**---------------------------------- BIG O NOTATION--------------------------**

Big O to describe the performance of an algorithm. Watch mosh hamedani

**----------------------------------BASIC MATHS------------------------------**

**Count Digits:**

int x=1234 -> Answer is 4 //solution-> removing last digit and increment the res. Time complexity: O(d)

**public int** countDigits(**int** x){  
 **int** res=0;  
 **while** (x>0){  
 x=x/10;  
 res++;  
 }  
 **return** res;  
}

**Palindrome number and reverse int :** reverse integer and check the reverse number is same the given numbers. //% =vagses

**public boolean** isPal(**int** n){//o(n)  
 **int** rev=0;  
 **int** temp=n;  
 **while**(temp !=0){  
 **int** ld=temp%10;  
 rev=rev\*10+ld;  
 temp=temp/10;  
 }  
 **return** (rev==n);  
}

**Factorial numbers:** !4=4\*3\*2\*1=24

//iterative solution  
**public int** factt(**int** n){  
 **int** res=1;  
 **for**(**int** i=2;i<=n;i++){  
 res=res\*i;  
 }  
 **return** res;  
}

**Trailing Zeros in Factorial:** 5! =120 then the trailing zero is 1.

//naïve method.Training Zeros in Factorial, , time complexity: O(n)  
 **public int** countZeros(**int** n){  
// int fact =1;  
// for(int i=2;i<=n;i++){  
// fact=fact\*i;  
// }   
 **var** fact=factt(n);  
 **int** res=0;  
 **while**(fact%10==0){  
 res++;  
 fact=fact/10;  
 }  
 **return** res;  
 }

\*extra-> integer overflow issue

1\*2\*3\*4\*5\*6\*7\*8\*9\*10….n

How many 2 and 5 pair have in factorial. But numbers of 5 are going to be less than numbers of 2 so I simply need to count how many number of 5 in this prime factorization. Every 5th number is going to have 5 as a prime factor.

Trailing zero count: n/5 + n/25 + n/125

//k<=log5^n, time complexity: O(log n)  
 **public int** countZeross(**int** n) {  
  
 **int** res=0;  
 **for**(**int** i=5;i<=n;i=i\*5){  
 res=res+(n/i);  
 }  
 **return** res;  
 }

**GCD or HCF of two numbers:**

A=4, B=6. largest number that divides both A and B. Ans is 2

A=100, B=200. One number divide itself to other number then output is lower number. GCD Ans is 100.

A=7, B=13. If two number don’t have common divisor expect 1. then Ans is 1

// Naïve solution: Start with smallest numbers and keep trying smaller numbers.

public int gcd(int a, int b) {

  int res = Math.min(a, b);

  while (res > 0) {

    if (a % res == 0 && b % res == 0) {

      break;

    }

    res--;

  }

  return res;

}

//Eucliclean Algorithm(basic)

public int gcdd(int a, int b) {

  while (a != b) {

    if (a > b)

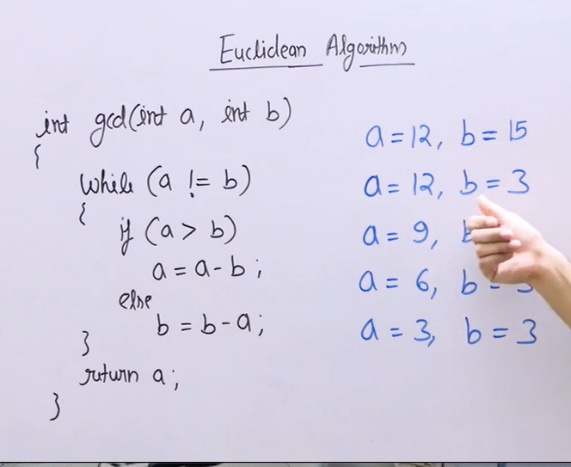
      a = a - b;

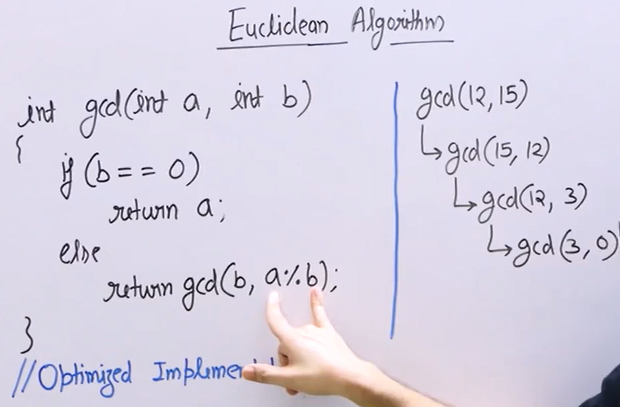
    else

      b = b - a;

  }

  return a;

}



// Eucliclean Algorithm

// x%y=x if x is smaller this will be for first call

public int gccdd(int a, int b) {

  if (b == 0)

    return a;

  else {

    return gcd(b, a % b);

  }

}

**LCM:**

A=4, B=6. largest number that divisible both A and B. Ans is 12. // A=12, B=15. Ans is 60

A=2, B=8. If largest number divisible by smallest number, then lcm is largest number. Ans is 8.

