

## Char Array & Strings - 2

Q1 → Remove all adjacent duplicates in string →

1047  
leetcode

i/p → ~~a~~~~b~~~~b~~~~a~~ c a

⇒ ~~a~~~~a~~ c a

⇒ ~~c~~~~a~~ ~~c~~ out of string  
return the ans.

string 

a	b	b	a	c	a
---	---	---	---	---	---

ans = "" → empty string

can we push 'a' in the ans string? → Yes, because the last character is not 'a'.

ans ⇒ "a"

can we push 'b'? → Yes

ans ⇒ "ab"

can we push 'b' → NO,

because the last character we

pushed is 'b'. And we have to remove that also.

ans → "a"

ans → ""

ans → "c"

ans → "ca" no element left, this is the ans.

↓ ↓ ↓ ↓  
a z x x z y

⇒ ~~a~~~~z~~~~z~~ y

⇒ ~~a~~~~y~~ ans

a	z	x	x	z	y
---	---	---	---	---	---

ans = ""

= "a"

= "az"

⇒ ~~'azx'~~

⇒ ~~'az'~~

⇒ ~~'a'~~

⇒ "ay"

code -

```
string removeDuplicates(string s){
```

```
    string ans = "";
```

```
    int i = 0;
```

```
    while (i < s.length()){
```

```
        if (ans.length() > 0){
```

```
            if (ans[ans.length()-1] == s[i]){
```

```
                ans.pop-back();
```

```
            }
```

```
        else {
```

```
            ans.push-back(s[i]);
```

```
        }
```

```
    }
```

```
    ans.push-back(s[i]);
```

```
}
```

```
return ans;
```

```
}
```

dry run

s.length = 6

i = 0

0 < 6 → T

if (ans.length() > 0) → F

↳ ans.push\_back(s[i])

i = 1

1 < 6 → T

ans.length = 1

s[i] == ans[ans.length()-1] → F

↳ insert b.

i = 2

2 < 6 → T

s[i] == ans[ans.length()-1] → T

↳ pop\_back() → b is removed

i = 3

3 < 6 → T

s[i] == ans[ans.length()-1] → T

↳ pop\_back() → a is removed

i = 4

4 < 6 → T

~~s[i] == ans[ans.length()-1]~~ ans.length() > 0 → F

push\_back(s[i]) → c is inserted

i = 5

5 < 6 → T

s[i] == ans[ans.length()-1] → F

push\_back(s[i]) → a is inserted.

i = 6, 6 < 6 → F, return ans.

Q2 → Remove all occurrences of a substring:-

(9/10)

s → d a a b c b a a b c b c

part = abc

d a b a b c

⇒ d a b

string removeOccurrences(string s, string part)

{

int pos = s.find(part);

while (pos != npos) {

s.erase(pos, pos + part.length());

pos = s.find(part);

}

return s;

}

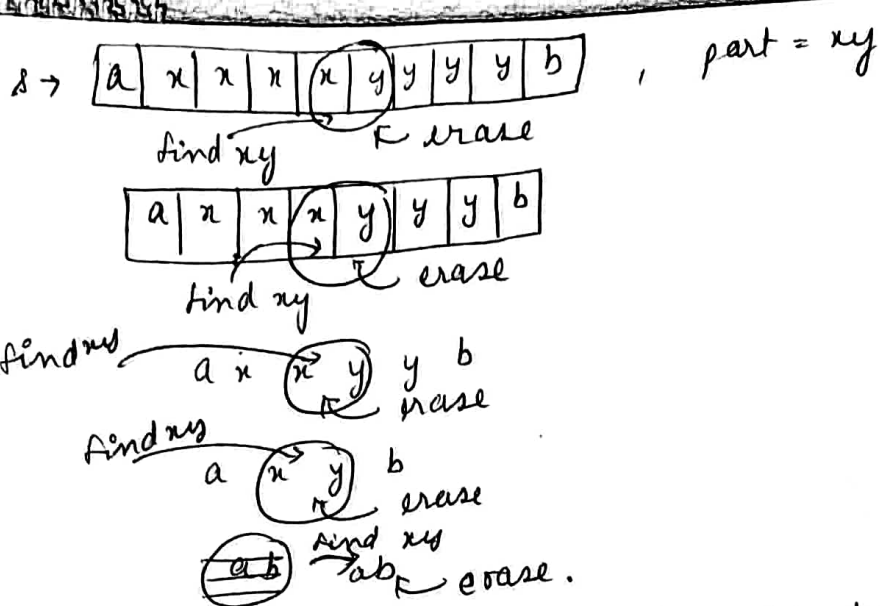
s.find(part)

returns

① or npos

① either a valid starting index

H.W → Implement s.find() and s.erase() functions.



