Day-2

(i) **Rotate Image :-** Given n×n 2D matrix rotate it by 90°(clockwise)

① **Brute force:-** Take one more 2D matrix & put ≡ row's of given matrix to column of new matrix.

eg:- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$

TC :- $O(n^2)$
SC :- $O(n^2)$

② **Optimised :-** Algo :- (Transpose, then reverse)

eg:- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ 1) Transpose → $\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$ 2) reverse every row → $\begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$ Ans

```
{
  for(i=0 —— i<n){
      for(j=0 —— j<i){
          swap( m[i][j], m[j][i]);
      }
  }
  for(i=0 —— i<n){
      reverse( m[i].begin(), m[i].end());
  }
}
```

⇒ code to transpose a matrix. (This code works only for n×n matrix (row == col))

TC :- $O(n^2) + O(n^2) \simeq O(n^2)$
SC :- $O(1)$

(ii) Merge overlapping Subintervals:-

① **Brute force**:- Keep a vector ans, sort the given vector, for every interval check all other intervals, if they merge with it or not, ~~maintain~~ if they merge, push it in ans.

TC:- $O(n\log n) + O(n^2) \simeq O(n^2)$
SC:- $O(n)$

② **optimal Approach**:- Sort given vector, and traverse through it ~~by keeping~~ and push first interval on it, if last element in ans overlaps with our interval then update the last element in DS, if not overlaps, then simply push in ans.

Intution:- Since anay are sorted, the interveles which will be merging are bound to be adjacent.

```
for( i=0 ——— i<n){
    if(ans.empty() or ans.back()[1] < intervals[i][0]){
        ans.push_back(interval[i]);
    }
    else{
        ans.back()[1] = max(ans.back()[1], intervals[i][1]);
    }
}
```

TC:- $O(n\log n) + O(n) \simeq (n\log n)$
SC:- $O(n)$

③ ~~Best~~ optimal, (~~in place~~) :- sort given vector, traverse in vector, keep 1st interval in temp and check if merge or not, if merge, update temp, else push in ans & ~~update~~ increase temp.

```
{ sort(intervals); v<i> temp = interval[0];
  for(auto it : intervals)
      if(it[0] < temp[1])
          temp[1] = max(it[1], temp[1]);
      else{
          ans.push_back(temp);
          temp = it;
      }
  }
}
```

TC:- $(n\log n) + O(n) \simeq (n\log n)$
SC:- $O(n)$

(iii) Merge two sorted Arrays :—

eg:- $v1 = \{1,2,3,0,0,0\}$, $v2 = \{2,5,6\}$ ($m=3$, $n=3$)

o/p:- $\{1,2,2,3,5,6\}$

size of $v1 = 6$ ($m+n$), size of $v2 = 3$

① **Brute force** :— make a new vector $v3$ of size $(m+n)$, sort $v3$, and put all ele. in $v1$ and remaining in $v2$.

TC :- $O((m+n)\log(m+n))$
SC :- $O(m+n)$

② **Optimal, without using extra space** :— As given array are sorted, so iterate on $v1$, keep two pointer, 1st on $v1$ & second on $v2$, now, if $v1[i1] > v2[i2]$, then swap, then & rearrange $v2$ (insert the newly swapped value in its right position) and repeat.

$v1 \rightarrow \{1,4,7,8,10\}$ swap $\{1,2,7,8,10\}$ rearrange $v2 \{3,4,9\}$
$v2 \rightarrow \{2,3,9\}$ $\{4,3,9\}$

~~for(i=0 ... n)~~

TC :- $O(m \times n)$
SC :- $O(1)$

③ **using Gap method** :— Initially take gap as $(m+n)/2$
Take as $p1 = 0$ & $p2 = gap$, move $p1$ & $p2$ by 1 every time till $j < N (m+n)$ & whenever $v1[p2] < v1[p1]$, then swap. everytime reduce gap as $gap = \frac{gap}{2}$ & repeat till $gap > 0$