**//naïve solution**

**public int** lcmm(**int** a,**int** b){  
 **int** res=Math.max(a,b);  
 **while**(**true**){  
 **if**(res%a==0 && res%b==0) {**return** res;}  
 res++;  
 }  
 **return** res;  
}

//Lcm : num1\* num2 / gcd //time com: log(min(a,b))

public int lcm(int a, int b) {

  if (a > b && a % b == 0)

    return a;

  else if (b > a && b % a == 0)

    return b;

  return (a \* b) / gccdd(a, b);

}

**Prime Number:** 2 is only even prime number.1 is not prime or composite.

//Naïve solution

public boolean isprimee(int n){  
 if(n==1) return false;  
 for(int i=2;i<n;i++){  
 if(n%2==0) return false;  
 }  
 return true;  
}

//sqrt Math.sqrt(n) vvi  
public boolean isPrime(int n){  
 if(n==1) return false;  
 if(n==2 || n==3) return true;  
 if(n%2==0|| n%3==0) return false;  
 for(int i=5 ;i\*i<=n;i=i+6){  
 if(n%i==0) return false;  
 if(n%(i+2)==0) return false;  
 }  
 return true;  
}

**Prime Factor:** 450 prime factor is= 2\*3\*3\*5\*5; 84=2\*2\*3\*7

Efficient solution:

public static void primeFactor(int n){  
 if(n<=1) return ;  
 for(int i=2;i\*i<=n;i++){  
 while(n%i==0){  
 System.out.println(i);  
 n=n/i;  
 }  
 }  
 if(n>1){  
 System.out.println(n);  
 }  
}

More efficient solution: time complexity=sqrt(n)

public static void primeFactor(int n){  
 if(n<=1) return ;  
 while(n%2==0){  
 System.out.println(2);  
 n=n/2;  
 }  
 while(n%3==0){  
 System.out.println(3);  
 n=n/3;  
 }  
 for(int i=5;i\*i<=n;i=i+6){  
 while(n% i==0){  
 System.out.println(i);  
 n=n/i;  
 }  
 }  
 if(n>3){  
 System.out.println(n);  
 }  
}

**Computing Power:**

public static int power(int a,int n){  
 return (int) Math.pow(a,n);  
}

**some computing power solution will be added future**

**Is power of 3/4/2:**

public static boolean isPowerOfThree(int n) {  
 if (n<=0) return false;  
  
 while(n%3==0) n/=3;  
  
 return n==1;  
}

13. [Check if Number is a Sum of Powers of Three (1780) and rop cutting pray same](https://zhenchaogan.gitbook.io/leetcode-solution/leetcode-1780-check-if-number-is-a-sum-of-powers-of-three)

14.fibonacci series

public static void fibno(int n){  
 int firstTerm = 0, secondTerm = 1;  
 System.out.println("Fibonacci Series Upto " + n + ": ");  
  
 while (firstTerm <= n) {  
 System.out.print(firstTerm + ", ");  
  
 int nextTerm = firstTerm + secondTerm;  
 firstTerm = secondTerm;  
 secondTerm = nextTerm;  
  
 }  
}

15. Write code of 1st 100 Prime number series, Fibonacci series

16. # 1 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1 print this in one loop (can be use recursion) \*\*

17. 10 9 8 7 6 5 4 3 2 1 3 4 5 6 7 8 9 10

public static void oneLoop(){  
 int a=1;  
 for(int i=1 ; i>0 ; i+=a){  
 if (i==10)a=-a;  
 System.out.println(i);;  
 }  
 // return ;  
}

18. Missing Number

public static void missingNumber(int[] nums) {  
 Arrays.sort(nums);  
 int aa=nums[1]-nums[0];  
 for(int i=0; i<nums.length-1;i++){  
 if(!(nums[i]+aa==nums[i+1])){  
 System.out.println(nums[i]+aa);  
 }  
 }  
}

19. [Sum of Two Integers](https://leetcode.com/problems/sum-of-two-integers)//without + and – operation => a++ with loop length

public static int sumof2digit(int a, int b){  
 if(a>b){  
 for(int i=0;i<b;i++)a++;  
 return a;  
 }else{  
 for(int i=0;i<a;i++)b++;  
 return b;  
 }  
}

**Convert Celsius To Fahrenheit:**

public double cToF(int C)  
{  
 double f = C \* 9/5 + 32;  
  
 return f;  
}

**Digits In Factorial:**

public int digitsInFactorial(int N){  
 if (N <= 0){  
 return 1;}  
 double digits = 0;  
 for (int i=2; i<=N; i++)  
 digits += Math.log10(i);  
 return (int)(Math.floor (digits)) + 1;  
}

**Modular Multiplicative Inverse :**

**Input: a = 10, m = 17**

**Output: 12 Since (10\*12) mod 17 = 1, 12 is modulo inverse of 10(under 17).**

public int modInverse(int a, int m)  
{  
 for(int i=0;i<m;i++){  
 if((i\*a)%m==1) return i;  
 }  
 return -1;  
}

**----------------------------------BITWISE OPERATION-----------------------**

\*Notes (3-digit count like 4+2+1=7, 2-digit count like 2+1=3, don’t count zero only 1

**Bitwise AND:**

X=3, Y=6 then X&Y=2

X=3= 00….011

Y=6= 00….110

---------------------------------

X&Y=> 00…010 = 2. So, Ans is 2

**Bitwise OR:**

X=3, Y=6 then X|Y=2

X=3= 00….011

Y=6= 00….110

---------------------------------

X|Y=> 00…111 = 7. So, Ans is 7.

**Bitwise XOR:**

X=3, Y=6 then X ^Y=2

X=3= 00….011

Y=6= 00….110

---------------------------------

X ^Y=> 00…101 = 5. So, Ans is 5.

**Bitwise NOT:**

X=1= 00….01

~X=11….10 =2^32-X=2^32-1-1 , So X is -2.

Explain: -X = 2^32-X . if 32 bits all is one then its 2^32-1 but when add +1 then its have one zero means it has one -1 mean 2^32-1-1=2^32-2 So x is -2

Range of int -2^31 to (2^32-1) means -2147483648 to 2147483648

\*Leading bits if zero then its positive but if its 1 then its negative number

If x=5 then ~X= -6

**Left Shift <<:**

X=3 then X<<1 is

X=3= 000….0011

X<<1= 000…..0110=6

X<<2= 000…..01100=12 (4+8=12 , counting from right side like 1,2,4,8 and add all 1 value like 4+8=12)

X<<4= 000…..0110000=48

If x=-1 then x<<1 output is -2(2^32-1-1 = 2^32-2 =>-2)

