★ Substrings :

eg: "abc" (0 1 2)

op:

i, j

a → (0,0)
ab → (0,1)
abc → (0,2)
i = 0, j = 0 → 2, i → 2

b → (1,1)
bc → (1,2)
i = 1, j = 1 → 2 (i → 2)

c → (2,2)
i = 2, j = 2 → 2 (i → 2)

```
console.log( str.slice(i, j+1) );
```

str.slice(0,1) ⇒ "a" ✓ "ab" ✗

str.slice(0,2) ⇒ "ab" ✓ "abc" ✗

(0,0) slice(0,1) ⇒ "a"
(0,1) slice(0,2) ⇒ "ab"
(0,2) slice(0,3) ⇒ "abc"
(1,1) slice(1,2) ⇒ "b"

```
for ( let i = 0; i < n; i++) {
    let substr = "";
    for( let j = i; j < n; j++) {
        substr += str[j];
        console.log (substr);
    }
}
```

① i = 0
j = i = 0 ✗ ✗ ✗

② i = 1
j = i = ✗ ✗ ✗

③ i = 2
j = i = ✗ ✗

* if you are using slice,

```
for ( ) {
    for ( ) {
        slice ( )
    }
}
```

⇒ for every (i, j) pair you are using slice()

how many pairs = $\frac{n(n+1)}{2}$

Total time = pairs * (n) → time to slice

$= \frac{n(n+1)}{2} * n$

$= O(n^3)$

0,0 1,1 2,2
0,1 1,2
0,2

3 + 2 + 1

n = 3

* To optimise,

running string
$O(N^2)$

① i = 0, substr = ""

a) j = 0, substr += str[0]
= "a"

b) j = 1, substr += str[1]
= "ab"

c) j = 2, substr += str[2]
= "abc"

② i = 1, substr = ""

a) j = 1, substr += str[1]
= "b"

b) j = 2, substr += str[2]
= "bc"

① $i = 0$

a) $j = 0$ slice(0,1) → n

b) $j = 1$ slice(0,2) → n

c) $j = 2$ slice(0,3) → n

② $i = 1$

a) $j = 1$ slice(1,2) → n

b) $j = 2$ slice(1,3) → n

③ $i = 2$

a) $j = 2$ slice(2,3) → n

$3 + 2 + 1 = 6$

$n + n + n + n + n + n$

$\Rightarrow 6n$

↓

no. of pairs

$n = 4 \Rightarrow 4 + 3 + 2 + 1 = 10$

$TC = 10n$

$n = 5 \Rightarrow 5 + 4 + 3 + 2 + 1 = 15$

$TC = 15n$

$$TC = \frac{n(n+1)}{2} * n$$

$$= \frac{(n^2+n) * n}{2} = \frac{n^3+n^2}{2} = O(n^3)$$

★ Distinct palindromic substrings :

Eg:

0	1	2	3	4	5
a	b	c	c	b	c

1) a
2) a b
3) a b c
4) a b c c
5) a b c c b
6) a b c c b c
7) b
8) b c
9) b c c
10) b c c b
11) b c c b c
12) c
13) c c
14) c c b
15) c c b c
16) c
17) c b
18) c b c
19) b
20) b c
21) c

"a", "b", "bccb", "c", "cc", ~~"c"~~, "cbc", ~~"b"~~, ~~"c"~~ (palindromic)

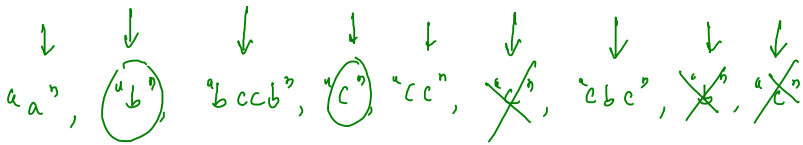
⇓

"a", "b", "bccb", "c", "cc", "cbc" (distinct)

⇓ lexicographical

"a",
"b",
"bccb",
"c",
"cbc",
"cc"

1. take the substrings
2. check whether each substring is a palindrome ?
3. distinct ? → use an object
4. lexicographical → dictionary order → sorted order.



key
is "a" present in map ?

map["a"] = 1

map["b"] = 1

map["c"] ++

1 ++

2

object

⇒ map : {

cnt of each palindromic substring

a : 1

b : ✗ 2

bccb : 1

c : ✗ ✗ 3

cc : 1

cbc : 1

}

* If is print all keys of the object,

a, b, bccb, c, cc, cbc

⇒ distinct

```
263  function palindromeSubStrs(str) {
264    const n = str.length;
265    const map = {};
266
267    for (let i = 0; i < n; i++) {
268      let substr = "";
269      for (let j = i; j < n; j++) {
270        substr += str[j];
271        if (isPalindrome(substr)) {
272          // add it to the object / map
273          if (map[substr] != undefined) {
274            map[substr]++;
275          } else {
276            map[substr] = 1;
277          }
278        }
279      }
280    }
281
282    // get all the keys from a map
283    const keys = Object.keys(map);
284
285    // lexicographical order => sorted order
286    keys.sort();
287
288    // print according output format
289    for (let i = 0; i < keys.length; i++) {
290      console.log(keys[i]);
291    }
292  }
```

