

Striver

Dsa

Sheet

"180"

Set Matrix Zero: Problem no - 1

1	1	1
1	0	1
1	1	1

\Rightarrow

1	0	1
0	0	0
1	0	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);
            }
        }
    }
}

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);
    }
}

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

```

Same as the
column

Better Approach :- Using Extra Space

1. Define 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 i th index of row array i.e. $\text{row}[i]$ and j th index of col array $\text{col}[j]$ as 1. → perform for every cell containing 0.
4. Traverse matrix and if $\text{row}[i]$ or $\text{col}[j]$ 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;`

`}`

`}`

`for (int j=0; j<m; j++) {`

~~`if (mat.get(i).get(j) == 0) {`~~

`mat.get(i).set(j, 0);`

`}`

`}`

`}`

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

Complexity -

$O(N) + O(M)$

row arr

col arr.

Optimal Approach :-

col = 1

1	1	1	1
1	0	1	1
1	1	0	1
0	1	1	1

1. We take the arrays inside the Matrix.

col[0] → row[n]

row[0] → col[m]

first fill the zeros inside the 2-D matrix.

col=1 ⁰	1	0	1	0
X ⁰	0	1	1	
-X ⁰	1	0	1	
0 ¹	1	1	1	

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 col0 variable.

finally change the 1st row if mat[0][0] = 0
and column if col0 = 0

main (ArrayList<ArrayList<Integer>> mat) {

// int row[] = new int[n]; --> mat[..][0]

// int col[] = new int[m]; --> mat[0][...]
int col0 = 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(i).set(j, 0);

else

col0 = 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);

}

}

Step 3: finally mark the 1st 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 - $O(2 * (N * M))$

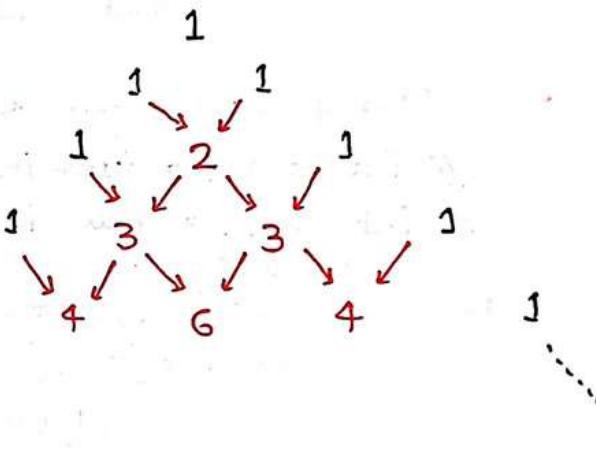
Pascal Triangle - Problem No-2

There are 3 variations -

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

$$R = 5, C = 3$$

$$\text{answer} = 6$$



② Print any Nth row of pascal triangle.

$$N = 5$$

$$\text{answer} = 1 \ 4 \ 6 \ 4 \ 1$$

③ print the entire pascal triangle.

$$N = 5$$

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

$${}^{R-1}C_{C-1} = {}^4C_2 = \frac{4 \times 3 \times 2 \times 1}{(2 \times 1)(2 \times 1)} \\ = 6$$

Brute force

Observation -

$${}^7C_2 = \frac{7!}{2! \times 5!} = \frac{7 \times 6 \times (5 \times 4 \times 3 \times 2 \times 1)}{2 \times 1 \times (5 \times 4 \times 3 \times 2 \times 1)} = \frac{7 \times 6}{2 \times 1}$$

$${}^{10}C_3 = \frac{10 \times 9 \times 8}{3 \times 2 \times 1} = \frac{10}{1} \times \frac{9}{2} \times \frac{8}{3}$$

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);
}

```

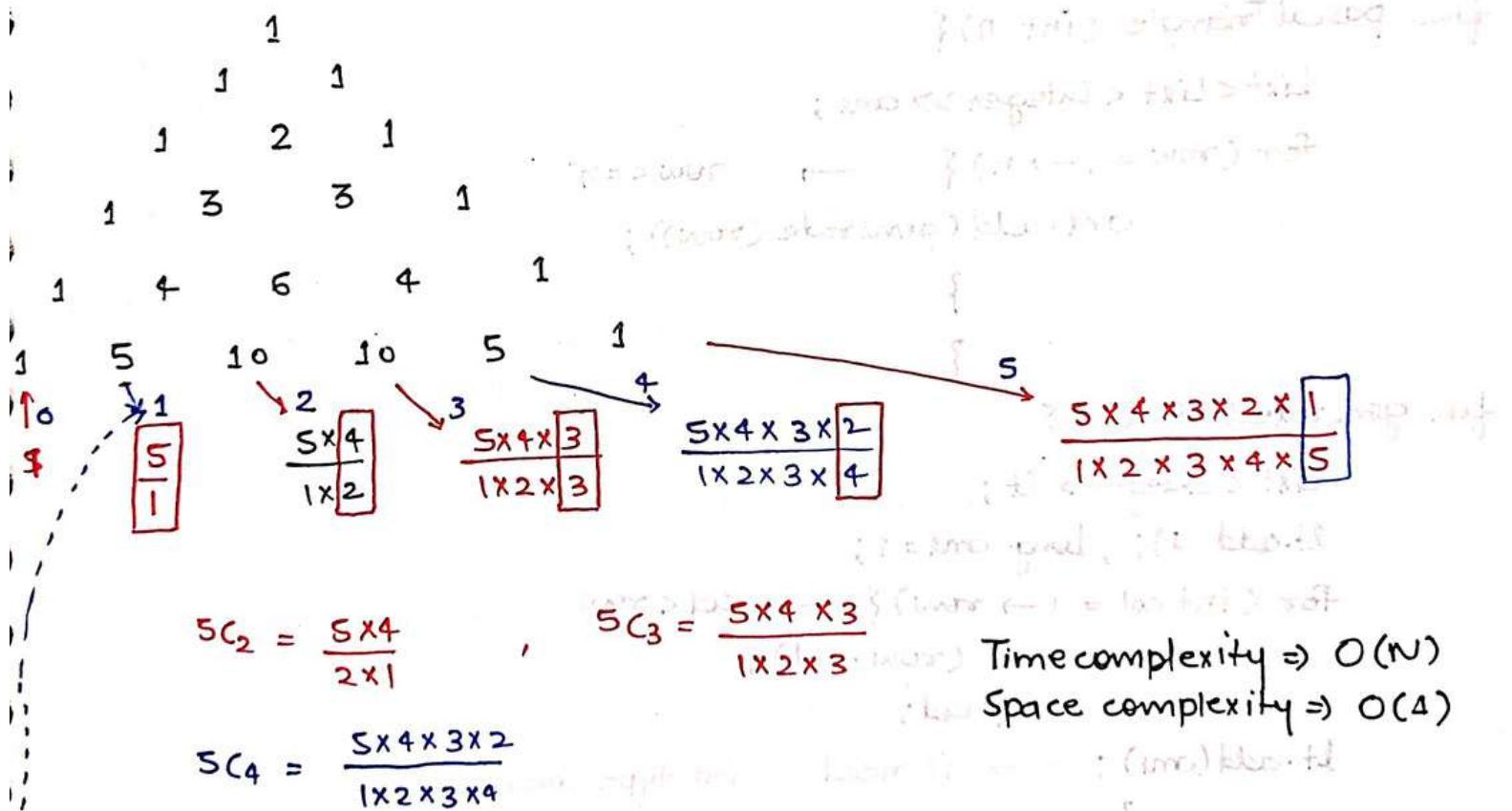
$$\left. \begin{array}{l} TC = O(\gamma) \\ SC = O(1) \end{array} \right\} \text{Time and Space}$$

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

N^{th} row has the N elements.

formula: $r-1 c_{c-1}$ for ($c=1$; $c \leq n$; $c++$) { ---> $O(N)$
 print (fun Ncr (n-1, c-1)); -> $O(r)$
 }

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



- ① The first and the last element is always 1.

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

```

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 brute force -

Time complexity:

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

Space complexity:

$$O(1)$$

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

Optimization: So we use the 2nd Variation for printing only row which is taking $O(N)$ for printing.

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

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;

}

↑ calculate row elements from left to right off

3. Next Permutation - Problem - 3

$\text{arr}[] = [3, 1, 2] = n! = 3! = 6 \text{ ways}$

1 2 3

Write them in Dictionary order = sorted order.

1 3 2

2 1 3

2 3 1

3 1 2

$123 < 132 < 213 \dots$

3 2 1

$123! \rightarrow$

like if we have to find the next permutation \rightarrow answer

$123!$

edge case : 3 2 1 \rightarrow the Next permutation

the answer will be 1 2 3

Bruteforce : 1. Generate all permutation in sorted order. \rightarrow recursion

2. Line Search for finding the given element.

3. Next index

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

Optimal solution:

$\text{arr}[] = \{2, 1, 5, 4, 3, 0, 0\} \quad n=7$

try to match everything

$\{2, 1, 5, 4, 3, 0, 0\}$ 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.

$\{2, 1, 5, 4, 3, 0, 0\}$

we can make

$\{2, 1, 5, 4, 3, 0, 0\}$

$\{0, 0, 3, 0, 3, 0\}$ both are less than 300 so it will same. not get anything greater

$\{2, 1, 5, 4, 3, 0, 0\}$

$\rightarrow 54300 \text{ (equal)}$

$\rightarrow 45300 \text{ (less)}$

$\rightarrow 34500 \text{ (less)}$

[2 1 5 4 3 0 0]

↓
checking for
which one is just greater
than this.

5 4 3 1 0 0 (bigger)

(but it is right after this it is
a question)

2 3

2 4

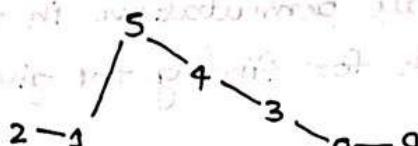
2 5

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

2 3

first two elements

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)
}

```

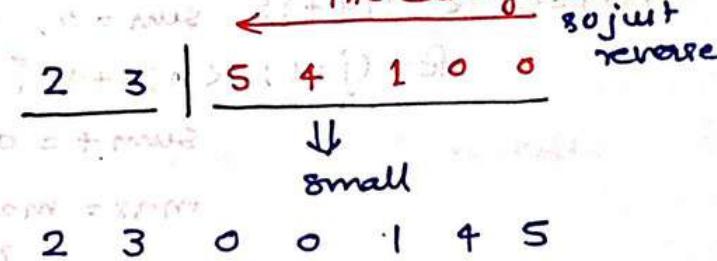
```
for(i=n-1; i>=ind; i++)
```

```
{ if (arr[i] > arr[ind])
```

```
    swap(arr[i], arr[ind]);
```

```
(n) O = unique number
    break;
}
```