X=-1=> 111…11(total 32 one) =2^32-1

X<<1=>111….10= 2^32-1-1=-2

**Right Shift >>:** Opposite of left shift.

X=>00…100001=33

X>>1=>00….010000 =16

X>>2=>00….001000 =8

Last bit removed then shifted right by 1 and first will add a zero

X=-2=>1111….10

X>>1=1111…11=2^32-1=-1

X>>2=1111…11=2^32-1=-1

**Unsigned Right Shift >>>:** for positive number unsigned right shift and right shift will be same but for negative also filed with 0 in unsigned

X=-2=>111…10

X>>>1==011….11=2^31-1=2147483647

X>>>2==001….11=2^30-1=1073741823

**Check if kth bit set or not:**

N=5, k=1 output: Yes, N=8, k=2 output: No

N=5=00…101

K=1 means last number that’s 1 so its Yes. If k=2 then it will No, cz position 2 is 0 value

//https://www.geeksforgeeks.org/check-whether-k-th-bit-set-not/  
public static void isKthBitSet(int n, int k)  
{  
 if (((n >> (k - 1)) & 1) >==1)  
 System.out.println("Yes3");  
 else  
 System.out.print("NOT SET");  
}

Count setbits:

public static int countSet(int n){  
 int res=0;  
 while(n>0){  
 if(n%2!=0){  
 res++;  
 }  
 n=n/2;  
 }  
 return res;  
}

**-------------------------------------RECURSION-----------------------------**

public static int fun(int n){  
 if(n==1) return 0;  
 else return 1+fun(n/2);  
 }  
 public static void fun2(int n){  
 if(n==0) return ;  
 System.out.println(n+" ");  
//recursion call in the last called tail recursion and this is fast than other recursion  
 fun2(n-1); }  
 public static boolean isPalidrome(String str, int a, int b){  
 if(a>=b)  
 return true;  
 return (str.charAt(a)==str.charAt(b) && isPalidrome(str,a+1,b-1));  
 }  
 //get sum of digit. Imp: n%10= gives last digits, n/10=remove the last digit  
 public static int getDigitSum(int n){  
 if(n==0)  
 return 0;  
 return getDigitSum(n/10)+n%10;  
 }  
 }

//rope cutting problem  
 // Input: n=5,a=2,b=5,c=1 O/P: 5 we make 5 pieces of length 1 each

// Input: n=23,a=12,b=9,c=11 O/P: 2 we make 2 pieces of length 11 and 12   
 public static int ropeCut(int n,int a,int b,int c){  
 if(n==0) return 0;  
 if(n<0) return -1;  
 if (a==1 || b==1||c== 1) return n;  
 else if ((a==2 || b==2||c== 2) && (n%2==0)) return n/2;  
 int res=Math.max(Math.max(ropeCut(n-a,a,b,c),ropeCut(n-b,a,b,c)),ropeCut(n-c,a,b,c));  
 if(res==-1) return -1;  
 return res+1;  
 }

**\*\* tower of Hanoi**

**----------------------------------ARRAYS-----------------------------------**

1. Kth largest numbers /second largest element

public static int kthLNumber(int[] nums, int k) {  
 //if second largest then k=2  
 Arrays.sort(nums);  
 return nums[nums.length-k];  
}

//another solution

public static int secondL(int arr[]){  
 int highest=arr[0];  
 int secondH=0;  
  
 for(int i=1;i< arr.length;i++){  
 if(arr[i]>highest){  
 secondH=highest;  
 highest=arr[i];  
 }  
 else if (secondH< arr[i]){  
 secondH=arr[i];  
 }  
 }  
 return secondH;  
}

1. Find the maximum sum of the contiguous array / Maximum sum of continuous subarray of length K. //A problem related to Kadane's algorithm

public static int maxSum(int a[]){  
 int maxsum=0,curSUm=0;  
 int count=0;  
 for(int i=0;i<a.length;i++ ){  
 curSUm=curSUm+a[i];  
 if(curSUm>maxsum){  
 maxsum=curSUm;  
 }  
 if(curSUm<0){  
 curSUm=0;  
 }  
 // return maxsum;// https://www.youtube.com/watch?v=HCL4\_bOd3-4  
 }  
 return maxsum;  
}

1. Remove duplicate and count array without duplicates from sorted array. Repeated element using hashset/map. Find elements that occurs only once in array.

public static int removeDup(int[] arr){  
 int res=1;  
 int []ar2=new int[arr.length];  
 ar2[0]=arr[0];  
 for(int i=1;i<arr.length;i++){  
 if(arr[i]!=arr[res-1]){  
 arr[res]=arr[i];  
 ar2[res]=arr[i];  
 res++;  
 }  
 }  
 System.out.println(Arrays.toString(ar2));  
 System.out.println(Arrays.toString(arr));  
 return res;  
}

1. Move all zero to the end

public static void moveZero(int [] arr){  
 int count =0;  
 int a=0;  
 for(int i=0;i<arr.length;i++){  
 if(arr[i]!=0){  
 a=arr[count];  
 arr[count]=arr[i];  
 arr[i]=a;  
  
 count++;  
 }  
 }  
 System.out.println(Arrays.toString(arr));  
}

1. Left rotate an Array by d

First rotate the first n array then swap other element after swap all all to get the dth rotate

public static void leftR(int []arr,int n){  
 revA(arr,0,n-1);  
 revA(arr,n,arr.length-1);  
 revA(arr,0,arr.length-1);  
 System.out.println(Arrays.toString(arr));  
  
}  
public static void revA(int[]arr,int low,int high){  
 int a=0;  
 while(low<high){  
 a=arr[low];  
 arr[low]=arr[high];  
 arr[high]=a;  
 low++;high--;  
 }  
}

1. leaders in an array. Take largest from right to left and print the leader. Equal value is not leader

public static void leadersA(int[]arr){  
 int curl= arr[arr.length-1];  
 int a=1;  
 System.out.print(curl+" ");  
 int aa[]=new int[arr.length];  
 aa[0]=curl;  
 for(int i=arr.length-2;i>=0;i--){  
 if(curl<arr[i]){  
 curl=arr[i];  
 aa[a]=arr[i];  
 a++;  
 System.out.println(curl);  
 }  
 }  
 System.out.println(Arrays.toString(aa));  
}

