SORTING I

Contents

- Basics of sorting { stable / unstable
  Inplace sorting }
- K^{th} smallest element
- Selection sort
- Merge two sorted arrays.
- Merge sort
- Inversion count.

Sorting : Arranging the data in a defined order using some parameter

1	2	3	5	7	⟶	Asc	⟶ para → value
9	6	3	2	1	⟶	Dsc	—11—
1	7	2	9	24	⟶	yes	⟶ para → no. of factor.
↓	↓	↓	↓	↓			
1	2	2	3	8			

Why Sorting ?

- It makes data easier to find
- —11— organised.

Stable / Unstable Sorting

If two data points have same parameter value.

Stable sort ⟶ Order is preserved.

Unstable sort ⟶ Order is not preserved.

A[] = 1 5 2 6 2 {GR}

= 1 2 2 5 6 1 2 2 5 6

{RG} {GR}

unstable stable

Sort the data in the increasing order of their marks.

Name	Marks
mohit	0
aman	100
shreesh	45
kajal	32
ayush	45
aditya	92

Name	Marks
mohit	0
kajal	32
shreesh	45
ayush	45
aditya	92
aman	100

↓

Stable

Name	Marks
mohit	0
kajal	32
ayush	45
shreesh	45
aditya	92
aman	100

↓

unstable.

Inplace Sorting Algo

Sorting in the same array without extra space

O(1)

→ No additional space → inplace

K^{th} Smallest

Given A[N] elements, find K^{th} smallest element.

K = 3

A[8] =	2	8	4	-4	6	7	5	10	-1
	0	1	2	3	4	5	6	7	8

Idea 1 : Sort the given input and return K-1

TC : O(n log n)

SC : O(1)

Idea 2 :

	2	8	4	-4	6	7	5	10	-1
	0	1	2	3	4	5	6	7	8
1	-4	8	4	2	6	7	5	10	-1
	0	1	2	3	4	5	6	7	8
2	-4	-1	4	2	6	7	5	10	8
	0	1	2	3	4	5	6	7	8
3	-4	-1	2	4	6	7	5	10	8
	0	1	2	3	4	5	6	7	8

Pseudo code

```
int  kth smallest ( A[] , k ) {

    for (i = 0 ; i < K ; i++ ) {
        midx = i
        for ( j = i ; j < n ; j++ ) {
            if ( A[j] < A[midx] ) {
                midx = j
            }
        }
        swap i and midx
    }
```

TC : O(nk)

SC : O(1)

Selection Sort

```
int selection ( A[] ) {
    for (i = 0 ; i < n-1 ; i++ ) {
        midx = i
        for ( j = i ; j < n ; j++ ) {
            if ( A[j] < A[midx] ) {
                midx = j
            }
        }
        swap i and midx
    }
```

Selection sort does sorting in min no. of swaps
= n-1

TC : $O(N^2)$
SC : $O(1)$

2 2 1 {GP}
⟶
1 2 2 {PG}

Eg

⟶ 2 1
⟶ 1 2

⟶ 3 1 2
⟶ 1 3 2
⟶ 1 2 3

Given A[N], we can only **swap adjacent** elements, sort the array in increasing order

A[8] =	2	8	4	-4	6	7	5	10	-1
	0	1	2	3	4	5	6	7	8
=	2	4	-4	6	7	5	8	-1	10
	0	1	2	3	4	5	6	7	8

HW. Read more on **bubble sort.**

Given two sorted array. A[N], B[M]. Create C[] which contains overall sorted data.

A[4] : { 7 10 11 14 }
B[3] : { 3 8 9 }

A[3] : { 3 6 10 }
B[2] : { 5 9 }.