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



Time complexity - $O(3N)$

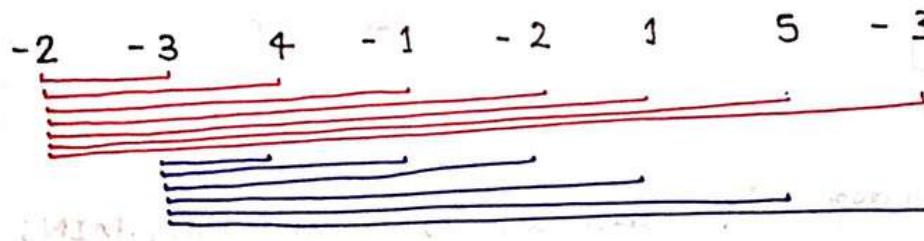
Space complexity - $O(1)$

Kadane's 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-MIN;
```

```
for (i=0; i<n; i++) {
```

```
    for (j=i; j<n; j++) {
```

```
        sum = 0;
```

```
        for (k=i to j) {
```

```
            sum = sum + arr[k];
```

```
        }
```

```
    }
```

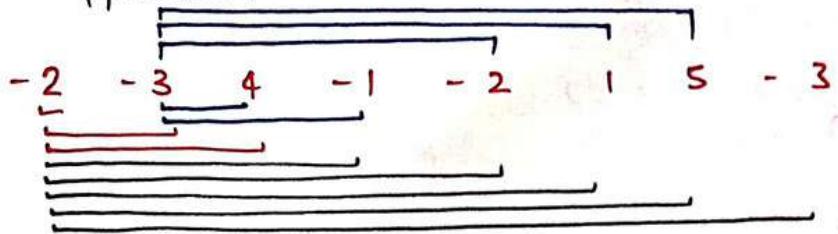
```
    }
```

```
}
```

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(N²)

Optimal Approach -

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
 $arr[] = [-2, -3, 4, -1, -2, 1, 5, -3]$
 $one = 7$

maxi = INT-MIN. ~~→ 247~~

sum = 0 ~~-2 + -3 = -5~~ $\neq 3 \times 2 + 5 = 7$ \rightarrow (check every time sum < 0)
 every step compare sum with the current sum = 0
 maximum and update maximum

$[-2, -3] \quad [-3]$ Sum < 0
 $\circled{-5} < \circled{-3}$

maxSubArray(arr, n) { long sum = 0, maxi = LONG-MIN;

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

 sum += arr[i];

 if (sum < 0) {

 sum = 0;

Check if there is no sum ≥ 0 so it is zero.

use this line over there

 maxi = Math.max(maxi, sum);

}

 return maxi;

}

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

Time complexity - $n \log n$
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 -

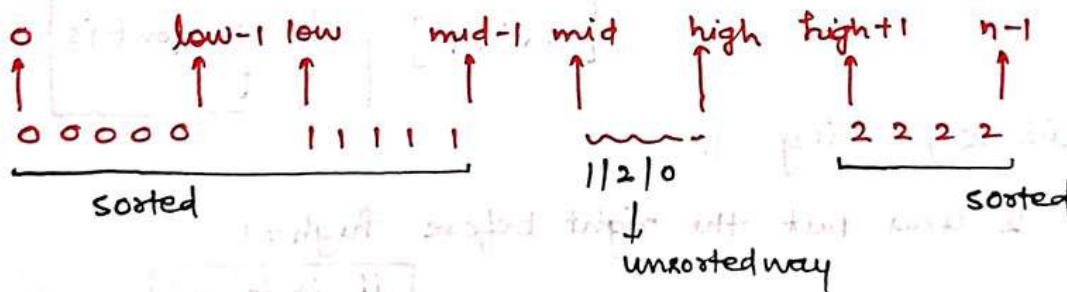
Dutch National flag Algorithm

pointers
 low
 mid
 high

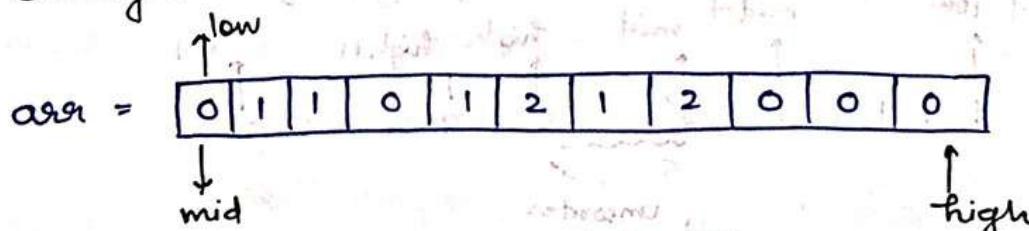
$[0 \dots low-1] \rightarrow 0$ extreme left

$[low \dots mid-1] \rightarrow 1$

$[high+1, n-1] \rightarrow 2$ extreme right



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

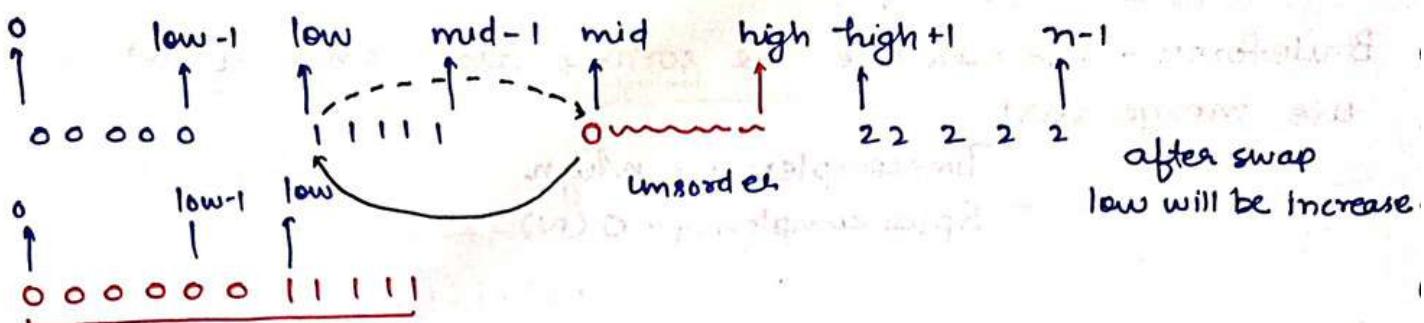


$$a[mid] = 0$$

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

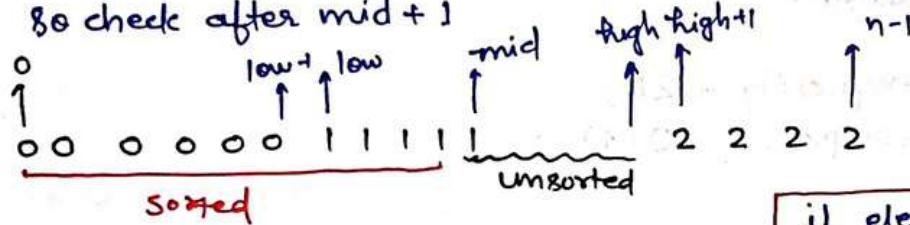
$$a[mid] = 2$$

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



So low is the correct place

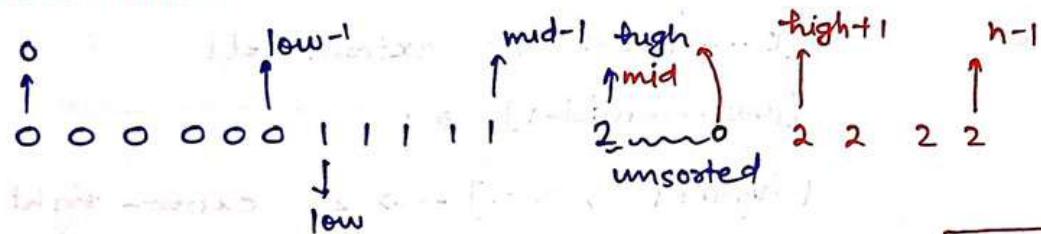
so check after $mid + 1$



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

[if element is zero]

Swap ($a[low], a[mid]$)
low++, mid++



[mid++]

[if element is 1]

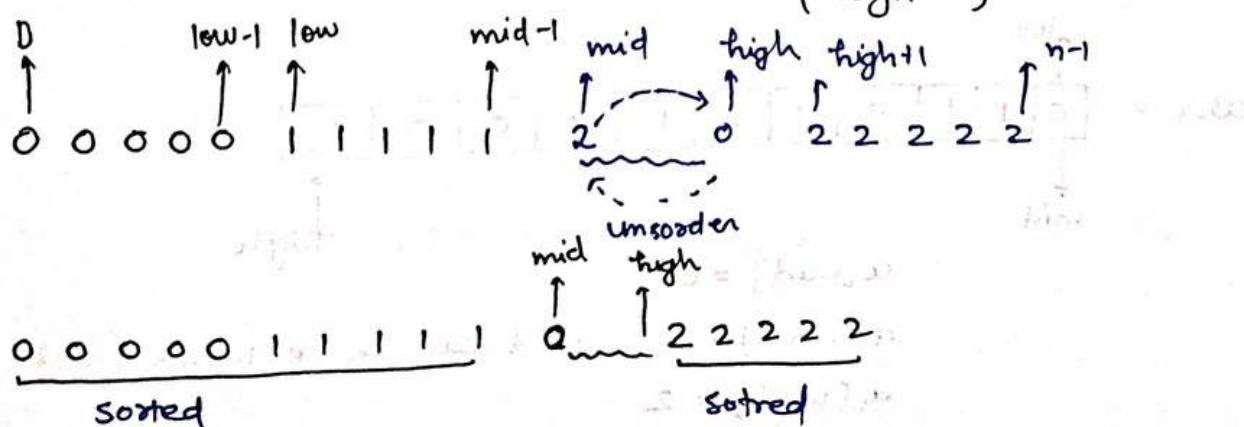
then the mid is pointing 2.

take the 2 and put the right before $high + 1$

[if it is 2]

[swap ($arr[mid], arr[high]$)

[$high--$]]



funToday (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 Buy 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 $i = 1 \rightarrow n$

② Use another $j = i + 1 \rightarrow n$

③ if $arr[j] > arr[i]$ take a diff and compare and store it to maxProfit variable.

Time complexity - $O(N^2)$

No extra space

Optimal Approach -

- ① Create max Pro 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 max Pro.

```
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 $n \times n$ 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	3
8	5	2
9	6	3

just using two loop.

Time complexity = $O(N^2)$

Space complexity = $O(N \times N)$

```
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
1	2	3	4
2	5	6	7
3	9	10	11

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

Transpose

0	1	2	3
1	5	9	13
2	2	6	10
3	3	7	11

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

the elements change -

$$[0][1] \rightarrow [1][0]$$

$$[0][2] \rightarrow [2][0]$$

$$[0][3] \rightarrow [3][0]$$

$$[1][2] \rightarrow [2][1]$$

→ Swapping behaviour

rotate (int [][] mat) {

 for (i=0; i<mat.length; i++) {

 for (j=i; j<mat[i].length; j++) {

 int temp=0;

 temp = mat[i][j];

 mat[i][j] = mat[j][i];

 mat[j][i] = temp;

Time complexity

- O(N×N)

Space - O(1)

 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];

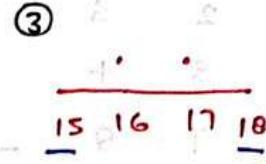
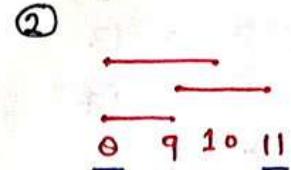
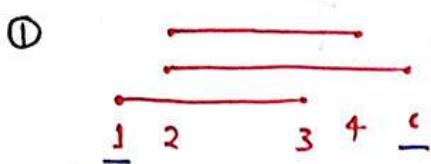
 mat[i][n-j-1] = temp;

Merge Intervals : Problem No 8

Merge overlapping Subintervals.

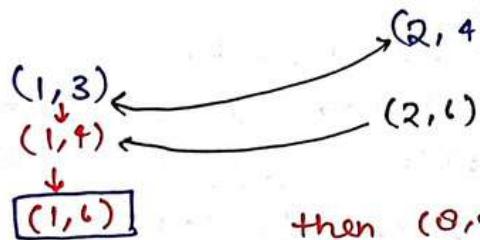
(1,3) (2,4) (8,9) (9,11) (8,10) (2,4) (15,18) (16,17)

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

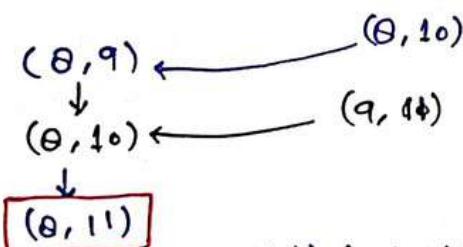


Brute force - ① Sort them up. According to first element and if the first element is equal then decide by second.

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



then (8,9) not overlapping $8 > 6$



(15,18) → (16,17) (16 < 18)

```
static List<List<Integer>> mergeInterval (int[][] arr) {
```

```
    int n = arr.length;
```

```
    Arrays.sort (arr, new Comparator<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;
    }
}

```

Time Complexity - $N \log N$
 $+ O(2N)$

ans.add(Arrays.asList(start, end));

}

return ans;

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 initially, go to (2, 4) and check, here 2 is smaller than 3. So overlapping so end update $\boxed{(1, 4)}$
- ② (2, 6) pointing and $2 < 4$ so update the end -
 $(1, 4) \rightarrow \boxed{(1, 6)}$
- 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.

$\boxed{8, 10}$ (9, 11) compare with 10 > 9 overlap.

update the end.

$\boxed{8, 11}$ ✓

new interval.
 $\boxed{15, 18}$ ✓

```

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;
}

```

Merge Sorted Array : Problem 9

Brute force - Using extra space -

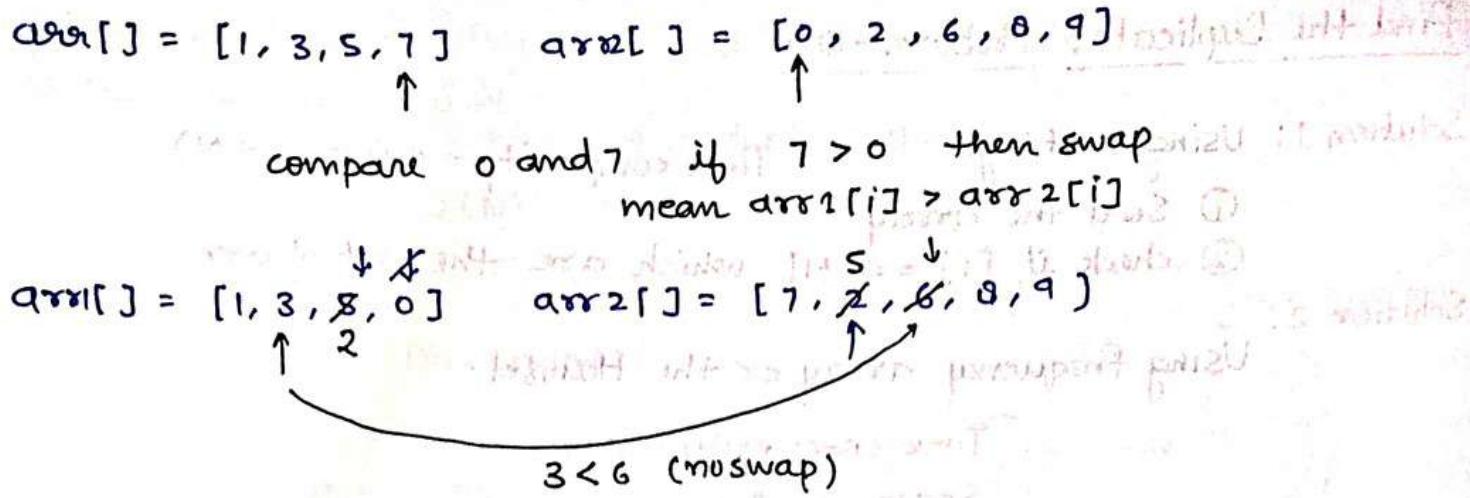
- ① Declare an array of size $n+m$.
- ② If $\text{arr1}[\text{left}] < \text{arr2}[\text{right}]$ insert $\text{arr1}[\text{left}]$ and $\text{left}++$;
- ③ If $\text{arr1}[\text{left}] \geq \text{arr2}[\text{right}]$ insert $\text{arr2}[\text{right}]$ and $\text{right}++$:
- ④ $\text{arr1}[\text{left}] == \text{arr2}[\text{right}]$ 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 - $O(N+M) + O(N+M)$

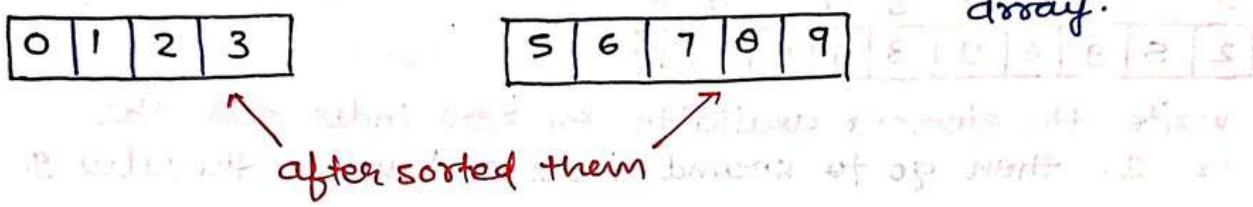
for filling the elements of
 arr1 and arr2 to arr3

↓
 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 || 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);
 Arrays.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 - $O(N \log N + N)$

- 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)

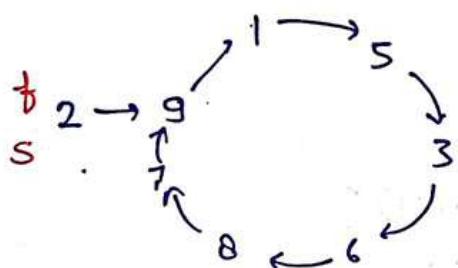
Solution 3: Linked List cycle method.

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

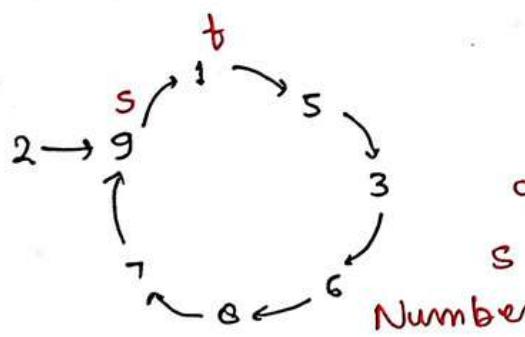
first write the element available on zero index and the value is 2. then go to second index and write the value 9.

2 →

Similarly create



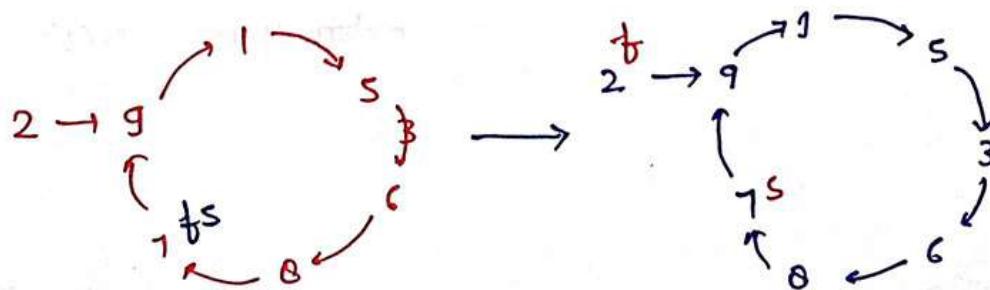
f will move 2 steps.
s will move 1 step.



after a certain time the sand will lying on the same
ber. 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) {

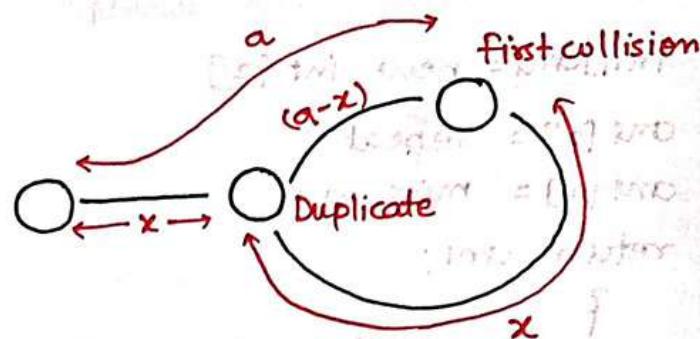
 slow = nums[slow];

 fast = nums[fast];

 }

 return slow;

}



Missing and Repeating Number : Problem: 11

Bruteforce -

- ① Run a loop $i = 1 \rightarrow n$
- ② for each integer i we will count the occurrence.
- ③ Store those elements that have 2 and 0 occurrence.

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], $n=6$

$x \rightarrow$ repeating, $y \rightarrow$ missing

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

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

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

$$= S - Sn$$

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

\Rightarrow If we open bracket then

$$\Rightarrow 1-S = (x-y) \text{ on observing}$$

$$\Rightarrow -4$$

$$x-y = -4 \quad \text{--- ①}$$

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

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

$$= 1 - 25 \quad \text{for square of first } N \text{ natural no.} \rightarrow \text{sum}$$

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

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

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

$$x+y = +6$$

$$\text{Now } x+y = +6$$

$$x-y = -4$$

$$\underline{2x = 2}$$

$$x = 1$$

$$x+y = 6$$

so

$$y = 6 - x = 6 - 1 \\ = 5$$

Time complexity - $O(N)$

Space complexity - $O(1)$

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

fun missing(arr){

long n = arr.length;

$$\log SN = \frac{(n * (n+1))}{2};$$

$$\text{long } S2N = \frac{(n * (n+1) * (2n+1))}{6};$$

$$S = 0, S2 = 0;$$

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

$$S = S + arr[i];$$

$$S2 = (\text{long}) arr[i] * (\text{long}) arr[i];$$

$$\text{long val1} = S - SN;$$

$$\text{long val2} = S2 - S2N;$$

$$\text{val2} = \frac{\text{val2}}{\text{val1}};$$

$$\text{long } x = (\text{val1} + \text{val2}) / 2;$$

$$\text{long } y = x - \text{val1};$$

int[] ans = {(int)x, (int)y};

return ans;

[0, 1, 2, 3, 4, 5]

[0, 1, 2, 3, 4]

In above step is address error it is because both

the arrays are pointing to same heap memory

else if any of the array not use heap memory then

then there will be no address error as this thing is absent in memory

so we can use direct pointer to access the array

and also we can use pointer to access the array

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 $i < j$

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

② $arr[i] > arr[j]$

pairs = 0

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

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

pairs = 7

pairs = 5

(5, 1) (5, 3) (5, 2) (5, 4) (3, 2)
(3, 1) (2, 1)

Brute force Approach -

fun. Inversions(int arr[], int n){

int cnt = 0;

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

for (j=i+1; j<n; j++) {

if (arr[i] > arr[j]) cnt++;

Time complexity - $O(N^2)$

Space complexity - $O(1)$

}

return cnt;

}

Optimize solution -

Intuition: take the array which are sorted.

[2, 3, 5, 6]
↑ ↑ ↑ ↑

[2, 2, 4, 4, 8]
↑ ↑ ↑ ↑ ↑

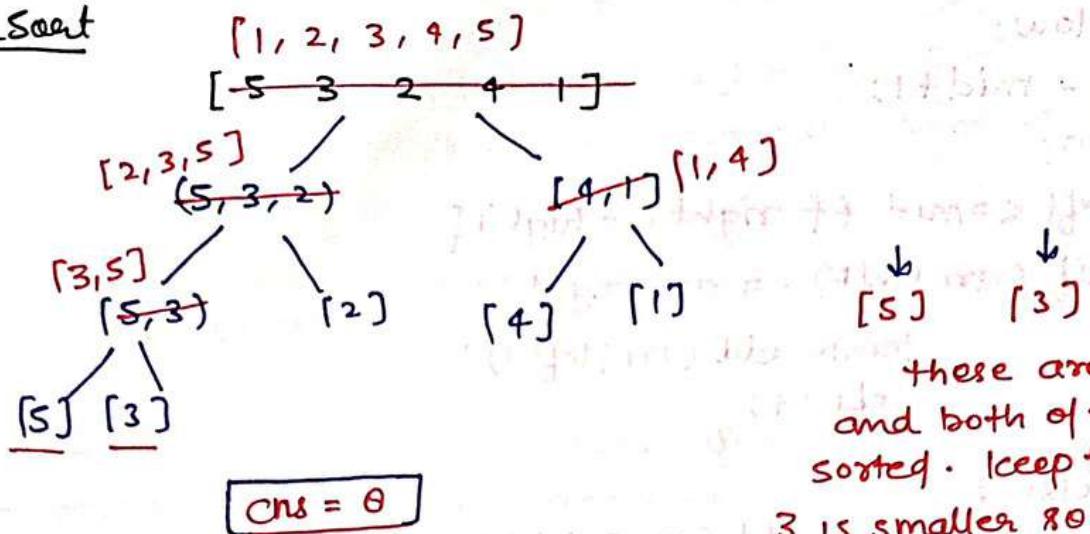
Here 3 > 2 so it can create a pair with 2

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



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.

we have

$[4]$ $[1]$

1 can form a pair with

4. ans+1 to ans.

$[3, 5]$

$[3, 5] \quad [2]$

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

$[2, 3, 5]$

$[2, 3, 5] \quad [1, 4]$

$\cancel{ans} + 1 + 1 + 1$ because $1 < 2$
 $ans + 1$ because $4 < 5$

$\begin{matrix} \text{left} & & \text{right} \\ [2, 3, 5] & & [1, 4] \\ \uparrow & & \uparrow \\ \text{low} & \text{mid} & \text{mid} + 1 \\ & & \downarrow \end{matrix}$ high

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

