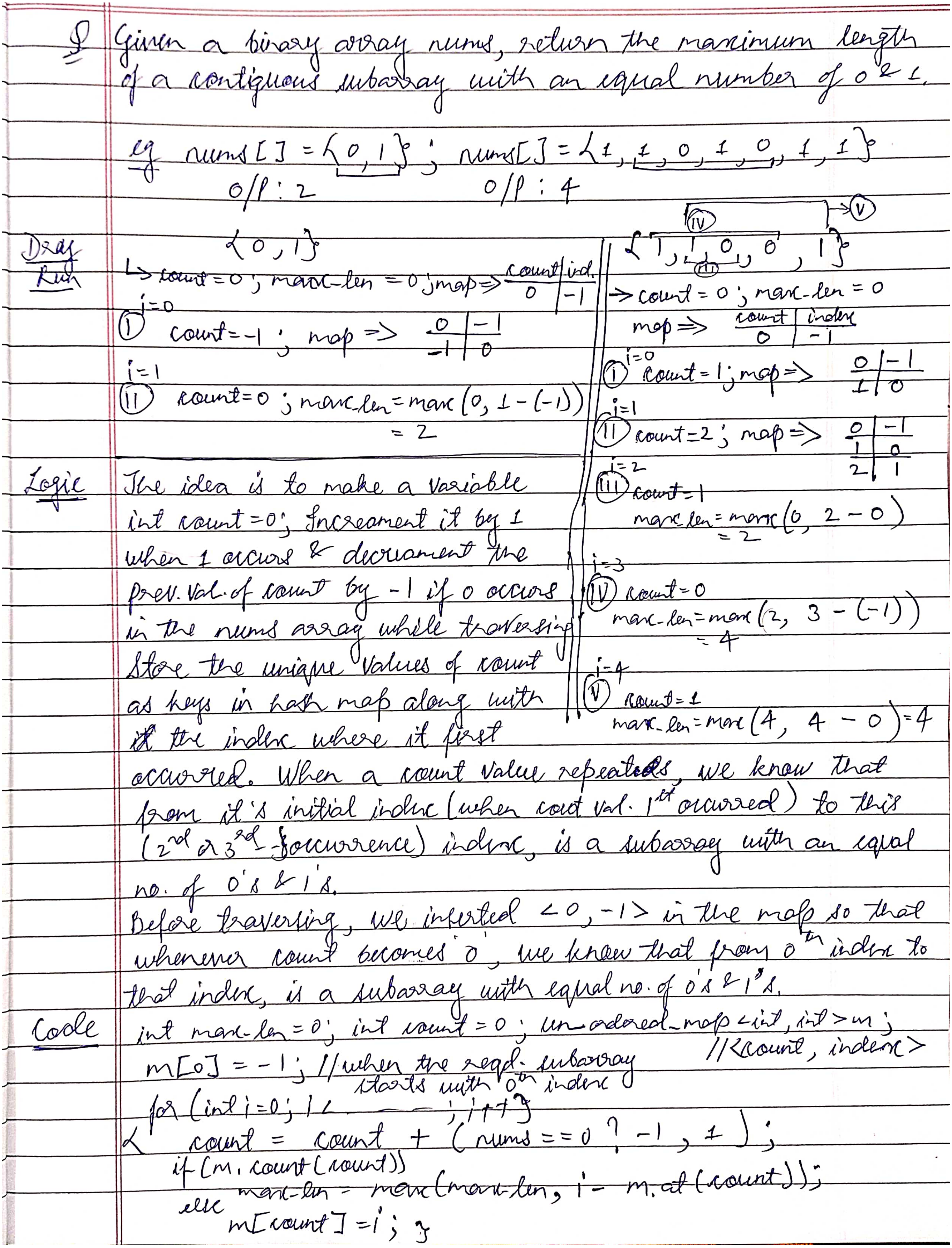
Optimized:

Time complexity: O(n); Space complexity: O(n)…because of the hash map



Q Given a binary array nums, return the maximum length of a contiguous subarray with an equal number of 0 & 1.

eg nums[] = {0, 1} ; nums[] = {1, 1, 0, 1, 0, 1, 1}

O/P : 2 O/P : 4

Dry Run

{0, 1}

↳ count = 0 ; max-len = 0 ; map ⇒

count	ind.
0	-1

i=0
(I) count = -1 ; map ⇒

0	-1
-1	0

i=1
(II) count=0 ; max-len = max(0, 1 - (-1))
= 2

{1, 1, 0, 0, 1} (III) (IV) → (V)

→ count = 0 ; max-len = 0

map ⇒

count	index
0	-1

i=0
(I) count = 1 ; map ⇒

0	-1
1	0

i=1
(II) count=2 ; map ⇒

0	-1
1	0
2	1

i=2
(III) ~~count~~=1
max-len = max(0, 2 - 0)
= 2

i=3
(IV) count=0
max-len = max(2, 3 - (-1))
= 4

i=4
(V) count=1
max-len = max(4, 4 - 0) = 4

Logic

The idea is to make a variable int count=0; Increament it by 1 when 1 occurs & decreament the prev. val. of count by -1 if 0 occurs in the nums array while traversing. Store the unique values of count as keys in hash map along with ~~it~~ the index where it first occurred. When a count value repeat~~ed~~s, we know that from it's initial index (when cout val. 1st occurred) to this (2nd or 3rd ~~f~~occurrence) index, is a subarray with an equal no. of 0's & 1's.

Before traversing, we inserted <0, -1> in the map so that whenever count becomes 0, we know that from 0th index to that index, is a subarray with equal no. of 0's & 1's.

Code

```
int max-len = 0; int count = 0; un-ordered_map<int, int> m;
                                              //<count, index>
m[0] = -1; //when the reqd. subarray
               starts with 0th index
for (int i=0; i< ____ ; i++)
{   count = count + (nums == 0 ? -1 , 1);
    if (m.count(count))
        max-len = max(max-len, i- m.at(count));
    else
        m[count] =i; }
```