Idea : Merge the array A and B into C. then apply sorting

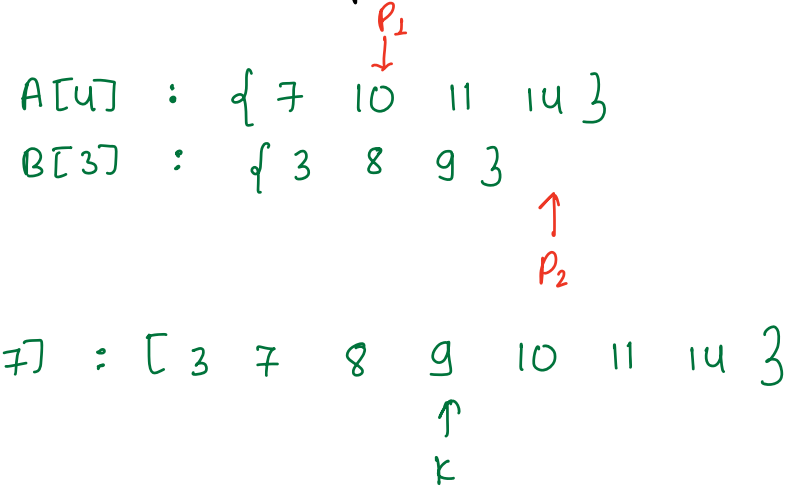
A[4] : { 7 10 11 14 }
B[3] : { 3 8 9 }

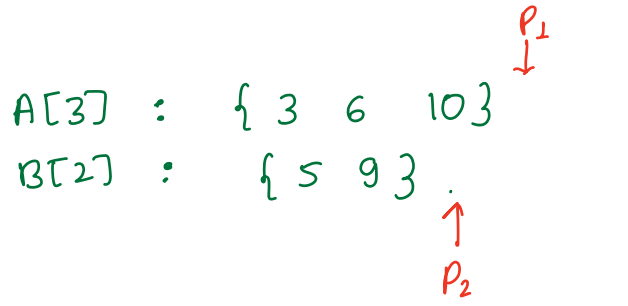
C[7] = 7 10 11 14 3 8 9

sort

C[7] = 3 7 8 9 10 11 14

Idea 2 : Take two pointers.





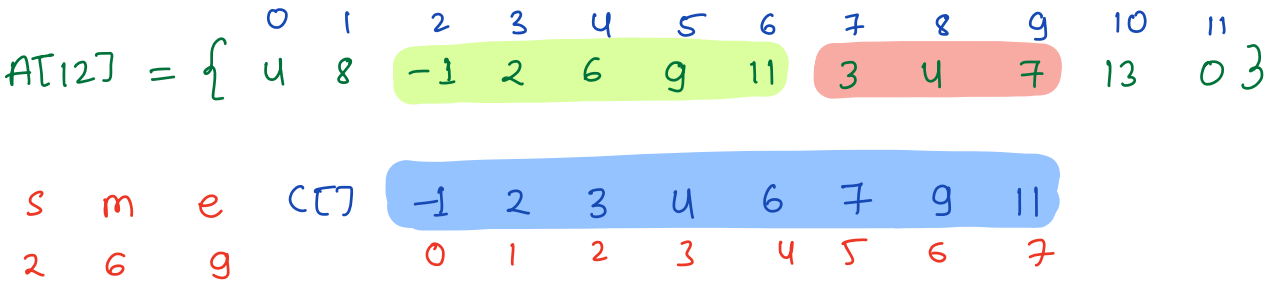

```
int[] merge ( A[N] , B[M] ) {

        C[M+N]    // init
        p1 = 0
        p2 = 0
        k  = 0

        while ( p1 < N && p2 < M ) {

            if ( A[p1] <= B[p2] ) {
                C[k] = A[p1]
                k++
                p1++
            }
            else {
                C[k] = B[p2]
                k++
                p2++
            }

        }
```

```
while (p1 < N) {
    C[k] = A[p1]
    k++
    p1++
}
while (p2 < M) {
    C[k] = B[p2]
    k++
    p2++
}
return C
}
```

TC : O(N+M)

SC : O(M+N)

Given A[N] elements & 3 indices s, m, e &
subarray [s m] is sorted
subarray [m+1 e] is sorted
Sort subarray from [s e]




```
int[] merge ( A[N] , s , m , e ) {

        C[e-s+1]  // init
        p1 = s
        p2 = m+1
        k  = 0

        while ( p1 <= m && p2 <= e ) {

                if ( A[p1] <= A[p2] ) {
                        C[k] = A[p1]
                        k++
                        p1++
                }
                else {
                        C[k] = A[p2]
                        k++
```

```
            p2 ++
        }
    }

    while (p1 ≤ m) {
        C[k] = A[p1]
        k++
        p1 ++
    }

    while (p2 ≤ e) {
        C[k] = A[p2]
        k++
        p2 ++
    }

}

for (i = 0 ; i < C.length ; i++) {
    A[i+s] = C[i]
}
```

TC: O(e-s)

SC: O(e-s)

Break : 8:44

Given A[N], sort it without inbuilt function.