```
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 && 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 - $O(n \log N)$
Extraspace - $O(N)$

Search in a Sorted 2D Matrix - Problem - 13

Given $m \times n$ 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.

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

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

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

Space complexity - $O(1)$

Optimal Solution - Binary search

	0	1	2	3
0	1	3	5	7
1	10	11	16	20
2	23	30	34	60

$$n = 3 \\ m = 4 \\ \text{target} = 30$$

$$0 - (n \times m - 1) \\ [0 - 11] \rightarrow \text{index}$$

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

$$0 \quad \text{mid} \quad 11 \\ \text{low} \quad 5 \quad \text{high} \\ \text{index}$$

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.

$$6 \quad \text{mid} \quad 11 \\ \text{low} \quad \frac{6+11}{2} = 8 \quad \text{high} \\ \text{Index}$$

$$8/4 = 2$$

$$8 \% 4 = 0$$

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

$$\text{low} = \text{mid} + 1$$

$$\text{Now, low} = 9, \text{high} = 11$$

9 mid high
low ⑩ index: 11

$$10/4 = 2$$

$$10 \% 4 = 2$$

(2, 2) stands for 34 and here we can see $34 > 30 \text{ so}$

$$\text{high} = \text{mid} - 1 \\ = 9$$

g ← mid
↑
low high

80

$$9/4 = 2$$

$$9 \% 4 = 1$$

so (2, 1) belongs to 30 matched.

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

$$\text{Time complexity} = N \log_2(m)$$

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

$$\text{target} = 25$$

- 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.

$$25 < 40 \text{ (so it correspond to left side)}$$

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 reach 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) { Time complexity - $\log(N \times M)$

 return true;

 ?

 if (mat[i][j] > x)

 j--;

 else

 i++;

 ?

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

Brute force 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 -

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

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

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

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

$$256^1 = \underline{256} \times (256)^0$$

① $[n \% 2 == 0] \rightarrow [x \times x]$ Time complexity - $\log_2(N)$
 $n/2$

② $[n \% 2 == 1] \rightarrow \text{ans} = \text{ans} \times x$
 $n = n - 1$

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 occurs more than $N/2$ times:

$N = 3$, $\text{nums}[] = [3, 2, 3]$

result = 3

$N = 7$, $\text{nums} = [2, 2, 1, 1, 1, 2, 2]$

result = 2

Brute force Approach -

- ① We run a loop that select a element one by one.
- ② Now each element run another loop and count the occurrence in the array.
- ③ If any element occurs $N/2$ times so 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 hash map and (key, value) pair.

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

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

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

Space - $O(N)$

Optimal Approach : Moore's Voting Approach -

$\frac{7-3}{5-2, 1-1}$ $\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$ $7 \quad 7 \quad 5 \quad 7 \quad 5 \quad 1$ $x \cancel{x} \cancel{x} \cancel{x} \cancel{x} \cancel{x} \cancel{x}$ $\text{cnt} = \emptyset \cancel{x} \cancel{x} \cancel{x} \cancel{x}$ $\text{ele} = \emptyset \cancel{7} \cancel{8} \cancel{8} \cancel{5}$	$\frac{5-1}{7-1}$ $\downarrow \quad \downarrow$ $5 \quad 7$ $\cancel{x} \cancel{x}$ $\text{cnt} = \emptyset \cancel{x} \cancel{x}$	$\frac{5-2}{7-2}$ $\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$ $5 \quad 5 \quad 7 \quad 7$ $\cancel{x} \cancel{x} \cancel{x} \cancel{x}$	$\frac{4}{5-4}$ $\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$ $5 \quad 5 \quad 5 \quad 5$
--	--	--	---

```

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 && ele1 != num[i]) {

 cnt1 = 1;

 ele1 = num[i];

}

 else if (cnt2 == 0 && ele1 == num[i]) {

 cnt2 = 1;

 ele2 = num[i];

 }

 else if (ele1 == num[i]) cnt1++;

 else if (ele2 == num[i]) cnt2++;

 else { cnt1--;

 cnt2--;

 }

 List<Integer> lt = new ArrayList<>();

 cnt1 = 0, cnt2 = 0;

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

 if (num[i] == ele1) cnt1++;

 if (num[i] == ele2) cnt2++;

 }

 int min = (int)(n/2)+1;

 if (cnt1 >= min) lt.add(ele1);

 if (cnt2 >= min) lt.add(ele2);

we checks $ele_2 \neq num[i]$

because it can be possible that

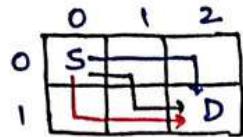
if we want to put the val of $num[i]$ in el_2 which is already taken by el_1 .

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

Space complexity - $O(1)$

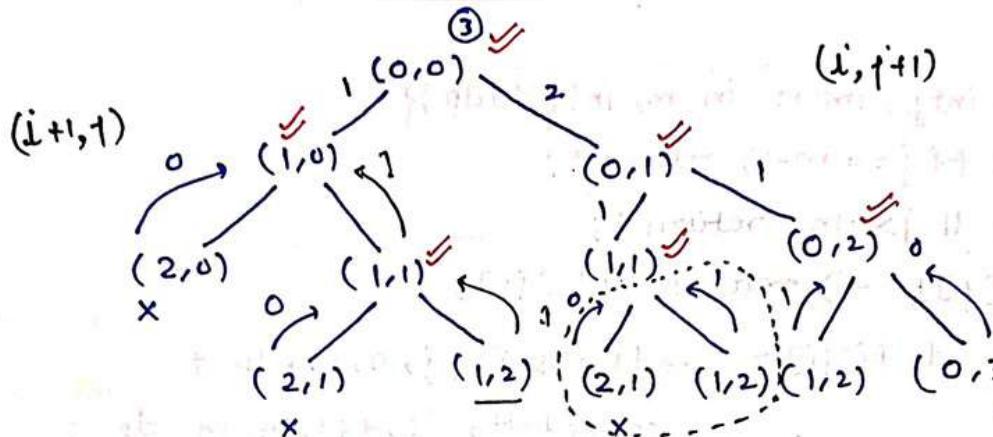
Unique Paths : Problem 17

grid of $m \times n$ 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 move only right and bottom.



Total 3 paths

Brute force Solution : Recursion



$i >= n$
 $j >= m$

Basecase $\rightarrow 0$

$i == n-1 \quad j == m-1 \quad \{ \text{return } 1 \}$

```
int 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 countpath (i+1, j) + countpath (i, j+1);  
}
```

Hashtable : Optimal Solution

DP	0	1	2
0	x^3	$-x^2$	$-x^1$
1	$-x^1$	$-x^1$	-1

if the value of a recursion call is available in hashtable 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];
}

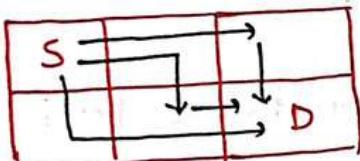
countPaths (int i, int j, int n, int m, int [][] dp) {
    if (i == n-1 && j == m-1) return 1;
    if (i >= n || j >= m) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    else return dp[i][j] = countPaths (i+1, j, n, m, dp) +
        countPaths (i, j+1, n, m, dp);
}

```

Time complexity - $O(n*m)$

Space complexity - $O(n*m)$

We can also solve it using combination -



$\begin{matrix} RRD \\ DRR \\ RDR \end{matrix}$

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

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

${}^3 C_2$
 $\begin{bmatrix} R R B \\ R B R \\ B R R \end{bmatrix}$

$$\begin{aligned} {}^{m+n-2} C_{m-1} &= 3 \\ {}^{m+n-2} C_{n-1} &= 3 \end{aligned}$$

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

```

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;

```

Reverse Pairs - Problem - 18

$\text{arr}[] = [40, 25, 19, 12, 9, 6, 2]$

Conditions: $i < j$ if $\text{arr}[i] > 2 * \text{arr}[j]$

O/P - 6	(6, 2)
+	(9, 2)
3	(12, 2)
+	(19, 2)
3	(25, 2)
+	(40, 2)
"	
15	

Similar to count Inversion
Slight modification



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

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

This function countPairs is called in the mergeSort.

then create a function countPairs and cut the pairs.

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

add the cut 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 divide

Space complexity - $O(N)$

for distorting the array.

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

$N = 5$, $\text{arr}[] = [2, 6, 5, 0, 11]$, target - 14

OP - Yes on adding 6+8 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 $(\text{arr}[i] + \text{arr}[j] == \text{sum})$ 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 hashmap or not.

TwoSum(n , arr , target) {

 HashMap<int, int> map;

 for ($i = 0 \rightarrow n$) {

 int num = $\text{arr}[i]$;

 if (map.containskey($\text{target} - \text{num}$)) {

 return 'Yes';

 }

 map.put($\text{arr}[i]$, 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 $j = n-1$

if we get the target
simply return

check $\text{arr}[i] + \text{arr}[j] > \text{target}$
then $j--$ otherwise $i++$.

twoSum(arr , n , target) {

 Sort(arr);

 int $i = 0$, $j = n-1$;

 while ($i < j$) {

 int sum = $\text{arr}[i] + \text{arr}[j]$;

 if ($\text{sum} == \text{target}$) return 'yes' ;

 else if ($\text{sum} < \text{target}$) ~~return~~ left++;

 else right-- ;

}

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

Space complexity - $O(1)$.

4 Sum Problem: Problem - 20

$$\text{arr}[a] + \text{arr}[b] + \text{arr}[c] + \text{arr}[d] == \text{target}$$

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 $\text{arr}[i] + \text{arr}[j] + \text{arr}[k] + \text{arr}[l] == \text{target}$ 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 declare 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 - $(\text{arr}[i] + \text{arr}[j] + \text{arr}[k])$

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, num8) {
```

```
    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 += num[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) num8[k]);
```

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

$M = \text{no of elements in set.}$

Space complexity - $O(2 * \text{no of quadruplets}) + O(N)$

```

List<List<Integer> lt = new ArrayList<>();
lt.add(ans);
return lt;
}
}

```

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

[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 (int[] arr, int target) {
    int n = arr.length;
    List<List<Integer>> ans();
    Arrays.sort(arr);
    for (int i=0; i<n; i++) {
        if (i>0 && arr[i]==arr[i-1]) continue;
        for (int 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 += arr[j] + arr[k] + arr[l];
                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 && arr[k]==arr[k-1]) k++;
                while (k<l && arr[l]==arr[l+1]) l--;
            }
            else if (sum<target) k++;
            else l--;
        }
    }
}

```

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

Space Complexity - $O(\text{No of quatos})$

Longest Consecutive Sequence in an Array: Problem - 21

nums[] = [100, 4, 200, 1, 2, 3], Q/P = 4

① length of longest consecutive sequence.

Brute force. Sorting

```

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

```

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

Optimal Approach: Using HashSet

① put every thing into the set.

```

fun. longestConsecutive (arr) {
    if (arr.length == 0) return 0; longestStreak = 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))
                currentNum += 1;
                currentStreak += 1;
        }
    }
}

```

Time complexity - $O(N) \approx O(3N)$
Space complexity - $O(N)$

$$\text{longestStreak} = \text{Math.max} (\text{longestStreak}, \text{currStreak})$$

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

$A[] = \begin{matrix} 15 \\ -2 \\ 2 \\ -8 \\ 1 \\ 7 \\ 10 \\ 23 \end{matrix}$

$$\text{sum} = 0 + 15 - 2 = 13 + 2 = 15 \neq 0$$

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) { // because i = -1 so we using i < n-1

i++;

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;

0, 4
7, 3
15, 2
13, 1
15, 0

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}

one = 4

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

i ↓ i → the subarray is from i to j

Brute force -

```
int cut = 0;
```

```
for (i=0; i<n; i++) {
```

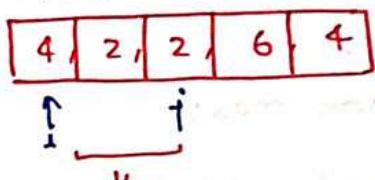
```
    for (j=i; j<n; j++) {
```

```
        int XOR = 0;
```

```
        for (k=i; k<=j; k++) {
```

```
            XOR = XOR ^ arr[k];
```

if (XOR == K) cut++;



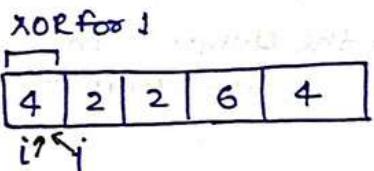
```
}
```

```
cut;
```

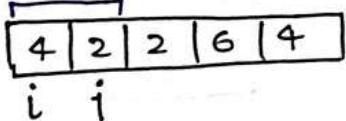
Time complexity - $\approx O(N^3)$

Space complexity - $O(1)$

Better Solution -



XOR for i



```
for (i=0; i<n; i++)
```

```
XOR = 0
```

```
for (j=i; j<n; j++) {
```

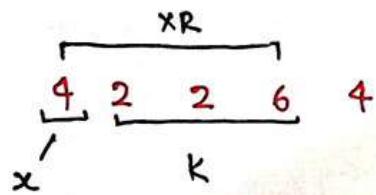
```
XOR = XOR ^ arr[j];
```

if (XOR == K) cut++;

Time complexity - $O(N^2)$

Space complexity - $O(1)$

Optimal Solution : Using Hashing



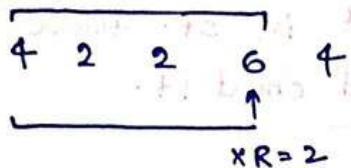
$$XR = x \wedge K$$

take the $\wedge K$ in both sides.