Q3 → Valid Palindrome - II → return true if s can be palindrome string after deleting at most 1 character.

680  
leetcode  
⇒  $s \rightarrow$ 

a	b	c	a
---	---	---	---

if ( $s[i] == s[j]$ ) → no need to remove  
↳  $i++$ ,  $j--$ , check for next elements

if ( $s[i] != s[j]$ ) → Two options.

OR  
condition → 

a	b	c	a
---	---	---	---

 at  $s[i]$  or  $s[j]$  remove or not, or  $s[j]$  or remove or not,

bool ~~Valid~~ ValidPalindrome (string s) {

int i = 0;

int n = s.length();

int j = n-1;

while (i < j) {

if ( $s[i] != s[j]$ )

// remove or not element at i or j  
return checkPalindrome(s, i+1, j) // or j or not element remove or not,

checkPalindrome(s, i, j-1);

else { // no need to remove anything.

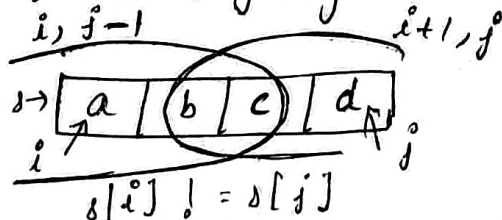
$i++$ ;

$j--$ ;

}

~~return true;~~

return true;



two options

① → 

a	b	c
---	---	---

 Palindrome

② → 

b	c	d
---	---	---

 → check Palindrome.

bool checkPalindrome (string s,

int i, int j) {

while (i < j) {

if ( $s[i] != s[j]$ ) {

return false;

$i++$ ;

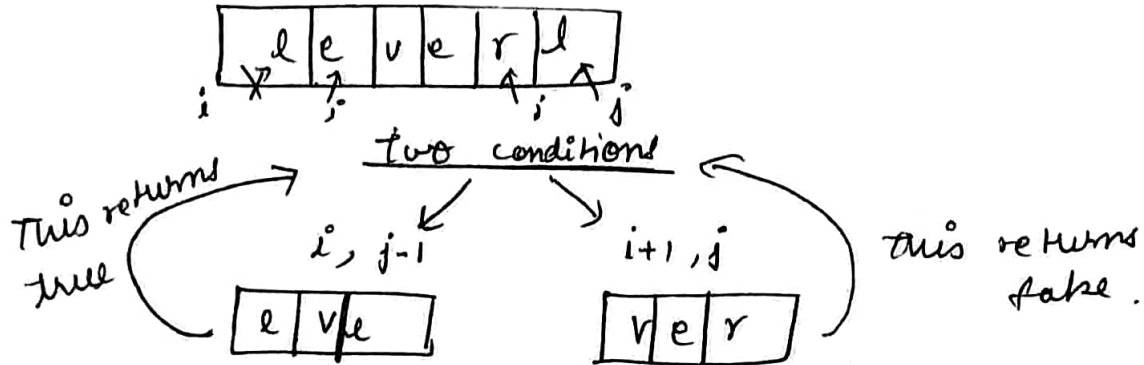
return  $j--$ ; false;

Two cases →

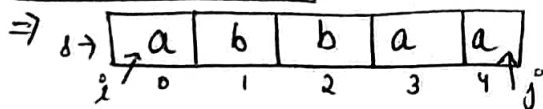
0-removal → level } True  
 1 removal → level  
                   ↓  
                 level

if more than 1 removals

eg, abcd, level } false.



Another method → true || false → true answer

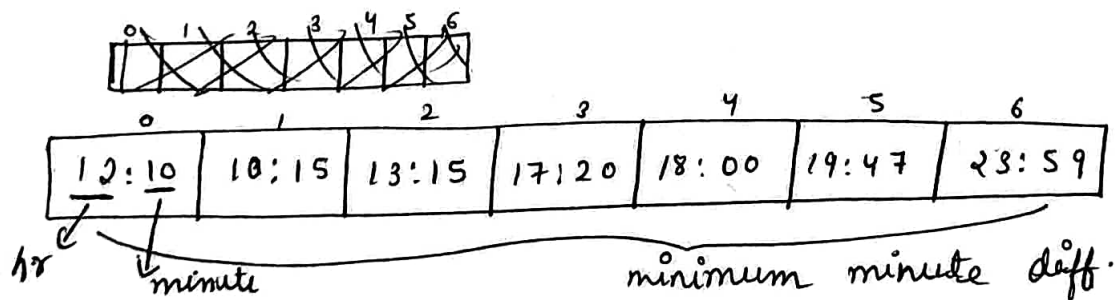


0 removal → abbaa → checkPalindrom → false

1 removal → bbaa, abaa, abaa, abba, abba  
                   ↓        ↓        ↓        ↓  
                 true    false    false    true

If one of any these is true, we will return true.