1. Best time to buy and sell I( Leetcode-121)

public int maxProfit1(int[] prices) {  
 int max=0;  
 int mins=prices[0];  
  
 for(int i=0;i<prices.length;i++){

check which is minimum  
 mins=Math.min(mins,prices[i]);

find profit  
 int profit=prices[i]-mins;

add max profit in maxp  
 maxp=Math.max(max,profit);  
  
 }  
 return maxp;  
}

1. Best Time to Buy and Sell Stock II(Leetcode-122)

Arr[]={1,5,3,8,12}. Then first diff is 5-1=4 then 8-3=5 then 12-8=4 total 13

public int maxProfit2(int[] prices) {  
 int p=0;  
 for(int i=1;i<prices.length;i++){  
 if(prices[i]>prices[i-1]){  
 p+=(prices[i]-prices[i-1]);  
 }  
 }  
 return p;  
}

1. Trapping rain water\*\*

public int trap(int[] height) {  
  
 int l = 0, r = height.length - 1, sum = 0, lMax = 0, rMax = 0;  
 while(l <= r){  
 lMax = Math.max(height[l], lMax);  
 rMax = Math.max(height[r], rMax);  
 //why? because, for example if the lMax is smaller,  
 //we can sure that how much water could be trapped at the left pointer position is decided by the left side.  
 if(lMax < rMax){  
 sum += lMax - height[l++];  
 }else{  
 sum += rMax - height[r--];  
 }  
 }  
 return sum;  
}

1. Sliding window technique\*\*\*\*
2. Prefix sum array\*\*\*

**------------------------------------Searching--------------------------------**

1. **Find Index of an array**

public static int indexBS(int []arr,int x){  
 int low=0,high=arr.length-1;  
 while (low<=high){  
 int mid=(low+high)/2;  
 if(arr[mid]==x) return mid;  
 else if (arr[mid] > x)high=mid-1;  
 else low=mid+1;  
 }  
 return -1;  
}

**recursive solution:**

public static int indexBS(int []arr,int low,int high,int x){  
 if(low>high) return -1;  
 int mid=(low+high)/2;  
 if(arr[mid]==x) return mid;  
  
 else if (arr[mid] > x)return indexBS(arr,low,mid-1,x);  
 else return indexBS(arr,mid+1,high,x);  
}

1. Index of first occurrences

public static int firstOcc(int [] arr,int x){  
  
 int low=0,high=arr.length-1;  
 while(low<=high){  
 int mid =(low+high)/2;  
 if(arr[mid]>x) high=mid-1;  
 else if (arr[mid]<x)low=mid+1;  
 else {  
 if((mid==0)|| arr[mid-1]!=arr[mid] )return mid;  
 else high=mid-1;  
 }  
  
 }  
 return -1;  
}

1. Index of last occurrences

public static int lastOcc(int [] arr,int x){  
  
 int low=0,high=arr.length-1;  
 while(low<=high){  
 int mid =(low+high)/2;  
 if(arr[mid]>x) high=mid-1;  
 else if (arr[mid]<x)low=mid+1;  
 else {  
 if((mid!= arr.length-1)|| arr[mid]!=arr[mid+1] )return mid;  
 else low=mid+1;  
 }  
  
 }  
 return -1;  
}

1. count occurrences in shorted /can use hash
2. count 1s in a sorted binary array. / Can use hash

**--------------------------------------Hashing--------------------------------**

**1.Count distinct elements**

public static int countDistinct(int[] arr){  
 HashSet<Integer> sett=new HashSet<>();  
 for (int j : arr) {  
 sett.add(j);  
 }  
 return sett.size();  
}

1. **Frequencies of an array elements**

public static void frequenciesElement(int[] arr){  
 HashMap<Integer,Integer> map=new HashMap<>();  
 for (int x:arr) {  
 map.put(x,map.getOrDefault(x,0)+1);  
 }   
 for(Map.Entry<Integer,Integer> e: map.entrySet()){  
 System.out.println(e.getKey()+" "+ e.getValue());  
 }  
}

1. **Intersection of two array**

public static int InerSection2Array(int[] arr1,int[] arr2){  
 HashSet<Integer> map1=new HashSet<>();  
 HashSet<Integer> map2=new HashSet<>();  
  
 int count=0;  
 for (int x:arr1) {  
 map1.add(x);  
 }  
 for (int y:arr2) {  
 map2.add(y);  
 }  
 for (Integer a : map1) {  
 if(map2.contains(a)){  
 count++;  
 }  
 }  
 return count;  
}

1. **Union of two unsorted array**

public static int Union2Array(int[] arr1,int[] arr2){  
 HashSet<Integer> map1=new HashSet<>();  
 int count=0;  
 for (int x:arr1) {  
 map1.add(x);  
 }  
 for (int y:arr2) {  
 map1.add(y);  
 }  
 System.out.println(Arrays.toString(map1.stream().sorted().toArray()));  
 return map1.size();  
}

1. **Pair with given sum in an unsorted array**

public static boolean sumArray(int [] arr,int sum){  
 HashSet<Integer> itt=new HashSet<>();

for(int i=0;i<arr.length;i++){  
 if(itt.contains(sum-arr[i])){  
 return true;  
 }  
 itt.add(arr[i]);  
 }  
 return false;  
}

1. **Subarray with zero sum {-3,2,1}=0**

public static boolean is0Subarray(int [] arr){  
 HashSet<Integer> h=new HashSet<>();  
 int preSum=0;  
 for (int j : arr) {  
 preSum += j;  
 if (h.contains(preSum)) {  
 return true;  
 }  
 if (preSum == 0) return true;  
  
 h.add(preSum);  
 }  
 return false;  
}

1. **Subarray with given sum**

public static boolean isSum(int [] arr,int sum){  
 HashSet<Integer> s=new HashSet<>();  
 int preSum=0;  
 for (int j : arr) {  
 preSum += j;  
 if(preSum==sum) return true;  
 if (s.contains(preSum-sum)) return true;  
 s.add(preSum);  
 }  
 return false;  
}

