Majority Element

Problem Description

Given an array of size **N**, find the majority element. The majority element is the element that appears more than floor(n/2) times.
You may assume that the array is non-empty and the majority element always exists in the array.

Problem Constraints

1 <= N <= $5*10^5$
1 <= num[i] <= 10^9

Input Format

Only argument is an integer array.

Output Format

Return an integer.

	0	1	2	3	4	5	6	7
A =	3	4	3	2	4	4	4	4
freq	2	5	2	1	5	5	5	5

ans = 4

```java
public class Solution {
    // DO NOT MODIFY THE ARGUMENTS WITH "final" PREFIX. IT IS READ
    public int majorityElement(final int[] A) {
        int n = A.length;
        int major = -1; // Current majority element
        int freq = 0; // freq of the major


        for(int i = 0; i < n; i++){
            int x = A[i];

            if(freq == 0){
                major = x;
                freq = 1;
            }
            else{
                if(major == x){
                    freq++;
                }
                else{
                    freq--;
                }
            }
        }

        // rechecking of majority would be needed
        // if it's not given that there can be majority element

        return major;
    }
}
```

Anti Diagonals

Problem Description

Give a **N * N** square matrix **A**, return an array of its anti-diagonals. Look at the example for more details.

Problem Constraints

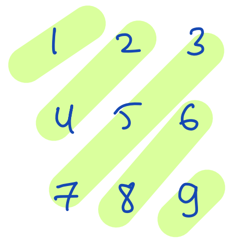
1<= **N** <= 1000
1<= **A[i][j]** <= 1e9

Input Format

Only argument is a 2D array **A** of size **N * N**.

Output Format

Return a 2D integer array of size (2 * **N**-1) * **N**, representing the anti-diagonals of input array **A**.
The vacant spaces in the grid should be assigned to 0.



Output

$$\begin{matrix} 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 5 & 7 \\ 6 & 8 & 0 \\ 9 & 0 & 0 \end{matrix}$$

How many diagonals ?

$N+M-1$
$N+N-1$
$\underline{\underline{2*N-1}}$

```java
public class Solution {
    public int[][] diagonal(int[][] mat) {
        int N = mat.length;

        // Final result
        int[][] result = new int[2*N - 1][N];

        for(int j = 0; j < N; j++){
            int r = 0;
            int c = j;

            ArrayList<Integer> row = new ArrayList<>(); // []

            while(r < N && c >= 0){
                row.add(mat[r][c]);
                r++;
                c--;
            }

            for(int z = 0; z < row.size(); z++){
                result[j][z] = row.get(z);
            }

        }

        for(int i = 1; i < N; i++){
            int r = i;
            int c = N-1;

            ArrayList<Integer> row = new ArrayList<>(); // []

            while(r < N && c >= 0){
                row.add(mat[r][c]);
                r++;
                c--;
            }

            for(int z = 0; z < row.size(); z++){
                result[N+i-1][z] = row.get(z);
            }
        }

        return result;
    }
}
```

Length of longest consecutive ones

Given a binary string **A**. It is allowed to do at most one swap between any 0 and 1. Find and return the length of the longest consecutive 1's that can be achieved.

