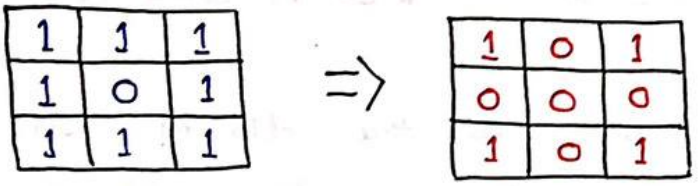
Striver

Dsa

Sheet

"180"

Set Matrix Zero : Problem no - 1



Time and Space for Brute -

Time - O((N * M) * (N+M))

Space - O(1)

Brute force

1. first, use two loops to traverse all the cells.
2. If any cell (i,j) contains the value 0, we will mark all cells in row i and column j with -1 except those which contain 0.

↓ perform for every cell containing 0.

3. finally mark all the cells containing -1 with 0.

```
main (Arraylist < Arraylist < Integer >> mat) {
    for (int i=0 ; i< n; i++) {
        for (int j=0; j<m ; j++) {
            if (mat.get (i).get (j) == 0) {
                matRow (mat, n, m, i);
                matCol (mat, n, m, j);
            }
        }
    }

    for (int i=0 ; i< n; i++) {
        for (int j=0; j<m ; j++) {
            if (mat.get (i).get (j) == -1) {
                mat.get (i).set (j, 0);
            }
        }
    }
```

// contains 0 or not //

```
matRow (mat,n, m, i){
    for (int j=0 ; j<m ; j++) {
        if (mat.get (i).get (j) != 0)
            matrix.get (i).set (j, -1);
    }
}
```

Same as the column

Better Approach :- Using Extra Space

1. Declear two array : a row array of size N and col array of size M.
2. Then, we will use two loops to traverse all the cells of matrix.
3. If any cell (i,j) contains the value 0, we will mark ith index of row array i.e. row[i] and jth index of col array col[j] as 1. ⟶ perform for every cell containning 0.
4. traverse matrix and if row[i] or col[j] contain 1 so marked as zero.

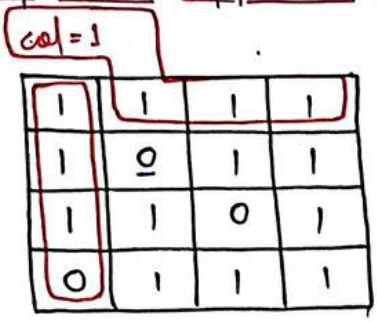
```
ZeroMatrix (ArrayList < ArrayList < Integer >> mat) {

    int[] row = new int[n];
    int[] col = new int[m];

    for (int i=0 ; i< n ; i++) {
        for (int j=0; j<m; j++) {
            if (mat.get(i).get(j) == 0) {
                row[i] = 1;
                col[j] = 1;
            }
        }
    }
// mark the array with 1 if contains 0 //

    for (int i=0 ; i< n; i++) {
        for (int j=0 ; j<m; j++) {
            if (row[i] == 1 || col[j] == 1) {
                mat.get(i).set(j, 0);
            }
        }
    }
    return mat;
```

Time and Space complexity :- $O(2*(N*M))$

Complexity - $O(N) + O(M)$

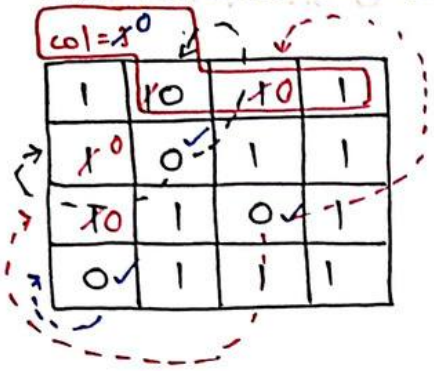
row arr. col arr.

Optimal Approach :-



1. We take the arrays inside the Matrix.
   col[0] → row[n]
   row[0] → col[m]

first fill the zero inside the 2-D Matrix.



After step 1 is completed, we modify the cells from (1,1) to (n-1, m-1) using val. from the 1st row, 1st col. and colo variable.

finally change the 1st row if mat[0][0] = 0 and column if colo = 0

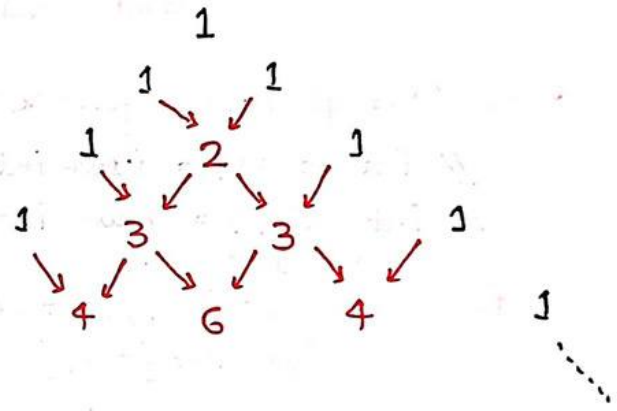
```
main (ArrayList < ArrayList < Integer>> mat){
    // int row[] = new int[n]; --> mat[..][0]
    // int col[] = new int[m]; ---> mat[0][...]
        int colo = 1;
  for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            if(mat.get(i).get(j)==0){
                mat.get(i).set(0,0);
                if(j!=0)
                    mat.get(0).set(j,0);
                else
                    colo = 0;
                }
            }
        }

// Step 2: Mark with 0 from (1,1) to (n-1, m-1)

    for(int i= 1 ; i<n; i++){
        for(int j= 1 ; j<m; j++){
            if(mat.get(i).get(j)!=0){
                if(mat.get(i).get(0)==0 || mat.get(0).get(j)==0)
                    mat.get(i).set(j,0);
                }
            }
        }
```

Step3: finally mark the 1 col & then 1st row.

```
if (mat.get(0).get(0) == 0)
        for (int j = 0; j < m; j++) {
                mat.get(0).set(j, 0);
                                }

if (col0 == 0)
        for (int i = 0; i < n; i++) {
                mat.get(i).set(0, 0);
                                }

return mat;
```

Time - O(2*(N*M))

Pascal Triangle - Problem No-2



There are 3 variations -

① Give R & C, find the element at place.

R = 5, C = 3

answer = 6

② Print any Nth row of pascal triangle. N = 5

answer = 1 4 6 4 1

③ print the entire pascal triangle.

N = 5 - - - - - - - - - -

$$^{n}C_{r} = \frac{n!}{r! \times (n-r)!}$$

↳ Brute force

$$^{R-1}C_{C-1} = {}^{4}C_{2} = \frac{4 \times 3 \times 2 \times 1}{(2 \times 1)(2 \times 1)} = 6$$

Observation -

$$^{7}C_{2} = \frac{7!}{2! \times 5!} = \frac{7 \times 6 \times \cancel{(5 \times 4 \times 3 \times 2 \times 1)}}{2 \times 1 \times \cancel{(5 \times 4 \times 3 \times 2 \times 1)}} = \frac{7 \times 6}{2 \times 1}$$

$$^{10}C_{3} = \frac{10 \times 9 \times 8}{3 \times 2 \times 1} = \frac{10}{1} \times \frac{9}{2} \times \frac{8}{3}$$

↳ write everything 3 places

```
fun NCR (n, r) {
    res = 1;
    for (int i=0; i<r; i++){
        res = res * (n - i);
        res = res / (i+1);
    }
}
return res;
```

TC - O(r) } Time and Space
SC - O(1) }

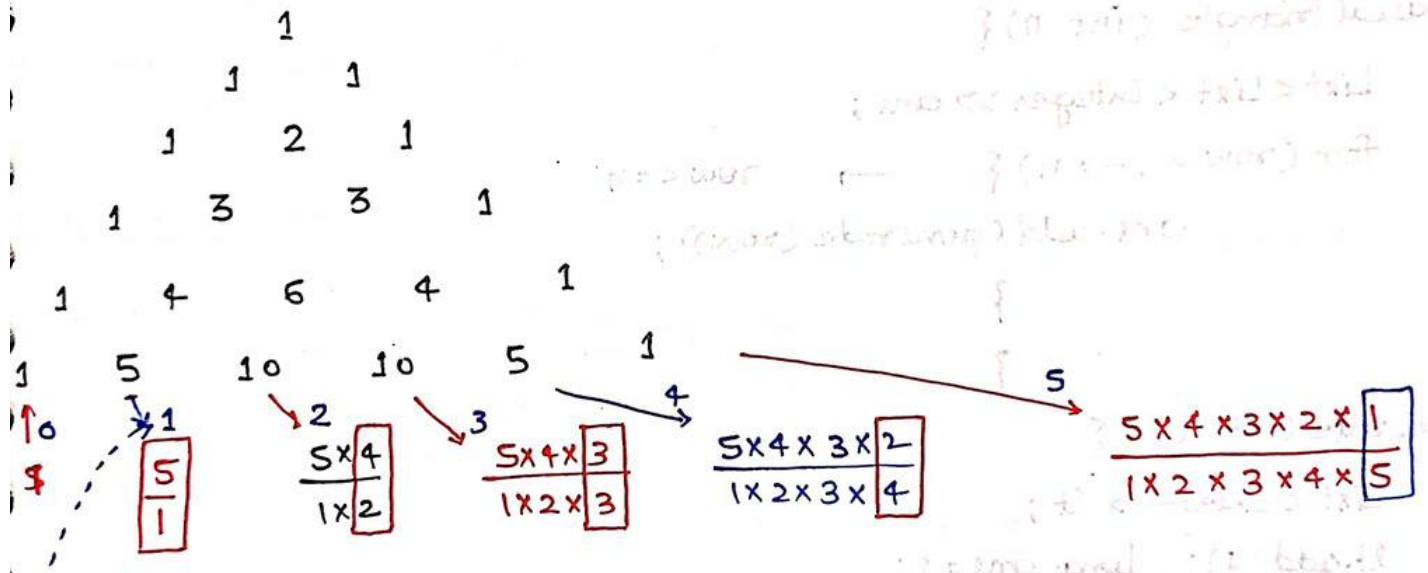
2nd Observation for the 2nd variation - : for priting the row.

Nth row has the N elements.

formula. $^{r-1}C_{c-1}$

```
for (c=1; c<=n; c++){ ---> O(N)
    print ( fun NCr (n-1, c-1)); -> O(r)
}
```

Time complexity - O(N x r)



$^5C_2 = \frac{5\times4}{2\times1}$, $^5C_3 = \frac{5\times4\times3}{1\times2\times3}$

$^5C_4 = \frac{5\times4\times3\times2}{1\times2\times3\times4}$

Time complexity ⇒ O(N)
Space complexity ⇒ O(1)

① The first and the last element is always 1.

$ans \times \frac{(row - col)}{column}$

```
ans = 1
print (ans) // for first element
for (i= 1; i<n; i++) {
    ans = ans * (n-i);
    ans = ans / i;
    print (ans); }
```

3. 3rd Variation -

Extreme bruteforce -

```
ans = [ ];
for (row = 1→n)
    {
     temp= [ ];
    for (col= 1→row)
        {
         temp.add (ncr (row-1, col-1));
        }
     ans.add (temp);
    }
return ans;
```

Time complexity :

$O(n*n*r) \sim O(n^3)$

Space complexity :

$O(1)$

Optimization : So we use the 2nd Variation for priting any row which is taking O(N) for priting.

So for whole the triangle it will be $O(N*N) \simeq O(N^2)$

```
fun. pascalTriangle (int n){
        List<List<Integer>> ans ;
        for (row = 1→ n) {      →     row<=n
                ans.add (generate (row));
                        }
                        }
fun. generate (int row) {
        List< Integer> lt ;
        lt.add (1); , long ans = 1 ;
        for ( int col = 1→ row) {  → col<row
                ans = ans * (row- col);
                ans = ans / col;
        lt.add(ans) ;  ---> it need a int type conversion
                    }
        return lt ;
                    }
```

3. Next Permutation - Problem - 3

arr[] = [3, 1, 2] = $n! = 3! = 6$ ways

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2 → like if we have to find the next permutation so answer
3 2 1 is 3 2 1

Write them in Dictionary order = sorted order.

123 < 132 < 213 .....

edge case : 3 2 1 so the Next permutation
↓
the answer will be 1 2 3

Bruteforce : 1. Generate all permutation in sorted sorder. → recursion
2. Line Search for finding the given element.
3. Next index

Time complexity = $\{N! \times N\}$

Optimal solution:

1. longer prefix match

arr[] = {2 1 5 4 3 0 0} n = 7

try to match every thing

{2 1 5 4 3 0 0} one no. on the left that we can re-arrange but no change. : get the same.

{2 1 5 4 3 0 0} No re-arrange because prefix match.

{2 1 5 4 3 0 0}

{2 1 5 4 3 0 0}

we can make
0 0 3
0 3 0
} both are less than 300 so it will same. not get anything greater

again

2 1 5 4 3 0 0 →
~~5 4 3 0 0~~ (equal)
~~4 5 3 0 0~~ (less)
~~3 4 5 0 0~~ (less)

[2 1 5 4 3 0 0]

checking for which one is just greater than this.

5 4 3 1 0 0 (biggu)
(but it is right after this it is a question)

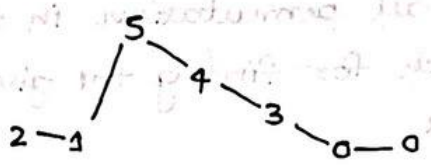
2 3
2 4
2 5

} In these we will take 23 according to dictionary and 20 is not possible because 0 is less than 1.

2 3
first two elements

1. figure out the break point.

we traverse from right and if the graph is increasing then we have nothing on the right that smaller.



↳ a[i] < a[i+1] → this will give us the break point.

2 1 5 4 3 0 0

2 3 0 0 1 4 5

2. look at the right of element and find the element just greater than the element, here which is 3.
So that you stay close.

5 4 1 0 0

3. Try to place the number in sorted order.

```
ind = -1;
for (i = n-2; i >= 0; i--) {
        if (a[i] < a[i+1]) {
                ind = i;
                break;
                }
        }
```

if it is -1 then the array will be same, because there was not any dip point. reverse the array and it will be the answer.

```
if (ind == -1) {   → Means the given array is the last permutation.
    rev(arr)
}

for (i = n-1; i >= ind; i++)
{
    if (arr[i] > arr[ind])
        swap(arr[i], arr[ind]);
        break;
}

reverse(arr, ind+1, n-1);
```

increasing order ← so just reverse

2 3 | 5 4 1 0 0

⇓ small

2 3 0 0 1 4 5

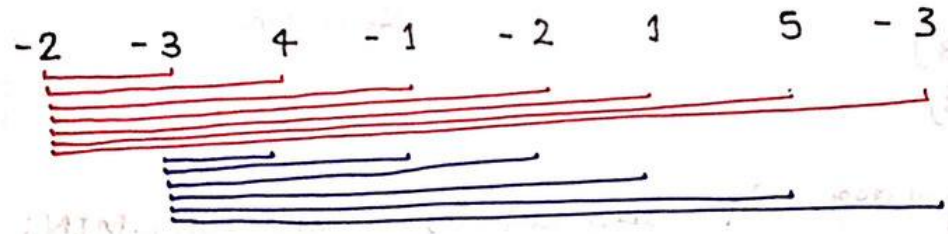
Time complexity - O(3N)
Space complexity - O(1)

Kadans Algorithm : Problem No-4 : Max Array (sub) Sum.

arr[] = [-2, -3, 4, -1, -2, 1, 5, -3]

ans = 7

Bruteforce: find all the sub array sum.

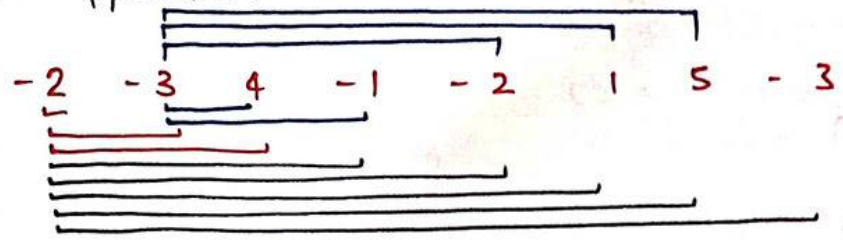


Similarly find all the possible sub array sum and select the max one.

```
max = INT_MIT;
for (i=0; i<n; i++){
    for (j=i; j<n; j++){
        sum = 0;
        for (k=i→j){
            sum = sum + arr[k];   max = max(sum, max);
        }
    }
}
```

Time complexity = $O(N^3)$
Space complexity = O(1)

Better Approach :



//max = INT_MAX; //

```
max = INT- MIN;
for(i=0; i<n;i++){  sum = 0;
        for (j= i ; j< n; j++) {
                sum += arr[j];
                max = max (sum, max);
                    }
                }
```

Time complexity = $O(N^2)$

Optimal Approach -

arr[] = [-2, -3, 4, -1, -2, 1, 5, -3]
index: 0

ans = 7

maxi = ~~INT-MIN~~ ~~-2~~ 7

sum = ~~0~~ ~~-2~~ ~~0~~ ~~-3~~ 0 ~~4~~ ~~3~~ ~~1~~ ~~2~~ +5 = 7 ---> (check every time sum < 0) then convert sum = 0

every step compare sum with maximum and update maximum

sum < 0

[-2, -3] (-5) < [-3] (-3)

```
maxSubArray (arr, n){  long sum =0, maxi = LONG-MIN;
    for(i=0; i<n;i++){
            sum += arr[i];
            if (sum<0){
                sum=0;
                }
            maxi = Math. max (maxi, sum);
        }
    return maxi;
    }
```

Check if there is no sum >= 0 so it is zero.

use this line over there

Time Complexity - O(N)

Sort an Array of 0's, 1's and 2's

arr[] = | 0 | 1 | 2 | 0 | 1 | 2 | 1 | 2 | 0 | 0 | 0 | 1 |

Bruteforce - We can use the sorting algo. and if we use merge sort

Timecomplexity - nlogn
Space complexity - O(N)

Better solution - keep three variable and iterate and increase the count of variable.

Time complexity - O(2N)
Space complexity - O(1)

Optimization -

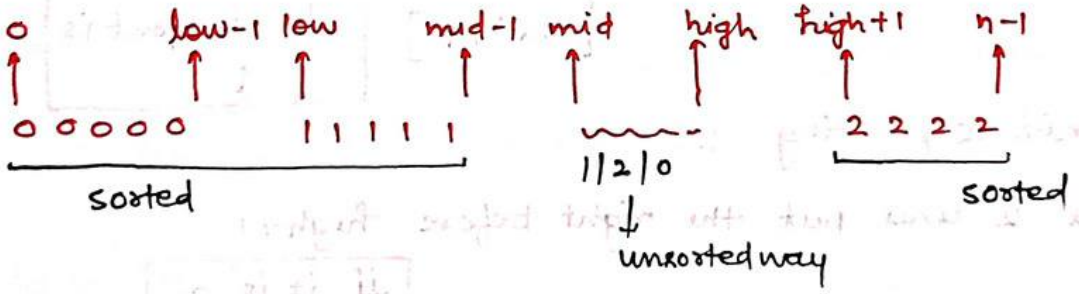
Dutch National flag Algorithm

pointers ← low, mid, high

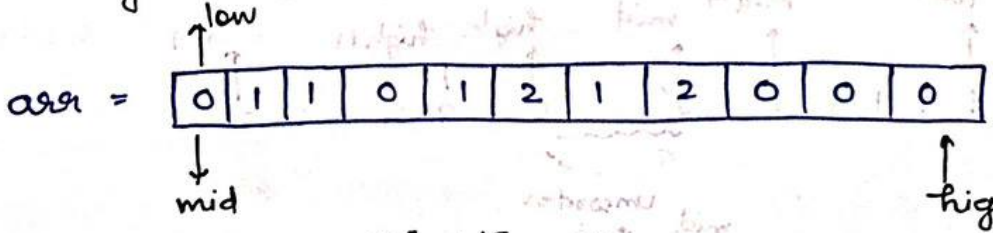
[0 .... low-1] → 0 extreme left

[low .... mid-1] --> 1

[high+1 , n-1] --> 2 extreme right



firstly the mid will be on the first position because the entire array is unsorted.

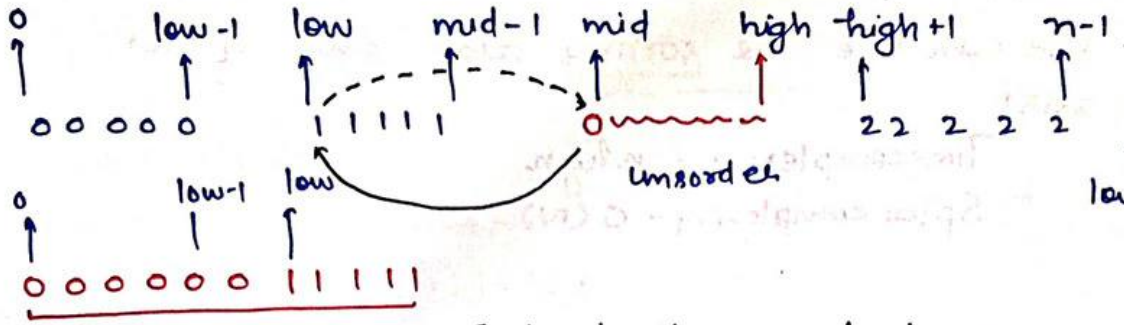


a[mid] = 0

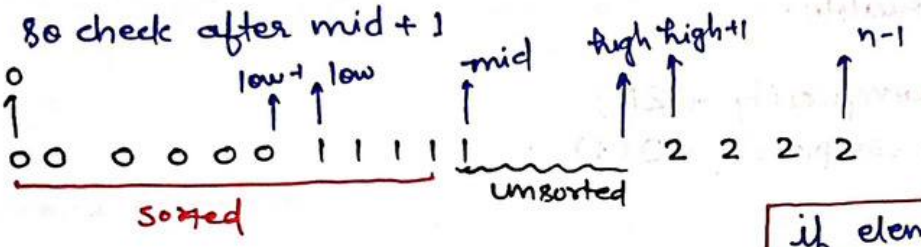
a[mid] = 1 mid can be contain 0, 1, 2

a[mid] = 2

but how can we decide that the number point by mid is in sorted order.



