Strings :-

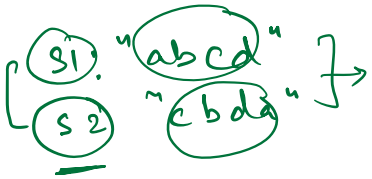
↳ Array of characters ✓

→ ~~Group~~ " " ~

→ ~~Collection~~ " " ✓

→ List " "

→ ~~Set~~ " " ✓

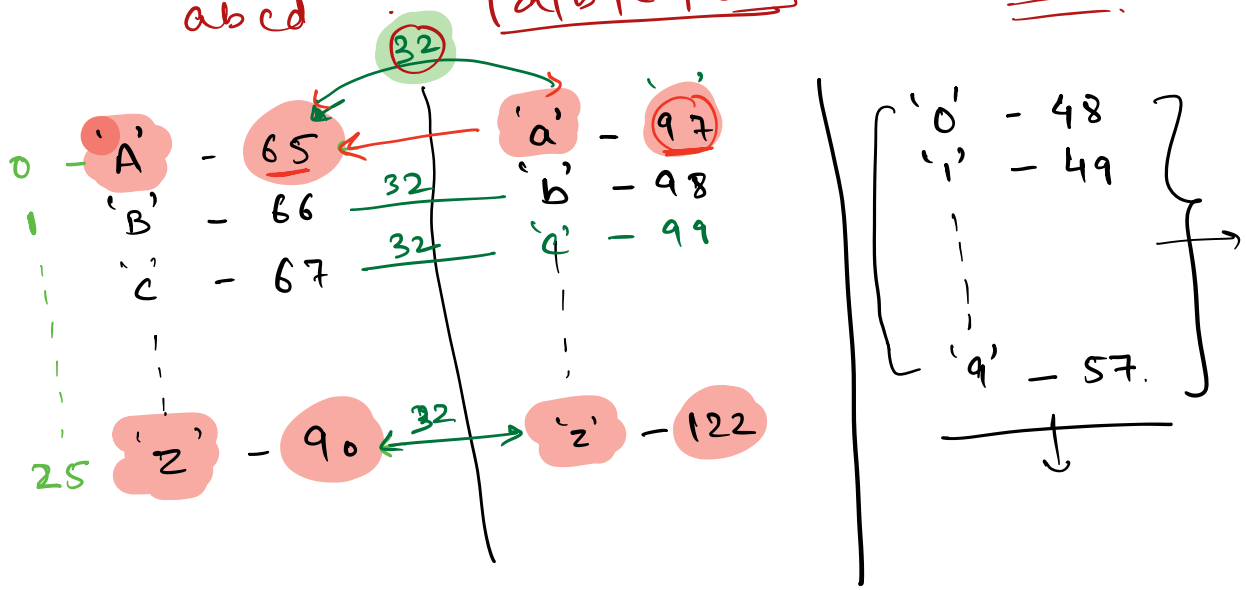


⇒ Sequential order of characters.

Strings :- Char a[ ].

"abcd" : | a | b | c | d |

"abcd" ↓ ASCII



Q.1 Given a String S, Toggle the case of every character.

S = "aBcDeF" → "AbCdEf".

Capital → small
small → capital.

'A' 'Z'

1) S[i] ≥ 65 && S[i] ≤ 90 → Capital char.
2) S[i] ≥ 97 && S[i] ≤ 122 → small char.

97 → 'a'
122 → 'z'

```
for (i=0 ; i< s.size() ; i++) {
    // small case chars to Capital case.
    if( s[i] ≥ 'a' && s[i] ≤ 'z' ) {
        s[i] -= 32;
    }
    else {
        s[i] += 32;
    }
}
```

s[i] ^ 32

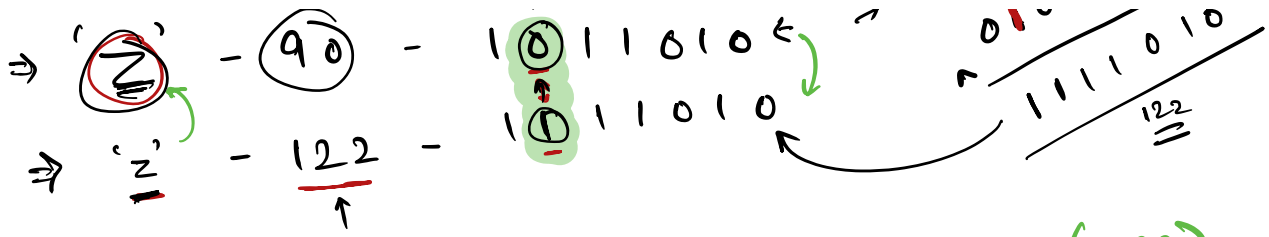
'a' - 'A' | 'b' - 'B'

32 → 2^5 ⇒ 0 0 0 0 0 (bits 4 3 2 1 0)

XOR

1 0 1 1 0 1 0 ← 90
0 0 0 0 0 0 0

6 5 4 3 2 1 0



1) Small case → Capital case (-32)
⇒ Unset 5th Bit

2) Capital case → Small case
→ Set 5th Bit

6 5 4 3 2 1 0
'Z' - 90 - 1011010 → XOR '32 → 1011010
'z' - 122 - 1111010 → 100000

122 ← 1111010
0100000
1011010

Q. Given a string S, Contains only lower case characters. Sort the string. (N)

s : d a b c a b c d.

