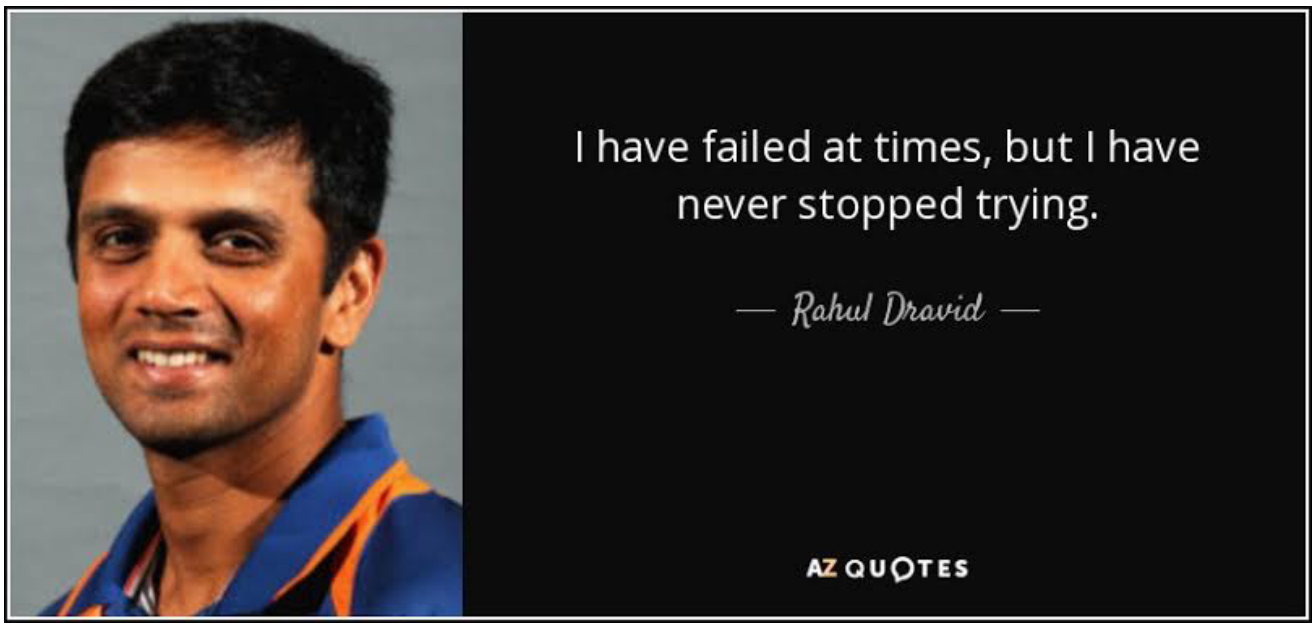


Agenda

1. String Basics
2. Flip
3. Sort
4. Reverse String
5. Longest Palindromic Substring

String

→ Array of characters
→ Sequence of character
→ Bunch of character → {acb} {abc} } order is very imp.

Characters

- single unit of data that is alphabet, digit & special symbol

Standards – ASCII – standard for representing character as numerical value

'A' – 65	'a' – 97	'0' – 48
'B' – 66	'b' – 98	'1' – 49
'C' – 67	'c' – 99	'2' – 50
⋮	⋮	⋮
'Z' – 90	'z' – 122	'9' – 57

```
char ch = "9"
ch = char(ch + 8)
   = char(57 + 8) = char(65) = 'A'
```

Strings → array of characters

```
string s = "a b d a"  = {a, b, d, a}
                        0  1  2  3
print(s[2]) = d
```

1Q) Given a character array, Toggle every character

↳ Capital ⇄ small

Example

→ AnaConDa [aNAcONdA]

Constraints

a <= ch <= z
A <= ch <= Z

You will only get Alphabets

Pseudo Code

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

Observation

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

'B' - 66 $\xrightarrow{+32}$ / $\xleftarrow{-32}$ 'b' - 98

'C' - 67 $\xrightarrow{+32}$ / $\xleftarrow{-32}$ 'c' - 99

'D' - 68 ⋮ 'd' - 100 ⋮

Time Complexity

O(N)

Space Complexity

O(1)

2Q) Given a char array, which contains only lower case characters, sort given array in alphabetic order

Example

s = d a b a c d b

Ans = a a b b c d d

Constraints

$1 <= N <= 10^5$

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

Brute force

Bubble sort – $O(N^2)$

comparitors methods – $O(n \log n)$ ✗

sort (start_index , end_ind , custom_com)

