"Art enables us to find ourselves and lose ourselves at the same time."

§. - Intro
- flip
- Sort
- palindrome

§. String:
- collection of char(s) → {a, c, b} = {a, b, c} 'acb' "abc"
- series of char(s)
- array of char(s)

order is important.

Array of char(s)

S = "Hello world"

S = ['H', 'e', 'l', 'l', 'o', '␣', 'W', 'o', 'r', 'l', 'd']

⇒ String mystring = " ____ "

'H' = ? // series of 0 and 1

⇒ How characters are stored in machines.

⇒ ASCII values ⇒ '0 to 127'

+32

−32

'A' →	65	'a' →	97	'0' →	48
'B' →	66	'b' →	98	'1' →	49
'c' →	67	'c' ⇒	99	'2' →	50
⋮	⋮	'd' ⇒	100	⋮	⋮
'Z' →	90	'z' →	122	'9' →	57
				"10" →	49 48

$\boxed{\text{Char} \quad ch = 'g'}$ → 1 Byte

37
25

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

ch1 = 'g' //57

⇒ Char ch2 = ch1 + 8 // 65

⇒ print (ch2) = ? ⇒ A

Q How to store 2digit no. as binary?

⇒ ['1' '0']

'10' ⇒ int

⇒ string

⇒ [

Q. ch = 'g' + '8' = ? ASCII

57 + 56 ⇒ 113

[?*)

⇒ Open non txt file in text editor?

⇒

⇒ an image MS paint → renamed it → .txt

string st=? ch[] st=?

Print "Hello World".

Q. Given a string of alphabets (lower case and upper case characters), toggle the case of each letter (upper case ⇄ lower case)

ex. " Hello" ⟹ "hELLO"

idea ① if upper case → lower case //add 32
② if lowercase → upper case // − 32

```
void toggle ( ch[] st ){
    int N = st.size();
    for (int i=0; i<N; i++ ){
        if ( st[i] ≥ 65 and st[i] ≤ 90){
            st[i] += 32
        }
        if ( st[i] ≥ 97 and st[i] ≤ 122){
            st[i] −= 32
        }
```

'A' 'Z' 'a' 'z'

```
    }
    return;
}
```

"Hello 123" ⇒ "hELLO 123"

(*) Solve the same problem without if/else

⇒ switch , ?:

'A' 65	0	1	0	0	0	0	0	1		'a' 97	0	1	1	0	0	0	0	1
	7	6	5	4	3	2	1	0										
'B' 66	0	1	0	0	0	0	1	0	+32 →	'b' 98	0	1	1	0	0	0	1	0
'C' 67	0	1	0	0	0	0	1	1		'c' 99	0	1	1	0	0	0	1	1

64+1

Obs^n ⇒ ① If capital letter unset 5th bit

⇒ ② If small letter set 5th bit

XOR ^ 1

```
void toggle (ch[] st){
    int N = st.size();
    for (int i=0; i<N; i++){
        st[i] = st[i] ^ (1<<5)
    }
    return;
}
```

input constraints
all the chars are alphabets

st ^ 32

Q. Given a char array ch[], which contains only lower case alphabets. sort the array. → alphabetical order

Ex. S = "d a c d d b b a c"

S = "a a b b c c d d d"

Constraints, $1 \leq N \leq 10^5$ → string size

$'a' \leq S[i] \leq 'z'$

BF. ① library sort function

sort (S);

T.C. = $O(N \log N)$
S.C. = $O(N)$ → merge sort

② Bubble sort:

T.C. = $O(N^2)$ → TLE $O(10^{10})$
S.C. = $O(1)$

③ optimal sol^n:

ex. ⇒ S = "d a c d d b b a c"

'a' ⇒ 2
'b' ⇒ 2

⇒ sorted string ?

