# Lecture ÷ Revision 1

## Agenda

- ✓ is prime
- ✓ Prefix sum
  - Basic technique
  - Sum of even no in range
- Subarray
  - Definition
  - Longest odd subarray
  - ✓ Longest inc subarray
- Sliding window
  - Idea
  - Min swaps
- Carry forward
  - Pick from both sides

Qu Check whether a number is prime or not

who have 2 factors [1, n]

Eg: 7 → 1 and 7

19 → 1 and 19

23 → 1 and 23

37 → 1 and 37

Approach

```
boolean isprime(int n) {

    int cntfactors = countfactors(n);

    if (cntfactors == 2) {

        return true;
    }
    return false;
}
```

TC: $O(\sqrt{n})$

SC: $O(1)$

n = 100 [ $i = 1$ to $i <= \sqrt{n}$ ] TC: sqrt(n)

2

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

100%3≠0

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  | 100%1==0 | 100%2==0 | 100%4==0 |  | 100%10==0 |  |
|  | 1 factor | 2 factor | 4 factor | $i$ | 10 factor | equal |
|  | $\frac{100}{1}$ factor | $\frac{100}{2}$ factor | $\frac{100}{4}$ factor | $\frac{n}{i}$ | $\frac{100}{10}$ factor |  |
| cnt | 2 | 2 | 2 |  | 1 |  |

## Que2 Prefix sum

```
arr[] = [ 2    3     5      1       4    7 ]
pf[]  = [ 2    5     10     11      15   22 ]
               ↑     ↑      ↑
              2+3  2+3+5  2+3+5+1
```

```
pf[0] = 2 = arr[0]

pf[1] = arr[0] + arr[1]
        pf[0]  + arr[1]

pf[2] = arr[0] + arr[1] + arr[2]
        pf[1]           + arr[2]

pf[3] = arr[0] + arr[1] + arr[2] + arr[3]
        pf[2]                    + arr[3]
  ⋮
pf[i] = pf[i-1] + arr[i]
```

```
int[] prefixsum( int[] arr) {
    int[] pf = new int[ arr.length];
    pf[0] = arr[0];

    for (i = 1; i < arr.length; i++) {
        pf[i] = pf[i-1] + arr[i];
    }

    return pf;
}
```

TC: O(n)

SC: O(n) || O(1)

↑ when we don't consider of space

**Qu 3:** count even no. in a range

$$arr[\,] = \left[\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 3 & 1 & 4 & 6 & 7 & 8 & 10 & 5 \end{array}\right]$$

Given q queries, for every query, tell the even no. count.

| l | r | ans |
|---|---|---|
| 0 | 5 | 3 |
| 2 | 7 | 4 |
| 3 | 4 | 2 |

$$arr[\,] = \left[\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 3 & 1 & 4 & 6 & 7 & 8 & 10 & 5 \end{array}\right]$$

$$pfEven[\,] = \left[\begin{array}{ccccccccc} 1 & 1 & \underset{1+0}{\underline{1}} & 2 & 3 & 3 & 4 & 5 & 5 \end{array}\right]$$

$$pfEven[0] = arr[0]\ \%\ 2 == 0 \rightarrow 1$$
$$\rightarrow 0$$

$$pfEven[1] = pfEven[0] + arr[1]\ \%\ 2 == 0 \rightarrow 1$$
$$\text{else} \rightarrow 0$$

$$pfEven[2] = countEven[0, 2]$$
$$= countEven[0, 1] + \text{check for 2nd idx}$$
$$= pfEven[1] + arr[2]\ \%\ 2 == 0 \rightarrow 1$$
$$\text{else} \rightarrow 0$$

$$\vdots \qquad \vdots$$

$$pfEven[i] = pfEven[i-1] + arr[i]\ \%\ 2 == 0 \longrightarrow 1$$
$$\text{else} \longrightarrow 0$$

Qu    Longest increasing subarray

arr[] = [ 5  6  3  5  7  8  9  1  2 ]    Ans = 5

arr[] = [ 12  13  1  5  4  7  8  10  10  11 ]

            0  1  2  3  5  5  6  7  8
arr[] = [ 5  6  3  5  7  8  9  1  2 ]
               ↑
               i

max = 1 , len=1.

i = 1    arr[1] > arr[0] → len++ , max = Math. max(max,len).

arr[1] <= arr[0] → reset your length (len==1)