Q4 Minimum Time Difference  
539



12:10 → 13:10

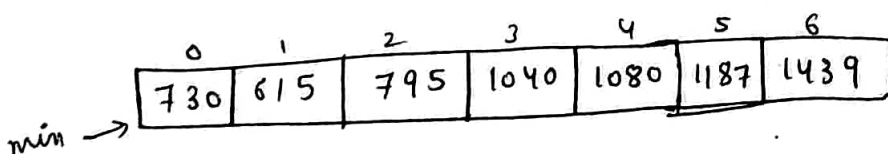
min. minute diff = 60 min

12:10 → 12:35

min. minute diff = 25 min

We will first convert these times into minutes.

To convert a string into integer → stoi() func is used.



~~int findMinDifference (vector<string>~~

sort min. array.

→ To reduce no. of comparison.

0	1	2	3	4	5	6
615	730	795	1040	1080	1187	1439

Now compare only with adjacent element and find the difference and return minimum difference.

```

int findMinDifference (vector<string> & timePoints) {
    vector<int> minutes;
    for (int i = 0; i < timePoints.size(); i++) {
        // Step 1 → convert time string into minute integer value.
        string curr = timePoints[i];
        int hours = stoi(curr.substr(0, 2));
        int minute = stoi(curr.substr(3, 2));
        int totalMinutes = hours * 60 + minute;
        minutes.push_back(totalMinutes);
    }

    // Step 2 → sort minute array.
    sort(minutes.begin(), minutes.end());

    // Step 3 → difference and calculate min difference
    int mini = INT_MAX;
    int n = minutes.size();
    for (int i = 0; i < n; i++) {
        int diff = minutes[i+1] - minutes[i];
        mini = min(mini, diff);
    }

    // return mini;

    // Special missing case →
    int lastDiff = (minutes[0] + 1440) - minutes[n-1];
    mini = min(mini, lastDiff);
}

```

One edge case will be failed here.

[11:59, 00:00]

→ it's output = 1

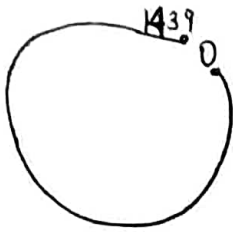
our output = 1439

minutes [1439 | 0]

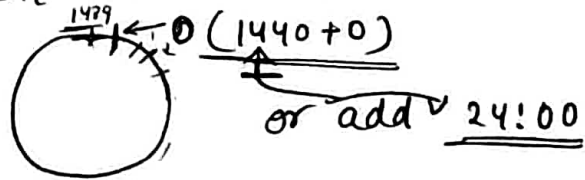
after sorting [0 | 1439]

