -1
}
```

p1 >= 0
p2 < N

s[-1]

TC: O(N)

```
int longPalindrome(char s[]) {
    int n= s.length
    int ans = 0
    for(i=0; i<n; i++) { // odd length          ] N^2
        // Centre: s[i]
        p1,p2 = i
        ans = max(ans, expand(s, p1,p2))
    }
                n-1
    for(i=0; i<n; i++) { // even length
        // Centre: s[i], s[i+1]
        p1 =i, p2 = i+1                          ] N^2
        ans = max (ans, expand(s, p1,p2))
    }
    return ans
}
```

i<n
i<n-1
both
works

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

Longest Palindromic Substring

✓ ① BF: $O(N^3)$

✓ ② Expand around centres $O(N^2)$

✓ ③ $O(N^2)$ : DP
↳ adv module

× ④ $O(N\log N)$ : Binary Search + Rabin Karp

⑤ $O(N)$: Man char algo
+
optimal class
for adv

Qn: Given a string, reverse the words.

Eg:- "i hate love dsa"

ans:- dsa love hate i

SC: O(1)

① Reverse each word

i etah evol asd

② Reverse whole string

dsa love hate i

"Todo"

reverse array

```
for(i=0; i<n/2; i++)
    swap(a[i], a[n-i-1])

or

s=0, e=N-1
while(s<e) {
    swap(A[s], A[e])
    s++, e--
}
```