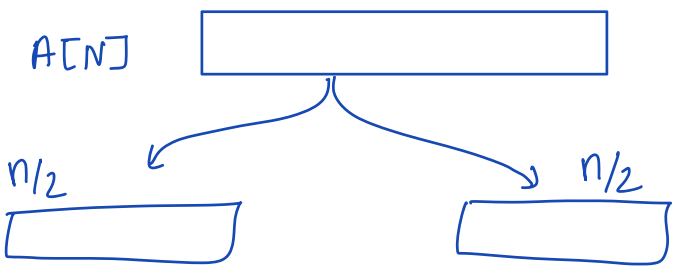
N = 100

Case I

Tc: $O(N^2)$

iterations: 10^4

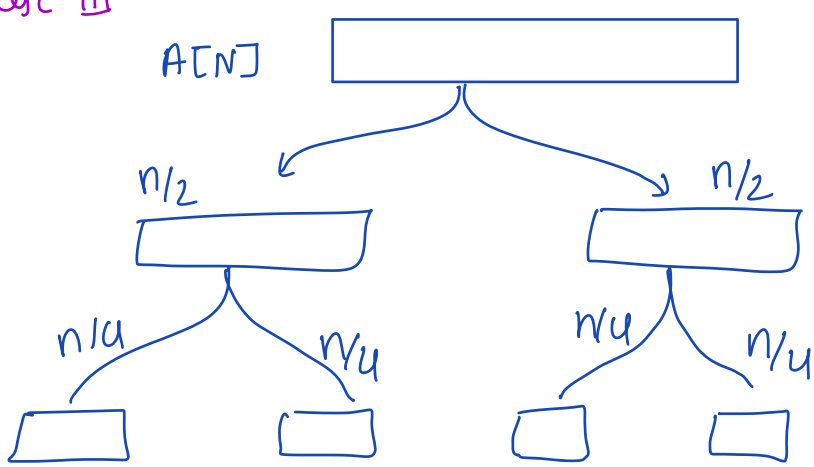
Case II



$\left(\frac{n^2}{2}\right) \times 2 + n$ $\left(\frac{100}{2}\right)^2 \times 2 + 100$

$= 5100$

Case III



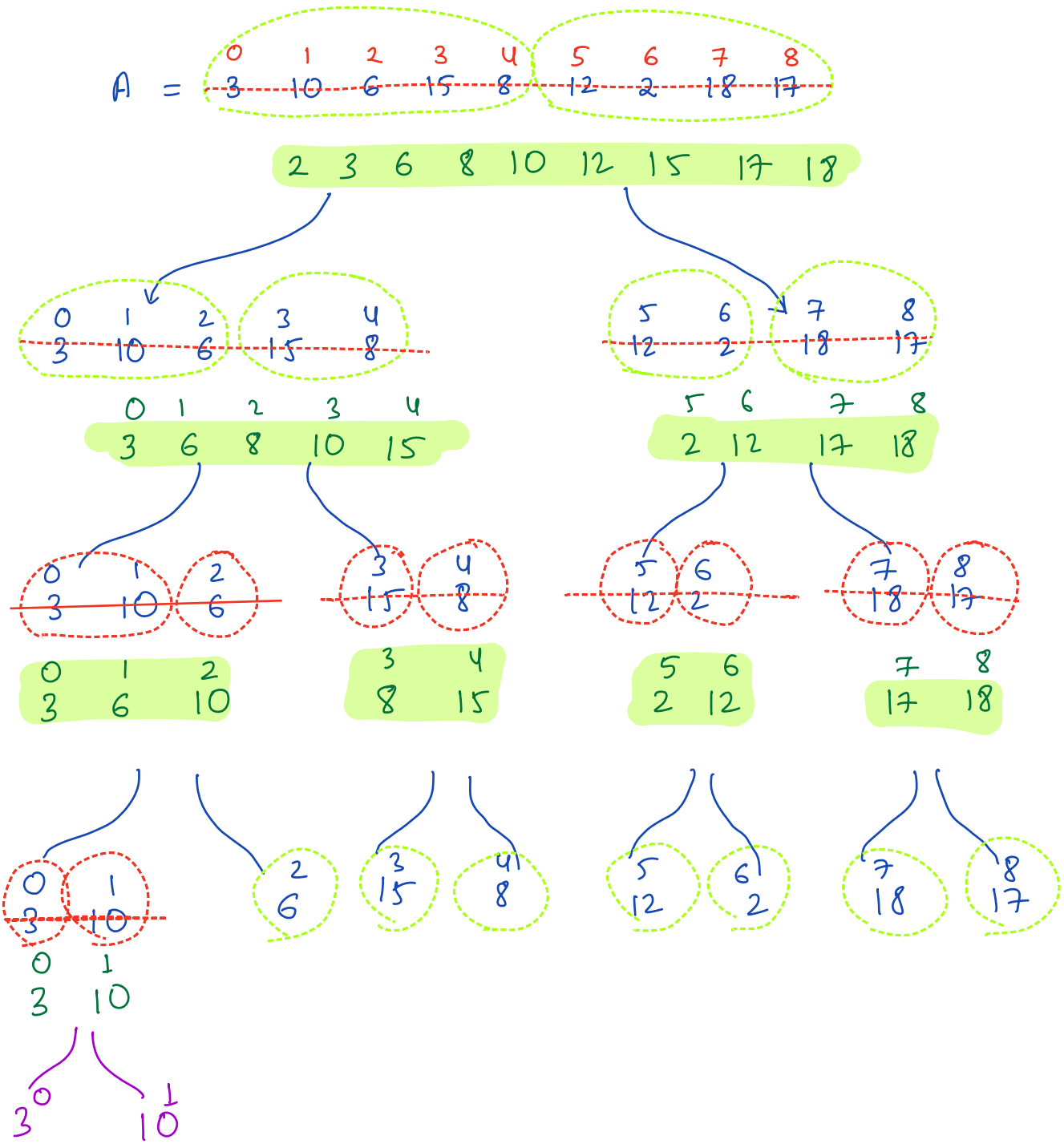
$\left(\frac{n}{4}\right)^2 \times 4 + 2 \times n$