1. **Longest subarray with given sum\*\*\*\***

public static int maxlSA(int [] arr,int sum){  
 HashMap<Integer,Integer> h=new HashMap<>();  
 int preSum=0,res=0;  
 for(int i=0;i<arr.length;i++){  
 preSum += arr[i];  
 if(preSum==sum){  
 res=i+1;  
 }  
 if (!h.containsKey(preSum)) {  
 h.put(preSum,i);  
 }  
 if(h.containsKey(preSum-sum)){  
 res=Math.max(res,i-h.get(preSum-sum));  
 }  
  
 }  
 return res;  
}

1. **Longest Subarray with Equal Number Of 0s and 1s**

**Same as before question” Longest subarray with given sum” just need to make zeros to -1 and pass zero as sum. ---**[**Another solution**](https://www.interviewbit.com/blog/largest-subarray-of-0s-and-1s/)**---**

**10. Longest consecutive subsequence**

public static int lonCSub(int [] arr){  
 Arrays.sort(arr);  
 int res=1,cur=1;  
 for(int i=0;i<=arr.length;i++){  
 if(arr[i]==arr[i-1]+1){  
 cur++;  
 }else{  
 res=Math.max(res,cur);  
 cur=1;  
 }  
 }

**11.** **Count Distinct Elements in every Window of size k**

public static void countDistinct(int arr[], int K) {  
 // Creates an empty hashMap map  
 HashMap<Integer, Integer> map= new HashMap<>();  
 // Traverse the first window and store count  
 // of every element in hash map  
 for (int i = 0; i < K; i++){  
 map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);  
 }  
 // Print count of first window  
 System.out.print(map.size()+" ");  
 // Traverse through the remaining array  
 for (int i = K; i < arr.length; i++) {  
  
 // Remove first element of previous window  
 // If there was only one occurrence  
 if (map.get(arr[i - K]) == 1) {  
 map.remove(arr[i - K]);  
 } else // reduce count of the removed element  
 map.put(arr[i - K], map.get(arr[i - K]) - 1);  
  
 // Add new element of current window  
 // If this element appears first time,  
 // set its count as 1,  
 map.put(arr[i], map.getOrDefault(arr[i], 0) + 1);  
 // Print count of current window  
 System.out.print(map.size()+" ");  
 }  
}

**12. More than N/K occurrence**

public static void nkOccures(int arr[], int k){  
 HashMap<Integer, Integer> map= new HashMap<>();  
 int a= arr.length/k;  
 for (int j : arr) {  
 map.put(j, map.getOrDefault(j, 0) + 1);  
 }  
 for(Map.Entry<Integer, Integer> p: map.entrySet()){  
 if (p.getValue() > a) {  
 System.out.println(p.getKey() + " ");  
 }  
 }  
}

**another solution**

public static void nkOccures(int arr[], int k){  
 HashMap<Integer, Integer> map= new HashMap<>();  
 int a= arr.length/k;  
 for (int j : arr) {  
 map.put(j, map.getOrDefault(j, 0) + 1);  
 if (map.get(j)>= a) {  
 System.out.println(j+ " ");  
 }  
 }

**---------------------------------------Strings-------------------------------**

1. **Is palindrome**

public static boolean ispal(String s){  
 int b=0,e=s.length()-1; // we can use string builder  
 while(b<e){  
 if(s.charAt(b)!=s.charAt(e)) return false;  
 b++;e--;  
 }  
 return true;  
}

1. **Is substring to other**

public static boolean isSubSeq(String s1,String s2){  
 int j=0;  
 for(int i=0;i<s1.length() & j<s2.length();i++){  
 if(s1.charAt(i)==s2.charAt(j)) j++;  
 }  
 return j==s2.length();  
}

1. **are anagram**

public static boolean areAnagram(String str1, String str2) {  
 if (str1.length() != str2.length()){  
 return false;  
 }  
 char[] charArray1 = str1.toCharArray();  
 char[] charArray2 = str2.toCharArray();  
 Arrays.sort(charArray1);  
 Arrays.sort(charArray2);  
 return Arrays.equals(charArray1, charArray2);  
}

1. **left most repeating character**
2. **left most non repeating character**
3. **reverse word in a string**

String s="abc bca cab";  
String [] words=s.trim().split(" ");  
StringBuilder r=new StringBuilder();  
for(var i=words.length-1;i>=0;i--){  
 r.append(words[i]+" ");  
}  
System.out.println(r.toString().trim());

//another solution

public static String revW(String s){  
 String [] words=s.trim().split(" ");  
 Collections.reverse(Arrays.asList(words));  
 return String.join(" ",words);  
}

1. **pattern searching -rabin karp algorithm**
2. **check if string are rotations**

public static boolean areRotations(String s1, String s2){  
 if(s1.length()!=s2.length()) return false;  
 return ((s1 + s1).contains(s2));  
}

1. **count vowels**

public static int countVowels(String s){  
 int count =0;  
 String vowels="aeiou";  
 for(var ch:s.toLowerCase().toCharArray()){  
 if(vowels.indexOf(ch)!=-1) count++;  
 }  
 return count;  
 }

1. **remove duplicates //using hashset<character>**
2. **most repeated character // using hashmap**
3. **first repeted character //using hashmap**
4. **capitalize of first char in a word**

public static String capitalize(String s){  
 if(s.trim().isEmpty()) return "";  
 String[] words=s  
 .trim()  
 .replaceAll(" +"," ")  
 .split(" ");  
 for(var i=0;i<words.length;i++){  
 words[i]=words[i].substring(0,1).toUpperCase()  
 +words[i].substring(1).toLowerCase();  
 }  
 return String.join(" ",words);  
}

1. **sdd**

**--------------------------------LinkedList------------------------------------**

public class Node{  
 private int value;  
 private Node next;  
  
 public Node(int value) {  
 this.value = value;  
 }  
}

private Node first;  
private Node last;

