Contents

[**1.** **Alternative Sorting: -** Given an array of interger, print the array in such way that the first element is first maximum and second element is first minimum and so on. 26](#_Toc20051219)

[**2.** **Arrange Given Numbers Set1: -** arrange given numbers to form the biggest number. 27](#_Toc20051220)

[**3.** **Arrange Given Numbers Set2 -** arrange given numbers to form the biggest number. 27](#_Toc20051221)

[**4.** **Array Can Be Divided Into Pairs :-** check if an array can be divided into pairs whose sum is divisible by K given an array of intergers and a number K, write a function that returns true if given array can be divided into pairs such that sum of every pairs is divisible by K. 28](#_Toc20051222)

[**5.** **Array Contains Contiguous Integers Simple Solution: -** check if array contains contiguous integers with duplicates allowed simple solution 29](#_Toc20051223)

[**6.** **Array Contains Contiguous Integers Using HashTable: -** check if array contains contiguous integers with duplicates using hash table. 29](#_Toc20051224)

[**7.** **Array Contains Contiguous Integers Using Visited Array: -** check if array contains contiguous integers with duplicates using visited array. 29](#_Toc20051225)

[**8.** **Array Contains Duplicate Elements with in K Distance: -** check if a given array contains duplicate elements with in K distance from each other.Given an unsorted array that may contain duplicates. Also given a number K which is smaller than size of array.write a function that returns true if array contains cuplicates with in K distance. 30](#_Toc20051226)

[**9.** **Array Elements Are Consecutive Method1** 30](#_Toc20051227)

[**10.** **Array Elements Are Consecutive Method2** 31](#_Toc20051228)

[**11.** **Array Is Subset of another Array Using Hashing: -** Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Using Hashing 32](#_Toc20051229)

[**12.** **Array Is Subset of another Array: -** Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Simple solution 33](#_Toc20051230)

[**13.** **Array Is Subset of another Array Using Sorting and Binary Search: -** Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Using Sorting and Binary Search 33](#_Toc20051231)

[**14.** **Array Is Subset of another Array Using Sorting and Merging: -** Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Using Sorting and Merging 34](#_Toc20051232)

[**15.** **Asynchronous Callback :-** An asynchronous call do not block the program from the code execution.When the call returns from the event the call returns back to the callback function.So in the context of java we have to create a new thread invoke the callback method inside that thread.Callback may be invoked from a thread but is not requirement.A callback may also start a new thread thus making themselves asynchronous.**When the tasks are not dependent on each other and may take some time for execution we should use asynchronous callbacks.** For e.g. when you orded your food other people can also order their food in the restaurant.They don’t have to wait for your order to finish, if you’re downloading a song from internet. 35](#_Toc20051233)

[**16.** **BigDecimal class**:- The Big Decimal class provides operations on double numbers for arithmetic,scale handling,rounding,comparison,format conversion and hasing.It can handle very large and very small floating point numbers with great precision but compensating with the time complexity a bit .A BigDecimal consists of a random precision interger unscaled value and a 32-bit integer scale.If grater than or equal to zero, the scale is the number of digit to the right of the decimal point.If less than zero,the unscaled value of the number is mulyiplied by 10^(-scale). 36](#_Toc20051234)

[**17.** **Calendar After:-** java.util.Calender.after() is a method in Calendar class of java.util package.The method returns true if the time represented by this Calender is after the time represented by when object.If it is not the case,flase is returned. 36](#_Toc20051235)

[**18.** **Calendar Before:-** java.util.Calender.before() is a method in Calender class of java.util package.The method returns true if the time represented by this Calender is before the time represented by when object.If it is not the case , flase is returned. 36](#_Toc20051236)

[**19.** **Check for majority element in a sorted array using binary search** 37](#_Toc20051237)

[**20.** **Check for majority element in a sorted array using liner search** 37](#_Toc20051238)

[**21.** **Check for majority element in an array using HashMap** 38](#_Toc20051239)

[**22.** **Check for majority element in an array using Moore’s voting algo** 38](#_Toc20051240)

[**23.** **Check for pair in array [] with sum as x using Hash Map** 39](#_Toc20051241)

[**24.** **Check for pair in array [] with sum as x using Quick Sort** 39](#_Toc20051242)

[**25.** **Check if it is possible to transform one string to another: -** given two strings s1 and s2. Check if it is poosible to convert s1 and s2 by performing following operations. **1.** Make some lowercase letters uppercase. **2.** Delete all lowercare letters. 40](#_Toc20051243)

[**26.** **Check plus Perfect Number: -** given an ‘n’ digit number x, check if it is a plus perfect number or not.A number is plus perfect number. If it is equal to the sum of its degits raised to the nth power**.** 41](#_Toc20051244)

[**27.** **Check Valid Mobile Number: -** Program to check valid mobile number.Mobile number validation criteria: **1.** the first digit ahould contains number between 7 to 9. **2.** The rest 9 digit can contain any number between 0 to 9. **3.** The mobile number can have 11 digits also by including 0 at starting. **4.** The mobile number can be of 12 digit also by including 91 at the starting. 41](#_Toc20051245)

[**28.** **Deep Cloning** 42](#_Toc20051246)

[**29.** **Shallow Cloning** 42](#_Toc20051247)

[**30.** **Clone Using Serialization: -** program for deep copy using Serialization and Deserialization. 43](#_Toc20051248)

[**31.** **Closest number in array: -** given an array of sorted integers.We nee to find the closest value to the given number.Array may contain duplicate values and negative numbers. 43](#_Toc20051249)

[**32.** **Closest numbers from list: -** closest numbers from a list of unsorted integers. 44](#_Toc20051250)

[**33.** **Closest Pair: -** find the closed pair from two sorted arrays.given two sorted arrays and a number x.find the pair whose sum is closest to x and the pair has an element from each array. 45](#_Toc20051251)

[**34.** **Coefficient of Variation** 45](#_Toc20051252)

[**35.** **Cohesion: -** Cohesion refers all about how a single class is designed.Cohesion is the object oriented principle most closely associated with making sure that a class is designed with a single.well focused purpose.The more fpopuced a class is the cohensiveness of thet class is more.The advantage of high cohension is that such classes are much easier to maintain tan classes with low cohesion. Another benefits of high cohesion is that classes with a well-fouced purpose tend to be more reusable than other classes.for e.g. Suppose we have a class that multiply two numbers, but the same class creates a pop up window displaying the result.This is the example of low cohesive class because the window and the multiplication operation don’t have much in common.To make it high cohesive, we would have to create a class Display and a class multiply. The Display will call multiply’s method to get the result and display it. This way to develop a high cohesive solution. 46](#_Toc20051253)

[**36.** **Common Elements in Three Sorted Arrays: -** find common elements in three sorted arrays. 46](#_Toc20051254)

[**37.** **Compare Two Strings using library function**: - the method compareTo () is used for comparing teo strings lexicographically in java.Each character of both the atrings is converted into Unicode value for comparison. 47](#_Toc20051255)

[**38.** **Compare Two Strings without library function: -** java program to compare two strings lexicographically. 47](#_Toc20051256)

[**39.** **Comparing Enum Members: -** An enum can contain constants, member etc.enum can be defined as a group of named constant.There are two ways for making comparison of enum members. **1.** By using == operator **2.** By using equals () method. 47](#_Toc20051257)

[**40.** **Comparision of Double and Float: -** size of mantissa for float is 24 and 53 for double. 48](#_Toc20051258)

[**41.** **Concurrent Map Interface** 48](#_Toc20051259)

[**42.** **Convert First Character Uppercase in a Sentence: -** java program to convert first character uppercase in a sentence. 48](#_Toc20051260)

[**43.** **Convert string into palindrome: -** Convert the string into palindrom string by changing only one character.Given a string str. Check if it is poosible to convert the string into palindrom string by changing only one character. 49](#_Toc20051261)

[**44.** **Copy Demo Using Array List** 49](#_Toc20051262)

[**45.** **Copy Demo Using CopyOnWriteArrayList** 49](#_Toc20051263)

[**46.** **Copy on Write Array Set Demo** 50](#_Toc20051264)

[**47.** **Count characters: -** Count characters at same position as in English alphabets. 50](#_Toc20051265)

[**48.** **Count Occurrences of Each Character** 50](#_Toc20051266)

[**49.** **Count Window Distinct Efficient** 51](#_Toc20051267)

[**50.** **Count Window Distinct Simple: -** Count distinct elements in every window of size K. given an array size n and integer k, return the count of distinct numbers in all windowa of size K. 51](#_Toc20051268)

[**51.** **Count 1’s in Sorted Binary Array** 52](#_Toc20051269)

[**52.** **Count Down Latch Demo** 52](#_Toc20051270)

[**53.** **Count of different ways: -** count of different ways to express N as the sum of 1, 3, and 4. 53](#_Toc20051271)

[**54.** **Count smaller elements on right side:-** write a function to count number of smaller elemants on right of each element in an array.given an unsorted array arr[] of distinct integers,construct another array countSmaller[] such that countSmaller[i] conatins count of smaller elemants on right side of each element arr[i] in array. 53](#_Toc20051272)

[**55.** **Count Ways to Divide Circle:-** count ways to divide circle using N non-intersecting chords.given a number N, find the number of ways you can draw N chords in a circle with 2\*N points such that no 2 chords intersect. 54](#_Toc20051273)

[**56.** **Loose coupling: -** In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to over come from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling.Example : If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. The examples of loose coupling are Interface, JMS. 54](#_Toc20051274)

[**57.** **Tight coupling: -** In general, tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.Example: If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI (Remote Method Invocation). 55](#_Toc20051275)

[**58.** **Which is better tight coupling or loose coupling?** In general, Tight Coupling is bad in but most of the time, because it reduces flexibility and re-usability of code, it makes changes much more difficult, it impedes test ability etc. loose coupling is a better choice because A loosely coupled will help you when your application need to change or grow. If you design with loosely coupled architecture, only a few parts of the application should be affected when requirements change. 55](#_Toc20051276)

[**59.** **Difference between tight coupling and loose coupling:-** 55](#_Toc20051277)

[**60.** **Cutting Ropes: -** ropes left after every removal fo smallest. 55](#_Toc20051278)

[**61.** **Data Conversion using valueOf () method** 56](#_Toc20051279)

[**62.** **Deep Copy:-** 56](#_Toc20051280)

[**63.** **Shallow Copy:-** 57](#_Toc20051281)

[**64.** **Delegation: -** Delegation is simply passing a duty off to someone/something else. 57](#_Toc20051282)

[**65.** **Delete Element from Array Using One Traversals: -** move from right to left 58](#_Toc20051283)

[**66.** **Delete Element from Array Using TwoTraversals: -** search x in array and move left to right. 58](#_Toc20051284)

[**67.** **Deadlock: -** Deadlock is a situation when two threads are waiting for each other and the waiting is never ends. Here both threads cant completes their tasks. 58](#_Toc20051285)

[**68.** **Download Web Page** 59](#_Toc20051286)

[**69.** **Expand a String if range is given** 59](#_Toc20051287)

[**70.** **Equals IgnoreCase Method:-** the equalsIgnoreCase() method compares two strings irrespective of the case(lower or upper) of the string.This methos returns if the arguments is not null and it represents an equivalent string ignoring case, else false. 60](#_Toc20051288)

[**71.** **Equilibrium Index Efficient** 61](#_Toc20051289)

[**72.** **Equilibrium Index Simple:** - Equilibrium index of an array is an index such that sum of elements at lower indexes is equal to the sum of elements at higher indexes. 61](#_Toc20051290)

[**73.** **Externalizable Interface: -** Externalizable interface present in java.io, is used for Externalization which extends Serialization interface. It consistes of two methods which we have to override to write/read object into/from stream which are: - **1.** To read object from stream: - **void readExternal (ObjectInput in) 2.** To write object from stream: - **void writeExternal (ObjectOutput out)** 62](#_Toc20051291)

[**74.** **Fail Fast Iterators:-**Fail-Fast iterators immediately throw ConcurrentModificationException if there is structural modification of the collection.Structural modification means adding, removing or updating any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes 63](#_Toc20051292)

[**75.** **How Fail Fast Iterator works?** To know whether the collection is structurally modified or not, fail-fast iterators use an internal flag called modCount which is updated each time a collection is modified.Fail-fast iterators checks the modCount flag whenever it gets the next value (i.e. using next () method), and if it finds that the modCount has been modified after this iterator has been created, it throws ConcurrentModificationException. 63](#_Toc20051293)

[**76.** **Important points of fail-fast iterators:** These iterators throw ConcurrentModificationException if a collection is modified while iterating over it. They use original collection to traverse over the elements of the collection. These iterators don’t require extra memory. Ex: Iterators returned by ArrayList, Vector, HashMap. 63](#_Toc20051294)

[**77.** **Fail Safe Iterators: -** Fail-Safe iterators don’t throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that’s why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes 63](#_Toc20051295)

[**78.** **Fibonacci Series** 64](#_Toc20051296)

