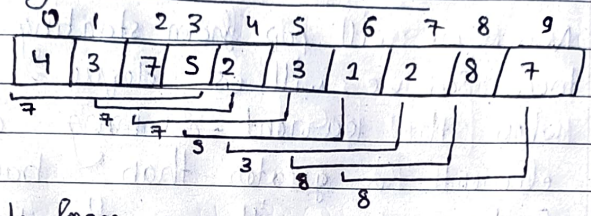
Date _____ Page _____

Day - 148

Queue - 4

* **Sliding Window Maximum:** $k = 4$



Brute force

Code

```
vector<int> ans;
for(i=0; i<=n-k; i++){
    int total = INT_MIN;
    for(j=i; j<i+k; j++){
        total = max(total, nums[i]);
    }
    ans.push_back(total);
}
return ans;
```

⇒ T.C. → $O(nk)$

Optimized approach

⇒ To optimize the approach, we have to find a way so that we will get max. element in that window in $O(1)$ time.

⇒ So, we can make a approach just like that the max. element is present in the front of queue.

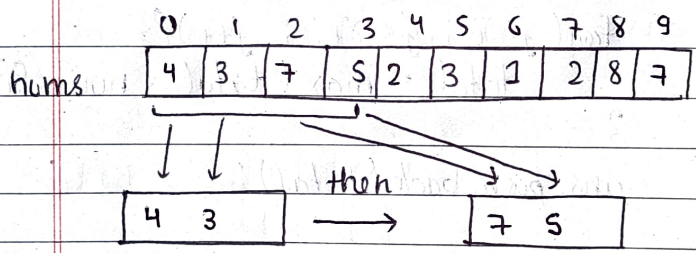
by

Date _____ Page _____

⇉ So, we will do that, we will use a queue and only store max. element in the queue as first max. second max. etc.

⇉ Now, we will pop from starting & ending too. So, we will use deque.

⇉ when the current ~~comming~~ coming element is greater than back element of deque, we will pop the back elements until we will get a element greater than curren element.



⇉ We will store index instead of values.

Code

```
deque<int> d;
vector<int> ans;
for(i=0; i<k-1; i++){
    if(d.empty())
        d.push_back(i);
```

Date. Page

```
    else{    !d.empty()&&
        while(nums[i] > nums[d.back()])
            d.pop_back();
        d.push_back(i);
    }
}

for(i= k-1; i<n; i++){
    while(!d.empty() && nums[i] > nums
                        [d.back()])
        d.pop_back();
    d.push_back(i);
    if (d.front <= i-k)
        d.pop_front();
    ans.push_back(nums[d.front()]);
}
return ans;
```

* Min no. of k consecutive Flip bits:

nums | 0 | 1 | 0 | 1 |

k = 2

⇒ we have to make this array 1 but we have to do this by using windows of size k.

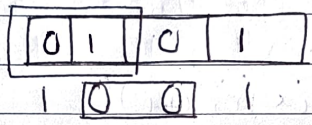
⇒ And there is another condition that for every window, we will make 0 to 1 & 1 to 0.

Date _____ Page _____

⇉ we have to return no. of flips required to make this array of 1s.

⇉ Also, flipping bits of window is consider as 1 flip.

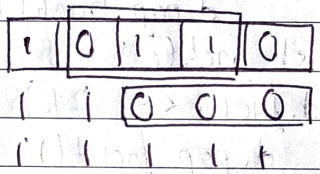
⇉ k=2



→ result

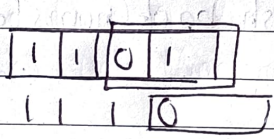
flip = ~~0~~ ~~1~~ 2 → ans

⇉ k=3



flip = ~~0~~ ~~1~~ 2
flip = 2 → ans

k=2



→ X

⇉ So, in this condition, we will return -1.

⇉ Brute force

⇉ This approach is very simple, we will find 0 then take window of k size & flip the bits.

⇉ After that again, we will do this.

Date _____ Page _____

Code

```
int flip = 0;
for(i=0; i<n; i++){
    if(nums[i] == 0){
        for(j=i; j<=i+k-1; j++){
            nums[j] = !nums[j];
        }
        flip++;
    }
}
return flip;
```

if(i+k-1 >= n) return -1;

T.C. → O(nk)

S.C. → O(1)

Optimized approach

⇒ We will use an array that will store the flips for every element.

⇒ So, by using this, we don't have to traverse the window & flipping the bits.

⇒ So,

1 → no. of flips → odd → 0

1 → no. of flips → even → 1

0 → no. of flips → odd → 1

0 → no. of flips → even → 0

Date _____ Page _____

⇒ This approach will also take O(nk) time.

⇒ So, we have to find a way that we can know how much time a elements flip. (last index of window)

⇒ For this, we will store the index, where the window flips the elements.

	0	1	2	3	4	5	6	7	8	9
k=4	0	0	1	0	0	1	1	0	1	0

⇒ → index

for 0. 3 → This showing that upto 3 all the no. are flips (size) no. of times.

for 1. Here, we have 0 & it flipped one time. So, no flipping required.

for 2. Here, we have 1 & it flipped one time, so, we have to flipped it.

⇒ Therefore, 3,5 → it showing that from 2 to 5, all the elements flipped two times.

⇒ So, in this way, we will solve our question.

Date _____ Page _____

Code

```
queue <int> q;
int flip = 0;
for( i=0; i<n; i++){
        if(!q.empty() && q.front()<i)
                q.pop();
        if( ~~num~~ q.size()%2 == nums[i]){
                if(i+k-1 >= n)
                        return -1;
                q.push(i+k-1);
                flip++;
        }
}
return flip;
```