Input Format

```
The only argument given is string A.
```

Output Format

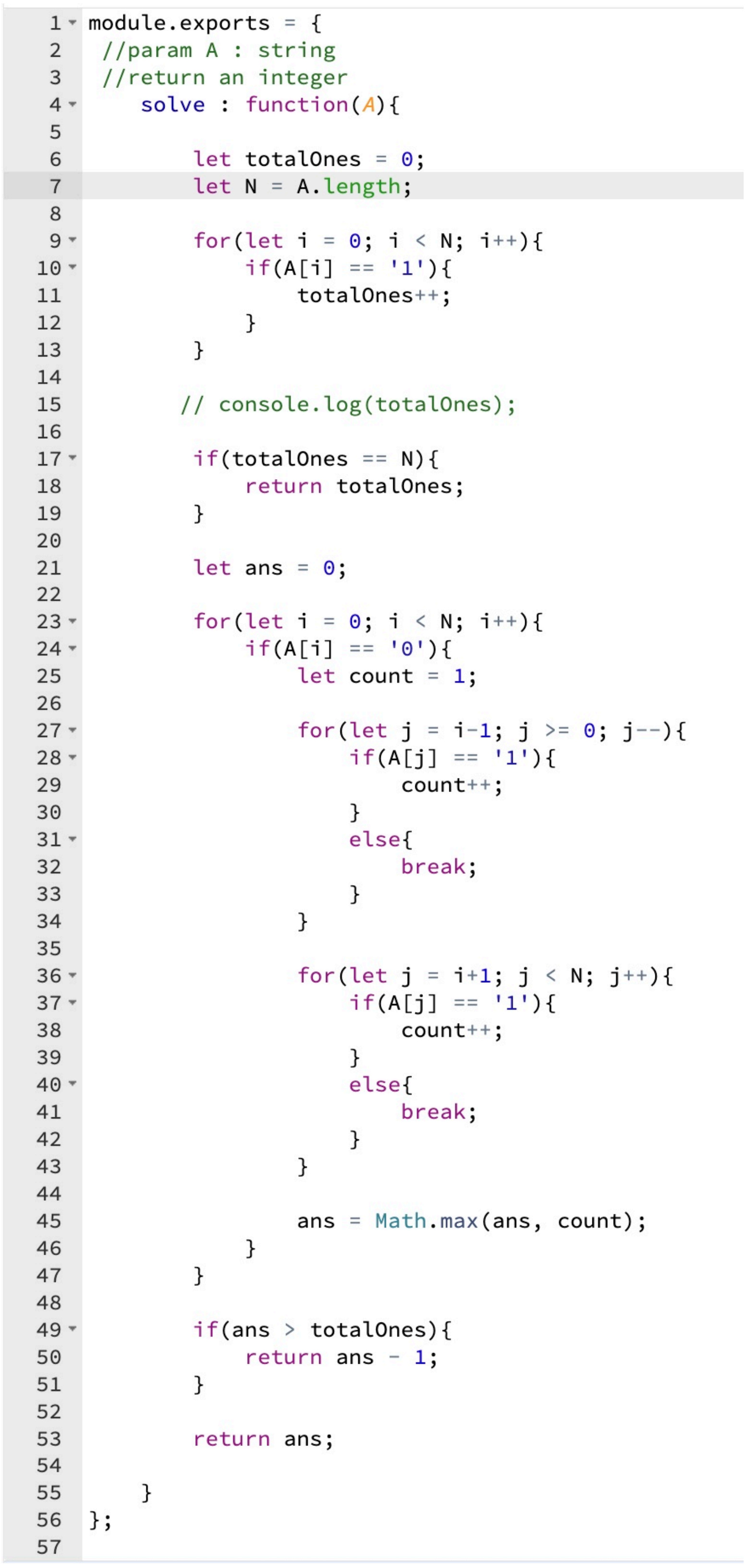
```
Return the length of the longest consecutive 1's that can be achieved.
```

$A = [1 \; \underset{1}{\cancel{0}} \; 1 \; 1 \quad 0 \; \underset{0}{\cancel{1}}]$ $\qquad ans = 4$

$A = [1 \; 1 \; \underset{1}{\cancel{0}} \; 1 \; 1 \; \underset{0}{\cancel{1}}]$ $\qquad ans = 5$

$A = [1 \; 1 \; 1 \; 1 \; 1]$ $\qquad ans = 5$

$A = [0]$ $\qquad ans = 0$



```
module.exports = {
 //param A : string
 //return an integer
    solve : function(A){

        let totalOnes = 0;
        let N = A.length;

        for(let i = 0; i < N; i++){
            if(A[i] == '1'){
                totalOnes++;
            }
        }

       // console.log(totalOnes);

        if(totalOnes == N){
            return totalOnes;
        }

        let ans = 0;

        for(let i = 0; i < N; i++){
            if(A[i] == '0'){
                let count = 1;

                for(let j = i-1; j >= 0; j--){
                    if(A[j] == '1'){
                        count++;
                    }
                    else{
                        break;
                    }
                }

                for(let j = i+1; j < N; j++){
                    if(A[j] == '1'){
                        count++;
                    }
                    else{
                        break;
                    }
                }

                ans = Math.max(ans, count);
            }
        }

        if(ans > totalOnes){
            return ans - 1;
        }

        return ans;

    }
};

```

Longest Palindromic Substring

Problem Description

Given a string A of size N, find and return the **longest palindromic substring** in A.