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

$$x = XR \wedge K$$

K is the XOR we are looking for



so there's a subarray ending at 6 of having the XOR of K

$$x = XR \wedge K$$

$$= 2 \wedge 6 = \underline{4}$$

do we have from some who is giving me a 4

$$arr[] = [\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow] \\ [4 \quad 2 \quad 2 \quad 6 \quad 4]$$

$$XOR = \emptyset \ 4 \wedge 2 = 6 \wedge 2 = 4 \wedge 6 = 2 \wedge 4 = 6$$

int $x = XOR \wedge K = 4 \wedge 6 = 2$ (we don't have 2 in Hashmap)

$$cnt = \emptyset \ 1 + 2 + 1 = \underline{4}$$

$$x = 6 \wedge 6 = 0$$

(2, 1)
(6, 1)
(4, 2)
(6, 1)

put initially.

Hashmap.

$$XR = 0;$$

Hashmap<>();

map.put(XR, 1);

int cnt = 0;

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

$$XR = XR \wedge arr[i];$$

$$\text{int } x = XR \wedge K;$$

if (map.contains(x)) cnt += map.get(x);

if (map.containsKey(XR)) map.put(XR, map.get(XR)+1);

else map.put(XR, 1);

$$x = 2 \wedge 6 \\ = 4$$

$$x = 6 \wedge 6 \\ = 0$$

Time complexity - $O(N)$

Space complexity - $O(N)$

longest Substring without repeat : Problem - 24

S = a b c a b c bb 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.

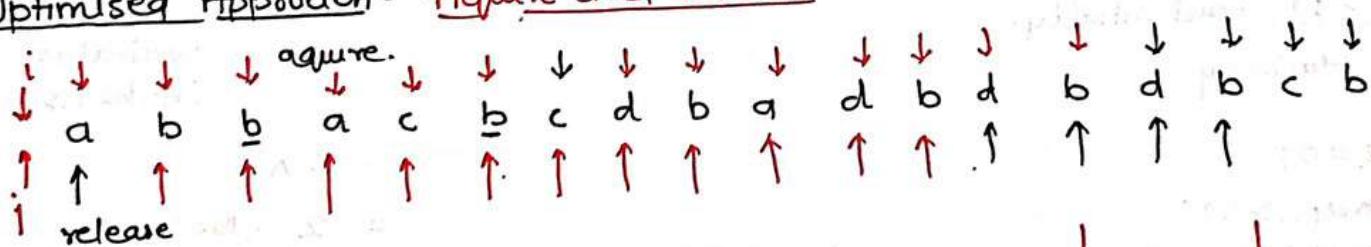
```

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 - quire and release



x ab
ba c
cd ba

quire until you became invalid.

release until you valid again

a, x o	x z x x o
d, x z t	x z x x o
c, x z x o	x z x x o
a, x	x z x x o
b, x z x x	x z x x o
a, x o x o	x z x x o

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) {
        // acquire
        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 = $3 \rightarrow 6 \rightarrow 8 \rightarrow 10$, reverse = $10 \rightarrow 8 \rightarrow 6 \rightarrow 3$
 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);`

`head.next.next = head;`

`head.next = null;`

`return curr;`

}

Middle element of linkedlist : Problem - 26

head = $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

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 moves 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

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

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

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 curr = 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 Nth from end of Linkedlist -

```

removeNthFromEnd ( ListNode head, int n ) {
    ListNode start = new ListNode ( 0 );
    start . next = head;
    ListNode f = head; → start;
    ListNode s = head; → start;

    for ( i = 1 ; i <= n ; i ++ )
        f = f . next;
    Time complexity - O(N)
    while ( f . next != null ) {
        fast = fast . next;
        slow = slow . next;
    }
    slow . next = slow . next . next;
    return start . next;
    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.

num1 = [2, 4, 3]

num2 = [5, 6, 4]

sum = [0, 0, 1]

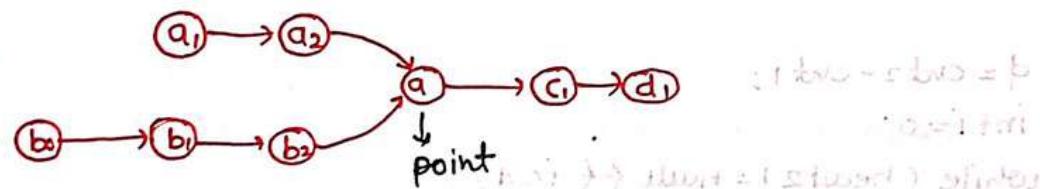
Case : 1 When one list is longer than other.
 Case : 2 When one list is null:
 Case : 3 the sum could have an extra / carry / carry of one at the end.

```

List Node dummy = new Node(0);
List Node 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.next;
  sum+=carry;
  carry = sum/10;
  List Node node = new List Node (sum%10);
  temp.next = node;
  temp = temp.next;
}
  
```

Time complexity - $\max(0, N, M)$
 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 them 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: $O(N+M) \rightarrow$ 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. archive.
- ④ Move both pointers.

getIntersection (head A, head B) {

 Node curr1 = head A;

 Node curr2 = head B;

 int current1 = 0;

 int cnt2 = 0;

 int d = 0;

 while (curr1 != null) cnt1++;

 curr1 = curr1.next;

 while (curr2 != null) cnt2++;

 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;

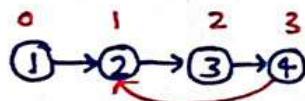
 head2 = head2.next;

 return head1;

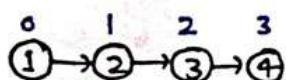
Time complexity - $O(2 \max(l_1, l_2)) + O(\text{abs}(l_1 - l_2)) + O(\min(l_1, l_2))$

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;
  
```

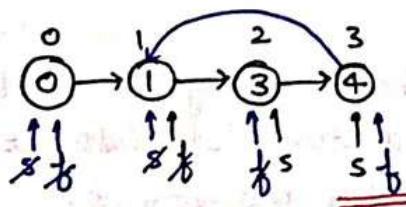
Time complexity - $O(N)$

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 = $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$

K = 3.

Output - $\underline{3} \rightarrow \underline{2} \rightarrow \underline{1} \rightarrow \underline{6} \rightarrow \underline{5} \rightarrow \underline{4} \rightarrow \underline{7} \rightarrow \underline{8}$

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).

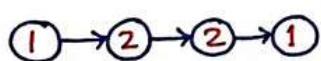
```

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.



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 mid
- ③ Compare and check

isPalindrome (Node head) {

 if (head == null || head.next == null) return true;

 Node slow = head;

 Node fast = head;

 while (fast != null && fast.next != null)

 slow = slow.next;

 fast = fast.next.next;

 slow.next = reverse (slow.next); // write the code for reverse

 slow = slow.next;

 Node dummy = head;

 while (slow != null)

 if (dummy.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.



tail connect with index 2.

Brute force - 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 head.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;
ListNode f = head;
while (f != null && f.next != null) {
    s = s.next;
    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;

```

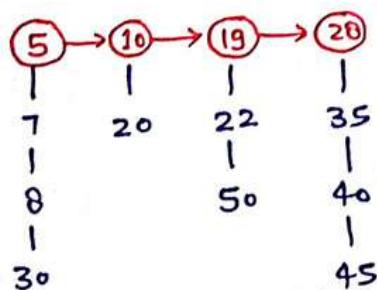
Time complexity - $O(N)$

Flattening 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 - $5 \rightarrow 7 \rightarrow 0 \rightarrow 10 \rightarrow 19 \rightarrow 20 \rightarrow 22 \rightarrow 28 \rightarrow$

$30 \rightarrow 35 \rightarrow 40 \rightarrow 45 \rightarrow 50$.

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 temp.bottom = b;

 return res.bottom;

- modified linked

Node flatten (Node root) {

 if (root == null || root.next == null) return root;

 root.next = flatten (root.next);

 root = mergesorted (root, root.next);

 return root;

}

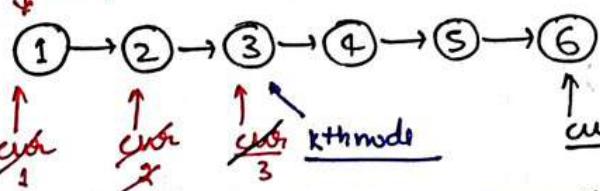
(k=3)

linked list

Rotatelist Problem - 37

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

↓ head



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

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

Bruteforce 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 - $O(\text{No of nodes} * k)$

Optimal Solution -

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

```
Node cur = head;  
int cut = 1;  
while (cut < k && cur != null) {  
    cut++;  
    cur = cur.next;  
}  
if (cur == null)  
    return head;
```

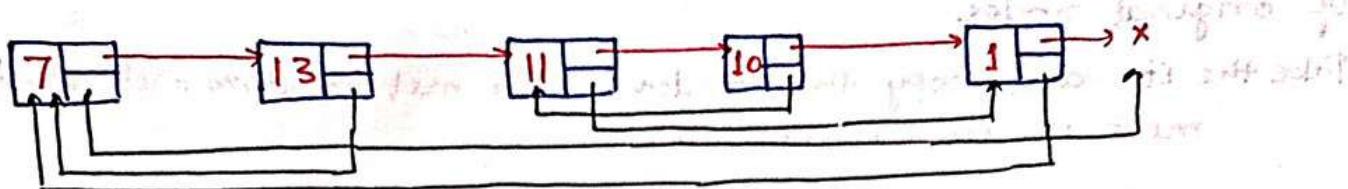
```
Node kthnode = cur;  
while (cur.next != null) {  
    cur = cur.next;  
}  
cur.next = head;  
head = kthnode.next;  
kthnode.next = null;  
return head;
```

Time complexity -

$$\begin{aligned}& O(\text{len of list}) + O(\text{length of list} - (\text{length of list} / k)) \\& = O(N).\end{aligned}$$

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.random != 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. $itr.next.random = itr.random.next$. 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;
```

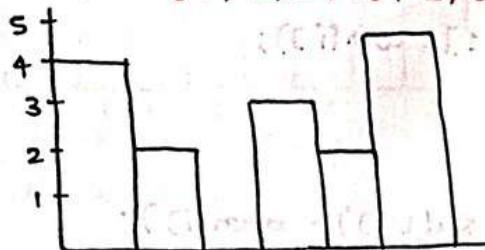
```
        itr = fast;
```

```
}
```

```
    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; // initialising j to 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 - $O(N * M)$.

Better Solution - Take 2 array prefix and suffix and precompute the left and right for each index beforehand. Then formula -

$\min(\text{prefix}[i], \text{suffix}[i] - \text{arr}[i])$ to compute 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], 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 (prefix[i], suffix[i]) - arr[i];
return watertrap;

```

Time complexity - $O(3 * N)$

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

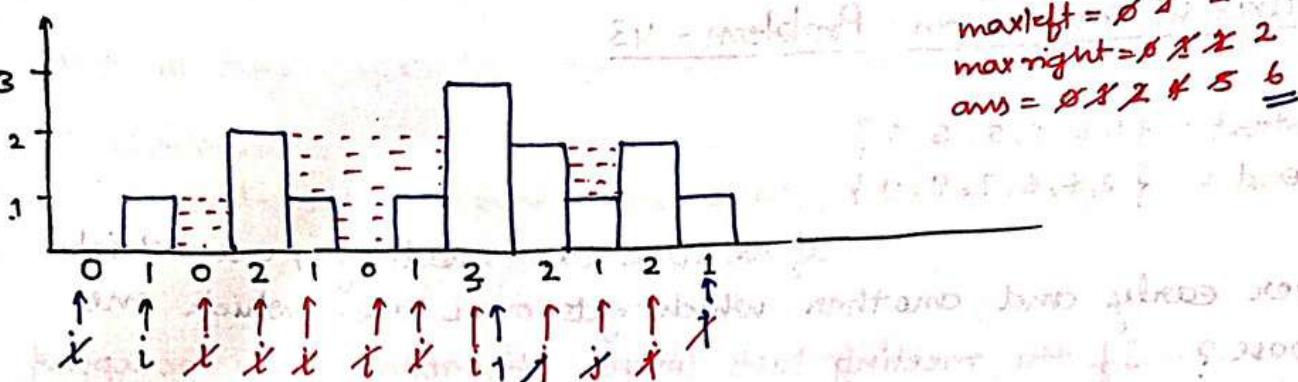
Two Pointer Approach : Optimized solution.

```

trap(arr) {
    n = arr.length;
    int left = 0, int right = n-1;
    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[right];
            }
            else
                res += maxRight - height[right];
        }
        right--;
    }
    return res;
}

```

Time complexity - $O(N)$.



Problem-41 : Remove Duplicates of sorted array

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

[1, 2, 3 - - - -]

Bruteforce Approach

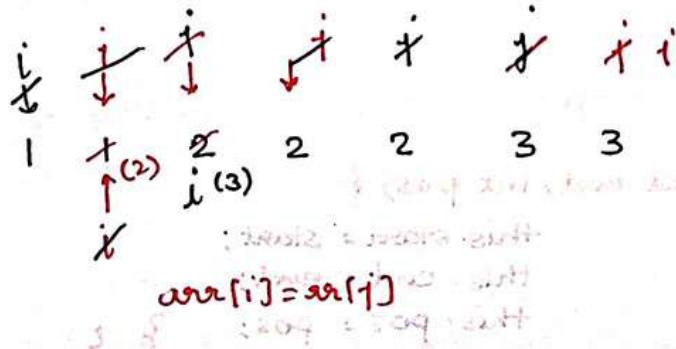
Used hashset for storing the values.

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

Space complexity - $O(N)$

Optimal Approach

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



N meetings in one room Problem - 43

$$N = 6, \text{ start} = \{1, 3, 0, 5, 8, 5\}$$

$$\text{end} = \{2, 4, 6, 7, 9, 9\}$$

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 other hand 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't happen.

otherwise skip and move ahead.

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

initially start will be 0 then first meeting can be happen

$$\text{start} = 1$$

$$\text{end} = 2$$

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.start = 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 meetingComparator 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 (meet lt.get(i).start > limit) {
```

```
            limit = meet 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;

} else if (platform == res) {

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 deadline 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	2	3	4	5

you can create size 4 array

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

$$\text{profit} = 40 + 20 = 60$$

1	2	3	4	5

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;

 countJob++;

 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.

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

$$----- \quad 2 = \frac{60}{10} = 6$$

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

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

```

fun main(W, arr, n) {
    Arrays.sort(arr, new Comparator() {
        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)
}
break;
}
}

```

initially $W = 50$, $W \geq 20$
 so we can take
 value = 100, then $50 - 20 = 30$

$$30 \geq 10$$

value = 160, then $30 - 10 = 20$

$20 \geq 10$, then

$$\text{value} = 160 + \left(\frac{120}{30} \right) \times 10 \text{ then } W = 0 \\ = 240$$

```
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 20 ss of note , total - 2

② $V = 121$

we need a 100Rs and 20 and 1, total - 3

```
fun main() {
```

```
    ArrayList<Int> ans();
```

```
    int coins[] = [1, 2, 5, 10, 20, 50, 100, 500, 1000];
```

```
    int n = coins.length;
```

```
    for (i = n - 1; i >= 0; i--) {
```

```
        while ( $V \geq coins[i]$ ) {
```

```
            V = V - coins[i];
```

Time complexity - $O(V)$

```
ans.add(coins[i]);
```

Space complexity - $O(1)$.