$=$ iterations

$= 2700$

Idea for merge sort

Keep on dividing the array till you have individual elements, then you merge.



Pseudo code

```
void mergeSort ( A[] , s , e ) {
    if(s==e)  return

    m = (s+e)/2
    mergeSort ( A , s , m )
    mergeSort ( A , m+1 , e )
    merge ( A , s , m , e )
}
```

$$T(n) = 2 * T(n/2) + n$$ (HW)

TC : O(nlogn)
SC : O(N)

Inversion Count

Given an integer array, count the number of inversion pairs in the array.

Inversion pair → (i, j) where $(i<j \;\&\&\; A[i] > A[j])$

A = 4 5 1 2 6 3
0 1 2 3 4 5
↓ ↓ ↓
3 3 1

= (7).

A = 8 3 4
0 1 2 → 2

(8 3)
(8 4)

A = 10 3 8 15 6
0 1 2 3 4
↓ ↓ ↓ ↓ ↓ → 5
3 0 1 6 0

Bruteforce : (i, j) where $(i<j \;\&\&\; A[i] > A[j])$

Two loops i, j and count the times above condition is true.

6 2 4 5 2 3 1 2 1 = (26)

A = 10 3 8 15 6 12 2 18 7 1
0 1 2 3 4 5 6 7 8 9

10 3 8 15 6
0 1 2 3 4

P_1

3 6 8 10 15
0 1 2 3 4

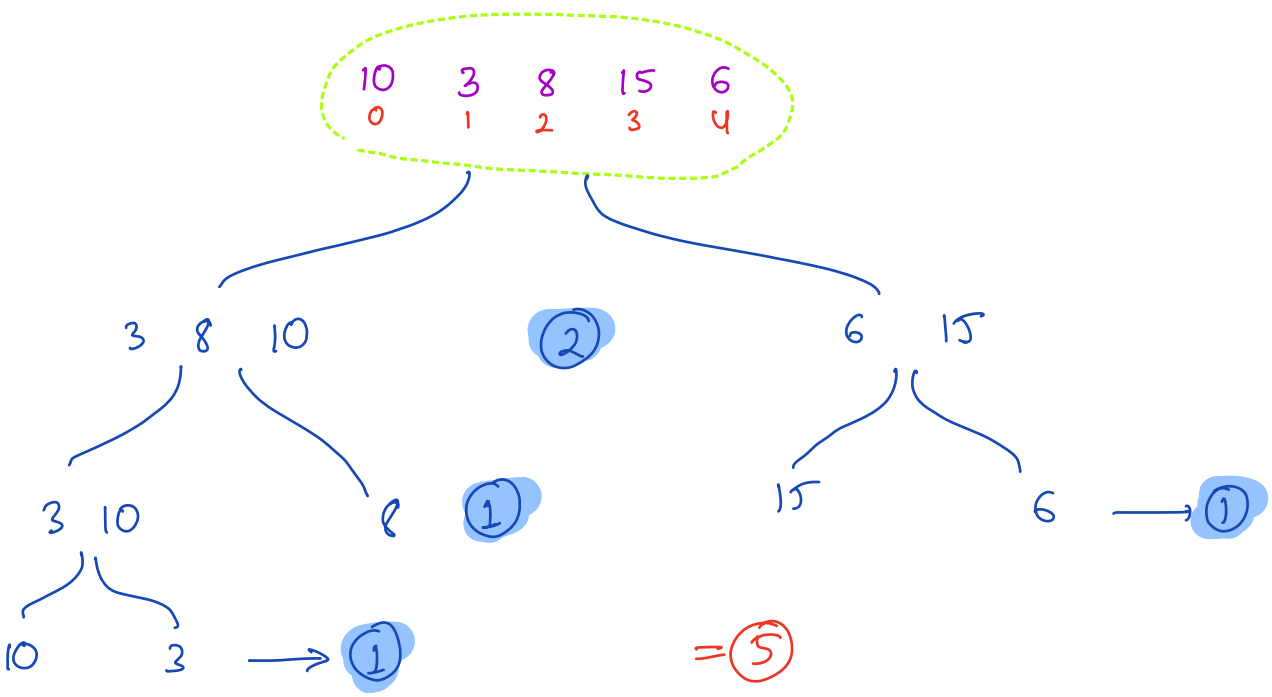
12 2 18 7 1
5 6 7 8 9

P_2

1 2 7 12 18
5 6 7 8 9

5 6 7 8
5 3 1 = 14

