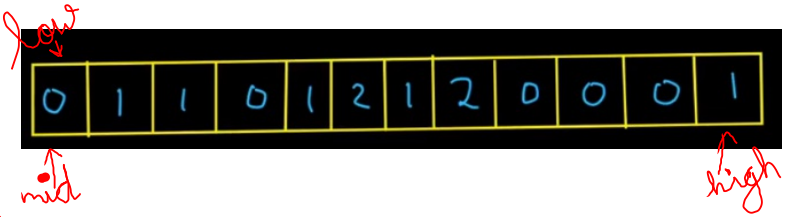
1. **Sort an array of 0's 1's 2's without using extra space or sorting algo**

T → O(n)
S → O(1)



0 to low-1 ⇒ 0
high+1 to n ⇒ 2
low to mid-1 ⇒ 1

value at mid

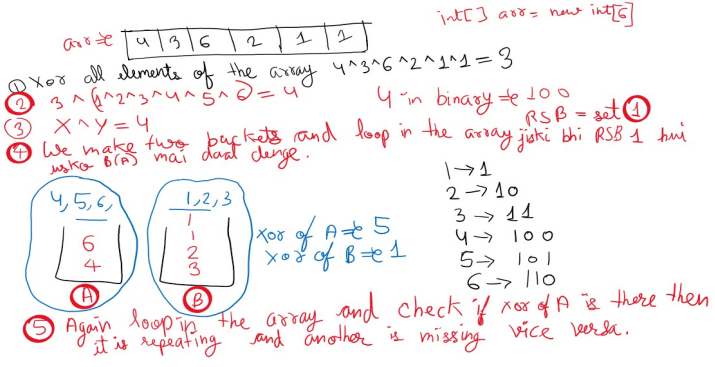
0 → swap low to mid
low++, mid++

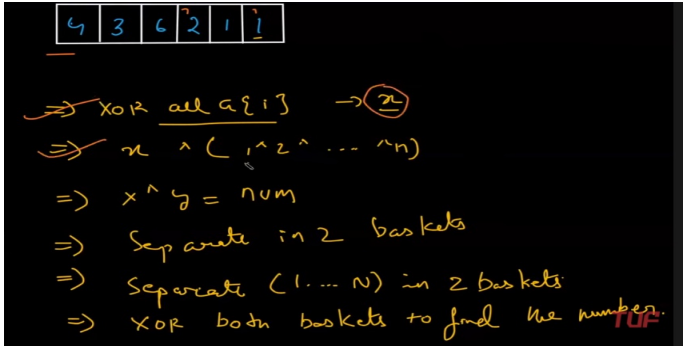
1 → mid++;

2 → swap mid to high
high--

```
class Solution {
    public void sortColors(int[] nums) {
        int lo = 0;
        int hi = nums.length - 1;
        int mid = 0;
        int temp;
        while (mid <= hi) {
            switch (nums[mid]) {
                case 0: {
                    temp = nums[lo];
                    nums[lo] = nums[mid];
                    nums[mid] = temp;
                    lo++;
                    mid++;
                    break;
                }
                case 1:
                    mid++;
                    break;
                case 2: {
                    temp = nums[mid];
                    nums[mid] = nums[hi];
                    nums[hi] = temp;
                    hi--;
                    break;
```

2. Repeat and Missing Number






```java
import java.io.*;
class GFG {
  static int x, y;
  static void getTwoElements(int arr[], int n) {
    /* Will hold xor of all elements
    and numbers from 1 to n */
    int xor1;
    /* Will have only single set bit of xor1 */
    int set_bit_no;
    int i;
    x = 0;
    y = 0;
    xor1 = arr[0];
    /* Get the xor of all array elements */
    for (i = 1; i < n; i++)
      xor1 = xor1 ^ arr[i];
    /* XOR the previous result with numbers from 1 to n*/
    for (i = 1; i <= n; i++)
      xor1 = xor1 ^ i;

    /* Get the rightmost set bit in set_bit_no */
    set_bit_no = xor1 & ~(xor1 - 1);

    /* Now divide elements into two sets by comparing rightmost set bit of xor1 with the bit at the same position in each element. Also, get XORs of two sets. The two XORs are the
output elements. The following two for loops serve the purpose */
    for (i = 0; i < n; i++) {
      if ((arr[i] & set_bit_no) != 0)
        /* arr[i] belongs to first set */
        x = x ^ arr[i];

      else
        /* arr[i] belongs to second set*/
        y = y ^ arr[i];
    }
    for (i = 1; i <= n; i++) {
      if ((i & set_bit_no) != 0)
        /* i belongs to first set */
        x = x ^ i;

      else
        /* i belongs to second set*/
        y = y ^ i;
    }

    /* *x and *y hold the desired output elements */
  }
  /* Driver program to test above function */
  public static void main(String[] args){
    int arr[] = { 1, 3, 4, 5, 1, 6, 2 };
    int n = arr.length;
    getTwoElements(arr, n);
    System.out.println(" The missing element is "+ x + "and the "+ "repeating number is "+ y);
  }
```

Press Esc to exit full screen

3. Merge two sorted Arrays without extra space

1st Method :- (gap method)

```
public class MergeTwoSortedArrays {
  // Function to find next gap.
  private static int nextGap(int gap) {
    if (gap <= 1) {
      return 0;
    }
    return (gap / 2) + (gap % 2);
  }

  private static void merge(int[] arr1, int[] arr2, int n, int m) {
    int i, j, gap = n + m;
    for (gap = nextGap(gap); gap > 0; gap = nextGap(gap)) {
      // comparing elements in the first array.
      for (i = 0; i + gap < n; i++)
        if (arr1[i] > arr1[i + gap]) {
          int temp = arr1[i];
          arr1[i] = arr1[i + gap];
          arr1[i + gap] = temp;
        }
      // comparing elements in both arrays.
      for (j = gap > n ? gap - n : 0; i < n && j < m; i++, j++)
        if (arr1[i] > arr2[j]) {
          int temp = arr1[i];
          arr1[i] = arr2[j];
          arr2[j] = temp;
        }

      if (j < m) {
        // comparing elements in the second array.
        for (j = 0; j + gap < m; j++)
          if (arr2[j] > arr2[j + gap]) {
            int temp = arr2[j];
            arr2[j] = arr2[j + gap];
            arr2[j + gap] = temp;
          }
      }
    }
  }

  // Driver Code
  public static void main(String[] args) {
    int[] a1 = { 10, 27, 38, 43, 82 };
    int[] a2 = { 3, 9 };

    // Function Call
    merge(a1, a2, a1.length, a2.length);

    System.out.print("First Array: ");
    for (int i = 0; i < a1.length; i++) {
```

gap = ceiling of $\frac{n}{2}$

a = | 1 | 4 | 7 | 8 | 10 | $n_1 = 5$

b = | 2 | 3 | 9 | $n_2 = 3$

gap = $\frac{5+3}{2} = \frac{8}{2} = 4$ ①

$\frac{4}{2} = 2$ ②

$\frac{2}{2} = 1$ ③

Here, we will use the gap method

1 4 7 8 10 2 3 9

if i is small then swap

then i++ and j++

T => O(n log n)

S => O(1)

1,4,7,8,10,2,3,9 (g=4)

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

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

g(2)

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

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

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

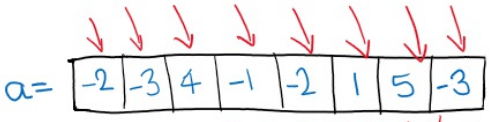
g(1)

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

2nd Method :-

```
7   public static void merge(int[] X, int[] Y)
8   {
9       int m = X.length;
10      int n = Y.length;
11
12      // Consider each element `X[i]` of array `X` and ignore the element if it is
13      // already in the correct order; otherwise, swap it with the next smaller
14      // element, which happens to be the first element of `Y`.
15      for (int i = 0; i < m; i++)
16      {
17          // compare the current element of `X[]` with the first element of `Y[]`
18          if (X[i] > Y[0])
19          {
20              // swap `X[i] with `Y[0]`
21              int temp = X[i];
22              X[i] = Y[0];
23              Y[0] = temp;
24
25              int first = Y[0];
26
27              // move `Y[0]` to its correct position to maintain the sorted
28              // order of `Y[]`. Note: `Y[1…n-1]` is already sorted
29              int k;
30              for (k = 1; k < n && Y[k] < first; k++) {
31                  Y[k - 1] = Y[k];
32              }
33
34              Y[k - 1] = first;
35          }
36      }
37  }
```

4. Kadane's Algorithm → Maximum Subarray Sum



Sum = 0 -2 0 -3 0 4 3 1 7 4

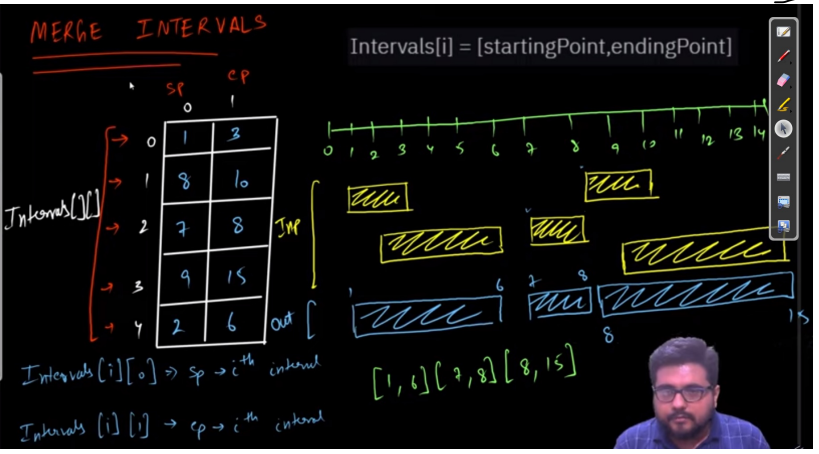
maxi = -2 4 7

max sum = 0;

```
int sum=0
int maxi = a[0]
for(int i=0; i<a.length; i++){
    sum += num[i];
    if( sum < 0){
        sum = 0;
    }if(sum > maxi){
        maxi = sum;
    }
}
```

```java
class Solution {
    public int maxSubArray(int[] nums) {
        int sum = 0;
        int maxi = nums[0];
        for(int i = 0;i<nums.length;i++) {
            sum += nums[i];
            if(sum > maxi) maxi = sum;
            if(sum < 0) sum = 0;
        }
        return maxi;
    }
}
```

5. Merge Overlapping Subintervals (Method 1)




```java
class Solution {
    public int[][] merge(int[][] intervals) {
        List<int[]> res = new ArrayList<>();
        if(intervals.length == 0 || intervals == null){
            return res.toArray(new int[0][]);
        }
        Arrays.sort(intervals, (a,b) -> a[0] - b[0]);

        int start = intervals[0][0];
        int end = intervals[0][1];

        for(int[] i: intervals){
            if(i[0] <= end){
                end = Math.max(end,i[1]);

            }else{
                res.add(new int[]{start,end});
                start = i[0];
                end = i[1];
            }
        }
        res.add(new int[]{start,end});
        return res.toArray(new int[0][]);
    }
}
```

```java
public class Main{
    public static int[][] mergeIntervals(int Intervals[][]){
        Arrays.sort(Intervals,(a,b)->Integer.compare(a[0],b[0])); // sorted on the basis of starting point

        ArrayList<int[]> list = new ArrayList<>();

        for(int[] interval : Intervals){
            if(list.size() == 0){
                list.add(interval);
            }else{
                int prevInterval[] = list.get(list.size()-1);
                if(interval[0] < prevInterval[1]){
                    prevInterval[1] = Math.max(prevInterval[1],interval[1]);
                }else{
                    list.add(interval);
                }
            }
        }

        return list.toArray(new int[list.size()][]);
    }
```

5. Merge Overlapping Subintervals (Method - 2)

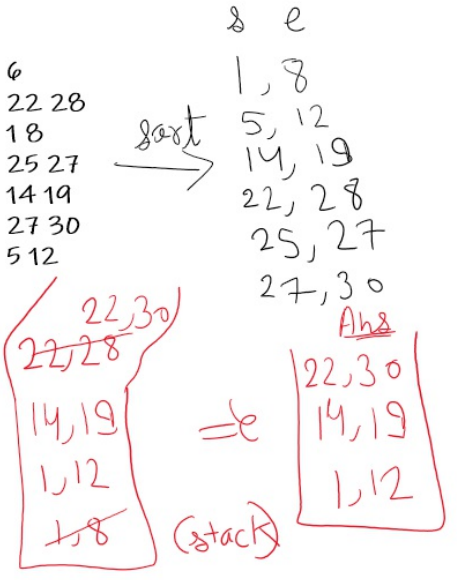
```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int n = Integer.parseInt(br.readLine());
        int[][] arr = new int[n][2];

        for (int j = 0; j < n; j++) {
            String line = br.readLine();
            arr[j][0] = Integer.parseInt(line.split(" ")[0]);
            arr[j][1] = Integer.parseInt(line.split(" ")[1]);
        }

        mergeOverlappingIntervals(arr);
    }

    public static void mergeOverlappingIntervals(int[][] arr) {
        // merge overlapping intervals and print in increasing order of start time
        Pair[] pairs = new Pair[arr.length];
        for(int i=0 ; i<arr.length; i++){
            pairs[i] = new Pair(arr[i][0],arr[i][1]);
        }
        Arrays.sort(pairs);

        Stack<Pair> st = new Stack<>();
        for(int i=0; i<pairs.length; i++){
            if(i==0){
                st.push(pairs[i]);
            }else{
                Pair top = st.peek();
                if(pairs[i].st > top.et){
                    st.push(pairs[i]);
                }else{
                    top.et = Math.max(top.et,pairs[i].et);
                }
            }
        }

        Stack<Pair> rs = new Stack<>();
        while(st.size()>0){
            rs.push(st.pop());
        }
        while(rs.size()>0){
            Pair p = rs.pop();
            System.out.println(p.st+" "+p.et);
        }
    }
    public static class Pair implements Comparable<Pair>{
        int st;
        int et;
        Pair(int st,int et){
            this.st = st;
            this.et = et;
        }

        // this > other return +ve
        // this = other return 0
        // this > other return -ve
        public int compareTo(Pair other){
            if(this.st != other.st){
                return this.st - other.st;
            }else{
                return this.et - other.et;
            }
        }
    }

}
```

Sample Input

```
6
22 28
1 8
25 27
14 19
27 30
5 12
```

Sample Output

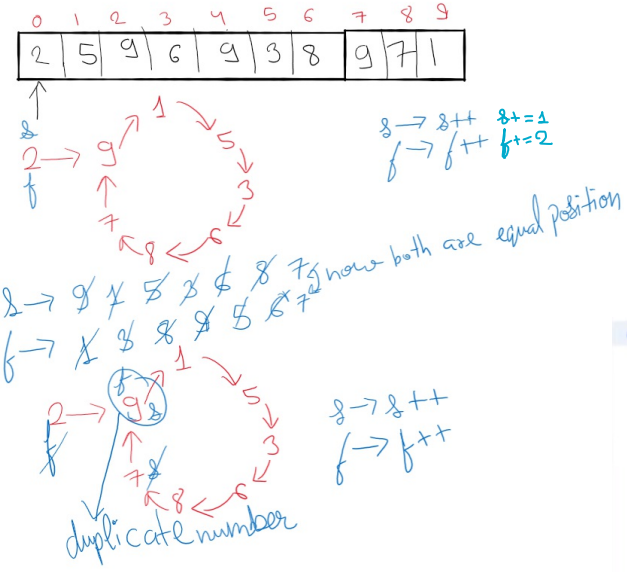
```
1 12
14 19
22 30
```

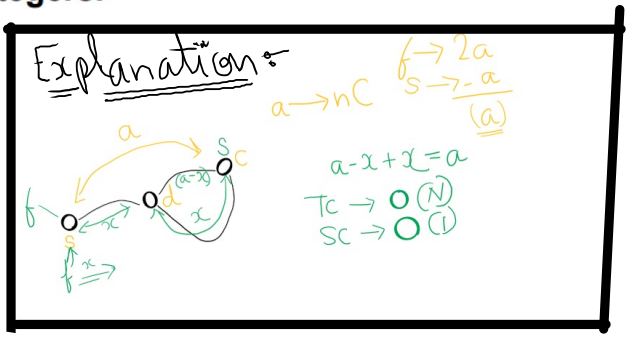


Here, we are going to sort the given inputs on the basis of their starting index then will add the first input to the stack

Now, we will check if first.end > new.start then first.end = max(first.end, new.end)

6. Find the duplicate in an array of N+1 integers.






```java
class Solution {
    public int 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;
    }
}
```

```java
class Solution {
    public int findDuplicate(int[] nums) {
        Set<Integer> seen = new HashSet<Integer>();
        for (int num : nums) {
            if (seen.contains(num))
                return num;
            seen.add(num);
        }
        return -1;
    }
}
```

Tc = O(n)
Sc = O(n)

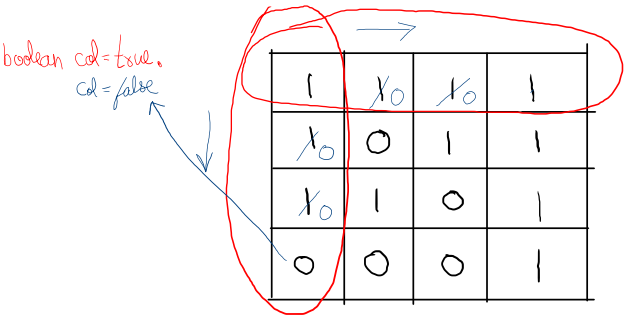
```java
class Solution {
    public int store(int[] nums, int cur) {
        if (cur == nums[cur])
            return cur;
        int nxt = nums[cur];
        nums[cur] = cur;
        return store(nums, nxt);
    }

    public int findDuplicate(int[] nums) {
        return store(nums, 0);
    }
}
```

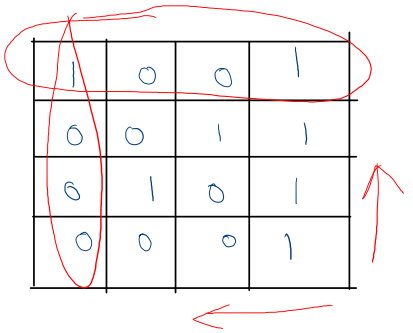
Tc = O(n)
Sc = O(n)

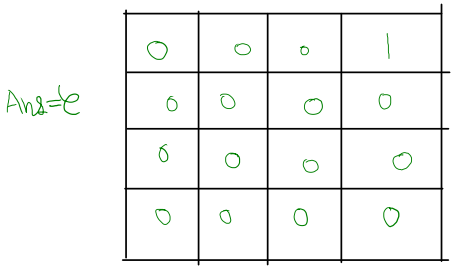
7. Set Matrix Zeros :-

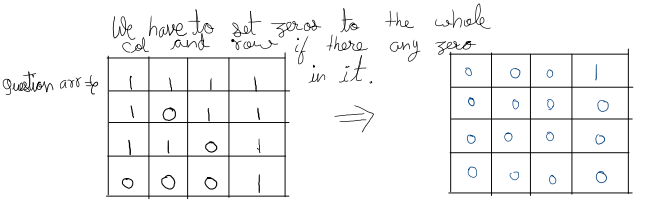
$TC = O(2 \times (N \times M))$

$SC = O(1)$



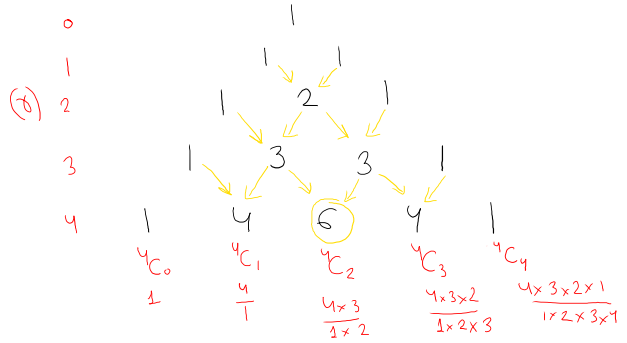







```java
class Solution {
    public void setZeroes(int[][] matrix) {
        int col0 = 1, rows = matrix.length, cols = matrix[0].length;

        for (int i = 0; i < rows; i++) {
            if (matrix[i][0] == 0) col0 = 0;
            for (int j = 1; j < cols; j++)
                if (matrix[i][j] == 0)
                    matrix[i][0] = matrix[0][j] = 0;
        }

        for (int i = rows - 1; i >= 0; i--) {
            for (int j = cols - 1; j >= 1; j--)
                if (matrix[i][0] == 0 || matrix[0][j] == 0)
                    matrix[i][j] = 0;
            if (col0 == 0) matrix[i][0] = 0;
        }
    }
}
```

Pascal's Triangle

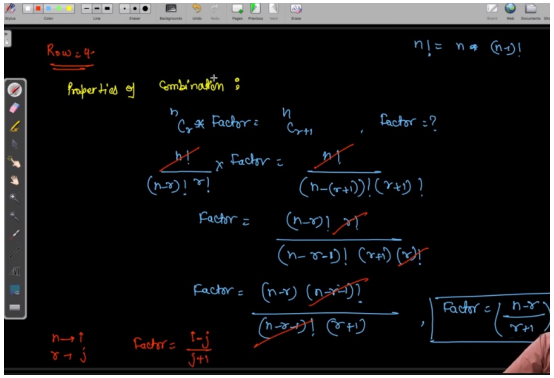


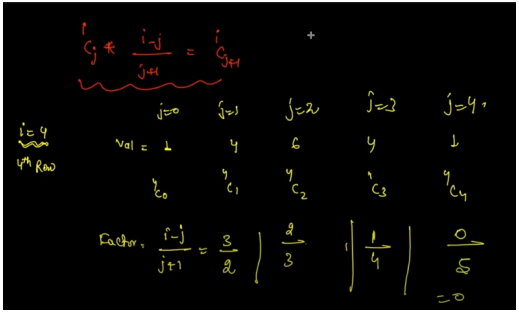

```
// Calculate value of
// [n * (n-1) *---* (n-k+1)] /
for (int i = 0; i < k; ++i) {
    res *= (n - i);
    res /= (i + 1);
}
```

Example 1:

```
Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

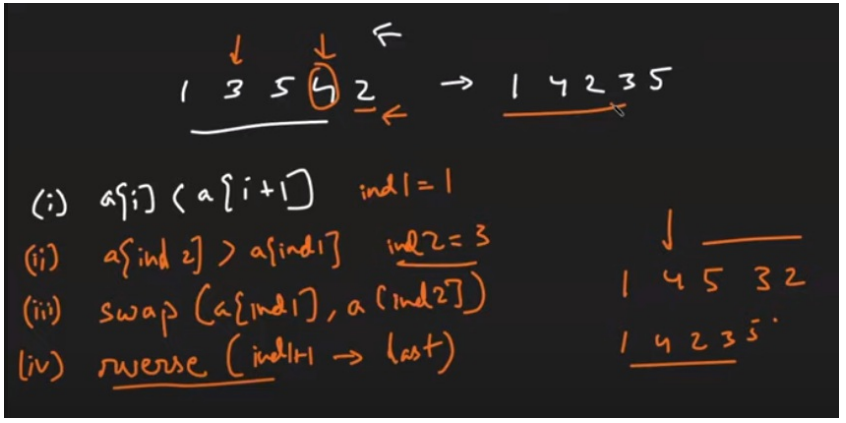
```java
class Solution {
    public List<List<Integer>> generate(int numRows) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        List<Integer> row, pre=null;
        for(int i=0; i<numRows; ++i){
            row = new ArrayList<Integer>();
            for(int j=0; j<=i; j++){
                if(j==0 || j==i){
                    row.add(1);
                }else{
                    row.add(pre.get(j-1) + pre.get(j));
                }
            }
            pre = row;
            res.add(row);

        }
        return res;
    }
}
```

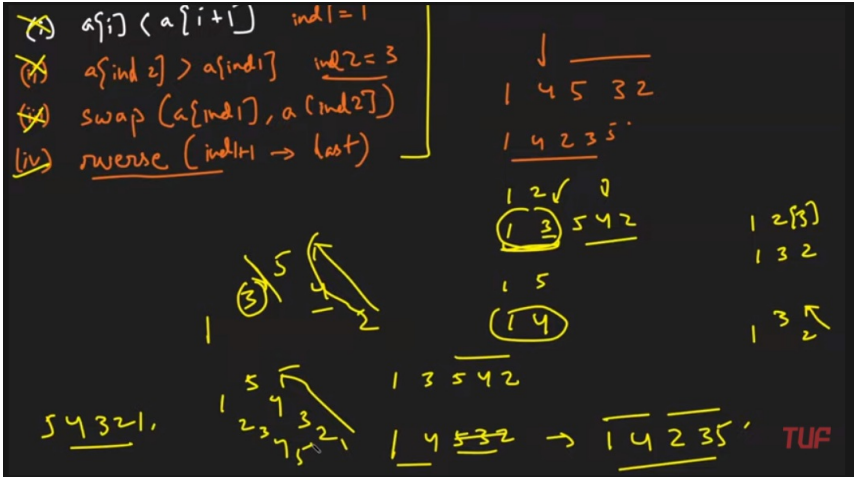





```java
// write your code here
    ArrayList<Integer> res = new ArrayList<>();

    int val = 1;
    for(int j = 0; j <= i; j++) {
        res.add(val);
        val = val * (i - j) / (j + 1);
    }

    return res;
```

Next Permutation

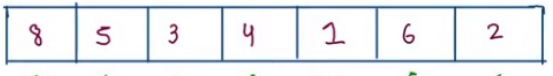




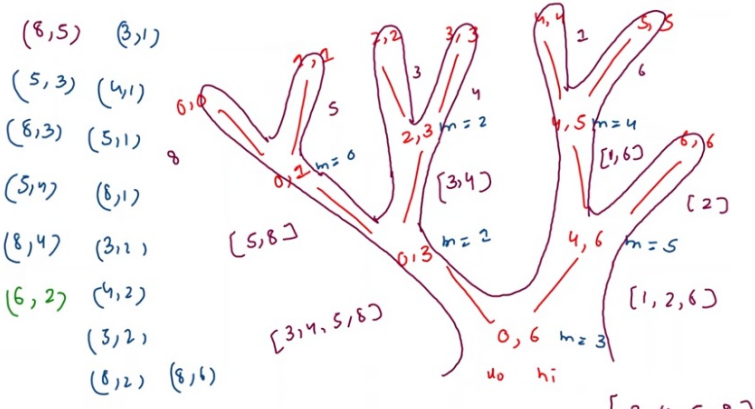

