Strings ⇒ sequence / stream of character
array of charact

"mohit" "feb" "tom" "1234"

ASCII codes
└ 256

1000001 ⇐ 'A' — 65
'B' — 66
'C' — 67
⋮
'Z' — 90

'a' — 97
'b' — 98
'c' — 99
⋮
'z' — 122

'0' — 48
'1' — 49
'2' — 50
⋮
'9' — 57

char → integers → 0 & 1

'10' → ✗
char ✗

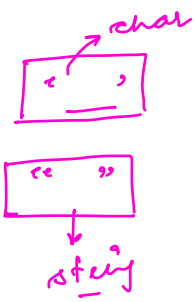
char temp = '1' ⇒ 49
int temp = 1;

Given a string S. Toggle the case of every character.

(english)

inbuilt()

upper case ⟶ lower case
lower case ⟶ upper case



" aAed Ef " = " AaEDeF "

'A' ⟶ 65	⟷ 32 ⟷	'a' ⟶ 97
'B' ⟶ 66	⟶ 32 ⟶	'b' ⟶ 98
'C' ⟶ 67	⟶ 32 ⟶	'c' ⟶ 99

capital / upper ⟶ (+32) ⟶ lower-case
lower-case ⟶ (−32) ⟶ capital / upper

(S)

s[i] → character ⇒ int

```
for ( i=0;  i<len;  i++)
{
    if( s[i] >='A' && s[i]<='Z')
    {
        // uppercase → lowercase
        s[i] += 32;
    }
    else
    {
        // lowercase → uppercase
        s[i] -= 32;
    }
}
```

T.C: O(N)

S.C: O(1) — mutable

immutable ⇒ O(N)

java / C# / pyn
|
immutable

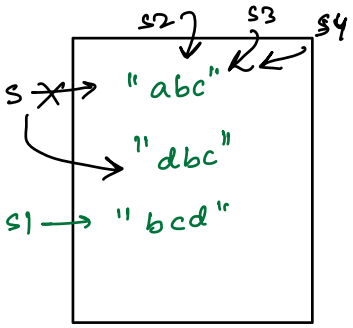
Strings are immutable

↓

changed

String s = "abc"

s[0] = 'd'



string pool

s1 = "bcd"

s2 = "abc"

s3 = "abc"

s4 = "abc"

s1[0] = 'd'

appending characters

1
2
3
4
:
N

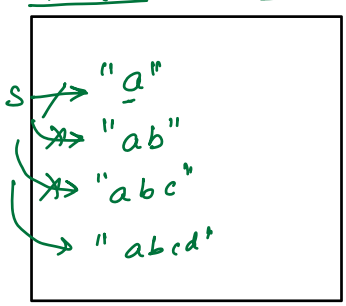
s = "a"

s = s + 'b' → O(1) → O(len)

s = s + 'c'

1
2
3
4
N

N * (N+1)/2 ≈ O(N²)

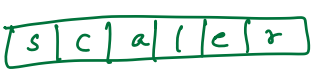


appen N char

T.C: O(N²) S.C: O(N²)

String Builder → array

"scaler"



5 4 3 2 1 0

'A' = 65 = 64 + 1 → 1 0 0 0 0 1

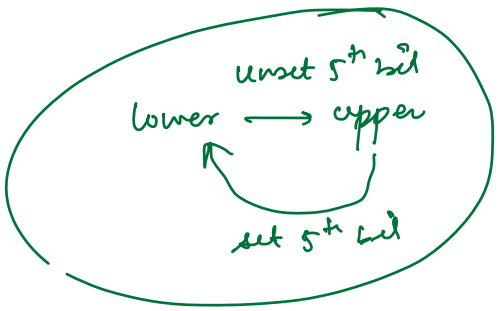
'B' = 66 = 64 + 2

'a' = 97 = 64 + 32 + 1 → 1 1 0 0 0 1

'b' = 98 = 64 + 32 + 2

5th bit → set ≡ lowercase

→ unset ≡ uppercase



≡ toggle state

↓

xor

s[i] ^= 32

given a string s . sort this string

↓

lowercase english alphabets ≈ (26)

sort ( )

s: g d a b a c d b e d

T.C: nlogn

⇓

a a b b c d d d e g

97+0 = 'a'