⇒ a a b b c c d d.

1) sort(S) → sorted string

⇒ O(N log N)

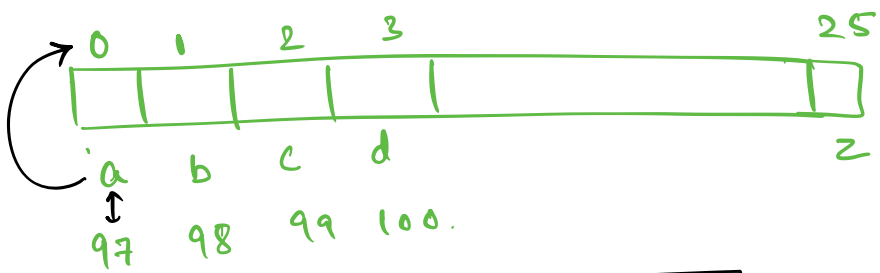
2) Lower case characters → 26 unique characters.

[a - z]

26

⇒ a b c a b d c a d.

int count[26] = {0};



index = S[i] - 'a'

'a' ⇒ index = 0

'b' ⇒ index = 1

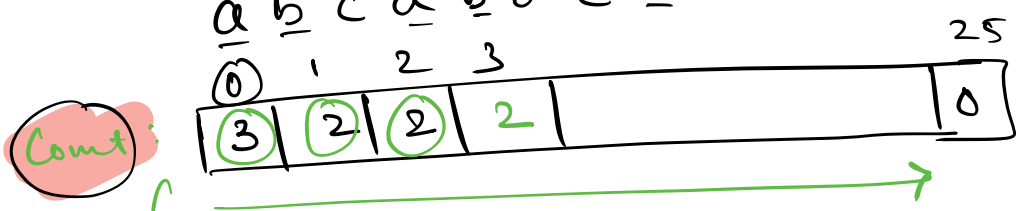
⋮

'z' ⇒ index = 25

```
int count[26] = {0};
for (i = 0; i < N; i++) {
    int indem = S[i] - 'a';
    Count[indem]++;
}
```

→ O(N)

a b c a b d c a d .



→ a a a b b c c d d .

```
for (i = 0; i < 26; i++) {
    for (j = 0; j < C[i]; j++)
        ans.append(i + 'a')
}
```

O(N)

T.C ⇒ O(N)

S.C ⇒ O(1)

COUNT-SORT

a b c a b d c

i	iteration
0	c[0]
1	c[1]
2	⋮
3	⋮
4	⋮
⋮	⋮
25	c[25]

Total iterations: $\sum_{i=0}^{25} c[i]$

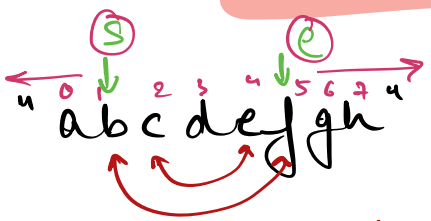
Sum of all the frequencies of char's.

⇒ 'N' iteration

Q. Given a string & a start (s) and end (e) index, Reverse the part of string from (s) to (e).

↓

Substring ≃ Subarray.



⇒ afedcbgh.

e > s

s = 1,
e = 5,
swap(b, f)
s = 2
e = 4
swap(c, e)
s =

```
reverse(str, s, e) {
    while(s < e) {
        swap(str[s], str[e]);
        s++;
        e--;
    }
}
```

TC ⇒ O(N)
SC ⇒ O(1)

Amazon | Adobe | MS. - - - -

Q. Given a string of sentence.

- No extra spaces at beginning/end.
- Every word is separated by a single space
- [a-z] lower case chars.

⇒ Reverse this string word by word.