```
}
```

```
}
```

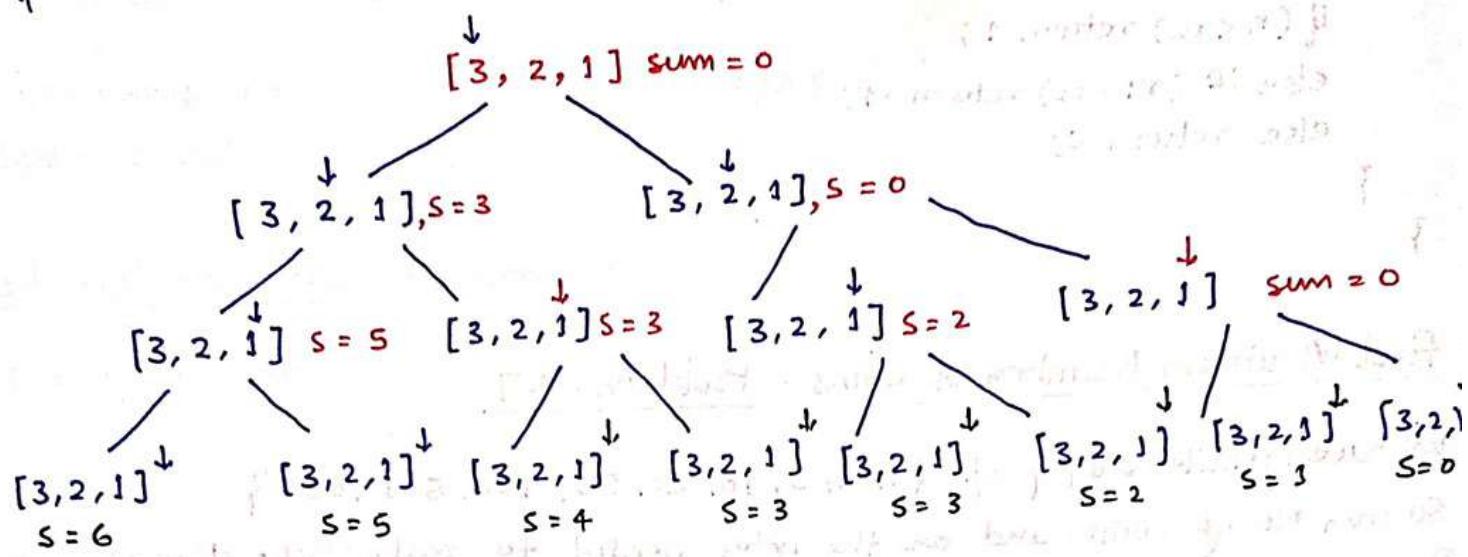
```
ans.size();
```

$O(\sqrt{V})$ - polynomial time

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

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

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



Powerset → Bruteforce

```
public static List<List<Integer>> generatePowerset (nums) {
```

```
    List<List<Integer>> ans = new ArrayList<>();
    backtrack (nums, new ArrayList<>(), 0, 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);
```

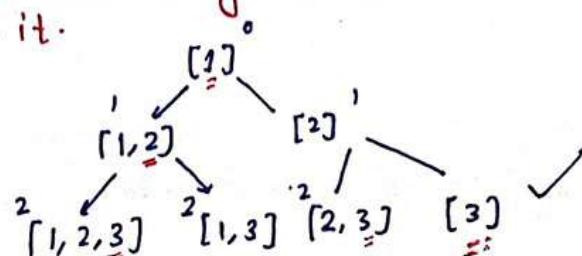
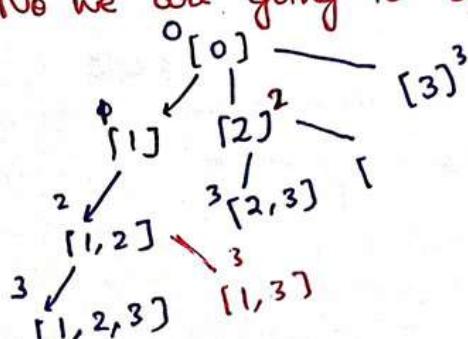
}

}

Time complexity - $2^n * N$

we are using a extra N space

Now 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> subset(arr, N) {
    ArrayList<> ans;
    helper(0, 0, arr, N, ans);
    Collections.sort(subset); // ans;
    return ans;
}

```

Time complexity - 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, int ind, ds, res, avg) {
    if (ind == arr.length) {
        collections.sort(ds);
        res.add(ds.toString());
        return;
    }
    ds.add(arr[i]);
    fun(arr, ind+1, ds, res, avg);
    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();
    findSubsets(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 || arr[i] != arr[i-1]) continue;
        ds.add(arr[i]);
        findSubsets(i+1, arr, ds, ans);
        ds.remove(ds.size() - 1);
    }
}

```

Combination Sum: 1

Problem - S1

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

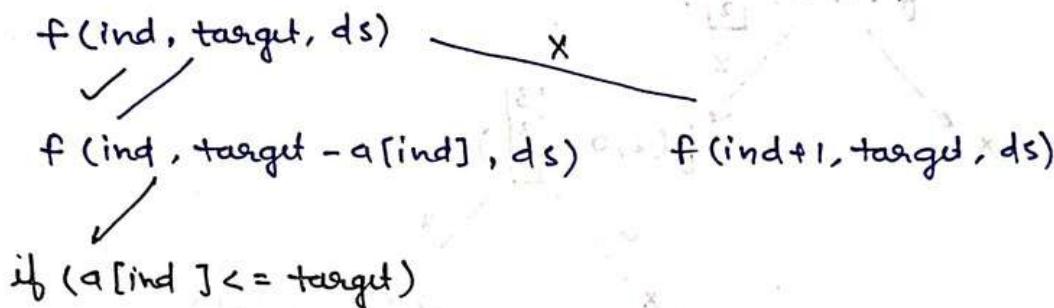
$\text{arr}[] = [2, 3, 6, 7]$, target = 7

output = $\underline{[2, 2, 3]}, [7]$

↓
2 is used 2 times.

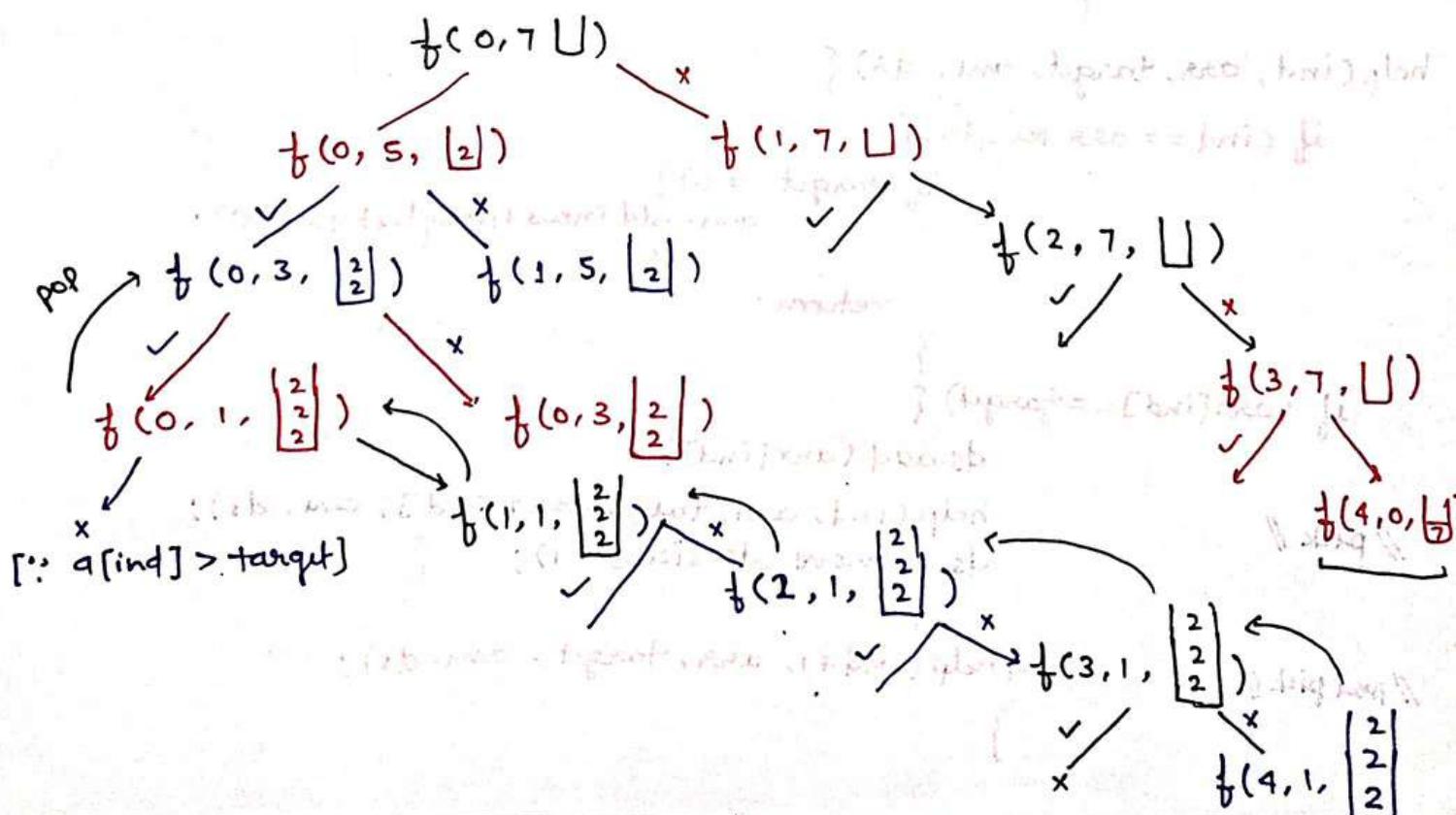
Intuition - for Questions like printing combinations or subsequences.
the first thing strike - recursion.

pick and not pick.



if pick the element, again comeback at the same index and as multiple occurrences of the same element is possible so the target reduces the value. $\text{target} - \text{arr}[\text{ind}]$.

If not pick move forward to next index. while backtracking pop the last element. keep repeating process while $\text{index} < \text{size}$.



$f(0, 7, \boxed{})$



k is the avg length.

$f(0, 5, \boxed{})$



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

Space complexity - $O(k * x)$

$f(0, 3, \boxed{})$

$f(0, 5, \boxed{})$

$f(0+1, 3, \boxed{})$

$f(1, 0, \boxed{})$

$f(2, 0, \boxed{})$

✓
X

$f(3, 0, \boxed{})$

X

$f(4, 0, \boxed{})$

X

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]);

 help(ind, arr, target - arr[ind], ans, ds);

 ds.remove(ds.size() - 1);

}

//pick//

//not pick//

 findHelp(ind + 1, arr, target, ans, ds);

}

Main Combination Sum II - Unique Combinations

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

Output = [[1, 1, 6], [1, 2, 5], [1, 7], [2, 4]]

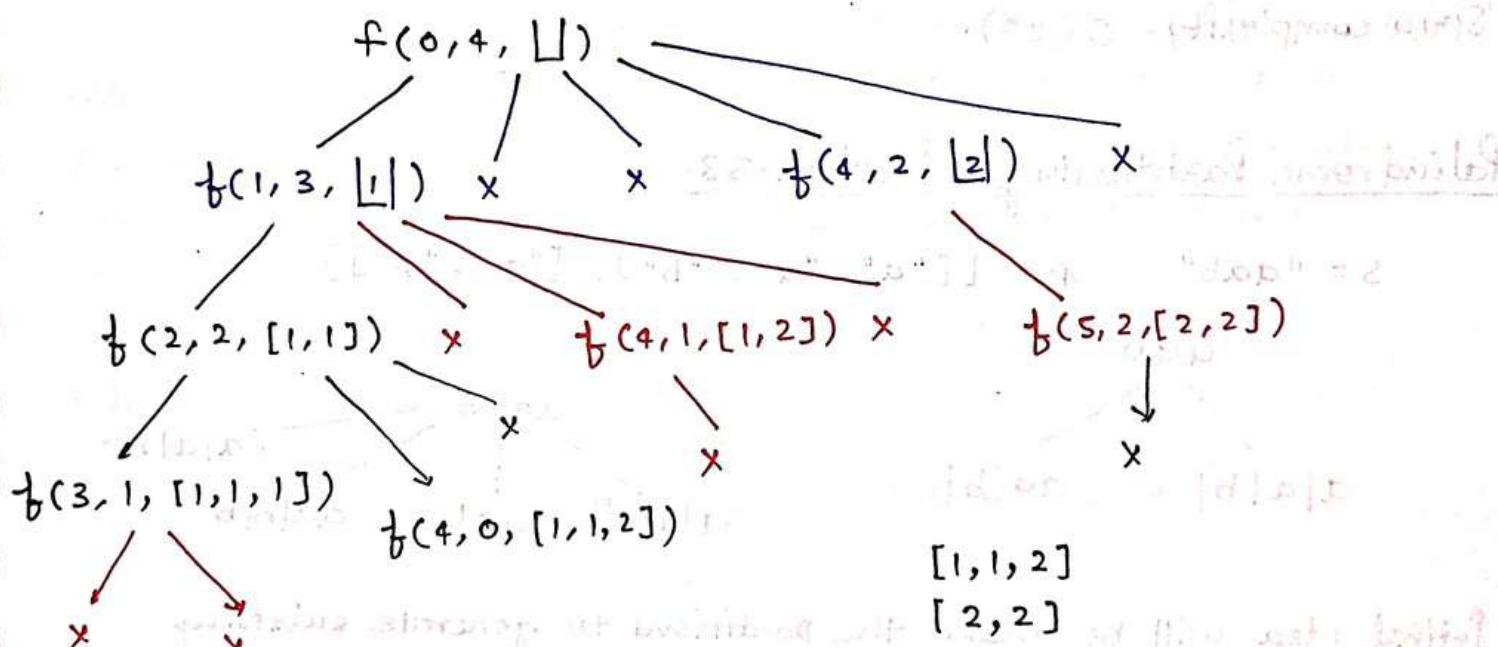
```

f(idx, target, ds, ans)
    if (arr[i] > target)
        break;
    ds.add(arr[i]);
    f(i+1, target - arr[i], ds, ans);
    ds.remove(arr[i]);
}

// Base Case
if (target == 0)
    ans.add(ds);

```

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, arr, ds) {
    if (target == 0) {
        arr.add (new ArrayList<String>(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], arr, ds);
        ds.remove(ds.size() - 1);
    }
}

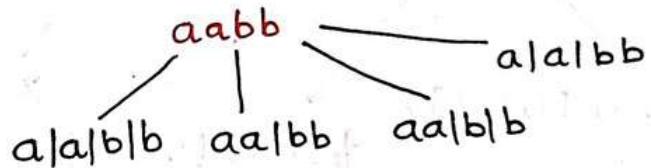
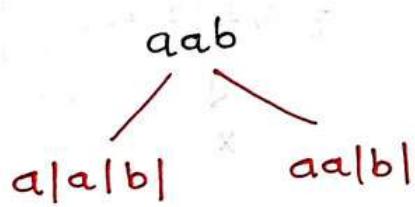
```

Time complexity - $O(2^n)$

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

Palindrome Partitioning - Problem-53.

$s = "aab"$ op. $\{["a", "a", "b"], ["aa", "b"]\}$

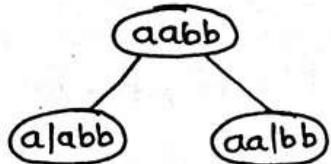


Initial idea will be make the partitions to generate substring and check if the substring generated out of the partition will be palindrome.