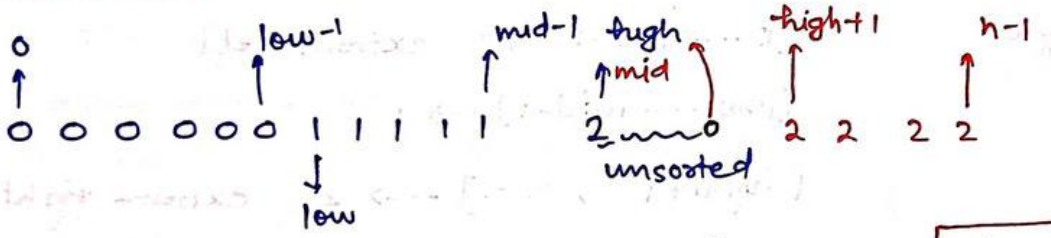
So low is the correct place



if element is zero

[swap (a[low], a[mid])
low++, mid++]

and we can after combine the mid pointer is also in sorted order



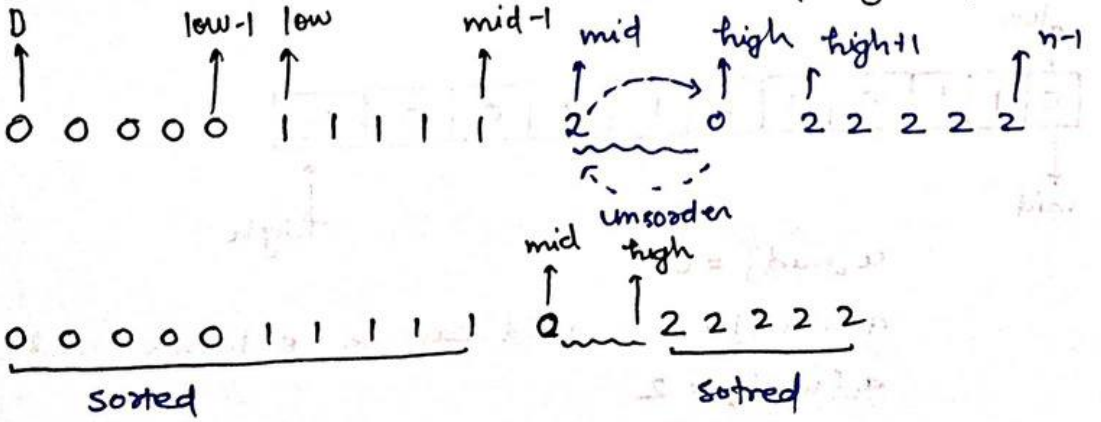
[mid++] if element is 1

then the mid is pointing 2.

take the 2 and put the right before high+1

if it is 2

[swap (arr[mid], a[high]
[high--]



```
fun Array (arr, n){
    low=0, mid = 0, high= n-1;
    while (mid <= high){
      if (arr[mid] == 0){
          swap (arr[low], arr[mid]);
          low++;
          mid++;
                        }
      else if (arr[mid] == 1){
            mid++;
              }
      else {
            swap(arr[mid], arr[high]);
            high--;
          }
                  }
```

Time complexity - O(N)
Space complexity - O(1)

Best Time to By and Sell : Problem no - 05

arr[ ] = [7, 1, 5, 3, 6, 4]

i is the day

profit = 5

if we buy a stock on day 2 then sell on day 5 we can gain max profit.

Brute force
① Use a loop i = 1 → n
② Use another j = i+1 → n
③ if arr[j] > arr[i] take a diff and compare and store it to maxProfit variable.

Time complexity - O(N²)

No extra space

Optimal Approach -

① Create maxPro and put 0.
② create a min price and store the large value (Integer: MAX-VALUE)
③ Run a loop 0 to n
④ Update the /max profit/ if greater than current element
min price
⑤ take a diff of min price with current element of array and compare and maintain maxPro.

```
maxProfit (arr) {
        int maxPro = 0;
        int MinPrice = MAX_VALUE;
      for (i=0; i<arr.length ; i++) {
        minPrice = Math.min (arr[i], minPrice);
        maxPro = Math.max (maxPro , arr[i] - minPrice);
                                    }
      }
```

Rotate Matrix 90° : Problem No 6

Bruteforce - Take a dummy Matrix of nxn and then take the first row and put into the put in the last Column of dummy Matrix and similarly for diff. rows.

1	2	3
4	5	6
7	8	9

⇒

7	4	1
8	5	2
9	6	3

just using two loop.

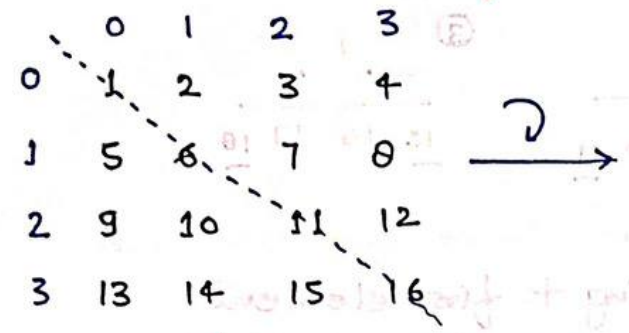
Time complexity = $O(N^2)$
Space complexity = $O(N*N)$

```
int[][] dummy;
for(i= 0→n) {
      for= (j= 0→n) {
        dummy [j][n-i-1] = mat[i][j];
      }      }
```

Optimal Approach :-

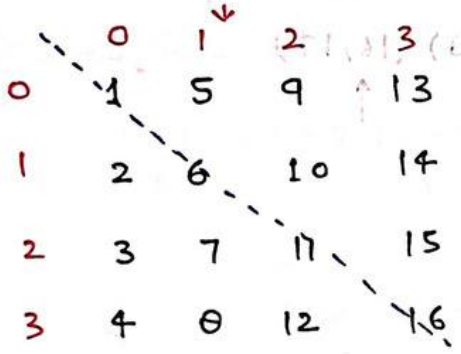
By observation we can see that first column of the original matrix is the reverse of the first row of rotated matrix. So firstly transpose the matrix and then reverse the each row.

① Transpose the Matrix
② Reverse every Row



	0	1	2	3
0	13	9	5	1
1	14	10	6	2
2	15	11	7	3
3	16	12	8	4

Transpose



Observes → The Diagonal elements is not changed after transpose the matrix.

the elements changes -

[0][1] → [1][0]
[0][2] → [2][0] → Swapping behaviour
[0][3] → [3][0]
[1][2] → [2][1]

```
rotate(int[][] mat){
    for(i=0; i<mat.length-1; i++){
        for(j=i+1; j<mat[0].length; j++){
            int temp=0;
            temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
                    }
                }

    for(i=0; i<mat.length; i++){
        for(j=0; j<n/2; j++){
                int temp=0;
                temp = mat[i][j];
                mat[i][j] = mat[i][n-j-1];
                mat[i][n-j-1] = temp;
                    }
                }
```

Time complexity - O(N*N)

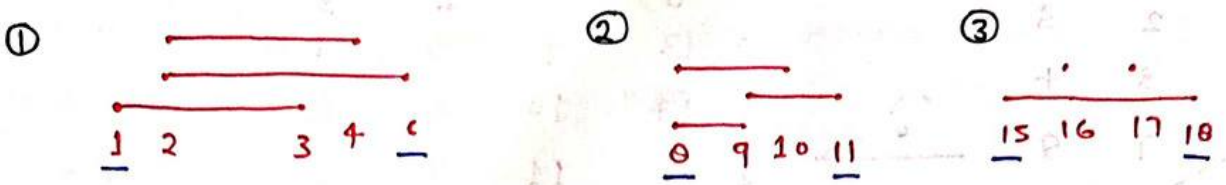
Space - O(1)

Merge Intervals : Problem No 8

Merge overlapping Subintervals.

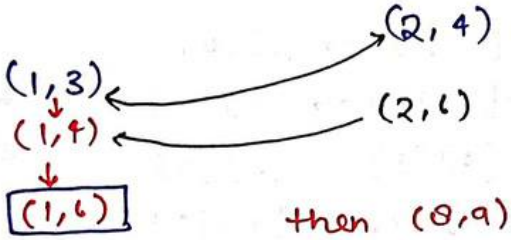
(1,3) (2,6) (8,9) (9,11) (8,10) (2,4) (15,18) (16,17)

(1,6) (8,11) (15,18)

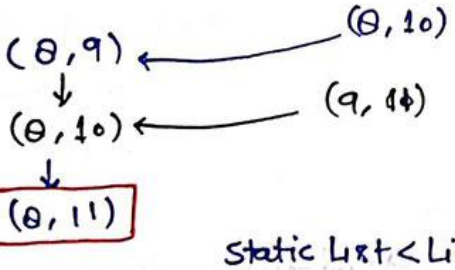


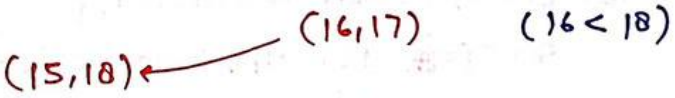
Bruteforce - ① Sort them up. According to first element and if the first element is equal then decide by second.

(1,3) (2,4) (2,6) (8,9) (8,10) (9,11) (15,18) (16,17)



then (8,9) not overlapping 8 > 6






```
Static List<List<Integer>> mergeInterval(int[][] arr){
    int n = arr.length;
    Arrays.sort(arr, newComparator<int[]>(){
        public int compare(int[] a, int[] b){
            return a[0] - b[0];
        }
    });

List<List<Integer>> ans = new ArrayList<>();
for(i=0; i<n; i++){
    int start = arr[i][0];
    int end = arr[i][1];
```

```
if(!ans.isEmpty() && end <= ans.get(ans.size()-1).get(1)){
        continue;
    }
for(j=i+1; j<n; j++){
        if(arr[j][0]<=end){
            end = Math.max(end, arr[j][1]);
                }
        else{
            break;
            }
                }
ans.add(Arrays.asList(start, end));
}
return ans;
```

Time Complexity - NlogN + O(2N)

Space complexity - O(N)

Optimal Solution -

(1,3) (2,4) (2,6) (8,9) (8,10) (9,11) (15,18) (16,17)

① (1,3) Store intially, go to (2,4) and check, here 2 is smaller than 3. So overlapping so end update (1,4)

② (2,6) pointing and 2 < 4 so update the end -

(1,4) → (1,6) ✓

③ (8,9) when this arrives so 6 < 8 so this is the next interval.

form new interval - (8,9)

(8,9) compare 9 with next interval 8,10

9 > 8 so update the end.

8,10

(9,11) compare with_ 10 > 9 overlapp.

update the end.

8,11 ✓

new interval.

15,18 ✓

```
mergeOverlapInterval (int[][] arr){
    n = arr.length;
    Comparator for sort.

List<List<Integer>> ans;
for (i=0; i<n; i++){
if (ans.isEmpty() || arr[i][0] > ans.get(ans.size()-1).get(1)){
        ans.add(Arrays.asList(arr[i][0], arr[i][1]);
        }
else{
    ans.get(ans.size()-1).set(1, Math.max(ans.get(ans.size()-1).
                                    get(1), arr[i][1]));
    }
  }
return ans;
        }
```

Time complexity = NlogN + O(N)
Space complexity = O(N).

Merge Sorted Array : Problem 9

Brute force - Using extra space.

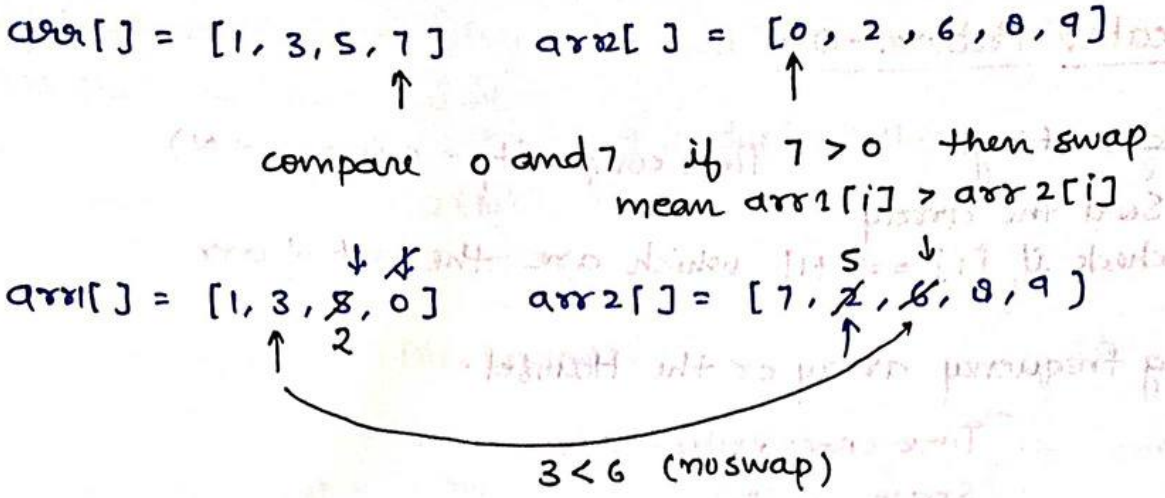
① Declear an array of size n+m.
② If arr1[left] < arr2[right] insert arr1[left] and left++;
③ If arr1[left] > arr2[right] insert arr2[right] and right++;
④ arr1[left] == arr2[right]
insert any of them and increase the pointer by 1.
⑤ If one pointer reaches the end so traverse the second left array and insert all of the element.

Time complexity - O(N+M) + O(N+M)

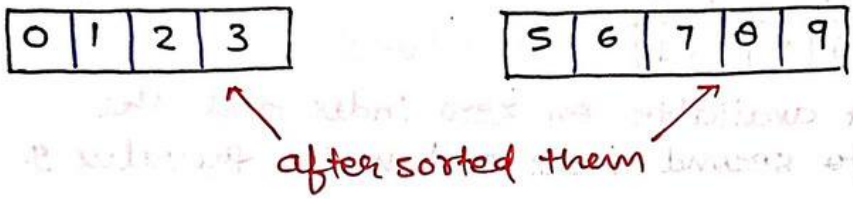
↙ for filling the elements of arr1 and arr2 to arr3

↓ is for filling back the two given arrays from arr3.

Optimal Approach - 1



So Now the Both the arrays the order is right place. the elements we got not in correct order by they are in correct array.




```
void merge (long[]arr1, long[] arr2, int n, int m) {
    int left = n-1;
    int right=0;
    while (left>=0 ff right<m){
        if ( arr1[left] > arr2[right]) {
            long temp= arr1[left];
            arr1[left] = arr2[right];
            arr2[right] = temp;
            left--;
            right++;
                                    }
        else {
            break;
            }
    }
Arrays.sort (arr1);
Array.sort (arr2);
```

Time complexity - O(min(N, M)) + O(N log N) + O(m log m).

Extra space - O(1)

Find the Duplicate: Problem - 08

Solution 1: Using sorting Time complexity - O(nlogN + N)

① Sort the array
② check if [i] == [i+1] which are the part of arr

Solution 2:
Using frequency array or the Hashset.

Time complexity - O(N)
Space - O(N)

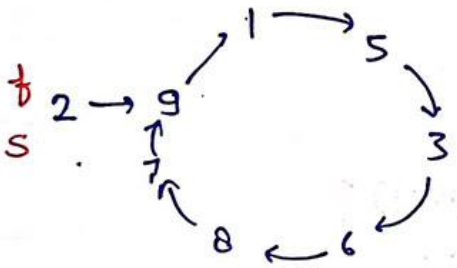
Solution 3: LinkedList cycle method.

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

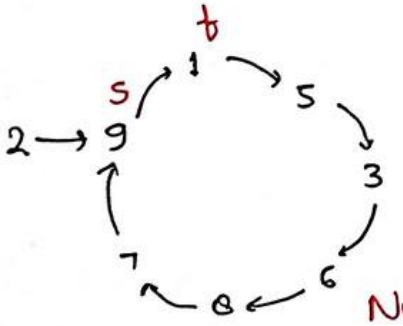
first write the element available on zero index and the value is 2. then go to second index and write the value 9.

2 →

Similary create

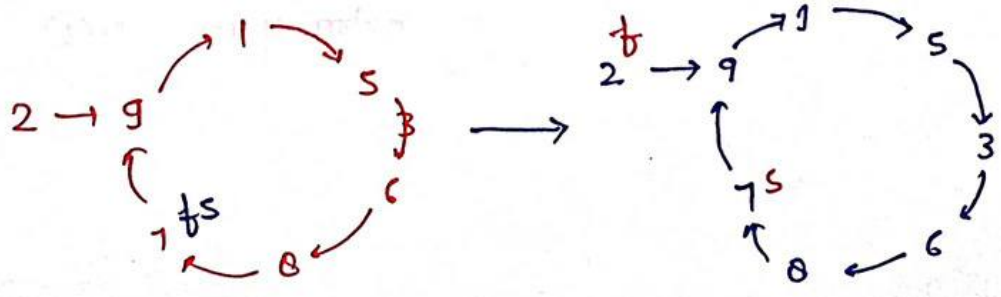


f will move 2 steps.
s will move 1 step.



after a certain time the s and f will lying on the same Number. so stop pointer

So Now take the fast pointer and place it on the first number. Now increase or move the pointer by 1.

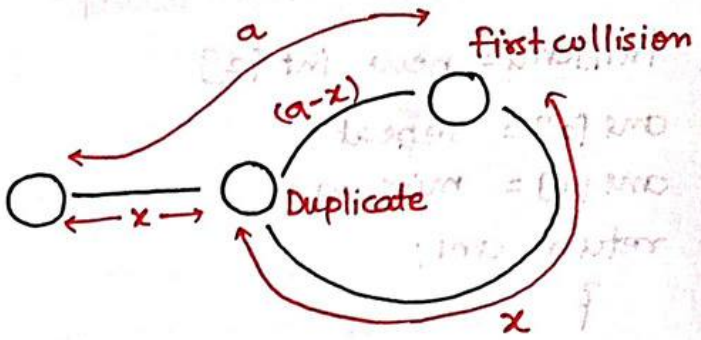


So the next point they meet will be the number which is the duplicate number.

9 is duplicate no.

Time complexity - O(N)

Space complexity - O(1)




```
findDuplicate (int[] nums){
    int slow = nums[0];
    int fast = nums[0];
    do{
      slow = nums[slow];
      fast = nums[nums[fast]];
    }while(slow != fast)

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

Missing and Repeating Number : Problem : 11

Bruteforce -

1. Run a loop i = 1 → n
2. for each integer i we will count the occurance.
3. Store those elements that have 2 and 0 occurance.

```
findMissingRepeatingNumber(arr){
        n = arr.length;
    int repeat = -1, missing = -1;
for(int i = 1; i<=n; i++){
                int cnt = 0;
            for(int j = 0; j<n; j++){
                    if(arr[j] == i) cnt++;
                                    }
        if(cnt == 2){
                        repeat = i;
                    }
```

```
else if (cnt == 0) missing = i;
if (repeating != -1 && missing != -1) {
                    break;
                  }
int[] ans = new int [2]
ans [0] = repeat;
ans [1] = missing;
return ans;
  }
```

Time complexity - $O(N^2)$
Space complexity - $O(1)$

Better Approach - Using Hashing.

Time complexity - $O(2N)$
Space - $O(N)$

N, for push the element in freq. array and second for traverse.

Optimal Solution - Using Maths.

arr[] = [4 3 6 2 1 1] , n = 6

x → repeating, y = missing

Sum of all elements (S) = 4+3+6+2+1+1 = S

and also sum the element (1 to 6) = 1+2+3+4+5+6 = Sn

Sum of first N Natural no = $\frac{n \times (n+1)}{2}$

= S - Sn

= [4+3+6+2+1+1] - [1+2+3+4+5+6]

⇒ If we open bracket then

⇒ 1 - 5 = (x - y) on observing

⇒ -4

x - y = -4 —— ①

again try some thing new on eq. and for solving x - y = -4 we need 1 more eq.

$= (1^2+2^2+3^2+4^2+5^2+6^2) - (1^2+2^2+3^2+4^2+5^2+6^2)$

$= 1 - 25$

$\Rightarrow -24 \Rightarrow x^2 - y^2$

for square of first N natural no. → sum

$$\frac{n*(n+1)\times(2n+1)}{6}$$

$(x-y)(x+y) = -24$

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

$x+y = +6$

Now

$$\begin{array}{r} x+y = +6 \\ x-y = -4 \\ \hline 2x = 2 \end{array}$$

$x = 1$

$x+y = 6$

So

$y = 6 - x = 6-1$

$= 5$

Time complexity - O(N)

Space complexity - O(1)

```
fun missing (arr){
        long n = arr.length;
 log SN = (n*(n+1))/2 ;
 long S2N= (n*(n+1)*(2n+1))/6 ;
S=0, S2=0;
for (i=0; i<n; i++){
 S = S+arr[i];
 S2+= (long) arr[i] *(long) arr[i];
                }

long val1 = S -SN;
long val2 = S2-S2N;
val2 = val2/val1 ;

long x = (val1+val2)/2;
long y =x-val1;
int[] one = {(int) x, (int) y};
return one;
 }
```

Count Inversions in an Array: Problem 12

If an array is given so you have to count the pairs. which are following certain conditions.

① $i<j$
② $arr[i] > arr[j]$

$N=5$, $arr[] = [1,2,3,4,5]$
pairs = 0

$N=5$, $arr[] = [5,3,2,1,4]$
pair = 7
(5,1) (5,3) (5,2) (5,4) (3,2)
(3,1) (2,1)

$N=5$ $arr[] = [5,4,3,2,1]$
pairs = 5

Bruteforce Approach -

```
fun. Inversions(int a[], int n){
int cnt = 0;
for (i=0; i<n; i++){
    for (j=i+1; i<n; j++){
        if (a[i] > a[j]) cnt++;
    }
}
return cnt;
}
```

Time complexity - $O(N^2)$
space complexity - $O(1)$

Optimize solution -

Intuition: take the array which are sorted.

[2, 3, 5, 6]
↑ ↑ ↑ ↑

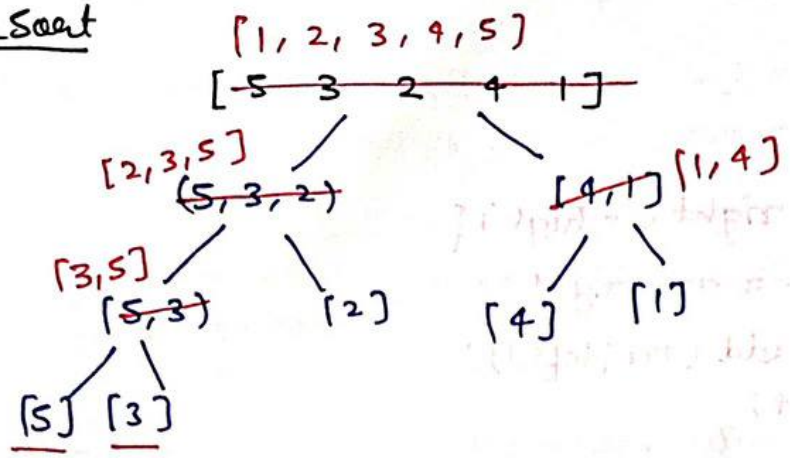
[2, 2, 4, 4, 8]
↑ ↑ ↑ ↑ ↑

Here $3 > 2$ so it can create a pair with 2. and because the array is sorted its means all the members on the right side of 3 is also can be made a pair with 2.

$+3 + 3 + 2 + 2 = 10$

Somehow map the array in that way its left part is sorted and right part is also sorted.

MergeSort



cnt = 0

[5] [3]

these are two arrays and both of them are sorted. I keep two pointer 3 is smaller so it can form a pair with 5.

so add +1 to ans.

[3,5]

we have

[4] [1]

1 can form a pair with 4. ans +1 to ans.

[3,5] [2]

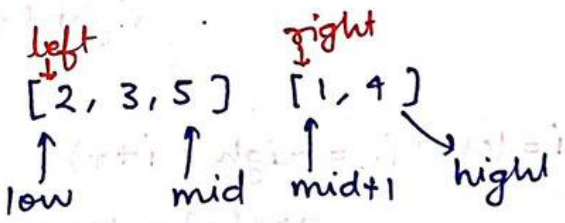
Similarly — find two pair — add +2 to ans.

[2,3,5]

[2,3,5] [1,4]

ans + 1 + 1 + 1 because 1 < 2

ans + 1 because 4 < 5