```java
class Solution {
    public void nextPermutation(int[] A) {
        if(A == null || A.length <= 1) return;
        int i = A.length - 2;
        while(i >= 0 && A[i] >= A[i + 1]) i--;
        if(i >= 0) {
            int j = A.length - 1;
            while(A[j] <= A[i]) j--;
            swap(A, i, j);
        }
        reverse(A, i + 1, A.length - 1);
}

public void swap(int[] A, int i, int j) {
    int tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
}

public void reverse(int[] A, int i, int j) {
    while(i < j) swap(A, i++, j--);
}
}
```

Inversion of Array (Using Merge Sort)

1. Given an array of integers. Find the Inversion Count in the array.
2. For an array, inversion count indicates how far (or close) the array is from being sorted. If array is already sorted then the inversion count is 0. If an array is sorted in the reverse order then the inversion count is the maximum.
3. Formally, two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j.



count = 1 + 2 + 2 + 1 + 4 + 4 + 1 = 15



$TC = n \log(n)$

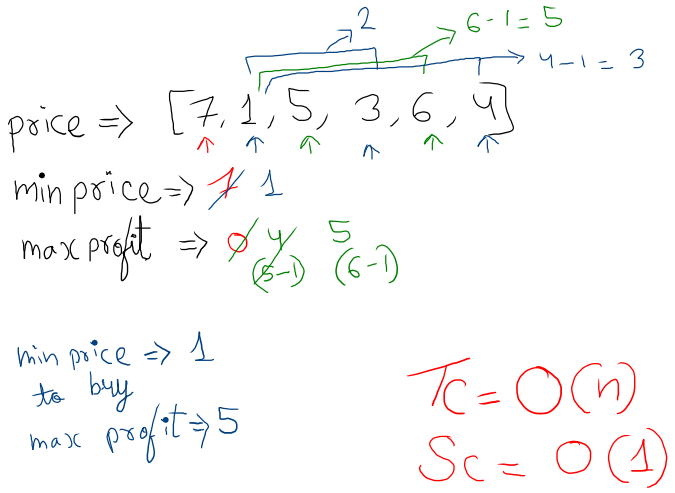
[3, 4, 5, 8] [1, 2, 6]

[1, 2, 3, 4, 5, 6, 8]

Count Inversions in an array | Set 1 (Using Merge Sort)

```java
import java.util.*;
import java.io.*;

public class Main {

  public static void main(String[]args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
      arr[i] = scn.nextInt();
    }
    mergeSort(arr, 0, n - 1);
    System.out.println(count);
  }

  static int count = 0;
  public static int[] merge2SortedArrays(int[] left, int[] right) {

    int i = 0;
    int j = 0;
    int k = 0;
    int[] merged = new int[left.length + right.length];

    while (i < left.length && j < right.length) {
      if (left[i] <= right[j]) {
        merged[k] = left[i];
        i++;
        k++;
      } else {
        count += left.length - i;
        merged[k] = right[j];
        j++;
        k++;
      }
    }
    while (i < left.length) {
      merged[k] = left[i];
      i++;
      k++;
    }
    while (j < right.length) {
      merged[k] = right[j];
      j++;
      k++;
    }

    return merged;
  }
  public static int[] mergeSort(int[] arr, int lo, int hi) {
    if (lo == hi) {
      int[] ba = new int[1];
      ba[0] = arr[lo];
      return ba;
    }

    int mid = (lo + hi) / 2;

    int[] left = mergeSort(arr, lo, mid);
    int[] right = mergeSort(arr, mid + 1, hi);

    int[] merged = merge2SortedArrays(left, right);

    return merged;
  }

}
```

Stock Buy and Sell



min price => 1
to buy
max profit => 5

TC = O(n)
SC = O(1)

121. Best Time to Buy and Sell Stock

Easy 10950 424 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i`th day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

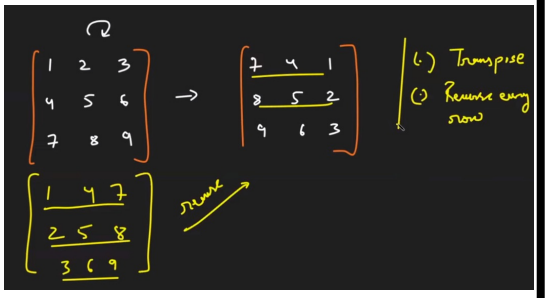
Return *the maximum profit you can achieve from this transaction.* If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

```java
class Solution {
    public int maxProfit(int[] prices) {
        int minprice = Integer.MAX_VALUE;
        int maxprofit = 0;
        for (int i = 0; i < prices.length; i++) {
            if (prices[i] < minprice)
                minprice = prices[i];
            else if (prices[i] - minprice > maxprofit)
                maxprofit = prices[i] - minprice;
        }
        return maxprofit;
    }
}
```

Rotate Matrix



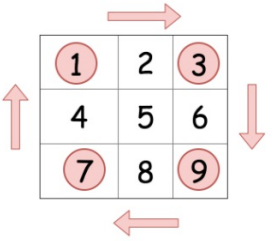

```java
class Solution {
    // time complexity = O(M)
    // space complexity = O(1)

    public void rotate(int[][] matrix) {
        transpose(matrix);
        reflect(matrix);
    }

    public void reverse(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n; i++) {
            for (int j = i; j < n; j++) {
                int tmp = matrix[j][i];
                matrix[j][i] = matrix[i][j];
                matrix[i][j] = tmp;
            }
        }
    }

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

Tc => O(n^2)
Sc => O(1)

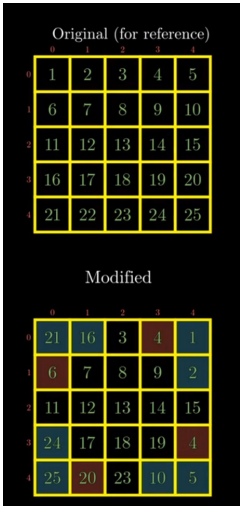
Approach 1: Rotate Groups of Four Cells

Intuition

Observe how the cells move in groups when we rotate the image.



We can iterate over each group of four cells and rotate them.



Tc => O(n^2)
Sc => O(1)

```java
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < (n + 1) / 2; i ++) {
            for (int j = 0; j < n / 2; j++) {
                int temp = matrix[n - 1 - j][i];
                matrix[n - 1 - j][i] = matrix[n - 1 - i][n - j - 1];
                matrix[n - 1 - i][n - j - 1] = matrix[j][n - 1 -i];
                matrix[j][n - 1 - i] = matrix[i][j];
                matrix[i][j] = temp;
            }
        }
    }
}
```

Search in a 2D matrix

GfG :-

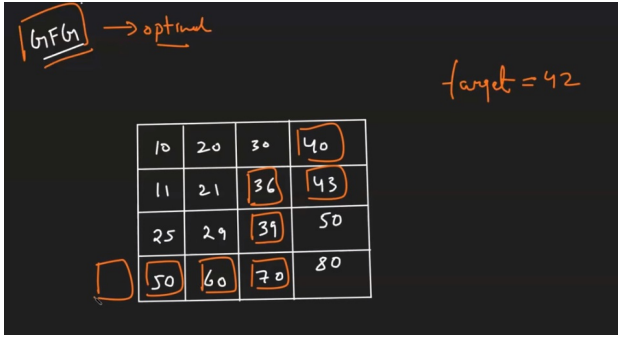
Search in a row wise and column wise sorted matrix

Difficulty Level : Medium • Last Updated : 24 May, 2021

Given an n x n matrix and a number x, find the position of x in the matrix if it is present in it. Otherwise, print "Not Found". In the given matrix, every row and column is sorted in increasing order. The designed algorithm should have linear time complexity.

Example:

```
Input: mat[4][4] = { {10, 20, 30, 40},
                     {15, 25, 35, 45},
                     {27, 29, 37, 48},
                     {32, 33, 39, 50}};
            x = 29
Output: Found at (2, 1)
Explanation: Element at (2,1) is 29
```




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

Leetcode :-

74. Search a 2D Matrix

Medium 4532 218 Add to List Share

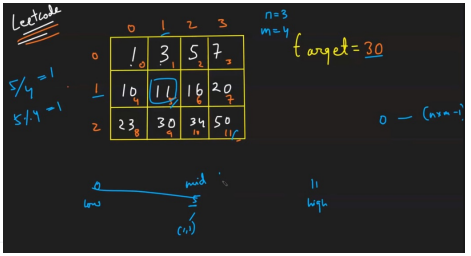
Write an efficient algorithm that searches for a value in an `m x n` matrix. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

We are doing binary search from the index 0 to (n*c)-1 i.e. 0 to 11

Example 1:

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




```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if(matrix.length == 0) return false;

        int n = matrix.length;
        int m = matrix[0].length;

        int lo = 0;
        int hi = (n*m)-1;

        while(lo<=hi){
            int mid = lo + (hi-lo)/2;
            if(matrix[mid/m][mid%m] == target){
                return true;
            }
            if(matrix[mid/m][mid%m] < target){
                lo = mid+1;
            }else{
                hi = mid-1;
            }
        }

        return false;
    }
}
```

Pow(X,n)

50. Pow(x, n)

Medium 3073 4308 Add to List Share

Implement pow(x, n), which calculates `x` raised to the power `n` (i.e., x^n).

Example 1:

```
Input: x = 2.00000, n = 10
Output: 1024.00000
```

Example 2:

```
Input: x = 2.10000, n = 3
Output: 9.26100
```

Example 3:

```
Input: x = 2.00000, n = -2
Output: 0.25000
Explanation: 2^-2 = 1/2^2 = 1/4 = 0.25
```

$\rightarrow 2^{10} = (2\times2)^5 = (4)^5$

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

$\rightarrow (4)^4 = (4\times4)^2 = (16)^2$

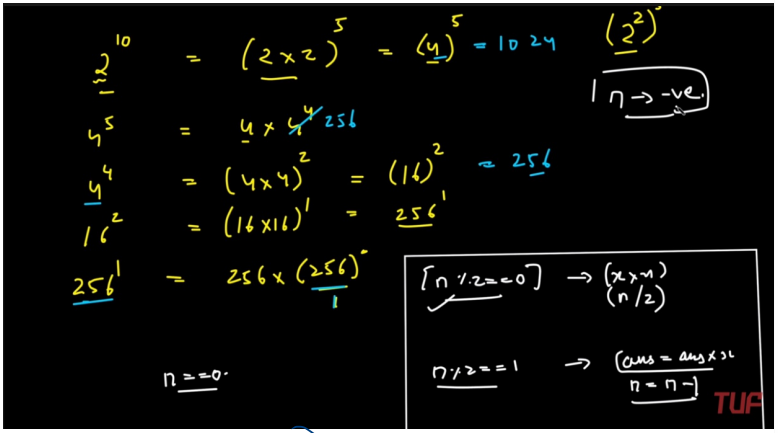
$\rightarrow (16)^2 = (16\times16)^1 = (256)^1$

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

$TC = O(\log_2 n)$

[n%2 == 0] → $(x\times x)^{n/2}$

[n%2 != 0] → Ans = Ans × x, n = n−1



$TC = O(\log_2 n)$

```java
class Solution {
    public double myPow(double x, int n) {
        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 (>N/2 times)

169. Majority Element

Easy 6488 287 Add to List Share

Given an array `nums` of size `n`, return *the majority element.*

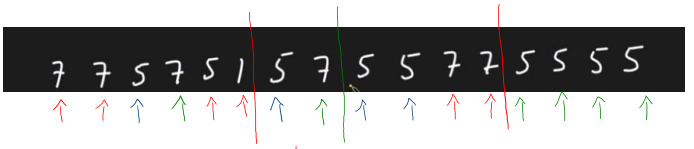
The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

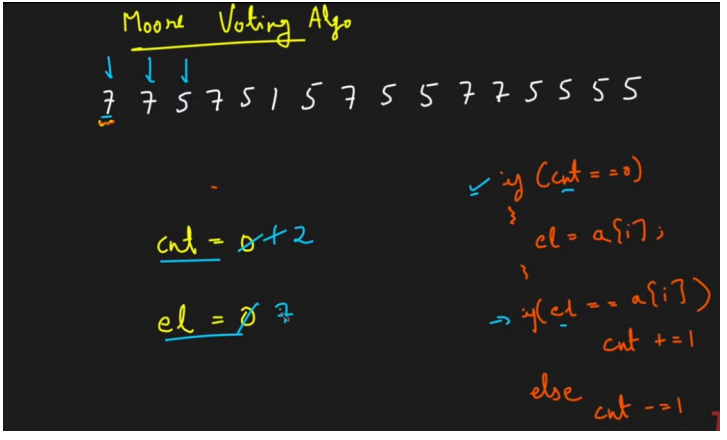
```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```




```
if(count==0){
    element = a[i];
}if(element == a[i]){
    count += 1;
}else{
    count -= 1;
}
```




```java
class Solution {
    public int majorityElement(int[] nums) {
        int count = 0;
        int candidate = 0;

        for (int num : nums) {
            if (count == 0) {
                candidate = num;
            }
            if(num==candidate) count += 1;
            else count -= 1;
        }

        return candidate;
    }
}
```

Majority Element (>N/3 times)

229. Majority Element II

Medium 3464 240 Add to List Share

Given an integer array of size `n`, find all elements that appear more than `⌊ n/3 ⌋` times.

Follow-up: Could you solve the problem in linear time and in O(1) space?

Example 1:

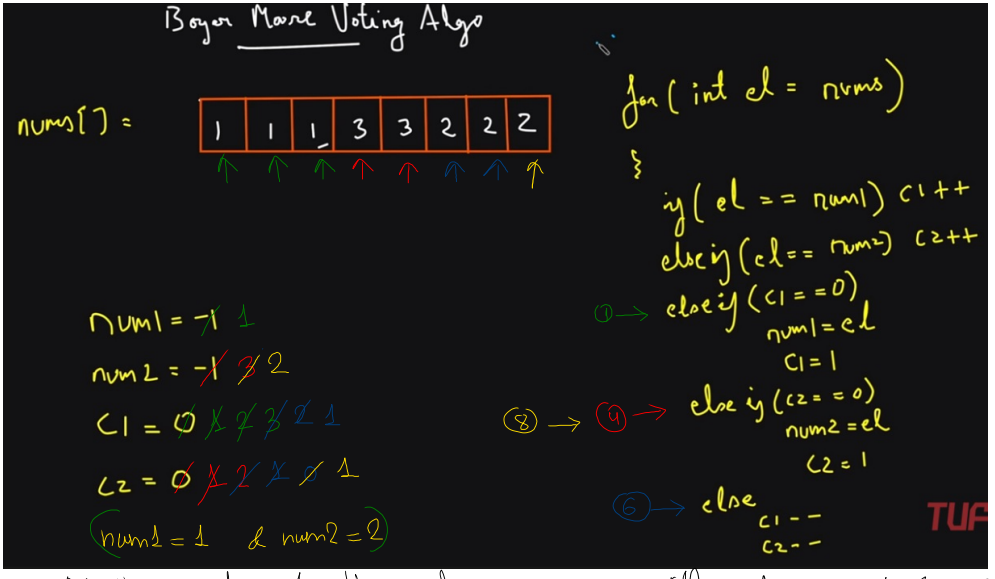
```
Input: nums = [3,2,3]
Output: [3]
```

Example 2:

```
Input: nums = [1]
Output: [1]
```

Example 3:

```
Input: nums = [1,2]
Output: [1,2]
```



$TC \simeq O(n) + O(n)$

$SC = O(1)$

At the end of the loop we will get num1 (n_1) & num2 (n_2)
So, now we will loop again in the array and find the counts of the n_1 & n_2
And if they are greater than $\frac{n}{3}$. then those will be the answer.

```java
class Solution {
    public List<Integer> majorityElement(int[] nums) {
        int n1=-1, n2=-1, c1=0, c2=0, len=nums.length;
        for(int i=0; i<len; i++){
            if(nums[i]==n1){
                c1++;
            }else if(nums[i]==n2){
                c2++;
            }else if(c1==0){
                n1 = nums[i];
                c1 = 1;
            }else if(c2==0){
                n2 = nums[i];
                c2 = 1;
            }else{
                c1--;
                c2--;
            }
        }
        c1=0;
        c2=0;
        for(int num: nums){
            if(num==n1){
                c1++;
            }else if(num==n2){
                c2++;
            }
        }

        List<Integer> ans = new ArrayList<Integer>();
        if(c1 > len/3){
            ans.add(n1);
        }
        if(c2 > len/3){
            ans.add(n2);
        }

        return ans;
    }
}
```

Grid Unique Paths

62. Unique Paths

Medium 6428 259 Add to List Share

A robot is located at the top-left corner of a `m x n` grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

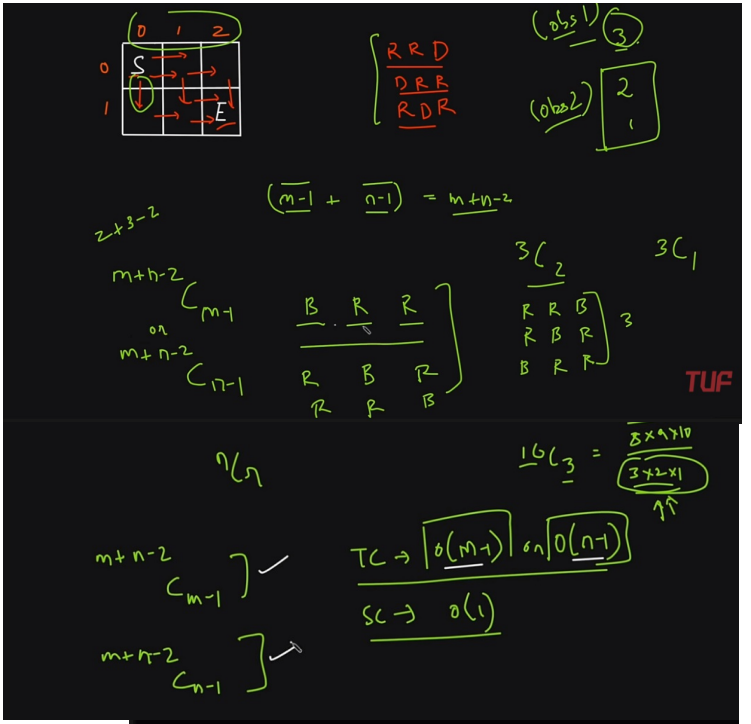
How many possible unique paths are there?

Example 1:


```
Input: m = 3, n = 7
Output: 28
```

Example 2:

```
Input: m = 3, n = 2
Output: 3
Explanation:
From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```




```java
class Solution {
    public int uniquePaths(int m, int n) {

        int N = n + m - 2;
        int r = m - 1;
        double res = 1;

        for (int i = 1; i <= r; i++)
            res = res * (N - r + i) / i;

        return (int)res;
    }
}
```

Reverse Pairs (Leetcode)

493. Reverse Pairs

Hard 1962 161 Add to List Share

Given an integer array `nums`, return *the number of* ***reverse pairs*** *in the array.*

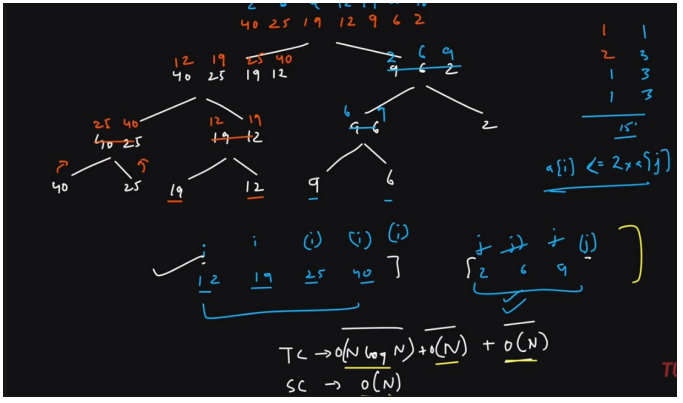
A reverse pair is a pair `(i, j)` where `0 <= i < j < nums.length` and `nums[i] > 2 * nums[j]`.

Example 1:

```
Input: nums = [1,3,2,3,1]
Output: 2
```

Example 2:

```
Input: nums = [2,4,3,5,1]
Output: 3
```




```java
class Solution {
    public int reversePairs(int[] nums) {
        return mergeSort(nums, 0, nums.length-1);
    }
    int mergeSort(int[] nums, int lo, int hi){
        if(lo>=hi) return 0;
        int mid = (lo+hi)/2;
        int inv = mergeSort(nums, lo, mid);
        inv += mergeSort(nums, mid+1, hi);
        inv += merge(nums, lo, mid, hi);
        return inv;
    }
    int merge(int[] nums, int low, int mid, int high){
        int cnt=0;
        int j=mid+1;

        for(int i=low; i<=mid; i++){
            while(j<=high && nums[i]>2*(long)nums[j]){
                j++;
            }
            cnt += (j-(mid+1));
        }

        ArrayList<Integer> temp = new ArrayList();
        int left=low, right=mid+1;
        while(left<=mid && right<=high){
            if(nums[left]<=nums[right]){
                temp.add(nums[left++]);
            }else{
                temp.add(nums[right++]);
            }
        }
        while(left<=mid){
            temp.add(nums[left++]);
        }
        while(right<=high){
            temp.add(nums[right++]);
        }
        for(int i=low; i<=high; i++){
            nums[i] = temp.get(i-low);
        }
        return cnt;
    }
}
```

```java
public class Solution {
  int Rp = 0;

  void merge(int [] res, int [] A, int B[]) {
    int n = A.length;
    int m = B.length;
    int i = 0;
    int j = 0;
    int k = 0;
    while (i < n && j < m) {
      if ((long)A[i] > 2 * (long)B[j]) {
        j++;
        Rp = Rp + (n - i);
      } else {
        i++;
      }
    }
    i = 0;
    j = 0;
    while (i < n && j < m) {
      if (A[i] <= B[j]) {
        res[k++] = A[i++];
      } else {
        res[k++] = B[j++];
      }
    }
    while (i < n) {
      res[k++] = A[i++];
    }
    while (j < m) {
      res[k++] = B[j++];
    }
  }
  void mergeSort(int [] A) {
    int n = A.length;
    if (n < 2) {
      return;
    }
    int mid = n / 2;
    int n1 = mid;
    int n2 = n - mid;
    int [] left = new int [n1];
    int [] right = new int [n2];
    for (int i = 0; i < n1; i++) {
      left[i] = A[i];
    }
    int j = 0;
    for (int i = (int)mid; i < n; i++) {
      right[j++] = A[i];
    }
    mergeSort(right);
    mergeSort(left);
    merge(A, left, right);
  }
  public int reversePairs(int[] A) {
    mergeSort(A);
    return Rp;
  }
}
```

2 Sum problem

1. Two Sum

Easy 24756 815 Add to List Share

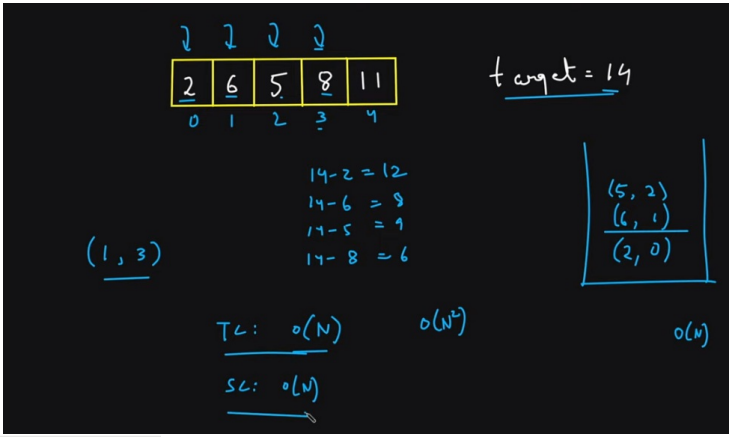
Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

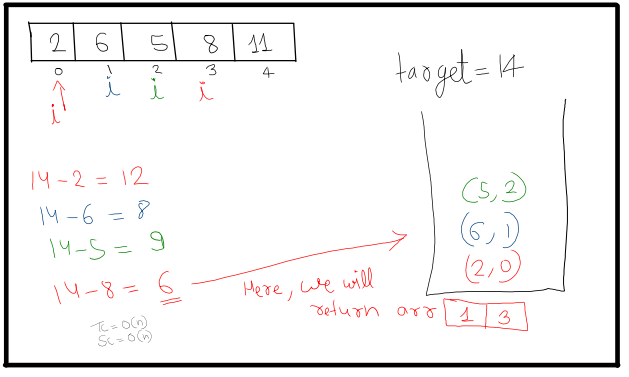
You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Output: Because nums[0] + nums[1] == 9, we return [0, 1].
```




```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer,Integer>  map = new HashMap();
        for(int i=0; i<nums.length; i++){
            map.put(nums[i],i);
        }
        for(int i=0; i<nums.length; i++){
            if(map.containsKey(target-nums[i]) && i!=map.get(target-nums[i])){
                return new int[] {i,map.get(target-nums[i])};
            }
        }
        return new int[2];
    }
}
```




```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer,Integer>  map = new HashMap();
        for(int i=0; i<nums.length; i++){
            if(map.containsKey(target-nums[i]) && i!=map.get(target-nums[i])){
                return new int[] {i,map.get(target-nums[i])};
            }
            map.put(nums[i],i);
        }
        return new int[2];
    }
}
```

4 Sum problem

18. 4Sum

Medium 4495 530 Add to List Share

Given an array `nums` of `n` integers, return *an array of all the* ***unique*** *quadruplets* `[nums[a], nums[b], nums[c], nums[d]]` such that:

- `0 <= a, b, c, d < n`
- `a, b, c,` and `d` are **distinct**.
- `nums[a] + nums[b] + nums[c] + nums[d] == target`

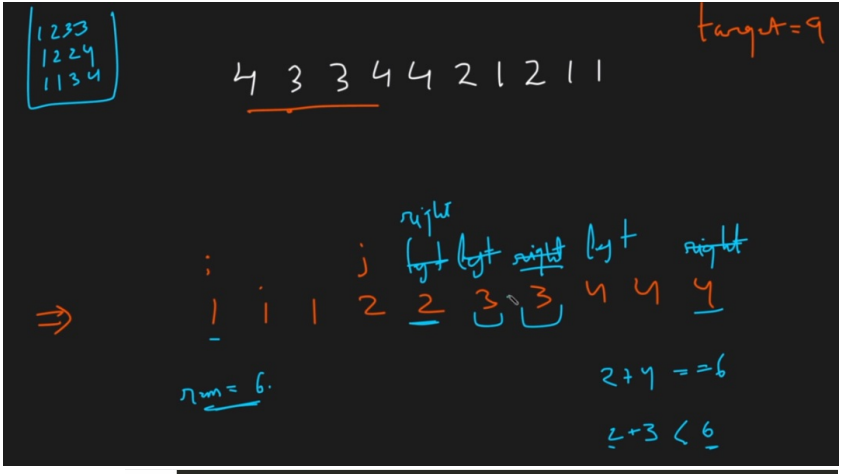
You may return the answer in **any order**.