diff = 1439

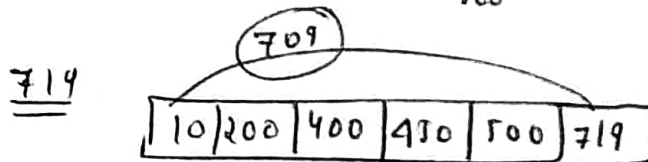
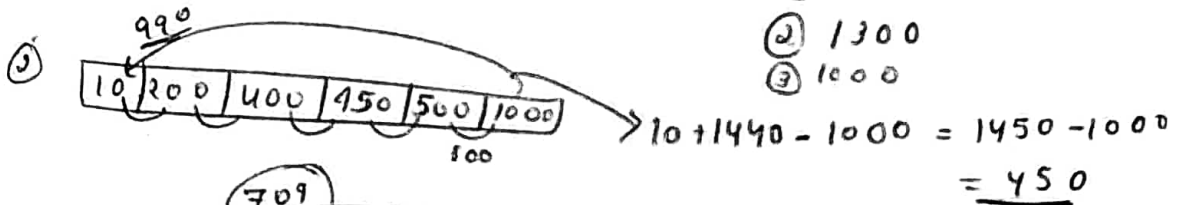
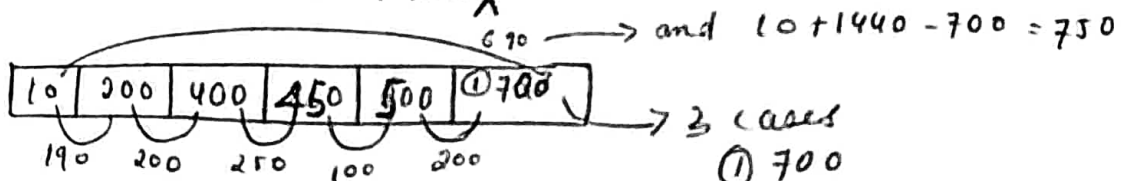
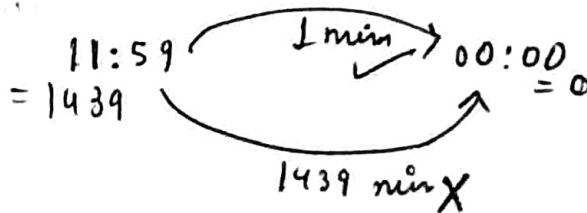
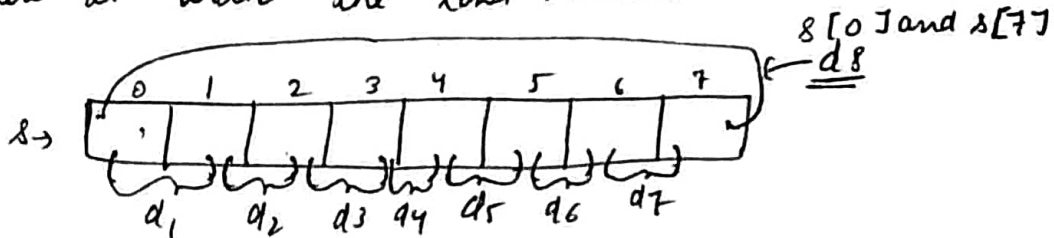
But time is circular.



In 24 hours  $\Rightarrow$  1440 minutes

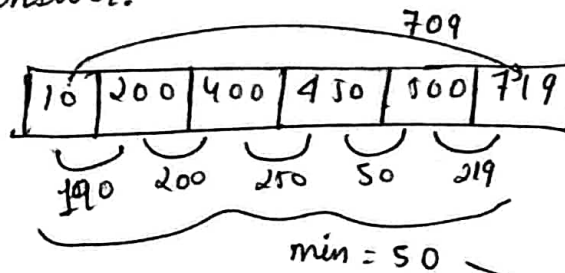


so we will add 1440 to the first element and also compare it with the last element.



$$1440 + 10 - 719 = 1450 - 719 = 731$$

If we are confused whether we have to add 1440 in first element or not, so we will do both and get our answer.



$$10 + 1440 - 719 = 731$$

$$\min(709, 731) = 709$$

$$\min(50, 709) = 50 \text{ ans.}$$

Now slight changes in our code  $\rightarrow$

// special case  $\rightarrow$

```
int lastDiff1 = (minutes[0] + 1440) - minutes[n-1];  
int lastDiff2 = minutes[n-1] - minutes[0];  
int lastDiff = min(lastDiff1, lastDiff2);  
mini = min(mini, lastDiff);  
return mini;
```

// we can replace this piece of code with the code we wrote previously at special case section.

Q  $\rightarrow$  Number of Palindromic Substring

647  
leetcode

i/p 

a	b	c
---	---	---

$\rightarrow$  continuous sequence of character within the string.

substrings

a, ab, abc, bc, b, c } 3 palindromic

i/p 

a	a	a
---	---	---

substrings  $\rightarrow$  (a), (aa), (aaa), a, aa, a } 3

i/p 

n	o	o	n
---	---	---	---

substrings  $\rightarrow$  (n), (o), (o), (n), no, oo, noon, noo, oon, on, n } 4 palindromic substrings.

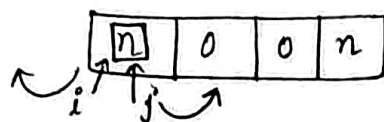
Approach 1  $\rightarrow$

① find all substrings. }  $\rightarrow O(n^2)$

② check palindrome and if it is palindrome }  $\rightarrow O(n)$   
increment counter by 1.

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

# # Approach 2 -



substring  $\rightarrow$  n, o

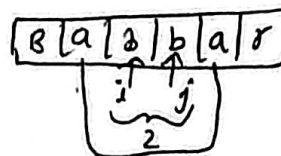


In odd length we start  $i$  and  $j$  from same index or in even length they both starts from different strings.

