Unique Array :-

i/p :- [0, 1, 1, 2, 2, 3, 4, 4]

o/p : [0, 1, 2, 3, 4, 3, 4, 4]

return the same array with 1st half having unique elements.

i j

0 1 1 2 2 3 4 4

```
while(j < n){
    if(nums[i] != nums[j])
        { nums[i+1] = nums[j]
          i++ }
    j++;
```

Number of Subsequences given condition on target

i/p : [2 3 10] target : 9

Ans = 3 (2, 3, (2,3))

1 2 3 10

1) Sort arr

2) l = 0 r = n-1

3) if (num[l] + num[r] <= target)

result += power(2, r-l).

l++;

else r--;

l r

2 3 4 5 6

no. of subsequence = 2^{r-l}

the empty sequence l to r sequence.

Number of Window Having Max & min fixed

I/p = 2 1 3 5 7 5 1 3

max = 5

min = 1

o/p = 4

2 1 3 5

1 3 5

5 1 3

5 1

```
int maxPos = -1
int minPos = -1
int culprit = -1
```

#⭐, #Trick

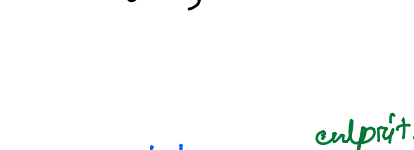
```
for (i:0 → n ){
    if (nums[i] > max || < min) culprit = i
    if (nums[i] == max) maxpos = i
    if (nums[i] == min) minpos = i
    int start = min(maxpos, minpos);
     possible = start - culprit
    ans += possible<=0 ? 0 : temp;
```

Dry Run

1, 5

		0	1	2	3	4	5	6	7
nums		2	1	3	5	7	5	1	3
maxpos	-1				3		5		
minpos	-1		1					6	
culprit	-1					4			

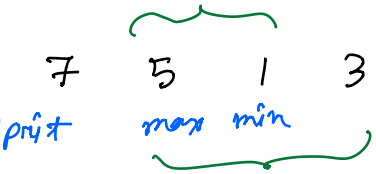
start	-1	-1	-1	1	1	1	5	5
possible	0	0	0	2	-3	-3	1	1
ans	0	0	0	2	2	2	3	4



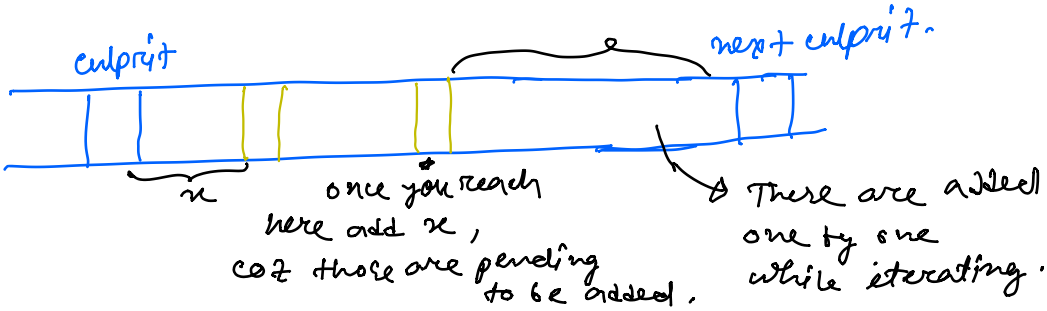
possible range including 5 and 1 and culprit is outside.

here we found a valid range of max & min and culprit is outside so there is possible subarray of interest to be specific no. of subarray is 2 ⇒ starting from idx 0 and starting from idx 1.

Similarly



when min(maxpos, minpos) > culprit we can add possible subarrays.



Maximum Confusion :-

I/p : T T F F T F T | T T F F F F T

ans = 9

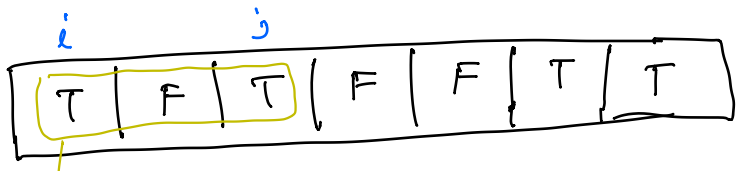
max streak after at most K no. of flips.

Approach 1 :- ① find what's the maximum streak for 'T' possible.

② find maximum 'F' streak possible

ans = max(①, ②).

Approach 2 :- (in 2 pass)



K = 2.

T = 2 } To make FFF 2T need to flip.
F = 1 } To make TTT 1F need to flip.

always take min (True, False).

T = 3 } now i has to move
F = 3 } making T = 2, F = 3 } where we can achieve 5 F (max answer)

Decreasing Deque :-

#LeetCode/Hard, #Trick

I/p : [1, 3, -1, 5, 3, 20, 6, 7, 3]

window size = 3

O/p : [3, 5, 5, 20, 20, 20, 7]

↓ Greatest element in first window of size 3.

Data structure deque<int> dq;

```
while (i < nums.size) {
    while (dq && nums[i] > q.back())
        {dq.pop_back();}
    dq.push_back(nums[i]);
    if (j - i + 1 >= k){
        ans.push_back(dq.front());
        if (nums[i] == q.front())
            q.pop_front();
        i++ }
    j++ }
```

dq

j	dq
0	[1]
1	[3]
2	[3 -1]
3	[5]
4	[5 3]
5	[20]
6	[20 6]
7	[20 7]
8	[7]