TC :- $O(\log n)$
SC :- $O(1)$

$N = (m+n) = 8$
$Gap = \frac{N}{2} = \frac{8}{2} = 4$

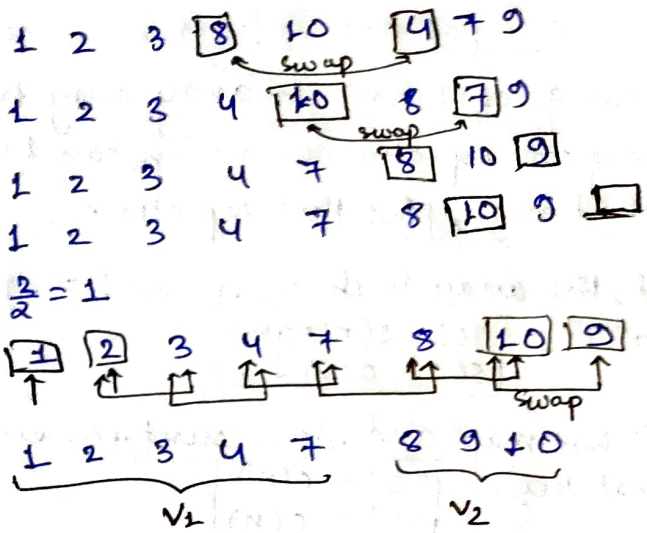
eg:- $\{1,4,7,8,10\}$, $\{2,3,9\}$
$i=0$, $j = gap = 4$, $j < N$

Gap = 4

1	4	7	8	10	2	3	9
0	1	2	3	4	5	6	7
1	4	7	8	10	2	3	9
				swap			
1	2	7	8	10	4	3	9
					swap		
1	2	3	8	10	4	7	9
1	2	3	8	10	4	7	9

Gap = $\frac{4}{2} = 2$

1	2	3	8	10	4	7	9
1	2	3	8	10	4	7	9
1	2	3	8	10	4	7	9



Gap = $\frac{7}{2} = 1$

1 2 3 4 7 | 8 9 10 — sorted

v_1 v_2

```
{
  gap = ceil((float)(n+m)/2);
  while(gap > 0){
      i=0, j=gap;
      while(j < (n+m)){
          if( j<n and v1[i] > v1[j])
              swap(v1[i], v1[j])
          else if( j>=n and i<n and v1[i] > v2[j-n])
              swap(v1[i], v2[j-n]).
          else if( j>=n and i>=n and v2[i-n] > v2[j-n])
              swap(v2[i-n], v2[j-n]);

          i++;
          j++;
      }
      if(gap==1) gap=0
      else gap = ceil((float) gap/2);
  }
}
```

(iv) **Find duplicate no. in array of n+1 integers** :- Given an array of integers in range of [1, n] & array array has (n+1) integers, their is only one repeated no., which can be repeated any no. of times, find that repeated no.
(n+1)

① **Brute force** :- Sort, the array & check if a[i] = a[i+1], if yes then return a[i].

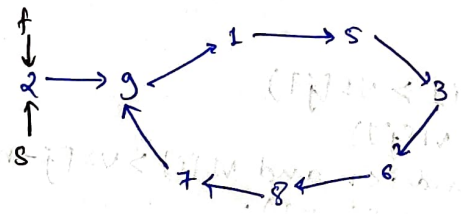
TC :- O(nlogn)
SC :- O(1)

② **Optimal** :- (Hashing) use map and store count for every integer if cnt > 1 return that key.