5 {(3,1) (6,1) (8,1)
(10,1) (15,1)



count = 0

```
int[] merge ( A[N] , s , m , e ) {

        C[e-s+1]  //  init
        p1 = s
        p2 = m+1
        k  = 0

        while ( p1 ≤ m && p2 ≤ e ) {

            if ( A[p1] <= A[p2] ) {
                C[k] = A[p1]
                k++
                p1++
            }
            else {
                C[k] = A[p2]
                k++
                p2++
            }  count += m - p1 + 1

        }

        while (p1 ≤ m) {
```

s m m+1 e

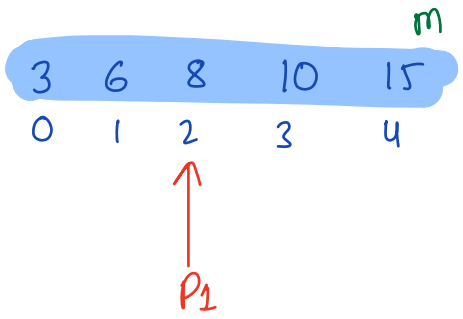
p1

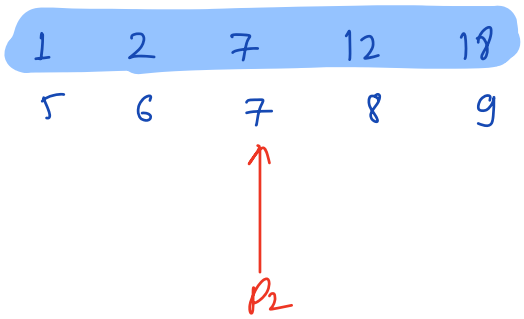
```
        C[k] = A[p1]
        k++
        p1++
    }
    while ( p2 ≤ e ) {
        C[k] = A[p2]
        k++
        p2++
    }
}
for ( i = 0 ; i < C.length ; i++ ) {
    A[i+s] = C[i]
}
void mergeSort ( A[], s , e ) {
    if(s==e) return

    m = (s+e)/2
    mergeSort ( A , s , m )
    mergeSort ( A , m+1 , e )
    merge ( A , s , m , e )
}
```

TC : $O(n \log N)$

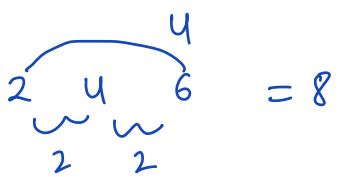
SC : $O(N)$.