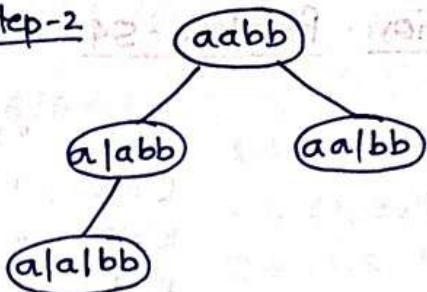
Generate every substring and checking for palindrome in every step.

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

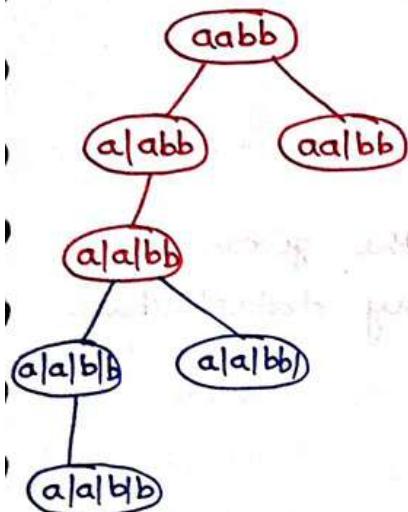
Step-1.



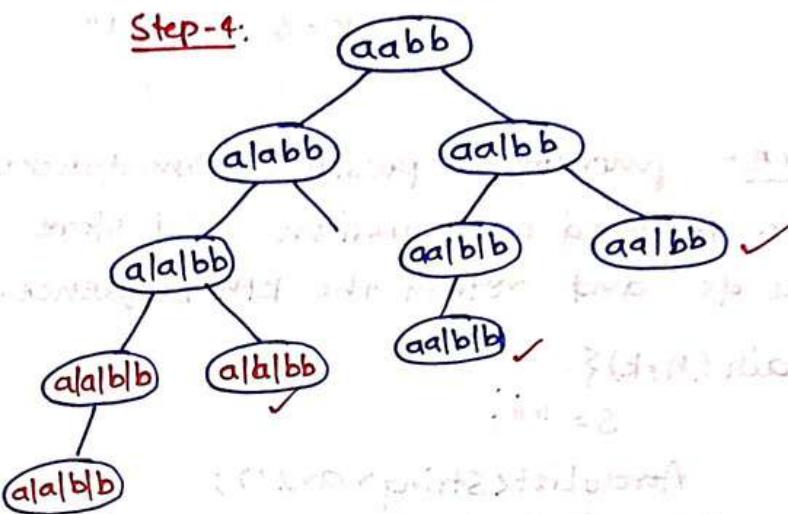
Step-2



Step-3



Step-4.



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

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

2^n to generate every substring
 $O(n/2) \rightarrow$ 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, idx, i)) {

 path.add(s.substring(idx, i+1));
 help(i+1, s, path, ans);
 path.remove(path.size() - 1);

}

}

we have to write
a function for checking
the palindrome.

find kth permutation - Problem - S4

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++) {
        st = i;
    }
    help(s.toCharArray(), 0, res);
    return res.get(k);
    ans();
}
```

```
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);
    }
}
```

Time complexity. $n! * n$

$$N = 4, K = 17 = 16^{\text{th}}$$

↓

(1, 2, 3, 4)

0 1 2 3

$$16/6 = 2$$

$$16 \% 6 = 4^{\text{th}}$$

permutation

$$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$$

$$\underline{1, 2, 4, k=4}$$

3 4 1 2

1 {2, 4} 2 (0-1)

x ② {1, 4} 2 (2-3)

$$\cancel{④} \{1, 2\} \frac{2}{6} (4-5)$$

$$4 \% 2 = 0$$

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

{1, 2}

1 {2} → 1 (0-1)

$$2 \{1\} \frac{1}{2} (1-1)$$

$$k = \frac{0}{1} \quad k = 0 \therefore 0$$

follow same process for 2

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 - $O(N)$

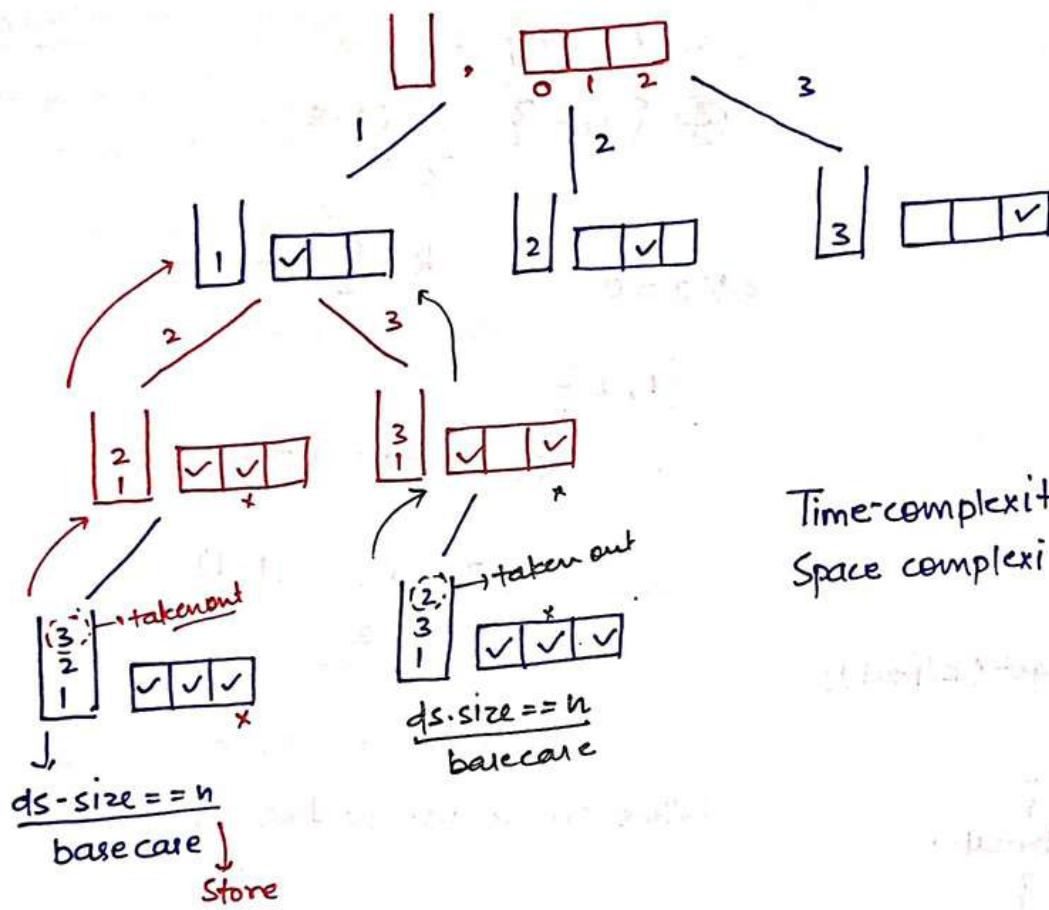
Print All Permutations of a String / Array Problem - 55

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 Extra Space Generate all the permutations.

$3! = 6$ total permutation.



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

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

main permute (nums) {

 List<List<int>> ans;

 List<int> ds;

 boolean freq[3];

 recursive (nums, ans, ds, freq);

 return ans;

}

recursive (nums, ans, ds, freq) {

 if (ds.size == nums.length) ans.add (new ArrayList<Integer>(ds));
 return;

 for (i=0; i<nums.length; i++) {

 if (!freq[i]) { freq[i] = true; ds.add(nums[i]); recursive (nums, ans, ds, freq); ds.remove(ds.size() - 1); freq[i] = false; }

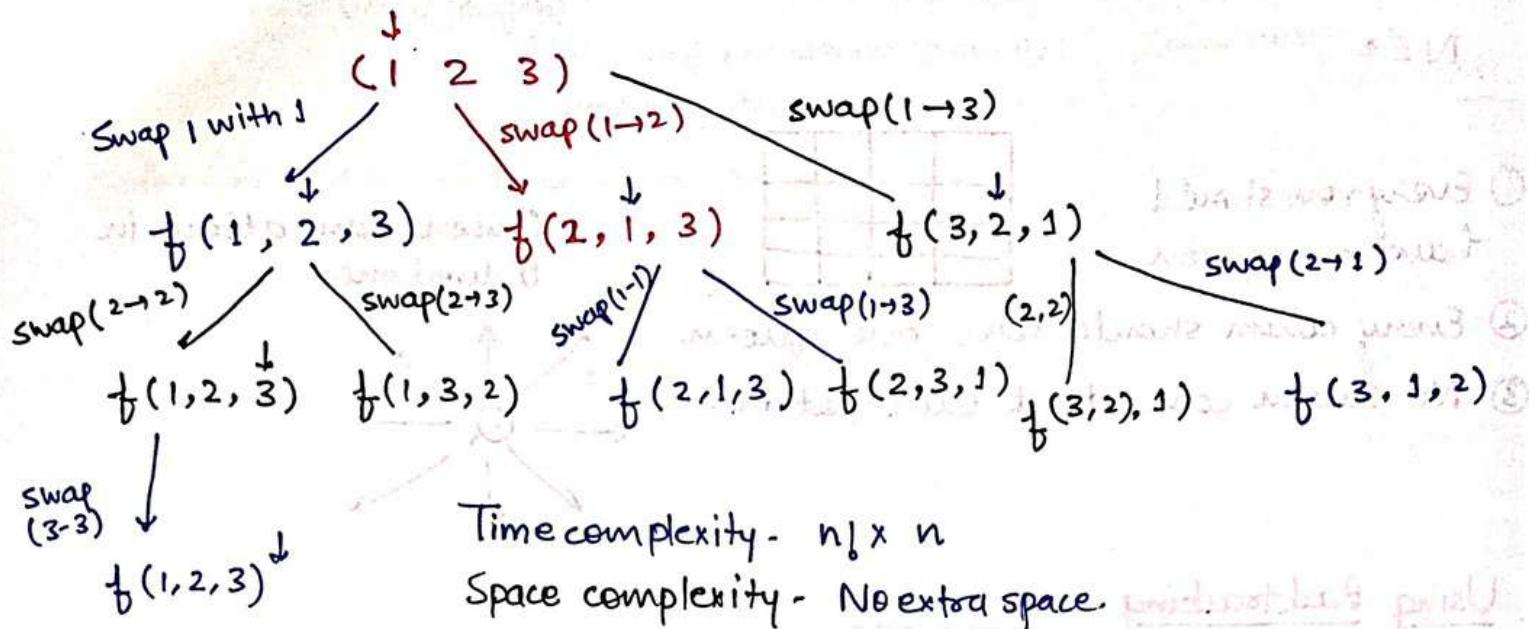
 }

 }

 }

 }

Without using Extra space - Swapping technique.



```
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);
```

```
}
```

```
void swap(int i, int j, nums) {
```

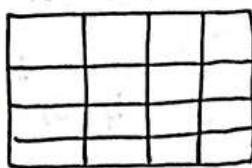
```
    int t = nums[i];
    nums[i] = nums[j];
    nums[j] = t;
```

```
}
```

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

N = 4

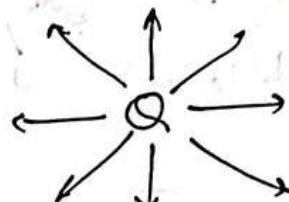
① Every row should have one queen



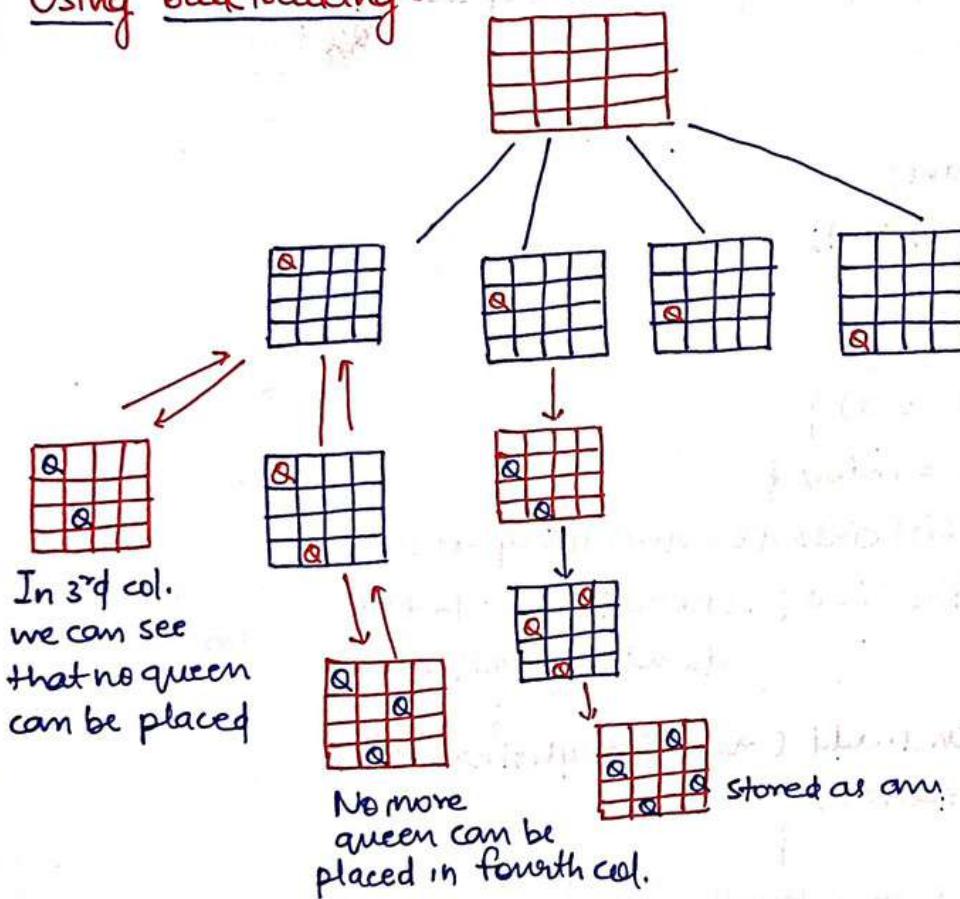
Queen can attack in 8 directions.

② Every column should have one queen

③ No Queen can attack each other.



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
        return; // function
    }

    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(constraint(board));
```

```
    return;
```

```
    for(row = 0; row < board.length; row++) {
```

```
        if(leftRow[row] == 0 && lowerDiagonal[row+col] == 0 && upper[board.length - 1 + col - row] == 0)
```

```
{board[row][col] = 'Q';
```

```
left[row] = 1;
```

```
lowerDiagonal[row+col] = 1;
```

```
upper[board.length - 1 + col - row] = 1;
```

```
solve();
```

```
backtrack
```

```
→ putting the reverse value over here.
```

```
}
```

```
}
```

```
}
```

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

Sudoku Solver - Problem no 57

Given a 9×9 incomplete sudoku, solve it such that it becomes valid sudoku.

① All rows filled with 1-9 exactly once

② All columns " " 1-9 " "

③ Each 3×3 submatrix filled with 1-9 exactly once

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

④ because it has to satisfy all the conditions so call another function isValid(). → checks that the putted no is violating the rules or not.

⑤ If it is violating, 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.

⑥ 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 parent function it tells we can not go this way.

```
bool main(char[9][9] board){
```

```
    for(i=0; i<9; i++) {
```

```
        for(j=0; j<9; j++) {
```

```
            if(board[i][j] == '.') {
```

```
                for(char c='1'; c<='9'; c++) {
```

```
                    if(isValid(board, i, j, c)) {
```

```
                        board[i][j] = c;
```

```
                        if(solveSudoku(board))
```

```
                            return true;
```

```
                        else
```

```
                            board[i][j] = '.';
```

```
                }
```

```
                return false;
```

```
}
```

```
return true;
```

```
}
```

Time complexity - $O(9^{2n})$

SC - $O(1)$

boolean isValid() {

```
    for(i=0; i<9; i++) {
```

```
        if(board[i][col] == c) return false;
```

```
        if(board[row][i] == c) return false;
```

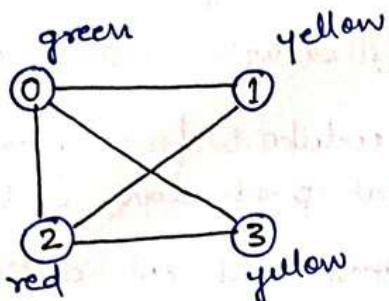
```
        if(board[3 * (row/3) + i/3][3 * (col/3) + i % 3] == c)
```

```
            return false;
```

```
    }
```

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] == c) 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 $n \times m$ ($n=m$). 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 \downarrow only

1	0	0	0
1	1	0	1
1	1	0	0
0	1	1	1

2 ways - DDRDRRR, DRDDDRR

```
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 - $O(4^{m \times n})$

Sc - $O(m \times n)$

```
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; // backtrack.