[**79.** **Fill Array With1:-** fill array with 1’s using minimum iterations of filling neighbors.given an array of 0s and 1s, in how many iterations the whole array can be filled with 1s if in a single iteration immediate neighbors of 1s can be filled. Note: if we cannot fill array with 1s, then print “-1”. 64](#_Toc20051297)

[**80.** **Final Example** 65](#_Toc20051298)

[**81.** **Find all the patterns Regular Expression Approach: -** find all the patterns of “1(0+) 1” in a given string using regular expression approach. 65](#_Toc20051299)

[**82.** **Find all the patterns Simple: -** count of occurrences of a “1(0+) 1” pattern in a string. 65](#_Toc20051300)

[**83.** **Find GCD of more than two (or array) numbers** 66](#_Toc20051301)

[**84.** **Program to find GCD or HCF of two numbers: -** GCD or HCF of two numbers is the largest number that divides both of them. 66](#_Toc20051302)

[**85.** **Given a list of tickets, find itinerary in order using the given list.** 66](#_Toc20051303)

[**86.** **LCM of given array elements** 67](#_Toc20051304)

[**87.** **Find missing elements of range using Hashing** 68](#_Toc20051305)

[**88.** **Given an array arr [0...n-1] of distinct elements and a range [low, high], find all numbers that are in range, but not in array. The missing elements should be printed in sorted order.** 68](#_Toc20051306)

[**89.** **Given a mathematical series as 3, 9, 21, 41, 71… For a given integer n, you have to find the nth number of this series.** 69](#_Toc20051307)

[**90.** **Given a dictionary that contains mapping of employee and his manager as a number of (employee, manager) pairs like below.** 69](#_Toc20051308)

[**91.** **Find an element using Space Efficient** 70](#_Toc20051309)

[**92.** **Find an element in array such that sum of left array is equal to sum of right array** 70](#_Toc20051310)

[**93.** **Find duplicates in O (n) time and O (1) extra space | Set 1 :-** let n be 7 and array be {1,2,3,1,3,6,6} the answaer should be 1,3 and 6. 71](#_Toc20051311)

[**94.** **Find the element that appears once in an array where every other element appears twice** 71](#_Toc20051312)

[**95.** **Find first and last positions Efficient** 71](#_Toc20051313)

[**96.** **Find first and last positions of an element in a sorted array** 72](#_Toc20051314)

[**97.** **Find Largest Subarray with 0 Sum Hashing** 72](#_Toc20051315)

[**98.** **Given an array of integers, find length of the largest subarray with sum equals to 0.** 73](#_Toc20051316)

[**99.** **Find four elements a, b, c and d in an array such that a+b = c+d: -** given an array of distinct intergers, find if there are two pairs (a, b) and (c, d) such that a+b = c+d, and a, b, cand d are distinct elements.If there are multiple answer, then print any of them. 73](#_Toc20051317)

[**100.** **Find position of an element in a sorted array of infinite numbers** 74](#_Toc20051318)

[**101.** **Find Second Most Frequent Character** 74](#_Toc20051319)

[**102.** **Given an array of positive and negative numbers, find if there is a subarray (of size at-least one) with 0 sum.** 75](#_Toc20051320)

[**103.** **Find the index of first 1 in an infinite sorted array of 0s and 1s :-** given an infinite sorted array consisting 0s and 1s.the problem is to find the index of first ‘1’ in that array.as the array is infinite,therefore it is guaranteed that number’1’ will be present in the array. 75](#_Toc20051321)

[**104.** **Find Triplet Use Sorting** 76](#_Toc20051322)

[**105.** **Find a triplet that sum to a given value: -** given an array and a value, find if there is a triplet in array whose sum is equal to the given value.if there is such a triplet present in array, then print the triplet and return true.else return false. For e.g. if the given array is {12, 3, 4, 1, 6, 9} and given sum is 24, then there is a triplet (12, 3, and 9) present in array whose sum is 24. 77](#_Toc20051323)

[**106.** **First Repeating Element in an Array** 77](#_Toc20051324)

[**107.** **Frequency of Sorted Array** 78](#_Toc20051325)

[**108.** **Frequency of Unsorted Array** 78](#_Toc20051326)

[**109.** **Generics Type Erasure** 78](#_Toc20051327)

[**110.** **Get list of files from FTPserver** 79](#_Toc20051328)

[**111.** **gOOGLE cASE of given sentence efficient** 79](#_Toc20051329)

[**112.** **Given a sentence, task is to rewrite in Google Case. It is a style of writing where we replace all lower case letters into upper case letters leaving the initial of all the words.** 80](#_Toc20051330)

[**113.** **Given a list of words with lower cases. Implement a function to find all Words that have the same unique character set.** 80](#_Toc20051331)

[**114.** **Group all occurrences of characters according to first appearance: -** given a string of lowercase, the task is to print the string in a manner such that a character comes first in string displays fisrt with all its accurrences in string. 81](#_Toc20051332)

[**115.** **Group multiple occurrence of array elements ordered by first occurrence: -** given an unsorted array with repetitions, the task is to group multiple occurrence of individual elements.the grouping should happen in a way that the order of first occurrences of all elements is maintained. 81](#_Toc20051333)

[**116.** **Hashcode Method in Java with HashCode** 82](#_Toc20051334)

[**117.** **Hashcode Method in Java without HashCode** 82](#_Toc20051335)

[**118.** **Hash Map Demo Using Concurrent Hash Map** 83](#_Toc20051336)

[**119.** **Hash Map Demo Using Hash Map** 83](#_Toc20051337)

[**120.** **Create a set and using .add () method we add the elements into the set** 83](#_Toc20051338)

[**121.** **Collections class consists of several methods that operate on collections.** 84](#_Toc20051339)

[**122.** **HashSet Using Constructor: -** in this method first we create an array then convert it to a list and then pass it to the HashSet constructor that accepts another collection. 84](#_Toc20051340)

[**123.** **Helper Class: -** Helper class is a java class which includes basic error handling. helper class contains functions that help in assisting the progem .this class intends to give quick implementation of basic functions such that programmers do not have to implement again and again.it is easy to access as all the member functions are static that is it can be accessed from anywhere.it implements the most commonly used functions excluding the functions that are already present in java library. 84](#_Toc20051341)

[**124.** **Initialization blocks and Constructors** 87](#_Toc20051342)

[**125.** **Instance and Static Variables** 87](#_Toc20051343)

[**126.** **Intersaction Using Hashing** 88](#_Toc20051344)

[**127.** **Intersection of two sorted arrays** 88](#_Toc20051345)

[**128.** **Intersection of two unsorted arrays using sorting and searching** 88](#_Toc20051346)

[**129.** **Iterating Over ArrayLists Using Foreach** 89](#_Toc20051347)

[**130.** **Iterating Over ArrayLists Using Iterator** 89](#_Toc20051348)

[**131.** **Iterating Over ArrayLists Using Loop** 89](#_Toc20051349)

[**132.** **Iterative Reverse Array** 90](#_Toc20051350)

[**133.** **Kth Largest Element Sorting** 90](#_Toc20051351)

[**134.** **Kth Smallest OR Largest Element Quick** 90](#_Toc20051352)

[**135.** **Kth Smallest OR Largest Element QuickSelect** 91](#_Toc20051353)

[**136.** **K’th Smallest/Largest Element in Unsorted Array:-** given an array and a number K where K is smaller than size of array, we need to find the Kth smallest element in the given array.it is given that all array elements are distinct. 92](#_Toc20051354)

[**137.** **Largest Rectangular Area in a Histogram: -** consider following histrogram with 7 bars of heights {6, 2, 5, 4, 5, 1, and 6}. The largest possible rectangle possible is 12. 92](#_Toc20051355)

[**138.** **Largest Number with one swap** 93](#_Toc20051356)

[**139.** **Largest subarray with equal number of 0s and 1s** 93](#_Toc20051357)

[**140.** **Largest Subarray Tricky** 94](#_Toc20051358)

[**141.** **Find last k digits in product of an array numbers** 94](#_Toc20051359)

[**142.** **Length of the largest subarray with contiguous elements** 95](#_Toc20051360)

[**143.** **Object level lock** 95](#_Toc20051361)

[**144.** **Class level lock** 96](#_Toc20051362)

[**145.** **Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.** 96](#_Toc20051363)

[**146.** **Longest increasing subarray** 97](#_Toc20051364)

[**147.** **Longest Increasing Subsequence: -** the longest increasing subsequence problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. for e.g. the length of LIS for {10,22,9,33,21,50,41,60,80} is 6 and LIS is {10,22,33,50,60,80} 98](#_Toc20051365)

[**148.** **Longest palindrome subsequence with O (n) space** 98](#_Toc20051366)

[**149.** **Longest Span Sum Auxiliary** 99](#_Toc20051367)

[**150.** **Longest Span with same Sum in two Binary arrays:-**given two binary arrays arr1 [] and arr2 [] of same size n. find length of the longest common span(i,j) where j>=I such that arr1[i]+arr1[i+1]+…+arr1[j]= arr2[i]+arr2[i+1]+…+arr2[j] 99](#_Toc20051368)

[**151.** **Longest Uncommon Subsequence BruteForce** 100](#_Toc20051369)

[**152.** **Longest Uncommon Subsequence Efficient** 101](#_Toc20051370)

[**153.** **Leaders in an Array Method1** 101](#_Toc20051371)

[**154.** **Leaders in an Array Method2** 101](#_Toc20051372)

[**155.** **Live Lock** 101](#_Toc20051373)

[**156.** **Longest Palindrome String** 102](#_Toc20051374)

[**157.** **Making Elements Distinct1** 103](#_Toc20051375)

[**158.** **Making Elements Distinct2** 103](#_Toc20051376)

[**159.** **Maximize sum of consecutive differences in a circular array** 104](#_Toc20051377)

[**160.** **Maximum difference of zeros and ones in binary string:-** given a binary string of 0s and 1s.the task is to find the length of substring which is having a maximum difference of number of 0s and number of 1s(no of 0s- no of 1s) In case of all print -1. 104](#_Toc20051378)

[**161.** **Maximum sum such that no two elements are adjacent** 105](#_Toc20051379)

[**162.** **Maximum Difference Efficient** 105](#_Toc20051380)

[**163.** **Maximum difference between two elements such that larger element appears after the smaller number: -** given an array arr [] of interger, find out the difference between any two elements such that larger element appears after the smaller number in arr []. For e.g. if arr is {2, 3, 10, 6, 4, 8, 1} then returned value should be 8 (diff between 10 and 2). If array is [7, 9, 5, 6, 3, 2] then returned value should be 2 (diff between 7 and 9). 105](#_Toc20051381)

[**164.** **Maximum Difference Tricky** 106](#_Toc20051382)

[**165.** **Maximum Elements of Array** 106](#_Toc20051383)

[**166.** **Maximum Index Difference Efficient** 107](#_Toc20051384)

[**167.** **Given an array arr[], find the maximum j – i such that arr[j] > arr[i]** 107](#_Toc20051385)

[**168.** **Maximum Length Bitonic Subarray | Set 1 (O(n) tine and O(n) space)** 108](#_Toc20051386)

[**169.** **Maximum Length Bitonic Subarray | Set 2 (O(n) time and O(1) Space)** 108](#_Toc20051387)

[**170.** **Maximum Occurring Character** 109](#_Toc20051388)

[**171.** **Find the maximum repeating number in O(n) time and O(1) extra space** 109](#_Toc20051389)

[**172.** **Maximum Sum** 109](#_Toc20051390)

[**173.** **Given two sorted arrays such the arrays may have some common elements: -** find the sum of the maximum sum path to reach from beginning of any array to end of any to the two arrays. 110](#_Toc20051391)

[**174.** **Find maximum sum possible equal sum of three stacks: -** given three stack of the positive numbers, the task is to find the possible equal maximum sum of the stacks with removal of top elements allowed.stacks are represented as array, and the first index of the array represent the top element of the stack. 111](#_Toc20051392)

[**175.** **Maximum sum subarray having sum less than or equal to given sum** 111](#_Toc20051393)

[**176.** **Maximum subarray sum in an array created after repeated concatenation** 112](#_Toc20051394)

[**177.** **Maximum Sum Subarray Using Dynamic Programming\_2** 112](#_Toc20051395)

[**178.** **Maximum Sum Subarray Using Dynamic Programming** 112](#_Toc20051396)

[**179.** **Maximum Sum Subarray Using Kadane Algo** 113](#_Toc20051397)

[**180.** **Maximum Sum Subarray Using Prefix Sum** 113](#_Toc20051398)

[**181.** **Largest sum subarray with at-least k numbers** 114](#_Toc20051399)

[**182.** **Maximum sum such that no two elements are adjacent** 114](#_Toc20051400)

[**183.** **Measure time Taken by a Function** 114](#_Toc20051401)

[**184.** **Memory Leaks** 115](#_Toc20051402)

[**185.** **Merge Array: - merge array n [] of size n into mPlusN [] of size m+n** 115](#_Toc20051403)