Observation I am interested in knowing frequency of each cha

s = d a b a c d b ---------- a a b b c d d

Example 1

'a' – 2
'b' – 2
'c' – 1
'd' – 2

aa – bb – c – dd

d a b a c d b

a a b b c d d

overwrite the value to get result

Example 2

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

(97) 'a' – 2
(98) 'b' – 3
(99) 'c' – 3
(100) 'd' – 0
(101) 'e' – 2

count

0	1	2	3	4	...	25
2	3	3	0			

'a' – 97 → 0th index
'b' – 97 → 1st index
'c' – 97 → 2nd index

Given the char value, I can use it to find the Index

pseudo code

① Creating a frequency array for given string

② using the frequency array I am generating sorting

ord

```
string sortstring ( char [] s ) {
    int n = s.length();
    int count[26] = {0}
    for (i=0 → n) {
        ind = s[i] - 97 // freq. arr index
        count[ind] += 1;
    }
    k=0
    for ( int i=0 → 26){
        // count[i] → represent the frequency
        char ch = 'a' + i;
        for ( j=1 → count[i] ) {
            s[k] = ch    k++
        }
    }
    return s
}
```

T.C = O(N)

S.C = O(26)
= O(1)

T.C
O(N)

S.C
O(1)

S = a b c e b c b a c e

Count =

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

Time Complexity

i	j : [1 - c[i]]	Total Iteration
0	[1 - c[0]]	c[0]
1	[1 - c[1]]	c[1]
2	[1 - c[2]]	c[2]
⋮	⋮	
25	[1 - c[25]]	c[25]
		O(n)

Total Iteration

= c[0] + c[1] + c[2] ...
+ c[25]

count of all freq

= string length

T.C = O(N) + O(N) = O(N)

Substring concept is same as subarray

↳ contiguous part of string

↳ Full string can be substring

↳ A single character can also be a substring

Subarray concept on strings is substringe

Q2) check if a given substr is **palindrome** or not

L → R

R → L

Obs:
if given string is palindrom first & last ch in 2 point should be same

Example

madam

mom

dad

level

civic

```
char ch[] = a n a m a d a m s p e
            0 1 2 3 4 5 6 7 8 9 10
```

```
boolean isPalindrome( ch[], s, e) {
                      ↓ start index
                        ↑ end index
{
    while (s < e) {
        if ( ch[s] != ch[e] ) {
            return False
        }
        s = s+1;    e = e-1;
    }
    return True;
}
```

TC = O(N)

SC = O(1)

4Q) Given a string, Calculate the length of longest palindromic substring

Ex

S = a b a c a b

output

5

Ex

S = a b c d e

output

1

Most Brute Force

1. Generating all sub strings
2. Above code to check if palindrome and count the length. Find max length

- $1 <= N <= 3 \times 10^3$

Pseudo Code

```
int long pal ( char s[] ) {
    int n = s.length;
    int ans = 0;
    for ( i=0; i< n; i++) { // start index
        for ( j= i ; j< n ; j++) { // end index
            if (is palindrome (s, i, j) {
                ans = max (ans , (j-i+1));
            }
```

```
        }
    }
    return ans
}
```

T.C = $O(n^3)$

= Time to generate all substring = $O(N^2)$

= Time to check pal = $O(N)$

= $O(N^3)$

S.C = $O(1)$

Interesting Observation

I give you mid point of ans

x b d y z z y d b d y z y d x

P_1 P_2

Given a palindrome You can split it into 2 half

If you are given the centre, you can find the length of palin-drome

length of palindrome

$[P_1, P_2] = P_2 - P_1 + 1$

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

$= P_2 - P_1 - 1$

Idea

→ Take every character as center and expand the centre and find length of palindrome

$$TC = O(N) \times O(N) = O(N^2)$$

palindrome with even character

even

x b d y z z y d b d y z y d x

P_1 P_2

$$P_2 - P_1 - 1$$

$$[P_1, P_2] = P_2 - P_1 + 1$$

Pseudo code

① We should consider center for odd length palindrome & even length palindrome separate

② Use same mechanism to expand the center till the string is palindrome

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

int longPal ( char s[]) {
    int n = s.length;
    int ans =0
    // palindrome with odd length
    for (int i=0; i<n; i++) {
        // center = s[i]
          p1 = i
          p2 = i
          ans = max( ans, expand (s, p1, p2)
    }
    // palindrome with even length
    for (int i=0; i<n-1; i++) {
        // center s[i], s[i+1]
          p1 = i
          p2 = i+1
```

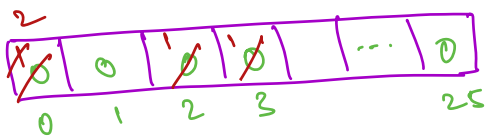
```
            ans = max(ans, expand(s, p1, p2))
        }
        return ans
    }
```

Doubts Session

```
for (i=0 → n) {
    ind = s[i] - 97 // freq. arr index
    count[ind] += 1;
}
```

S = a c d e a e

a - 2
b - 0
c - 1
d - 1
e - 2



I will store the freq(a) in 0th index

97 → 0 = 'a' - 97 97
98 → 1 = 'b' - 97 98
99 → 2 = 'c' - 97 99
⋮

freq(a) → count[0]
freq(b) → count[1]
⋮
freq(z) → count[25]

[1, 2, 3]

[sabarish]

S = reverse (a)

sub array

{}

{1} {2} {3}

{1, 2} {2, 3}

{1, 2, 3}