```
int numberofInversion(arr){
    return mergesort(arr, 0, n-1);
}

fun mergeSort(int[]arr, int low, int high){
    int cnt = 0;
    if (low >= high) return cnt;
    int mid = (low + high)/2;
    cnt += mergesort(arr, low, mid);
    cnt += mergesort(arr, mid+1, high);
    cnt += merge(arr, low, mid, high);
    return cnt;
}
```

```
merge(int[]arr, int low, int mid, int high){
ArrayList<Integer> lt:
int left = low;
int right = mid+1;
int cnt= 0;
while (left <=mid ff right <= high){
        if (arr[left] <= arr[right]{
                temp.add (arr[left]);
                left++;
                        }
        else {
                temp.add (arr[right]);
                right++;
                cnt= cnt + (mid - left+1);
                }
        }
while (left<= mid){temp.add (arr[left]);
                left++;
                }
while (right <= high) { temp.add (arr[right]);
                right++;
                }
for (i= low; i<= high ; i++){
                arr[i] = temp.get (i- low);
                        }
        return cnt;
        }
```

Time complexity - (nlogN)
Extraspace - O(N)

Search in a Sorted 2D Matrix - Problem - 13

Given m*n 2D matrix. program to find the element in the given Matrix.

① Integers in each row are sorted from left to right.
② first integer of each row is greater than last integer of previous row.

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ 10 & 11 & 16 & 20 \\ 23 & 30 & 34 & 60 \end{bmatrix}$$, target = 30 , O-P - true.

Naive Approach - We traverse the matrix using 2 loop and check that the element is exist in the matrix or not.

Time complexity - O(M×n)
Space complexity - O(1)

Optimal Solution - Binary search

	0	1	2	3
0	1 (0)	3 (1)	5 (2)	7 (3)
1	10 (4)	11 (5)	16 (6)	20 (7)
2	23 (8)	30 (9)	34 (10)	60 (11)

n = 3
m = 4

target = 30

0 - (n×m-1)
0 - 11 → indexes.

$\frac{0+11}{2} = 5$

low = 0, mid = (5) index, high = 11

because there are 4 elements in the row.

5/4 = 1 means corresponds to (1,1) which is 11
5%4 = 1 but 11 < 30

So low becomes 6 and high will be similar as before.

low = 6, mid = $\frac{6+11}{2}$ = (8) index, high = 11

8/4 = 2
8%4 = 0

It correspond to (2,0) which is 23 and 23 < 30

low = mid + 1

Now low = 9 , high = 11

Time complexity = $\log_2(m \times n)$

Space complexity = O(1)

9 (mid) (10) (high) 11

low index.

10/4 = 2

10%.4 = 2

(2,2) stands for 34 and here we can se 34 > 30 so

high = mid - 1

= 9

9 ← mid

low high

so 9/4 = 2

9%4 = 1

so (2,1) belongs to 30 matched.

We can also apply the binary search in each row. and checks for the elements -

Time complexity = N $\log_2(m)$

If the first element of each row is not greater than the previous row last element. : If this condition is not given

10	20	30	(40)
11	21	36	43
25	29	39	50
50	60	70	80

target = 25

① firstly place the pointer on the last element of the first row every thing is sorted. and on the left of 40 every element is smaller than 40 and at the bottom every thing is bigger than 40.

25 < 40 (so it correspond to left side)

so move the pointer towards left.

10	20	(30)	40
11	21	36	43
25	29	39	50
50	60	70	80

Now again look for 25.

again 30>25 so pointer moves to left

10	20	30	40
11	21	36	43
25	29	39	50
50	60	70	80

Now 20<25 so pointer moves to the Bottom and similarly the pointer will reached at the destination.

If elements not found: so your pointer jump out of the boundary. we can return -1.

```
int i=0, j=m-1
while (i<n && j>=0) {
    if (mat[i][j]==x) {
            return true;
                }
if (mat[i][j]>x)
        j--;
else
    i++;
        }
```

Time complexity - O(log N×m)

Leetcode: Code -

```
if (mat.length==0) return false;
int n = mat.length;
int m = mat[0].length;
int low=0;
int high = (n*m-1);
while (low<=high){
  int mid = low + (high-low)/2;
  if (mat[mid/m][mid%m]==target){
          return true;
              }
  if (mat[mid/m][mid%m]<target){
          low = mid+1;
              }
  else {
        high = mid-1;
      }
}
return false.
```

Implement Pow(x,n) - Problem No- 14

x = 2.0000 , n = 10

Output = 1024.0000

Bruteforce Approach - Looping from 1 to N and keep a variable (double) Now every time loop runs, multiply with x. and in last return ans.

Time complexity - O(N)

Space complexity - O(1)

Optimal Solution -

$2^{10} = (2 \times 2)^5 = (2^2)^5 = (4)^5 \Rightarrow 1024$

$4^5 = 4 \times (4)^4 \; 256$

$4^4 = (4 \times 4)^2 = (16)^2 = 256$

$16^2 = (16 \times 16)^1 = (256)^1$

$256^1 = 256 \times (256)^0$

① [n%2 == 0] → [x*x] (n/2) Timecomplexity - $\log_2(N)$

② [n%2 == 1] → ans = ans * x
n = n-1

③ n == 0 stop

```
double ans = 1.0;
long nn = n;
if (nn<0) nn = -1 * nn;
while (nn>0) {
if (nn%2==1){ans = ans * x;
            nn = nn-1;
          }
else { x = x*x;
       nn= nn/2;
     }
}
if (n<0) ans = (double)(1.0)/(double)(ans);
return ans;
```

Majority Element that occures more than N/2 times :

N = 3 , nums [] = [3, 2, 3] N = 7 nums = [2, 2, 1, 1, 1, 2, 2]

result = 3 result = 2

Bruteforce Approach -

① We run a loop that select a element one by one.

② Now each element run another loop and count the occurance in the array.

③ If any element occures $N/2$ times then simply return.

```
for (i = 0 → n) { int cnt = 0;
    for (j = 0 → n) {
        if (arr[i] == arr[j]){ cnt++;
                              }
    }
if (cnt > (n/2))
        return arr[i];
}
```

Time complexity = $O(N^2)$

Space complexity = $O(1)$

Better Approach - Use a better data structure to reduce the no of lookup operation.

Use HashMap and (Key , value) pair.

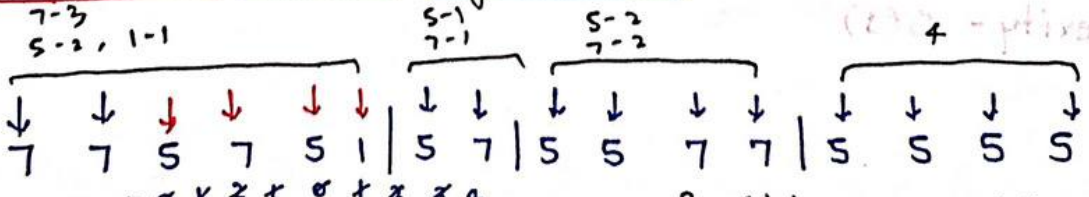
Time complexity - Insertion for one element take $\log N$. & for N element it take $N \log N$.

and for find or traversing the Hashmap for checking majority element take $O(N)$

Time complexity - $\boxed{O(N) + N \log N}$

Space - $O(N)$

Optimal Approach : Moore's Voting Approach -




```
for (int num : nums){
    if (cnt == 0) ele = num;
    if (num == ele) cnt++;
    else cnt--;
}
```

Majority Element : Problem - 16

```
majority Element (int[] arr){
    int n= arr.length;
    int cnt1=0, cnt2=0;
    int ele1 = Integer.MIN;
    int ele2 = Integer.MIN;
   for(i=0; i<n:i++){
            if (cnt1==0 && ele2!= nums[i]){
                    cnt1=1;
                    ele1 = nums[i];
                                    }
            else if (cnt2==0 && ele1 != nums[i]){
                    cnt2 = 1;
                    ele2 = nums[i];
                                    }
            else if ( ele1 = nums[i]) cnt1++;
            else if (ele2 = nums(i)) cnt2++;
            else { cnt1--;
                   cnt2--;
                 }
        }
List < Integer> lt = new ArrayList <>();
cnt1=0, cnt2=0;
for (i=0; i<n; i++){
            if (nums[i] == ele1) cnt1++;
            if (nums[i] == ele2) cnt2++;
                    }
int min = (int) (n/3) +1;
if (cnt1 >= min) lt.add (ele1);
if (cnt2 >= min) lt.add (ele2);
```

we checks ele2 != nums[i] because it can be possible that if we want to put the val of nums[i] in el2 which is already taken by ele1.

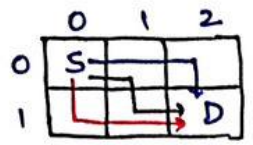
// manually checking the majority element.

Timp complexity - O(N) + O(N)

Space complexity - O(1)

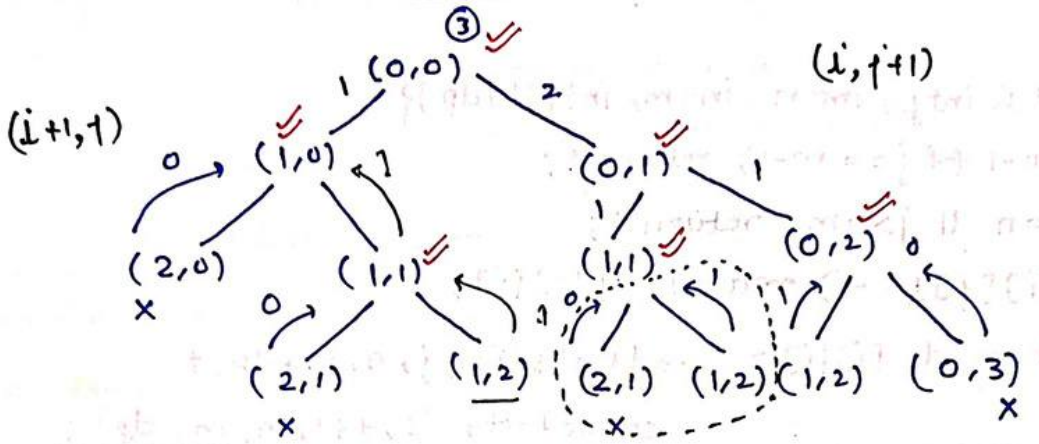
Unique Paths : Problem 17

grid of m*n is given. and robot is initially located on (0,0) and we have to find all the possible unique paths to reach (m-1, n-1). the robot can be move only right and bottom.



Total 3 paths.

Bruteforce Solution · Recursion



i>=n
j>=m

Base case → 0

i == n-1
j == m-1
} return 1

```
Pnt countpath (int i, int j, int n, int m){
        if (i == (n-1) && j == (m-1)) return 1;
        if (i>=n || j>=m) return 0;
        else return countpaths (i+1, j) + countPath (i, j+1);
                                                    }
```

Hashtable : Optimal Solution.

DP

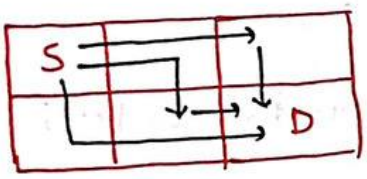
	0	1	2
0	-1	-1	-1
1	-1	-1	-1

if the value of a recursion call is available in hash table so we take the value from it and we don't need to call recursion.

```
public int uniquepath (int m, int n) {
    int dp[] = new int [m] [n];
    for (int[] row : dp) {
        Arrays.fill (row, -1);
                    }
    int num = countPaths (0, 0, m, n, dp);
    if (m == 1 || n == 1)
        return num;
    return dp [0] [0] ;
        }
countPaths (int i, int j, int n, int m, int[][] dp ){
    if (i == n-1 || j == m-1) return 1;
    if (i >= n || j >= m) return 0;
    if ( dp[i][j] != -1) return dp[i][j];
    else return dp [i][j] = countPaths (i+1, j, n, m, dp)+
                            countPaths (i, j+1, n, m, dp) ;
```

Time complexity - O(n*m)
Space complexity - O(n*m)

We can also solve it using combination.




```
[ R R D
  D R R
  R D R ]
```

Observation 1 - for reaching at destination we have to traverse total no of 2 in rows and 1 in the bottom means.

$(\overline{n-1} + \overline{n-1}) = (m+n-2)$ for sure
$= 3$ step

$^{3}C_{2}$

```
[ R R B
  R B R
  B R R ]
```

$^{m+n-2}C_{m-1} = 3$

$^{m+n-2}C_{n-1} = 3$

```
int N = n+m-2;
int r = m-1
double res = 1;
for (i=1 ; i<=r; i++)
  res = res * (N-r+i)/i;
return (int) res;
```

O(m-1) or O(n-1) Time complexity.

Reverse Pairs - Problem - 18

arr[] = [40, 25, 19, 12, 9, 6, 2]

Conditions: $i<j$ ff $a[i] > 2*a[j]$

O/P - 6

+	(6, 2)
3	(9, 2)
+	(12, 2)
3	(19, 2)
+	(25, 2)
2	(40, 2)
+	⋮
11	
15	

Similar to count Inversion

Slight modification

↓

before merging we just need to count the pair which follows the condition.

cnt += countPairs (arr, low, mid, high);

This function countPairs is called in the mergeSort.

then creat a function countPairs and cnt the pairs.

```
fun countPairs (int arr[], int low, int mid, int high) {
        int righ = mid+1;
        int cnt = 0;
        for(int i = low; i<= mid; i++){
            while (right <= high ff arr[i] > 2 * arr[right])
                                    right++;
            cnt += (right - (mid+1));
                                    }
```

add the cnt all the pair that is exist on the left side of right

Time complexity - $\log n + O(N) + 2(n) \Rightarrow 2n \log N$

for devide

Space complexity - $O(N)$.

for distorting the array.

Two Sum: Check if the Given Pair sum exists in Array -

N = 5, arr[] = [2, 6, 5, 0, 1], target - 14

OP - Yes on adding 6+0 we get 14.

If Not found simply return - 1.

Brute force - ① take i = 0 to n-1 and j = i+1 to n
check if (arr[i] + arr[j] == sum) Time complexity $O(N^2)$.

Better approach : Using Hashing - Use a Hashmap.

We will use a hashMap for check the other element target - current element exist in the hash map or not.

```
TwoSum (n, arr, target) {
    HashMap<int, int> ();
    for (i=0→n) {
        int num = arr[i];
        if (map.containsKey (target - num) {
                return 'Yes';
                }
        map.put (arr[i], i);
        }
    return No;
    }
```

Time complexity - O(N).
Space complexity - O(N).

Optimal Approach : Two Pointer Approach:

① Sort the array.
② take i = 0 and j = n-1 check arr[i] + arr[j] > target then j-- other wise i++.

if we get the target simply return

Time complexity - O(N) + n log N
Space complexity - O(1).

```
twoSum(arr, n, target) {
    Sort (arr);
int i=0, j= n-1;
while (i<j) { int sum = arr[i] + arr[j];
if (sum == target) return 'yes';
else if (sum < target) ~~return~~ left++;
else right--;
    }
```

4 Sum Problem: Problem-20

arr[a] + arr[b] + arr[c] + arr[d] == target.

Bruteforce -

① Run 1st loop from 0 to n-1
② Run 2nd loop from 1 to n-1
③ Run 3rd loop from 2 to n-1
④ Run 4th loop from 3 to n-1

Check for arr[i] + arr[j] + arr[k] + arr[l] == target or not.

Time complexity - $O(N^4)$
Space complexity - $O(N)$.

Better Approach - Using 3 loop and a DS.

① first loop that will run from 0 to n-1
② Inside it j = i+1 to n-1
③ Before third loop declear a hashset to store the specific array elements as we intend to search the four element.
④ Then the third loop k = j+1 to n-1
⑤ Inside this calculate the value of target - (arr[i] + arr[j] + arr[k])
⑥ If the value that we calculate so it is present in Hash Set so sort the value and store in the DS.

```
fun fourSum (arr, nums){
    int n = arr.length;
    Set< List < Integer>> ans = new HashSet<>();
    for (i=0; i<n ; i++) {
        for (j= i+1; j<n; j++){
        Set<Long> st = new HashSet <>();
        for (k= j+1 ; k<n; k++) {
            long sum = num[i] + num[j];
            sum + = nums [k];
            long four = target-sum;
            if (st. contains (four)){
                List <Integer> temp ();
```

```
temp.add(arr[i]);
temp.add(arr[j]);
temp.add(arr[k]);
temp.add((int) four);
temp.sort(Integer::compareTo);
ans.add(temp);
}
/hash/ st.add((long) nums[k]);
}
}
}
List<List<Integer>> lt= new ArrayList<>();
lt.add(one);
return lt;
}
```

Time complexity - $O(N^3 \cdot \log(M))$

M = no of elements in set.

Space complexity - O(2 * no of quadruplets) + O(N)

Optimal Approach - We have to remove hashset which is used to lookup in the array and set ds which is used to store the unique quads.

① Sort the array because we don't want duplicates.

target = 0

arr[] = [1 1 1 2 2 2 3 3 3 4 4 4 5 5]

i j k k l l duplication.

fix

[1,1,1,5]

[1,1,2,4]

[1,1,3,3] just ignore duplicate by comparing the previous

[1 1 1 2 2 2 3 3 3 4 4 4 5 5]

i j k l

[1 1 1 2 2 2 3 3 3 4 4 4 5 5]

i j k l

```
List<List<Integer>> fourSum(arr, target){
    int n = arr.length;
    List<List<Integer>> ans();
    Arrays.sort(arr);
    for(i=0; i<n; i++){
        if(i>0 && a[i]==arr[i-1]) continue;
    for(j=i+1; j<n; j++){
        if(j>i+1 && arr[j]==arr[j-1]) continue;
        int k = j+1;
        int l = n-1;
       while(k<l){
            long sum = arr[i];
            sum += nums[j]+nums[j]+nums[k]; // arr used.
            if(sum==target){
              List<Integer> temp= new ArrayList<>();
              temp.add(arr[i]);
              temp.add(arr[j]);
              temp.add(arr[k)];
              temp.add(arr[l]);
              ans.add(temp);
              k++;
              l--;
              // Here we Need to skip duplicates.
    while(k<l && nums[k]==nums[k-1]) k++;
    while(k<l && nums[l]==nums[l+1] l--;
                }
            else if(sum<target) k++;
            else l--;
           }
       }
```

Time complexity - $O(N^2) + O(N) = O(N^3)$

Space complexity - O(No of quatos).

Longest Consecutive Sequence in an Array: Problem-21

nums[] = [100, 4, 200, 1, 2, 3] , o/p = 4

① length of longest Consecutive sequence.

Bruteforce · Sorting

```
longestConsecutive (arr){
    if (arr.length == 0) return 0;
    Arrays.sort (arr);
    int ans = 1;
    int curr = 1;
    prev = nums[0];
    for (i = 1 → n){
        if (nums[i] == prev+1) cnt++;
        else if (nums[i] != prev){
                curr = 1;
                }
        pre = nums[i];
        ans = Math.max (ans, cnt);
            }
```

Time complexity - nlogn + O(N)
Space complexity - O(1)

Optimal Approach: Using HashSet

① put every thing into the set.

```
fun. longestConsecutive (arr){
    if (arr.length == 0) return 0;
    HashSet<Integer> st();
    for (int num : arr){
        st.add(num);
        }
    for (int num ; arr){
        if (!st.contains(num-1)){
            int currentNum = num;
            int currentStreak = 1;
            while (st.contains (currentNum+1))
                current += 1;
                currentStreak += 1;
            longestStreak = Math.max (longestStreak, curStreak)
        }
    }
```

longestStreak = 0;

Time complexity - O(N) ≈ O(3N)
Space complexity - O(N)

Longest subarray with 0 sum - Problem - 22

```
int solve (arr){
    max = 0;
  for (i=0; i<arr.length; i++)
    {
      int sum = 0;
      for(j= i ; j<arr.length; j++){
        sum + = arr[j];
        if (sum == 0)
              max = Math.max (max, j-i+1);

    }
  return max;
```

Brute force

Time complexity - $O(N^2)$

Space complexity - $O(1)$

Optimal

N = 8 , A[] = [15, -2, 2, -8, 1, 7, 10, 23]

take a variable sum and initialize it with 0.

add every element to the sum and put into the hashmap.

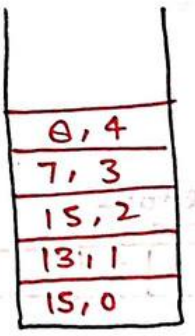
and check the element exist in the hashmap or not

A[] = 15 , -2 , 2 , -8 , 1 , 7 , 10 , 23

sum = 0 + 15 - 2 = 13 + 2 = 15 ~~✗~~ 15

length = current index - the element in the hashmap

= 2 - 0 = 2 = 5 - 0 = 5




```
HashMap <int> (); int i= -1 , sum = 0;
map.put (sum, -i);
while (i<n-1) {  i++;   // because i = -1 so we using i<n-1
        sum = sum + arr[i];
        if (map. containsKey (sum) == false) map.put (sum, i);
        else {
    int len = i-map.get (sum);
        max = Math.max (max, len);
          }
      }
return max;
```

Time complexity - $O(N)$

Space complexity - $O(N)$.

Count subarray with XOR as K · Problem-24

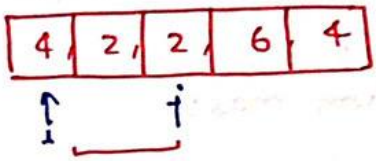
arr[] = [4, 2, 2, 6, 4] K = 6

{4, 2}
{6}
{2, 2, 6}
{4, 2, 2, 6, 4}

ans = 4

// → Similar element cut each other in ^ (XOR) -

i j → the subarray is from i to j

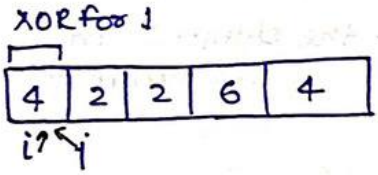


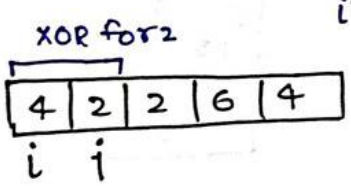
Bruteforce -

```
int cnt=0;
for(i=0; i<n;i++){
  for(j=i; j<n;j++){
        int XOR=0
    for(k=i; k<=j; k++){
        XOR = XOR ^ a[k];
                }
    if(xorr==k) cnt++;
  }
}
cnt;
```

Time complexity - ≈O(N³)
Space complexity - O(1)

Better Solution -

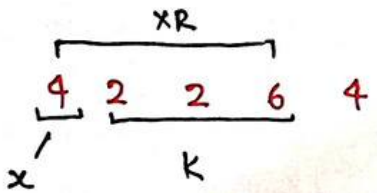