TC :- O(n)
SC :- O(n)

③ **Linked list Cycle Method (optimised)** Here is tortoise method,

arr { 2 5 9 6 9 3 8 9 7 1 }
0 1 2 3 4 5 6 7 8 9



fast = slow = num[0]
→ move fast by 2 step & slow by 1, until they became equal.
→ move fast to again num[0]
→ now move both pointer by 1 step, until they are equal.
→ return slow.

```
{
    slow = fast = num[0];
    do {
        slow = num[slow];
        fast = num[num[fast]];
    } while (slow != fast);
    fast = num[0];
    while (slow != fast) {
        slow = num[slow];
        fast = num[fast];
    }
    return slow;
}
```

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

(iv) **Repeating and missing number** :- Given an, of size n having elements in [1, n] and 1 ele. is missing, & 1 ele. appears twice, so find missing & repeating no.

eg:- {3, 1, 3} O/P:- M→2, R→3

① **Brute force** :- Sort the array, & find repeating ele. by checking if (a[i] == a[i+1]) & in the same loop to find missing number check if (a[i] = a[i-1] + 1).

TC :- O(nlogn)
SC :- O(1)

② **using hasing** :- Take array of size (n+1) & mark every index as 0, Traverse on num & mark arr[num[i]]++, now on next loop check for arr[i] = 0 ← (missing) & arr[i] = 2 ← (repeating)

TC :- O(n)
SC :- O(n)

③ **Using Math** :- As this stores squares of values, so for higher value, this approach fails. (we need to take long long)

eg:- {4, 3, 6, 2, 1, 1} x → missing, y → repeating.

$S = 1+2+\dots+n = \frac{n(n+1)}{2}$, $S2 = 1^2+2^2+\dots+n^2 = \frac{n(n+1)(2n+1)}{6}$

So, $S - \{4+3+6+2+1+1\} \Rightarrow \{1+2+3+4+5+6\} - \{4+3+6+2+1+1\}$ ($5 \to x$, $1 \to y^2$)

$\Rightarrow 5-1 \Rightarrow (x-y) = 4$ —— ①

$= 4$

now, $S2 - \{4^2+3^2+6^2+2^2+1^2+1^2\}$

$\Rightarrow \{1^2+2^2+3^2+4^2+5^2+6^2\} - \{y^2+3^2+6^2+2^2+1^2+1^2\}$

$\Rightarrow 25-1 \quad (x^2-y^2 = 24)$ —— ②

$= 24$

solving eqn ① & ② $\Rightarrow x^2-y^2 = 24 \Rightarrow (x-y)(x+y) = 24$

$\Rightarrow 4(x+y) = 24 \Rightarrow (x+y) = 6 \Rightarrow x = 6-y$

$x-y = 4$

$6-y-y = 4 \Rightarrow 2 = 2y \Rightarrow y = 1$ ← repeating

$x = 6-y$

$x = 6-1 = 5 \Rightarrow x = 5$ ← missing.

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

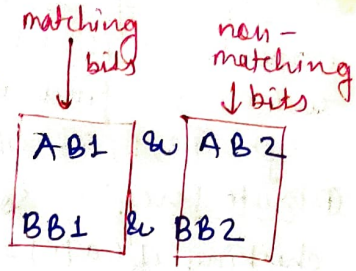
```
{ S = n(n+1)/2, S2 = n(n+1)(2n+1)/2
  m=0, r=0
  for(i=0 — n){
      S -= a[i];
      S2 -= a[i]*a[i];
  }
  m = (S + P/S)/2;
  r = m - S;
}
```

④ **Using XOR property :—**

→ XOR all a[i] ⟶ x

→ x^(1^2^3^---^n) ⇒ x^y = num

→ Seperate an in 2 Baskets



→ Seperate {1,2,---n} in 2 Baskets

→ now XOR both baskets to find the no.

{ AB1^BB1 , AB2^BB2 }

TC :— O(n)
SC :— O(1)

note :— This method doesn't tell which one occur twice & which one is missing . But that can be easily find out in one other loop

```
{
    xor1, set_bit_no;
    int i;
    x=0, y=0, n= an.size();
    xor1 = an[0];
    for(i=1 —— i<n) xor1^=arr[i]; // xor of all array element.
    for(i=1 —— i<=n) xor1^=i ;    // xor prev result with {1—n}.
    set_bit_no = xor1 & ~(xor1-1); // get rightmost set bit in set_bit_no
    for(i=0 —— i<n){
        if(an[i] & set_bit_no) x^=an[i];    //set 1
        else y^=an[i];                      //set 2
    }
    for(i=1 —— i<=n){
        if(i & set_bit_no) x^=i;    // Building another baskets
        else y^=i;                  // and doing xor with previous
    }                               // result(Basket).
    int x_count=0; // to check which is missing & which is repeating
    for(i=0 — i<n){
        if(an[i]==x)
            x_count++;
    }
    if(x_count==0)
        return {y,x};
    return {x,y}
}
```