'c' ⇒ 2
'd' ⇒ 3

"aabbccddd"

'a' → (−97) ⇒ 0
'b' → (−97) ⇒ 1
'c' → (−97) ⇒ 2
⋮
'y' → (−97) ⇒ 24
'z' → (−97) ⇒ 25

int count[26] = {0}

count[0] ⇒ count of 'a'
count[1] ⇒ count of 'b'
⋮ ? ⋮

'b' → count[1]

ch == 98 ⇒ count[i]++;

count[ch − 97]

ex. ⇒ S = "d a c d d b b a c"

	a	b	c	d								
count =	0	0	0	0	0	....	0	0	0	0
	0	1	2	3	4	....	22	23	24	25

freq 'a' freq 'b' freq 'z'

count ['d' − 97] ++;

'd' → 100

'a'

count [ch − 'a'] ++;

```
void sortString (char[] s) {
    int N = s.size(); char ch;
```

```
int c[26] = {0}   // S.C.

O(N) →  for (i=0; i<N; i++) {
            int index = S[i] - 97
            c[index]++;
        }

O(N) →  for (i=0; i<26; i++) {
            ch = 'a' + i;
            for (int j=0; j<c[i]; j++) {
                print(ch);
            }
        }
        return;
}
```

'a' - 97

'c' - 97

S = "d a c d d b b a c"

N = 9

	a	b	c	d	e	. . . . .	y	z
C =	2	2	2	3	0	0	0	0
	0	1	2	3	4	. . . . .	24	25

$i = \cancel{0} \to \cancel{1} \to 3$

$ch \Rightarrow 'a' + \cancel{0}^{1} \Rightarrow 97 + \cancel{0}^{1} \Rightarrow \cancel{97}\ 98 \xrightarrow{\text{char}} \cancel{'a'}\ \cancel{'b'}\ 'd'$

$c[i] \Rightarrow \cancel{2}\ \cancel{2}\ 3$

"aabbccddd"

i	j	iteration
0	[0, c[0])	c[0]
1	[0, c[1])	+ c[1]
⋮	⋮	+ ⋮
25	[0, c[25]	c[25]

total iterations = sum of frequency of each char

$= C[0] + C[1] + \ldots C[25]$

$= N$

Total T.C = $O(N)$

S.C = $O(26) \Rightarrow O(1)$

Bucket Sort better than library function.

also known as counting sort

5. Substring : (continous part of a string)

"Hello" $\Rightarrow$ "eo" ? ✗

Q. Given a string S, given starting and end point of the substring, check whether the substring is palindrome or not.

palindrome : naman

n a m a n

reverse

madam | madam

Hello | olleH

ex. S = "abcdcbz", s = 1, e = 5

0 1 2 3 4 5 6

b c d c b

ans = ✓

10:40

```
bool isPalindrome (char s[], int start, int end) {
    while (start < end) {
        if (s[start] != s[end]) {
            return false
        }
```

```
        else {
            start++;
            end--
        }
    }
    return true;
}
```

substring length

T.C = O(N)

S.C = O(1)

Q. Given a string S, find the length of the longest palindromic substring in S.

ex. S = "anamadam", ans = 5

a n

Brute force approach: ① for all the substrings.
② check if it's a palindrome

```
int longestPalindrome (char[] St) {
    int N = St.size();
```

```
int maxLength = 0;
for (int s=0; s<N; s++) {
        for (int e=s; e<N; e++) {
            if (isPalindrome(St, s, e)) {
                maxlength = max(malength, e-s+1)
            }
        }
}
return maxlength;
}
```

T.C. = $O(N^3)$
S.C. = $O(1)$

[e d b a a b c f]
ex:

Optimised sol^n:

```
int longestPalindrome (char[] St) {
    int N = St.size();
```

```
int maxLength = 0;

for (int i=0; i< N; i++) {

    // checking only odd length
    //i is the center of my string

    left = i;
    right = i;

    while (left >0 and right<N) {

        if (st[left] != st[right]) {
            break;
        }
        left--;
        right++;
    }
    maxlength = max (maxlength, right-left+1)
```

```
// even length
    left = i
    right = i+1;
while (left ≥ 0 and right < N) {
        if (st[left] != st[right]) {
            break;
        }
        left++;      left--;
        right--;     right++;
}
}   maxlength = max(maxlength, right-left+1)
return maxlength;
}
```

T.C. = $O(N^2)$

S.C. = $O(1)$

St = NITINPQ

ans ⇒ ~~0~~ 1

DP ⇒ Recursion + Memory