Substring of string A is `A[i...j]` where 0 <= i <= j < len(A)

Palindrome string:
A string which reads the same backwards. More formally, A is palindrome if reverse(A) = A.

Incase of conflict, return the substring which occurs first (with the least starting index).

Problem Constraints

1 <= N <= 6000

Input Format

First and only argument is a string A.

Output Format

Return a string denoting the longest palindromic substring of string A.

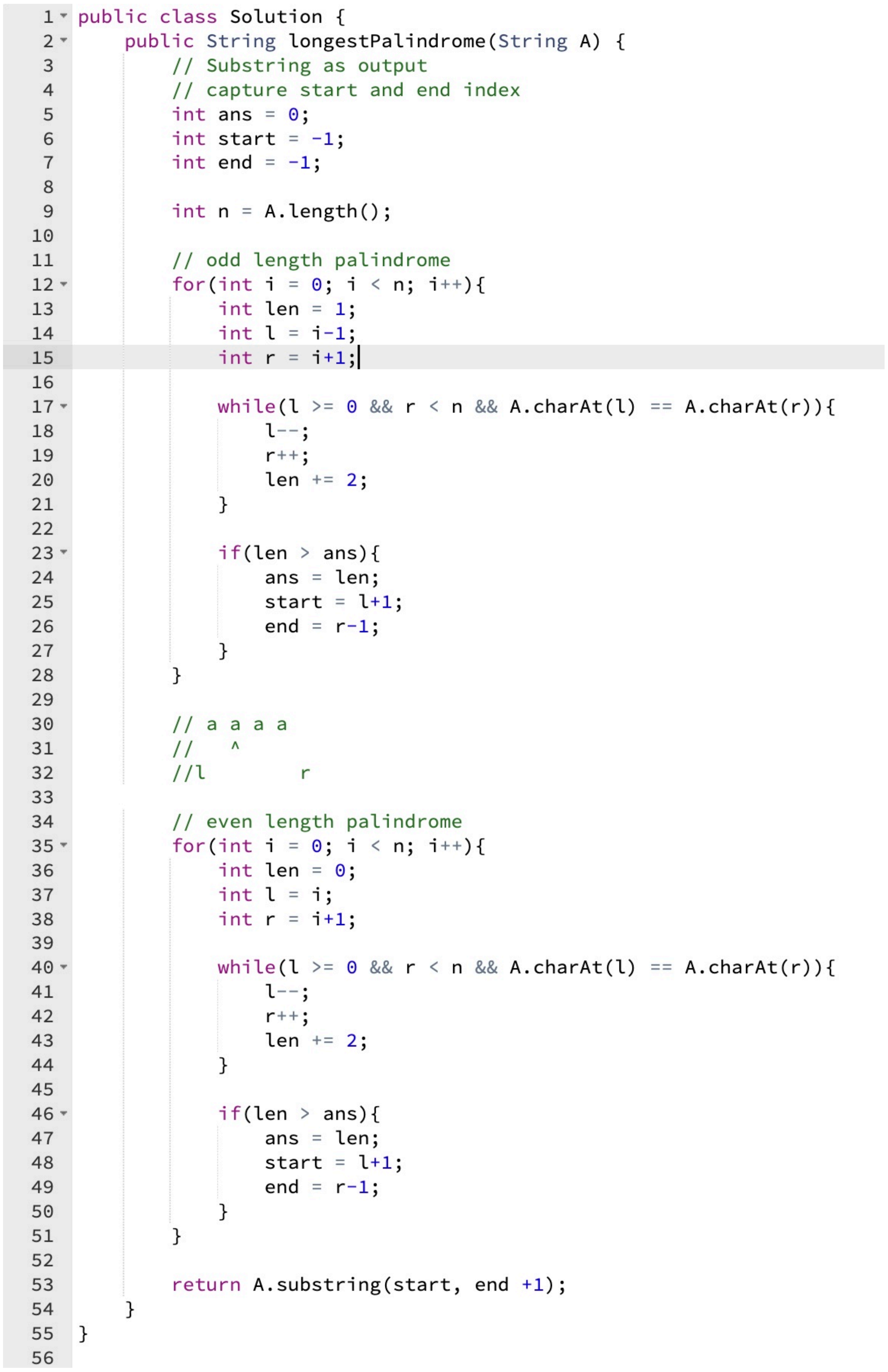
a bcbcb x b   Output bcbcb

a n a madam   Output madam

fe acabaca bgf   Output acabaca

a d a e b c d f d c b e t g g t e   output e b c d f d c b e

aa   output = aa



```java
public class Solution {
    public String longestPalindrome(String A) {
        // Substring as output
        // capture start and end index
        int ans = 0;
        int start = -1;
        int end = -1;

        int n = A.length();

        // odd length palindrome
        for(int i = 0; i < n; i++){
            int len = 1;
            int l = i-1;
            int r = i+1;

            while(l >= 0 && r < n && A.charAt(l) == A.charAt(r)){
                l--;
                r++;
                len += 2;
            }

            if(len > ans){
                ans = len;
                start = l+1;
                end = r-1;
            }
        }

        // a a a a
        //   ^
        //l        r

        // even length palindrome
        for(int i = 0; i < n; i++){
            int len = 0;
            int l = i;
            int r = i+1;

            while(l >= 0 && r < n && A.charAt(l) == A.charAt(r)){
                l--;
                r++;
                len += 2;
            }

            if(len > ans){
                ans = len;
                start = l+1;
                end = r-1;
            }
        }

        return A.substring(start, end +1);
    }
}
```

Given an array of strings