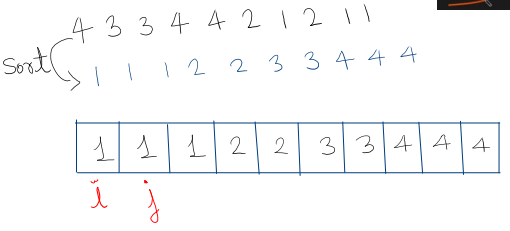
Example 1:

```
Input: nums = [1,0,-1,0,-2,2], target = 0
Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]
```

Example 2:

```
Input: nums = [2,2,2,2,2], target = 8
Output: [[2,2,2,2]]
```






```java
class Solution {
    public List<List<Integer>> fourSum(int[] num, int target) {
        ArrayList<List<Integer>> res = new ArrayList<List<Integer>>();

        if(num==null || num.length==0){
            return res;
        }
        int n = num.length;
        Arrays.sort(num);

        for(int i=0; i<n; i++){
            for(int j=i+1; j<n; j++){
                int target_2 = target - num[j] - num[i];

                int front = j+1;
                int back = n-1;

                while(front < back){
                    int two_sum = num[front] + num[back];
                    if(two_sum < target_2) front++;
                    else if(two_sum > target_2) back--;
                    else{
                        List<Integer> quad = new ArrayList();
                        quad.add(num[i]);
                        quad.add(num[j]);
                        quad.add(num[front]);
                        quad.add(num[back]);
                        res.add(quad);

                        while(front<back && num[front]==quad.get(2)) ++front;
                        while(front<back && num[back]==quad.get(3)) --back;

                    }
                }
                while(j+1<n && num[j+1]==num[j]) ++j;
            }
            while(i+1<n && num[i+1]==num[i]) ++i;

        }
        return res;
    }
}
```

Longest Consecutive Sequence

128. Longest Consecutive Sequence

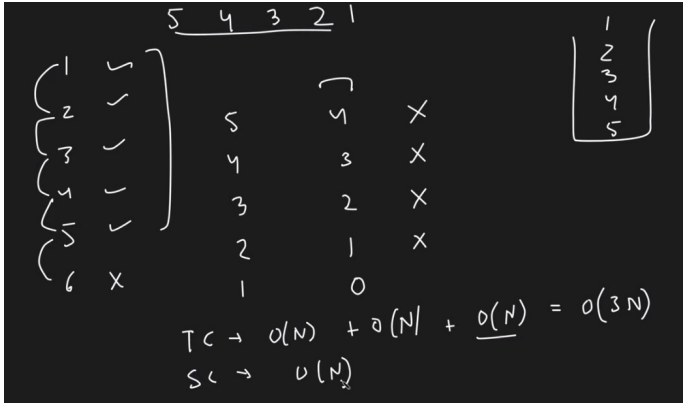
Medium 6907 318 Add to List Share

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in `O(n)` time.

Example 1:

```
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4].
Therefore its length is 4.
```




```java
class Solution {
    public int longestConsecutive(int[] nums) {
        Set<Integer> num_set = new HashSet<Integer>();
        for (int num : nums) {
            num_set.add(num);
        }

        int longestStreak = 0;

        for (int num : num_set) {
            if (!num_set.contains(num-1)) {
                int currentNum = num;
                int currentStreak = 1;

                while (num_set.contains(currentNum+1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }

                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }

        return longestStreak;
    }
}
```

Largest Subarray with 0 sum

Largest subarray with 0 sum

Easy Accuracy: 46.94% Submissions: 68721 Points: 2

Given an array having both positive and negative integers. The task is to compute the length of the largest subarray with sum 0.

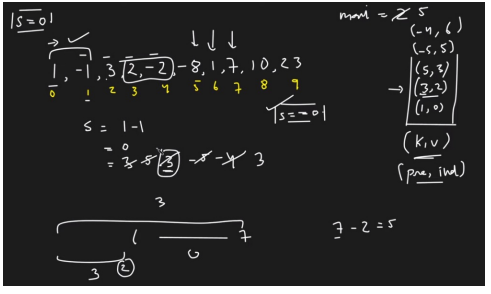
Example 1:

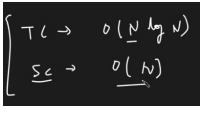
```
Input:
N = 8
A[] = {15,-2,2,-8,1,7,10,23}
Output: 5
Explanation: The largest subarray with
sum 0 will be -2 2 -8 1 7.
```

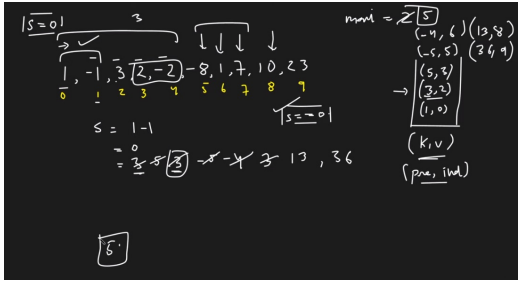
Your Task:
You just have to complete the function **maxLen()** which takes two arguments an array **A** and **n,** where n is the size of the array A and returns the length of the largest subarray with 0 sum.

Expected Time Complexity: O(N).
Expected Auxiliary Space: O(N).

Constraints:
1 <= N <= 10^5
-1000 <= A[i] <= 1000, for each valid i








```java
int maxLen(int A[], int n)
{
    // Your code here
    HashMap<Integer, Integer> mpp = new HashMap<Integer, Integer>();
    int maxi = 0;
    int sum = 0;
    for(int i = 0;i<n;i++) {
        sum += A[i];
        if(sum == 0) {
            maxi = i + 1;
        }
        else {
            if(mpp.get(sum) != null) {
                maxi = Math.max(maxi, i - mpp.get(sum));
            }
            else {
                mpp.put(sum, i);
            }
        }
    }
    return maxi;
}
```

Count number of subarrays with given XOR(this clears a lot of problems)

Count the number of subarrays having a given XOR

Difficulty Level : Hard • Last Updated : 13 May, 2021

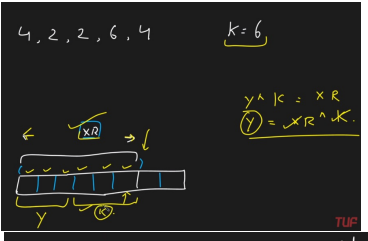
Given an array of integers arr[] and a number m, count the number of subarrays having XOR of their elements as m.

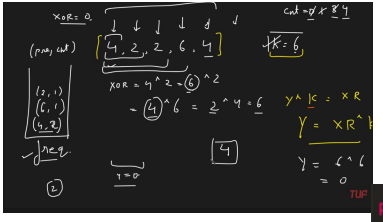
Examples:

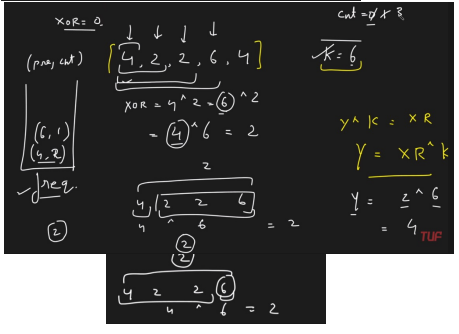
```
Input : arr[] = {4, 2, 2, 6, 4}, m = 6
Output : 4
Explanation : The subarrays having XOR of
              their elements as 6 are {4, 2},
              {4, 2, 2, 6, 4}, {2, 2, 6},
               and {6}

Input : arr[] = {5, 6, 7, 8, 9}, m = 5
Output : 2
Explanation : The subarrays having XOR of
              their elements as 5 are {5}
              and {5, 6, 7, 8, 9}
```

TC → O(N log N)

SC → O(N)








```java
public class Solution {
    public int solve(int[] A, int B) {
        HashMap<Integer, Integer> freq = new HashMap<Integer, Integer>();
        int cnt = 0;
        int xorr = 0;
        int n = A.length;
        for(int i = 0;i<n;i++) {
            xorr = xorr ^ A[i];
            if(freq.get(xorr ^ B) != null)
                cnt += freq.get(xorr ^ B);
            if(xorr==B)  {
                cnt++;
            }
            if(freq.get(xorr) != null)
                freq.put(xorr, freq.get(xorr) + 1);
            else freq.put(xorr, 1);
        }
        return cnt;
    }
}
```

Longest substring without repeat

3. Longest Substring Without Repeating Characters

Medium 17651 823 Add to List Share

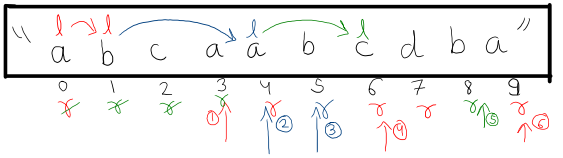
Given a string `s`, find the length of the **longest substring** without repeating characters.

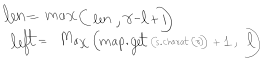
Example 1:

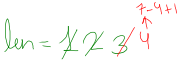
```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

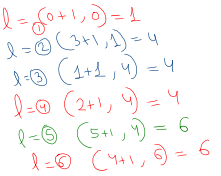
Example 2:

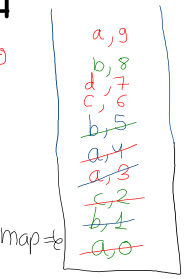
```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```











len = 4.

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        HashMap<Character, Integer> mpp = new HashMap<Character, Integer>();
        int left=0,right=0;
        int n=s.length();
        int len=0;
        while(right<n){
            if(mpp.containsKey(s.charAt(right))){
                left = Math.max(mpp.get(s.charAt(right)) + 1, left);
            }
            mpp.put(s.charAt(right),right);
            len = Math.max(len, right-left+1);
            right++;
        }
        return len;
    }
}
```

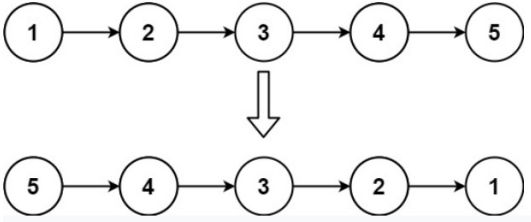
Reverse a LinkedList

206. Reverse Linked List

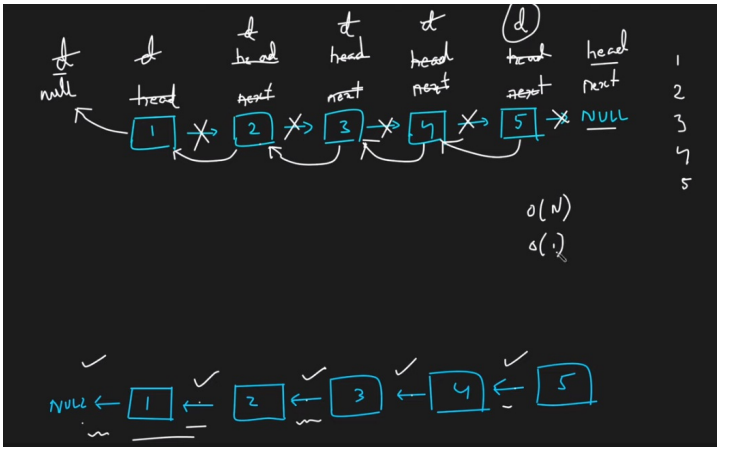
Easy 8603 152 Add to List Share

Given the `head` of a singly linked list, reverse the list, and return *the reversed list.*

Example 1:




```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```




```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode newHead = null;
        while (head != null) {
            ListNode next = head.next;
            head.next = newHead;
            newHead = head;
            head = next;
        }
        return newHead;
    }
}
```

Find middle of LinkedList

876. Middle of the Linked List

Easy 3256 88 Add to List Share

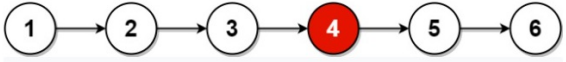
Given the `head` of a singly linked list, return *the middle node of the linked list.*

If there are two middle nodes, return **the second middle** node.

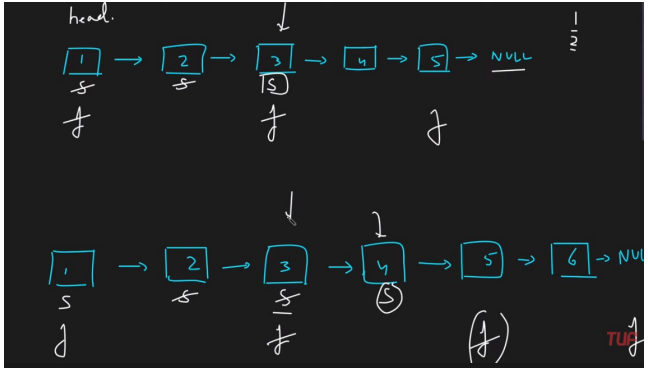
Example 1:


```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:




```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we
return the second one.
```




```java
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

Complexity Analysis

- Time Complexity: $O(N)$, where N is the number of nodes in the given list.
- Space Complexity: $O(1)$, the space used by `slow` and `fast`.

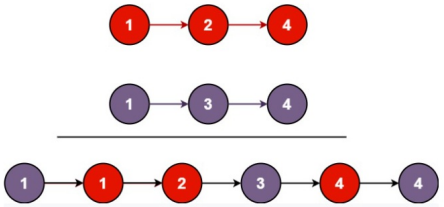
Merge two sorted Linked List

21. Merge Two Sorted Lists

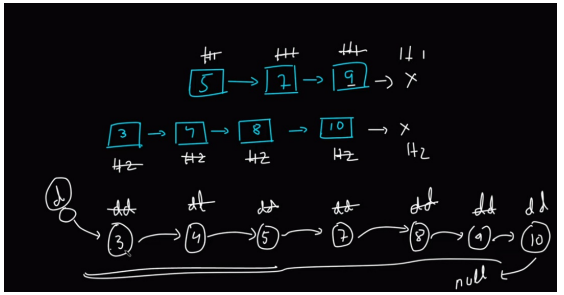
Easy 8447 866 Add to List Share

Merge two sorted linked lists and return it as a **sorted** list. The list should be made by splicing together the nodes of the first two lists.

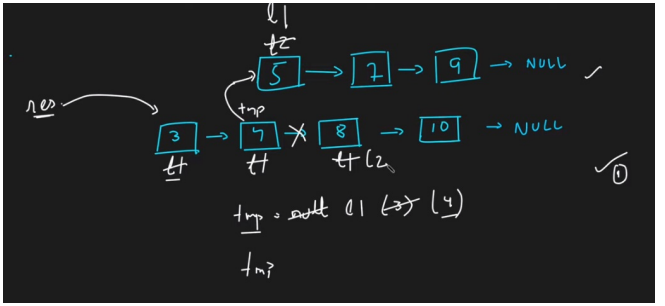
Example 1:

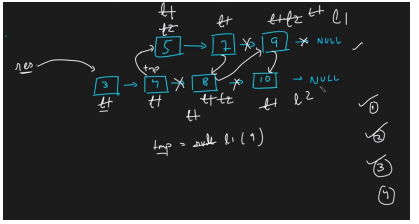



```
Input: l1 = [1,2,4], l2 = [1,3,4]
Output: [1,1,2,3,4,4]
```



```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2){
        if(l1==null || l2==null) return l1!=null?l1:l2;
        ListNode dummy = new ListNode(-1);
        ListNode prev = dummy;
        ListNode c1 = l1;
        ListNode c2 = l2;
        while(c1!=null && c2!=null){
            if(c1.val<c2.val){
                prev.next = c1;
                c1 = c1.next;
            }else{
                prev.next = c2;
                c2 = c2.next;
            }
            prev = prev.next;
        }
        prev.next = (c1!=null? c1:c2);

        return dummy.next;

    }
}
```






```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        // iterative
        if (l1 == null) return l2;
        if (l2 == null) return l1;
        if (l1.val > l2.val) {
            ListNode temp = l1;
            l1 = l2;
            l2 = temp;
        }
        ListNode res = l1;
        while (l1 != null && l2 != null) {
            ListNode tmp = null;
            while (l1 != null && l1.val <= l2.val) {
                tmp = l1;
                l1 = l1.next;
            }
            tmp.next = l2;

            // swap
            ListNode temp = l1;
            l1 = l2;
            l2 = temp;
        }
        return res;
    }
}
```

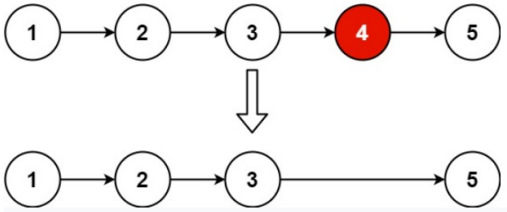
Remove N-th node from back of LinkedList

19. Remove Nth Node From End of List

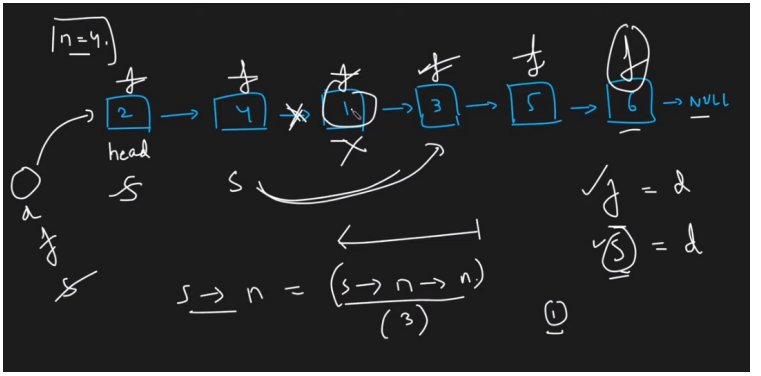
Medium 6910 362 Add to List Share

Given the `head` of a linked list, remove the n^{th} node from the end of the list and return its head.

Example 1:




```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```




```java
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        ListNode start = new ListNode();
        start.next = head;
        ListNode fast = start;
        ListNode slow = start;

        for(int i=1; i<=n; i++){
            fast = fast.next;
        }
        while(fast.next != null){
            fast = fast.next;
            slow = slow.next;
        }
        slow.next = slow.next.next;

        return start.next;
    }
}
```

Delete a given Node when a node is given. (0(1) solution)

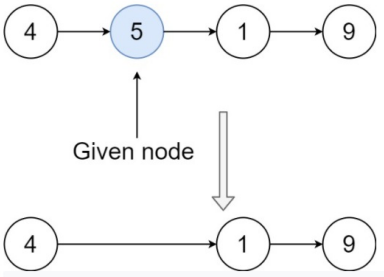
237. Delete Node in a Linked List

Easy 3256 10099 Add to List Share

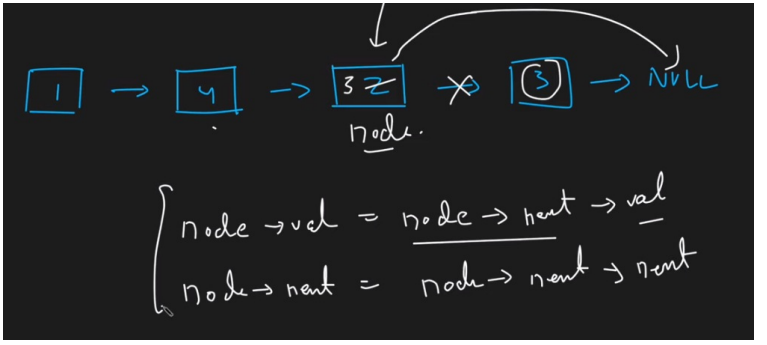
Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the `head` of the list, instead you will be given access to **the node to be deleted** directly.

It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

Example 1:




```
Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list
should become 4 -> 1 -> 9 after calling your function.
```



Add two numbers as LinkedList

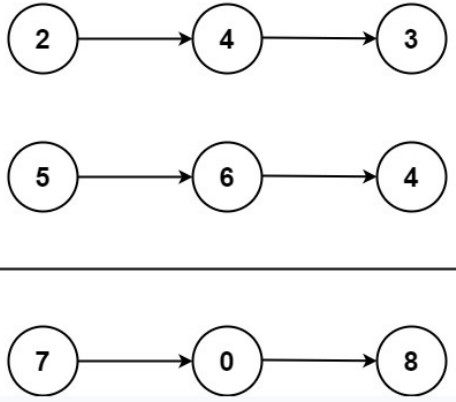
2. Add Two Numbers

Medium 14024 3125 Add to List Share

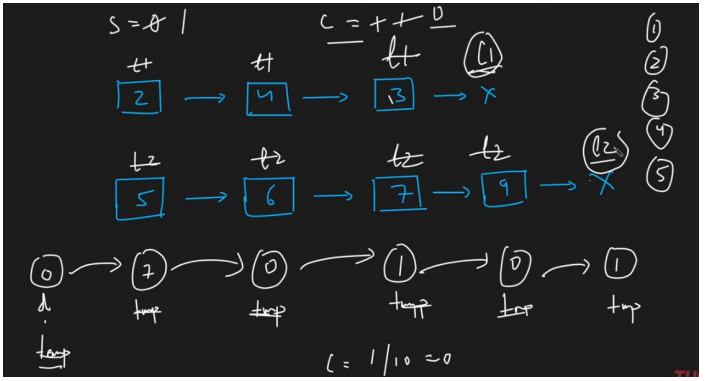
You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:




```
Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.
```



```java
class Solution {
    public ListNode addTwoNumbers(ListNode c1, ListNode c2) {
        if(c1==null || c2==null){
            return c1!=null?c1:c2;
        }
        ListNode head = new ListNode(-1);
        ListNode itr = head;
        int carry = 0;
        while(c1!=null || c2 !=null || carry!=0){
            int sum = carry + (c1!=null?c1.val:0) + (c2!=null?c2.val:0);
            int ld = sum%10;
            carry = sum/10;
            itr.next = new ListNode(ld);
            itr = itr.next;
            if(c1!=null) c1 = c1.next;
            if(c2!=null) c2 = c2.next;
        }
        return head.next;
    }
}
```

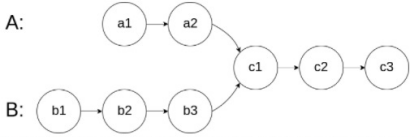
Find intersection point of Y LinkedList

160. Intersection of Two Linked Lists

Easy 6762 721 Add to List Share

Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

For example, the following two linked lists begin to intersect at node `c1`:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

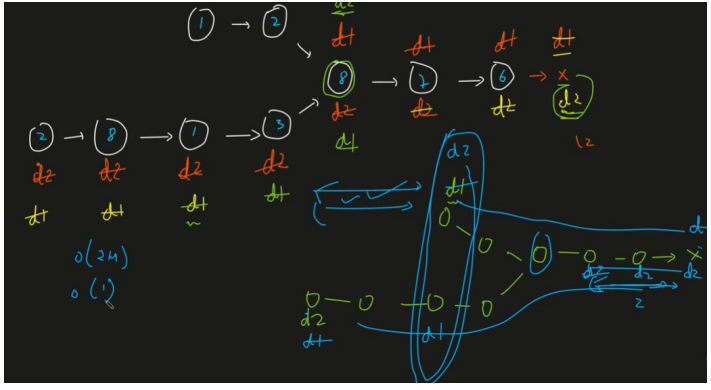
Note that the linked lists must **retain their original structure** after the function returns.

Custom Judge:

The inputs to the **judge** are given as follows (your program is **not** given these inputs):

- `intersectVal` - The value of the node where the intersection occurs. This is `0` if there is no intersected node.
- `listA` - The first linked list.
- `listB` - The second linked list.
- `skipA` - The number of nodes to skip ahead in `listA` (starting from the head) to get to the intersected node.
- `skipB` - The number of nodes to skip ahead in `listB` (starting from the head) to get to the intersected node.

The judge will then create the linked structure based on these inputs and pass the two heads, `headA` and `headB` to your program. If you correctly return the intersected node, then your solution will be **accepted**.