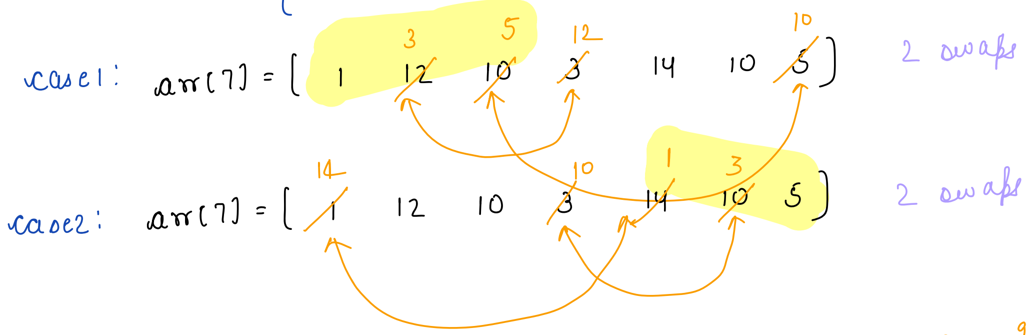
```
int LIS( int [] arr) {
    int max=1, len=1;
    for( i=1; i< arr.length; i++) {
        if( arr[i] > arr[i-1]) {
            len +=1;
            max=Math.max(max.len);
        } else {
            len=1;
        }
    }
    return max;
}
```

TC: O(n)
SC: O(1)

**Min swaps. :-** Given arr[n] and integer B. find and return the min no of swaps required to bring all no less than B together.
or equal to

Ex:  arr[7] = [ 1 12 10 3 14 10 5 ]

B = 6   [ el less than 6 = 1, 3, 5 ]



case1: arr[7] = [ 1 ~~12~~ 3 ~~10~~ 5 ~~3~~ 12 14 10 ~~5~~ 10 ]
case2: arr[7] = [ ~~1~~ 14 12 10 ~~3~~ 10 ~~14~~ 1 ~~10~~ 3 5 ]

2 swaps

2 swaps

0 1 2 3 4 5 6 7 8 9
arr[10] = [ 1 12 6 3 8 13 15 13 4 5 ]

B = 7.   1 6 3 4 5
1 6 3 4 5
1 6 3 4 5
1 6 3 4 5
1 6 3 4 5
1 6 3 4 5

window = 5.

arr[10] = [

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| arr[10] = [ | 1 | 12 | 6 | 3 | 8 | 13 | 15 | 13 | 4 | 5 ] |

B = 7.

| s | e | count <= B | ans |
|---|---|---|---|
| 0 | 4 | 3 | 5 - 3 = 2 swaps |
| 1 | 5 | 2 | 5 - 2 = 3 swaps |
| 2 | 6 | 2 | 5 - 2 = 3 swaps |
| 3 | 7 | 1 | 5 - 1 = 4 swaps |
| 4 | 8 | 1 | 5 - 1 = 4 swaps |
| 5 | 9 | 2 | 5 - 2 = 3 swaps |

[0th - removed
5th - added] every same

↓
count - 1
↑
0th idx
is <= B

+ 1
↑
5th idx <= B

```
int minswaps( int[] arr, B){

    // Calculate window length
     int window=0;
    for(i=0; i< arr.length; i++){

        if ( arr[i] <= B) {

            window +=1;
        }
    }

    // Handle first window separately

     int count=0;

     for ( i=0; i< window ; i++){

         if ( arr[i] <=B) {

             count++;
         }
     }
int min = window - count;
int s = 1;
int e = window;
while( e < n) {
    // s-1 idx is removed
    if ( arr[s-1] <= B) {
        count --;
    }
  // eth idx is added
  if ( arr[e] <= B) {
        count ++;
  }
int swaps = window - count;

ans = min ( ans . swaps);

s++;
e++:
}
return ans;
}
```

TC: O(n)

SC: O(1)

Qu Pick from both sides

- Max possible sum
- B operations, remove leftmost & rightmost el

A = [ 2 3 −1 4 2 1 ]

B = 4

Case1:

| | | | |
|---|---|---|---|
| 1st: 2 | ans = 8 | 1st: 2 | ans = 9 ✓ |
| 2nd: 3 | | 2nd: 1 | |
| 3rd: 1 | | 3rd: 2 | |
| 4th: 2 | | 4th: 4 | |

B = 4.

| left | right | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | | |
| 0 | 4 | A = [ 2 | 3 | −1 | 4 | 2 | 1 ] | | sum(2,5) |
| 1 | 3 | A = [ 2 | 3 | −1 | 4 | 2 | 1 ] | sum(0,0) | + sum(3,5) |
| 2 | 2 | A = [ 2 | 3 | −1 | 4 | 2 | 1 ] | sum(0,1) | + sum(4,5) |
| 3 | 1 | A = [ 2 | 3 | −1 | 4 | 2 | 1 ] | sum(0,2) | + sum(5,5) |
| 4 | 0 | A = [ 2 | 3 | −1 | 4 | 2 | 1 ] | sum(0,3) | |

prefix

suffix sum

s e

0 − 4 [ sum(2,5) ] sfx[2]

sfx[n − e]

1 3 sum(0,0) sum(3,5) = sfx[3]

pf[0] sfx(n − 3)

Thankyou ☺