* No inbuilt fun^ is allowed.
* Constant extra space. ✓

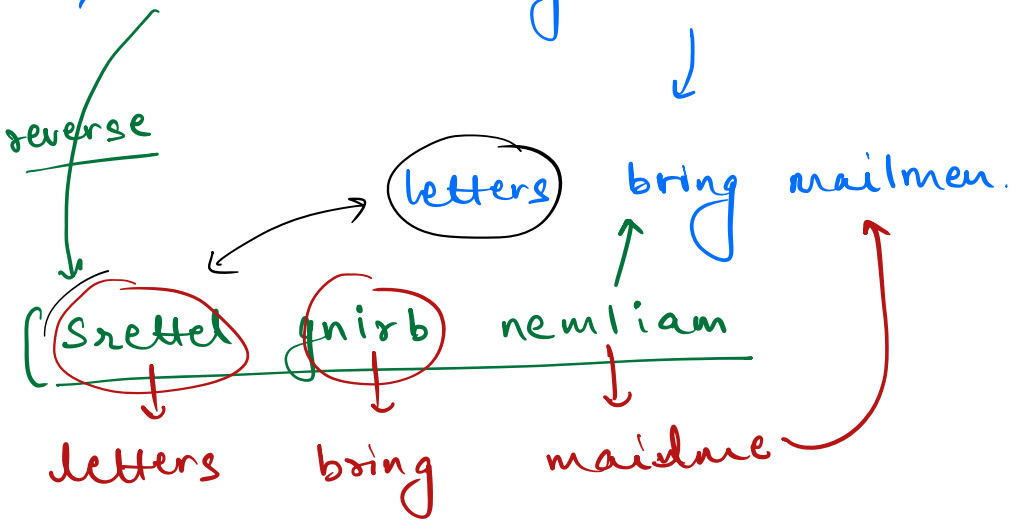
"here is a picture"

↳ "picture a is here"

⇒ "are you as clever as I am"

↳ "am I as clever as you are"

⇒ mailmen bring letters →

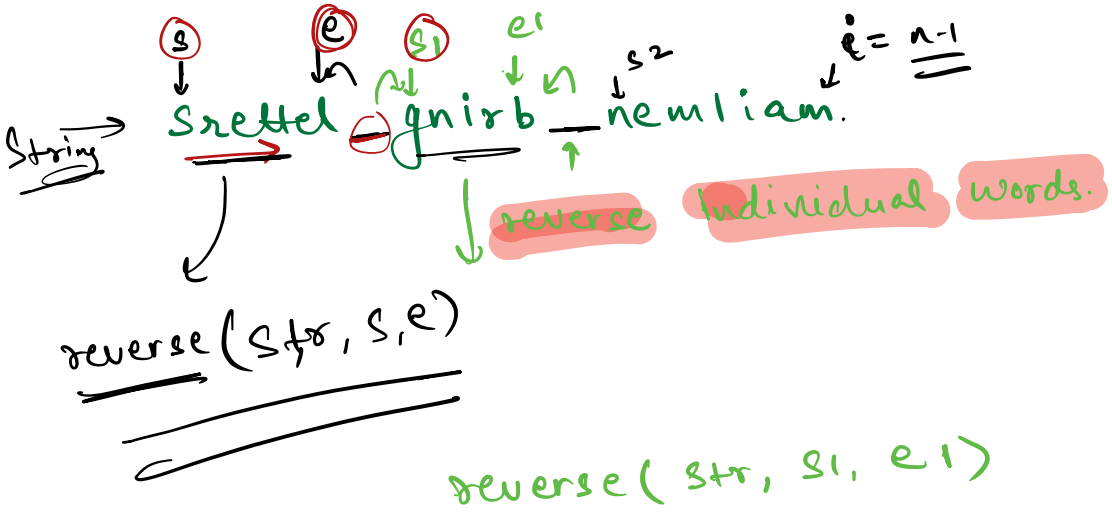


#

Step I :- Reverse complete string. → O(N)

↳ reverse(s).

Step II :- Reverse individual words. →



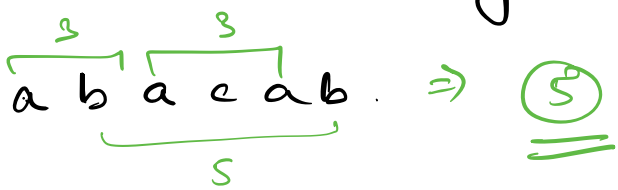
HW :- Implement this method.

TC →

SC →

Q: Given a String. Calculate the length of largest Palindromic substring.