Longest prefix of all the strings

A[] = ["abcdefgh"
"aefghijk"
"abcefgh"
]

output = "a"

A[] = ["abab",
"ab",
"abcd"
]

output = "ab"

why sorting wont help?

A = [
"abc"
"bac"
"cba"
]

$\longrightarrow$

output = ""

abc
abc
abc

~~abc~~

```
A =     abcd          char = d
        abcdef
        abcikl        ans = abc
        abcfjk
```

① Take shortest length out of all strings.
② Traverse column wise

Pseudocode

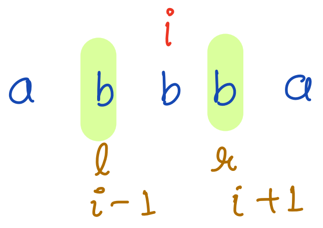
N = no. of strings
L = length of min length string

$O(N*L)$

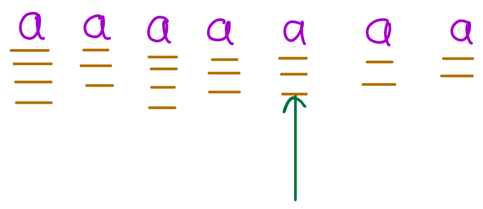
```
ans = ""
minlength = // min length of all strings.

minlength * N

for c=0 ; c< minlength ; c++ {   → minlen
    char ch = A[0].charAt(c)
    flag =true
    for (i → 0 to N-1) {   → N times.
        if (ch != A[i].charAt(c)) {
            flag = false
            break
        }
    if(!flag) break
    ans += ch
}
```

Palindromic Substring code

```
ans = 0

for i ⟶ 0 to N-1 {
        len = 1
        l = i-1
        r = i+1
        while ( l>=0 && r<N) {
                if ( s[l] != s[r]) break
                l -= 1
                r += 1
                len += 2
        }

        ans = max (ans, len)
}
print(ans)
```



TC : O (N²)
SC : O (1)



```
ans = 0                                          TC : O(N)
                                                 SC : O(1)
for i ⟶ 0 to N-1 {
    if (A[i] == 0) {
        count = 1      // count of ones.
        // #ones on left
        for j ⟶ i-1 to 0 {
            if (A[j] == 1) count ++
            else break
        }

        // #ones on right
        for j ⟶ i+1 to N-1 {
            if (A[j] == 1) count ++
            else break
        }
        ans = max(ans, count)
    }
}
```

[0 1 1 1 0 1 1 0 1 1 0]

$69.53\% \longrightarrow 75\%$

Personal goal $\longrightarrow$ 100%

Attendance $> 95\%$

PSP $> 95\%$

$\rightarrow$ get personal referral from me.