'a'+0 → 'a'

'a'+1 → 'b'

'a'+2 → 'c'

⋮

'a'+25 → 'z'

```
for( i=0; i<26; i++)
{
    char x = i+'a';
    // find freq -
    for( i=0→n)
    {
    }
}
```

T.C: 26*N

}

⇓

g d a b a c d b e d

arr[5] = ___ .

freq['c'] = 7;

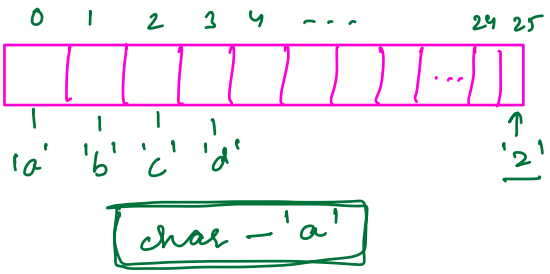
'a'

freq[97]

97→122

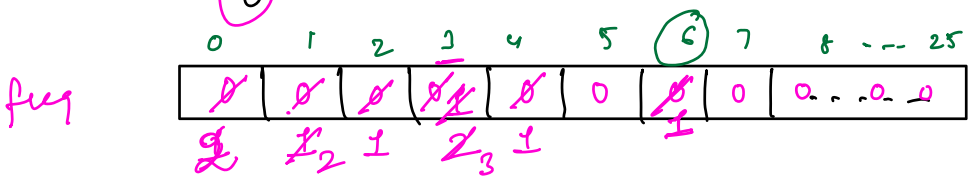
a: ~~0~~ ~~1~~ 2
b: ~~0~~ ~~1~~ 2
c: ~~0~~ 1
d: ~~0~~ ~~1~~ ~~2~~ 3
e: ~~0~~ 1
f: 0
g: ~~0~~ 1

'a' − 'a' = 0 → 97−97
'b' − 'a' = 1 → 98−97
'c' − 'a' = 2 → 99−97
'd' − 'a' = 3 → 100−97
'e' − 'a' = 4



char − 'a'

g d a b a c d b e d



freq['g' - 'a']++;

↓

103 - 97 = 6

2 2 1 3 1 0 1 0 0 ...

maintain freq array

```
int freq[26];   // initialising array with 0.

for( i=0; i<n; i++)
{
                freq[ s[i] - 'a' ]++;
}
```

O(N)
O(26) = O(1)

|a| - freq of a

```
for( i=0; i<26; i++)
{
    char x = i + 'a';
    for( j=0; j<freq[i]; j++)
    {
        print(x); // ans += x
    }
}
```

i		
0	0 - |a|	|a|
1	0 - |b|	|b|
2	0 - |c|	|c|
3		|d|
:		|e|
25		:

Total no of characters

T.C: O(N)

S.C: O(26)

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

↓ freq array ↓ sort using freq array

S.C.: $O(26) + O(1)$ / $O(N)$ → return the sorted string

$O(1)$ → mutable

$O(N)$ → Immutable

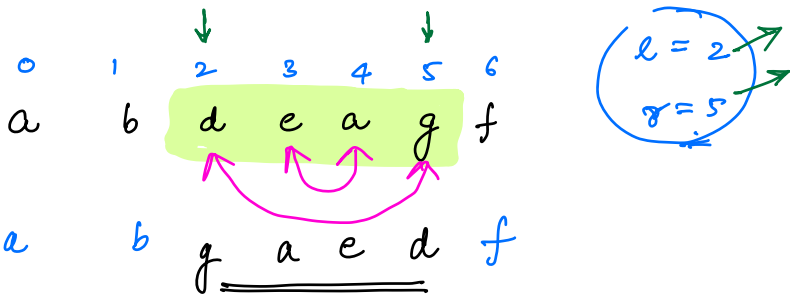
Given a string s and two indices l and r.

Reverse the substring from l to r.

subarray / contiguous

dea ✓

deg ✗




```
void reverse( s, l, r)
{  while ( l < r )
   {  swap( s[l], s[r]);
      l++;
      r--;
   }
}
```

Amazon

Q array of characters → form a sentence

↓

Reverse word by word.