```java
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        if(headA == null || headB == null) return null;

        ListNode a = headA;
        ListNode b = headB;

        //if a & b have different len, then we will stop the loop after second iteration
        while( a != b){
            //for the end of first iteration, we just reset the pointer to the head of another linkedlist
            a = a == null? headB : a.next;
            b = b == null? headA : b.next;
        }

        return a;

    }
}
```

Detect a cycle in Linked List

141. Linked List Cycle

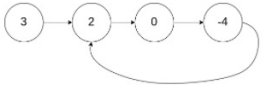
Easy 5584 657 Add to List Share

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

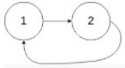
Return `true` *if there is a cycle in the linked list.* Otherwise, return `false`.

Example 1:

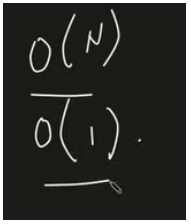


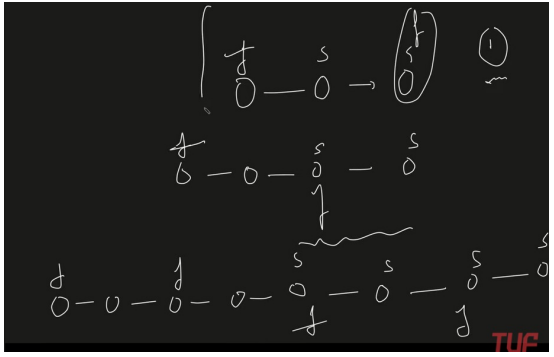

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to
the 1st node (0-indexed).
```

Example 2:




```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to
the 0th node.
```






```java
public class Solution {
    public boolean hasCycle(ListNode head) {
        if(head == null || head.next == null)
            return false;

        ListNode fast = head;
        ListNode slow = head;

        while(fast.next!=null && fast.next.next!=null){
            fast = fast.next.next;
            slow = slow.next;
            if(fast == slow)
                return true;
        }

        return false;
    }
}
```

Reverse a LinkedList in groups of size k.

25. Reverse Nodes in k-Group

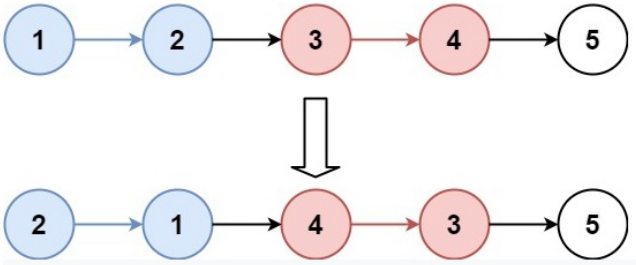
Hard 5018 444 Add to List Share

Given a linked list, reverse the nodes of a linked list *k* at a time and return its modified list.

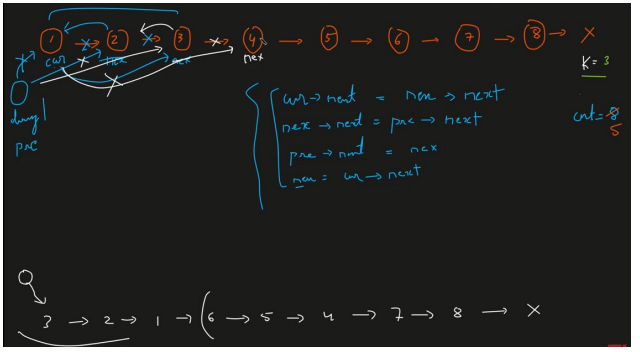
k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of *k* then left-out nodes, in the end, should remain as it is.

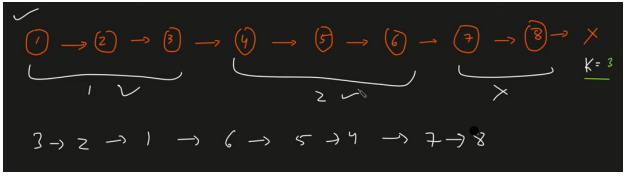
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:




```
Input: head = [1,2,3,4,5], k = 2
Output: [2,1,4,3,5]
```





Check if a LinkedList is palindrome or not.

234. Palindrome Linked List

Easy 6532 476 Add to List Share

Given the `head` of a singly linked list, return `true` if it is a palindrome.

Example 1:


```
Input: head = [1,2,2,1]
Output: true
```

Example 2:


```
Input: head = [1,2]
Output: false
```

```java
class Solution {
    public boolean isPalindrome(ListNode head) {
        if(head==null||head.next==null)
            return true;

        ListNode slow=head;
        ListNode fast=head;

        // find middle of linkedlist
        while(fast.next!=null&&fast.next.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }

        // reverse the right half
        slow.next=reverseList(slow.next);

        // move slow to right half
        slow=slow.next;

        // check for left half right half equal or not
        while(slow!=null){
            if(head.val!=slow.val)
                return false;
            head=head.next;
            slow=slow.next;
        }
        return true;
    }
    ListNode reverseList(ListNode head) {
        ListNode pre=null;
        ListNode next=null;
        while(head!=null){
            next=head.next;
            head.next=pre;
            pre=head;
            head=next;
        }
        return pre;
    }
```

Find the starting point of the Loop of LinkedList

142. Linked List Cycle II

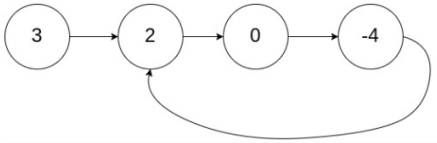
Medium 5146 355 Add to List Share

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

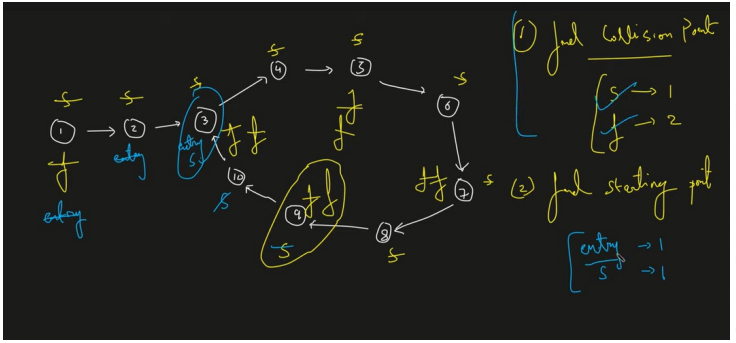
Example 1:

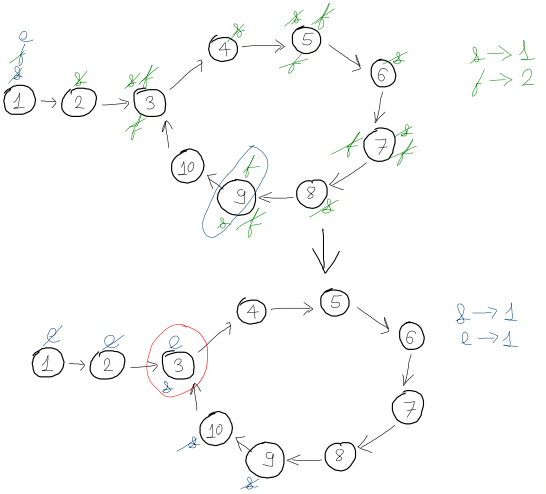



```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.
```

Tc = O(N)

Sc = O(1)






```java
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode detectCycle(ListNode head) {
        if (head == null || head.next == null)
            return null;

        ListNode slow  = head;
        ListNode fast  = head;
        ListNode entry = head;

        while (fast.next!=null && fast.next.next!=null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) {                    // there is a cycle
                while(slow != entry) {             // found the entry location
                    slow  = slow.next;
                    entry = entry.next;
                }
                return entry;
            }
        }
        return null;
    }
}
```

Flattening of a LinkedList

Flattening a Linked List

Medium Accuracy: 33.91% Submissions: 68179 Points: 4

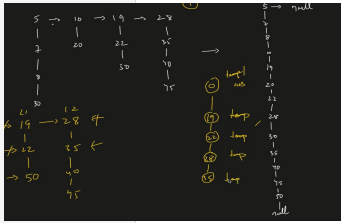
Given a Linked List of size N, where every node represents a sub-linked-list and contains two pointers:
(i) a **next** pointer to the next node,
(ii) a **bottom** pointer to a linked list where this node is head.
Each of the sub-linked-list is in sorted order.
Flatten the Link List such that all the nodes appear in a single level while maintaining the sorted order.
Note: The flattened list will be printed using the bottom pointer instead of next pointer.

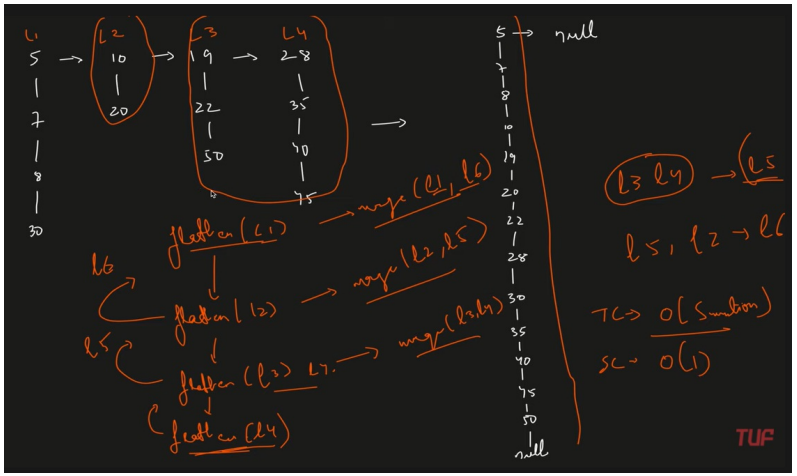
Example 1:

```
Input:
5 -> 10 -> 19 -> 28
|     |     |     |
7     20    22    35
|           |     |
8           50    40
|                 |
30                45
```

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

Explanation:
The resultant linked lists has every node in a single level.
(**Note:** | represents the bottom pointer.)






```java
class GfG
{
    Node mergeTwoLists(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;
        else temp.bottom = b;
        return res.bottom;

    }
    Node flatten(Node root)
    {

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

            // recur for list on right
            root.next = flatten(root.next);

            // now merge
            root = mergeTwoLists(root, root.next);

            // return the root
            // it will be in turn merged with its left
            return root;
    }
```

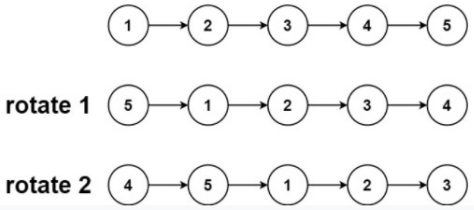
Rotate a LinkedList

61. Rotate List

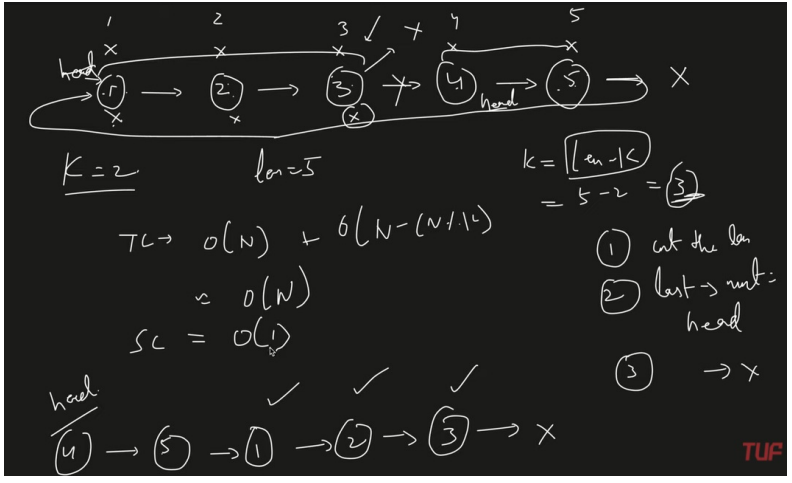
Medium 3181 1197 Add to List Share

Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:




```
Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]
```




```java
class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        // edge cases
        if (head == null || head.next == null|| k == 0) return head;

        // compute the length
        ListNode cur = head;
        int len = 1;
        while (cur.next != null) {
            len++;
            cur = cur.next;
        }


        // go till that node
        cur.next = head;
        k = len - k % len;
        while (k-- > 0) cur = cur.next;

        // make the node head and break connection
        head = cur.next;
        cur.next = null;


        return head;
    }
}
```

Day7: (2-pointer)

1. Clone a Linked List with random and next pointer

138. Copy List with Random Pointer

Medium 6325 875 Add to List Share

A linked list of length `n` is given such that each node contains an additional random pointer, which could point to any node in the list, or `null`.

Construct a **deep copy** of the list. The deep copy should consist of exactly `n` **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the `next` and `random` pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list**.

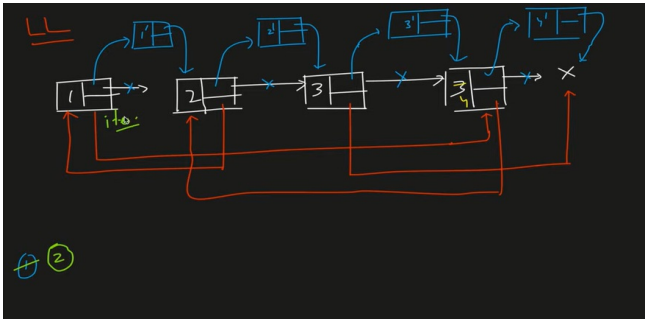
For example, if there are two nodes `X` and `Y` in the original list, where `X.random --> Y`, then for the corresponding two nodes `x` and `y` in the copied list, `x.random --> y`.

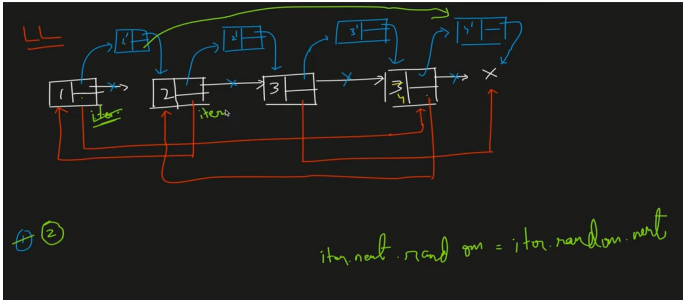
Return *the head of the copied linked list.*

The linked list is represented in the input/output as a list of `n` nodes. Each node is represented as a pair of `[val, random_index]` where:

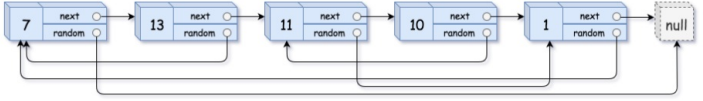
- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from `0` to `n-1`) that the `random` pointer points to, or `null` if it does not point to any node.

Your code will **only** be given the `head` of the original linked list.



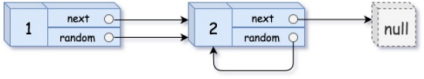


Example 1:

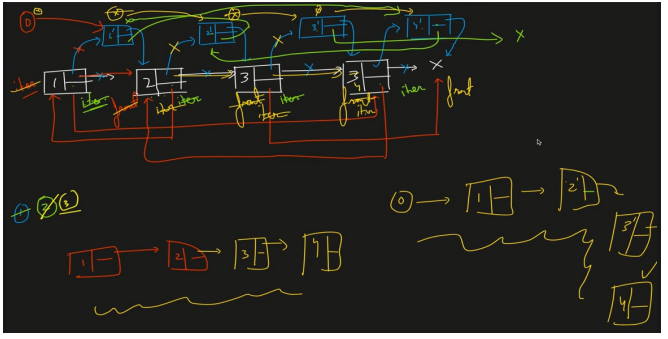



```
Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]
```

Example 2:




```
Input: head = [[1,1],[2,1]]
Output: [[1,1],[2,1]]
```



TC → O(N) + O(N) + O(N) ≈ O(N)

SC → O(1)

1. Clone a Linked List with random and next pointer

```java
/*
// Definition for a Node.
class Node {
    int val;
    Node next;
    Node random;

    public Node(int val) {
        this.val = val;
        this.next = null;
        this.random = null;
    }
}
*/

class Solution {
    public Node copyRandomList(Node head) {
          Node iter = head;
          Node front = head;

          // First round: make copy of each node,
          // and link them together side-by-side in a single list.
          while (iter != null) {
            front = iter.next;

            Node copy = new Node(iter.val);
            iter.next = copy;
            copy.next = front;

            iter = front;
          }

          // Second round: assign random pointers for the copy nodes.
          iter = head;
          while (iter != null) {
            if (iter.random != null) {
              iter.next.random = iter.random.next;
            }
            iter = iter.next.next;
          }

          // Third round: restore the original list, and extract the copy list.
          iter = head;
          Node pseudoHead = new Node(0);
          Node copy = pseudoHead;

          while (iter != null) {
            front = iter.next.next;

            // extract the copy
            copy.next = iter.next;
            copy = copy.next;

            // restore the original list
            iter.next = front;

            iter = front;
          }

          return pseudoHead.next;
    }
}
```

2. 3 sum

15. 3Sum

Medium 13579 1307 Add to List Share

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

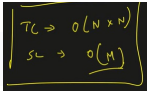
```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
```

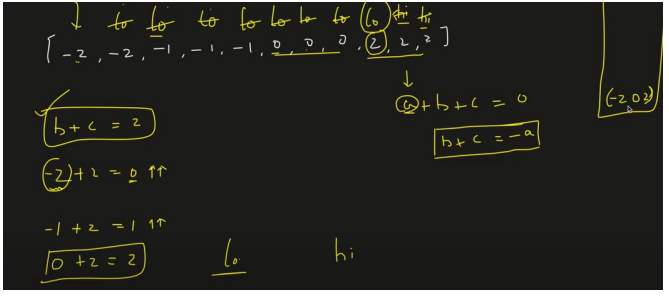
Example 2:

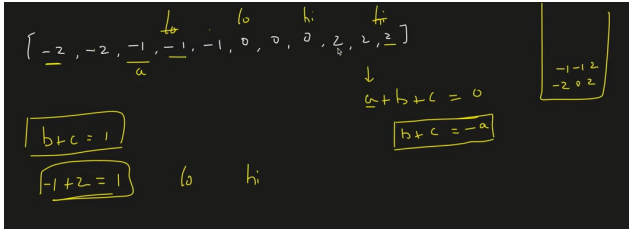
```
Input: nums = []
Output: []
```

Example 3:

```
Input: nums = [0]
Output: []
```








```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res = new LinkedList<>();
        for(int i=0; i<nums.length-2; i++){
            if(i==0 || (i>0 && nums[i]!=nums[i-1])){
                int lo=i+1, hi=nums.length-1, sum=0-nums[i];
                while(lo<hi){
                    if(nums[lo]+nums[hi]==sum){
                        res.add(Arrays.asList(nums[i],nums[lo],nums[hi]));
                        while(lo<hi && nums[lo]==nums[lo+1]) lo++;
                        while(lo<hi && nums[hi]==nums[hi-1]) hi--;
                        lo++;
                        hi--;
                    }
                    else if(nums[lo]+nums[hi]<sum) lo++;
                    else hi--;
                }
            }
        }
        return res;
    }
}
```

3. Trapping rainwater

42. Trapping Rain Water

Hard 14571 204 Add to List Share

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:


```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array
[0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section)
are being trapped.
```

```java
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        int left=0; int right=n-1;
        int res=0;
        int 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[right]>=maxright) maxright= height[right];
                else res+=maxright-height[right];

                right--;
            }
        }
        return res;
    }
}
```

4. Remove Duplicate from Sorted array

26. Remove Duplicates from Sorted Array

Easy 4952 8321 Add to List Share

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

Do **not** allocate extra space for another array. You must do this by **modifying the input array in-place** with O(1) extra memory.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

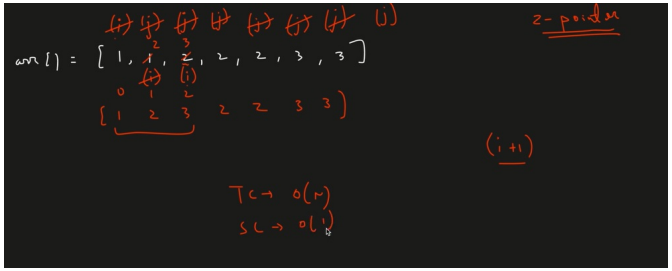
If all assertions pass, then your solution will be **accepted**.