1. **Addlast**

public void addLast( int item){  
 var node=new Node(item);  
 if(first== null){  
 first=last=node;  
 }  
 else{  
 last.next=node;  
 last=node;  
 }  
}

1. **Add first**

public void addFirst( int item){  
 var node=new Node(item);  
 if(first== null){  
 first=last=node;  
 }  
 else{  
 node.next=first;  
 first=node;  
 }  
}

1. **Index of**

public int indexOf(int item){  
 int index=0;  
 var current=first;  
 while(current!=null){  
 if(current.value==item) return index;  
 current=current.next;  
 index++;  
 }  
 return -1;  
}

1. **contains**

public boolean contains(int item){  
 return indexOf(item) !=-1;  
}

1. **remove first**

public void removeFirst(){  
 if(first ==null){  
 throw new NoSuchElementException("no item");  
 }  
 if(first==last){  
 first=last=null;  
 return;  
 }

1. **remove last**

public void removeLast(){  
 if(first==null){  
 throw new NoSuchElementException("no element");  
 }  
 var previous=getPrevious(last);  
 last=previous;  
 last.next=null;  
}  
private Node getPrevious(Node node){  
 var current=first;  
 while(current!=null){  
 if(current.next==last) return current;  
 current=current.next;  
 }  
 return null;  
}

1. **reverse**

public void reverse(){  
 //[10<-20<-30]  
 if(first==null)return;  
 var previous=first;  
 var current=first.next;  
 while(current !=null){  
 var next=current.next;  
 current.next=previous;  
 previous=current;  
 current=next;  
 }  
 last=first;  
 last.next=null;  
 first=previous;  
}

1. **Kth node from the end**

public int kthNodeFEnd(LinkedList list,int k){  
// var a=list.getFirst();  
// var b=list.getFirst();  
 if(list.isEmpty()){  
 throw new IllegalStateException("list is empty");  
 }  
 var a=first;  
 var b=first;  
 for(int i=0;i<k-1;i++){  
 b=b.next;  
 if(b==null){  
 throw new IllegalArgumentException("k is greather than the list");  
 }  
 }  
 while (b!=last){  
 a=a.next;  
 b=b.next;  
 }  
 return a.value;  
 }

1. **linkedlist is palindrome or not**

public static boolean ispalL(LinkedList n){  
 var a =n.toArray();  
 int c=0;  
 int d=a.length-1;  
 while(c<d){  
 if(a[c]!=a[d]){  
 //System.out.println("not palidrome");  
 return false;  
 }  
 c++;  
 d--;  
 }  
 return true;  
// System.out.println("palidrome");  
 }

1. **remove duplicates from ll-can use hashset**

public void removeDuplicate(Node head){  
 Node curr=head;  
 while (curr != null && curr.next != null) {  
  
 if(curr.value==curr.next.value){  
 curr.next=curr.next.next;  
 }  
 else {  
 curr=curr.next;  
 }  
 }  
}

1. **detect loop/cycle\*\*\***

public Node detectCycle(Node head){  
 Node slow=head;  
 Node fast=head;  
 while (fast!=null && fast.next!=null){  
 slow=slow.next;  
 fast=fast.next.next;  
 if(slow==fast){  
 return slow;  
 }  
 }  
 return null;  
}

1. **remove loop/cycle\*\*\* // jdi cycle delete korte hoy then previous node er next null kroe dilei hobe**

public Node detectFirstNode(Node head){  
 Node meet=detectCycle(head);  
 Node start=head;  
  
 while(start!=meet){  
 start=start.next;  
 meet=meet.next;  
 }  
 return start;  
}

///extra for delete

Var previous =start;

Previous.next=null;

1. **middle of ll-> first pointer and slow pointer\*\*\***

public void printMiddle(Node node){  
 var a=first;  
 var b=first;  
 while(b!=last & b.next!=last){  
 b=b.next.next;  
 a=a.next;  
 }  
 if(b==last){  
 System.out.println(a.value);  
 }else {  
 System.out.println(a.value+" "+a.next.value);  
 }  
}

1. **merge two shorted list\*\*\***

public Node twoSortedMerge(Node a, Node b){  
 if(a==null) return b;  
 if(b==null) return a;  
  
 Node head=null, tail=null;  
 if(a.value<=b.value){  
 head=tail=a;  
 a=a.next;  
 }  
 else {  
 head=tail=b;  
 b=b.next;  
  
 }  
 while (a!=null & b!=null){  
 if(a.value<=b.value){  
 tail.next=a;  
 tail=a;  
 a=a.next;  
 }else {  
 tail.next=b;  
 tail=b;  
 b=b.next;  
 }  
 if(a==null){  
 tail.next=b;  
 }else {  
 tail.next=a;  
 }  
   
 }  
 return head;  
}

**-----------------------------------Stacks------------------------------**

**1.Stack implementation using LinkedList and arrays**

**-----------------------------------Queues-------------------------------**

1. **reverse a queue using stack**
2. **queue implementation using arrays/ LinkedList/stack**
3. **priority queue**
4. **Maximums Of All Subarrays Of Size K(deque)**

**-----------------------------------Binary Tree-------------------------------**

public class Node{  
 private int value;  
 private Node leftc;  
 private Node rightC;  
  
 public Node(int value) {  
 this.value = value;  
 }  
}  
  
private Node root;  
public void insert(int value){  
 var node=new Node(value);  
 if(root ==null){  
 root=node;  
 return;  
 }  
  