O(N²) O(N) O(1) O(N²) O(NlogN) O(N²)

1. generate substr, generate (i,j) pair
   ⇒ $O(N^2)$
2. for every (i,j) pair
   ⇒ checking palindrome → O(N)
   ⇒ adding to map → O(1)

   $O(N^2 * N)$ $O(N^3)$
3. getting keys → $O(N^2)$ [every pair can be palindrome]
4. Sort → $O(N^2 \log N^2)$ [sorting all keys ⇒ all keys can have all substrings]
5. printing all keys → $O(N^2)$

TC: $O(N^3 + N^2 + N^2 \log N^2 + N^2)$
⇒ $O(N^3)$ → $O(N^2)$ [Dynamic programming]

SC: $O(N^2)$ → (object/map space if can have N^2 keys)

* ptice :

0 1 2

① Ronak : "A B C"

② Anurag : "B A B C"

③ Samarjeet : "C C A A B B"

Questions : 9 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

Answer Key : "A A A A B B B B B"

0 1 2 3 4 5 6 7 8

	Ronak	Anurag	Samerjeet
1)	A ✓	B ✗	C ✗
2)	B ✗	A ✓	C ✗
3)	C ✗	B ✗	A ✓
4)	A ✓	C ✗	A ✓
5)	B ✓	B ✓	B ✓
6)	C ✗	A ✗	B ✓
7)	A ✗	B ✓	C ✗
8)	B ✓	C ✗	C ✗
9)	C ✗	B ✓	A ✗
	4 marks	4 marks	4 marks

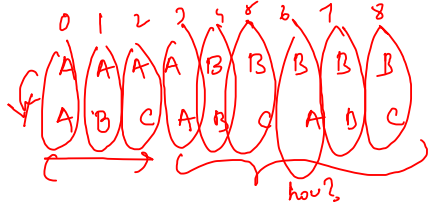
op: print the person who got highest marks,

⇒ Ronak
Anurag
samarjeet

Rut : ~~0~~ 1 ~~2~~ ~~3~~ 4

Aut : ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

Sut : ~~0~~ 1 ~~2~~ ~~3~~ 4



rotate Array → ⟳

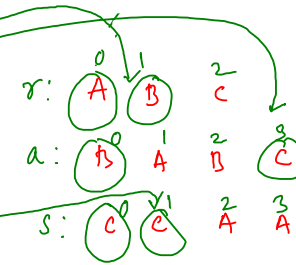
★ How to know the Answer of every student for every question?

r[i%3] ⇒ r[7%3] ⇒ r[1] ⇒ B

a[i%4] ⇒ a[7%4] ⇒ a[3] ⇒ C

s[i%6] ⇒ a[7%6] ⇒ s[1] ⇒ C

length



n%3 ⇒ (0, 1, 2)

n%4 ⇒ (0, 1, 2, 3)

n%6 ⇒ (0, 1, 2, 3, 4, 5)

★ Autari' :

Eg: Players – Unknown – Battle – Grounds

op: PUBG

#1 : Split("-")

0 1 2 3

⇒ ["players", "Unknown", "Battle", "Grouds"]

⇒ "PUBG"

TC : split + iterate ⇒ O(N+N) ⇒ O(N)

SC : Split will give you array ⇒ O(N)

#2 :

1. whenever you find '-'
   add S[i+1] to ans = S[0]

0 1 2 3 4 5 6 7 8

A B C – D E F – G H I

3

"A" ⇒ "AD"

⇒ "ADG"

TC : O(N)

SC : O(1)

A b c – D e f

"D"

"A"

★ String to Integer Array :

eg: "123, 456, 789" → string

↓

[123, 456, 789]

↓

Number

#1 :

split(',')

⇒ ['123', '456', '789'] ⇒ arr/op

0 1 2

⇒ arr[i] = Number(arr[i])

parseInt(arr[i])

TC: O(N)

SC: O(N) / O(1)

#2 :

"123, 456, 789"

arr = [123, 456, 789]

rs = "123" "456" "789"

when "," encountered,

arr.push(Number(rs))

rs = ""

rs = "123" → "123,"

// unexplored rs

arr.pus(Number(rs))

TC: O(N)

SC: O(N) / O(1)