[**186.** **Minimum sum of multiplications of n numbers** 116](#_Toc20051404)

[**187.** **Minimum absolute difference of XOR values of two subarrays** 116](#_Toc20051405)

[**188.** **Find minimum difference between any two elements** 117](#_Toc20051406)

[**189.** **Minimum Difference between Two Elements Efficient** 117](#_Toc20051407)

[**190.** **Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted** 117](#_Toc20051408)

[**191.** **Find the smallest positive number missing from an unsorted array** 118](#_Toc20051409)

[**192.** **Missing Number Using List** 119](#_Toc20051410)

[**193.** **Missing Number Using Sum Formula** 119](#_Toc20051411)

[**194.** **Missing Number Using Total Number** 119](#_Toc20051412)

[**195.** **Missing Number Using XOR** 120](#_Toc20051413)

[**196.** **Move All Zeroes To End** 120](#_Toc20051414)

[**197.** **Find if neat arrangement of cups and shelves can be made:-** given three different types of cups(a[]) and saucers(b[],and n number of shelves,find if neat arrangement of cups and shelves can be made.Arrangement of the cups and saucers will be neat if it follows the below rules:- **1.** No shelf can contain both cups and saucers. **2.** There can be no more than 5 cups in any shelf. **3.** There can be no more than 10 saucers in any shelf. 120](#_Toc20051415)

[**198.** **Number Format Demo I18N** 121](#_Toc20051416)

[**199.** **Number of horizontal or vertical line segments to connect 3 points** 121](#_Toc20051417)

[**200.** **Find number of pairs in an array such that their XOR is 0** 122](#_Toc20051418)

[**201.** **Numbers with prime frequencies** 122](#_Toc20051419)

[**202.** **Number or Not** 123](#_Toc20051420)

[**203.** **Occurring Odd Number with Bitwise XOR** 123](#_Toc20051421)

[**204.** **Occurring Odd Number with Hashing** 124](#_Toc20051422)

[**205.** **Occurring Odd Number without Hashing** 124](#_Toc20051423)

[**206.** **Find the two numbers with odd occurrences in an unsorted array** 124](#_Toc20051424)

[**207.** **Optional Class | isPresent ():** - the isPresent () function in Optional class is used to evaluate whether the value if assigned to variable is present or not. 125](#_Toc20051425)

[**208.** **Overloading Of Thread Class Run Method** 125](#_Toc20051426)

[**209.** **Overloading Of Thread Class Start Method** 125](#_Toc20051427)

[**210.** **Overriding Default Method** 126](#_Toc20051428)

[**211.** **Overriding Private Method** 126](#_Toc20051429)

[**212.** **Overriding Public Method** 127](#_Toc20051430)

[**213.** **Given a sorted array and a number x, find the pair in array whose sum is closest to x** 127](#_Toc20051431)

[**214.** **Position of Element after Stable Sort: -** given an array of integers which may contain duplicate elements, an element of this array is given to us, we need to tell the final position of this element in the array, if a stable sort algorithm is applied. 127](#_Toc20051432)

[**215.** **Prefix Sum Array – Implementation and Applications in Competitive Programming** 128](#_Toc20051433)

[**216.** **Prime Number** 128](#_Toc20051434)

[**217.** **Print 2DArray foreach** 128](#_Toc20051435)

[**218.** **Print 2DArray Simple** 129](#_Toc20051436)

[**219.** **Print 2DArray toString** 129](#_Toc20051437)

[**220.** **Print all combinations of factors (Ways to factorize):-** write a program to print all the combinations of factors of given number n. 129](#_Toc20051438)

[**221.** **Efficient program to print all prime factors of a given number: -** if the input number is 12, then output should be “2 2 3” and if the input number is 315, then output should be “3 3 5 7”. 130](#_Toc20051439)

[**222.** **Print Intermediate value in an array** 130](#_Toc20051440)

[**223.** **Print number of words, vowels and frequency of each character** 131](#_Toc20051441)

[**224.** **Print All Distinct Elements of a given integer array:** - given an integer array, print all distinct elements in array.the given array may contain duplicates and the output should print every element only once. The given array is not sorted. 132](#_Toc20051442)

[**225.** **Print All Duplicates** 132](#_Toc20051443)

[**226.** **Print Frequencies of All Elements Method1** 132](#_Toc20051444)

[**227.** **Print Frequencies of All Elements Method2 by adding n to keep track of counts.** 133](#_Toc20051445)

[**228.** **Probability of reaching a point with 2 or 3 steps at a time: -** A person starts walking from position X=0, find the probability to each exactly on X=N if she can only take either 2 steps or 3 steps.Probability from step length 2 is given i.e. P, probability for step length 3 is 1-P. 133](#_Toc20051446)

[**229.** **Producer Consumer Problem** 134](#_Toc20051447)

[**230.** **Random nextInt** 135](#_Toc20051448)

[**231.** **Read Mobile Numbers File** 135](#_Toc20051449)

[**232.** **Rearrange an array such that ‘arr[j]’ becomes ‘i’ if ‘arr[i]’ is ‘j’:-** given an array of size n where all elements are in range from 0 to n-1, change contents of arr [] so that arr[i] =j is changed to arr[j] = i. 135](#_Toc20051450)

[**233.** **Rearrange an Array Set3** 136](#_Toc20051451)

[**234.** **Rearrange array in alternating positive & negative items with O (1) extra space** 137](#_Toc20051452)

[**235.** **Rearrange Array in Maximum Minimum Set1:-** given a sorted array of positive integers, rearrange the array alternately i.e. first element should be maximum value,second minimum value, third second mas, fourth second min and so on. 137](#_Toc20051453)

[**236.** **Rearrange Array In Maximum Minimum Set2** 138](#_Toc20051454)

[**237.** **Rearrange positive and negative numbers in O (n) time and O (1) extra space** 138](#_Toc20051455)

[**238.** **Remove Duplicates Using Hashing** 139](#_Toc20051456)

[**239.** **Remove Duplicates Using Sorting** 139](#_Toc20051457)

[**240.** **Remove the forbidden strings** 140](#_Toc20051458)

[**241.** **Itrative Reverse Array** 141](#_Toc20051459)

[**242.** **Recursive Reverse Array** 141](#_Toc20051460)

[**243.** **Reversal algorithm for right rotation of an array** 142](#_Toc20051461)

[**244.** **Reverse Number** 142](#_Toc20051462)

[**245.** **Rotational Array Method1** 142](#_Toc20051463)

[**246.** **Rotational Array Method2** 143](#_Toc20051464)

[**247.** **Rotational Array Reversal Algo** 144](#_Toc20051465)

[**248.** **Segregate 0s and 1s in an array using two indexes to traverse** 144](#_Toc20051466)

[**249.** **Segregate 0s and 1s in an array using Count 0s or 1s** 145](#_Toc20051467)

[**250.** **Segregate Even and Odd Nodes in Linked List** 145](#_Toc20051468)

[**251.** **Segregate Even and Odd numbers** 146](#_Toc20051469)

[**252.** **Semaphore Demo** 147](#_Toc20051470)

[**253.** **Serialization and Deserialization SerialVersionUID Example** 148](#_Toc20051471)

[**254.** **Given a set (HashSet or TreeSet) of strings in Java, convert it into an array of strings.** 148](#_Toc20051472)

[**255.** **Set to Array Stream** 148](#_Toc20051473)

[**256.** **Set to Array toArray Function** 149](#_Toc20051474)

[**257.** **Size of file on the Internet using Java: -** to get the size of file from server first you need to connect to the server using URL and HttpURLConnection class. To get the size of file we use getContentLength() method.As the size of file can be too large we use BigInteger class.You cannot use integer datatype as it can generate an error in case the size of file is greater than 2GB. 149](#_Toc20051475)

[**258.** **Smallest Missing Number** 149](#_Toc20051476)

[**259.** **Find the smallest positive integer value that cannot be represented as sum of any subset of a given array** 150](#_Toc20051477)

[**260.** **Smallest subarray with product divisible by k** 150](#_Toc20051478)

[**261.** **Smallest Subarray with Sum Efficient** 150](#_Toc20051479)

[**262.** **Smallest subarray with sum greater than a given value** 151](#_Toc20051480)

[**263.** **Find the smallest window in a string containing all characters of another string** 151](#_Toc20051481)

[**264.** **Given an array A [] consisting 0s, 1s and 2s, write a function that sorts A []. : -** the function should put all 0s first then all 1s and all 2s in last. 152](#_Toc20051482)

[**265.** **Sort an Array of 0s and 1s and 2s Simple Counting** 153](#_Toc20051483)

[**266.** **Sorted and Rotated Array** 154](#_Toc20051484)

[**267.** **Sorted Array Delete** 154](#_Toc20051485)

[**268.** **Sorted Array Insert** 155](#_Toc20051486)

[**269.** **Sorted Array Search** 155](#_Toc20051487)

[**270.** **Find a sorted subsequence of size 3 in linear time** 156](#_Toc20051488)

[**271.** **Sorting a 2D Array according to values in any given column in Java** 157](#_Toc20051489)

[**272.** **Sorting HashMap keys ArrayList** 157](#_Toc20051490)

[**273.** **Sorting HashMap keys constructor** 157](#_Toc20051491)

[**274.** **Sorting HashMap keys putAll** 158](#_Toc20051492)

[**275.** **Splitting Numeric String:** - given a numeric string (length <= 32), split it into two or more integers (if possible), such that: - **1.** Difference between current and previous number is 1. **2.** No number contains leading zeroes. If it is possible to separate a given numeric string then print “Possible” followed by the first number of the increasing sequence, else print “Not Possible”. 158](#_Toc20051493)

[**276.** **Standard Normal Distribution** 159](#_Toc20051494)

[**277.** **Star Pattern** 160](#_Toc20051495)

[**278.** **Static blank final variable** 160](#_Toc20051496)

[**279.** **String Interning Method** 161](#_Toc20051497)

[**280.** **String Literal Vs String Object** 161](#_Toc20051498)

[**281.** **Subarray With Given Sum Efficient** 161](#_Toc20051499)

[**282.** **Subarray With Given Sum Simple** 162](#_Toc20051500)

[**283.** **Find the sum up to n terms of the sequence: 5 + 55 + 555 + … up to n.** 162](#_Toc20051501)

[**284.** **Sum of Bitwise and of all pairs** 163](#_Toc20051502)

[**285.** **Sum of f (a[i], a[j]) over all pairs in an array of n integers** 163](#_Toc20051503)

[**286.** **Sum of the products of all possible Subsets** 163](#_Toc20051504)

[**287.** **Sum of XOR of all pairs in an array** 164](#_Toc20051505)

[**288.** **Super Example** 164](#_Toc20051506)

[**289.** **Write a Java program to take an input string and exchange the first and last word and revers the middle word.** 164](#_Toc20051507)

[**290.** **Symmetric difference of two sorted array** 165](#_Toc20051508)

[**291.** **Given an array of pairs, find all symmetric pairs in it: -** Two pairs (a, b) and (c, d) are said to be symmetric if c is equal to b and a, is equal to c and d. For e.g. (10, 20) and (20, 10) are symmetric 166](#_Toc20051509)

[**292.** **Synchronous Callback: -** the code execution will block or wait for the event before continuing. Until your event returns a response your program will not execute any further.so basically the callback performs all its work before returning to the call statement. Then problem with synchronous callback is that they appear to lag. Any process having multiple tasks where the tasks must be executed in sequence and doesn’t occupy much time should usr Synchronous Callbacks. For e.g. yor are in a movie queue for ticket you can’t get one until everyone in frony of yoe gets one. 166](#_Toc20051510)

[**293.** **Java Program to take Screenshots** 167](#_Toc20051511)

[**294.** **This Example** 167](#_Toc20051512)

[**295.** **Starvation: -** In Starvation, threads are also waiting for each other. But here waiting time is not infinite after some interval of time, waiting thread always gets the resources whatever is required to execute thread run() method. 167](#_Toc20051513)

[**296.** **String Palindrome Method1** 168](#_Toc20051514)

[**297.** **String Palindrome Method2** 168](#_Toc20051515)

[**298.** **String Palindrome Method3** 168](#_Toc20051516)

[**299.** **String Palindrome Recursion Method** 169](#_Toc20051517)

[**300.** **String Palindrome Rotational Method** 169](#_Toc20051518)

[**301.** **Differences between extending Thread class and implementing Runnable interface** 170](#_Toc20051519)

[**302.** **Thread Runnable** 170](#_Toc20051520)

[**303.** **Thread interrupt method1** 170](#_Toc20051521)

[**304.** **Thread interrupt method2** 171](#_Toc20051522)

[**305.** **Thread interrupt method3** 172](#_Toc20051523)

[**306.** **Thread isAlive Method: -** how can one thread know when another thread has ended? 172](#_Toc20051524)

[**307.** **Thread Join Method** 172](#_Toc20051525)

[**308.** **Thread without Join Method** 173](#_Toc20051526)