Example 1:

```
Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```



Java Autocomplete

```java
class Solution {
    public int removeDuplicates(int[] nums) {
        if (nums.length == 0)
            return 0;

        int i = 0;

        for (int j = 1; j < nums.length; j++) {
            if (nums[j] != nums[i]) {
                i++;
                nums[i] = nums[j];
            }
        }

        return i + 1;
    }
}
```

5. Max consecutive ones

485. Max Consecutive Ones

Easy 1921 391 Add to List Share

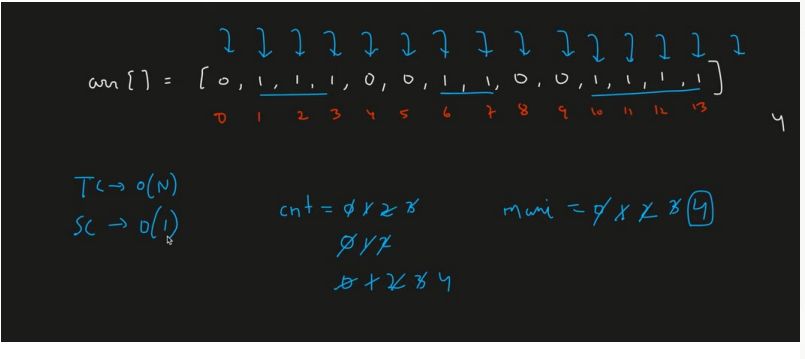
Given a binary array `nums`, return *the maximum number of consecutive* `1`*'s in the array*.

Example 1:

```
Input: nums = [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are
consecutive 1s. The maximum number of consecutive 1s is 3.
```

Example 2:

```
Input: nums = [1,0,1,1,0,1]
Output: 2
```



Java Autocomplete

```java
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int cnt = 0;
        int maxi = 0;
        for(int i = 0;i<nums.length;i++) {
            if(nums[i] == 1) {
                cnt++;
            }
            else {
                cnt = 0;
            }

            maxi = Math.max(maxi, cnt);
        }
        return maxi;
    }
}
```

Day8: (Greedy)

1. N meeting in one room

N meetings in one room

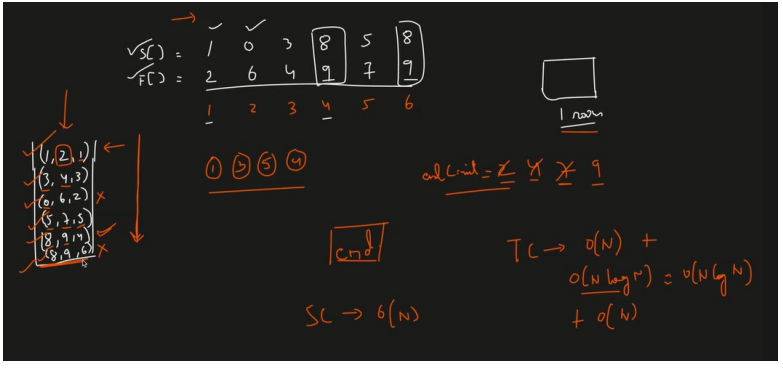
Easy Accuracy: 43.1% Submissions: 54523 Points: 2

There is **one** meeting room in a firm. There are **N** meetings in the form of **(start[i], end[i])** where **start[i]** is start time of meeting **i** and **end[i]** is finish time of meeting **i.**
What is the **maximum** number of meetings that can be accommodated in the meeting room when only one meeting can be held in the meeting room at a particular time?

Note: Start time of one chosen meeting can't be equal to the end time of the other chosen meeting.

Example 1:

```
Input:
N = 6
start[] = {1,3,0,5,8,5}
end[] =  {2,4,6,7,9,9}
Output:
4
Explanation:
Maximum four meetings can be held with
given start and end timings.
```




```java
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;
    }
}
class Meeting {
    static void maxMeetings(int start[], int end[], int n) {
        ArrayList<meeting> meet = new ArrayList<>();

        for(int i = 0; i < start.length; i++)
            meet.add(new meeting(start[i], end[i], i+1));

        meetingComparator mc = new meetingComparator();
        Collections.sort(meet, mc);
        ArrayList<Integer> answer = new ArrayList<>();
        answer.add(meet.get(0).pos);
        int limit = meet.get(0).end;

        for(int i = 1;i<start.length;i++) {
            if(meet.get(i).start > limit) {
                limit = meet.get(i).end;
                answer.add(meet.get(i).pos);
            }
        }

        for(int i = 0;i<answer.size(); i++) {
            System.out.print(answer.get(i) + " ");
        }
    }
}
```

2. Minimum number of platforms required for a railway

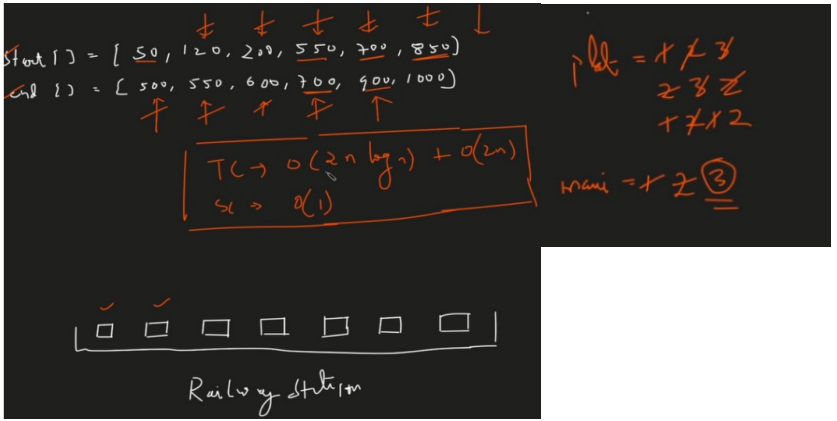
Minimum Platforms

Medium Accuracy: 46.78% Submissions: 78945 Points: 4

Given arrival and departure times of all trains that reach a railway station. Find the minimum number of platforms required for the railway station so that no train is kept waiting.
Consider that all the trains arrive on the same day and leave on the same day. Arrival and departure time can never be the same for a train but we can have arrival time of one train equal to departure time of the other. At any given instance of time, same platform can not be used for both departure of a train and arrival of another train. In such cases, we need different platforms.

Example 1:

```
Input: n = 6
arr[] = {0900, 0940, 0950, 1100, 1500, 1800}
dep[] = {0910, 1200, 1120, 1130, 1900, 2000}
Output: 3
Explanation:
Minimum 3 platforms are required to
safely arrive and depart all trains.
```




```java
class Platform{
    static int findPlatform(int arr[], int dep[], int n){
        Arrays.sort(arr);
        Arrays.sort(dep);

        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        while (i < n && j < n) {
            if (arr[i] <= dep[j]) {
                plat_needed++;
                i++;
            }else if (arr[i] > dep[j]) {
                plat_needed--;
                j++;
            }
            if (plat_needed > result){
                result = plat_needed;
            }
        }
        return result;
    }
}
```

3. Job sequencing Problem

Job Sequencing Problem

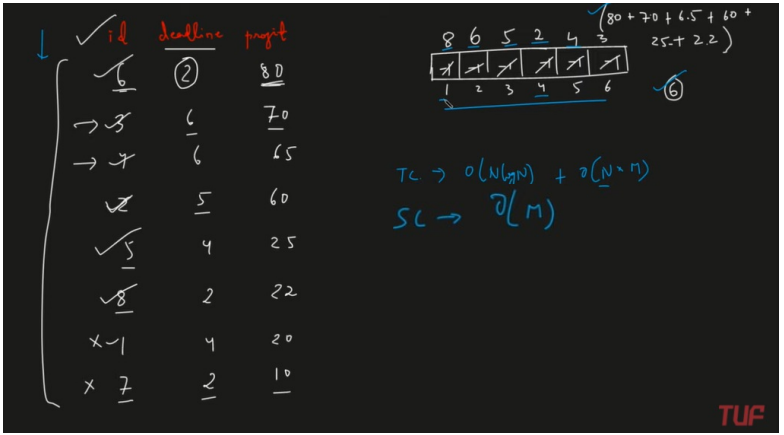
Medium Accuracy: 48.94% Submissions: 46082 Points: 4

Given a set of **N** jobs where each **job$_i$** has a deadline and profit associated with it. Each job takes ***1*** unit of time to complete and only one job can be scheduled at a time. We earn the profit if and only if the job is completed by its deadline. The task is to find the number of jobs done and the **maximum profit**.

Note: Jobs will be given in the form (Job$_{id}$, Deadline, Profit) associated with that Job.

Example 1:

Input:
N = 4
Jobs = {(1,4,20),(2,1,10),(3,1,40),(4,1,30)}
Output:
2 60
Explanation:
Job$_1$ and Job$_3$ can be done with maximum profit of 60 (20+40).




```java
class solve{
    // return an array of size 2 having the 0th element equal to the count
    // and 1st element equal to the maximum profit
    int[] JobScheduling(Job arr[], int n){
        Arrays.sort(arr, (a, b) -> (b.profit - a.profit));
        int maxi = 0;
        for(int i = 0;i<n;i++) {
            if(arr[i].deadline > maxi) {
                maxi = arr[i].deadline;
            }
        }

        int result[] = new int[maxi + 1];
        for(int i = 1;i<=maxi;i++) {
            result[i] = -1;
        }

        int countJobs = 0, jobProfit = 0;
        for (int i = 0; i < n; i++){
            for (int j = arr[i].deadline; j > 0; j--) {
                // Free slot found
                if (result[j] == -1){
                    result[j] = i;
                    countJobs++;
                    jobProfit += arr[i].profit;
                    break;
                }
            }
        }
        int ans[] = new int[2];
        ans[0] = countJobs;
        ans[1] = jobProfit;
        return ans;
    }
}
```

4. Fractional Knapsack Problem

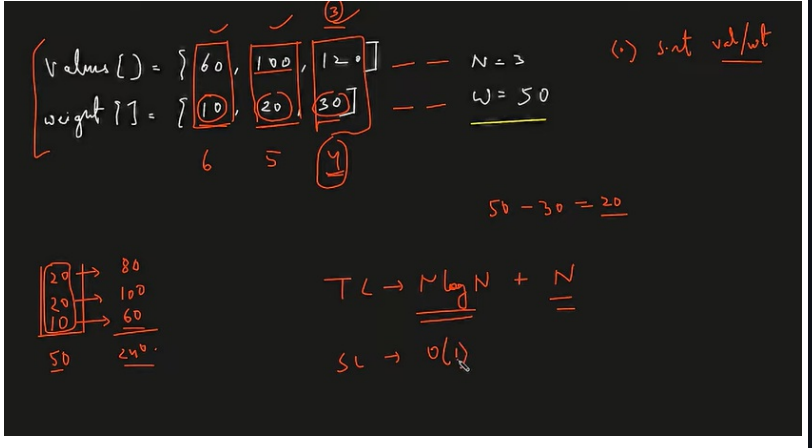
Fractional Knapsack

Medium Accuracy: 45.14% Submissions: 46130 Points: 4

Given *weights* and *values* of **N** items, we need to put these items in a knapsack of capacity **W** to get the *maximum* total value in the knapsack.
Note: Unlike 0/1 knapsack, you are allowed to break the item.

Example 1:

```
Input:
N = 3, W = 50
values[] = {60,100,120}
weight[] = {10,20,30}
Output:
240.00
Explanation:Total maximum value of item
we can have is 240.00 from the given
capacity of sack.
```



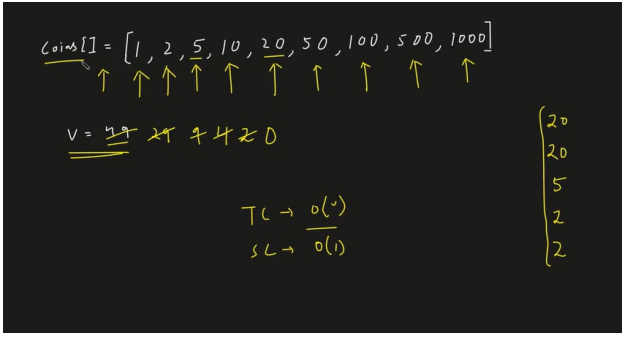

```java
class itemComparator implements Comparator<Item>
{
    @Override
    public int compare(Item a, Item b)
    {
        double r1 = (double)(a.value) / (double)(a.weight);
        double r2 = (double)(b.value) / (double)(b.weight);
        if(r1 < r2) return 1;
        else if(r1 > r2) return -1;
        else return 0;
    }
}
class solve{
    double fractionalKnapsack(int W, Item arr[], int n) {
        Arrays.sort(arr, new itemComparator());

        int curWeight = 0;
        double finalvalue = 0.0;

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

            if (curWeight + arr[i].weight <= W) {
                curWeight += arr[i].weight;
                finalvalue += arr[i].value;
            }

            else {
                int remain = W - curWeight;
                finalvalue += ((double)arr[i].value / (double)arr[i].weight) * (double)remain;
                break;
            }
        }

        return finalvalue;

    }
}
```

5. Greedy algorithm to find minimum number of coins

Given a value V, if we want to make a change for V cents, and we have an infinite supply of each of C = { C1, C2, .., Cm} valued coins, what is the minimum number of coins to make the change? If it's not possible to make a change, print -1.

Examples:

```
Input: coins[] = {25, 10, 5}, V = 30
Output: Minimum 2 coins required
We can use one coin of 25 cents and one of 5 cents

Input: coins[] = {9, 6, 5, 1}, V = 11
Output: Minimum 2 coins required
We can use one coin of 6 cents and 1 coin of 5 cents
```




```java
static void findMin(int V)
{

    Vector<Integer> ans = new Vector<>();

    int deno[] = {1, 2, 5, 10, 20, 50, 100, 500, 1000};
    int n = deno.length;

    for (int i = n - 1; i >= 0; i--)
    {
        while (V >= deno[i])
        {
            V -= deno[i];
            ans.add(deno[i]);
        }
    }

    for (int i = 0; i < ans.size(); i++)
    {
        System.out.print(
            " " + ans.elementAt(i));
    }
}
```

Day9 (Recursion):

1. Subset Sums

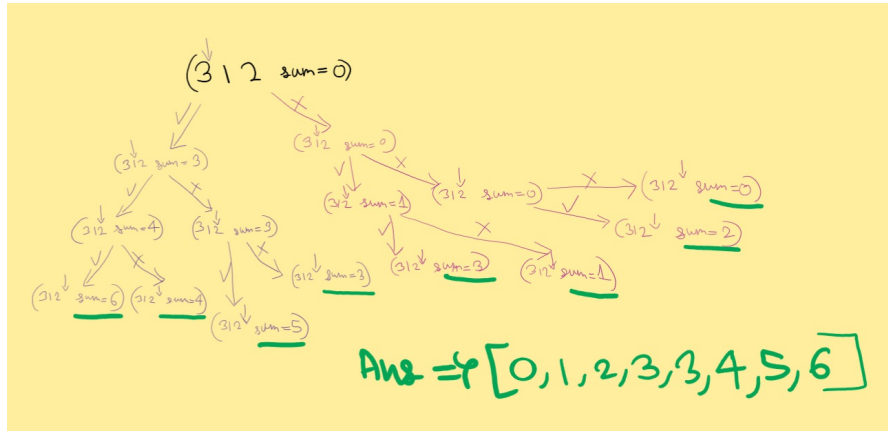
Subset Sums

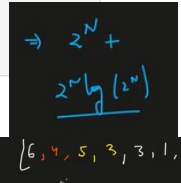
Basic Accuracy: 58.12% Submissions: 10324 Points: 1

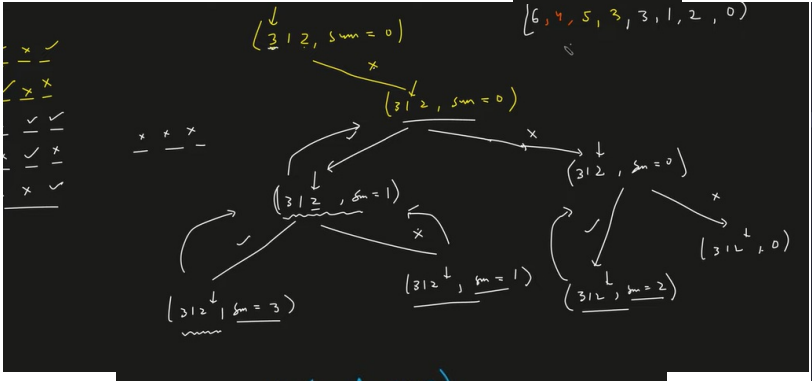
Given a list **arr** of **N** integers, print sums of all subsets in it.

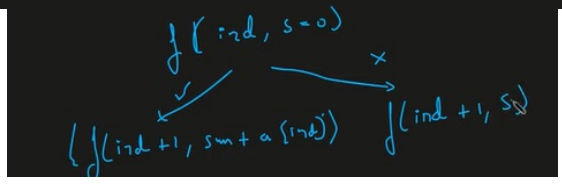
Example 1:

```
Input:
N = 2
arr[] = {2, 3}
Output:
0 2 3 5
Explanation:
When no elements is taken then Sum = 0.
When only 2 is taken then Sum = 2.
When only 3 is taken then Sum = 3.
When element 2 and 3 are taken then
Sum = 2+3 = 5.
```










```java
// } Driver Code Ends


//User function Template for Java//User function Template for Java
class Solution{
    void func(int ind, int sum, ArrayList<Integer> arr, int N, ArrayList<Integer> sumSubset) {
        if(ind == N) {
            sumSubset.add(sum);
            return;
        }

        // pick the element
        func(ind + 1, sum + arr.get(ind), arr, N, sumSubset);

        // Do-not pick the element
        func(ind + 1, sum, arr, N, sumSubset);
    }

    ArrayList<Integer> subsetSums(ArrayList<Integer> arr, int N){
        // code here
        ArrayList<Integer> sumSubset = new ArrayList<>();
        func(0, 0, arr, N, sumSubset);
        Collections.sort(sumSubset);
        return sumSubset;
    }
}
```

2. Subset-II

90. Subsets II

Medium 3648 127 Add to List Share

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

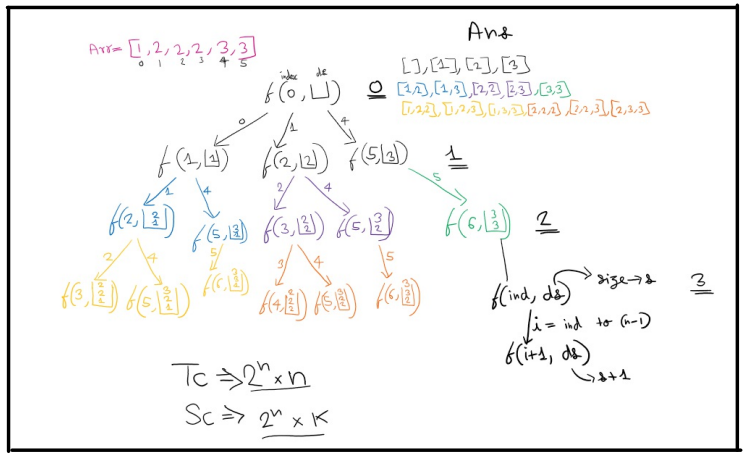
The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

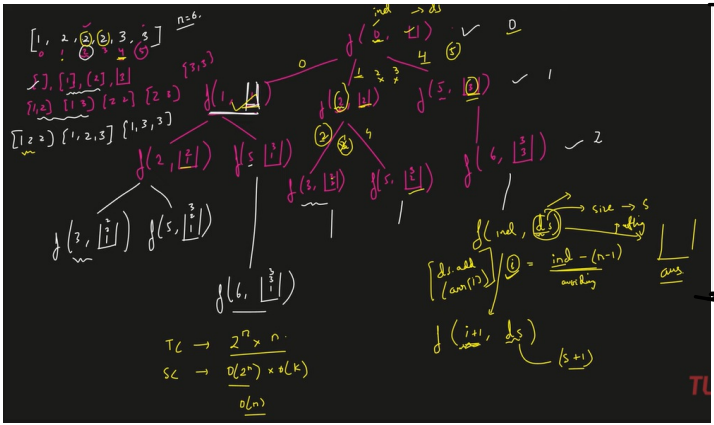
Example 1:

```
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
```

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```






```java
class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ansList = new ArrayList<>();
        findSubsets(0, nums, new ArrayList<>(), ansList);
        return ansList;
    }
    public void findSubsets(int ind, int[] nums, List<Integer> ds, List<List<Integer>> ansList){
        ansList.add(new ArrayList<>(ds));
        for(int i=ind; i<nums.length; i++){
            if(i!=ind && nums[i]==nums[i-1]) continue;
            ds.add(nums[i]);
            findSubsets(i+1, nums, ds, ansList);
            ds.remove(ds.size()-1);
        }
    }
}
```

Faster than 99%

```java
class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> res = new ArrayList<>();
        solve(nums, 0, res, new ArrayList<Integer>());
        return new ArrayList(res);
    }


    void solve(int[] nums, int start, List<List<Integer>> res, List<Integer> temp) {
        if(start <= nums.length) {
            List<Integer> copy = new ArrayList<>(temp);
            res.add(copy);
        }

        for(int i = start; i < nums.length; ) {
            temp.add(nums[i]);
            solve(nums, i+1, res, temp);
            temp.remove(temp.size() - 1);
            i++;
            while(i < nums.length && nums[i] == nums[i-1]) i++;
        }
    }
}
```

3. Combination sum-1

39. Combination Sum

Medium 7701 184 Add to List Share

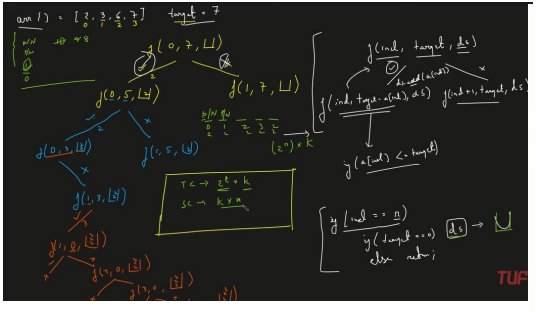
Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used
multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```



```java
class Solution {

    private void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds) {
        if(ind == arr.length) {
            if(target == 0) {
                ans.add(new ArrayList<>(ds));
            }
            return;
        }

        if(arr[ind] <= target) {
            ds.add(arr[ind]);
            findCombinations(ind, arr, target - arr[ind], ans, ds);
            ds.remove(ds.size() - 1);
        }
        findCombinations(ind + 1, arr, target, ans, ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> ans = new ArrayList<>();
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }
}
```

4. Combination sum-2

40. Combination Sum II

Medium 3756 104 Add to List Share

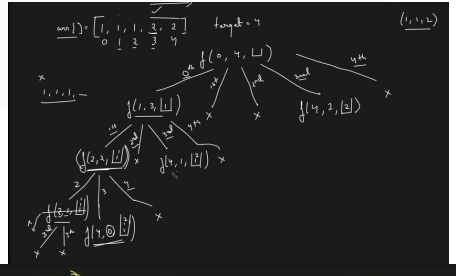
Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target` .

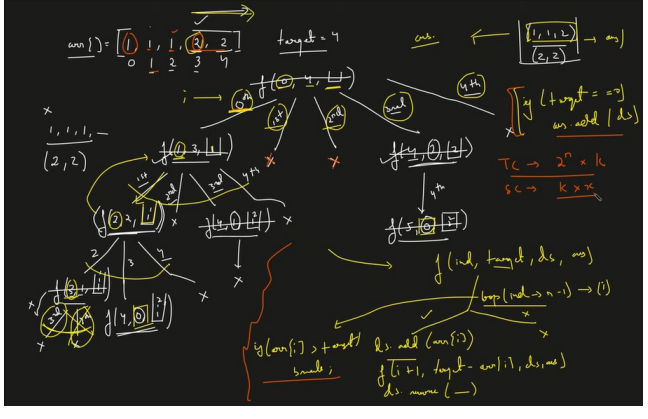
Each number in `candidates` may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```





```java
class Solution {
    private void findCombinations(int ind, int[] arr, int target, List<List<Integer>> ans, List<Integer> ds) {
        if(target == 0) {
            ans.add(new ArrayList<>(ds));
            return;
        }

        for(int i = ind; i < arr.length;i++) {
            if(i > ind && arr[i] == arr[i-1]) continue;
            if(arr[i]>target) break;

            ds.add(arr[i]);
            findCombinations(i+1, arr, target - arr[i], ans, ds);
            ds.remove(ds.size() - 1);
        }
    }
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> ans = new ArrayList<>();
        Arrays.sort(candidates);
        findCombinations(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }
}
```

5. Palindrome Partitioning

131. Palindrome Partitioning

Medium 4630 141 Add to List Share

Given a string `s`, partition `s` such that every substring of the partition is a **palindrome**. Return all possible palindrome partitioning of `s`.

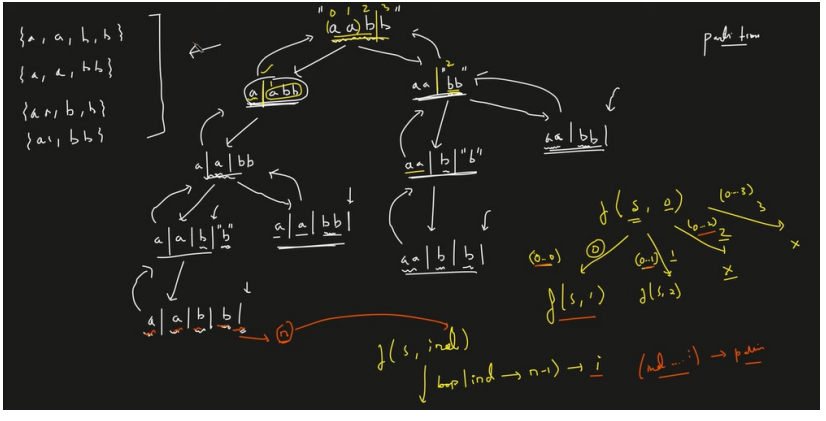
A **palindrome** string is a string that reads the same backward as forward.

Example 1:

```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```

Example 2:

```
Input: s = "a"
Output: [["a"]]
```



```java
class Solution {
    public List<List<String>> partition(String s) {
        List<List<String>> res= new ArrayList<>();
        List<String> path = new ArrayList<>();
        func(0, s, path, res);
        return res;
    }

    void func(int index, String s, List<String> path, List<List<String>> res) {
        if(index == s.length()) {
            res.add(new ArrayList<>(path));
            return;
        }
        for(int i = index; i < s.length(); ++i) {
            if(isPalindrome(s, index, i)) {
                path.add(s.substring(index, i+1));
                func(i+1, s, path, res);
                path.remove(path.size()-1);
            }
        }
    }

    boolean isPalindrome(String s, int start, int end) {
        while(start <= end) {
            if(s.charAt(start++) != s.charAt(end--))
                return false;
        }
        return true;
    }
}
```

6. K-th permutation Sequence

60. Permutation Sequence

Hard 2895 380 Add to List Share

The set `[1, 2, 3, ..., n]` contains a total of `n!` unique permutations.

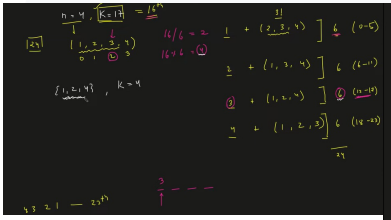
By listing and labeling all of the permutations in order, we get the following sequence for `n = 3`:

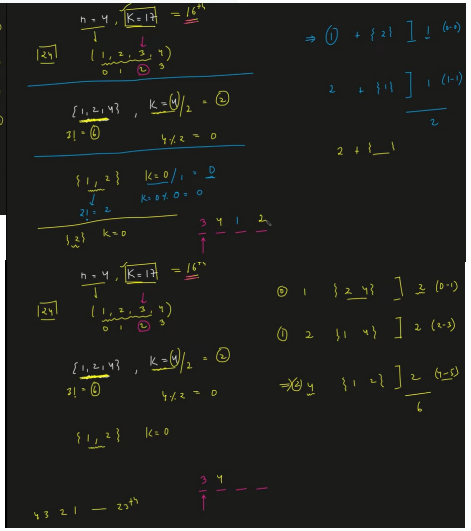
1. `"123"`
2. `"132"`
3. `"213"`
4. `"231"`
5. `"312"`
6. `"321"`

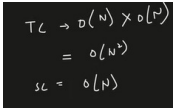
Given `n` and `k`, return the `k^th` permutation sequence.