```
for(i=0; i<n; i++)
        XOR=0
for(j=i; j<n;j++){
    XOR = XOR ^ arr[j];
    if(XOR==k) cnt++;
                }
```

Timecomplexity - O(N²)
Space complexity - O(1)

Optimal Solution : Using Hashing

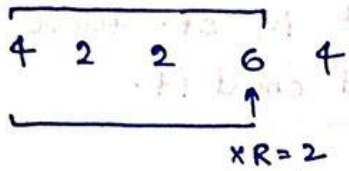


$XR = x \wedge k$

take the $\wedge k$ in both side.

$XR \wedge k = (x \wedge k) \wedge k$

$\boxed{x = XR \wedge k}$ k is the XOR we are looking for



So there a subarray ending at 6 of having the XOR of k

$x = XR \wedge k$

$= 2 \wedge 6 = 4$

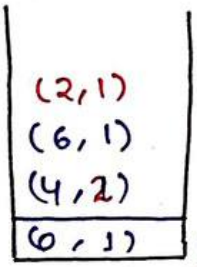
do we have from some who is giving me a 4

arr[] = [4 2 2 6 4]

XOR = 4 ∧ 2 = 6 ∧ 2 = 4 ∧ 6 = 2 ∧ 4 = 6

int x = XOR ∧ k = 4 ∧ 6 = 2 (we don't have 2 in Hashmap)

cnt = 1 + 2 + 1 = 4



put initially.

Hashmap.

$x = 6 \wedge 6 = 0$ which is available in the Hp.

$x = 4 \wedge 6$

$= 2$ (No)

$x = 2 \wedge 6$

$= 4$

$x = 6 \wedge 6$

$= 0$

```
XR = 0;
Hashmap<> ();
map.put (xr, 1);
int cnt = 0;
for (i=0; i<n; i++) {
        XR = XR ∧ a[i];
        int x = XR ∧ k;
        if (map.contains (x)) cnt += map.get(x);
        if (map.containsKey(XR)) map.put (XR, map.get (XR)+1));
        else map.put (XR, 1);
    }
```

Time complexity - O(N)

Space complexity - O(N)

Longest Substring without repeat : Problem - 24

S = a b c a b c b b O/P = 3

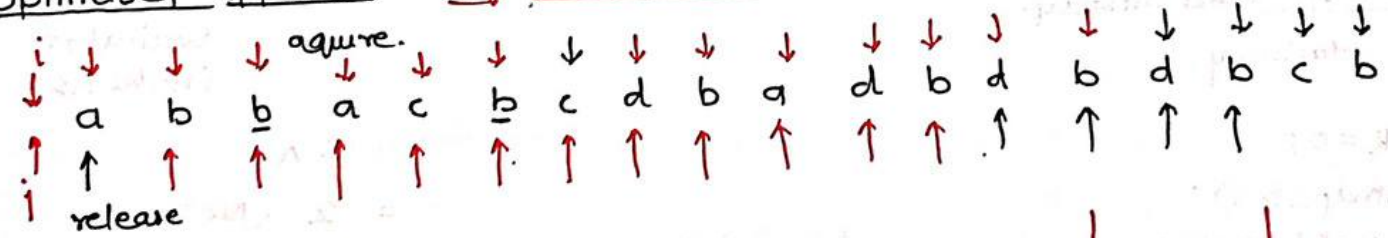
S = b b b b O/P = 1

Bruteforce Approach - Use two loops - one for traversing string and another nested loop for finding different substring. We check all substring one by one and check for each and every element if the element is not present in st. then put it into st otherwise break the loop and count it.

```
int max = /MAX/ MIN;
for (i=0; i<n; i++) {
    Set<Character> st();
    for (j = i ; j<n ; j++) {
        char ch = str.charAt(j);
        if (st.contains(ch)) {
            max = Math.max(max, j-i);
        }
        st.add(ch);
    }
}
```

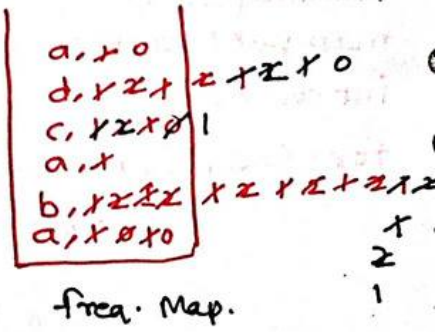
Time complexity - $O(N^2)$
Space complexity - $O(N)$

Optimised Approach - Aquire and release



~~a~~ ~~ab~~ ~~bc~~ cdba

[aquire until you became invalid.
release until you valid again]



freq. Map.

```
fun. Solution (String str) {
int ans = 0;   boolean f1 = false , f2 = false;
int i = -1;
int j = -1;
HashMap <int, int> map ();
while (true) {
      //aquire
      while (i< str.length ()-1) {
            i++;  f1 = true;
            char ch = str.charAt (i);
            map.put (ch, map.getOrDefault (ch,0)+1);
            if (map.get (ch) == 2) break;  // added a repeating ch
            else
                int len = i-j
                ans = Math.max (ans, len);
                                    }
      //release.
      while (j<i) {
            j++;  f2 = true;
            char ch = str.charAt (j);
            map.put (ch, map.get (ch) -1);
            if (map.get (ch) == 1)
                  break;                  // break because its valid
      // else {
            }  // }
      if (f1 == false && f2 == false)
                  break;
            }
```

Time complexity - O(N)

Space complexity - O(N).

Reverse a Linkedlist : Problem - 25

head = 3→6→8→10 , reverse = 10→8→6→3

head = [ ], reverse = [ ]

```
ListNode reverseList (ListNode head){
        ListNode curr = head;
        ListNode pre = null;
        while(curr != null) {
            ListNode next = curr.next;
            curr.next = pre;
            pre = curr;
            curr = next;
             }
        return pre;
```

Using Recursion.

```
ListNode reverseList (ListNode head) {
    if( head== null || head.next)
                    return head;
    ListNode curr = reverseList (head.next)
    hed.next.next = head;
    head.next = null;
    return curr;
    }
```

Middle element of linkedlist : Problem - 26

head = 1→2→3→4→5

mid = 3.

Brute force : Use a loop and count the element : check odd-even case then find element.

Efficient solution : Tortoise - Hare - Solution

Take two point slow and fast and move the pointer while (f != null && f.next != null) then the s pointer automatically move to the mid position.

```
ListNode middleNode(ListNode head){
    ListNode slow = head, fast = head;
  while(f.next!=null && f!=null){
            s=s.next;
        f = f.next.next;
              }
    return slow.
```

Time complexity - O(N)

Merge Two Sorted Linkedlist - Problem - 27

$l_1 = 3 \to 7 \to 10$, $l_2 = 1 \to 2 \to 5 \to 8 \to 10$

O/P = $1 \to 2 \to 3 \to 5 \to 7 \to 8 \to 10 \to 10$

Use a dummy Node and traverse both Linkedlist and compare the values which one is smaller link the dummy Node with the value.

```
ListNode temp = new ListNode(0);
ListNode tail = temp;
while (true) {
    if (head1 = null) curr.next = head2;
                      break;
    if (head2 = null) curr.next = head1;
                      break;
    if (head1.val >= head2.val){
                  curr.next = head2;
                  head2 = head2.next;
                              }
    else {
          curr.next = head1;
          head1 = head1.next;
        }
          }
    return temp.next;
```

Time complexity = O(N+M)

Problem - 28 : Remove N^{th} from end of Linkedlist -

```
removeNth from ( ListNode head, int n) {
        ListNode start = new ListNode (0);
        star.next = head;
        ListNode f = head; → staat;
        List Node s = head; → start;

        for (i = 1 ; i<=n; i++)
                        f = f.next;
        while (f.next != null) {
                fast = fast.next;
                slow = slow.next;
                        }
        slow.next = slow.next.next;
        return start.next;
```

Time complexity - (ON)
Space complexity - O(1)

Delete a given Node : Problem 30.

[1 → 4 → 2 → 3] , Node 2.

O/P = 1 → 4 → 3

copy the next Node data of the current Node and connect the current Node with the current.next.next.

```
deleteNode (Node t) {
        if (t == null) return;
        t.next = t.next.val;
        t.next = t.next.next;
                }
```

Time complexity - O(1)
Space complexity - O(1)

Add two Numbers represented by a linkedlist : Problem 29.

num 1 = [2, 4, 3]
num 2 = [5, 6, 4]

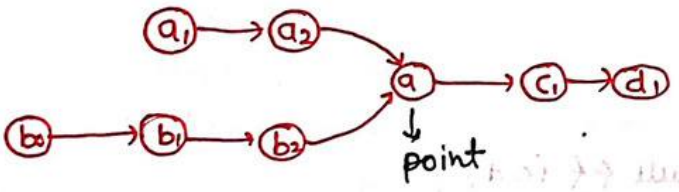
sum = [8, 0, 7]

Case: 1. When one list is longer than other.
Case: 2 When one list is null:
Case: 3 the sum could have an extra /query/ carry of one at the end.

```
ListNode dummy = new Node (0):
ListNode temp = dummy:
Int carry=0;
while (l1 = null || l2 != null || carry == 1) {
        int sum = 0;
        if (l1 != null) sum + = l1.val;
                        l1 = l1.next;
        if (l2 != null) sum + = l2.val;
                        l2 = l2.val;
        sum + = carry:
        carry = sum / 10;
        ListNode node = new ListNode (sum %. 10);
        temp.next = node;
        temp = temp.next;
        }
```

Time complexity - max (O, N, M)
Space - max (n, m) + 1.

Intersection of two Linkedlist : Problem - 31



Bruteforce - A common attribute is given so.

① keep one linkedlist to check the node is present in the other list. Here we choose first.
② Iterate through the other list.
③ Check if both nodes same then ok.

Time complexity - $O(N^2)$

```
Intersection (head1, head2) {
while (head1 != null) {
        Node temp = head1;
        while (head2 != null){
        if (head2 == temp) return Yes;
        temp = temp.next;
        }
head2 = head2.next;
}
```

We can also solve this using Hashing: O(N+M) → Time complexity - Space complexity - O(N)

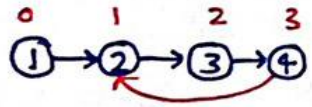
Difference in length:
① Find len. of both list
② Find the positive diff.
③ Move the dummy pointer to diff. archeive.
④ Move both pointer.

```
getInseation (headA, headB) {
    Node curr1 = headA;
    Node curr2 = headB:
    int currcnt1 = 0;
    int cnt2 = 0;
    int d = 0;
    while (curr1 != null) cnt1++;
                          curr1 = curr1.next;
    while (curr2 != null) cnt++;
                          curr2 = curr2.next;
    if (cnt1 > cnt2) {
                d = cnt1 - cnt2;
                int i = 0;
                while (head1 != null && i<d){
                        i++;
                    head1 = head1.next;
                                        }
                }
    if (cnt > cnt2) {
                 d = cnt2 - cnt1;
                 int i = 0;
                 while (head2 != null && i<d) {
                   i++;
                   head2 = head2.next;
                 }                          }
    while (head1 != head2) {
                 head1 = head1.next;
                 head2 = head2.next;
                    }
    return head1;
```

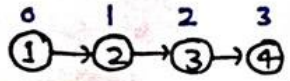
Time complexity - $O(2\max(l_1, l_2)) + O(\text{abs}(l_1 - l_2)) + O(\min(l_1, l_2))$

Space - O(1)

Detect a Cycle in Linkedlist. Problem no - 32



Output - true



output - false

Solution : Hashing

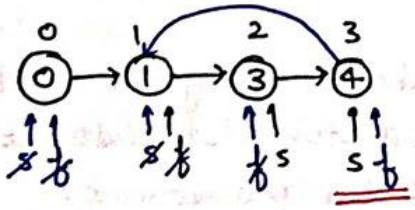
1. Use a hashtable for storing nodes.
2. Start iterating through the list.
3. If the current node is present in the hashtable already. this indicate the cycle is present.
4. else move insert.

```
HashSet<Node> st = new HashSet<>();
while (head != null) {
        if (st.contains (head)) return true;
        st.add (head);
        head = head.next;
            }
return false;
```

Time complexity - O(N)
Space complexity - O(N).

Optimal : Slow and fast pointer.

```
if (head == null) return false;
Node f = head;
Node s = head;
while (f.next != null && f.next.next != null)
        f = f.next.next;
        s = s.next;
        if (s == f) return true;
return false;
```



Reverse linkedlist in a group of K. - Problem 33

head = ① → ② → ③ → ④ → ⑤ → ⑥ → ⑦ → ⑧

K = 3.

Output - ③ → ② → ① → ⑥ → ⑤ → ④ → ⑦ → ⑧

Size less than K so no change.

① Create a dummy node. Point next to this node to head of linkedlist.
② Get the length of linkedlist.
③ Reverse k.
④ check if (pre.next != null) then
call the recursion and pass (next, k).

```
ListNode curr = head;
ListNode next = head;
ListNode pre = null;
int cnt = 0;
int s = 0; , ListNode temp = head;
while (temp != null) {
    temp = temp.next;
    s++;
}
if (s<k) return head;
while (curr != null && cnt < k)
{
    next = curr.next;
    curr.next = pre;
    pre = curr;
    curr = next;
}
if (next != null) {
    ListNode rest = reverseKGroup(next, k);
    head.next = rest;
}
return pre;
```

Time complexity. O(n)

Check if a Linkedlist is palindrome or not. Problem 34.

① → ② → ② → ①

O/P - Yes.

Bruteforce. Take a array list and traverse the linkedlist and store data into it and traverse it and check for palindrome.

Time complexity - O(N).
Space complexity - O(N).

Optimal Solution -

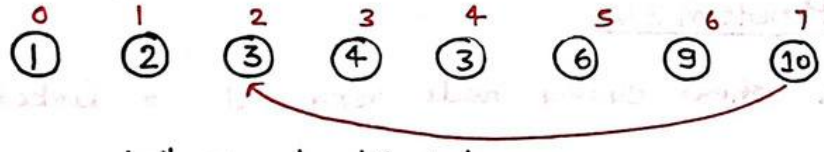
① find the mid element
② Reverse the linkedlist after pre.
③ Compare and check

```
isPalindrome (Node head){
    if (head == null || head.next == null) return true;
    Node slow = head;
    Node fast = head;
    while (f != null && f.next != null)
        s = s.next;
        f = f.next.next;
    slow.next = reverse (slow.next); // write the code for reverse
    slow = slow.next;
    Node dummy = head;
    while (slow != null)
        if (dum.val != slow.val) return false;
        dummy = dummy.next;
        slow = slow.next;
    return true;
```

Time complexity - O(N)
Space complexity - O(1)

Linkedlist cycle II - Problem 35.

1.



tail connect with index 2.

Bruteforce - Using HashSet.

① Iterate the list
② for each Node visited by head pointer, check if the node is present in the hash table.
③ If loop detect yes.

```
HashSet <int> st ();
while(head != null){
        if (st.contains(head)) return head;
        st.add (head);
        head = head.next;
        }
```

Optimal Solution.

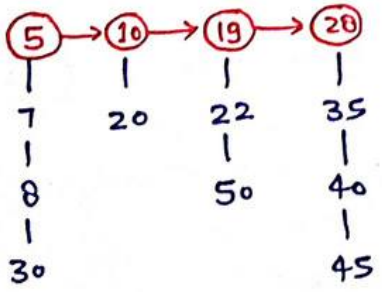
① Check for head or hed.next null. condition.
② find the mid (s==f) condition if it is then break.
③ if not then return null;
④ move the s to head and Now run a loop while (s!=f) and return s.

```
ListNode s = head:
List Node f = head;
while (f!= null ff f.next!= null){
        s=s.next;                          Time complexity - O(N)
        f=f.next: next;
        if (s==f) break;
        }
if (s!=f) return null;
if (head==null || head.next==null) return null;
s=head;
while(s!=f){
        s=s.next;
        f=f.next;
          }
return s;
```

Flatenning a linkedlist. Problem 36

Given a linkedlist of size n where every node represents a linkedlist and contains two pointer.

① A next pointer to the next node.
② a bottom pointer to a linkedlist where this node is head.

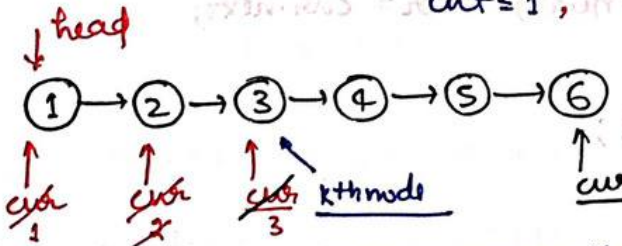


Output - 5→7→8→10→19→20→22→28→30→35→40→45→50.

```
Node mergeSorted (Node a, Node b) {
    Node temp = new Node (0);
    Node res = temp;
    while (a != null && b != null) {
      if (a.data < b.data) {
            temp.bottom = a;
            temp = temp.bottom;
            a = a.bottom;
                        }
      else {
            temp.bottom = b;
            temp = temp.bottom;
            b = b.bottom;
          }
    }
if (a != null) temp.bottom = a;   // If elements of a is left then add all
else                                 to temp
    temp.bottom = b;
return res.bottom;
                        }
Node flatten (Node root) {
        if (root == null || root.next == null) return root;
        root.next = flatten (root.next);
        root = mergeSorted (root, root.next);
        return root;
        }
```

Rotatelist Problem - 37

cnt = 1, while (cnt < k && cur != null)



Now move the cur to cur.next != null.

connect the cur.next = head. and head = kthnode.next and make kthnode.next = null.

Brute force Approach -

① We have to move the last element to first for each K.

② for each K find the last element of list move it to first.

```
fun Rotate { head, k) {
    if (head == null || head.next == null) return head;
    for (i=0 ; i<k; i++) {
      Node temp = head;
      while (temp.next.next != null ) temp = temp.next.
      Node end = temp.next;
      temp.next = null;
      end.next = head ;
      head = end;
    }
return head;
```

Time complexity - O (No of nodes * k)

Optimal Solution -

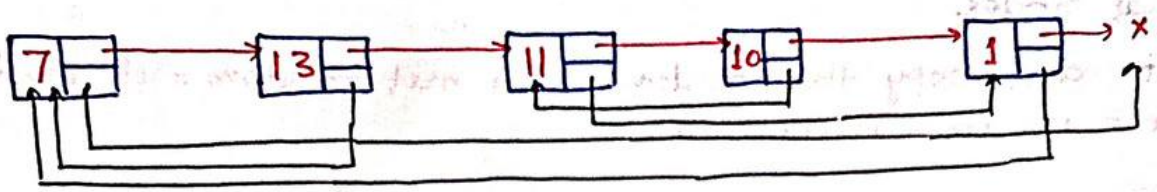
```
if (head == null || head.next == null || k == 0)
                                    return head;

Node cur = head;
int cnt = 1 ;
while (cnt < k && cur != null)
            cnt++;
            cur = cur.next;

if (cur == null)
            return head;

Node kthnode = curr;
while (cur.next != null)  cur = cur.next;
cur.next = head;
head = kthnode.next;
kthnode.next = null;
return hedd;
```

Time complexity -

O(len of list) + O(length of list - (length of list %. k))

= O(N).

Clone a linkedlist with random and Next pointer : Problem 30



Bruteforce - HashMap

1. Iterate through entire list.
2. for each node create a deep copy of each node and hash it with it.
3. Now again iterate through the given list. for each node, link the deep node present as has value of the original node.

```
fun RandomList(head){
        HashMap<Node, Node>();
        Node temp= head;
        while(temp!= null){
            Node newNode = new Node(temp.val);
            map.put(temp, newNode);
            temp= temp.next;
                        }
        Node t = head;
        while(t!= null){
            Node node = map.get(t);
            node.next = (t.next!=null)? map.get(t.next) : null;
            node.random = (t.next!= null) ? hashmap.get(t.random) : null
            t = t.next;
                        }
        return map.get(head);
```

Time complexity - O(N) + O(N)

Space complexity - O(N)

Optimize Approach -

- Create deep nodes of all nodes. we point it to the next of original nodes
- Take the itr and copy the random itr.next.random = itr.rand.next
  move the itr = itr.next.next.

```
copyRandomList (head) {
    Node temp = head;
    while (temp != null){
        Node newNode = new Node (temp.val);
        newNode.next = temp.next;
        temp.next = newNode;
        temp = temp.next;next;
                    }
    Node itr = head;
    while (itr != null) {
        if (itr.random=null) {
            itr.next.random = itr.random.next;
        itr = itr.next;next;
                    }
    Node dum = new Node (0);
    itr = head;
    temp = dum;
    Node fast;
    while(itr != null){
        fast = itr.next.next
        temp.next = itr.next;
        itr.next = fast;
        temp = temp.next;
        itr = fast;
                }
return dum.next;
```

Trapping RainWater : Problem - 39.

arr[] = [4, 2, 0, 3, 2, 5] op= 9

* Observe that water stored at particular index is the minimum of maximum elevation

Bruteforce -

```
trap(arr){
    n = arr.length;
    int watertrap = 0;
   for(i=0; i<n; i++){
        int j= i;
        int leftMax = 0, rightMax= 0;
        while(j>=0){
           leftMax = Math.max(leftMax, arr[j]);
           j--;
                }
        j = i
       while (j<n){
          rightMax = Math.max(RightMax, arr[j]);
           j++;      }
       watertrap = watertrap + Math.min(leftMax, rightMax) - arr[i]
        }
   return watertrap;
```

Time complexity - O(N*M).

BetterSolution - Take 2 array prefix and suffix and precompute the left and right for each index beforehand. then formula -

min(prefix[i], suffix[i] - arr[i]) to computer water

```
trap(arr){
    n = arr.length;
    int prefix[];     // create
    int suffix[];     // create.
    prefix[0] = arr[0];
    suffix[n-1] = arr[n-1];
```

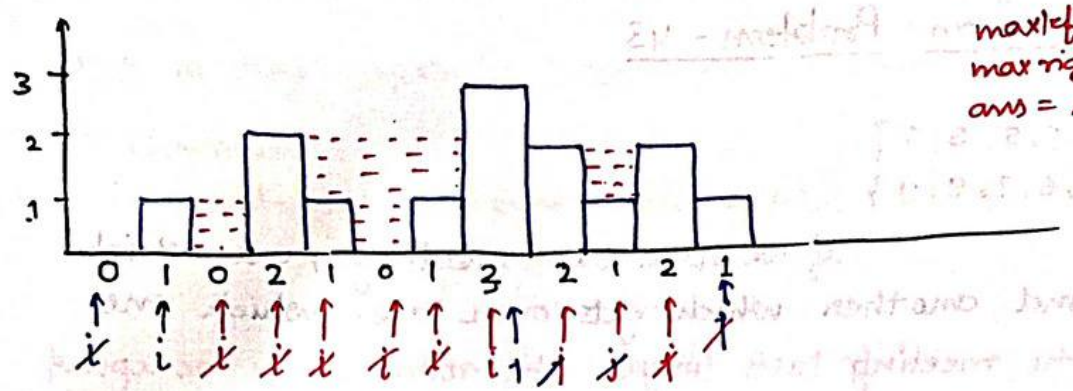
```
for (i=1 ; i<n; i++)
        prefix[i] = Math.max (prefix[i-1], arr[i]);

for (i= n-2 ; i>=0; i--) {
        suffix[i] = Math.max (suffix[i+1], arr[i]);

int watertrap = 0;
for (i=0; i<n; i++)
        watertrap += Math.min (pretix[i], suff[i]) - arr[i]);

return watertrap;
```

Time complexity - O(3 * N)
Space complexity - O(N) + O(N)

Two Pointer Approach : Optimized solution.