[**309.** **Three way partitioning of an array around a given range: -** given an array and a range [lowval, highval], partition the array around the range such that array is divided in three parts: - **1.** All elements smaller than lowval come first. **2.** All elements in range lowval to highval come next. **3.** All elements of three sets can appear in any order. The individual elements of three sets can appear in any end. 173](#_Toc20051527)

[**310.** **Thread Pool** 174](#_Toc20051528)

[**311.** **Two Elements Whose Sum Is Closest To Zero Method1** 175](#_Toc20051529)

[**312.** **Two elements whose sum is closest to zero using sorting** 175](#_Toc20051530)

[**313.** **Two Given Sets Are Disjoint Simple** 176](#_Toc20051531)

[**314.** **Two Given Sets Are Disjoint Use Hashing** 176](#_Toc20051532)

[**315.** **Two Given Sets Are Disjoint Use Sorting And Merging** 177](#_Toc20051533)

[**316.** **Two repeating elements in given array using array elements as index** 177](#_Toc20051534)

[**317.** **Two repeating elements in given array using count array** 178](#_Toc20051535)

[**318.** **Two repeating elements in given array using make two equations** 178](#_Toc20051536)

[**319.** **Two Repeating Elements In Given Array Method1** 178](#_Toc20051537)

[**320.** **Two repeating elements in given array using XOR** 179](#_Toc20051538)

[**321.** **Two Way Sorting Using Negative Multiplication** 179](#_Toc20051539)

[**322.** **Sort all even numbers in ascending order and then sort all odd numbers in descending order** 180](#_Toc20051540)

[**323.** **Unchecked Exceptions** 180](#_Toc20051541)

[**324.** **Union of two sorted arrays** 181](#_Toc20051542)

[**325.** **Union of two unsorted arrays using sorting and searching** 181](#_Toc20051543)

[**326.** **Union Using Hashing** 182](#_Toc20051544)

[**327.** **Unsorted Array Delete** 182](#_Toc20051545)

[**328.** **Unsorted Array Insert** 183](#_Toc20051546)

[**329.** **Unsorted Array Search** 183](#_Toc20051547)

[**330.** **Verification In Java** 184](#_Toc20051548)

[**331.** **Wrapper classes** 184](#_Toc20051549)

[**332.** **Sort Employee list on basis of name in ascending order by implementing Comparable interface and overriding its compareTo method in java** 185](#_Toc20051550)

[**333.** **Sort Employee list on basis of name in ascending order by implementing Comparator interface and overriding its compare method in java** 186](#_Toc20051551)

[**334.** **Sort Employee list on basis of name and id in ascending order by implementing Comparator interface and overriding its compare method in java** 186](#_Toc20051552)

[**335.** **Sort Employee list on basis of name and id in ascending order by implementing Comparator interface in inner and static nested class and overriding its compare method in java** 187](#_Toc20051553)

[**336.** **Arithmetic Exception** 188](#_Toc20051554)

[**337.** **ArrayIndexOutOfBounds Exception: -** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to size of the array. 188](#_Toc20051555)

[**338.** **Assertion Error: -** Exception rises explicitly by the programmer or by API developer to indicate that assert statement fails. 188](#_Toc20051556)

[**339.** **Checked Exceptions** 188](#_Toc20051557)

[**340.** **Class Cast Exception** 189](#_Toc20051558)

[**341.** **ClassNotFound Exception: -** ClassNotFoundException occurs when you try to load a class at runtime using Class.forName () or loadClass () methods and requested classes are not found in Classpath. Most of the time this exception will occur when you try to run application without updating Classpath with JAR files. This exception is **checked exception** derived from java.lang.Exception class and you need to provide explicit handling for it.This exception also occurs when you have two classes loaders and if a ClassLoader tries to access a class which is loaded by another classLoader in java. Java classLoader is a part of java runtime environment that dynamically loads java classes in JVM. The java runtime system does not need to know about files and files syatem because of classloader. 189](#_Toc20051559)

[**342.** **ClassNotFoundException Vs NoClassDefFoundError** 189](#_Toc20051560)

[**343.** **Concurrent Modification Exception** 189](#_Toc20051561)

[**344.** **Exception InInitializer Error: -** Exception occurs whenever while executing static variable assignment and static block if any Exception occurs. 190](#_Toc20051562)

[**345.** **FileNotFoundException: -** this exception is raised when a file is not accessible or does not open. 190](#_Toc20051563)

[**346.** **IllegalArgument Exception: -** this occurs when the argument passed is not positive. The exception occurs explicitly either by programmer or by API developer to indicate that method has been invoked with illegal argument. 190](#_Toc20051564)

[**347.** **IndexOutOfBound Exception** 191](#_Toc20051565)

[**348.** **InterruptedException: -** it is thrown when a thread is waiting, sleeping or doing some processing and it is interrupted. 191](#_Toc20051566)

[**349.** **IOException: -** it is thrown when an io/op operation failed or interrupted. 191](#_Toc20051567)

[**350.** **NoClassDefFoundError: -** NoClassDefFoundError occur when class present during complie time and program was compiled and linked successfully but class was not present during runtime.it is error which is derived from LinkageError. LinkageError occurs when a class has some dependencies on another class and letter class changes after compilation of former class. NoClassDefFoundError is the result of implicit loading of class because of calling a method or accessing a variable from that class. This error is more difficult to debug and find the reason why this error occurred, so in this case you should always check the classes which are dependent on this class. 192](#_Toc20051568)

[**351.** **NoSuchElement Exception** 192](#_Toc20051569)

[**352.** **NoSuchMethodException: -** it is thrown when accessing a method which is not found. 192](#_Toc20051570)

[**353.** **NullPointerException: -** this exception is raised when referring to the members of null object.Null represents nothing. 192](#_Toc20051571)

[**354.** **NumberFormatException: -** this exception is raised when a method could not convert a string into a numeric format. 193](#_Toc20051572)

[**355.** **StackOverflow Error** 193](#_Toc20051573)

[**356.** **StringIndexOutOfBoundsException: -** it is thrown by String class methos to indicate that an index is either negative than the size of the string. 193](#_Toc20051574)

[**357.** **TreeMap putAll NullPointerException: -** occurs if the map contains a null key or it is null in putAll (). 193](#_Toc20051575)

[**358.** **Best ranked person in given list, or can be used to assign winner in game in which person with lowest time to finish a task wins.** 194](#_Toc20051576)

[**359.** **BrainFuck Interpreter** 194](#_Toc20051577)

[**360.** **Implemented by Cricket data to communicate with observers OR Java program to demonstrate working of onserver pattern OR Building Cricket Score APP** 195](#_Toc20051578)

[**361.** **Game of Brussels sprouts: -** given n number of spots, and two players. Players need to connect any two apots without intersecting any of the drawn line, the player who cannot give a move, loses. Determine who will win player1 or player2. 196](#_Toc20051579)

[**362.** **Three N plus One (3n+1) Problem** 197](#_Toc20051580)

[**363.** **Circular Queue Example: -** implementation with generics in java, insert and remove element from Circuler Queues in java. 197](#_Toc20051581)

[**364.** **Priority Queue1 with Default Comaparator** 199](#_Toc20051582)

[**365.** **Priority Queue2 with customized comaparator** 199](#_Toc20051583)

[**366.** **Cyclic Barrier Custom** 200](#_Toc20051584)

[**367.** **Read Lock Test: -** Program to demonstrate usage of radlock () method of ReadWriteLock in java. Void readLock () more than one threads can acquire readlock at a time, provided no other thread is acquiring writeLock at same time in java. 201](#_Toc20051585)

[**368.** **Write Lock Test: -** Program to demonstrate usage of WriteLock () method of ReadWriteLock in java. Void writeLock () only one threads can acquire writeLock at a time. Means writeLocK can only be optained if no other thread is acquiring read or writelock at a time in java. 201](#_Toc20051586)

[**369.** **Reentrant Lock Custom Test** 202](#_Toc20051587)

[**370.** **Semaphore Custom Consumer Producer** 203](#_Toc20051588)

[**371.** **Semaphore Custom Test** 205](#_Toc20051589)

[**372.** **Thread Death Demo: -** calling stop method on thread throws ThreadDeath error in java. 206](#_Toc20051590)

[**373.** **Add Element at Specific Position** 207](#_Toc20051591)

[**374.** **Clone of Linked List** 207](#_Toc20051592)

[**375.** **Doubly Linked List at First** 207](#_Toc20051593)

[**376.** **Element Exist in Linked List: -** given an element, how do you find out whether that element exist in the linkedlist or not. If it exist retrieve the position of that element. 209](#_Toc20051594)

[**377.** **Find Circular Singly Linked List Intersection Point Example** 209](#_Toc20051595)

[**378.** **Generic Single Linked List Insert Delete First Example** 210](#_Toc20051596)

[**379.** **Get position of last occurrence of a given element in a linkedlist.** 211](#_Toc20051597)

[**380.** **Get Number of Elements** 211](#_Toc20051598)

[**381.** **InOrder Traversal of BinaryTree** 211](#_Toc20051599)

[**382.** **PreOrder Traversal of BinaryTree** 212](#_Toc20051600)

[**383.** **PostOrder Traversal of BinaryTree** 213](#_Toc20051601)

[**384.** **Insert an Element at the head and tail of a linkedlist** 214](#_Toc20051602)

[**385.** **Join ArrayList at End of the linkedlist** 214](#_Toc20051603)

[**386.** **LinkedList as Queue** 214](#_Toc20051604)

[**387.** **LinkedList as Stack** 214](#_Toc20051605)

[**388.** **LinkedList Circular Calculate Nodes** 215](#_Toc20051606)

[**389.** **Traverse the element of LinkedList in Reverse direction** 216](#_Toc20051607)

[**390.** **LinkedList Intersection Point Example** 216](#_Toc20051608)

[**391.** **Queue Implementation Using LinkedList** 217](#_Toc20051609)

[**392.** **Remove first and last occurrence of s given element in the linkedlist** 218](#_Toc20051610)

[**393.** **Remove All Elements from linkedlist** 219](#_Toc20051611)

[**394.** **Remove Elements of linkedlist from both the ends** 219](#_Toc20051612)

[**395.** **Replace Element at specified position of linkedlist with the given element.** 219](#_Toc20051613)

[**396.** **Retrieve but Not Remove the elements of linkedlist from both the ends.** 220](#_Toc20051614)

[**397.** **Retrieve and Remove and only retrieve an element from specified position of a linkedlist** 220](#_Toc20051615)

[**398.** **Reversing Singly LinkedList** 220](#_Toc20051616)

[**399.** **Singly LinkedList at First** 221](#_Toc20051617)

[**400.** **Singly LinkedList at Last** 222](#_Toc20051618)

[**401.** **Singly LinkedList Circular or Not** 224](#_Toc20051619)

[**402.** **Singly LinkedList Delete Specific Node** 225](#_Toc20051620)

[**403.** **Singly LinkedList Generic Example** 226](#_Toc20051621)

[**404.** **Singly LinkedList Insert Node In Between** 227](#_Toc20051622)

[**405.** **Sorted Doubly LinkedList** 228](#_Toc20051623)

[**406.** **Stack Example Generic** 230](#_Toc20051624)

[**407.** **Stack Implementation Using LinkedList** 231](#_Toc20051625)

[**408.** **ArrayList Custom App** 233](#_Toc20051626)

[**409.** **ArrayList Synchronization Fails Example** 234](#_Toc20051627)

[**410.** **ArrayList Synchronization Succeeds Example** 234](#_Toc20051628)

[**411.** **Arrays Sort: -** Program to understand comparatoe of superclass can be used by subclasses 235](#_Toc20051629)

[**412.** **Concurrent HashMap Example: -** program to use put, get, containsKey, containsValue, remove, size methods 236](#_Toc20051630)

[**413.** **Concurrent Skip List Map Example: -** program to use put, get, containsKey, containsValue, remove, size methods 236](#_Toc20051631)

[**414.** **Concurrent Skip List Map Example to make map unmodifiable** 237](#_Toc20051632)

[**415.** **CopyOnWriteArrayList Synchronization Succeeds Example** 237](#_Toc20051633)

[**416.** **HashCode\_and\_equal\_1:-** create a map which contains address of each employee and uses employee object as a key. Store address of some employees in this map. Now create a method that accepts map and employee object as parameters and return address of this employee. **Approach 1 (without overriding hashcode and equals method).** 238](#_Toc20051634)

[**417.** **HashCode\_and\_equal\_2 Approach2 (Overriding hashcode and equals method)** 238](#_Toc20051635)

[**418.** **HashMap Custom App** 239](#_Toc20051636)

[**419.** **HashMap Test: -** program to create method that provides functionality smiliar to putIfAbsent () method of ConcurrentHashMap and to be used with HashMap. 241](#_Toc20051637)

[**420.** **HashMap to ArrayList** 242](#_Toc20051638)

[**421.** **HashSet Custom App** 242](#_Toc20051639)