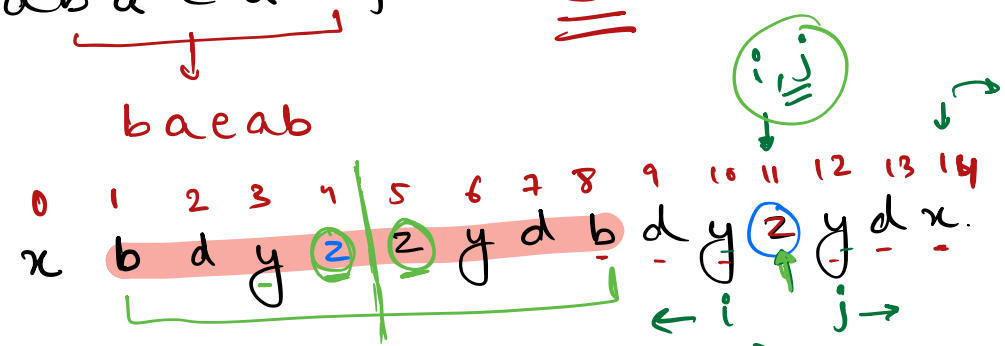
Amazon | GS | Ola | Directi | MMT



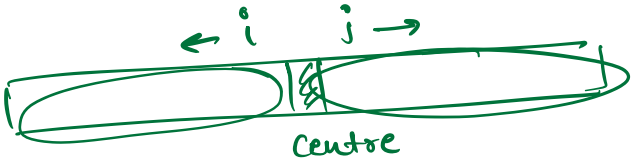
Quiz :- a b a e a b f → 5

baeab

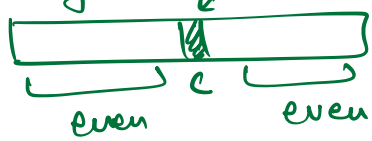
Quiz :-

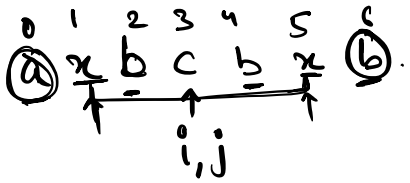



```
→ int palindromeLen(s, ci, cj) {
    i = ci , j = cj;
    while(i >= 0 && j < N && s[i] == s[j]) {
        i--;
        j++;
    }
    return j - i - 1;
}
```



1. Odd length palind.





i = 3 i = 2 i = 1 i = 0

j = 3 j = 4 j = 5 j = 6

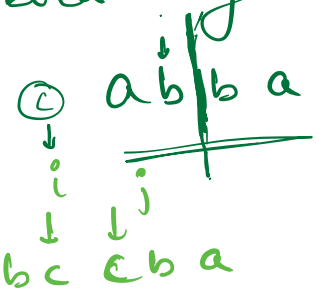
(0, 6)

(i, j)

↓

[i+1, j-1]

2. Even length palind.



i = c
j = c+1

```
ans = INT_MIN;
for (i = 0; i < N; i++) {
    // odd length palindrome
    ans = max(ans, palindromeLen(S,i,i));
    // Even length
    ans = max(ans, palindromeLen(S,i,i+1));
}
```

i+1 = N

TC ⇒ O(N²)
SC ⇒ O(1)

DP

TC : O(N²)
SC : O(N²)

Brute force :- $O(N^3)$

→ for every substring, check if palindrom

↓

$= O(N^2)$

$O(N)$ Algo → Manacher's Algorithm.

Strings in JAVA :-

↳ Immutable in java.

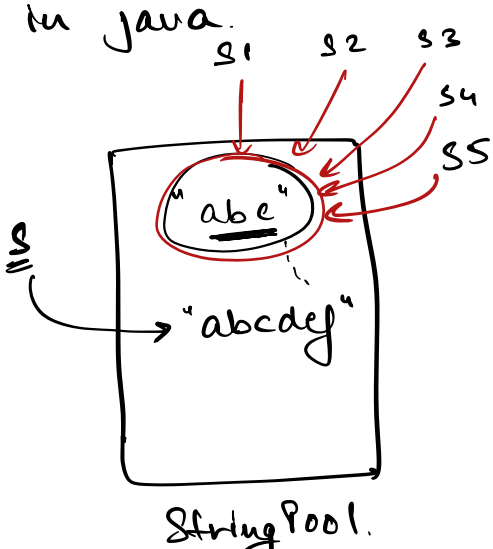
String s = "abc"

s[0] = 'd'; ✗

s = s + "def"

↓

abcdef.



StringPool.

StringBuilder sb = new StringBuilder();

sb.append('a')

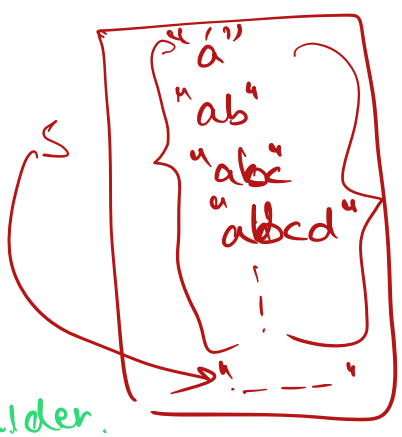
——— ('b')

String s = "a"

= "a" + "b" = "ab"

= "ab" + "c" = "abc"

N times



StringBuffer / StringBuilder.

0 1 2 3 4 5 6

String s = "abc def";

s[3] = 'x'; ✗