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 guarantee 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 $\text{fun}(x, n) = m$. If no such no exist, we will return -1.

$\text{fun}(x, n)$ returns the value of x raised to the pow n.

$$\downarrow \\ x^n$$

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(Nth Root (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;
```

```
}
```

① $\text{fun}(\text{mid}, \text{n}, \text{m}) == 1$ mean
it is our answer

② $\text{fun}(\text{mid}, \text{n}, \text{m}) == 0$ get no
is smaller than our answer

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

```
    if (ans == m) return 1;
```

```
    return 0;
```

```
}
```

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

$r = 3, c = 3$

1 3 8

2 3 4

1 2 5

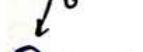
Median = 3

Brute force - Sort the array 2D \rightarrow using a array and
simply find the median.

Optimal Approach.

just look in the matrix how many elements are there less than or equal to 0. search in particular row. In the first row there are 3 element ≤ 0

$$\begin{array}{cccccc}
 \text{first row} & & & & & \\
 \text{Second row} & " & " & 2 & " & " \\
 \text{third} & " & " & " & 2 & " \\
 & & & & \hline
 & & & & \underline{08 - 1} = 7
 \end{array}$$

 including 8 there are 7 no which is less than 8.

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

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

14 04


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

$$[5 \dots \dots 7] = \text{mid} = 6$$

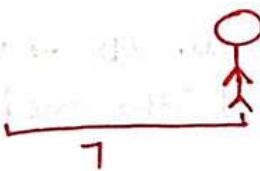
first row = 3

Second row = 2

Third = 2

07

go to left mid - 1
high



Optimal Approach.

$$\text{low} \quad \text{high}$$

$[1 \dots \dots \dots 10^9] \rightarrow \text{starting search space}$

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

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

$$= 8$$

\downarrow

$\text{low} \quad \text{high}$

$[1 \dots \dots \dots 15]$

\uparrow

like for easyness implement
let high is 15.

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

	Second row	" " 2 "	"
	third	" " 2 "	"

$$\frac{08 - 1}{2} = 7$$

$\overbrace{7 \leftarrow \text{move}}$ including 8 there are 7 no which is less than 8.

so there are 7 no.

$$[1 \dots \dots \dots 7] \Rightarrow \frac{1+7}{2} = \frac{8}{2} = 4$$

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

$$\text{first row} = 2$$

$$\text{Second row} = 1$$

$$\text{third row} = 1$$

$$\frac{04}{04}$$

$\overbrace{4 \leftarrow \text{move}}$ including me there are only 4 member small or equal on left. because there are 4 me including 4. so low+1, mid+1

$\downarrow \text{low} \quad \downarrow \text{high}$

$$[5 \dots \dots \dots 7] = \text{mid} = 6$$

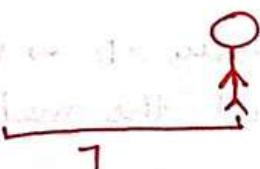
$$\text{first row} = 3$$

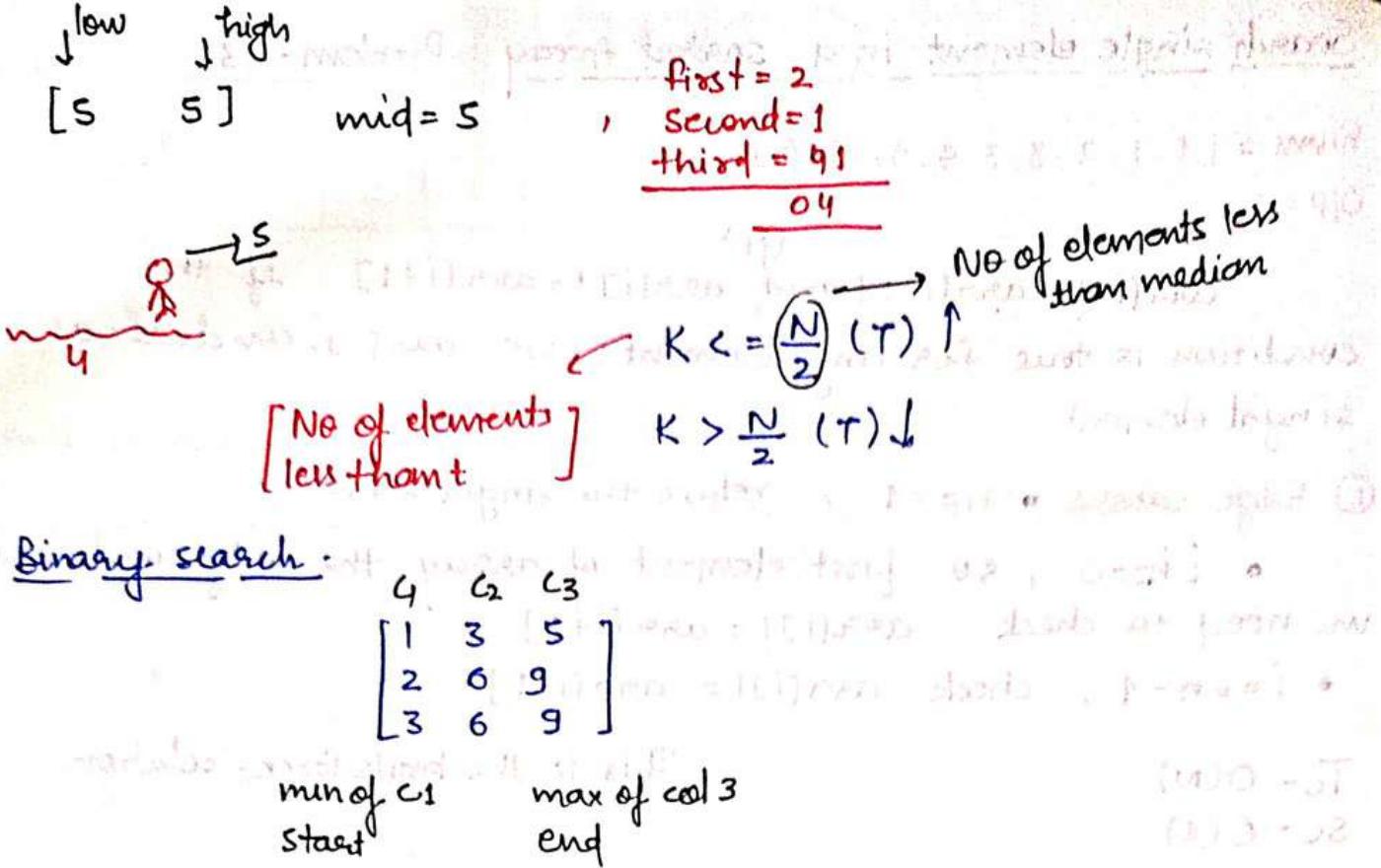
$$\text{Second row} = 2$$

$$\text{Third} = 2$$

$$\frac{07}{07}$$

go to left $\frac{\text{mid}-1}{\text{high}}$





Binary search

C ₁	C ₂	C ₃
1	3	5
2	6	9
3	6	9

min of C₁
 start

max of C₃
 end

```
main findMedian(int[][] A, int row, int col) {
```

```

int low = 1;
int high = 108;
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 - $\log_2(2^{3^2}) \times N \times \log_2 M$

Search single element in a sorted array Problem - 63.

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

O/P = 2

(ff) $\text{arr}[i] \neq \text{arr}[i-1]$ and $\text{arr}[i] \neq \text{arr}[i+1]$: If this condition is true for any element: we arr[i], conclude as singal element.

① Edge cases:

- $i=0$, so first element of array the only condition we need to check $\text{arr}[i] \neq \text{arr}[i+1]$
- $i=n-1$, check $\text{arr}[i] \neq \text{arr}[i-1]$

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.

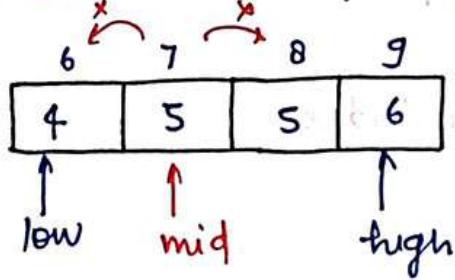
low	mid	high
1	3	6
0	5	10

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

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

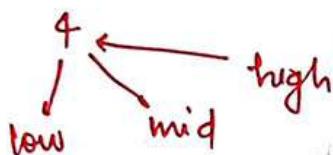
because mid is on odd the it mean previous even (even, odd) mean

so our element is on the right half.



mid is trying to find where is the pos.

so because $4 \neq 5$ but $5 == 5$ so we 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 &&
```

```
else
```

```
    high = mid - 1;
```

Tc - $\log_2(N)$

Sc - O(1)

```
return -1;
```

Search Element in sorted rotated Array . Problem - 64

Bruteforce -

linear traverse and get the index.

Time complexity - $O(N)$

Space complexity - $O(1)$

0	1	2	3	4
4	5	1	2	3

target element = 2
cmll = 3

Optimize Approach -

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

↑
low

↑
mid

↑
high

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

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) if 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.

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

↑ ↑ ↑
low mid high

$$\begin{aligned} \text{mid} &= \frac{\text{low} + \text{high}}{2} \\ &= \frac{0+8}{2} = 4 \end{aligned}$$

checked the sorted half.

arr[low] <= arr[mid] X

got right half.

check. arr[mid] <= target if target <= arr[high]
(low = mid + 1) X

else mid - 1 = high ✓

0	1	2	3
7	8	9	1

↑ ↑ ↑
low mid high

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

arr[low] <= target if 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.

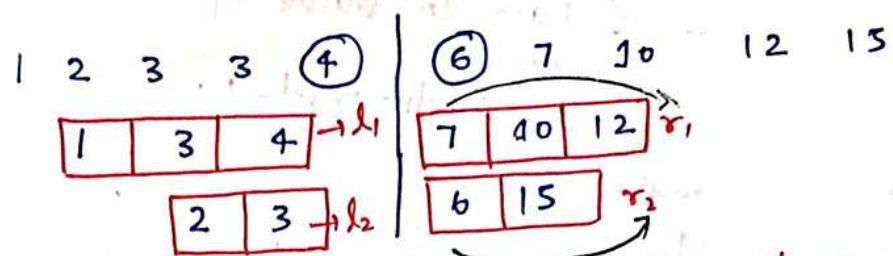
② nums = [1, 2], nums2 = [3, 4]

merged array = [1, 2, 3, 4] \Rightarrow 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 merged 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

$$\begin{aligned}l_1 &\leq r_2 \\l_2 &\leq r_1\end{aligned}$$

if the array is even length then if we find,

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

$$\text{arr1} \rightarrow \begin{array}{|c|c|c|c|} \hline 0 & 1 & 2 & 3 \\ \hline 7 & 12 & 14 & 15 \\ \hline \end{array}$$

$$\text{arr2} \rightarrow \begin{array}{|c|c|c|c|c|} \hline 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 2 & 3 & 4 & 9 & 11 \\ \hline \end{array}$$

Total 10 elements 5 will be on left and 5 on the right.
so I can take min 0 elements from arr1 and max 4 elements from array 2.

low

0

high

4

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

(so pick 2 elements from arr1 initially.)

cut 2 = $\frac{5-2}{2} = 3$ (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. $l_1 = 12, r_1 = 14 \quad [12 <= 4 \text{ (No)}$

$$l_2 = 3 \quad r_2 = 4$$

Not a valid partition

So we have to decide whether to

move. we can see that: 12 has to be smaller than 4. for the

valid partition.

So f have to move left

$$\text{so } \text{high} = \text{mid} - 1.$$

low high
0 1

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

arr 1 =	0	1	2	3
	7	12	14	15

(No element is taking from arr 1).

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

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

If you don't have nothing assign minimal.

$x^{l_1} \quad | \quad r_1 \quad 12 \quad 14 \quad 15$

$$l_1 = \text{INT-MIN}; \quad 1 \quad 2 \quad 3 \quad 4 \quad 9 \quad l_2 \quad | \quad r_2$$

$$l_1 <= r_1 \text{ (Yes)}$$

$$l_2 <= r_1 \text{ (No)}$$

so we have to decrease 9 and

increase 7

$$\text{low} = \text{mid} + 1 = 0 + 1 = 1$$

low -
 $0+1$

high
 1

$$\text{mid} = \frac{1+1}{2} = 0$$

$$\text{cut } 2 = 5 - 1 = 4$$

$$\text{arr } 1 = \boxed{7 \quad | \quad 12 \quad 14 \quad 15}$$

$$\text{arr } 2 = \boxed{1 \quad 2 \quad 3 \quad 4 \quad | \quad 9 \quad 11}$$

$$\begin{array}{c} l_1 \quad r_1 \\ 7 \quad | \quad 12 \quad 14 \quad 15 \\ 1 \quad 2 \quad 3 \quad 4 \quad | \quad 9 \quad 11 \\ l_2 \quad r_2 \end{array}$$

$$l_1 \leq r_2 \text{ (yes)}$$

$$l_2 \leq r_1 \text{ (yes)}$$

so take the min from right which is 9
and max from arr part 1, which is 7.

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

$$l_1 = \text{arr } 1[\text{cut } 1 - 1]$$

$$r_1 = \text{arr } 1[\text{cut } 1]$$

$$l_2 = \text{arr } 2[\text{cut } 2 - 1]$$

$$r_2 = \text{arr } 2[\text{cut } 2]$$

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)

$$\begin{array}{c} ⑥ \\ 1 \quad 2 \quad 3 \quad 4 \quad 7 \quad \boxed{9} \quad | \quad 11 \quad 12 \quad 14 \quad 15 \quad 16 \end{array}$$

Super simple for odd len.
median will lie on the left half

$$\text{median} = \max(l_1, l_2)$$

$$\frac{n_1+n_2+1}{2} \Rightarrow \frac{12}{2} = 6$$

```

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

Kth element of Two Sorted Arrays. Problem - 66

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

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

k = 5

output = 6.

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

Time complexity - 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

2 3 6 15	$\xrightarrow{l_1}$	x_1	3 4 7 10 12	
	$\xleftarrow{l_2}$	x_2		

this can't be possible.

because.

$$15 <= 3 \text{ (No).}$$

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

2 3 6 15
0 1 2 3

low high
0 3

$\xleftarrow{K=03}$

(change 0 to 1).

edge case.
there are one or elements
on the arr1 of and if
 $K < arr1$ then this condit
tion arises

$\xrightarrow{K=7}$ low 0 1 high 4

if $K > arr1$ then we have pick
element from arr2

$l_1 = \text{arr1}[\text{cut}-1]$, $r_1 = \text{arr1}[\text{cut}-1]$
 $l_2 = \text{arr2}[\text{cut2}-1]$, $r_2 = \text{arr2}[\text{cut2}]$
 " "
 if 0 then put
 INT-MIN

80,
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 A_i pages. m students need to contiguous books. Out of all the permutations, the goal is to find the permutation where the student with the most books allocated him gets the min. no of pages.

$$N = 4$$

$$A[] = [12 \ 34 \ 67 \ 90]$$

$$M = 2$$

$$\text{Output} = 113$$

there are 2 guys, M_1 , and M_2 .

$$M_1 \quad M_2$$

$$[12] \quad [34, 67, 90] \rightarrow \text{maxpage} = 191$$

because each gets
min. 1 book

$$[12, 34] \quad [67, 90] \rightarrow \text{maxpage} = 157$$

$$[12, 34, 67] \quad [90] \rightarrow \text{maxpage} = 113$$

minimum of all 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 contiguous order

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

$$12 \\ \text{low}$$

$$363 \\ \text{high}$$

$$\text{arr}[] = [12 \ 34 \ 67 \ 90]$$

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

what will be the low so

because lets assume

$$m = 3, \text{ arr}[] = [12, 12, 12]$$

$$m = 1, \text{ arr} = [12, 12, 12]$$

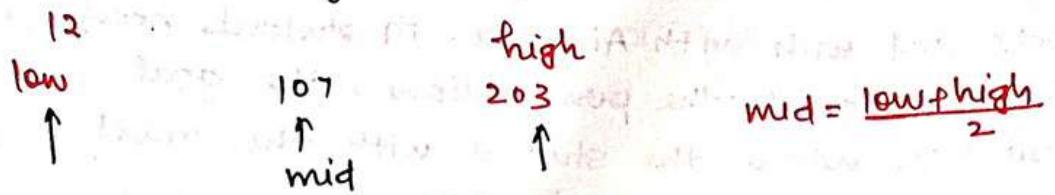
min = 36 because

we give all the pages to

1 student, high = 36

so 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 -



Student 1 → 12 (because its not exceed 107)

we can take → 34, Student 1 val = $12 + 34 < 107$

again when we try to give 67 to student 1 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

so, 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).