[**422.** **Hashtable Test1 making map unmodifiable using Collections.unmodifiableMap** 245](#_Toc20051640)

[**423.** **Hashtable Test2 Iterator on KeySet, values and entrySet is fail-fast.** 245](#_Toc20051641)

[**424.** **Hashtable Test3:-** Iterate on keys by obtaining keyset, Iterate on values by obtaining values, Iterate on entry by obtaining entrySet 246](#_Toc20051642)

[**425.** **Hashtable Test4 put, get, containsKey, conatinsValue, remove ans size method** 247](#_Toc20051643)

[**426.** **Linked HashMap Custom App display values corresponding to keys** 247](#_Toc20051644)

[**427.** **LinkedHashMap Test1:-** Iterate on keys by obtaining KeySet, iterate on values by obtaining values and iterate on entry by obtaining entrySet 251](#_Toc20051645)

[**428.** **LinkedHashMap Test2:-** iterate on keyset, values and entryset is fail-fast or fail-safe 251](#_Toc20051646)

[**429.** **LinkedHashMap Test3 put, get, conatinskey, conatinsvalue, remove and size method** 252](#_Toc20051647)

[**430.** **LinkedHashMap Example making it unmodifiable** 252](#_Toc20051648)

[**431.** **LinkedHashSet Custom App** 252](#_Toc20051649)

[**432.** **Linked Hash Set Example isEmpty, size and clear method** 257](#_Toc20051650)

[**433.** **Linked HashSet Iterator Example iterate using iterator, enumeration and enhanced for loop** 257](#_Toc20051651)

[**434.** **Linked Hash Set Iterator Fail Fast Example using iterator, enumeration and enhanced for loop** 257](#_Toc20051652)

[**435.** **Linked Hash Set Synchronizing Example using Collections.synchronizedSet** 258](#_Toc20051653)

[**436.** **Linked Hash Set Example making set unmodifiable using Collections.unmodifiableSet** 258](#_Toc20051654)

[**437.** **Removing Duplicate Elements Using HashSet** 258](#_Toc20051655)

[**438.** **Removing Duplicate Elements Using LinkedHashSet** 259](#_Toc20051656)

[**439.** **Reverse ArrayList** 259](#_Toc20051657)

[**440.** **Reverse LinkedList** 259](#_Toc20051658)

[**441.** **SortMap by Key** 260](#_Toc20051659)

[**442.** **SortMap by Key descending** 260](#_Toc20051660)

[**443.** **SortMap by Key Ascending** 260](#_Toc20051661)

[**444.** **SortMap by Value Ascending by implementing Comparator interface and overriding its compare () method** 261](#_Toc20051662)

[**445.** **Program to sort set using treeMap’s constructor** 261](#_Toc20051663)

[**446.** **Program to sort set in Asecnding order by implementing TreeSet and implementing Comaparator interface** 261](#_Toc20051664)

[**447.** **Synchronized HashMap** 262](#_Toc20051665)

[**448.** **Synchronized HashSet** 262](#_Toc20051666)

[**449.** **Synchronized List** 263](#_Toc20051667)

[**450.** **TreeMap Example using put, get, conatinskey, conatinsvalue, remove and size method.** 263](#_Toc20051668)

[**451.** **TreeMap Fail Fast Example** 264](#_Toc20051669)

[**452.** **TreeMap Iterator Example** 264](#_Toc20051670)

[**453.** **TreeMap Example to make it unmodifiable** 265](#_Toc20051671)

[**454.** **Vector Custom Example** 265](#_Toc20051672)

[**455.** **Arithmetic Progression: -** count of AP subsequences in an array 266](#_Toc20051673)

[**456.** **Bellman Ford Algorithm:** - works for Graphs with negative weight edges. And suites well for distributed systems 267](#_Toc20051674)

[**457.** **Bitmasking: -** count ways to assign unique cap to every person. There 100 different types of caps each having a unique id from 1 to 100. Also there ‘n’ persons each having a collection of variable number of caps. One day all of these persons decide to go in a party wearing a cap but to look unique they decided that none them will wear the same type of caps. So count the total number of arrangements or ways such that none of them is wearing same type of cap. 268](#_Toc20051675)

[**458.** **BoxStacking:-** you are given a set of n types of rectangular-3d boxes, where the i^th box has height h(i) , width w(i) and depth d(i) ( all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2d base of the lower box are each strictly larger than those of the 2d base of higer box. 269](#_Toc20051676)

[**459.** **Clustering Partitioning an Array such that sum of square differences is minimum** 270](#_Toc20051677)

[**460.** **Clustering Partitioning Array1 using dynamic programming** 270](#_Toc20051678)

[**461.** **Compute Sum of Digits in all numbers from 1 to n** 271](#_Toc20051679)

[**462.** **Compute Sum of Digits1** 271](#_Toc20051680)

[**463.** **Count number of binary strings without consecutive 1’s. Given a positive integer N, count all possible distinct binary strings of length N such that there are no consecutive 1’s** 272](#_Toc20051681)

[**464.** **Count of N Digit Numbers whose sum of digits equals to given sum** 272](#_Toc20051682)

[**465.** **Count of N Digit Numbers1 whose sum of digits equals to given sum using dynamic programming** 272](#_Toc20051683)

[**466.** **Count All Palindromic Subsequence in a given string. Find how many palindromic subsequence (need not necessarily br distinct) can be formed in a given string. Note that the empty string is not considered as a polindrom.** 273](#_Toc20051684)

[**467.** **Count All Possible Walks from a source to a distination with exactly K edges using simple solution.** 273](#_Toc20051685)

[**468.** **Count All Possible Walks1 using dynamic pogramming** 274](#_Toc20051686)

[**469.** **Given a teo strings S and T, find count of distinct occurrences of T in S as a subsequence** 274](#_Toc20051687)

[**470.** **Recursive java program to find n-th Rencontres number (counting partial derangements)** 275](#_Toc20051688)

[**471.** **Recursive java program to find n-th Rencontres number using dynamic programming** 275](#_Toc20051689)

[**472.** **Count Number of Ways to reach destination in a Maze.** Given a maze with obstacles, count number of paths to reach rightmost-bottommost cell from tomost-leftmost cell. A cell in given maze has value -1 if it is a blockage or dead end, else 0. From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only 276](#_Toc20051690)

[**473.** **Count Number of Ways1 to jump to reach end:** - given an array of numbers where each element represents the max number of jumps that can be made forward from that element. For each array element, count number of ways jumps can be made from that element to reach the end of the array. If an element is 0, then move cannot be made through that element. The element that cannot reach to the end should have a count “-1”. 277](#_Toc20051691)

[**474.** **Count Possible Decodings of a given digit sequence.** Let 1 represent ‘A’ , 2 represent ‘B’ etc. given a digit sequence , count the number of possible decoding of the given digit sequence. 277](#_Toc20051692)

[**475.** **Count Possible Ways to construct buildings: -** given an input number of sections and each section has 2 plots on either sides of the road. Find all possible ways to construct building in the lopts such that there is a space between any 2 buildings. 278](#_Toc20051693)

[**476.** **Egg Dropping: -** the following is the description of the instance of this famous puzzle involving n=2 egg and a building with k = 36 floors. 278](#_Toc20051694)

[**477.** **Find All Distinct Subset sum of an array: -** given a set of intergers, find distinct sum that can be generated from the susets of a given sets and print them in an increasing order. It is given that sum of array elements is small. 279](#_Toc20051695)

[**478.** **Find All Distinct Subset1 using dynamic programming** 279](#_Toc20051696)

[**479.** **Find Maximum Dot Product of two arrays with insertion of 0’s.** Given two arrays of positive integers of size m and n where m>n. we need to maximize the dot product by inserting 0’s in the second array but we cannot disturb the order of elements. 280](#_Toc20051697)

[**480.** **Find Minimum Sum such that one of every three consecutive elements is taken given an array of n non-negative numbers, the task is to find the minimum sum of elements such that at least one element is picked out of every 3 consecutive elements in the array.** 280](#_Toc20051698)

[**481.** **Find Number Endless Points: -** give a binary NxN matrix, we need to find the total number of matrix positions from which there is an endless path. 280](#_Toc20051699)

[**482.** **Floyd Warshall Algorithm is for solving the all pairs shortest path problem.** The problem is to find shortest distances between every pair of vertices in a given dege weighted directed graph. 281](#_Toc20051700)

[**483.** **Friends Pairing Problem: -** given n friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out total number of ways in which friends can remain single or can be paired up. 282](#_Toc20051701)

[**484.** **Gold Mine Problem: -** given a gold mine of n\*m dimensions. Each field in this mine conatins a positive integer which is the amount of gold in tons. Initially the miner is at first column but can be at any row. He can move only (right->, right up/, right down\) that is from a given cell, the miner can move to cell diagonally up towards the right or diagonally down towards the right. Find the maximum amount of gold he can collect. 282](#_Toc20051702)

[**485.** **Knapsack Problem: -** given wights and values of n items, put these items in a knapsack of capacity W to get the maximum toral value in the knapsack. Overlapping subproblems. 283](#_Toc20051703)

[**486.** **Knapsack Problem1 dynamic programming based implementation** 283](#_Toc20051704)

[**487.** **Knight Remain Chessboard: -** probability of knight to remain in the chessboard.given an NxN chessboard and a knight at position(x, y). The knoght has to take exactly K steps, where at each step it chooses any of the 8 directions uniformly at random.what is the probability that the knight remains in the chessboard after taking K steps, with the condition that it can’t enter the board again once it leaves it. 284](#_Toc20051705)

[**488.** **Largest Independent Set: -** given a binary tree, find size of the largest independent set in it. A suset of all tree nodes is an independent set if there is no edge between any two nodes of the subset. 285](#_Toc20051706)

[**489.** **Length of Longest Substring without repeating characters.** Given a sting, find the length of the longest substring without repeating characters. 285](#_Toc20051707)

[**490.** **Longest Alternating Subsequence** 286](#_Toc20051708)

[**491.** **Longest Arithmetic Progression: -** given a set of numbers, find the length of the longest arithmetic progession in it. 287](#_Toc20051709)

[**492.** **Longest Bitonic Subsequence: -** given an array arr [0…n-1] containing n positive integers, a subsequence of arr [] is called Bitnonic if it is first increasing, then decreasing. Write a function that takes an array as argument and return the length of the longest bitonic subsequence. 287](#_Toc20051710)

[**493.** **Longest Common Increasing Subsequence: -** given two array, find the length of the longest common increasing subsequence and print one of such sequences. 288](#_Toc20051711)

[**494.** **Longest Common Increasing Subsequence1** 288](#_Toc20051712)

[**495.** **Longest Common Subsequence of three strings:** - given 3 strings of all having length<100, the task is to find the longest common subsequence in all the three given sequences. 289](#_Toc20051713)

[**496.** **Longest Consecutive Path:-** find length of the longest consecutive path from a given starting character.given a matrix of characters.find length of the longest path from a given character,such that all characters in the path are consecutive to each other,i.e.,every character in the path is next to previous in alphabetical order. It is allowed to move in all 8 directions from a cell. 290](#_Toc20051714)

[**497.** **Longest Even Length Substring** :- such that sum of first and second half is same.given a string’str’ of digits, find length of the longest substring of ‘str’ , such that the length of the substring is 2k digit and sum of left k digits is equal to the sum of the right k digits. 291](#_Toc20051715)

[**498.** **Longest Even Length Substring1 using dynamic programming [O (n2) and O (n2) extra space]** 291](#_Toc20051716)

[**499.** **Longest Even Length Substring2 using dynamic programming [A O (n2) and O (n) extra space]** 292](#_Toc20051717)

[**500.** **Longest Palindromic Subsequence: -** given a sequence, find the length of the longest palindromic subsequence in it. 292](#_Toc20051718)

[**501.** **Longest Path Matrix** 293](#_Toc20051719)

[**502.** **Longest Repeating Subsequence** 293](#_Toc20051720)

[**503.** **Longest Subsequence** 294](#_Toc20051721)

[**504.** **Longest Zig Zag Subsequence** 294](#_Toc20051722)

[**505.** **Matrix Chain Multiplication** 295](#_Toc20051723)

[**506.** **Matrix Chain Multiplication1** 295](#_Toc20051724)

[**507.** **Maximize Sum Differences** 296](#_Toc20051725)

[**508.** **Maximum Path Sum** 296](#_Toc20051726)

[**509.** **Maximum Path Sum Position** 297](#_Toc20051727)

[**510.** **Maximum Path Sum Triangle** 297](#_Toc20051728)

[**512.** **Maximum Product Cutting1** 298](#_Toc20051729)

[**513.** **Maximum Profit: -** Maximum profit by buying and selling a share at most twice.In a daily share trading, a buyer buys shares in the morning and sells it on same day. If the trader is allowed to make at most 2 transactions in a day, whereas second transaction can only start after first one is complete (Sell->buy->sell->buy).Given stock prices throughout day, find out maximum profit that a share trader could have made. 298](#_Toc20051730)