Example 1:

```
Input: n = 3, k = 3
Output: "213"
```








```java
class Solution {
    public String getPermutation(int n, int k) {
        int fact = 1;
        List<Integer> numbers = new ArrayList<>();
        for(int i = 1;i<n;i++) {
            fact = fact * i;
            numbers.add(i);
        }
        numbers.add(n);
        String ans = "";
        k = k - 1;
        while(true) {
            ans = ans + numbers.get(k / fact);
            numbers.remove(k / fact);
            if(numbers.size() == 0) {
                break;
            }

            k = k % fact;
            fact = fact / numbers.size();
        }
        return ans;
    }
}
```

Day10: (Recursion and Backtracking)

1. Print all Permutations of a string/array

46. Permutations

Medium 7829 155 Add to List Share

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

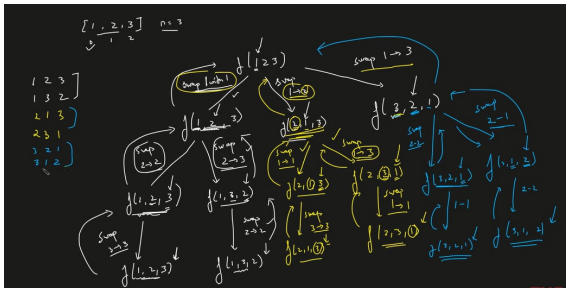
```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

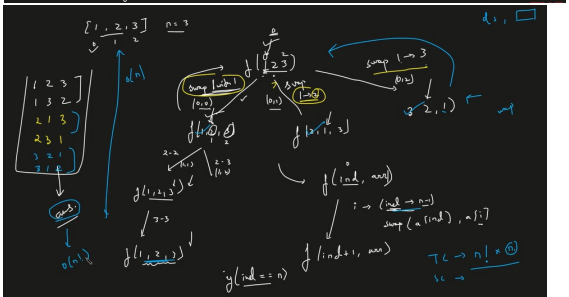
Example 2:

```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Example 3:

```
Input: nums = [1]
Output: [[1]]
```





```java
class Solution {
    private void recurPermute(int index, int[] nums, List<List<Integer>> ans) {
        if(index == nums.length) {
            // copy the ds to ans
            List<Integer> ds = new ArrayList<>();
            for(int i = 0;i<nums.length;i++) {
                ds.add(nums[i]);
            }
            ans.add(new ArrayList<>(ds));
            return;
        }
        for(int i = index;i<nums.length;i++) {
            swap(i, index, nums);
            recurPermute(index+1, nums, ans);
            swap(i, index, nums);
        }
    }
    private void swap(int i, int j, int[] nums) {
        int t = nums[i];
        nums[i] = nums[j];
        nums[j] = t;
    }
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        recurPermute(0, nums, ans);
        return ans;
    }
}
```

2. N queens Problem

51. N-Queens

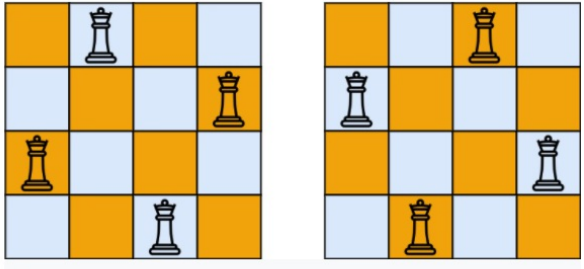
Hard 4301 129 Add to List Share

The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

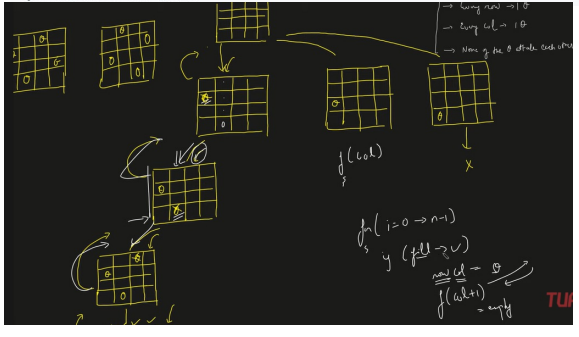
Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:



Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],
["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above



```java
class Solution {
    public List<List<String>> solveNQueens(int n) {
        char[][] board = new char[n][n];
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                board[i][j] = '.';
            }
        }
        List<List<String>> res = new ArrayList<List<String>>();
        int leftRow[] = new int[n];
        int upperDiag[] = new int[2*n-1];
        int lowerDiag[] = new int[2*n-1];
        solve(0, board, res, leftRow, lowerDiag, upperDiag);
        return res;
    }
    private void solve(int col, char[][] board, List<List<String>> res, int leftRow[], int lowerDiag[], int upperDiag[]){
        if(col == board.length){
            res.add(construct(board));
            return;
        }

        for(int row=0; row<board.length; row++){
            if(leftRow[row]==0 && lowerDiag[row + col]==0 && upperDiag[col-row + board.length-1]==0){
                board[row][col] = 'Q';
                leftRow[row] = 1;
                lowerDiag[row + col] = 1;
                upperDiag[col-row+board.length-1] = 1;
                solve(col+1, board, res, leftRow, lowerDiag, upperDiag );
                board[row][col] = '.';
                leftRow[row] = 0;
                lowerDiag[row + col] = 0;
                upperDiag[col-row+board.length-1] = 0;
            }
        }
    }

    private List<String> construct(char[][] board){
        List<String> res = new ArrayList<>();
        for(int i=0; i<board.length; i++){
            String s = new String(board[i]);
            res.add(s);
        }
        return res;
    }
}
```

3. Sudoku Solver

37. Sudoku Solver

Hard 3875 128 Add to List Share

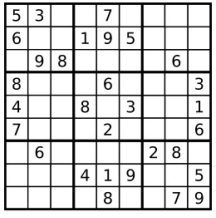
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

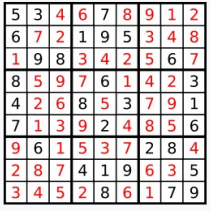
1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

The `'.'` character indicates empty cells.

Example 1:



```
Input: board = [["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],
[".","9","8",".",".",".",".","6","."],
["8",".",".",".","6",".",".",".","3"],
["4",".",".","8",".","3",".",".","1"],
["7",".",".",".","2",".",".",".","6"],
[".","6",".",".",".",".","2","8","."],
[".",".",".","4","1","9",".",".","5"],
[".",".",".",".","8",".",".","7","9"]]
Output: [["5","3","4","6","7","8","9","1","2"],
["6","7","2","1","9","5","3","4","8"],
["1","9","8","3","4","2","5","6","7"],
["8","5","9","7","6","1","4","2","3"],
["4","2","6","8","5","3","7","9","1"],
["7","1","3","9","2","4","8","5","6"],
["9","6","1","5","3","7","2","8","4"],
["2","8","7","4","1","9","6","3","5"],
["3","4","5","2","8","6","1","7","9"]]
Explanation: The input board is shown above and the only valid
solution is shown below:
```



```
class Solution {
    public void solveSudoku(char[][] board) {
        if(board==null || board.length==0){
            return;
        }
        solve(board);
    }

    public boolean solve(char[][] board){
        for(int i=0; i<board.length; i++){
            for(int j=0; j<board[0].length; j++){
                if(board[i][j] == '.'){
                    for(char c='1'; c<='9'; c++){
                        if(isValid(board ,i ,j ,c)){
                            board[i][j] = c;
                            if(solve(board)){
                                return true;
                            }else{
                                board[i][j] = '.';
                            }
                        }
                    }
                    return false;
                }
            }
        }
        return true;
    }

    private boolean isValid(char[][] board, int row, int col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] != '.' && board[i][col] == c) return false; //check row
            if(board[row][i] != '.' && board[row][i] == c) return false; //check column
            if(board[3 * (row / 3) + i / 3][ 3 * (col / 3) + i % 3] != '.' && board[3 * (row / 3) + i / 3][3 * (col / 3) + i % 3] == c){
                return false;  // check 3*3 block
            }
        }
        return true;
    }
}
```

4. M coloring Problem

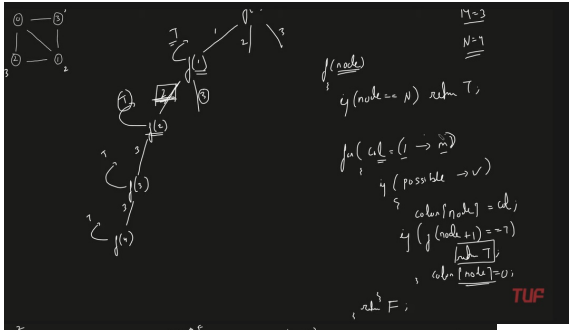
M-Coloring Problem

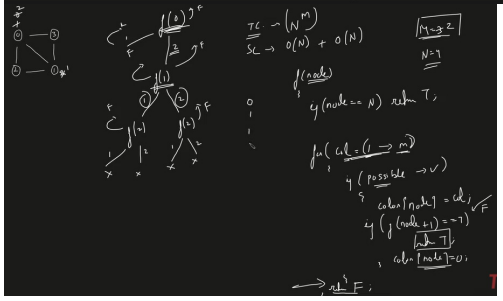
Medium Accuracy: 47.46% Submissions: 21089 Points: 4

Given an undirected graph and an integer **M**. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:

```
Input:
N = 4
M = 3
E = 5
Edges[] = {(0,1),(1,2),(2,3),(3,0),(0,2)}
Output: 1
Explanation: It is possible to colour the
given graph using 3 colours.
```





```java
class solve
{
    private static boolean isSafe(int node, List<Integer>[] G, int[] color, int n, int col) {
        for(int it: G[node]) {
            if(color[it] == col) return false;
        }
        return true;
    }
    private static boolean solve(int node, List<Integer>[] G, int[] 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;
    }
    public static boolean graphColoring(List<Integer>[] G, int[] color, int i, int m)
    {
        int n = G.length;
        if(solve(i, G, color, n, m) == true) return true;
        return false;
        // Your code here
    }
}
```

5. Rat in a Maze

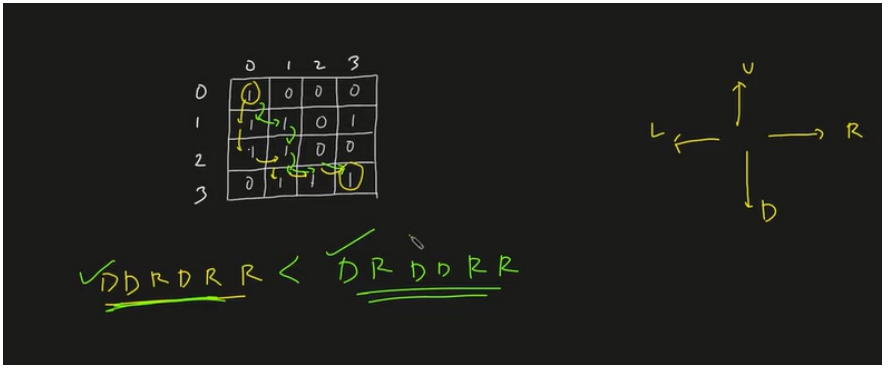
Rat in a Maze Problem - I

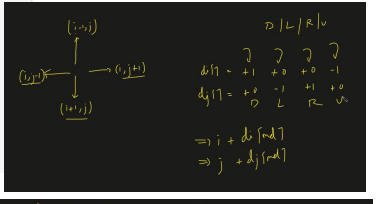
Medium Accuracy: 37.73% Submissions: 86956 Points: 4

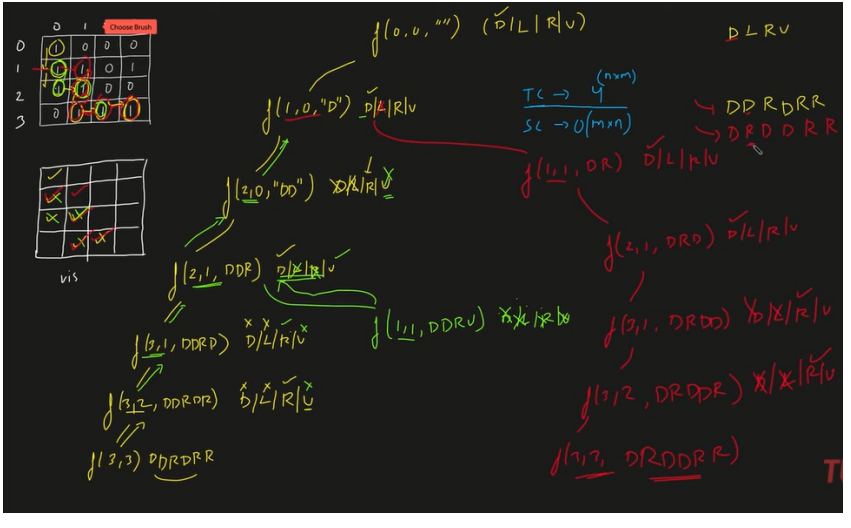
Consider a rat placed at **(0, 0)** in a square matrix of order **N * N**. It has to reach the destination at **(N - 1, N - 1)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are **'U'(up)**, **'D'(down)**, **'L' (left)**, **'R' (right)**. Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.
Note: In a path, no cell can be visited more than one time.

Example 1:

```
Input:
N = 4
m[][] = {{1, 0, 0, 0},
         {1, 1, 0, 1},
         {1, 1, 0, 0},
         {0, 1, 1, 1}}
Output:
DDRDRR DRDDRR
Explanation:
The rat can reach the destination at
(3, 3) from (0, 0) by two paths - DRDDRR
and DDRDRR, when printed in sorted order
we get DDRDRR DRDDRR.
```








```java
class Solution {
    private static void solve(int i, int j, int a[][], int n, ArrayList<String> ans, String move,
    int vis[][]) {
        if(i==n-1 && j==n-1) {
            ans.add(move);
            return;
        }
        // String dir = "DLRU";
        // for(int ind = 0; ind<4;ind++) {
        //     int nexti = i + di[ind];
        //     int nextj = j + dj[ind];
        //     if(nexti >= 0 && nextj >= 0 && nexti < n && nextj < n
        //      && vis[nexti][nextj] == 0 && a[nexti][nextj] == 1) {

        //         vis[i][j] = 1;
        //         solve(nexti, nextj, a, n, ans, move + dir.charAt(ind), vis, di, dj);
        //         vis[i][j] = 0;

        //     }
        // }
        // downward
        if(i+1<n && vis[i+1][j] == 0 && a[i+1][j] == 1) {
            vis[i][j] = 1;
            solve(i+1, j, a, n, ans, move + 'D', vis);
            vis[i][j] = 0;
        }

        // left
        if(j-1>=0 && vis[i][j-1] == 0 && a[i][j-1] == 1) {
            vis[i][j] = 1;
            solve(i, j-1, a, n, ans, move + 'L', vis);
            vis[i][j] = 0;
        }

        // right
        if(j+1<n && vis[i][j+1] == 0 && a[i][j+1] == 1) {
            vis[i][j] = 1;
            solve(i, j+1, a, n, ans, move + 'R', vis);
            vis[i][j] = 0;
        }

        // upward
        if(i-1>=0 && vis[i-1][j] ==0 && a[i-1][j] == 1) {
            vis[i][j] = 1;
            solve(i-1, j, a, n, ans, move + 'U', vis);
            vis[i][j] = 0;
        }
    }
    public static ArrayList<String> findPath(int[][] m, int n) {
        int vis[][] = new int[n][n];
        for(int i = 0;i<n;i++) {
            for(int j = 0;j<n;j++) {
                vis[i][j] = 0;
            }
        }

        ArrayList<String> ans = new ArrayList<>();
        if(m[0][0] == 1) solve(0, 0, m, n, ans, "", vis);
        return ans;
    }
}
```

6. Word Break (print all ways)

Word Break

Medium

Prev Next

1. You are given n space-separated strings, which represents a dictionary of words.
2. You are given another string that represents a sentence.
3. You have to determine if this sentence can be segmented into a space-separated sequence of one or more dictionary words.

Input Format

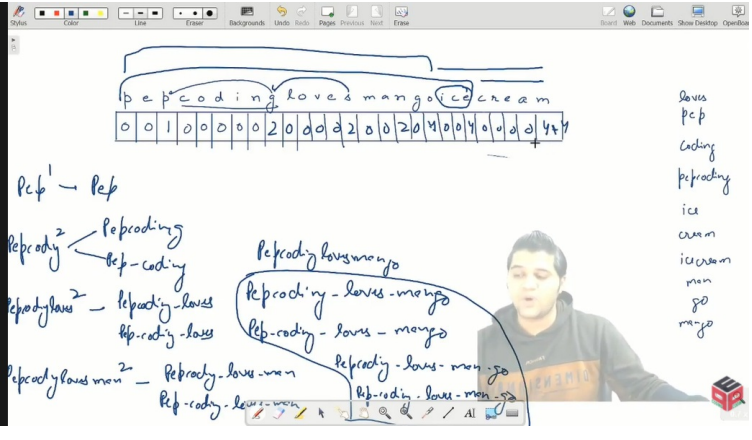
A number n
n strings representing words
A string representing a sentence

Sample Input

2
pep coding
pepcoding

Sample Output

true




```java
import java.io.*;
import java.util.*;

public class Main {

    public static boolean solution(String sentence, HashSet<String> dictionary) {
        //write your code here
        int[]  dp = new int[sentence.length()];
        for(int i=0; i<dp.length; i++){
            for(int j=0; j<=i; j++){
                String w2check = sentence.substring(j,i+1);
                if(dictionary.contains(w2check)){
                    if(j>0){
                        dp[i] += dp[j-1];
                    }else{
                        dp[i] += 1;
                    }
                }
            }
        }
        return dp[sentence.length()-1]>0;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        HashSet<String> dictionary = new HashSet<String>();
        for (int i = 0; i < n; i++) {
            dictionary.add(scn.next());
        }
        String sentence = scn.next();
        System.out.println(solution(sentence, dictionary));
    }

}
```

Day11: (Binary Search)

1. N-th root of an integer (use binary search) (square root, cube root, ..)

N, M

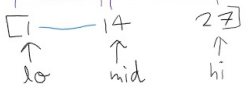
N=2, M=4 $\sqrt[2]{4} = 2$

N=4, M=15 $\sqrt[4]{15} = 1.96798$

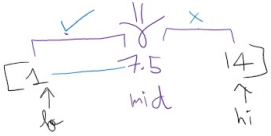
$T_c = N \times \log_2(M \times 10^d)$

d → decimal places

Q. N=3, M=27

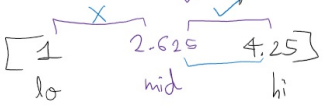


mid=14 so, 14×14×14



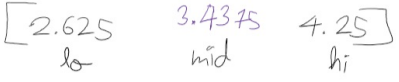
so, 7.5×7.5×7.5 is greater 27

After 7.5 left → 4.25



so, 2.625×2.625×2.625 is smaller than 27

so, now we take right half



So q. How long we are going to do this?

→ till High − low > $1e^{-6}$

```java
import java.io.*;
class GFG {
    private static double multiply(double number, int n) {
        double ans = 1.0;
        for(int i = 1;i<=n;i++) {
            ans = ans * number;
        }
        return ans;
    }
    private static void getNthRoot(int n, int m) {
        double low = 1;
        double high = m;
        double eps = 1e-6;

        while((high - low) > eps) {
            double mid = (low + high) / 2.0;
            if(multiply(mid, n) < m) {
                low = mid;
            }
            else {
                high = mid;
            }
        }

        System.out.println(low + " " + high);

        // just to check
         System.out.println(Math.pow(m, (double)(1.0/(double)n)));
    }
        public static void main (String[] args) {
                int n = 17, m = 89;
                getNthRoot(n, m);
        }
}
```

2. Matrix Median

Matrix Median

Medium | 172 | 9 | Add to favorites

Asked In: AMAZON

Given a matrix of integers **A** of size **N x M** in which each row is sorted.

Find an return the overall median of the matrix **A**.

Note: No extra memory is allowed.

Note: Rows are numbered from top to bottom and columns are numbered from left to right.

Input Format

The first and only argument given is the integer matrix A.

Output Format

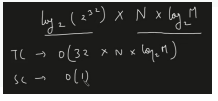
Return the overall median of the matrix A.

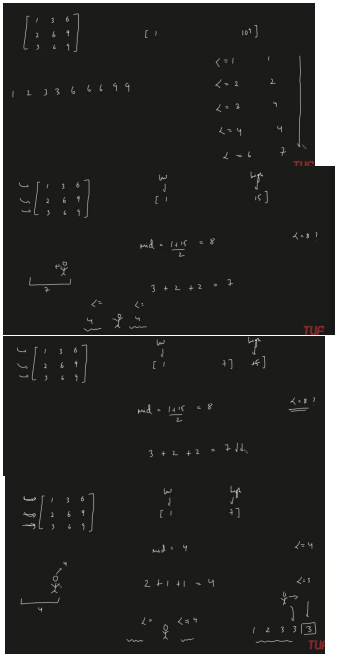
For Example

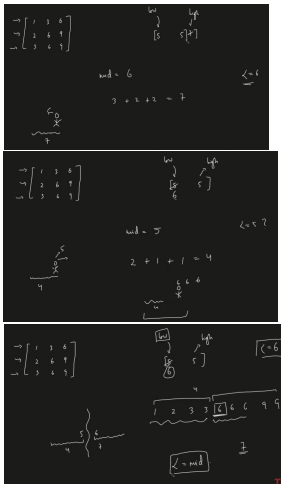
```
Input 1:
    A = [   [1, 3, 5],
            [2, 6, 9],
            [3, 6, 9]   ]
Output 1:
    5
Explanation 1:
    A = [1, 2, 3, 3, 5, 6, 6, 9, 9]
    Median is 5. So, we return 5.
```

Constraints

```
1 <= N, M <= 10^5
1 <= N*M  <= 10^6
1 <= A[i] <= 10^9
N*M is odd
```







```java
public class Solution {
    private int countSmallerThanMid(ArrayList<Integer> row, int mid) {
        int l = 0, h = row.size() - 1;
        while(l <= h) {
            int md = (l + h) >> 1;
            if(row.get(md) <= mid) {
                l = md + 1;
            }
            else {
                h = md - 1;
            }
        }
        return l;
    }
    public int findMedian(ArrayList<ArrayList<Integer>> A) {
        int low = Integer.MIN_VALUE;
        int high = Integer.MAX_VALUE;
        int n = A.size();
        int m = A.get(0).size();
        while(low <= high) {
            int mid = (low + high) >> 1;
            int cnt = 0;
            for(int i = 0;i<n;i++) {
                cnt += countSmallerThanMid(A.get(i), mid);
            }

            if(cnt <= (n * m) / 2) low = mid + 1;
            else high = mid - 1;
        }
        return low;
    }
}
```

3. Find the element that appears once in sorted array, and rest element appears twice (Binary search)

540. Single Element in a Sorted Array

Medium 3397 90 Add to List Share

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once.*

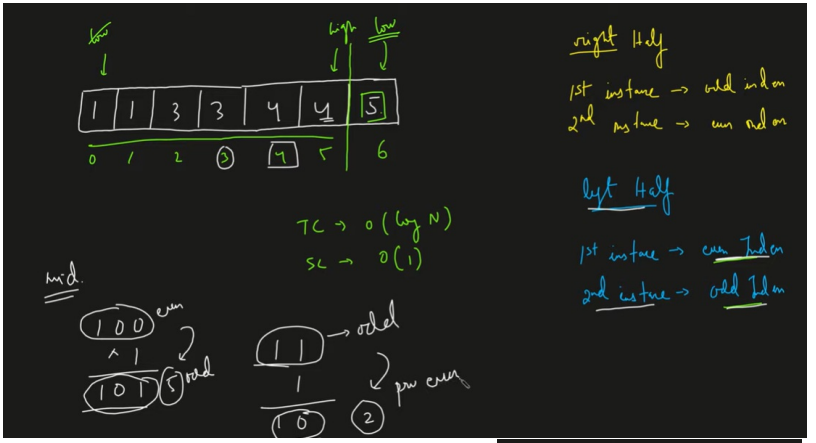
Your solution must run in `O(log n)` time and `O(1)` space.

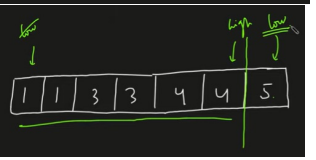
Example 1:

```
Input: nums = [1,1,2,3,3,4,4,8,8]
Output: 2
```

Example 2:

```
Input: nums = [3,3,7,7,10,11,11]
Output: 10
```






```java
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int lo = 0;
        int hi = nums.length-2;
        while(lo <= hi){
            int mid = (lo + hi) >> 1;
            if(nums[mid] == nums[mid^1]){
                lo = mid + 1;
            }else{
                hi = mid - 1;
            }
        }
        return nums[lo];
    }
}

// in order to check for the left half
// 1st instance => even index{0,2,4,...}
// 2nd instance => odd index{1,3,5,....}
```

4. Search element in a sorted and rotated array/ find pivot where it is rotated

33. Search in Rotated Sorted Array

Medium 10495 770 Add to List Share

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k (1 <= k < nums.length)` such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

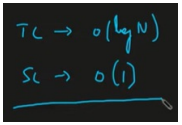
You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```




```java
class Solution {
    public int search(int[] a, int target) {
        int low = 0, high = a.length - 1;
        while(low <= high) {
            int mid = (low + high) >> 1;
            if(a[mid] == target) return mid;

            // the left side is sorted
            if(a[low] <= a[mid]) {
//                figure out if the element lies on the left half or not
                if(target >= a[low] && target <= a[mid]) {
                    high = mid - 1;
                }
                else {
                    low = mid + 1;
                }
            }
//              right half is sorted
            else {
                if(target >= a[mid] && target <= a[high]) {
                    low = mid + 1;
                }
                else {
                    high = mid - 1;
                }
            }
        }
        return -1;
    }
}
```

5. Median of 2 sorted arrays

4. Median of Two Sorted Arrays

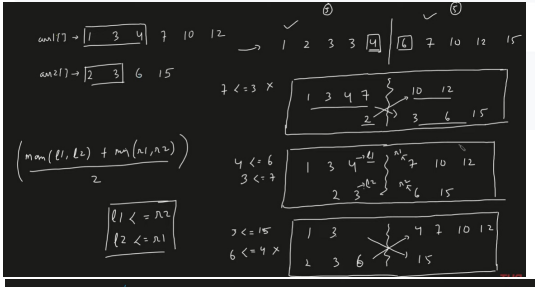
Hard 12996 1732 Add to List Share

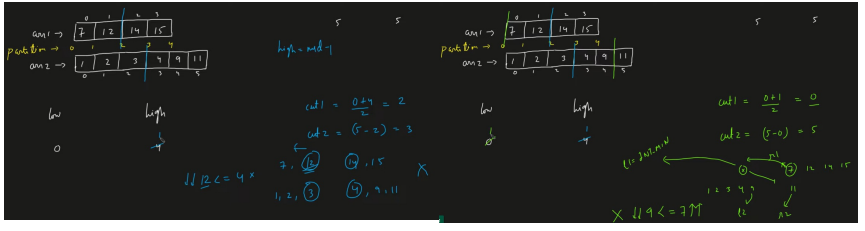
Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return **the median** of the two sorted arrays.