No extra space

every word is seperated by a single space

there are no extra spaces at the beginning or at end.

1. "here_is_a_boy"

h	e	r	e		i	s		a		b	o	y

boy a is here

② "Are you as clever as I am"

↓

"am I as clever as you Are"

③ "mailman bring letters"

↓

"letters bring mailman"

reverse the whole string

"srettel_gnirb_niamliam"

reverse word by word

letters bring mailman

l = 0 , r = 0

while (r < N → len(s))

{

while (s[r] != '_' && r < n)

{ r++;

}

→ because for the last word, there is no space to end the word

l → r − 1

reverse(s, l, r−1);

l = r + 1;

r = l;

}

"srettel _ gnirb _ niam liam"

letters bring

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

O(N) → reverse the whole string

O(N) → reverse word by word

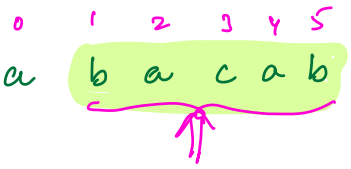
S.C: O(1)

Amazon Direct OA

Given with a string s. (lowercase alphabets)
Find length of longest palindromic substring.

dad
madam
nitin
malayalam

s = reverse(s)



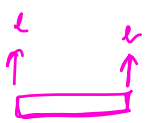
a b c a l m

ans = 1

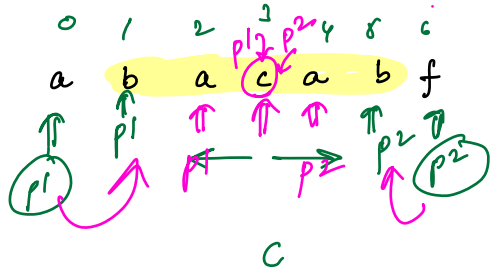
B-F:- consider all substrings (start, end)

```
for( int l=0; l<n; l++)
{
    for( int r = l; r<n; r++)
    {
        // l-r
        if( pal (s, l, r))
        {
            // if pal update your ans
            ans = max (ans, r-l+1);
        }
    }
}
```

T.C: N^3



palindrome center



s[p1] == s[p2]

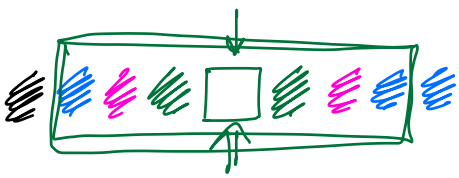
p2-p1-i

p1+1
p2-1

b - a + 1

(p2-1) - (p1+1) + 1

p2-p1-1

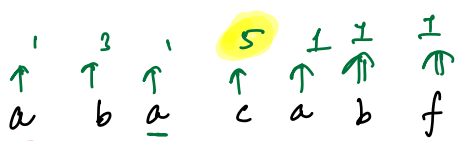


```
int expand( string s, int p1, int p2)
{
    while (p1>=0 && p2<N && s[p1] == s[p2])
    {
        p1--;
        p2++;
    }
    return p2-p1-1;
}
```

diff optn for ctr.

N * O(N)

≈ O(N²)



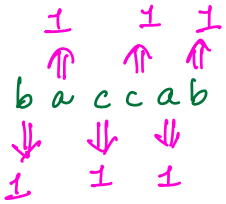
'a' → 1

'b' → 3

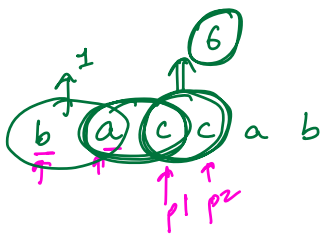
'a' → 1

odd

even length palin



ans = 6



baccabe

odd length palindrome ⇒ every element as a single center

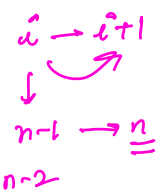
even palindrome ⇒ every 2 elements

```
ans = 0;
// odd
for( int i=0; i<n; i++)
{
    ans = max ( expand( s, i, i) , ans);
}
```

N*N

```
// even
for( int i=0; i<n-1; i++)
{
    ans = max(expand( s, i, i+1), ans)
}
```

N*N



T.C: N^2

$O(N^2)$ → dp

O(N) ⇒ Manacher's algo