```
trap(arr) {
    n = arr.length;
    int left=0, int right = n-1 ; 0
    int res =0 , maxleft = 0, maxright= 0;
    while (left<=right) {
        if (height[left] <= height[right])
            if (height[left] >= maxleft)
                maxleft = height[left];
            else
                res += maxleft - height[left];
        left++;
        else{
            if (height[left] >= height maxright) {
                maxright = height[left]
            }
            else
                res+ = maxright - height[right];
        right--;
    return res;
```

Time complexity - O(N).



maxleft = 0 1 2
maxright = 0 1 2
ans = 0 1 2 4 5 6

Problem-41 : Remove Duplicates of sorted array.

- arr[] = [1, 1, 2, 2, 2, 3, 3]

↓ unique element to the front K=3

[1, 2, 3 _ _ _ _]

Bruteforce Approach.

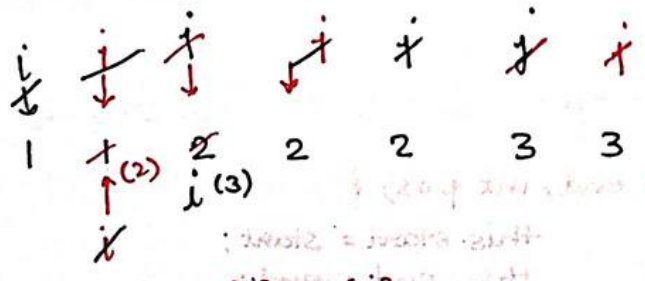
Time complexity - O(NlogN) + O(N)
Space complexity - O(N)

Used hashset for storing the values.

Optimal Approach.

```
remove duplicates (arr){
    i=0;
  for (int j=1 ; j< n ; j++){
     if (arr[i] != arr[j]){
          i++;
        arr[i]= arr[j];
     }
  }
  return i+1;
}
```

Time complexity - O(N)



N meetings in one room Problem - 43

N = 6, start = {1, 3, 0, 5, 8, 5}
end = {2, 4, 6, 7, 9, 9}

if we have two meetings, one which gets over early and another which gets over late. which one we choose? If the meeting lasts longer the room stays occupied and we loose our time.

On otherhand if we choose the meeting that finished earlier so we can arrange more meetings.

So given start and end time: sort this in ascending order of end time.

If the start time is greater than the end time then the meeting can be happen.

otherwise skip and move ahead.

	↓					
start	1	3	0	5	8	5
end	2	4	6	7	9	9

intially start will be 0 then first meeting can be happen

start = 1
end = 2

then check the next start which is 3

2 < 3 so meeting can happen
end updated and now 4.

3rd meeting can't happen.

end = 4, and start = 5 (true)

end = 7 (updated)

the end = 7 and next start = 8 (true)

```
class meeting {
        int start;
        int end;
        int pos;
    meeting { int start, int end, int pos) {
                this.stard = start;
                this.end = end;
                this.pos = pos;   } }
```

```
class meetingComparator implements Comparator<meeting>
{
    @override
        public int compare(meeting o1, meeting o2){
                if(o1.end < o2.end){
                    return -1;
                else if(o1.end > o2.end){
                        return 1;
                else if(o1.pos < o2.pos){
                        return -1;

                return 1;
    }       }

main(int start[], int end[], int n){
    ① ArrayList<meeting> lt = new ArrayList<>();
    ③ meeting comparator  mc = new meetingComparator

② for(i=0; i<n; i++){
            lt.add(new meeting (start[i], end[i], i+1));
                    }

    Collections.sort(lt, mc);
    ArrayList<Integer> ans();
    ans.add(lt.get(0).pos);
    int limit = lt.get(0).end;

    for(i=0; i<n; i++){
            if(~~mee~~ lt.get(i).start > limit){
                        limit = ~~mee~~ lt.get(i).end;
                        ans.add(meet.get(i).pos);
                                }

                }

    return ans.size();
```

Time complexity - $O(N) + O(N \log N)$

Space complexity - $O(N)$.

Minimum Platforms Problem - 44.

find the min no of platform req. for railway station so that no train kept waiting

N = 6

arr[] = {0900, 0940, 0950, 1100, 1500, 1800}

dep[] = {0910, 1200, 1120, 1130, 1900, 2000}

Sort all the starting and dep.

arr[] = {0900, 0940, 0950, 1100, 1500, 1800}

dep[] = {0910, 1120, 1130, 1200, 1900, 2000}

```
Array.sort(arr);
Array.sort(dep);
int platform = 1;
int res = 1;
int j = 0;
while (i<n && j<n){
        if (arr[i] ≤ dep[j])
                platform ++;
                i++;
        else if (arr[i] > dep[j]){
                platform --;
                j++;
                    }
        if (platform > res){
                res = platform;
                    }
return res;
```

[Maximum no of trains at particular time.]

Job Sequencing Problem - Problem - 45.

N = 4

Jobs = (1, 4, 20) (2, 1, 10) (3, 1, 40) (4, 1, 30)

Output = 2 60

Sort the jobs according to the profit.

perform the max deadline profit in the last because it has a big dead line try to finish other jobs so it maximize the profit.

so here the total profit = 60
and max job.deadline = 4.

Create an array of 4+1 and put -1 on it.

3			1	
~~-1~~	-1	-1	~~-1~~	-1
1	2	3	4	5

you can create size 4 array

id	dead - line	Profit
3	~~1~~	40
4	1	30 x
1	4	20 ✓
2	1	10

profit = 40 + 20 = 60

```
main (Job arr[], int n){
    Array.sort(arr, (a,b) → (b.profit - a.profit));
    int max = 0;
    for (i=0; i<n; i++){
        if (arr[i].deadline > maxi)
            maxi = arr[i].deadline;
    int res[] = new int [maxi+1];
    for (i=1; i<=maxi; i++){
        res[i] = -1;
    }
    int countJob = 0, int Profit = 0;
    for (i=0; i<n; i++){
        for (int j = arr[i].deadline; j>0; j--){
            if (res[j]==-1){ res[j] = i;
                countJobs++;
                JobProfit += arr[i].profit; break;
```

```
int ans = new int[2]
ans[0] = countJobs;
ans[1] = profit;
return ans;
```

Time complexity - $O(N \log N) + O(N*M)$

Space complexity - $O(M)$.

fractional Knapsack Problem - 46

N = 3 , W = 50, values[] = {100, 60, 120},
weight[] = {20, 10, 30}.

To achieve the max item item should sorted in decreasing order with respect to their value and weight first.

value/weight fo item 1 = $\frac{100}{20} = 5$

_ _ _ _ _ _ 2 = $\frac{60}{10} = 6$

\- - - - - - - 3 = $\frac{120}{30} = 4$

Item No.	1	2	3
value	60	100	120
weight	10	20	30

initially W = 50 , W ≥ 20
so we can take
value = 100 , then 50-20 = 30
30 ≥ 10
value = 160, then 30-10 = 20
20 ≥ 20 , then
value = 160 + $\left(\frac{120}{30}\right)$ x 20 then W = 0
= 240

```
fun main(W, arr, n){
Arrays.sort(arr, new itercomparator());
int currweight = 0;
double finalvalue = 0;0;
for(i=0; i<n; i++){
    if(currweight + arr[i].weight <= W){
      currweight += arr[i].weight;
      finalvalue += arr[i].value;
                                    }
    else{
        int remain = W - currweight;
        finalvalue += ((double)arr[i].value/ (double)arr[i].weight) * (double)
                                                                    remain;
        break;
       }
```

```
class iterComparator implements Comparator<Item> {
  public int compare(Item a, Item b){
    double r1 = (double)(a.value)/(double)(a.weight);
    double r2 = (double)(b.value)/(double)(b.weight);
    if (r1<r2) return 1;
    else if (r1>r2) return -1;
    else return 0;
  }
}
```

Find Minimum Number of coins - Problem - 47

We have infinite supply of {1, 2, 5, 10, 20, 50, 100, 500, 1000}
So min No of coins and or the notes needed to make the change

① V = 70
So we can take 50 and 20ss of notes, total - 2

② V = 121
we need a 100 Rs and 20 and 1, total - 3

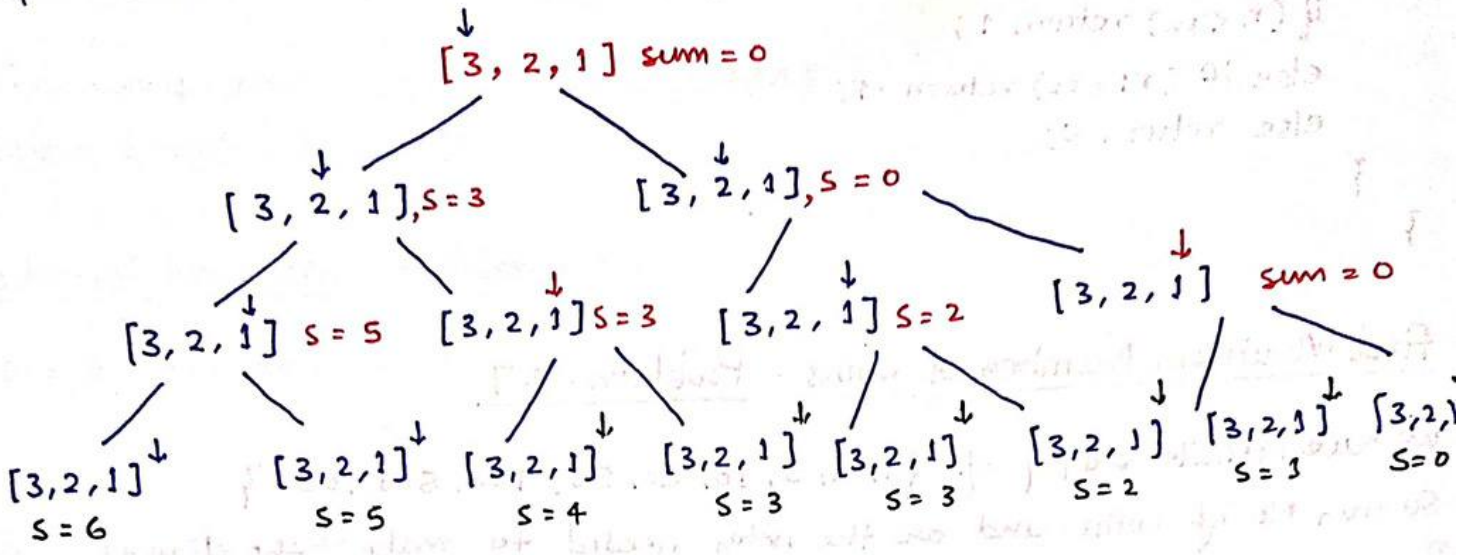
```
fun main(){
  ArrayList<> ans();
  int coins[] = [1, 2, 5, 10, 20, 50, 100, 500, 1000];
  int n = coins.length;
  for (i = n-1 ; i>=0; i--){
      while (V >= coins[i]){
          V = V - coins[i];
          ans.add(coins[i]);
                  }
              }
ans.size();
```

Time complexity - O(V)
Space complexity - O(1).

Subset Sum: Sum of all subsets. Problem No- 49

N = 3 , arr[] = {5, 2, 1}

Output - 0, 1, 2, 3, 5, 6, 7, 8



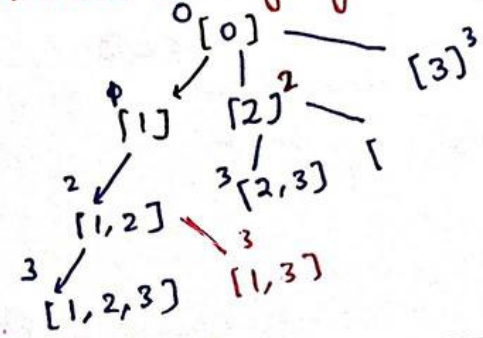
Powerset → Bruteforce

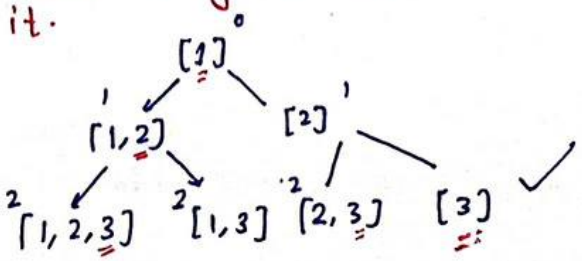
```
public static List<List<int>> generatePowSet(nums){
    List<List<Integer>> ans = new ArrayList<>();
    backtrack(nums, new ArrayList<>(), 0, sub ans);
    return ans;
    }

fun backtrack(nums, currentlt, index, ans){
    ans.add(new ArrayList<>(currentlt));
    for(i = index; i< nums.length; i++){
        currentlt.add(nums[i]);
        backtrack(nums, currentlt, i+1, ans);
        currentlt.remove(currentlt.size()-1);
                                    }
                                }
```

Time complexity - $2^n * N$

we are using a extra N so

No we are going to optimize it.





```
void helper(ind, int sum, arr, N, ans) {
        if (ind == N) ans.add(sum);
                      return;
Pick //     helper(ind+1, sum + arr.get(ind), arr, N, ans);
Not Pick // helper(ind+1, sum, arr, N, ans);
                                    }

static ArrayList<int> lt subset(arr, N){
        Arraylist <>() ans;
        helper(0, 0, arr, N, ans);
        Collections.sort(subsum) // ans;
        return ans;
                                    }
```

Timecomplexity - 2^n

Subset sum : Print all unique subsets.

arr[] = [1, 2, 2]

o/p = [[], [1], [2] [1,2] [1,2,2] [2,2]]

Bruteforce -

At every index we make a decision pick or not pick. this help us generating all the possible combination : use the set for unique sets.

```
main (){
    int arr[] = {1,2,2};
    List<String> ans (); = subsetwithDup (arr);
    print (ans);
      }

List<String> subsetwithDup (arr){
  List<String> ans;
  HashSet<String> res;
  List<Integer> ds;
  fun (arr, 0, ds, res);
  for (String it : res) {
                          ans.add(it);
                        }
  return ans; }
```

```
fun(int arr, ind, ds, res, arr){
    if(ind == arr.length){
        collections.sort(ds);
        res.add(ds.toString());
        return;
    }
    ds.add(arr[i]);
    fun(arr, ind+1, ds, res, arr);
    ds.remove(ds.size()-1);
    fun(arr, index+1, ds, res);
}
```

Time complexity - $O(2^n * (k \log(x)))$ · [2^n generating for every subset and $k \log x$ to insert every combination of avg length k in set of size x]

Space complexity - $O(2^n * k)$

Optimal Approach -

```
main(arr){
Arrays.sort(arr);
List<List<Integer>> ans();
find subsets(0, arr, new ArrayList<>(), ans);
return ans;
}

findsubsets(ind, arr, ds, ans){
    ans.add(new ArrayList<>(ds));
    for(i = ind; i < arr.length; i++){
        if(i != ind && nums[i] == nums[i-1]) continue;
        ds.add(arr[i]);
        findsubsets(i+1, arr, ds, ans);
        ds.remove(ds.size()-1);
    }
```

Combination Sum: 1 Problem - 51

① Given an array of distinct integers and target, return the list of all unique combinations.

arr[] = [2,3,6,7], target = 7

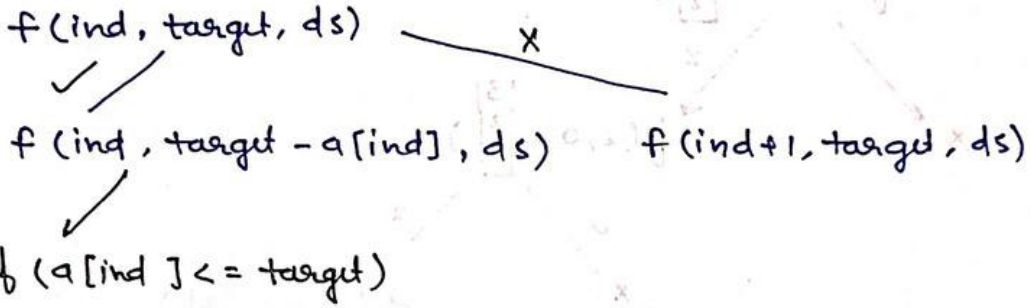
output = [[2,2,3], [7]]

↓

2 is used 2 times.

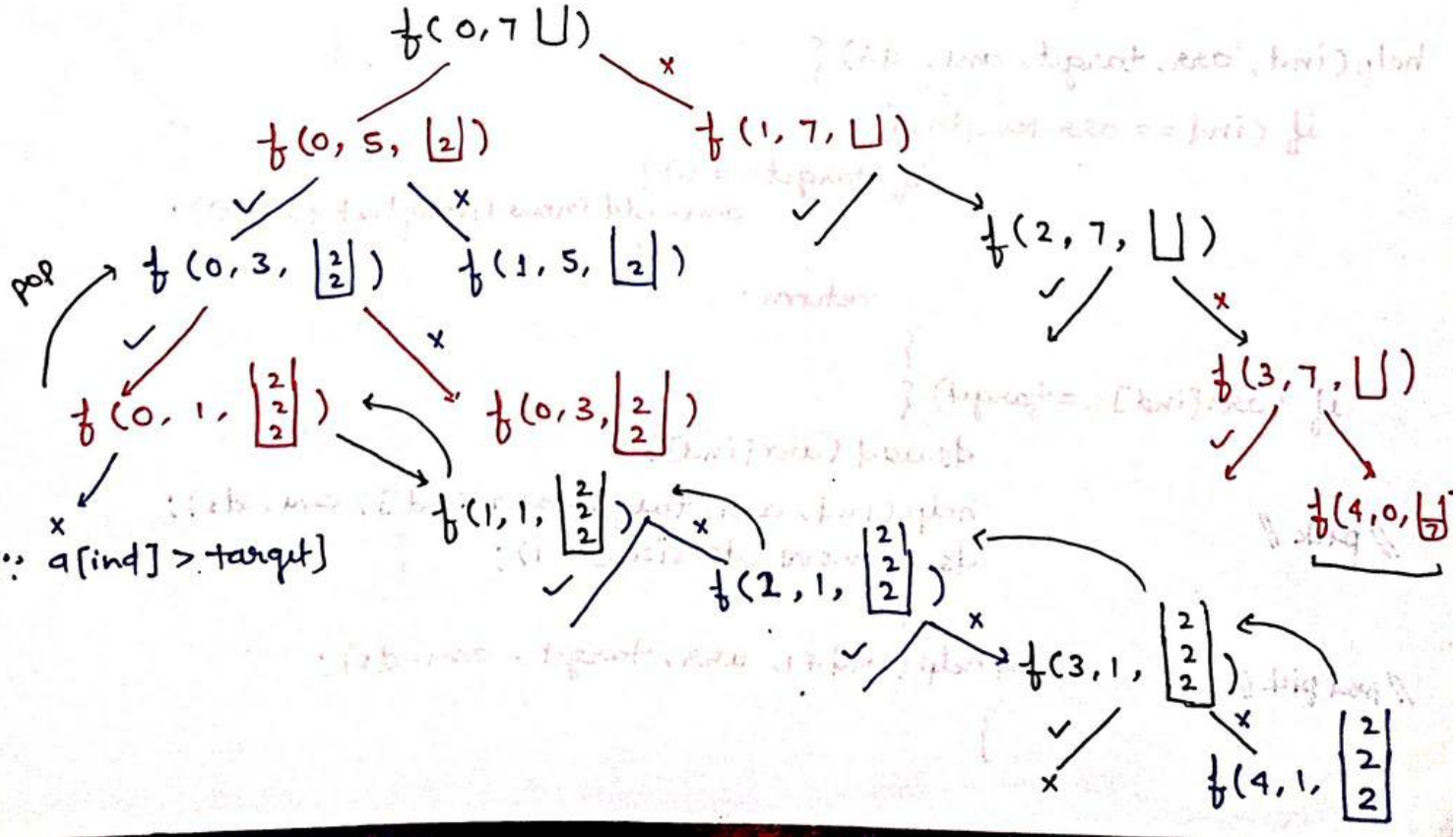
Intuition - for Questions like printing combinations or subsequences. the first thing strike - recursion.

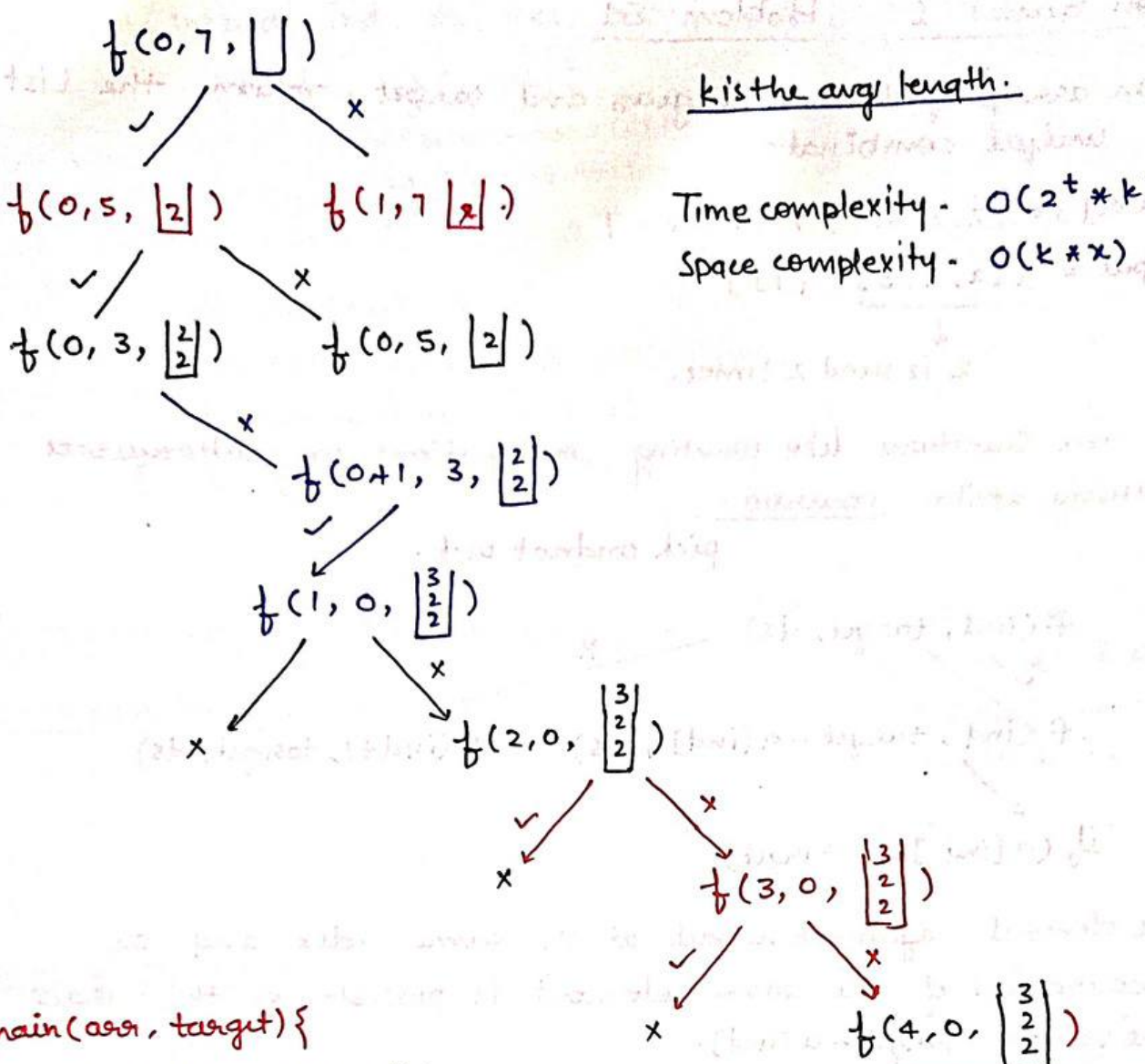
pick and not pick.



if pick the element, again comeback at the same index and as multiple occurances of the same element is possible so the target reduces the value. target - a[ind].

If not pick move forward to next index. while backtrack pop the last element. keep repeating process while index < size.





k is the avg length.

Time complexity - $O(2^t * k)$

Space complexity - $O(k * x)$

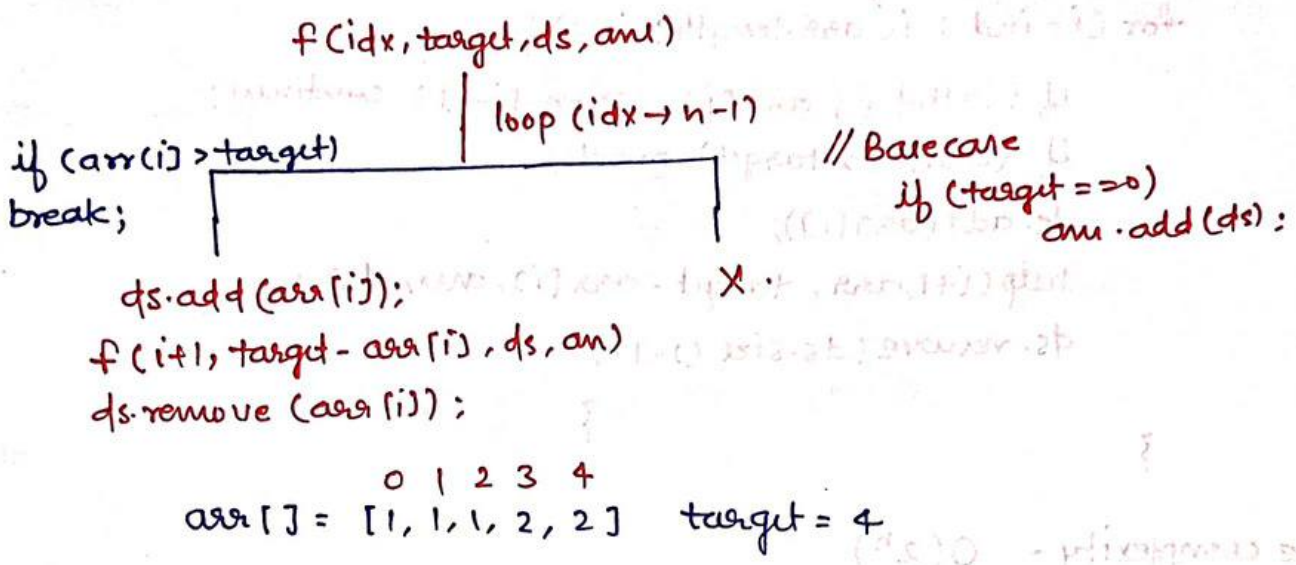
$\therefore$ [2,2,3]