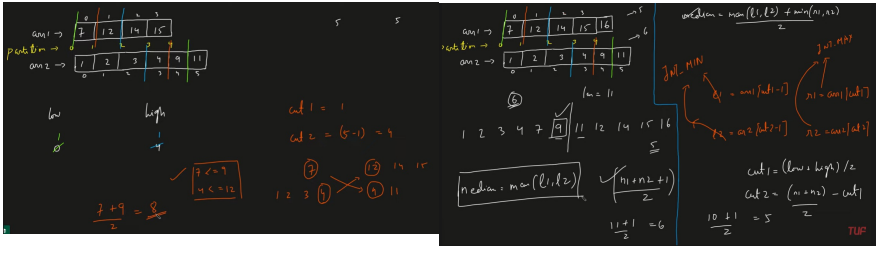
The overall run time complexity should be `O(log (m+n))`.

Example 1:

```
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.
```







```java
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if(nums2.length < nums1.length) return findMedianSortedArrays(nums2, nums1);
        int n1 = nums1.length;
        int n2 = nums2.length;
        int low = 0, high = n1;

        while(low <= high) {
            int cut1 = (low+high) >> 1;
            int cut2 = (n1 + n2 + 1) / 2 - cut1;


            int left1 = cut1 == 0 ? Integer.MIN_VALUE : nums1[cut1-1];
            int left2 = cut2 == 0 ? Integer.MIN_VALUE : nums2[cut2-1];

            int right1 = cut1 == n1 ? Integer.MAX_VALUE : nums1[cut1];
            int right2 = cut2 == n2 ? Integer.MAX_VALUE : nums2[cut2];


            if(left1 <= right2 && left2 <= right1) {
                if( (n1 + n2) % 2 == 0 )
                    return (Math.max(left1, left2) + Math.min(right1, right2)) / 2.0;
                else
                    return Math.max(left1, left2);
            }
            else if(left1 > right2) {
                high = cut1 - 1;
            }
            else {
                low = cut1 + 1;
            }
        }
        return 0.0;
    }
}
```

6. K-th element of two sorted arrays

K-th element of two sorted Arrays

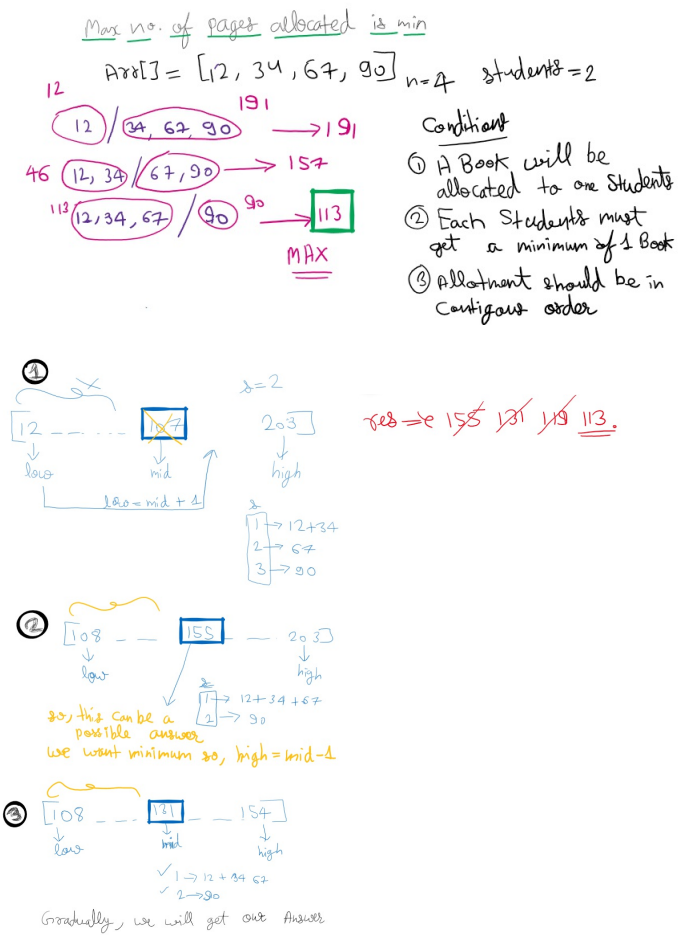
Medium Accuracy: 50.09% Submissions: 34356 Points: 4

Given two sorted arrays **arr1** and **arr2** of size **N** and **M** respectively and an element **K**. The task is to find the element that would be at the k'th position of the final sorted array.

Example 1:

```
Input:
arr1[] = {2, 3, 6, 7, 9}
arr2[] = {1, 4, 8, 10}
k = 5
Output:
6
Explanation:
The final sorted array would be -
1, 2, 3, 4, 6, 7, 8, 9, 10
The 5th element of this array is 6.
```

7. Allocate Minimum Number of Pages



we can code differently by not using arraylist and then the TC will be nlogn and SC will be 1

```java
public class Solution {
    private boolean isPossible(ArrayList<Integer> A, int pages, int students) {
        int cnt = 0;
        int sumAllocated = 0;
        for(int i = 0;i<A.size();i++) {
            if(sumAllocated + A.get(i) > pages) {
                cnt++;
                sumAllocated = A.get(i);
                if(sumAllocated > pages) return false;
            }
            else {
                sumAllocated += A.get(i);
            }
        }
        if(cnt < students) return true;
        return false;
    }
    public int books(ArrayList<Integer> A, int B) {
        if(B > A.size()) return -1;
        int low = A.get(0);
        int high = 0;
        for(int i = 0;i<A.size();i++) {
            high = high + A.get(i);
            low = Math.min(low, A.get(i));
        }
        int res = -1;
        while(low <= high) {
            int mid = (low + high) >> 1;
            //cout << low << " " << high << " " << mid << endl;
            if(isPossible(A, mid, B)) {
                res = mid;
                high = mid - 1;
            }
            else {
                low = mid + 1;
            }
        }
        // return res -> this is also correct
        return low;
    }
}
```

AGGRCOW - Aggressive cows

#binary-search

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).

His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Input

t – the number of test cases, then *t* test cases follows.
* Line 1: Two space-separated integers: N and C
* Lines 2..N+1: Line i+1 contains an integer stall location, xi

Output

For each test case output one integer: the largest minimum distance.

Example

Input:

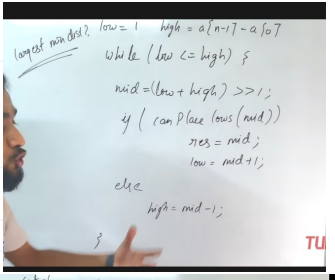
```
1
5 3
1
2
8
4
9
```

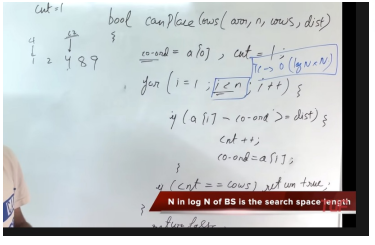
Output:

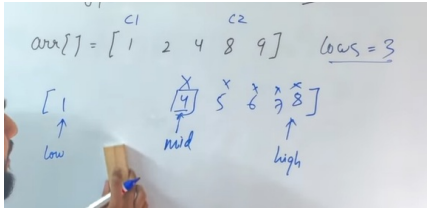
```
3
```

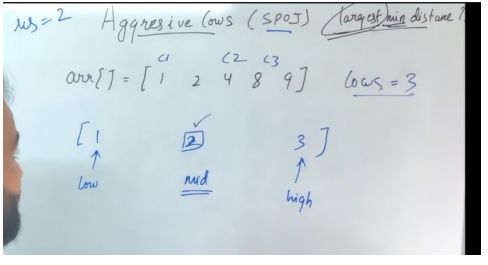
Output details:

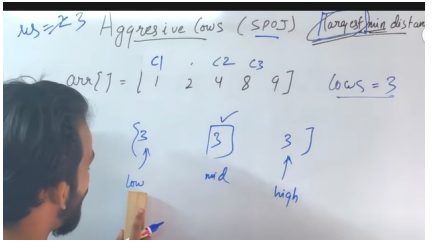
FJ can put his 3 cows in the stalls at positions 1, 4 and 8, resulting in a minimum distance of 3.

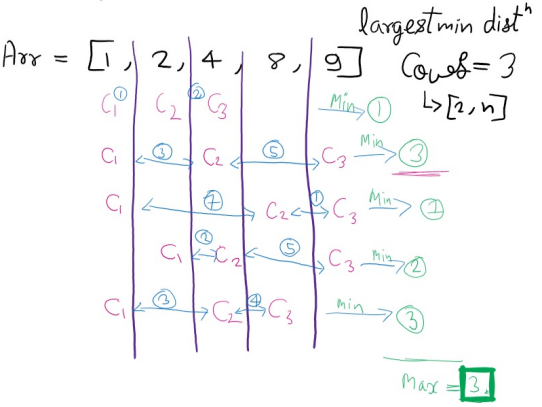


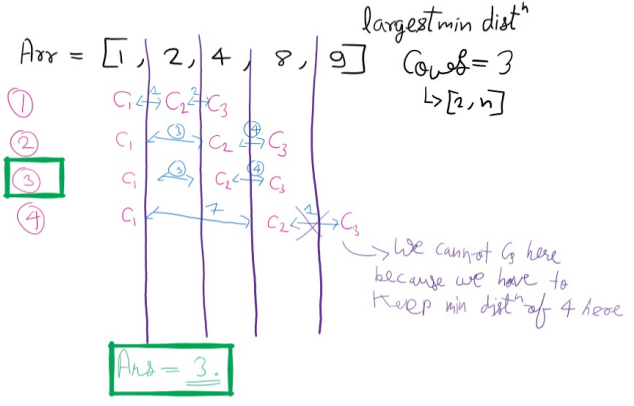












Day12: (Bits) (Optional, very rare topic in interviews, but if you have time left, someone might ask)

1 **Check if a number if a power of 2 or not in O(1)**

231. Power of Two

Easy 1917 245 Add to List Share

Given an integer `n`, return *`true`* *if it is a power of two. Otherwise, return* *`false`*.

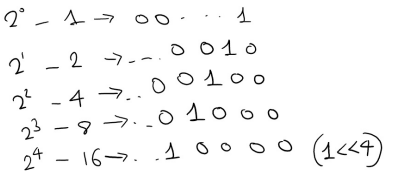
An integer `n` is a power of two, if there exists an integer `x` such that `n == 2^x`.

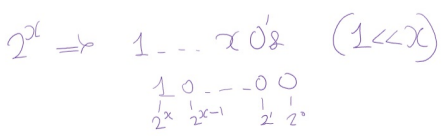
Example 1:

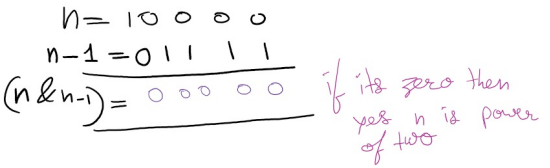
```
Input: n = 1
Output: true
Explanation: 2^0 = 1
```

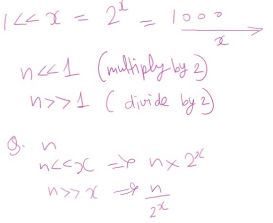
Example 2:

```
Input: n = 16
Output: true
Explanation: 2^4 = 16
```










```java
class Solution {
    public boolean isPowerOfTwo(int n) {
        //1st approach
        return n<=0?false:((n&(n-1))==0);

        //2nd approach
        if(n<0) return false;
        int count=0;
        while(n!=0){
            n=n&(n-1);
            count++;
        }
        if(count==1) return true;
        else return false;
    }
}
```

Count total set bits

338. Counting Bits

Easy 4989 246 Add to List Share

Given an integer `n`, return *an array* `ans` *of length* `n + 1` *such that for each* `i` (`0 <= i <= n`), `ans[i]` *is the* ***number of*** `1`***'s*** *in the binary representation of* `i`.

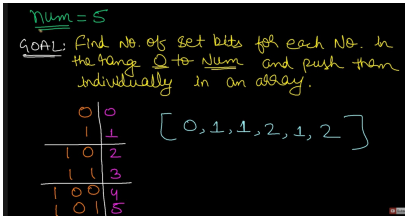
Example 1:

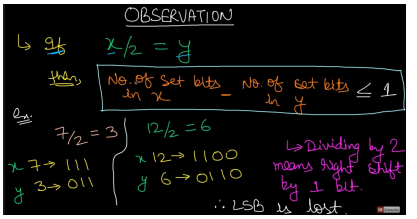
```
Input: n = 2
Output: [0,1,1]
Explanation:
0 --> 0
1 --> 1
2 --> 10
```

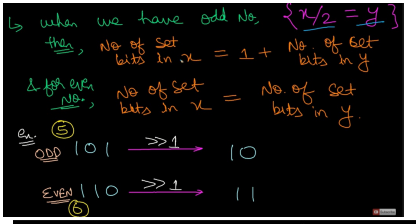
Example 2:

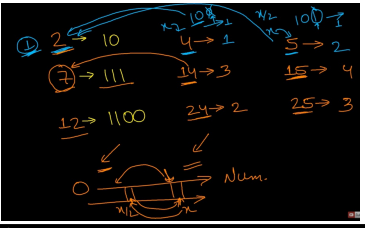
```
Input: n = 5
Output: [0,1,1,2,1,2]
Explanation:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
```

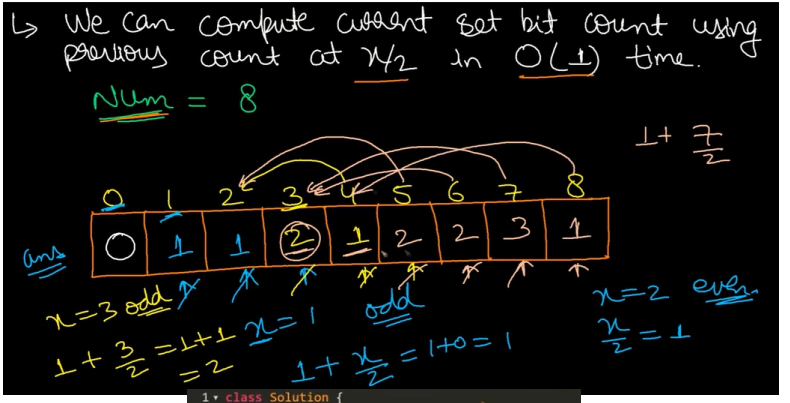
```
class Solution {
    public int[] countBits(int n) {
        int[] ans = new int[n+1];
        for(int i=1;i<= n ;i++){
            ans[i] = ans[i&(i-1)] +1;
        }
        return ans;
    }
}
```












```
class Solution {
public:
    vector<int> countBits(int num) {
        //mem[i] = No of 1s from 0 to number i
        vector<int> mem(num+1);
        mem[0] = 0;

        for(int i=1;i<=num;++i)
            mem[i] = mem[i/2] + i%2;

        return mem;
    }
};
```

Count Set Bits In First N Natural Numbers

Easy

1. You are given a number n.
2. You have to print the count of set bits of first n natural numbers.

Input Format

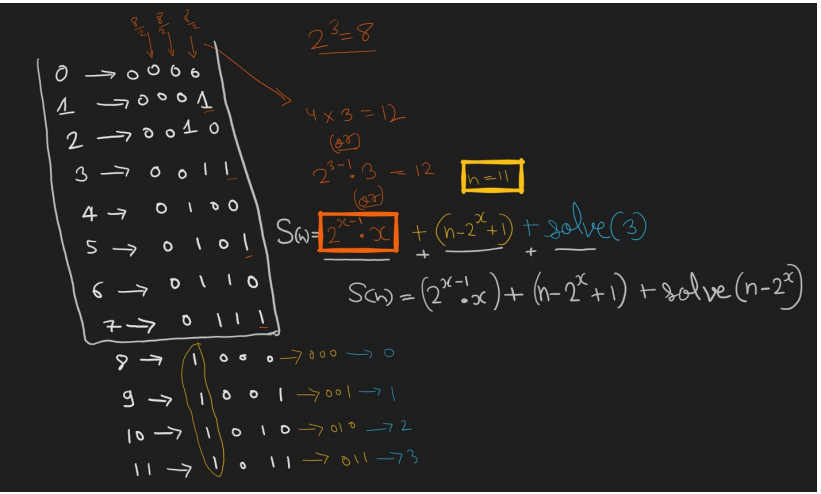
A number n

Sample Input

17

Sample Output

35




```java
import java.io.*;
import java.util.*;

public class Main {

  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    System.out.println(solution(n));
  }

  public static int solution(int n){
    //write your code here
    if(n==0){
        return 0;
    }
    int x = largestPowerOf2inRange(n);
    int bitstill2x = x *(1<<(x-1));
    int msb2xton = n - (1<<x) + 1;
    int rest = n - (1 << x);
    int ans = bitstill2x + msb2xton + solution(rest);
    return ans;
  }
  public static int largestPowerOf2inRange(int n){
      int x=0;
      while((1<<x) <= n){
          x++;
      }
      return x-1;
  }

}
```

Divide Integers without / operator

29. Divide Two Integers

Medium 2248 8133 Add to List Share

Given two integers `dividend` and `divisor`, divide two integers without using multiplication, division, and mod operator.

Return the quotient after dividing `dividend` by `divisor`.

The integer division should truncate toward zero, which means losing its fractional part. For example, `truncate(8.345) = 8` and `truncate(-2.7335) = -2`.

Note: Assume we are dealing with an environment that could only store integers within the **32-bit** signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, assume that your function **returns** $2^{31} - 1$ **when the division result overflows**.

Example 1:

```
Input: dividend = 10, divisor = 3
Output: 3
Explanation: 10/3 = truncate(3.33333..) = 3.
```

Example 2:

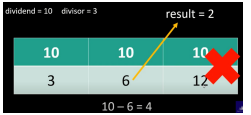
```
Input: dividend = 7, divisor = -3
Output: -2
Explanation: 7/-3 = truncate(-2.33333..) = -2.
```

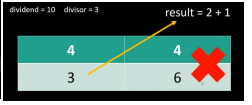
Steps that we are following:

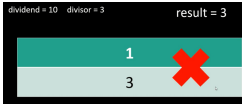
1. while we can subtract divisor from dividend.
   - if yes, then we double the divisor.
   - increment the count
   - check again
2. Add the count into result;
3. Subtract the temporary variable from dividend

Time O(logN^2)
Space O(1)

```
class Solution {
    public int divide(int dividend, int divisor) {
        if(dividend == 1<<31 && divisor == -1) return Integer.MAX_VALUE;

        boolean sign = (dividend>=0) == (divisor >=0) ? true : false;

        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);
        int result=0;
        while(dividend - divisor >= 0 ){
            int count=0;
            while(dividend  - (divisor << 1 << count) >= 0 ){
                count++;
            }

            result += 1 << count;
            dividend -= divisor << count ;
        }

        return sign?result: -result ;
    }
}
```








```
class Solution {
    public int divide(int dividend, int divisor) {
        if (dividend == Integer.MIN_VALUE && divisor == -1) return Integer.MAX_VALUE; //Cornor case when -2^31 is divided by -1 will give 2^31 which doesnt exist so overflow

        boolean negative = dividend < 0 ^ divisor < 0; //Logical XOR will help in deciding if the results is negative only if any one of them is negative

        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);
        int quotient = 0, subQuot = 0;

        while (dividend - divisor >= 0) {
            for (subQuot = 0; dividend - (divisor << subQuot << 1) >= 0; subQuot++);
            quotient += 1 << subQuot; //Add to the quotient
            dividend -= divisor << subQuot; //Substract from dividend to start over with the remaining
        }
        return negative ? -quotient : quotient;
    }
}
```

4. Power Set (this is very important)

Power Set

Easy Accuracy: 48.41% Submissions: 14115 Points: 2

Given a string S find all possible subsequences of the string in lexicographically-sorted order.

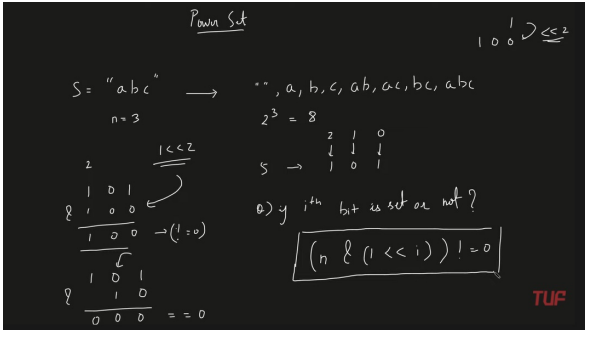
Example 1:

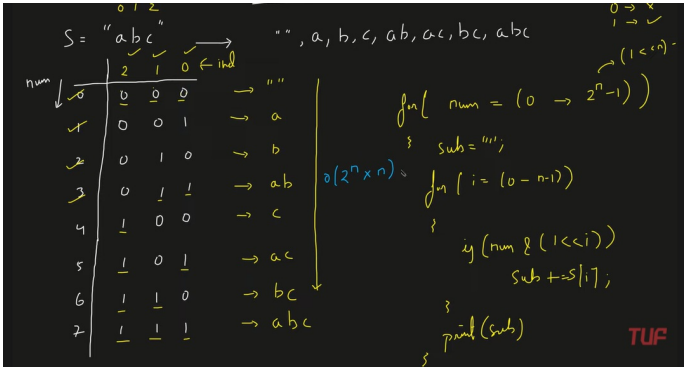
```
Input : str = "abc"
Output: a ab abc ac b bc c
Explanation : There are 7 substrings that
can be formed from abc.
```

Example 2:

```
Input: str = "aa"
Output: a a aa
Explanation : There are 3 substrings that
can be formed from aa.
```

Your Task:
You don't need to read ot print anything. Your task is to complete the function **AllPossibleStrings()** which takes S as input parameter and returns a list of all possible substrings(non-empty) that can be formed from S in lexicographically-sorted order.






```cpp
vector<string> AllPossibleStrings(string s){
                int n = s.size();
                vector<string> ans;
                for(int num = 0; num < (1 << n); num++) {
                    string sub = "";
                    for(int i = 0;i<n;i++) {
                        if(num & (1<<i)) {
                            sub += s[i];
                        }
                    }
                    if(sub.size() > 0)
                        ans.push_back(sub);
                }
                sort(ans.begin(), ans.end());
                return ans;
            }
```

Find MSB in o(1)

You are given a positive integer 'N'. Your task is to find the greatest integer less than or equal to 'N' which is a power of 2.