[**514.** **Maximum Subsequence Sum: -** Maximum subsequence sum such that no three are consecutive. Given a sequence of positive numbers, find the maximum sum that can be formed which has no three consecutive elements present. 299](#_Toc20051731)

[**515.** **Maximum Sum Alternating Subsequence: -** Given an array, the task is to find sum of maximum sum alternating subsequence starting with first elements. Here alternating sebsequence means decreasing, then increasing then decreasing. For e.g. 10, 5,14,3 is alternating sequence. 299](#_Toc20051732)

[**516.** **Maximum Sum BiTonic SubSequence: -** Given an array of intergers. A subsequence of arr [] is called Bitonic if it is first increasing, then decreasing. 300](#_Toc20051733)

[**517.** **Maximum Sum of Pairs** 301](#_Toc20051734)

[**518.** **Maximum Sum of Pairs1** 301](#_Toc20051735)

[**519.** **Maximum Sum of Path** 301](#_Toc20051736)

[**520.** **Maximum Sum Rectangle** 302](#_Toc20051737)

[**521.** **Maximum Weight Path** 303](#_Toc20051738)

[**522.** **Minimum number of deletions to make a string palindrome** 304](#_Toc20051739)

[**523.** **Find minimum adjustment cost of an array** 304](#_Toc20051740)

[**524.** **Find the minimum cost to reach destination using a train** 305](#_Toc20051741)

[**525.** **Minimum Cost Reach Destination1** 305](#_Toc20051742)

[**526.** **Minimum cost to fill given weight in a bag** 306](#_Toc20051743)

[**527.** **Minimum Cost to Make Two Strings Identical** 307](#_Toc20051744)

[**528.** **Minimum Initial Points** 307](#_Toc20051745)

[**529.** **Minimum Insertions Palindrome** 308](#_Toc20051746)

[**530.** **Minimum Insertions Palindrome1** 308](#_Toc20051747)

[**531.** **Minimum Insertions Palindrome2** 308](#_Toc20051748)

[**532.** **Minimum insertions to sort an array** 309](#_Toc20051749)

[**533.** **Find minimum number of coins that make a given value** 309](#_Toc20051750)

[**534.** **Minimum Number of Coins1** 310](#_Toc20051751)

[**535.** **Minimum number of deletions to make a sorted sequence** 310](#_Toc20051752)

[**536.** **Minimum number of squares whose sum equals to given number n** 311](#_Toc20051753)

[**537.** **Minimum Number of Squares1** 311](#_Toc20051754)

[**538.** **Minimum steps to delete a string after repeated deletion of palindrome substrings** 312](#_Toc20051755)

[**539.** **Minimum Sum Path in 3D Array** 312](#_Toc20051756)

[**540.** **Minimum sum subsequence such that at least one of every four consecutive elements is picked** 313](#_Toc20051757)

[**541.** **Minimum time to finish tasks without skipping two consecutive** 313](#_Toc20051758)

[**542.** **Number of permutation with K inversions** 314](#_Toc20051759)

[**543.** **Number of subsequences of the form a^i b^j c^k** 314](#_Toc20051760)

[**544.** **Find number of times a string occurs as a subsequence in given string** 315](#_Toc20051761)

[**545.** **Optimal Binary Search Tree** 315](#_Toc20051762)

[**546.** **Optimal Binary Search Tree1** 316](#_Toc20051763)

[**547.** **Optimal Strategy Game** 316](#_Toc20051764)

[**548.** **Palindrome Partitioning** 317](#_Toc20051765)

[**549.** **Palindrome Partitioning1** 318](#_Toc20051766)

[**550.** **Partition Problem** 318](#_Toc20051767)

[**551.** **Partition Problem1** 319](#_Toc20051768)

[**552.** **Partition Set into Two Subsets** 319](#_Toc20051769)

[**553.** **Partition Set into Two Subsets1** 320](#_Toc20051770)

[**554.** **Path with Maximum Average Value** 320](#_Toc20051771)

[**555.** **Perfect Sum Problem** 321](#_Toc20051772)

[**556.** **Permutation** 322](#_Toc20051773)

[**557.** **Permutation1** 322](#_Toc20051774)

[**558.** **Print Maximum Number** 323](#_Toc20051775)

[**559.** **Print Maximum Number1** 323](#_Toc20051776)

[**560.** **Probability of getting at least K heads in N tosses of Coins** 324](#_Toc20051777)

[**561.** **Recursively break a number in 3 parts to get maximum sum** 324](#_Toc20051778)

[**562.** **Recursively break a number in 3 parts to get maximum sum using Dynamic programming. Time Complexity: O (n) Auxiliary Space: O (n)** 324](#_Toc20051779)

[**563.** **Given two strings str1 and str2, find the shortest string that has both str1 and str2 as subsequences.** 325](#_Toc20051780)

[**564.** **Shortest Common Supersequence1** 325](#_Toc20051781)

[**565.** **Shortest Common Supersequence2** 326](#_Toc20051782)

[**566.** **Shortest path with exactly k edges in a directed and weighted graph a simple solution** 326](#_Toc20051783)

[**567.** **Shortest path with exactly k edges in a directed and weighted graph using Dynamic Programming.** 326](#_Toc20051784)

[**568.** **Subset with sum divisible by m: -** Given a set of non-negative distinct integers, and a value m, determine if there is a subset of the given set with sum divisible by m. 327](#_Toc20051785)

[**569.** **Sum of average of all subsets** 328](#_Toc20051786)

[**570.** **Total number of non-decreasing numbers with n digits** 328](#_Toc20051787)

[**571.** **Unbounded Knapsack (Repetition of items allowed)** 329](#_Toc20051788)

[**572.** **Ways to arrange Balls such that adjacent balls are of different types** 329](#_Toc20051789)

[**573.** **Ways to arrange Balls such that adjacent balls are of different types using Dynamic Programming** 330](#_Toc20051790)

[**574.** **Count number of ways to partition a set into k subsets** 330](#_Toc20051791)

[**575.** **Weighted Job Scheduling in O (n Log n) time** 331](#_Toc20051792)

[**576.** **Given a text and a wildcard pattern, implement wildcard pattern matching algorithm that finds if wildcard pattern is matched with text** 332](#_Toc20051793)

[**577.** **Given a sequence of words, and a limit on the number of characters that can be put in one line (line width). Put line breaks in the given sequence such that the lines are printed neatly. Assume that the length of each word is smaller than the line width.** 332](#_Toc20051794)

[**578.** **Ways to write n as sum of two or more positive integers: -** for a given number>0 find the number of different ways in which n can be written as a sum of at two or more positive integers. 333](#_Toc20051795)

[**579.** **Anagram Using HashMap** 334](#_Toc20051796)

[**580.** **Anagram Using Iterative** 334](#_Toc20051797)

[**581.** **Anagram Using Sort and Equals** 335](#_Toc20051798)

[**582.** **Anagram Using String Builder** 335](#_Toc20051799)

[**583.** **Armstrong Number** 336](#_Toc20051800)

[**584.** **Array Element Count** 336](#_Toc20051801)

[**585.** **ArrayList to Array** 337](#_Toc20051802)

[**586.** **Array to ArrayList** 337](#_Toc20051803)

[**587.** **Array Reverse** 337](#_Toc20051804)

[**588.** **Check Mobile Number** 338](#_Toc20051805)

[**589.** **Common Elements Using Iterative** 338](#_Toc20051806)

[**590.** **Common Elements Using retainAll** 338](#_Toc20051807)

[**591.** **Compare Text Files** 339](#_Toc20051808)

[**592.** **Convert Decimal to Other** 339](#_Toc20051809)

[**593.** **Write a java program to count the total number of occurrences of a given character in a string without using any loop?** 340](#_Toc20051810)

[**594.** **Java program to count the number of words in a string?** 340](#_Toc20051811)

[**595.** **Java program to count the number of words in a string other method** 340](#_Toc20051812)

[**596.** **Delegation** 340](#_Toc20051813)

[**597.** **Detect Deadlock** 341](#_Toc20051814)

[**598.** **Find Difference between Two Dates** 342](#_Toc20051815)

[**599.** **Find Difference between Two Dates in Java Using TimeUnit** 342](#_Toc20051816)

[**600.** **Duplicate Characters in String** 343](#_Toc20051817)

[**601.** **Duplicates in Array Using Brute Force** 343](#_Toc20051818)

[**602.** **Duplicates in Array Using HashSet** 344](#_Toc20051819)

[**603.** **Find the duplicate words and their number of occurrences in a string?** 344](#_Toc20051820)

[**604.** **Java Program to Count Occurrences of Each Character in String** 344](#_Toc20051821)

[**605.** **Equality of Two Arrays Using deep equals** 344](#_Toc20051822)

[**606.** **Equality of Two Arrays Using equals** 345](#_Toc20051823)

[**607.** **Equality of Two Arrays Using Iterative** 345](#_Toc20051824)

[**608.** **Equality of Two Arrays Using Sort and Equals** 345](#_Toc20051825)

[**609.** **Fibonacci Series** 345](#_Toc20051826)

[**610.** **Java Program to Append Text to an Existing File** 346](#_Toc20051827)

[**611.** **Find Duplicate Elements1 Using Brute Force Method** 346](#_Toc20051828)

[**612.** **Find Duplicate Elements2 Using HashSet** 346](#_Toc20051829)

[**613.** **First Repeated NonRepeated Char** 347](#_Toc20051830)

[**614.** **Floyd’s triangle is the right angled triangle consists of natural numbers** 347](#_Toc20051831)

[**615.** **Pattern 1: dd/MM/yyyy (Ex: 10/09/2016)** 348](#_Toc20051832)

[**616.** **Pattern 2: yyyy-MM-dd (Ex: 2016-09-10)** 348](#_Toc20051833)

[**617.** **Pattern 3: dd MMMM yyyy (Ex: 10 September 2016)** 348](#_Toc20051834)

[**618.** **Pattern 4: E, dd MMM yyyy (Ex: Sat, 10 Sep 2016)** 348](#_Toc20051835)

[**619.** **Pattern 5: dd-MMM-yyyy HH:mm:ss (Ex: 10-Sep-2016 18:40:47)** 348](#_Toc20051836)

[**620.** **Pattern 6: EEEE, MMM dd yyyy, hh:mm:ss a (Ex: Saturday, Sep 10 2016, 06:45:51 PM)** 349](#_Toc20051837)

[**621.** **Pattern 7: dd-MMM-yyyy HH:mm:ss z (Ex: 10-Sep-2016 18:49:53 IST)** 349](#_Toc20051838)

[**622.** **Pattern 8: dd-MMM-yyyy HH:mm:ss Z (Ex: 10-Sep-2016 19:01:39 +0530)** 349](#_Toc20051839)

[**623.** **Find Frequency of All Digits in Number Using Collection** 349](#_Toc20051840)

[**624.** **Find Frequency of All Digits in Number without Using Collection** 350](#_Toc20051841)

[**625.** **Harshad number or Niven number is a number which is divisible by the sum of its digits.** 350](#_Toc20051842)

[**626.** **Integer to String** 351](#_Toc20051843)

[**627.** **String to Integer** 351](#_Toc20051844)

[**628.** **Intersection Iterative Using Iterative Method** 351](#_Toc20051845)

[**629.** **Intersection of Multiple Arrays Using retainAll () Method** 351](#_Toc20051846)

[**630.** **Java Program to Find Largest Number Less than Given Number and Without a Given Digit** 352](#_Toc20051847)

[**631.** **Launching External Apps** 352](#_Toc20051848)

[**632.** **Leaders in Array1** 352](#_Toc20051849)

[**633.** **Leaders in Array2** 353](#_Toc20051850)

[**634.** **List of All Files1** 353](#_Toc20051851)

[**635.** **We are sorting an ArrayList of strings using first form of Collections.sort () method** 354](#_Toc20051852)

[**636.** **We are sorting an ArrayList of strings using first form of Collections.sort () method with two arguments.** 354](#_Toc20051853)

[**637.** **Sort an ArrayList of Custom Objects: -** we sort an arraylist of student object. To do this, student class must implement Comparable interface and override comapreTo () method. 355](#_Toc20051854)

[**638.** **Sort an ArrayList of Custom Objects Using Comparator: -** we sort an arraylist of student objects by using our aown Comaparator. This comparator sorts the student objects based on theirs percentage. 355](#_Toc20051855)

[**639.** **Sort an ArrayList in the Reverse Order: -** you can sort the list in the reverse order also by passing the Comparator returned by collections.reverseOrder () as comaparator to collections.sort () method. 356](#_Toc20051856)

[**640.** **Longest Substring without Repeating Char** 356](#_Toc20051857)

[**641.** **Matrix Addition** 357](#_Toc20051858)

[**642.** **Matrix Creation** 358](#_Toc20051859)

[**643.** **Matrix Multiplication** 358](#_Toc20051860)

[**644.** **Matrix Subtraction** 359](#_Toc20051861)