Doubt





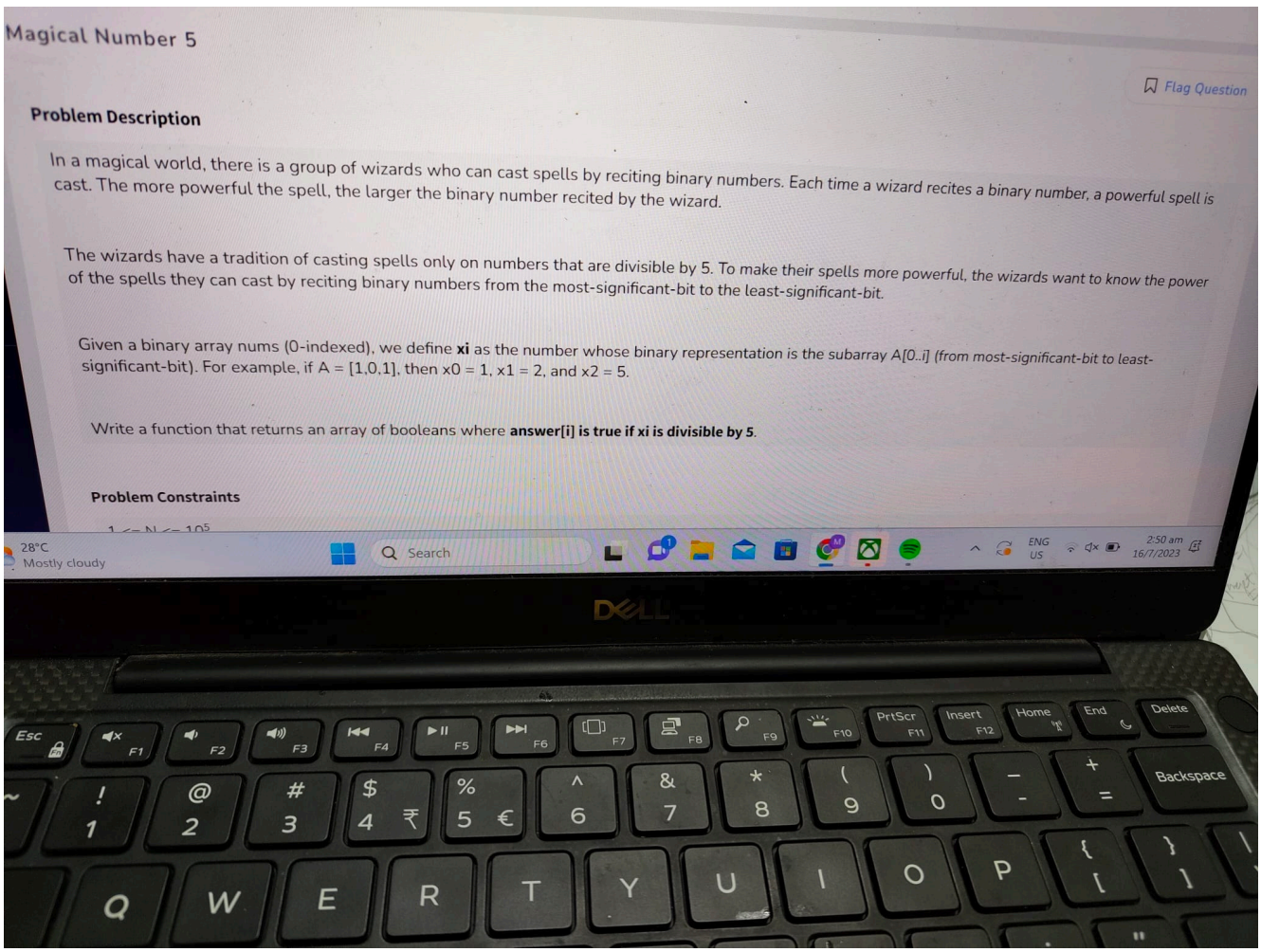
elements $m - p_1 + 1$



$= 8$

0 1 0
1 0 0

2 4
2 6
4 6



A = [1 0 1]

0 1 2

1 2 5

X[0] = A[0:0] = 1 = 1 % 5 = NO

X[1] = A[0:1] = 10 = 2 % 5 = No

X[2] = A[0:2] = 101 = 5 % 5 = No

X(i) = A[0:i]

num = 0

A = [1 0 1]
    0 1 2

i = 0 2 * num + A[i] = 1 % 5
i = 1   = 2 % 5
                      = 5 % 5

↑ i