For Example:

```
If N = 14, then the nearest integer that is less than or equal to 14 and is a power of two is 8(2^3).
So, the answer is 8.
```

Follow Up:

```
Can you solve this in constant time and space complexity?
```

Input Format:

```
The first line contains an integer 'T' which denotes the number of test cases. Then, the 'T' test
cases area as follow.

The first and only line of each test case contains a single integer 'N'.
```

Output Format:

```
For each test case, print the nearest integer that is less than or equal to 'N' and also is a power
of 2.

Output for each test case will be printed in a separate line.
```

Sample Input 1:

```
2
4
22
```

Sample Output 1:

```
4
16
```

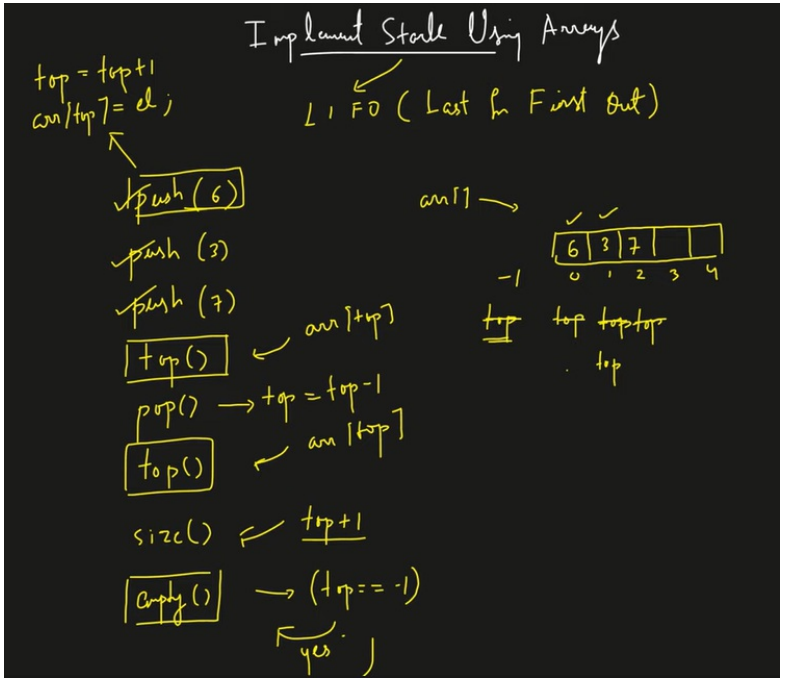
Explanation For Sample 1:

```
For the first test case, 4 itself is a power of two.
For the second test case, the nearest integer that is less than or equal to 22 and also is a power of
two is 16.
```

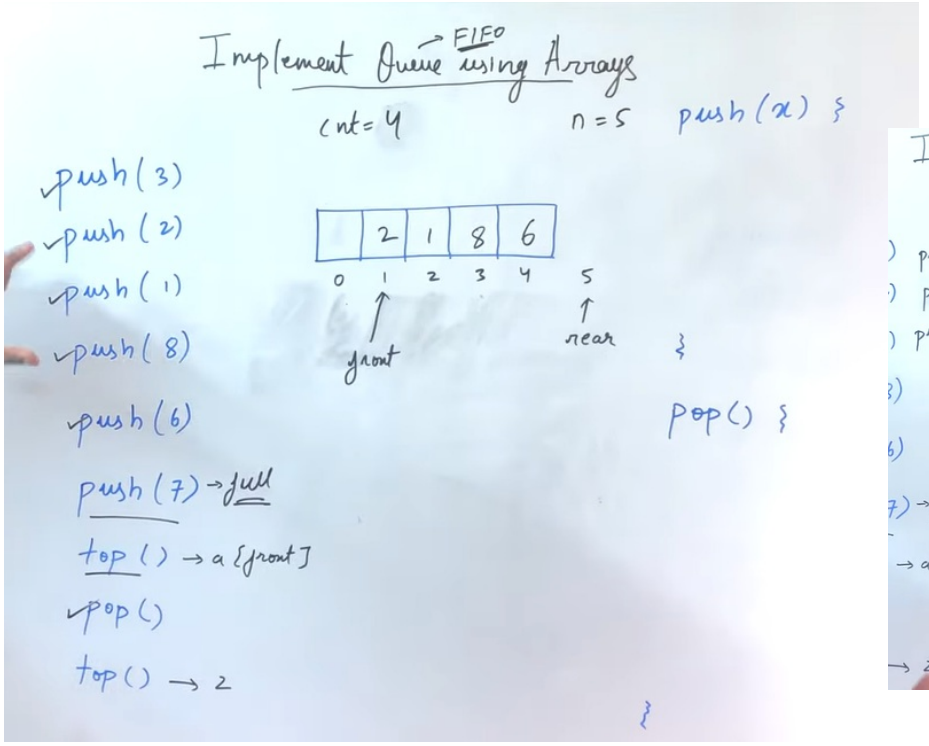
Find the square of a number without using multiplication or division operators.

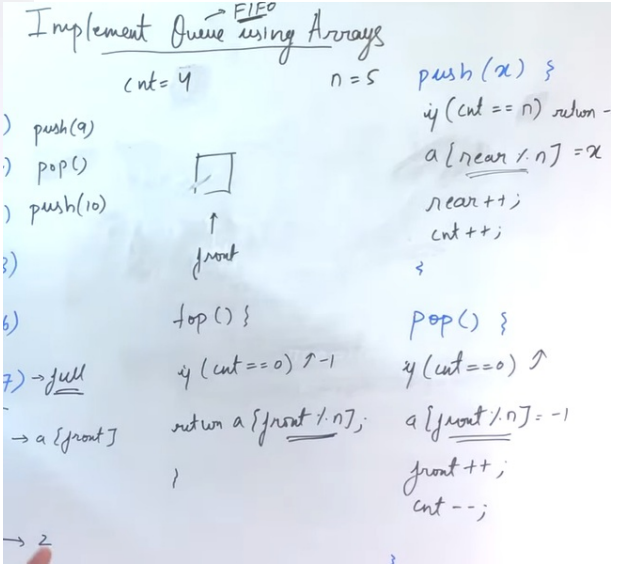
Day13: (Stack and Queue)

1. Implement Stack Using Arrays



2. Implement Queue Using Arrays





3. Implement Stack using Queue (using single queue)

225. Implement Stack using Queues

Easy 1486 743 Add to List Share

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

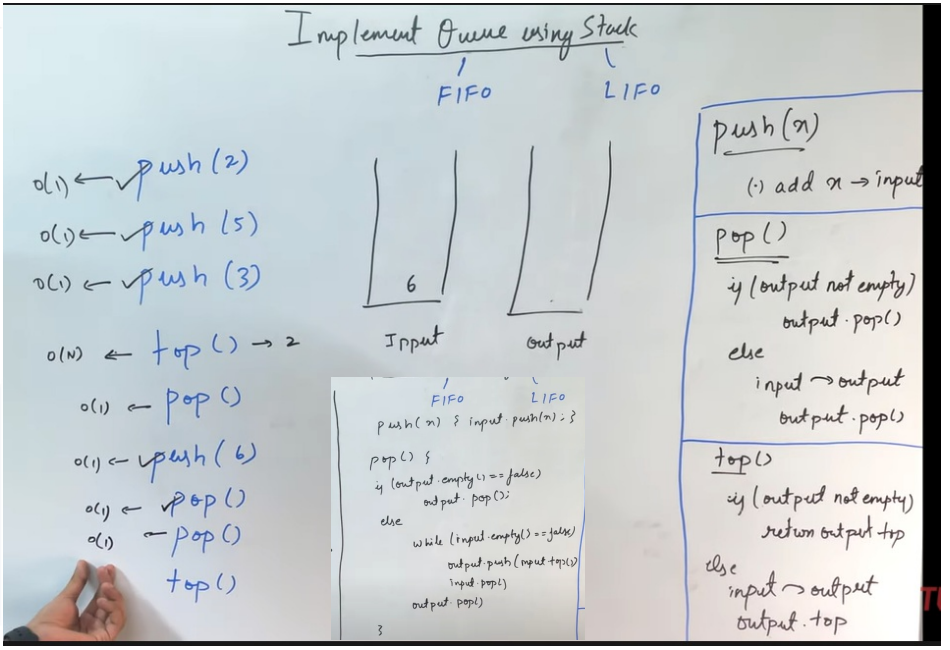
- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Example 1:

```
Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation
MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False
```

Constraints:

- `1 <= x <= 9`
- At most `100` calls will be made to `push`, `pop`, `top`, and `empty`.
- All the calls to `pop` and `top` are valid.



4. **Implement Queue using Stack (0(1) amortised method)**

5. Check for balanced parentheses

20. Valid Parentheses

Easy 9483 375 Add to List Share

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> st = new Stack<>(); // We will put following characters in stack.

        for(int i = 0; i < s.length(); i++){ // Run a loop to insert, check and remove
            char ch = s.charAt(i); // Get the charcters from the string

            if(ch == '(' || ch == '{' || ch == '['){ // If the brackets are opening in the starting, we will only insert them to the stack
                st.push(ch);
            }
            // Now we will check for every type of closing bracket's,
            else if(ch == ')'){  // like in this one we are dealing with closing circulary bracket
                if(st.size() == 0){ // If the stack size is zero and closing bracket's left, return false;
                    return false;
                }
                else if(st.peek() != '('){ // If the opening bracket is not similar to that which we are looking for return false;
                    return false;
                }
                else{ // If the closing bracket is similar to the opening bracket remove it from stack.
                    st.pop();
                }
            }
            else if(ch == '}'){  // In this one we are dealing with closing curly bracket
                if(st.size() == 0){
                    return false;
                }
                else if(st.peek() != '{'){
                    return false;
                }
                else{
                    st.pop();
                }
            }
            else if(ch == ']'){  // And in this one we are dealing with closing square bracket,
                if(st.size() == 0){
                    return false;
                }
                else if(st.peek() != '['){
                    return false;
                }
                else{
                    st.pop();
                }
            }
        }
        // Coming out of the loop;
        if(st.size() == 0){ // Check if stack size if 0 all remove it means it's Valid Parentheses;
            return true; // return true;
        }
        else{ // Otherwise
            return false;
        }
    }
}
```

```
class Solution {
    public boolean isValid(String s) {
        // If the length is odd, return false
        if (s == null || s.length() % 2 != 0) {
            return false;
        }

        int sLen = s.length();
        if (sLen == 0) {
            return true;
        }

        // First Char cannot be closing bracket
        char firstChar = s.charAt(0);
        if (firstChar == ')' || firstChar == '}' || firstChar == ']') {
            return false;
        }
        // Last Char cannot be open bracket
        char lastChar = s.charAt(sLen - 1);
        if (lastChar == '(' || lastChar == '{' || lastChar == '[') {
            return false;
        }

        Deque<Character> stack = new ArrayDeque<>();
        for (int i = 0; i < sLen; i++) {
            char c = s.charAt(i);
            if (c == '(') {
                stack.push(')');
            } else if (c == '{') {
                stack.push('}');
            } else if (c == '[') {
                stack.push(']');
            } else if (stack.isEmpty() || c != stack.pop()) {
                return false;
            }

            // Since there are more characters in stack than remaining characters in S, we
            // can early exit.
            if (stack.size() > sLen - i) {
                return false;
            }
        }

        return stack.isEmpty();
    }
}
```

6. Next Greater Element

496. Next Greater Element I

Easy 1256 102 Add to List Share

The **next greater element** of some element `x` in an array is the **first greater** element that is **to the right** of `x` in the same array.

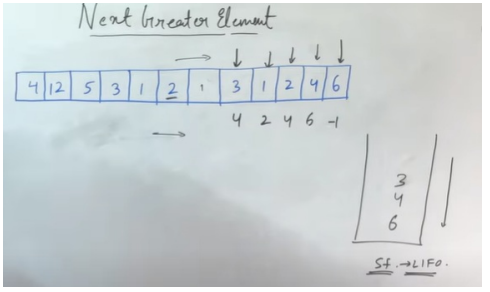
You are given two **distinct 0-indexed** integer arrays `nums1` and `nums2`, where `nums1` is a subset of `nums2`.

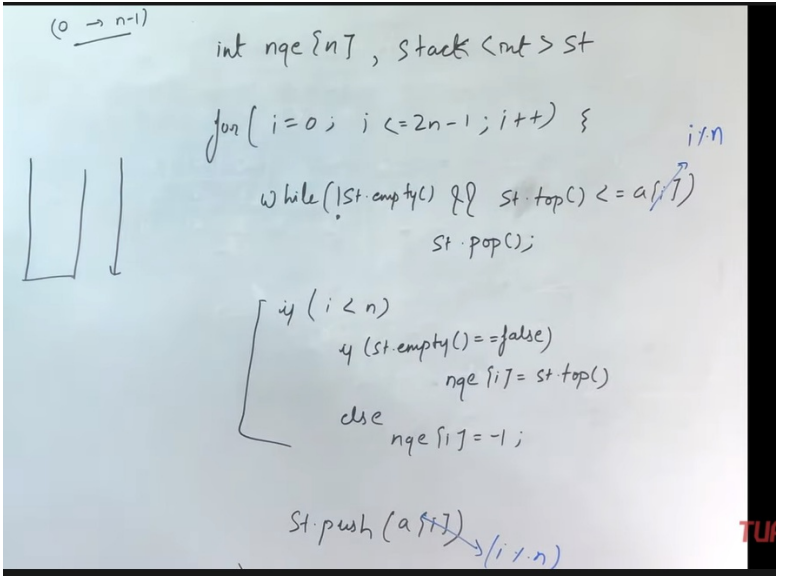
For each `0 <= i < nums1.length`, find the index `j` such that `nums1[i] == nums2[j]` and determine the **next greater element** of `nums2[j]` in `nums2`. If there is no next greater element, then the answer for this query is `-1`.

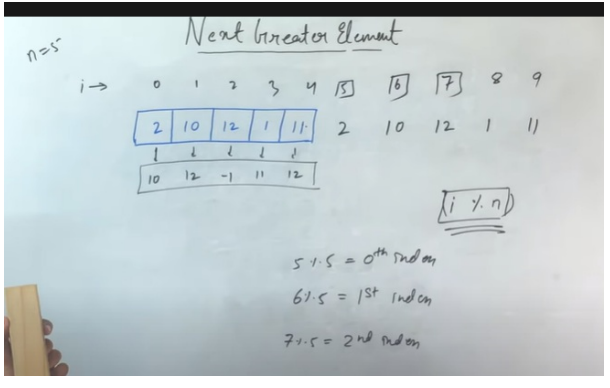
Return *an array* `ans` *of length* `nums1.length` *such that* `ans[i]` *is the* ***next greater element*** *as described above.*

Example 1:

```
Input: nums1 = [4,1,2], nums2 = [1,3,4,2]
Output: [-1,3,-1]
Explanation: The next greater element for each value of nums1 is as follows:
- 4 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.
- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.
```







7. Sort a Stack

Sort a stack

Easy Accuracy: 50.51% Submissions: 47712 Points: 2

Given a stack, the task is to sort it such that the top of the stack has the greatest element.

Example 1:

```
Input:
Stack: 3 2 1
Output: 3 2 1
```

Example 2:

```
Input:
Stack: 11 2 32 3 41
Output: 41 32 11 3 2
```

Your Task:
You don't have to read input or print anything. Your task is to complete the function **sort()** which sorts the elements present in the given stack. (The sorted stack is printed by the driver's code by popping the elements of the stack.)

Expected Time Complexity: O(N*N)
Expected Auxilliary Space: O(N) recursive.

Day - 14

1. Next Smaller Element

Similar to previous question next greater element, just do pop the greater elements out ..

Nearest Smaller Element

Easy 93 4 Add to favorites

Asked In: AMAZON MICROSOFT

Given an array, find the **nearest** smaller element G[i] for every element A[i] in the array such that the element has an **index smaller than i**.

More formally,

```
G[i] for an element A[i] = an element A[j] such that
j is maximum possible AND
j < i AND
A[j] < A[i]
```

Elements for which no smaller element exist, consider next smaller element as -1.

Input Format

The only argument given is integer array A.

Output Format

Return the integar array G such that G[i] contains nearest smaller number than A[i].If no such element occurs G[i] should be -1.

For Example

```
Input 1:
    A = [4, 5, 2, 10, 8]
Output 1:
    G = [-1, 4, -1, 2, 2]
Explaination 1:
    index 1: No element less than 4 in left of 4, G[1] = -1
    index 2: A[1] is only element less than A[2], G[2] = A[1]
    index 3: No element less than 2 in left of 2, G[3] = -1
    index 4: A[3] is nearest element which is less than A[4], G[4] = A[3]
    index 4: A[3] is nearest element which is less than A[5], G[5] = A[3]

Input 2:
    A = [3, 2, 1]
Output 2:
    [-1, -1, -1]
Explaination 2:
    index 1: No element less than 3 in left of 3, G[1] = -1
    index 2: No element less than 2 in left of 2, G[2] = -1
    index 3: No element less than 1 in left of 1, G[3] = -1
```

2. LRU cache (vvvv. imp)

146. LRU Cache

Medium 10586 418 Add to List Share

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Example 1:

Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
lRUCache.get(4);    // return 4
```

```java
class LRUCache {
    Node head = new Node(0,0), tail = new Node(0,0);
    Map<Integer, Node> map = new HashMap();
    int capacity;

    public LRUCache(int _capacity) {
        // this.capacity = capacity;
        capacity = _capacity;
        head.next = tail;
        tail.prev = head;
    }

    public int get(int key) {
        if(map.containsKey(key)){
            Node node = map.get(key);
            remove(node);
            insert(node);
            return node.value;
        }else{
            return -1;
        }
    }

    public void put(int key, int value) {
        if(map.containsKey(key)){
            remove(map.get(key));
        }
        if(map.size() == capacity){
            remove(tail.prev);
        }
        insert(new Node(key, value));
    }

    private void remove(Node node){
        map.remove(node.key);
        node.prev.next = node.next;
        node.next.prev = node.prev;
    }
    private void insert(Node node){
        map.put(node.key, node);
        Node headNext = head.next;
        head.next = node;
        node.prev = head;
        headNext.prev = node;
        node.next = headNext;
    }
    class Node{
        Node prev, next;
        int key, value;
        Node(int _key, int _value){
            key = _key;
            value = _value;
        }
    }
```

3. LFU Cache

460. LFU Cache

Hard 2561 178 Add to List Share

Design and implement a data structure for a Least Frequently Used (LFU) cache.

Implement the `LFUCache` class:

- `LFUCache(int capacity)` Initializes the object with the `capacity` of the data structure.
- `int get(int key)` Gets the value of the `key` if the `key` exists in the cache. Otherwise, returns `-1`.
- `void put(int key, int value)` Update the value of the `key` if present, or inserts the `key` if not already present. When the cache reaches its `capacity`, it should invalidate and remove the **least frequently used** key before inserting a new item. For this problem, when there is a **tie** (i.e., two or more keys with the same frequency), the **least recently used** `key` would be invalidated.

To determine the least frequently used key, a **use counter** is maintained for each key in the cache. The key with the smallest **use counter** is the least frequently used key.

When a key is first inserted into the cache, its **use counter** is set to `1` (due to the `put` operation). The **use counter** for a key in the cache is incremented either a `get` or `put` operation is called on it.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Input

```
["LFUCache", "put", "put", "get", "put", "get", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [3], [4, 4], [1], [3], [4]]
```

Output

```
[null, null, null, 1, null, -1, 3, null, -1, 3, 4]
```

Explanation

```
// cnt(x) = the use counter for key x
// cache=[] will show the last used order for tiebreakers (leftmost element is  most recent)
LFUCache lfu = new LFUCache(2);
lfu.put(1, 1);   // cache=[1,_], cnt(1)=1
lfu.put(2, 2);   // cache=[2,1], cnt(2)=1, cnt(1)=1
lfu.get(1);      // return 1
                 // cache=[1,2], cnt(2)=1, cnt(1)=2
lfu.put(3, 3);   // 2 is the LFU key because cnt(2)=1 is the smallest, invalidate 2.
                 // cache=[3,1], cnt(3)=1, cnt(1)=2
lfu.get(2);      // return -1 (not found)
lfu.get(3);      // return 3
                 // cache=[3,1], cnt(3)=2, cnt(1)=2
lfu.put(4, 4);   // Both 1 and 3 have the same cnt, but 1 is LRU, invalidate 1.
                 // cache=[4,3], cnt(4)=1, cnt(3)=2
lfu.get(1);      // return -1 (not found)
lfu.get(3);      // return 3
                 // cache=[3,4], cnt(4)=1, cnt(3)=3
lfu.get(4);      // return 4
                 // cache=[3,4], cnt(4)=2, cnt(3)=3
```

```java
class LFUCache {

    final int capacity;
    int curSize;
    int minFrequency;
    Map<Integer, DLLNode> cache;
    Map<Integer, DoubleLinkedList> frequencyMap;

    /**/
    /*
    * @param capacity: total capacity of LFU Cache
    * @param curSize: current size of LFU cache
    * @param minFrequency: frequency of the last linked list (the minimum frequency of entire LFU cache)
    * @param cache: a hash map that has key to Node mapping, which used for storing all nodes by their keys
    * @param frequencyMap: a hash map that has key to linked list mapping, which used for storing all
    * double linked list by their frequencies
    * */
    public LFUCache(int capacity) {
        this.capacity = capacity;
        this.curSize = 0;
        this.minFrequency = 0;

        this.cache = new HashMap<>();
        this.frequencyMap = new HashMap<>();
    }

    /** get node value by key, and then update node frequency as well as relocate that node **/
    public int get(int key) {
        DLLNode curNode = cache.get(key);
        if (curNode == null) {
            return -1;
        }
        updateNode(curNode);
        return curNode.val;
    }

    /**
     * add new node into LFU cache, as well as double linked list
     * condition 1: if LFU cache has input key, update node value and node position in list
     * condition 2: if LFU cache does NOT have input key
     *  - sub condition 1: if LFU cache does NOT have enough space, remove the Least Recent Used node
     *  in minimum frequency list, then add new node
     *  - sub condition 2: if LFU cache has enough space, add new node directly
     * */
    public void put(int key, int value) {
        // corner case: check cache capacity initialization
        if (capacity == 0) {
            return;
        }

        if (cache.containsKey(key)) {
            DLLNode curNode = cache.get(key);
            curNode.val = value;
            updateNode(curNode);
        }
        else {
            curSize++;
            if (curSize > capacity) {
                // get minimum frequency list
                DoubleLinkedList minFreqList = frequencyMap.get(minFrequency);
                cache.remove(minFreqList.tail.prev.key);
                minFreqList.removeNode(minFreqList.tail.prev);
                curSize--;
            }
            // reset min frequency to 1 because of adding new node
            minFrequency = 1;
            DLLNode newNode = new DLLNode(key, value);

            // get the list with frequency 1, and then add new node into the list, as well as into LFU cache
            DoubleLinkedList curList = frequencyMap.getOrDefault(1, new DoubleLinkedList());
            curList.addNode(newNode);
            frequencyMap.put(1, curList);
            cache.put(key, newNode);
        }
    }

    public void updateNode(DLLNode curNode) {
        int curFreq = curNode.frequency;
        DoubleLinkedList curList = frequencyMap.get(curFreq);
        curList.removeNode(curNode);

        // if current list the the last list which has lowest frequency and current node is the only node in that list
        // we need to remove the entire list and then increase min frequency value by 1
        if (curFreq == minFrequency && curList.listSize == 0) {
            minFrequency++;
        }

        curNode.frequency++;
        // add current node to another list has current frequency + 1,
        // if we do not have the list with this frequency, initialize it
        DoubleLinkedList newList = frequencyMap.getOrDefault(curNode.frequency, new DoubleLinkedList());
        newList.addNode(curNode);
        frequencyMap.put(curNode.frequency, newList);
    }

    /**
     * @param key: node key
     * @param val: node value
     * @param frequency: frequency count of current node
     * (all nodes connected in same double linked list has same frequency)
     * @param prev: previous pointer of current node
     * @param next: next pointer of current node
     * */
    class DLLNode {
        int key;
        int val;
        int frequency;
        DLLNode prev;
        DLLNode next;

        public DLLNode(int key, int val) {
            this.key = key;
            this.val = val;
            this.frequency = 1;
        }
    }

    /*
    * @param listSize: current size of double linked list
    * @param head: head node of double linked list
    * @param tail: tail node of double linked list
    * */
    class DoubleLinkedList {
        int listSize;
        DLLNode head;
        DLLNode tail;
        public DoubleLinkedList() {
            this.listSize = 0;
            this.head = new DLLNode(0, 0);
            this.tail = new DLLNode(0, 0);
            head.next = tail;
            tail.prev = head;
        }

        /** add new node into head of list and increase list size by 1 **/
        public void addNode(DLLNode curNode) {
            DLLNode nextNode = head.next;
            curNode.next = nextNode;
            curNode.prev = head;
            head.next = curNode;
            nextNode.prev = curNode;
            listSize++;
        }

        /** remove input node and decrease list size by 1**/
        public void removeNode(DLLNode curNode) {
            DLLNode prevNode = curNode.prev;
            DLLNode nextNode = curNode.next;
            prevNode.next = nextNode;
            nextNode.prev = prevNode;
            listSize--;
        }

    }
}

/**
 * Your LFUCache object will be instantiated and called as such:
 * LFUCache obj = new LFUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```

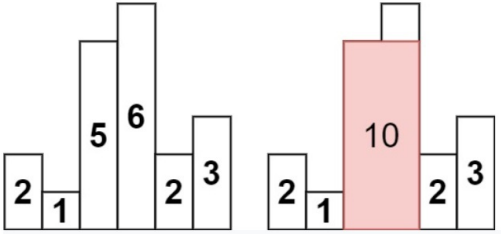
4. Largest rectangle in histogram Two pass:

84. Largest Rectangle in Histogram

Hard 7464 118 Add to List Share

Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram.*

Example 1:




```
Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10
units.
```

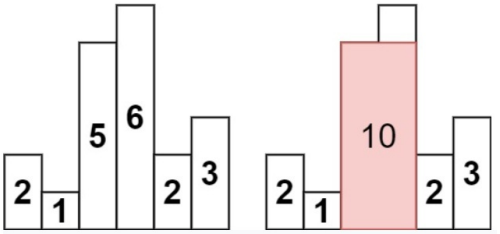
4. Largest rectangle in histogram One pass:

84. Largest Rectangle in Histogram

Hard 7464 118 Add to List Share

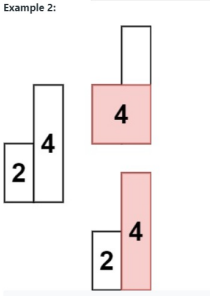
Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram.*

Example 1:



Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1. The largest rectangle is shown in the red area, which has an area = 10 units.

Example 2:



Input: heights = [2,4]
Output: 4