[**645.** **Matrix Transpose** 360](#_Toc20051862)

[**646.** **Missing Number in Array** 361](#_Toc20051863)

[**647.** **Most Frequent Element** 361](#_Toc20051864)

[**648.** **Number Is Binary or Not** 362](#_Toc20051865)

[**649.** **Pairs of Elements in Array Using Alternative** 362](#_Toc20051866)

[**650.** **Find All Pairs Of Elements in an Array Whose Sum Is Equal to a Given Number** 363](#_Toc20051867)

[**651.** **Pattern1** 363](#_Toc20051868)

[**652.** **Pattern2** 364](#_Toc20051869)

[**653.** **Pattern3** 364](#_Toc20051870)

[**654.** **Pattern4** 364](#_Toc20051871)

[**655.** **Pattern5** 365](#_Toc20051872)

[**656.** **Pattern6** 365](#_Toc20051873)

[**657.** **Pattern7** 366](#_Toc20051874)

[**658.** **Pattern8** 366](#_Toc20051875)

[**659.** **Pattern9** 367](#_Toc20051876)

[**660.** **Pattern10** 367](#_Toc20051877)

[**661.** **Pattern11** 368](#_Toc20051878)

[**662.** **Pattern12** 368](#_Toc20051879)

[**663.** **Pattern13** 369](#_Toc20051880)

[**664.** **Pattern14** 369](#_Toc20051881)

[**665.** **Pattern15** 370](#_Toc20051882)

[**666.** **Pattern16** 370](#_Toc20051883)

[**667.** **Pattern17** 371](#_Toc20051884)

[**668.** **Pattern18** 371](#_Toc20051885)

[**669.** **Pattern19** 372](#_Toc20051886)

[**670.** **Pattern20** 372](#_Toc20051887)

[**671.** **Pattern21** 373](#_Toc20051888)

[**672.** **Pattern22** 373](#_Toc20051889)

[**673.** **Pattern23** 374](#_Toc20051890)

[**674.** **Find the Percentage of Uppercase Letters, Lowercase Letters, Digits and Other Special Characters in a String** 374](#_Toc20051891)

[**675.** **Permutations of String** 375](#_Toc20051892)

[**676.** **Print Diamond1** 375](#_Toc20051893)

[**677.** **Print Diamond2** 376](#_Toc20051894)

[**678.** **Read and Write Image from Local Disc** 377](#_Toc20051895)

[**679.** **Read and Write Image from URL** 377](#_Toc20051896)

[**680.** **Remove All Vowels** 377](#_Toc20051897)

[**681.** **Remove Duplicates Elements** 378](#_Toc20051898)

[**682.** **Remove WhiteSpaces** 378](#_Toc20051899)

[**683.** **To Find the Most Repeated Word in Text File** 379](#_Toc20051900)

[**684.** **Find All Repeated Words in Text File and Their Occurrences in Java** 379](#_Toc20051901)

[**685.** **Java Program to Reverse and Add a Number Until You Get a Palindrome** 380](#_Toc20051902)

[**686.** **Reverse Array without Additional Array** 381](#_Toc20051903)

[**687.** **Reverse Each Word** 381](#_Toc20051904)

[**688.** **Reverse String with Preserving Position** 381](#_Toc20051905)

[**689.** **Reverse the String** 382](#_Toc20051906)

[**690.** **Find Roman Equivalent of a Decimal Number** 382](#_Toc20051907)

[**691.** **Saddle point of a matrix is an element in the matrix which is smallest in its row and largest in its column. A matrix can have many or no saddle points.** 383](#_Toc20051908)

[**692.** **Second Largest Number** 384](#_Toc20051909)

[**693.** **Performance of the Selection Sort Algorithm** 384](#_Toc20051910)

[**694.** **Separate Zeroes from NonZeros1** 384](#_Toc20051911)

[**695.** **Separate Zeroes from NonZeros2** 385](#_Toc20051912)

[**696.** **Set Execute Permission of a File in Java** 385](#_Toc20051913)

[**697.** **Set File Permissions in Java 7** 385](#_Toc20051914)

[**698.** **Set Read Permission of a File in Java** 386](#_Toc20051915)

[**699.** **Set Write Permission of a File in Java** 386](#_Toc20051916)

[**700.** **Sort Text File** 386](#_Toc20051917)

[**701.** **Spiral Matrix In AntiClockwise** 387](#_Toc20051918)

[**702.** **Create spiral of numbers or spiral matrix or circular matrix of order n\*n in both clockwise and anti-clockwise directions.** 388](#_Toc20051919)

[**703.** **Stop a Thread in Java Using a Boolean Variable** 389](#_Toc20051920)

[**704.** **Stop Thread Using Interrupt** 389](#_Toc20051921)

[**705.** **String Rotation** 390](#_Toc20051922)

[**706.** **Java Program to Find Continuous Sub Array in Array Whose Sum Is Equal to Number** 390](#_Toc20051923)

[**707.** **Sub Array Whose Sum Is Number2** 390](#_Toc20051924)

[**708.** **Sum of All Digits of Number** 391](#_Toc20051925)

[**709.** **Swap Two Strings** 391](#_Toc20051926)

[**710.** **Symmetric Matrix** 391](#_Toc20051927)

[**711.** **Write a java program or function to replace specific string in text file. Your program should take one text file as input and replace a specific string in that text file with new string.** 392](#_Toc20051928)

[**712.** **Trigonometric Functions** 393](#_Toc20051929)

[**713.** **Find Union of Multiple Arrays Using HashSet** 393](#_Toc20051930)

[**714.** **Word Count in File** 394](#_Toc20051931)

[**715.** **Given a Boolean matrix mat [M] [N] of size M X N, modify it such that if a matrix cell mat[i] [j] is 1 (or true) then make all the cells of ith row and jth column as 1. Using two temporary arrays.** 394](#_Toc20051932)

[**716.** **Boolean Matrix1 Method 2 (A Space Optimized Version of Method 1)** 395](#_Toc20051933)

[**717.** **Find a common element in all rows of a given row-wise sorted matrix: -** given a matrix where every row is sorted in increasing order. Write a function that finds and returns a common element in all rows. If there is no common element, then return -1. 396](#_Toc20051934)

[**718.** **Count Number of islands where every island is row-wise and column-wise separated: -** given a rectangular matrix which has only two possible values ’X’ and ‘O’. The values ‘X’ always appear in form of rectangular islands and these islands are always row-wise and column-wise and column-wise separated by at least on line of ‘O’s. Note that islands can only be diagonally adjacent. 397](#_Toc20051935)

[**719.** **Create a matrix with alternating rectangles of O and X** 397](#_Toc20051936)

[**720.** **Given an n x n square matrix, find sum of all sub-squares of size k x k** 398](#_Toc20051937)

[**721.** **Find Sum of All Sub Squares1** 399](#_Toc20051938)

[**722.** **Maximum size square sub-matrix with all 1s:-** given a binary matrix, find out the maximum size square sub-matrix with all 1s. 399](#_Toc20051939)

[**723.** **Given a 2D array, print it in spiral form. See the following examples.** 400](#_Toc20051940)

[**724.** **Given an n x n matrix and a number x, find position of x in the matrix if it is present in it. Else print “Not Found”. In the given matrix, every row and column is sorted in increasing order. The designed algorithm should have linear time complexity.** 401](#_Toc20051941)

[**725.** **Implement Binary Search Tree (BST)** 401](#_Toc20051942)

[**726.** **Find min and max value from Binary Search Tree (BST)** 403](#_Toc20051943)

[**727.** **Find height of a Binary Search Tree (BST)** 404](#_Toc20051944)

[**728.** **Implement Binary Search Tree (BST) Level order traversal (breadth first).** 405](#_Toc20051945)

[**729.** **Implement Binary Search Tree (BST) pre-order traversal (depth first).** 406](#_Toc20051946)

[**730.** **Implement Binary Search Tree (BST) in-order traversal (depth first).** 407](#_Toc20051947)

[**731.** **Implement Binary Search Tree (BST) post-order traversal (depth first).** 408](#_Toc20051948)

[**732.** **How to delete a node from Binary Search Tree (BST)?** 409](#_Toc20051949)

[**733.** **Binary to Decimal** 411](#_Toc20051950)

[**734.** **Decimal to Binary** 411](#_Toc20051951)

[**735.** **Write a program to find common elements between two arrays.** 412](#_Toc20051952)

[**736.** **Write a program to find out duplicate characters in a string.** 412](#_Toc20051953)

[**737.** **Duplicate Number** 412](#_Toc20051954)

[**738.** **Find out middle index where sum of both ends are equal.** 413](#_Toc20051955)

[**739.** **How to check the given Binary Tree is Binary Search Tree (BST) or not?** 413](#_Toc20051956)

[**740.** **Is Perfect Number** 414](#_Toc20051957)

[**741.** **Write a program to find the sum of the first 1000 prime numbers.** 414](#_Toc20051958)

[**742.** **Write a program to find two lines with max characters in descending order.** 415](#_Toc20051959)

[**743.** **Write a program to find maximum repeated words from a file.** 416](#_Toc20051960)

[**744.** **Write a program to get a line with max word count from the given file.** 417](#_Toc20051961)

[**745.** **Armstrong Number** 418](#_Toc20051962)

[**746.** **Write a program to implement ArrayList.** 418](#_Toc20051963)

[**747.** **Binary Check** 419](#_Toc20051964)

[**748.** **Bubble Sort: -** bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items ans swapping them is they are in wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements bubble to the top of the list, because it only uses comparisons to operate on elements, it is a comparison sort. 419](#_Toc20051965)

[**749.** **Insertion Sort:-** Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time.it is much less efficient on large lists than more advanced algorithms such a quicksort ,heapsort or merge sort . Every repetition of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list, unitl no input elements remains.the choice of which element to remove from the input is arbitrary, and can be made using almost any choice algorithm. 420](#_Toc20051966)

[**750.** **Binary Insertion Sort** 421](#_Toc20051967)

[**751.** **Bitonic Sort** 421](#_Toc20051968)

[**752.** **Bubble Sort** 422](#_Toc20051969)

[**753.** **Bubble Sort1** 422](#_Toc20051970)

[**754.** **Comb Sort** 423](#_Toc20051971)

[**755.** **Counting Sort** 423](#_Toc20051972)

[**756.** **Cycle Sort** 424](#_Toc20051973)

[**757.** **Gnome Sort** 424](#_Toc20051974)

[**758.** **Heap Sort** 425](#_Toc20051975)

[**759.** **Iterative Merge Sort** 426](#_Toc20051976)

[**760.** **Iterative Merge Sort1** 426](#_Toc20051977)

[**761.** **Merge Sort** 428](#_Toc20051978)

[**762.** **Merge Sort 3 Way** 429](#_Toc20051979)

[**763.** **Iterative Quick Sort** 430](#_Toc20051980)

[**764.** **Iterative Quick Sort1** 430](#_Toc20051981)

[**765.** **Quick Sort** 431](#_Toc20051982)

[**766.** **QuickSort\_using\_Doubly\_LinkedList** 432](#_Toc20051983)

[**767.** **Pancake Sort** 433](#_Toc20051984)

[**768.** **Pigeonhole Sort** 434](#_Toc20051985)

[**769.** **Radix Sort** 434](#_Toc20051986)

[**770.** **Recursive Bubble Sort** 435](#_Toc20051987)

[**771.** **Recursive Insertion Sort** 435](#_Toc20051988)

[**772.** **Selection Sort** 436](#_Toc20051989)

[**773.** **Shell Sort** 436](#_Toc20051990)

[**774.** **How to sort a Stack using a temporary Stack?** You have a stack with full of integers. Sorting it in the ascending order using another temporary rray by using all stack functionality. 437](#_Toc20051991)

[**775.** **Deadlock** 437](#_Toc20051992)

[**776.** **How to get distinct elements from an array by avoiding duplicate elements?** 438](#_Toc20051993)

[**777.** **Distinct File Words** 438](#_Toc20051994)

[**778.** **Write a program to remove duplicates from sorted array.** 439](#_Toc20051995)

[**779.** **Fibonacci** 439](#_Toc20051996)

[**780.** **Write a program to implement hashcode and equals.** 440](#_Toc20051997)

[**781.** **Find longest substring without repeating characters.** 440](#_Toc20051998)

[**782.** **Write a program to find sum of each digit in the given number using recursion.** 441](#_Toc20051999)

[**783.** **Prime Num Check** 441](#_Toc20052000)

[**784.** **Write a singleton class:-**Singleton class means you can create only one object for the given class. You cab create a singleton class by making its constructor as private, so that you can restrict the creation of the object. Provide a static method to get instance of the object, where in you can handle the object creation inside the class only. 442](#_Toc20052001)

[**785.** **Write a program to convert string to number without using Integer.parseInt () method.** 442](#_Toc20052002)

[**786.** **How to swap two numbers without using temporary variable?** 442](#_Toc20052003)

[**787.** **Number Reverse** 443](#_Toc20052004)