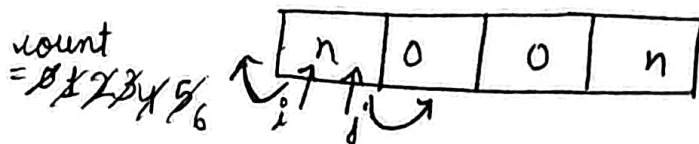
Substring  
 $\swarrow \searrow$   
 Odd length Even length



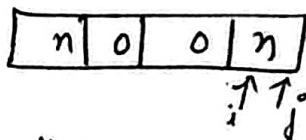
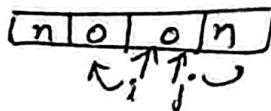
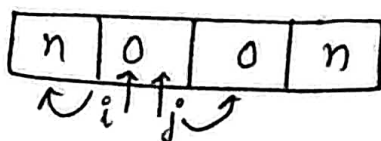
length  $\rightarrow$  1  
 3  
 5  
 7  
 9 } odd length



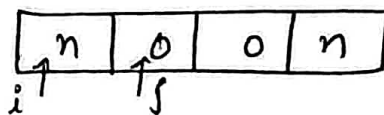
lengths = 2, 4, 6  
 even length.



- [n] ✓
- [o] ✓
- [noo] X
- [o] ✓
- [oon] X
- [n] ✓



now even length  $\rightarrow$



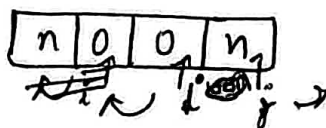
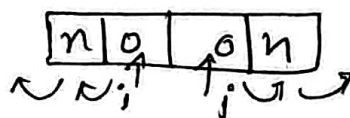
[no] X

[oo] ✓

[noon] ✓

~~[noo]~~  
~~[oon]~~

[on] X





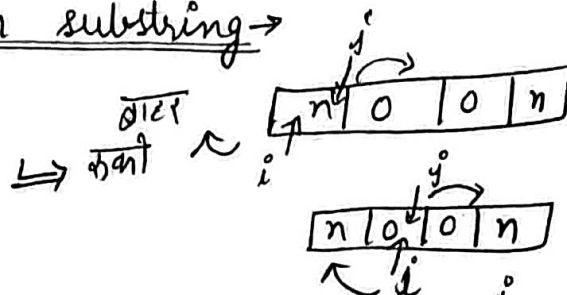
once again →

n o o n

count = 0 1 2 3 4 5 6

odd length substring →

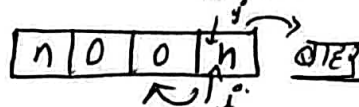
n  
o  
noo  
oon



n ✓  
o ✓  
noo ✗  
o ✓  
oon ✗  
n ✓

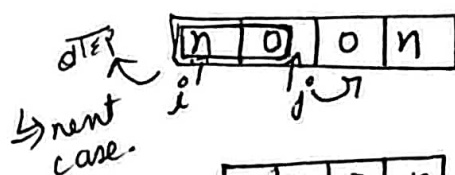
match → count increment  
i--, j++

no match → go to next case



even length substring →

no  
oo  
on  
noon



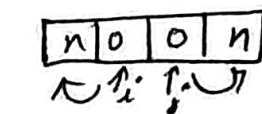
not matched  
→ next case

no ✗

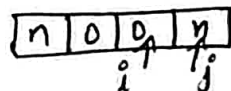
oo ✓

noon ✓

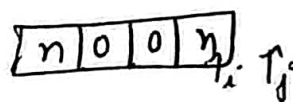
on ✗



matched



not matched



• code →

```
int countSubstring(string s) {
```

```
    int count = 0;
```

```
    int n = s.length();
```

```
    for (int i = 0; i < n; i++) {
```

```
        // odd
```

```
        int oddKasns = expandAroundIndex(s, i, i);
```

```
        count += oddKasns;
```

```
        // even
```

```
        int evenKasns = expandAroundIndex(s, i, i+1);
```

```
        count += evenKasns;
```

```
    }
```

```
    return count;
```

```

int expandAroundIndex(string s, int i, int j, int count) {
    int count = 0;
    while (i >= 0 && j < s.length() && s[i] == s[j]) {
        count++;
        i--;
        j++;
    }
    return count;
}

```

Tab tak match karega, tab tak count increment kardo and i piche or j sage kardo.

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