(Vi) Inversion of Array (Pre.req :- Merge sort)

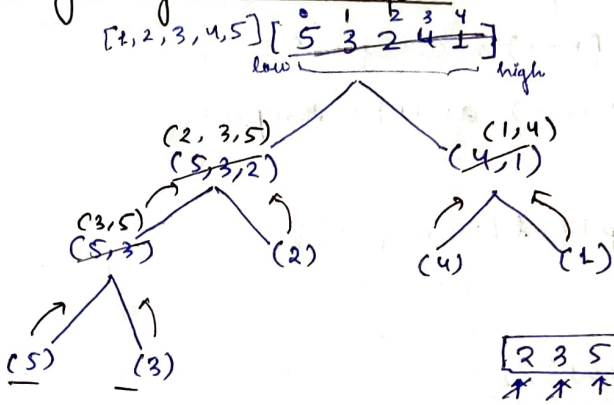
eg:- {2, 5, 1, 3, 4} o/p:- 4 {(2,1), (5,1), (5,3), (5,4)}

Qⁿ → print ~~ctt~~ no. of pairs such that $i < j$ and $arr[i] > arr[j]$

① **Brute force :-** for every i (0-n) check if $arr[i] > arr[j]$ (i-n) and return cnt.

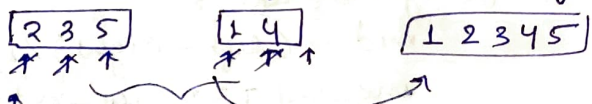
TC :- $O(n^2)$
SC :- $O(1)$

② **using Merge sort Technique :-**



low = 0, high = n-1
mid = (low + high)/2
at every step, it is divided as,
(low — mid) & (mid+1 — high)

→ so, in merge sort while merging left part is sorted in itself and same as right part



This merging technique is used in merge sort, so here we'll modify it a bit and use it for our question.

eg: → [5] [3] a[i] > a[j], so in first part everything after 5 will be greater than 3.

here (5,3) is a pair.. → +1

→ [3 5] [2] , everything right after 3 is greater than 2 so they can form pair with 2.

(3,2), (5,2) → +2

→ [4] [1] , (4,1) → +1

→ [2 3 5] [1 4] (2,1), (3,1), (5,1), (5,4) → +4

sum = 8

TC :- $O(n \log n)$
SC :- $O(n)$

#code

```
int merge(arr[], temp[], left, mid, right){
    inv_cnt = 0,
    i = left;
    j = mid;
    k = left;
```

```
while ( (i <= mid-1) && (j <= right)){
        if (arr[i] <= arr[j]){
            temp[k++] = arr[i++];
        }
        else {
            temp[k++] = arr[j++];
            inv_cnt = inv_cnt + (mid - i);
        }
}
while (i <= mid-1)  temp[k++] = arr[i++];
while (j <= right) temp[k++] = arr[j++];
for(i = left; i <= right; i++) arr[i] = temp[i];
return inv_cnt;
}
int merge_sort ( arr[], temp[], left, right) {
    int mid, inv_cnt = 0;
    if (right > left){
        mid = (left + right)/2;
        inv_cnt += merge_sort(arr, temp, left, mid);
        inv_cnt += merge_sort(arr, temp, mid+1, right);
        inv_cnt += merge (arr, temp, left, mid+1, right);
    }
    return inv_cnt;
}
int main() {
    arr[] = {5, 3, 2, 1, 4};
    int n=5;
    temp[n];
    ans = merge_sort (arr, temp, 0, n-1);
    cout << ans << endl;
    return 0;
}
```