 var current=root;  
 while (true){  
 if(value< current.value){  
 if(current.leftc==null){  
 current.leftc=node;  
 }  
 current=current.leftc;  
 }else {  
 if(current.rightC==null){  
 current.rightC=node;  
 }  
 current=current.rightC;  
 }  
 }  
}  
public boolean find(int value){  
 var current =root;  
 while(current!=null){  
 if(value<current.value){  
 current=current.leftc;  
 }  
 else if(value> current.value){  
 current=current.rightC;  
 }else return true;  
 }  
 return false;  
}  
public void traversePreOrder(){  
 traversePreOrder(root);  
}  
private void traversePreOrder(Node root){  
 //root(print)  
 //left  
 //right  
 if(root==null) return;  
 System.out.println(root.value);  
 traversePreOrder(root.leftc);  
 traversePreOrder(root.rightC);  
}  
public void traverseInOrder(Node root){  
 //left  
 //root(print)  
 //right  
 if(root==null) return;  
 traversePreOrder(root.leftc);  
 System.out.println(root.value);  
 traversePreOrder(root.rightC);  
}  
public void traversePostOrder(Node root){  
 //left  
 //right  
 //root(print)  
 if(root==null) return;  
 traversePreOrder(root.leftc);  
 traversePreOrder(root.rightC);  
 System.out.println(root.value);  
}  
public int height(){  
 return height(root);  
}  
//find height using post order traversal  
private int height(Node root){  
 if(root==null) return -1;  
 if(isLeaf(root)) return 0;  
 return 1+ Math.max(  
 height(root.leftc),  
 height(root.rightC)  
 );  
}  
private boolean isLeaf(Node node){  
 return node.leftc ==null && node.rightC ==null;  
}  
  
public int min(){  
 return min(root);  
}  
  
//find minimum value in a binary tree using post order traversal  
private int min(Node root){  
 if(isLeaf(root)) return root.value;  
 var left= min(root.leftc);  
 var right= min(root.rightC);  
 return Math.min(Math.min(left,right),root.value);  
  
}  
  
//find minimum value in binary search tree O(log n)  
private int minBsT(){  
 if(root==null)  
 throw new IllegalStateException("root null");  
 var current=root;  
 var last=current;  
 while(current!=null){  
 last=current;  
 current=current.leftc;  
 }  
 return last.value;  
}  
public boolean equals(Trees other){  
 if(other==null) return false;  
 return equals(root,other.root);  
}  
private boolean equals(Node first, Node second){  
 if(first ==null && second == null){  
 return true;  
 }  
 if(first !=null && second != null)  
 return first.value == second.value  
 && equals(first.leftc,second.leftc)  
 && equals(first.rightC,second.rightC);  
  
 return false;  
}  
  
//validate binary search tree using pre order  
public boolean isBinarySearchTree(){  
 return isBinarySearchTree(root,Integer.MIN\_VALUE,Integer.MAX\_VALUE);  
}  
private boolean isBinarySearchTree(Node root,int min, int max){  
 if(root==null) return true;  
 if(root.value < min || root.value >max) return false;  
 return isBinarySearchTree(root.leftc,min, root.value-1) &&  
 isBinarySearchTree(root.rightC,root.value+1,max);  
}  
  
public void printNodeAtDistance(int distance){  
 printNodeAtDistance(root,distance);  
}  
private void printNodeAtDistance(Node root,int distance){  
 if(root==null) {  
 return;  
 }  
 if(distance==0){  
 System.out.println(root.value);  
 return;  
 }  
 printNodeAtDistance(root.leftc,distance-1);  
 printNodeAtDistance(root.rightC,distance-1);  
  
  
}

**-------------------------------Tries-----------------------------------------**

public class Trie {  
  
 public static void main(String[] args) {  
 List<String> words = List.of("hello", "dog", "hell", "cat", "a", "hel","help","helps","helping");  
 Trie trie = new Trie(words);  
  
 System.out.println(trie.suggest("h"));  
 }  
  
 public class TrieNode {  
 Map<Character, TrieNode> children;  
 char c;  
 boolean isWord;  
  
 public TrieNode(char c) {  
 this.c = c;  
 children = new HashMap<>();  
 }  
  
 public TrieNode() {  
 children = new HashMap<>();  
 }  
  
 public void insert(String word) {  
 if (word == null || word.isEmpty())  
 return;  
 char firstChar = word.charAt(0);  
 TrieNode child = children.get(firstChar);  
 if (child == null) {  
 child = new TrieNode(firstChar);  
 children.put(firstChar, child);  
 }  
  
 if (word.length() > 1)  
 child.insert(word.substring(1));  
 else  
 child.isWord = true;  
 }  
  
 }  
  
 TrieNode root;  
  
 public Trie(List<String> words) {  
 root = new TrieNode();  
 for (String word : words)  
 root.insert(word);  
  
 }  
  
 public boolean find(String prefix, boolean exact) {  
 TrieNode lastNode = root;  
 for (char c : prefix.toCharArray()) {  
 lastNode = lastNode.children.get(c);  
 if (lastNode == null)  
 return false;  
 }  
 return !exact || lastNode.isWord;  
 }  
  
 public boolean find(String prefix) {  
 return find(prefix, false);  
 }  
  
 public void suggestHelper(TrieNode root, List<String> list, StringBuffer curr) {  
 if (root.isWord) {  
 list.add(curr.toString());  
 }  
  
 if (root.children == null || root.children.isEmpty())  
 return;  
  
 for (TrieNode child : root.children.values()) {  
 suggestHelper(child, list, curr.append(child.c));  
 curr.setLength(curr.length() - 1);  
 }  
 }  
  
 public List<String> suggest(String prefix) {  
 List<String> list = new ArrayList<>();  
 TrieNode lastNode = root;  
 StringBuffer curr = new StringBuffer();  
 for (char c : prefix.toCharArray()) {  
 lastNode = lastNode.children.get(c);  
 if (lastNode == null)  
 return list;  
 curr.append(c);  
 }  
 suggestHelper(lastNode, list, curr);  
 return list;  
 }  
}  
  
  
  
  
public class Tries {  
 public static void main(String[] args) {  
 HashSet<String > s=new HashSet<>();  
 s.add("one");  
 s.add("ton");  
 s.add("don");  
 if(s.contains("on")){  
 System.out.println(true);  
 }  
  