```
main(arr, target){
List<List<Integer>> ans();
help(0, arr, target, ans, new ArrayList<>());
return ans;
}

help(ind, arr, target, ans, ds){
    if (ind == arr.length){
            if (target == 0){
                ans.add(new ArrayList<>(ds));
            }
        return;
    }
    if (arr[ind]<=target){
            ds.add(arr[ind]);
// pick //  help(ind, arr, target-arr[ind], ans, ds);
            ds.remove(ds.size()-1);
        }
// not pick //  find help(ind+1, arr, target, ans, ds);
    }
```

main combination Sume II · unique combinations.

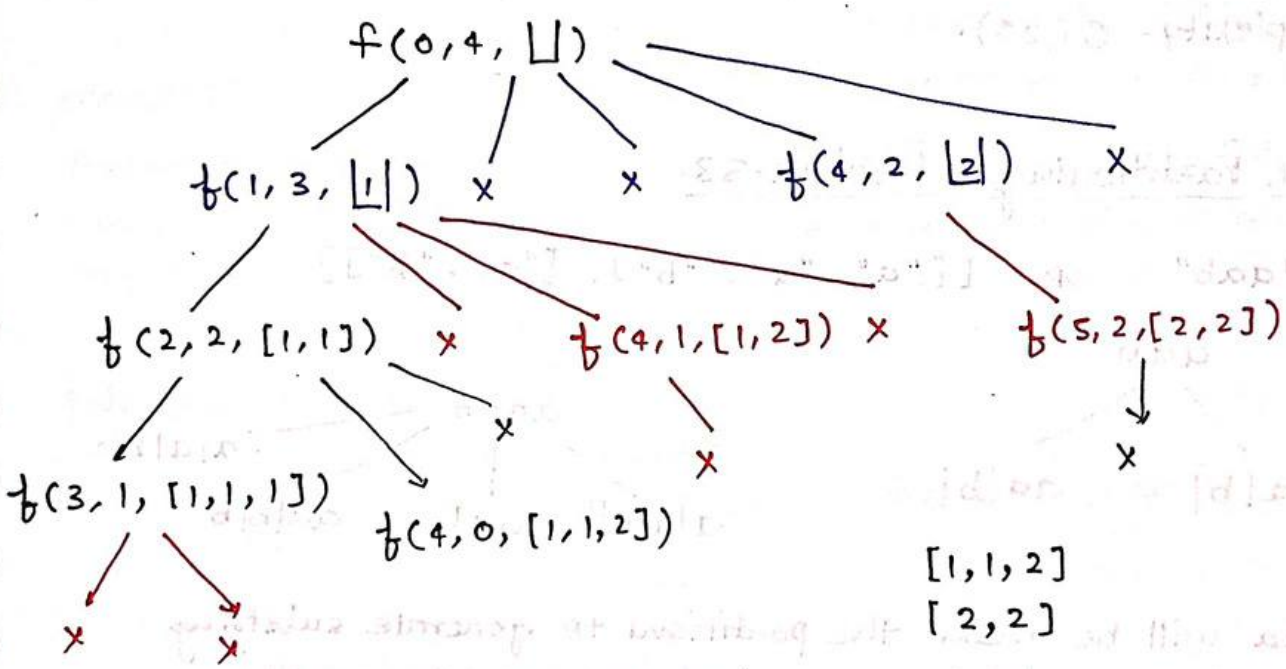
candidates = {10, 1, 2, 7, 6, 1, 5} , target = 8

Output = [[1, 1, 6] . [1, 2, 5] . [1, 7] , [2, 6]]



0 1 2 3 4

arr[] = [1, 1, 1, 2, 2] target = 4




```
main(arr, target) {
        List<List<int>> ans();
        Arrays.sort(arr);
        help(0, arr, target, ans, new ArrayList<>());
        return ans;
            }
```

```
help(ind, arr, target, ans, ds){
    if (target==0){
            ans.add(new ArrayList<>(ds));
            return;
            }
    for(i=ind; i<arr.length; i++){
        if(i>ind && arr[i]==arr[i-1]) continue;
        if(arr[i]>target) break;
        ds.add(arr[i]);
        help(i+1,arr, target-arr[i],ans, ds);
        ds.remove(ds.size()-1);
                                    }
}
```

Time complexity - $O(2^n)$
Space complexity - $O(2^n)$.

Palindrome Partitioning - Problem-53.

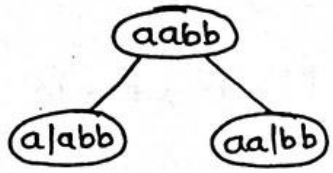
s = "aab" op. [["a","a","b"], ["aa","b"]]

Initial idea will be make the partitions to generate substring and check if the substring generated out of the partition will be palindrome.

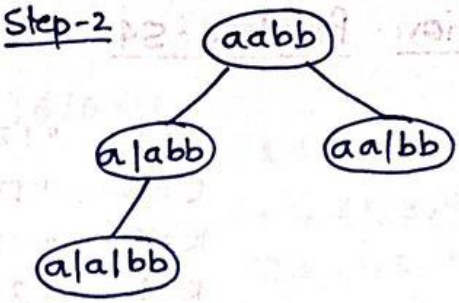
Generate every substring and checking for palindrome in every step.

We consider substrings starting from 0th index [0,0]. is a palindrom. so partition right after the 0th index. [0,1] is another palindrome make a partition after first index.

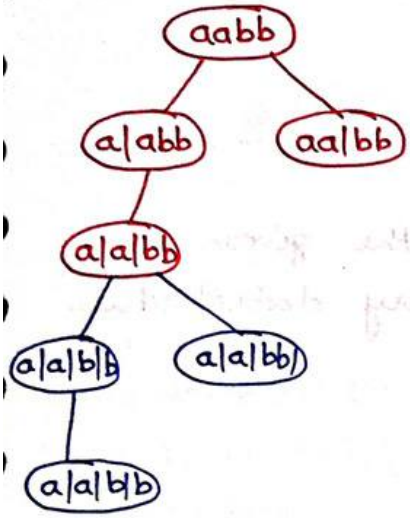
Step-1.



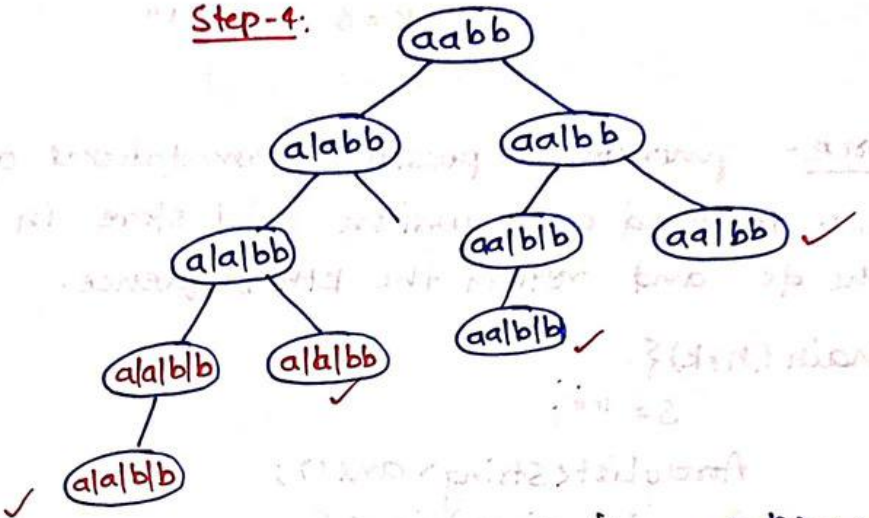
Step-2



Step-3



Step-4:



Time complexity - $O(2^n) * k * (n/2)$

Space complexity - $O(k * x)$.

2^n to generate every substring

O/n_2 → check palindrome, k is average length.

```
main(s){
ArrayList<ArrayList<int>>ans();
ArrayList<int> path();
help(0, s, ans, path);
    }
help(ind, s, ans, path){
        if(ind == s.length())
            ans.add(new ArrayList<>(path));
            return;
for(i=idx; i<s.length; i++){
    if(isPalindrome(s, indx, i)){
        path.add(s.substring(idx, i+1);
        help(i+1, s, path, ans);
        path.remove(path.size()-1);
                }
            }
        }
```

we have to write a fun for checking the palindrome.

find kth permutation · Problem - 54

N = 3 , K = 3.

Output: "213"

K = 1 "123"
K = 2 "132"
K = 3 "213"
K = 4 "231"
K = 5 "312"
K = 6 "321"

Bruteforce - generate all possible permutations of the given sequence Achieved by recursion and store in any datastructure sort the ds. and return the kth sequence.

```
main(n,k){
        s= "";
      ArrayList<String> ans();
      for(i=1; i≤n ;i++){
                        s+=i;
                       }
      help(s.charArray(),0, res);
          to                  Collections.sort(res);
      return. res.get(k);
           ans.
       }
```

Time complexity. n!*n

```
help(s[], idx, ans){
        if(idx==s.length){
          String str= new String(s);
          res.add (str);
          return;        }
             }
      for(i=idx; i<s.length; i++){
            swap(s, i, idx);
            help(s, idx+1, res);
            swap(s, i, idx);
                              }
       }
```

N = 4, K = 17 = 16th.

(1, 2, 3, 4)
0 1 2 3

3 4 1 2

16/6 = 2

16 % 6 = ④ th permutation

1, 2, 4, k = 4

1 + (2,3,4)]6 0-5
2 + (1,3,4)]6 6-11
③ + (1,2,4)]6 12-17
4 + (1,2,3)]6 18-23

↳ 1 {2,4} 2 (0-1)
② {1,4} 2 (2-3)
④ {1,2} 2 (4-5)
6

k = 4/2 = 2

4 % 2 = 0

{1,2}

1 {2} → 1 (0-1)

2 {1} → 1 (1-1)
2

k = 0/1 k = 0 % = 0

follow same process for 2

```
main(n,k){
 fact=1;
ArrayList<int> ds();
for(i=1;i<n;i++){
    fact = fact * i;
    ds.add(i);
                }
ds.add(n);
String ans = "";
k=k-1;
while(true){
ans = ans + " " + ds.get(k/fact);
ds.remove(k/fact);
if(ds.size()==0){
                    break;
                    }
k= k% fact;
fact= fact / ds.size();
  }
return ans;
```

Time complexity - $O(N^2)$
Space complexity - $O(N)$

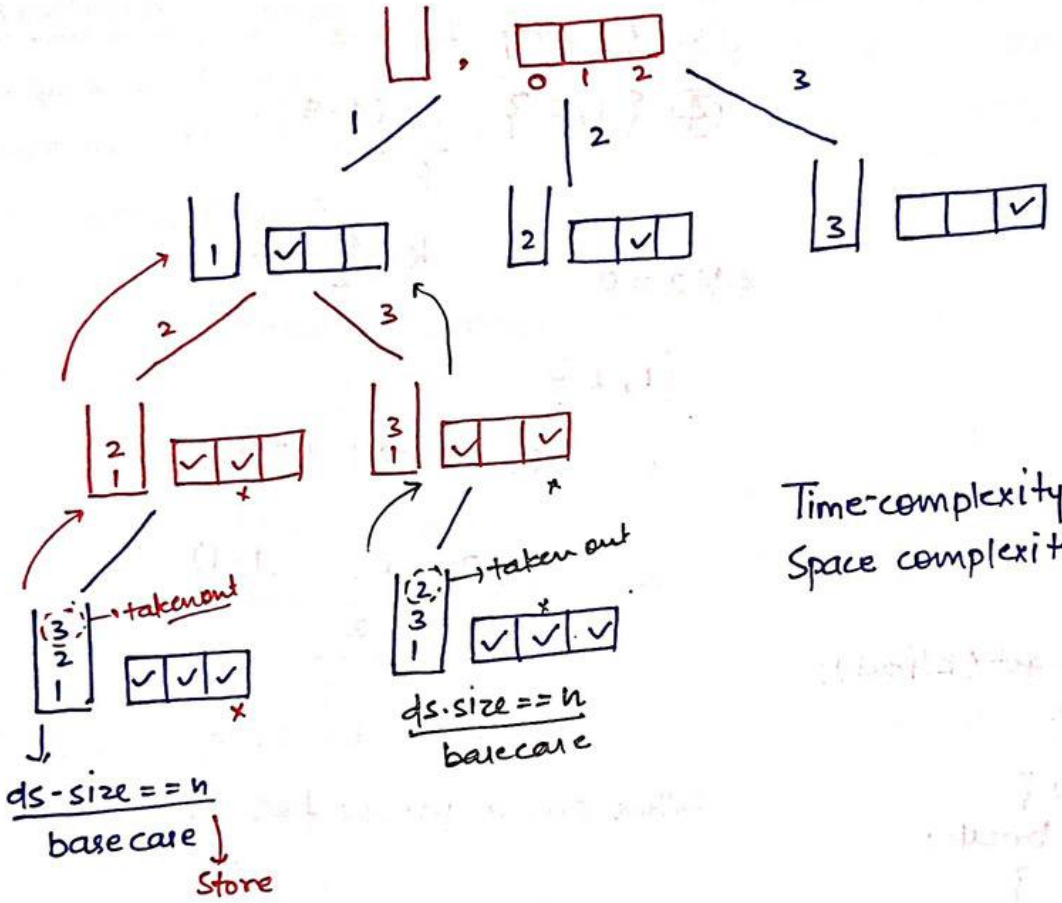
Print All Permutations of a String/Array Problem-55.

nums = [1,2,3]

o/p = [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

Using a Extra Space Generate all the permutations.

$3!_0$ = 6 total permutation.

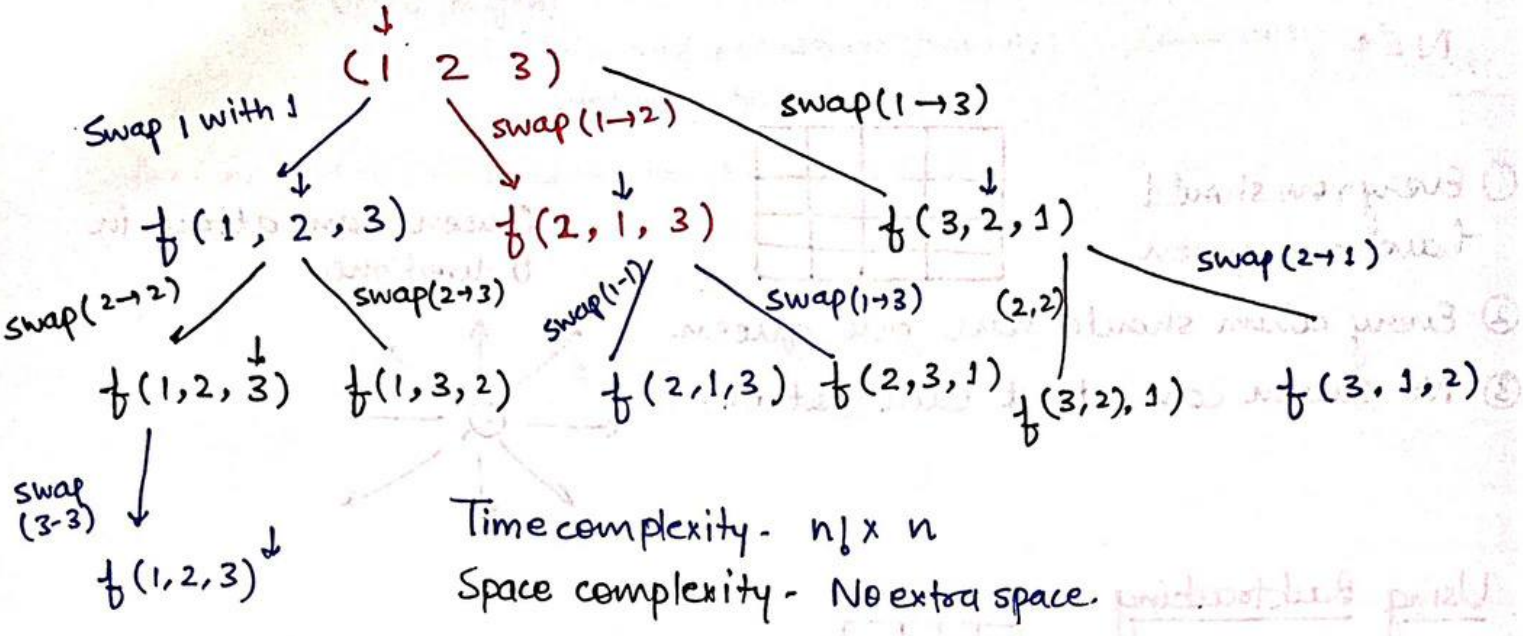


Time-complexity - $n!_0 \times O(n)$

Space complexity - $O(N) + O(N)$

```
main permute (nums) {
        List<List<int>> ans;
        List<int> ds;
        boolean freq[];
        recursive (nums, ans, ds, freq);
        return ans;
                }
recursive (nums, ans, ds, freq) {
        if (ds.size == nums.length) ans.add (new ArrayList<> (ds));
                                    return;
        for (i=0; i<nums.length; i++) {
                if (!freq[i]) {  freq[i] = true;
                                 ds.add[nums[i]);
                                 recursive (nums, ds, ans, freq);
                                 ds.remove (ds.size()-1)); freq[i] = false;
}}
```

Without using Extra space - Swapping technique.



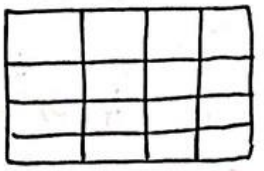
Time complexity - $n! \times n$

Space complexity - No extra space.