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 $\text{high} = \text{mid} - 1$; (Right remove portion).



$$1 \rightarrow 12 + 34 + 67 < 131$$

2 $\rightarrow 90$ can be a valid answer

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

so $\text{high} = \text{mid} - 1$



ans, $131 < 155$ (our prev ans)

so ans will be 131

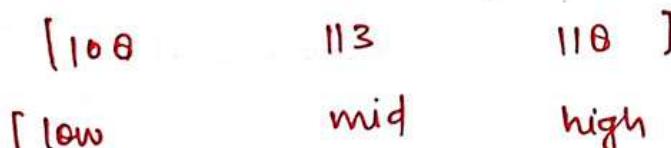
$$1 \rightarrow 12 + 34 + 67$$

$$2 \rightarrow 90$$

again $119 < 131$ so

and 119 is a valid barrier so we reduce the answer

$\text{high} = \text{mid} - 1$



$$1 \rightarrow 12 + 34 + 67 < 113$$

$$2 \rightarrow 90$$

ans = 113 (because mid < 119)



1 → 12 + 34 we can't add 67 because 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

$$\underline{\text{low} = \text{mid} + 1}$$



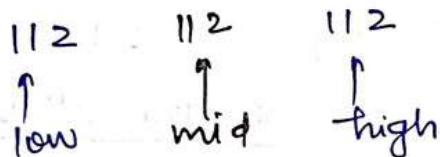
by keeping a barrier 111

Student 1 → 12 + 34

Student 2 → 67

Student 3 → 90 (X) → only 2 students.

hence low = mid + 1.

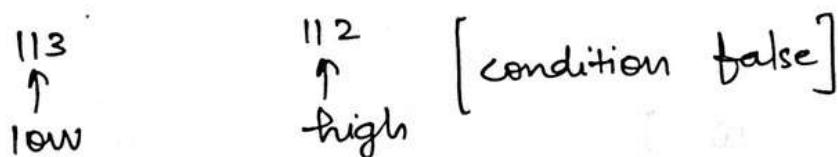


Student 1 → 12 + 34 not able to take 67. exceed 112.

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 (students > m)
```

```
            low = mid + 1;
```

```
        else
```

```
            high = mid - 1;
```

```
}
```

```
    return low;
```

```
}
```

```
countstudent (arr, mid) {
```

```
    n = arr.length;
```

```
    int student = 1;
```

```
    long pages = 0;
```

```
    for (i = 0; i < n; i++) {
```

```
        if (pages + arr[i] <= pmid)
```

```
            pages += arr[i];
```

```
        else {
```

```
            student += 1;
```

```
            pages = arr[i];
```

```
}
```

```
}
```

```
    return student;
```

```
}
```

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

Aggressive Cows - Problem-68

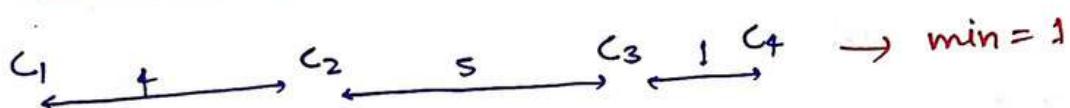
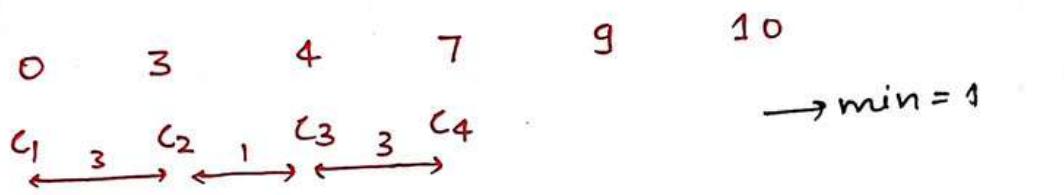
Aggressive cow \rightarrow (min dist between cows) is max

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

N stalls.

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

So if the array so the some stalls will be in contiguous order and the distance will be min between two

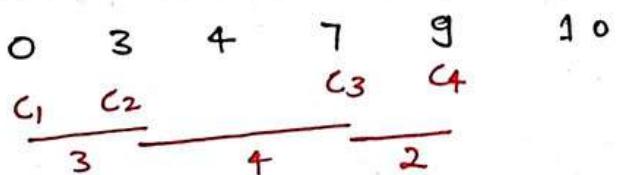


max between these $\min = 3$

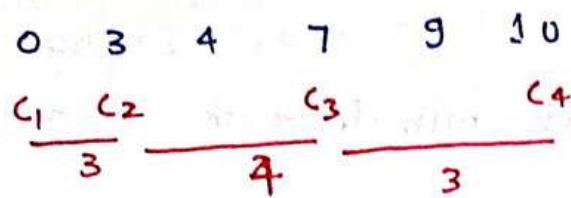
at start we need to try the cows will be put at min distance of 1. initially put cow 1 at 0 initially at every step.

arr[] = [0 3 4 7 9 10]
 $c_1 \xleftarrow{3} c_2 \xleftarrow{1} c_3 \xleftarrow{3} c_4$ cows = 4

- ① ✓ Now we try to put the cow in the min distance of two.
- ② ✓ Now we try to put the cow in the min distance of two.

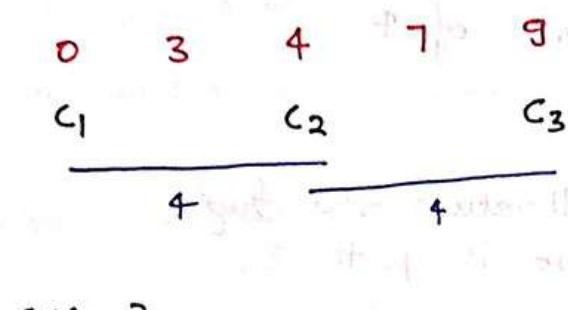


③ try to put at min distance of 3.



Yes 3 valid.

④ try to put at min distance of 4



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

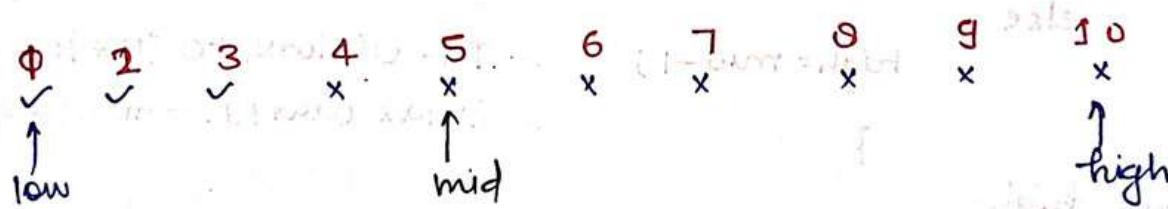
ans = 3

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

Space complexity - $O(1)$.

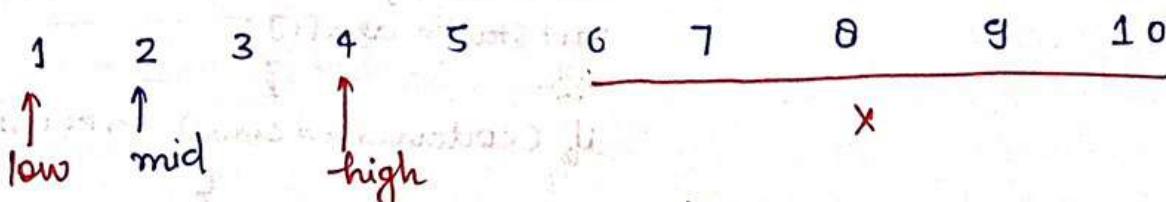
Optimised solution -

create a range , Φ



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

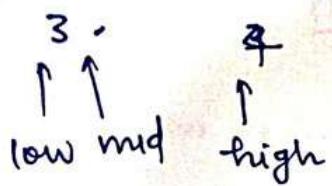
remove the right half; , $high = mid - 1$;



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

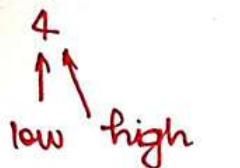
so go to $low = mid + 1$.

Now dist = 2 valid



also valid to put the cows at min distance

$$\text{low} = \text{mid} + 1$$



$$\text{mid} = (\text{low} + \text{high})/2$$

$$\text{mid} = 4$$

we can't put the cows at distance of 4

so

$$[\text{high} = \text{low} \text{ mid} - 1]$$

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

main (arr, k) {

 sort (arr);

 n = arr.size();

 low = 1; high = ~~start~~ 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;

}

$Tc = O(N \log N) + O(N * \log(\max(arr) - \min(arr)))$

canPut (arr, mid, k) {

 cntcows = 1, laststall = arr[0];

 for (i = 1; i < arr.length; i++) {

 cntcows++;

 laststall = arr[i];

}

 if (cntcows >= cows) return true;

}

 return false;

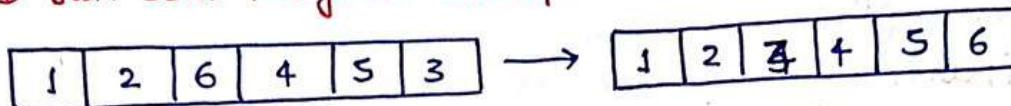
Kth largest / Smallest element in an Array Problem - 69

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

largest = 6, smallest = 3

Bruteforce

- ① Just sort the given array.



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

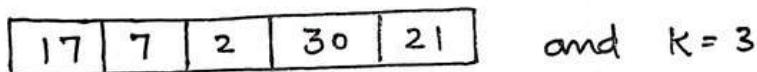
Time complexity - $O(n \log n)$.

Solution: Using Heap

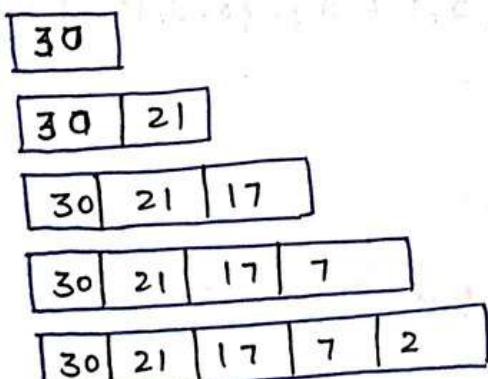
- The idea is to construct a max-heap of elements. Since the top of the max is the largest elements of the heap. we will remove top $(K-1)$ elements from the heap.

The top element will be Kth largest element.

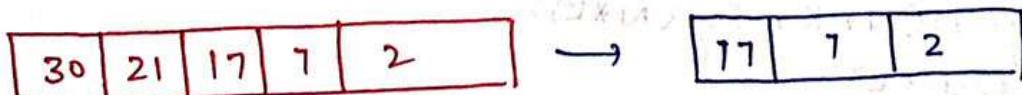
Let the array be.



Gen. the max heap.



Now extract K-1 values.



mean 2 value.

extract

- To get k^{th} 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
    Priority Queue<Integer> pq = new Priority Queue<((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();
}
    
```

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

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

Space - $O(K)$

```

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();
}
    
```

Merge K Sorted Arrays - Problem 70

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

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

Bauteforce : 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;
```

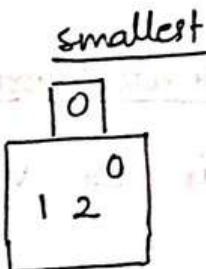
public int compareTo (Pair o)

return this.val - this.o.val;

↓ ↓ ?
 | 3 5 7
 ↓
 2 4 6 8
 ↓ ↓
 0 9 10 11

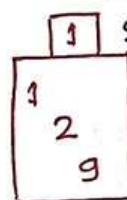
pqr →

initially put
the idx=0 element
in pqr.

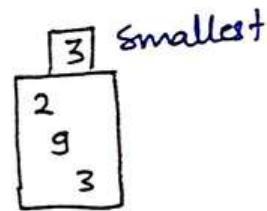


ans = 0 1 3

Now, pqr will



Now put 3



main (ArrayList<ArrayList<Ints>>)

{
 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();

list.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

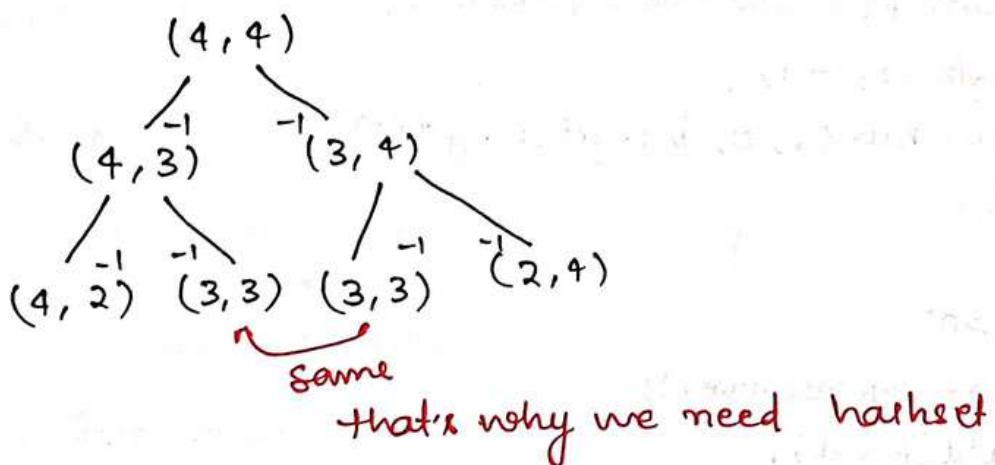
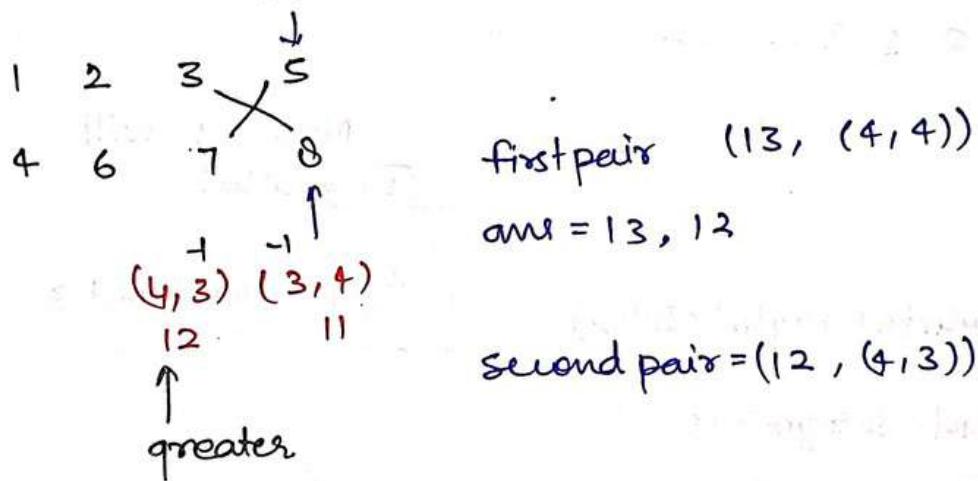
Output : [10, 9, 9, 8]

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



Using 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 indices so we use the pairs
    // to make sure no indices 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.hash (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;
    }
}

main (arr, K) {
    Map<int, int> mp();
    for (int i : arr) {
        mp.put (i, mp.getOrDefault (i, 0) + 1);
    }
    Priority Queue <Pair> p = new Priority Queue <> (new cpair ());
    for (Map.Entry<int, int> e: mp.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 - $O(N \log N)$
Space complexity - $O(N)$.