 }  
 public static int ALPHABETIC\_SIZE=26;  
 private class Node {  
 private char value;  
 private Node[] children = new Node[ALPHABETIC\_SIZE];  
 private HashMap<Character, Node> childrean = new HashMap<>();  
 private boolean isEndOfWord;  
  
 public Node(char value) {  
 this.value = value;  
 }  
  
 @Override  
 public String toString() {  
 return "value=" + value;  
 }  
 public boolean hasChild(char ch){  
 return childrean.containsKey(ch);  
 }  
 public void addChild(char ch){  
 childrean.put(ch,new Node(ch));  
 }  
 public Node getChild(char ch){  
 return childrean.get(ch);  
 }  
 }  
  
 private Node root=new Node(' ');  
  
 public void insert(String word){  
 var current=root;  
 for(var ch:word.toCharArray()){  
 if(!current.hasChild(ch)){  
 current.addChild(ch);  
 }  
 current=current.getChild(ch);  
 }  
 current.isEndOfWord=true;  
 }  
 }

**--------------------------------------Graph--------------------------------------------------------------------------------------------------------------**

public class Node {  
 private String label;  
  
 public Node(String label) {  
 this.label = label;  
 }  
 @Override  
 public String toString() {  
 return label;  
 }  
  
}

private Map<String, Node> nodes=new HashMap<>();  
 private Map<Node, List<Node>> adjeancyList=new HashMap<>();  
  
 public void addNode(String label){  
 var node=new Node(label);  
 nodes.putIfAbsent(label,node);  
 adjeancyList.putIfAbsent(node,new ArrayList<>());  
 }  
 public void addEdge(String from, String to){  
 var fromNode=nodes.get(from);  
 if(fromNode ==null){  
 throw new IllegalArgumentException();  
 }  
 var toNode=nodes.get(to);  
 if(toNode== null){  
 throw new IllegalArgumentException();  
 }  
  
 adjeancyList.get(fromNode).add(toNode);  
 adjeancyList.get(toNode).add(fromNode);  
  
 }  
 public void print(){  
 for(var source:adjeancyList.keySet()){  
 var targets=adjeancyList.get(source);  
 if(!targets.isEmpty()){  
 System.out.println(source+" is connected to "+targets);  
 }  
 }  
 }  
  
 public void removeNode(String label){  
 var node=nodes.get(label);  
 if(node==null){  
 return;  
 }  
 for(var n: adjeancyList.keySet()){  
 adjeancyList.get(n).remove(node);  
 }  
 adjeancyList.remove(node);  
 nodes.remove(node);  
  
 }  
 public void removeEdge(String from , String to){  
 var fromNode=nodes.get(from);  
 var toNode=nodes.get(to);  
 if(fromNode==null || toNode==null){  
 return;  
 }  
 adjeancyList.get(fromNode).remove(toNode);  
 }  
  
 ////DFS recursive solution  
 public void traverseDepthFirstR(String root){  
 traverseDepthFirstR(nodes.get(root),new HashSet<>());  
 }  
 private void traverseDepthFirstR(Node root, Set<Node> visited){  
 System.out.println(root);  
 visited.add(root);  
  
 for(var node: adjeancyList.get(root)){  
 if(!visited.contains(node)){  
 traverseDepthFirstR(node,visited);  
 }  
 }  
  
 }  
 //iterative solution  
 public void traverseDepthFirst(String root){  
 var node=nodes.get(root);  
 if(node==null){  
 return;  
 }  
 Set<Node> visited1=new HashSet<>();  
 Stack<Node> stack=new Stack<>();  
 stack.push(node);  
  
 while (!stack.isEmpty()){  
 var current = stack.pop();  
 if(visited1.contains(current)){  
 continue;  
 }  
 System.out.println(current);  
 visited1.add(current);  
  
 for(var neighbour: adjeancyList.get(current)){  
 if(!visited1.contains(neighbour)){  
 stack.push(neighbour);  
 }  
 }  
 }  
 }  
  
 public void traverseBredthFirst(String root){  
 var node=nodes.get(root);  
 if(node==null){  
 return;  
 }  
 Set<Node> visited1=new HashSet<>();  
 Queue<Node> queue=new ArrayDeque<>();  
 queue.add(node);  
  
 while (!queue.isEmpty()){  
 var current = queue.remove();  
 if(visited1.contains(current)){  
 continue;  
 }  
 System.out.println(queue);  
 visited1.add(current);  
  
 for(var neighbour: adjeancyList.get(current)){  
 if(!visited1.contains(neighbour)){  
 queue.add(neighbour);  
 }  
 }  
 }  
 }  
 public List<String > topologicalSort(){  
 Stack<Node> stack=new Stack<>();  
 Set<Node> visited=new HashSet<>();  
 for(var node: nodes.values()){  
 topologicalSort(node,visited,stack);  
 }  
 List<String> sorted=new ArrayList<>();  
 while (!stack.empty()){  
 sorted.add(stack.pop().label);  
 }  
 return sorted;  
 }  
 public void topologicalSort(  
 Node node, Set<Node> visited, Stack<Node> stack  
 ){  
 if(visited.contains(node)) return;  
  
 visited.add(node);  
  
 for(var neighbour: adjeancyList.get(node)){  
 topologicalSort(neighbour,visited,stack);  
 }  
 stack.push(node);  
 }  
 public boolean hasCycle(){  
 Set<Node> all=new HashSet<>();  
 all.addAll((nodes.values()));  
  
 Set<Node> visiting=new HashSet<>();  
 Set<Node> visited=new HashSet<>();  
 while(!all.isEmpty()){  
 var current=all.iterator().next();  
 if(hasCycle(current,all,visiting,visited)) return true;  
 }  
 return false;  
 }  
 private boolean hasCycle(Node node, Set<Node> all,  
 Set<Node> visiting, Set<Node> visited){  
 all.remove(node);  
 visiting.add(node);  
  
 for(var neighbour : adjeancyList.get(node)){  
 if(visited.contains(neighbour))continue;  
  
 if(visiting.contains(neighbour)) return true;  
  
 if( hasCycle(neighbour,all,visiting,visited)) return true;  
 }  
 visiting.remove(node) ;  
 visited.add(node);  
 return false;  
 }