```
permute(nums) {
    List<List<int>> ans;
    recursion(0, nums, ans);
    return ans;
        }
recursion(index, nums, ans) {
    if (nums.length == index) {
            List<int> ds = new ArrayList<>();
            for (i=0; i<nums.length; i++) {
                ds.add (nums[i]);
                }
            ans.add (new ArrayList<>(ds));
            return;
            }
    for (i = index; i<nums.length; i++) {
            swap (i, index, nums);
            recursion(index+1, nums, ans);
            swap (i, index, nums);
                }
        }
voidswap(int i, int j, nums) {
            int t = nums[i];
            nums[i] = nums[j];
            nums[j] = t;
            }
```

N-Queen's Problem - Problem no-56.

N = 4

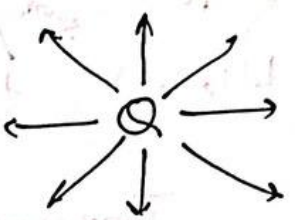


① Every row should have one queen

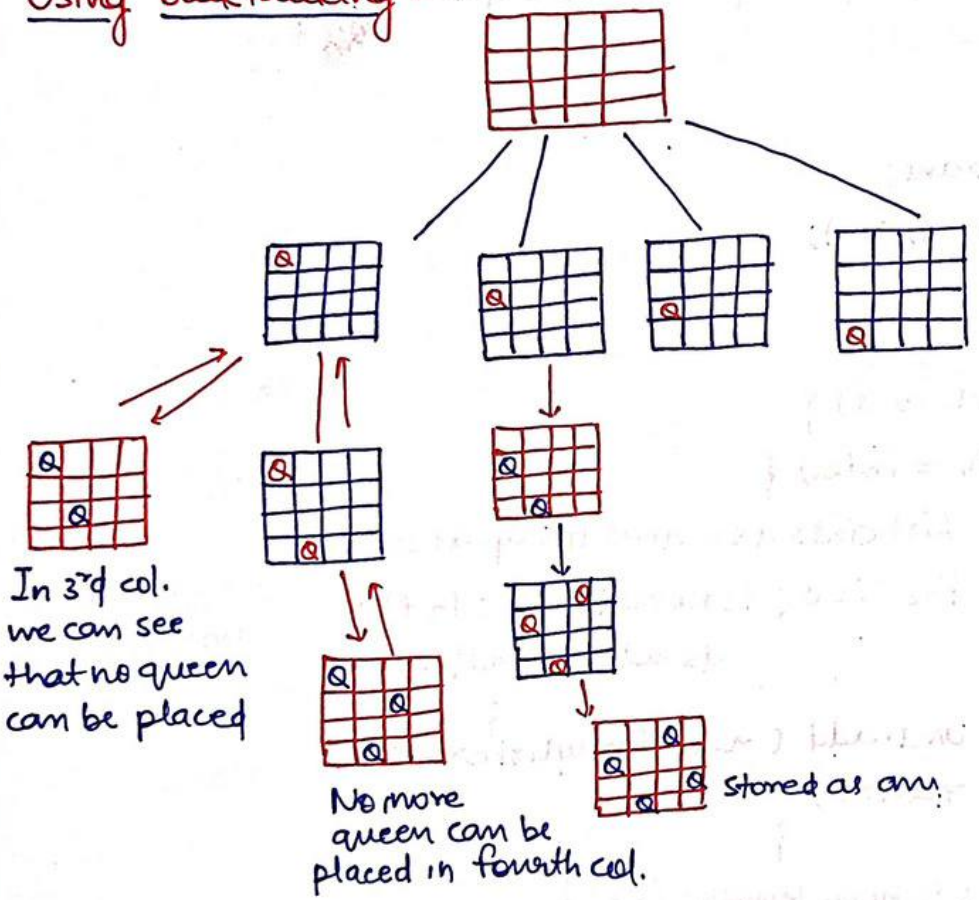
② Every colum should have one queen

③ No Queen can attack each other.

Queen can attack in 8 directions.



Using Backtracking -




```
main NQueen(n){
        char[][] board = new char[n][n];
        for(i=0; i<n; i++){
                for(j=0; j<n; j++){
                        board[i][j] = '.';
        List<List<String>> ans;
        dfs(0, board, ans);
        return ans;
            }
```

```
void dfs(int col, char[][] board, List<List<String>> ans){
    if(col == board.length)
                    res.add(construct(board)); // construct is a function
                    return;(board
    for(int row=0; row<board.length; row++){
            if(validate(board, row, col)){
                    board[row][col] = 'Q';
                    dfs(col+1, board, res);
                    board[row][col] = '.';
                                }
                            }
                        }

boolean validate(board, row, col){
    int duprow = row;
    int dupcol = col;
    while(row>=0 && col>=0){
            if(board[row][col] == 'Q') return false;
            row--;
            col--;
                    }
    row = duprow;
    col = dupcol;
    while(col>=0){
            if(board[row][col] == 'Q') return false;
            col--;
                }
    row = duprow;
    col = dupcol;
    while(col>=0 && row<board.length){
        if(board[row][col] == 'Q') return false;
        col--;
        row++;
            }
    return true;
        }
```

Time complexity - $O(N! * N)$

SC - $O(N^2)$

```
main(n){
  char[][] board = new char[n][n];
  for(i=0; i<n; i++){
        for(j=0; j<n; j++){
              board[i][j] = '.';
  List<List<String>> ans;
  int leftrow[] = new int[n];
  int upperdiagonal[] = new int[n];
  int lowerdiagonal[] = new int[n];
  solve(0, board, res, leftrow, upperdiagonal, lowerdiagonal);
return res;
  }

solve(){ if(col == board.length) res.add(construct(board));
                                  return;
        for(row=0; row<board.length; row++){
if(leftRow[row]==0 && lowerdiagonal[row+col]==0 && upper[board.len-1+col-row]==0){

    board[row][col] = 'Q';
    left[row] = 1;
    lowerdiagonal[row+col] = 1;
    upperdiagonal[board.len-1+col-row] = 1;
    solve();
    backtrack
    putting the reverse value over there.

  }
 }
}
```

Time complexity - $O(N! \times N)$

SC - $O(N)$

Soduko Solver - Problem no 57

Given a 9x9 incomplete soduko, solve it such that it becomes valid soduko.

① All rows filled with 1-9 exactly once
② All colums " " 1-9 " "
③ Each 3x3 submatrix filled with 1-9 exactly once

Intution - plain matrix traversal on the soduko. if any empty cell pause and try to filled (1-9) in that particular cell.

ⅱ because it has to satify all the condtions so call another function isValid(). → checks that the putted no is voilating the rules or not.

ⅲ If it is voilating, we try with the next no. If it is not we call the same function recursively, but this time with the updated state of the board.

ⅳ If at any point we are unable to put any of the value between 1-9 so we need to backtrack now. and we return false to parant funtion it tells we can not go this way

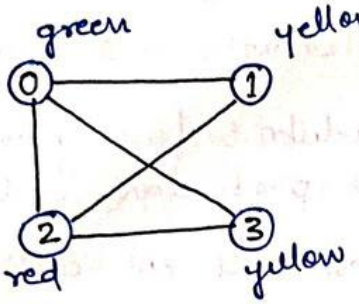
```
bool main(char[][] board){
    for(i=0;i<9;i++){
        for(j=0;j<9;j++){
         if(ch[i][j]=='.'){
          for(char c='1';c<='9';c++){
              if(isValid(board,i,j,c)){
                  board[i][j]=c;
                 if(solvesoduko(board))
                          return true;
                 else board[i][j]='.';
                }
              }
            return false;
          }
        }
    }
    return true;
    }
```

Timecomplexity - $O(9^{2n})$
SC - O(1)

```
boolean isValid(board, row, col, char c){
for(i=0;i<9;i++){
  if(boar[i][col]==c) return false;
  if(bod[row][i]==c) return false;
if(board[3*(row/3)+i/3][3*(col/3)+i%3]==c)
    return false;
}
return true;
```

M-coloring problem.

Given an undirected graph and a number m, determine if graph can be colored with the most m such colors., such that two adjacent nodes of graph not be same.



M = 3

yes we can fill all the node with the given color.

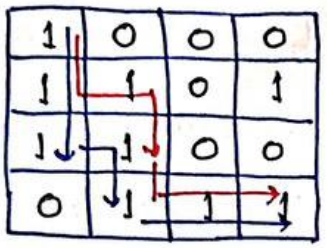
Time complexity - (N^m)
SC- O(N) + O(N)

```
boolean graphcoloring (List<int> G, color[], int i, int c) {
        int n= G.length;
        if (solve(i, G.color, n, c) == true) return true;
        return false;
                }
boolean isSafe (node (int), List<int> G, color[], int n, int c) {
    for (int it : G[node]) {
            if (color[it] == col) return false;
                    }
     return true;
            }
boolean solve (int node, List<int> G[], color[], int n, int m) {
     if (node == n) return true;
     for (int i = 1 ; i<=m ; i++) {
        if (isSafe (node, G, color, n, i)){
                color [node] = i;
                if (solve(node+1, G, color, n, m) == true)
                        return true;
                color [node] = 0;
                    }
            }
     return false;
            }
```

Rat in a Maze - Problem - 59

Consider a rat at (0,0) in a square matrix of n * m (n = m)
It has to reach the destination (n-1, m-1).

rat can move 'U' (up), 'D' (down), 'L' (left), 'R' (right)
He can move through 1 only.



2 ways - DDRDRR , DRDDRR

```
main (arr[][], n) {
    List<String> ans();
int vis[][];
    if (arr[0][0] == 0) return ans;
    String path = "";
    solve (0, 0, arr, n, lt, vis, path);
                }
```

Tc - O (4^{m*n})
Sc - O(m*n)

```
solve (int x, int y, int arr[][], n, ans, vis[][], path) {
    if (x == n-1 && y == n-1) lt.add(path);
                              return;

    vis[x][y] = 1;
    if (isSafe (x+1, y, vis, arr, n)) {
                    solve (x+1, y, arr, n, ans, vis, path + 'D');
                                }

// for right  x, y+1   same as down

// for left   x, y-1

// for down   x-1, y
                    vis [x] [y] = 0 ; // back track.  }

if
    isSafe () {
        if ((x>=0 && x<n) && (y>=0 && y<n) && vis[x][y] != 1 &&
                                    arr[x][y] == 1) {
                    return true; }
                return false; }
```

boundary condition

The Nth root of an Integer. Problem-61

N = 3, M = 27 $\quad \sqrt[3]{27} = 3$

N = 4, M = 69 $\quad \sqrt[4]{69} = -1$

Bruteforce - We can gurantee that our answer will lie between the range from 1 to m. i.e. the given no. So perform a linear search on the range and we will find the no x such that fun(x,n) = m if no such no exist, we will return -1.

fun(x,n) returns the value of x raised to the pow n.

⇓

x^n

```
main(n,m){
    for(i=1; i<=m; i++){
        long val = fun(i,n);
        if(val == (long) m) return i;
        else if (val > (long) m) break;
                                }
    return -1;
      }

long fun(b, exp){
    ans = 1
    base = b;
  while (exp > 0){
        if(exp%.2==1){
                exp--;
                ans = ans * base;
                    }
        else {
                exp/=2;
                base = base * base;
             }
      }
    return ans;
  }
```

Tc - O(NthRoot(N))

SC -

Binary Search :

```
main (n, m) {
        int low = 1, high = m ;
    while (low <= high) {
        int mid = (low+high)/2 ;
        int midN = fun(mid , n, m);
        if (midN == 1) return mid;
        else if (midN == 0) {
                low = mid+1;
                }
        else {
            high = mid - 1;
            }
        }
    }
    return -1;
    }

int fun(int mid, n, m) {
        ans = 1;
    for (i = 1 ; i <= n ; i++) {
        ans = ans * mid;
        if (ans > m) return 2;
                }
    if (ans == m) return 1 ;
    return 0;
    }
```

① fun(mid, n, m) == 1 means it is our answer

② fun(mid, n, m) = 0 get no is smaller than our answer

③ fun(mid, n, m) == 2 mean the get no is higher than our ans.

Median of A Matrix (row wise sorted) Problem - 62

r = 3, c = 3

1	3	8
2	3	4
1	2	5

Median = 3

Bruteforce - Sort the array 2D → using a array and simply find the median.

Optimal Approach.

low → 1 ..... high → 10^9] → starting search space

$mid = \frac{low + high}{2}$

$= \frac{1+15}{2}$

$= 8$

like for easyness implement let high is 15.

[1 ..... 15]

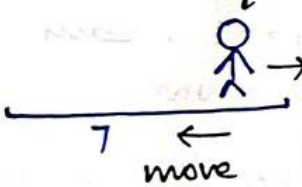
$\rightarrow \begin{bmatrix} 1 & 3 & 6 \\ 2 & 6 & 9 \\ 3 & 6 & 9 \end{bmatrix}$

just look in the matrix how many elements are there less than orequal to 8. search in particular row. In the first row there are 3 element $<= 8$

second row " " 2 " "

third " " " 2 " "

08 - 1 = 7



so there are 7 no.

[1 ..... 7] ⇒ $\frac{1+7}{2} = \frac{8}{2} = 4$

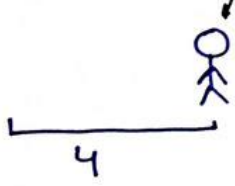
again, go in the matrix and check the numbers which is less than or equal to 4.

first row = 2

Second row = 1

third row = 1

04



low high

[5 ..... 7] = mid = 6

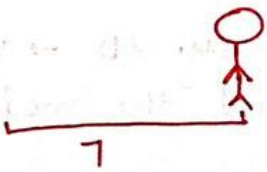
first row = 3

second row = 2

Third = 2

07

go to left mid-1 high



Optimal Approach.

low high
[1 ---------- 10^9] → starting search space
↑

$mid = \frac{low+high}{2}$

$= \frac{1+15}{2}$

$= 8$

like for easyness implement
let high is 15.

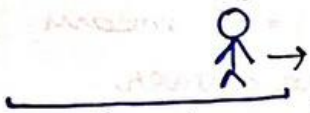
low
↓
[1 ---------- 15]
↑ high

→ [1 3 6]
→ [2 6 9]
[3 6 9]

just look in the matrix how many elements are there less than orequal to 8. search in particular row. In the first row there are 3 element <= 8

second row " " 2 " "
third " " " 2 " "
08 - 1 = 7

so there are 7 no.

8
↓

→ including 8 there are 7 no which is less than 8.
7 ← move

[1 ---------- 7] ⇒ $\frac{1+7}{2} = \frac{8}{2} = 4$

again, go in the matrix and check the numbers which is less than or equal to 4.

first row = 2
Second row = 1
third row = 1
04

4
↓

← including me there are only 4 member small or equal on left, because there are 4 no including 4. so low+1, mid+1
4

low high
↓ ↓
[5 ---------- 7] = mid = 6

first row = 3
second row = 2
Third = 2
07

go to left mid-1
high


7

low ↓ high ↓

[5 5] mid = 5

first = 2
second = 1
third = 91
04

No of elements less than median →

$K <= \left(\frac{N}{2}\right)$ (T) ↑

$K > \frac{N}{2}$ (T) ↓

[No of elements less than t]

Binary search:

c1	c2	c3
1	3	5
2	6	9
3	6	9

min of c1 (start) max of col 3 (end)

```
main findMedian(int[][] A, int row, int col) {
    int low = 1;
    int high = 10^8;
    int n = row;
    int m = col;
    while(low <= high){
        int mid = (low+high)>>1;
        int cnt=0;
        for(i=0; i<n; i++){
            cnt += help(A[i], mid, col);
        }
        if(cnt <= (n*m)/2)
            low = mid+1
        else
            high = mid-1
        }
    return low;
```

```
help(int[] A, mid, n) {
    int low = 0, int h = n-1;
while(low <= h) {
    int m = (l+h)>>1;
    if(A[m] <= mid){
        l = m+1;
    }
else
    h = m-1; }
return l;
```

Time complexity - $\log_2(2^{32}) \times N \times \log_2 M$

Search single element in a sorted Array Problem-63.

nums = [1,1,2,3,3,4,4,8,8]

O/P = 2

arr[i] != arr[i-1] and (ff) arr[i] != arr[i+1] : If this condition is true for any element : we arr[i], conclude as singal element.

① Edge cases: • n==1 , return the singal element.

• i i==0 , s0 first element of array the only condition we need to check arr[i] != arr[i+1]

• i == n-1 , check arr[i] != arr[i-1]

This is the brute force. solution.

Tc- O(N)
SC- O(1)

Optimized : using Binary Search.

arr[] = [1 1 2 2 3 3 4 5 5 6 6]

(1,1) (2,2) (3,3) (5,5) (6,6)

even odd even odd even odd odd even odd even

(even, odd) → element is on right half.
(odd, even) → element is on left half.

check for the edge cases.

low mid high

1 1 2 2 3 3 4 5 5 6 6

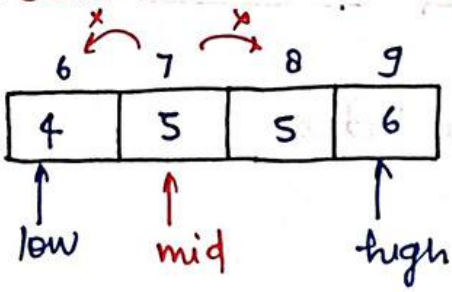
0 1 2 3 4 5 6 7 8 9 10

we put low on index 1 because we have already check idx 0 and 10 . in edge cases.

$mid = \frac{9+1}{2} = 5$

because mid is on odd the it mean previous even (even, odd) mean

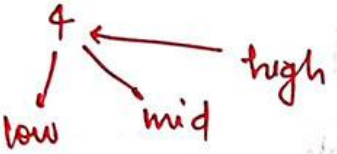
So our element is on the right half.



$\frac{6+9}{2} = 7.5 = 7$

mid is trying to find where is the pos.

So became 4 != 5 but 5 == 5 so be have to eliminate right half.



Singal element = 4

```
fun(arr,n) {
    if (n==1) return arr[0];
    if (arr[0] != arr[1]) return arr[0];
    if (arr[n-1] != arr[n-2]) return arr[n-1];
    low = 1, high = n-2;
    while (low <= high){
        mid = (low+high)/2;
    if (arr[mid] != arr[mid+1] && arr[mid] != arr[mid-1])
            return arr[mid];
    // left half.
if ((mid %2 == 1 && arr[mid-1] == arr[mid]) ||
            low = mid+1;        ((mid %2 == 0 &&
                                arr[mid] == mid[mid
else
    high = mid-1;
                        }
    return -1;
```

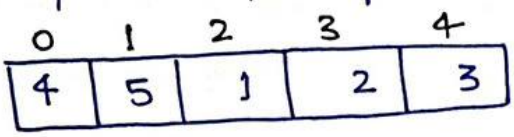
TC - $\log_2(N)$
SC - O(1)

Search Element in sorted rotated Array. Problem - 64

Bruteforce -

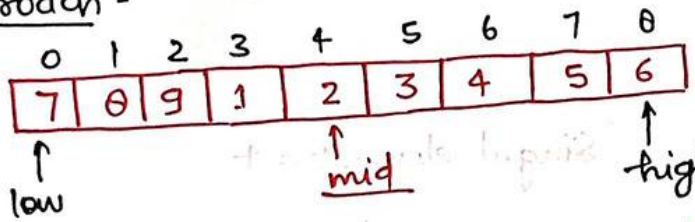
Linear traverse and get the index.

Time complexity - O(N)
Space complexity - O(1)



target element = 2
ans = 3

Optimize Approach -



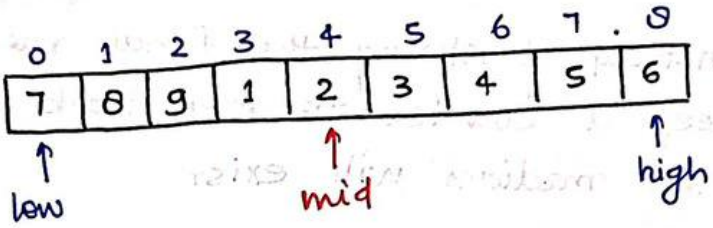
$mid = \frac{low+high}{2} = \frac{0+8}{2} = 4$, target value = 8

key observation: first we identify the sorted half of the array. Once found we determine if the target is located within this sorted half

```
main(arr, n, target) {
    low = 0, high = n-1;
    while (low <= high) {
        mid = (low+high)/2;
        if (arr[mid] == target) return mid;
        // if left part is sorted
        if (arr[low] <= arr[mid]) {
            if (arr[low] <= k && k <= arr[mid])
                // element exist
                high = mid - 1;
            else
                // element doesn't exist.
                low = mid + 1;
        }
```

```
else{
    // right part is sorted.
    if (arr[mid] <= target && target <= arr[high])
        {
            ~~high = mid~~ low = mid+1;
        }
    else {
        high = mid - 1;
        }
    }
}
return -1
}
```

We can clearly observe for every index one of the 2 halves will always be sorted.



$mid = \frac{low+high}{2}$

$= \frac{0+8}{2} = 4$

for check the sorted half.

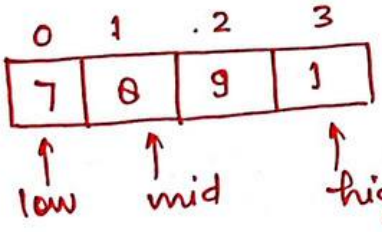
arr[low] <= arr[mid] ✗

goto right half.

check. arr[mid] <= target && target <= arr[high]
✓ ✗

(low = mid+1) ✗

else mid-1 = high ✓



$mid = \frac{0+3}{2} = \frac{1.5}{} = 1$ ✓ (return)

arr[low] <= target && target <= arr[mid] ✓

low =

Median of two sorted arrays. Problem-65

①

nums1 = [1,3], nums = [2]

O/P = 2.00000

merged array = [1,2,3] and median 2.

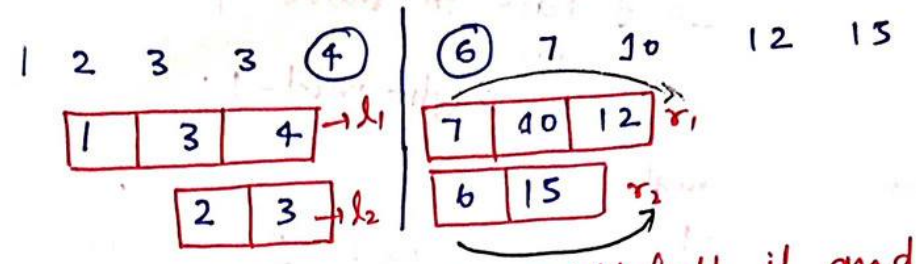
② nums = [1,2], nums2 = [3,4]

merged array = [1,2,3,4] → median = $\frac{2+3}{2}$ = 2.5

Naive Approach. our task is to merge into a array. and store the final sorted array in O(m+n) space new array.

Optimised Naive Approach. instead of storing the final merge sorted array, we can keep a counter to keep track of required position where the median will exist.

Optimised solution -



left half is smaller than the right half if and only if

$l_1 <= r_2$

$l_2 <= r_1$

if the array is even length then if we find,

$\max(l_1, l_2) + \min(r_1, r_2)$ and devide them by 2 so its the avg.

Total 10 elements 5 will be on left and 5 on the right.

So I can take min 0 elements from arr 1 and max 4 elements from array 1.

low	high
0	4

$cut\ 1 = \frac{0+4}{2} = 2$

(so pick 2 elements from arr 1 initially.)

cut 2 = $\frac{5-2}{}$ = 3 (how many element we can pick from 2nd)

arr 1 =

0	1	2	3
7	12	14	15

arr 2 =

1	2	3	4	9	11
0	1	2	3	4	5

here. $l_1 = 12$, $r_1 = 14$
$l_2 = 3$ $r_2 = 4$

[12 <= 4 (No) Not a p valid partition

So we have to decide wheather to move. we can see that.
12 has to smaller than 4. for the valid partition.

So I have to move left
so high = mid - 1.

low 0 high 1

cut 1 = $\frac{0+1}{2}$ = 0

(No element is taking from arr 1).

arr1 =

0	1	2	3
7	12	14	15

arr2 =

1	2	3	4	9	11
0	1	2	3	4	5

cut 2 = $\frac{5-0}{}$ = 5

If you dont have nothing assign minimal.

l_1 = INT-MIN; 1 2 3 4 9 | 7 12 14 15
l_2 | 11
r_1 r_2

$l_1 <= r_1$ (Yes)
$l_2 <= r_1$ (No)

So we have to decrease 9 and increase 7

low = mid + 1 = 0 + 1 = 1

low - 0+1 = 1

high 1 = 1

mid = $\frac{1+1}{2}$ = 1

cut 2 = 5 - 1 = 4

$cut1 = \frac{low + high}{2}$

$cut2 = \frac{(n_1+n_2)}{2} - cut1$

arr 1 =

7	12	14	15

arr 2 =

1	2	3	4	9	11

l_1 7 | 12 14 15 r_1

1 2 3 4 l_2 | 9 11 r_2

$l_1 <= r_2$ (yes)

$l_2 <= r_1$ (yes)

So take the min from right which is 9

and max from arr part 1, which is 7.

$= \frac{7+9}{2} = \underline{\underline{8}}$

$l_1 = arr1[cut1 - 1]$ $\quad r_1 = arr1[cut1]$

$l_2 = arr2[cut2 - 1]$ $\quad r_2 = arr2[cut2]$

If none elements on right half means r_1 and r_2 doesn't contain any value then, put the INT_MAX into it.

if length = 11 (odd)

⑥ ⑤

1 2 3 4 7 [9] | 11 12 14 15 16

Super simple for odd len.

median will lied on the left half $\frac{n_1+n_2+1}{2} \Rightarrow \frac{12}{2} = 6$

median = max (l_1, l_2)

```
main(arr1, arr2) {
if (arr2.size() < arr1.size())
                return main(arr2, arr1);
int n1 = arr1.size();
int n2 = arr2.size();
low = 0, high = n1;
while (low <= high){
    int cut1 = (low+high)/2;
    int cut2 = (n1+n2+1)/2 - cut1;      (works similar for even
                                          and odd cases).
  int left1 = cut1 == 0 ? INT_MIN: arr1[cut1-1];
  int right1 = cut1 == n1 ? INT_MAX : arr1[cut1];
  int left2 = cut2 == 0 ? INT_MIN : arr2[cut2-1];
  int right2 = cut2 == n2 ? INT_MAX: arr2[cut2];
  if (left1 <= right2 && left2 <= right1){
      if ((n1+n2) % 2 == 0)
          return (max(left1, left2) + min(right1, right2))/2.0;
      else
          return max(left1, left2);
  else if (left1 > right2){
      high = cut1 - 1;
                    }
  else {
      low = cut1 + 1;
    }
  }
return 0.0;
```

Time complexity - $O(\log(\min(m,n)))$

K^{th} element of Two Sorted Arrays. Problem - 66

arr1 = [2, 3, 6, 7, 9]

arr2[] = [1, 4, 8, 10]

k = 5

output = 6.

Bruteforce - Create another array (m+n) size and put all the element into it in sorted order then traverse and find the kth element.

Timecomplexity - O(m+n) + O(k)

Space complexity - O(m+n)

Optimise Approach -

1 2 3 3 4 | 6 7 10 12 15

1 3 4 | 7 10 12

2 3 | 6 15

So the question boils down how many elements I pick from 1st array for the 1st half and how many elements I need to pick from array2 for the 1st half.

if I pick

1 | 3 4 7 10 12

2 3 6 15 | X

l_1, l_2, r_1, r_2

0	1	2	3	4	5
1	3	4	7	10	12

2	3	6	15
0	1	2	3

this can't be possible.

because.

15 <= 3 (No).

K = 03 ← low 0, high 3

(change 0 to 1).

edge case.

there are 4 elements on the arr1 of and if k < arr1 then this condition arrives

K = 7 → low 0 1, high 4

if K > arr1 then we have pick element from arr2

l_1 = arr1 [cut1 - 1] , r_1 = arr1 [cut1]

l_2 = arr2 [cut2 - 1] , r_2 = arr2 [cut2]

|| ||

if 0 then put 0 then INT_MAX

INT MIN

So,

the code is similar to the prev. ques.

```
main (arr1, arr2) {
  if (arr1.length > arr2.length)
                return main (arr2, arr1);
  int low = max (0, k-m) , high = min (k, m);
  while (low <= high) {
        cut1 = (low+high)/2
        cut2 = k - cut1;
    l1 = cut1 == 0 ? INT_MIN : arr1 [cut1 - 1];
    l2 = cut2 == 0 ? INT_MIN : arr2 [cut2 - 1];
    r1 = cut1 == n ? INT_MAX : arr1 [cut1];
    r2 = cut2 == m ? INT_MAX : arr2 [cut2];

  if (l1 <= r2 && l2 <= r1)
          return max (l1, l2);
  else if (l1 > l2)
          return
  high = cut1 - 1;          Time complexity - O(log(min(n, m)));
  else
     low = cut1 + 1;
    }
  }
  return 1;
  }
```

Allocate minimum number of Pages - Problem-67.

we have N books and each with Ai pages. M students need to contigous books. out of all the permutations, the goal to find the permutation where the student with the most books allocated him. gets to min. no of pages.

N = 4

A[] = | 12 | 34 | 67 | 90 |

M = 2

Output = 113.

because each get min. 1 book

there are 2 guys. M1, and M2.

M1	M2	
[12]	[34, 67, 90]	→ maxpage = 191
[12, 34]	[67, 90]	→ maxpage = 157
[12, 34, 67]	[90]	→ maxpage = 113

minimum of all there three val = 113

① Each student gets at least one book.

② Each book should be allocated to only one student

③ Book allocation should be in contigous order

if no of students is greater than the no of book return -1;

12 (low) 203 (high) arr[] = [12 34 67 90]

high is the sum of all the elements in that array because lets assume

M = 1. arr = [12, 12, 12]

min = 36 because we give all the pages to 1 student, high = 36

what will be the low so assume

M = 3, arr[] = [12, 12, 12]

NO we can give 12, 12, 12 so every students and the min = 12, so low can be consider as min of arr.

So I define my search space its between -

12 (low) ↑ 107 (mid) ↑ 203 (high) ↑

mid = $\frac{low+high}{2}$

Student1 → 12 (because its not exceed 107)
we can take → 34, Student1 has = 12+34 < 107
again when we try to give 67 to student1 it becomes 113. which exceeds hence, we are not allocation it.

we allocate 67 to student 2

Student 2 → 67 < 107

we can not allocate 90 to student 2.

Sto, we allocate 90 to student 3.

Student 3 = 90 < 107.

But we dont have 3 student so we can not allocate books to the 2 students by keeping a barrier of 107.

So we have to increase the barrier.

So low = mid+1. (because 107 is not the right barrier so remove the left half).

[100 (low) ↑ 155 (mid) ↑ 203] (high) ↑

we can allocate student 1 → 12+34+67 < 155
but we are not allow further

Student 2 → 90 < 155

Student 1 = something
Student 2 = 90

So it is OK to 155 as a barrier, so it can be a our answer.

try for other valid allocation

we know for sure, 155 is giving us a valid answer so 156, 157.... 203 will also can be a barrier, but we have to find the minimal.

so high = mid-1; (Right remove portion).

[108 131 154]
low mid high

1 → 12+34+67 < 131

2 → 90 can be a valid answer

again we can say 132, 133... 154 can be a valid barrier.

so high = mid-1

[108 119 130]
low mid high

ans, 131 < 155 (our prev ans)

so ans will be 131

1 → 12+34+67

2 → 90

and 119 is a valid barrier so we reduce the answer

again 119 < 131 so ans = 119.

high = mid-1

[108 113 110]
low mid high

1 → 12+34+67 < 113

2 → 90

ans = 113 (because mid < 119)

high = mid-1

[100 110 112]
low mid high

1 → 12+ 34 we can't add 67 becaus of barrier.

2 → 67 we can't " 90 " " "

3 → 90 (we have only 3 students for distribution)

hence -

100, 109, 110 not valid so search space

low = mid+1

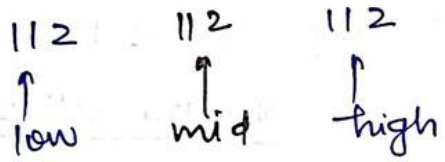
by keeping a barrier 111

Student 1 → 12+34

student 2 → 67

student 3 → 90 (X) → only 2 students.

hence low = mid+1.

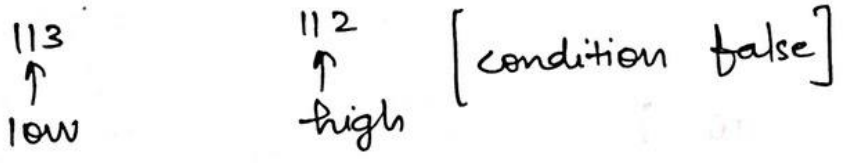


Student 1 → 12+34 not able to take 67. exceed 112.

Student 2 → 67

Student 3 → 90 (X) → only 2 students

low = mid+1



print the min ans.

```
main(arr, n, m){
    if (m>n) return -1;      // m is the no of student.
int low = min element of arr;
int high = sum of ele of arr;
while (low <= high){
    int mid = (low+high)/2
    int students = countstudent(arr,mid);
    if (student>m)
            low = mid+1;
    else
        high = mid-1;
    }
return low;
}
countstudent(arr, mid){
    n = arr.length;
    int student = 1;
long pagu = 0;
for (i=0; i<n; i++){
        if (pagu + arr[i]<= pamid)
                pagu += arr[i];
        else{
            student+= 1;
            pages = arr[i];
            }
        }
    return student;
}
```

Time complexity - $O(N * \log(\text{sum}[] - \max(arr[]+1))$

Agressive Cows - Problem-68

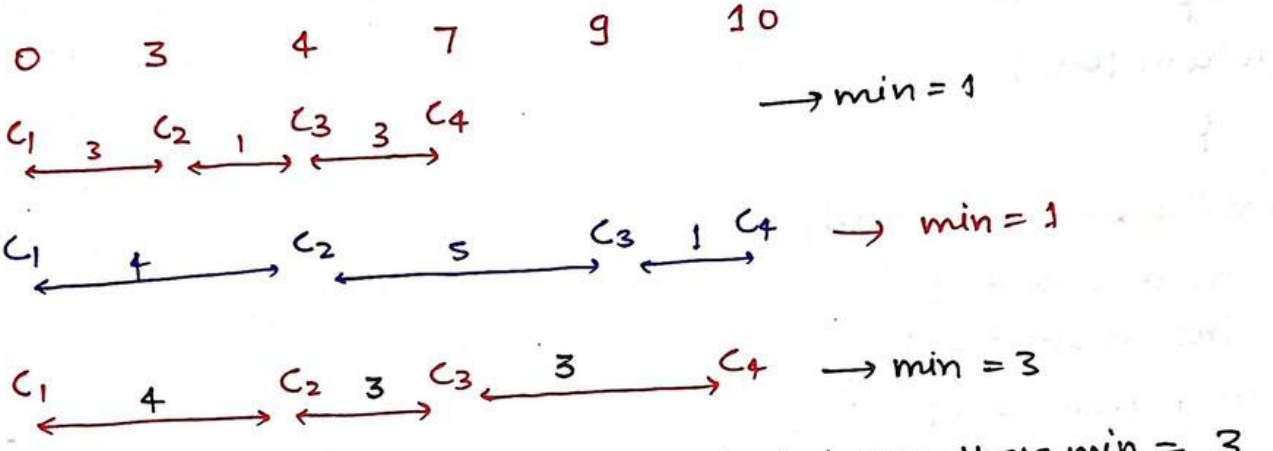
Aggresive cow → (min dist between cows) is max

arr[] = [0 3 4 7 10 9] cows = 4

Nstalls.

We have to put the 4 cows on the stalls in such a way the min distance between any of two cows is max.

Sort the array so the some stalls will be in contigious order and the distance will be min between two



at start weneed to try the cows will be put at min distance of 1. initially put cow1 at 0 initially at every step.

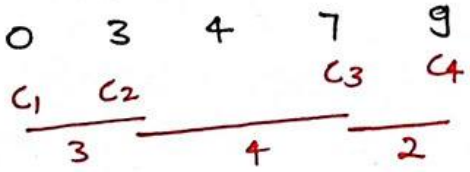
arr[] = [0 3 4 7 9 10] cows = 4

C1 C2 C3 C4

3 1 3

① ✓

② ✓ Now we try to put the cow in the min distance of two.



③ try to put at min distance of 3.

0 3 4 7 9 10

c_1 c_2 c_3 c_4

3 4 3

Yes 3 valid.

④ try to put at min distance of 4

0 3 4 7 9 10

c_1 c_2 c_3

4 4

but further we can't go min distance between the max

swap

ans = 3

Time complexity - $O(max - min) * O(N) \approx O(N^2)$

Space complexity - $O(1)$.

Optimised solution -

- create a range , 0 10

0 1 2 3 4 5 6 7 8 9 10

✓ ✓ ✓ x x x x x x

low mid high

min distance = 5 but we can put the cows at distance of 5.

remove the right half. , high = mid - 1 ;

1 2 3 4 5 6 7 8 9 10

x

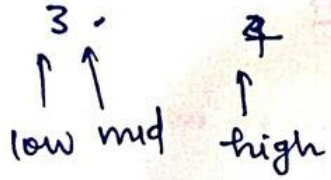
low mid high

$\frac{1+4}{2} = 2.5$

Now dist = 2 valid

so go to low = mid + 1.

3.



also valid to put the cows at min distance

3.

low = mid + 1

mid = (low + high)/2

mid = 4

we can't put the cows at distance of 4

so

[high = ~~low~~ mid - 1.]

so will return the high.
because it go to 3.

```
main (arr, k) {
    sort (arr);
   n = arr.size();
low = 1 ; high = ~~stall~~ arr[n-1] - arr[0]
while (low <= high) {
        mid = (low + high)/2 ;
       if (canPut (arr, mid, k) == = true)
                  low = mid + 1;

       else
                  high = mid - 1;
                  }
return high.
        }
```

TC - O(NlogN) + O(N * log (max(arr[]) - min(arr[]))

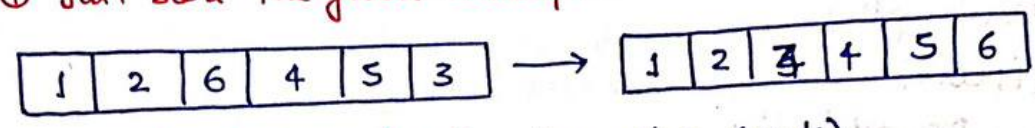
```
canPut (arr, mid, k) {   → cows
                cntcow = 1, lastStall = arr[0];
                for (i = 1 ; i < arr.length ; i++) {
                        cntcows++;
                        lastStall = arr[i];
                                    }
                        if (cntcows >= cows)  return true;
                                    }
                        return false;
}
```

Kth largest / Smallest element in an Array Problem - 69

Array = [1, 2, 6, 4, 5, 3] , K = 3

largest = 4, smallest = 3

Bruteforce - ① Just sort the given array.



smallest element will be = (K-1) , largest = (n-K)

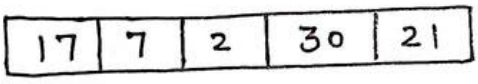
Time complexity - O(n log n).

Solution: Using Heap.

- The idea is to construct a max-heap of elements. Since the top of the max is the largest elements of the heap. we will remove top (K-1) elements from the heap.

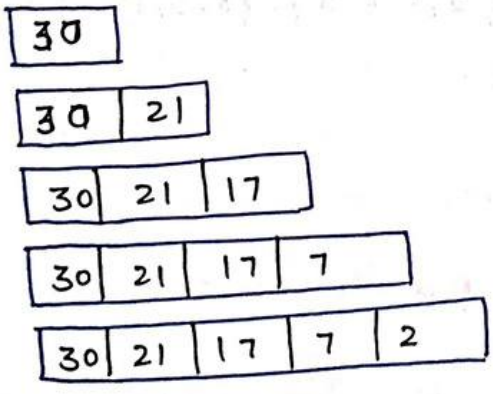
The top element will be Kth largest element.

let the array be.



and K = 3

Gen. the max heap.



Now extract K-1 values.

means 2 value.

extract

- To get Kth smallest element, we will use min-heap.

After removal of top K-1 elements, the K^{th} smallest element is top of the priority queue.

```
main(arr, k) {    // largest element
    PriorityQueue<Integer> pq = new PriorityQueue<>((a,b) -> b-a);
    n = arr.length;
    for (i: arr)
        pq.add(i);
    f = k-1;
    while (f > 0) {
        pq.remove();
        f--;
    }
    return pq.peek();
}
```

```
main(arr, k){ // min element
    n = arr.length.
    for (i: arr)
        pq.add(i);
    f = k-1;
    while (f>0) pq.remove();
                f--;
    return pq.peek();
}
```

Time complexity - $O(K + (n-k) * \log k)$

Space - $O(K)$

Merge K Sorted Arrays - Problem 70

K = 3, N = 4, arr = {{1, 3, 5, 7}, {2, 4, 6, 8}, {0, 9, 10, 11}}

Output - 0 1 2 3 4 5 6 7 8 9 10 11

Bruteforce: • Create an array of size N*K.

• traverse the matrix till the end and put all the elements in the array of size N*K

• sort the array

Time complexity - $O(N * K * \log(N*K))$

Space " = $O(N*K)$

Using Merging concept -

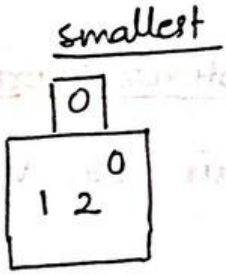
```
public static class Pair implements comparable < Pair > {
        int li;
        int di;
        int val;
Pair (int li, int di, int val) {
        this.li = li;
        this.di = di;
        this.val = val;
    }
```

```
public int compareTo (pair o)
    return this.val - ~~this~~ o.val;
```

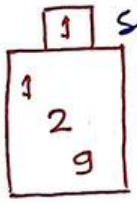
ans = 0 1 3 .....

p.q →

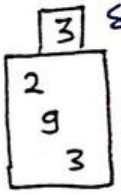
initially put the idx = 0 element in pq.



Now. pq. will



Now put 3




```
main (Arraylist < Arraylist < Int >>)
{
    ~~Arraylist < Integer > lt ();~~
    PriorityQueue < Pair > pq = new PriorityQueue ();
    for (i=0; i < list.size (); i++) {
        Pair p = new Pair (i, 0, list.get (i).get (0));
        pq.add (p);
                                    }

    while (pq.size () > 0)
            Pair p = pq.remove ();
            lt.add (p.val);
            p.di++;
            if (p.di < list.get (p.di).size ()) {
                p.val = list.get (p.li).get (p.di);
                pq.add (p);
                                    }
```

K maximum sum combinations from two arrays.

A = [1, 2, 4, 3], B = [2, 5, 1, 6], C = 4

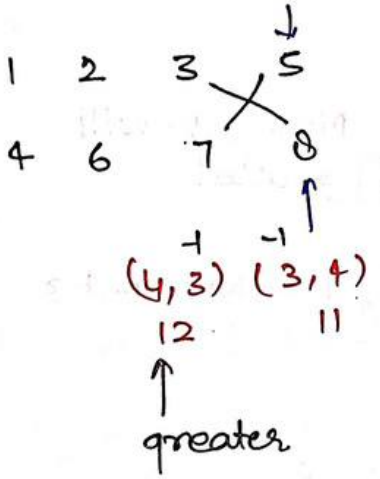
Output : [10, 9, 9, 8]

Bruteforce - find every pair sum arrange them in descending order and take the the first c number.

Time complexity - $O(N^2 * \log(N^2))$

Space complexity - $O(N^2)$

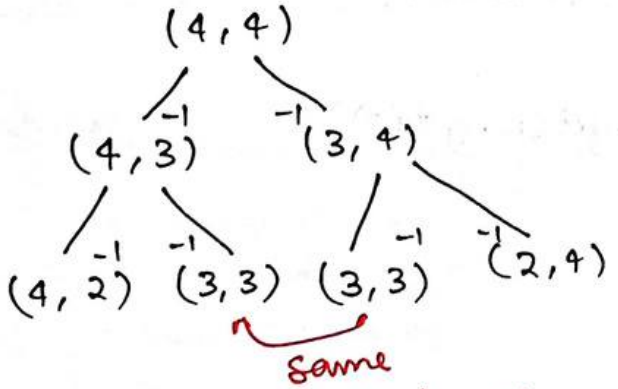
Optimal Approach - We have to take care that no repeated pair so we have to use the hashset.



first pair (13, (4, 4))

ans = 13, 12

second pair = (12, (4, 3))



that's why we need hashset

Useing maxheap and hashset -

```
main (A[], B[], N, C) {
    sort(A);
    sort (B);
// Maxheap Pair of format (sum, (i, j))
  PriorityQueue < PairSum> sum ();
// pairs is use to store the indicies so we use the pairs
 to make sure no indicies repeat itself.
  HashSet<Pair> pairs();
// initialize heap with max sum.
 int l = N-1;
 int m = N-1;
 pairs.add (new Pair (l,m));
 sums. add (new PairSum ( A[l]+B[m], l, m));
// Iterate upto K.
    for (i=0; i< C ;i++) {
        PairSum max = sum.poll ();
        print (max.sum);
        l= max. l-1;
        m = max.m;
     if (l>=0 && m>=0 && ! pairs.contains (new Pair (l,m))
        {
          sums.add (new PairSum(A[l] + B[m], l, m));
          pairs.add (new Pair (l,m));
        }
      l = max.l;
      m= max.m-1;
    if (l>=0 && m>=0 && ! pairs.contains (new Pair (l,m))
        {
          sums.add (new PairSum ( A[l]+B[m], l , m)) :
          pairs.add (new Pair (l,m));
        }
}
```

```
public static class Pair {
    Public Pair (int l, int m) {
        this.l = l;
        this.m = m;
    }
    int l;
    int m;
@Override public boolean equals(Object o)
{
    if (o == null) return false;
    if (!(o instanceof Pair)) return false;
    Pair obj = (Pair) o;
    return (l == obj.l && m == obj.m);
}
@override public int hashcode (){
        return Objects.has (l,m);
    }
}

public static class PairSum implements Comparable<PairSum> {
    public PairSum (int sum, int l, int m) {
        this.sum = sum;
        this.l = l;
        this.m = m;
    }
    int sum;
    int l;
    int m;
@override public int compareTo (PairSum o) {
        return Integer.compare (o.sum, sum);
    }
}
```

Time complexity - $O(N \log N)$

Space complexity - $O(N)$

Top K frequent Elements.

```
class Pair { int val;
             int freq;
             pair (int a, int b) {
               val = a;
               freq = b;
             }
           }
class cpair implements comparator <pair> {
    public int compare (pair a, pair b) {
      return b.freq = a.freq ? b.val - a.val : b.freq - a.freq
                                                    }
                                                    }
main (arr, K) {
   Map <int, int> mp();
   for (int i : arr) {
          mp.put (i, mp.getOrdefault (i,0)+1);
                    }
   PriorityQueue < Pair> p = new PriorityQueue <> (new cpair ())}
 for (Map.Entry <int, int> e:map.entrySet()) {
        p.add (new pair (e.getkey(), e.getValue ()));
                                                    }
 ArrayList <int> lt ();
 int j=0;
 while (j<k) {
        pair pp = p.poll ();
        int val = pp.val;
        if (! lt.contains (val))
              lt.add (val);
          j++;
          }
 int ans[]= new int [lt.size()];
 for (i=0 ; i< lt.size ;i++)   ans[i]= lt.get (i);
```

Time complexity - O(NlogN)
Space complexity - O(N).