[**788.** **Write a program to sort a map by value.** 443](#_Toc20052005)

[**789.** **Reverse Singly Linked List** 443](#_Toc20052006)

[**790.** **Write a program to print all permutations of a given string.** 445](#_Toc20052007)

[**791.** **String Recursive Reversal** 445](#_Toc20052008)

[**792.** **Write a program to find top two maximum numbers in an array.** 445](#_Toc20052009)

[**793.** **Composition: -** Composition is a restricted form of Aggregation in which two entities are highly dependent on each other. It represents part of relationship. In Composition, both the entities are dependent on each other, where there is composition between two entities, the composed object cannot exist without the other entity. 446](#_Toc20052010)

[**794.** **Aggregation: -** It represents HAS-A relationship.It is unidirectional association i.e. a one way relationship. In Aggregation, both the entities can serive individually which means ending one entity will not effect the other entity. Code reuse is best achieved by aggregartion. 447](#_Toc20052011)

[**795.** **Association: -** Association is relationship between two separate classes which establishes through their objects.Association can be one-to-one , one-to-many, many-to-one, many-to-many. 448](#_Toc20052012)

[**796.** **Train Ticket Booking: - There is only 1 ticket available in train, and two passengers are trying to book that ticket at same time.** 448](#_Toc20052013)

[**797.** **Check string contains special characters in java using regex.** 449](#_Toc20052014)

[**798.** **Generate account number dynamically:-** Java program to design a class account using the inheritance and static that show all function of bank and generate account number dynamically. 449](#_Toc20052015)

[**799.** **Highest score Student: -** WAP that accepts 10 students’ records (roll no and score) and prints them in decreasing order of scores. In case there are multiple records printing to the same student, the program should choose a single record containing the highest score. Then program should capable of accepting a multiline score. Each subsequent line of input will contain a student record, that is roll no and score. The output should consist of the conbimation of roll no and corresponding score in decreasing order of same. 450](#_Toc20052016)

[**800.** **WAP to sort the numeric sub array using Arrays parallelSortUserInput.** 451](#_Toc20052017)

# Alternative Sorting: - Given an array of interger, print the array in such way that the first element is first maximum and second element is first minimum and so on.

|  |
| --- |
| Input : arr[] = {7, 1, 2, 3, 4, 5, 6} Output : 7 1 6 2 5 3 4  Input : arr[] = {1, 6, 9, 4, 3, 7, 8, 2} Output : 9 1 8 2 7 3 6 4  Time Complexity: O(n Log n) Auxiliary Space : O(1)\*/  public class AlternativeSorting {  public static void main(String[] args) {  int arr[] = { 1, 12, 4, 6, 7, 10 };  int n = arr.length;  alternateSort(arr, n);  }  static void alternateSort(int arr[], int n) {  Arrays.sort(arr);  int i = 0, j = n - 1;  while (i < j) {  System.out.print(arr[j--] + " ");  System.out.print(arr[i++] + " ");  }  if (n % 2 != 0)  System.out.print(arr[i]);  }  } |

# Arrange Given Numbers Set1: - arrange given numbers to form the biggest number.

|  |
| --- |
| public class ArrangeGivenNumbers\_Set1 {  public static void main(String[] args) {  Vector<String> arr;  arr = new Vector<>();  // output should be 6054854654  arr.add("54");  arr.add("546");  arr.add("548");  arr.add("60");  printLargest(arr);  }  static void printLargest(Vector<String> arr) {  Collections.sort(arr, new Comparator<String>() {  @Override  public int compare(String X, String Y) {  String XY = X + Y;  String YX = Y + X;  return XY.compareTo(YX) > 0 ? -1 : 1;  }  });  Iterator it = arr.iterator();  while (it.hasNext())  System.out.print(it.next());  }  } |

# Arrange Given Numbers Set2 - arrange given numbers to form the biggest number.

|  |
| --- |
| Input : [1, 34, 3, 98, 9, 76, 45, 4, 12, 121] Output : 99876454343121211  Input : [12, 121] Output : 12121  public class ArrangeGivenNumbers\_Set2 {  public static void main(String[] args) {  Integer arr[] = { 1, 34, 3, 98, 9, 76, 45, 4, 12, 121 };  List<Integer> l = Arrays.asList(arr);  System.out.println(largestNumber(l));  }  public static String largestNumber(List<Integer> arr) {  int n = Collections.max(arr).toString().length();  ArrayList<ExtendedNum> en = new ArrayList<ExtendedNum>();  for (int i = 0; i < arr.size(); i++)  en.add(new ExtendedNum(arr.get(i), n));  Collections.sort(en, (p1, p2) -> (int) (p2.modifiedValue - p1.modifiedValue));  StringBuilder sb = new StringBuilder();  for (int i = 0; i < en.size(); i++)  sb.append(new StringBuilder(Long.toString(en.get(i).originalValue)));  BigInteger bi = new BigInteger(sb.toString());  return bi.toString();  }  }  class ExtendedNum {  int originalValue;  long modifiedValue;  public ExtendedNum(int originalValue, int n) {  this.originalValue = originalValue;  String s = Integer.toString(originalValue);  StringBuilder sb = new StringBuilder(s);  StringBuilder ans = new StringBuilder();  while (ans.length() <= n + 1)  ans.append(sb);  s = ans.toString().substring(0, n + 1);  modifiedValue = Long.parseLong(s);  }  public String toString() {  return "[" + modifiedValue + ", " + originalValue + "]";  }  } |

# Array Can Be Divided Into Pairs :- check if an array can be divided into pairs whose sum is divisible by K given an array of intergers and a number K, write a function that returns true if given array can be divided into pairs such that sum of every pairs is divisible by K.

|  |
| --- |
| Input: arr[] = {9, 7, 5, 3}, k = 6 Output: True We can divide array into (9, 3) and (7, 5). Sum of both of these pairs is a multiple of 6.  Input: arr[] = {92, 75, 65, 48, 45, 35},k = 10 Output: True We can divide array into (92, 48), (75, 65) and (45, 35). Sum of all these pairs is a multiple of 10.  Input: arr[] = {91, 74, 66, 48}, k = 10 Output: False  **public** **class** Array\_Can\_Be\_Divided\_Into\_Pairs {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 92, 75, 65, 48, 45, 35 };  **int** k = 10;  **boolean** ans = *canPairs*(arr, k);  **if** (ans) System.***out***.println("True");  **else** System.***out***.println("False");  }  **static** **boolean** canPairs(**int** ar[], **int** k) {  **if** (ar.length % 2 == 1) **return** **false**;  HashMap<Integer, Integer> hm = **new** HashMap<>();  **for** (**int** i = 0; i < ar.length; i++) {  **int** rem = ar[i] % k;  **if** (!hm.containsKey(rem)) {  hm.put(rem, 0);  }  hm.put(rem, hm.get(rem) + 1);  }  **for** (**int** i = 0; i < ar.length; i++) {  **int** rem = ar[i] % k;  **if** (2 \* rem == k) {  **if** (hm.get(rem) % 2 == 1) **return** **false**;  }  **else** **if** (rem == 0) {  **if** (hm.get(rem) % 2 == 1) **return** **false**;  }  **else** {  **if** (hm.get(k - rem) != hm.get(rem)) **return** **false**;  }  }  **return** **true**;  }  } |

# Array Contains Contiguous Integers Simple Solution: - check if array contains contiguous integers with duplicates allowed simple solution

|  |
| --- |
| Input : arr[] = {5, 2, 3, 6, 4, 4, 6, 6} Output : Yes The elements form a contiguous set of integers which is {2, 3, 4, 5, 6}.  Input : arr[] = {10, 14, 10, 12, 12, 13, 15} Output : No  **public** **class** ArrayContainsContiguousIntegersSimpleSolution {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };  **int** n = arr.length;  **if** (*areElementsContiguous*(arr, n))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** **boolean** areElementsContiguous(**int** arr[], **int** n) {  Arrays.*sort*(arr);  **for** (**int** i = 1; i < n; i++)  **if** (arr[i] - arr[i - 1] > 1)  **return** **false**;  **return** **true**;  }  } |

# Array Contains Contiguous Integers Using HashTable: - check if array contains contiguous integers with duplicates using hash table.

|  |
| --- |
| **public** **class** ArrayContainsContiguousIntegersUsingHashTable {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };  **int** n = arr.length;  **if** (*areElementsContiguous*(arr, n))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** Boolean areElementsContiguous(**int** arr[], **int** n) {  HashSet<Integer> us = **new** HashSet<Integer>();  **for** (**int** i = 0; i < n; i++)  us.add(arr[i]);  **int** count = 1;  **int** curr\_ele = arr[0] - 1;  **while** (us.contains(curr\_ele) == **true**) {  count++;  curr\_ele--;  }  curr\_ele = arr[0] + 1;  **while** (us.contains(curr\_ele) == **true**) {  count++;  curr\_ele++;  }  **return** (count == (us.size()));  }  } |

# Array Contains Contiguous Integers Using Visited Array: - check if array contains contiguous integers with duplicates using visited array.

|  |
| --- |
| **public** **class** ArrayContainsContiguousIntegersUsingVisitedArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 2, 3, 6, 4, 4, 6, 6 };  **int** n = arr.length;  **if** (*areElementsContiguous*(arr, n))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** **boolean** areElementsContiguous(**int** arr[], **int** n) {  **int** max = Integer.***MIN\_VALUE***;  **int** min = Integer.***MAX\_VALUE***;  **for** (**int** i = 0; i < n; i++) {  max = Math.*max*(max, arr[i]);  min = Math.*min*(min, arr[i]);  }  **int** m = max - min + 1;  **if** (m > n)  **return** **false**;  **boolean** visited[] = **new** **boolean**[n];  **for** (**int** i = 0; i < n; i++)  visited[arr[i] - min] = **true**;  **for** (**int** i = 0; i < m; i++)  **if** (visited[i] == **false**)  **return** **false**;  **return** **true**;  }  } |

# Array Contains Duplicate Elements with in K Distance: - check if a given array contains duplicate elements with in K distance from each other.Given an unsorted array that may contain duplicates. Also given a number K which is smaller than size of array.write a function that returns true if array contains cuplicates with in K distance.

|  |
| --- |
| Input: k = 3, arr[] = {1, 2, 3, 4, 1, 2, 3, 4} Output: false All duplicates are more than k distance away.  Input: k = 3, arr[] = {1, 2, 3, 1, 4, 5} Output: true 1 is repeated at distance 3.  Input: k = 3, arr[] = {1, 2, 3, 4, 5} Output: false  Input: k = 3, arr[] = {1, 2, 3, 4, 4} Output: true time complexity Θ(n)  **public** **class** ArrayContainsDuplicateElementswithinKDistance {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 5, 3, 4, 3, 5, 6 };  **if** (*checkDuplicatesWithinK*(arr, 3))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** **boolean** checkDuplicatesWithinK(**int** arr[], **int** k) {  HashSet<Integer> set = **new** HashSet<>();  **for** (**int** i = 0; i < arr.length; i++) {  **if** (set.contains(arr[i]))  **return** **true**;  set.add(arr[i]);  **if** (i >= k)  set.remove(arr[i - k]);  }  **return** **false**;  }  } |

# Array Elements Are Consecutive Method1

|  |
| --- |
| /\*Use visited array Time Complexity: O(n) Extra Space: O(n)  a) If array is {5, 2, 3, 1, 4}, then the function should return true because the array has consecutive numbers from 1 to 5.  b) If array is {83, 78, 80, 81, 79, 82}, then the function should return true because the array has consecutive numbers from 78 to 83.  c) If the array is {34, 23, 52, 12, 3}, then the function should return false because the elements are not consecutive.  d) If the array is {7, 6, 5, 5, 3, 4}, then the function should return false because 5 and 5 are not consecutive.\*/  **public** **class** ArrayElementsAreConsecutive\_Method\_1 {  **public** **static** **void** main(String[] args) {  ArrayElementsAreConsecutive\_Method\_1 consecutive = **new** ArrayElementsAreConsecutive\_Method\_1();  **int** arr[] = { 5, 4, 2, 3, 1, 6 };  **int** n = arr.length;  **if** (consecutive.areConsecutive(arr, n) == **true**)  System.***out***.println("Array elements are consecutive");  **else**  System.***out***.println("Array elements are not consecutive");  }  **boolean** areConsecutive(**int** arr[], **int** n) {  **if** (n < 1)  **return** **false**;  **int** min = getMin(arr, n);  **int** max = getMax(arr, n);  **if** (max - min + 1 == n) {  **boolean** visited[] = **new** **boolean**[n];  **int** i;  **for** (i = 0; i < n; i++) {  **if** (visited[arr[i] - min] != **false**)  **return** **false**;  visited[arr[i] - min] = **true**;  }  **return** **true**;  }  **return** **false**; // if (max - min + 1 != n)  }  **int** getMin(**int** arr[], **int** n) {  **int** min = arr[0];  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] < min)  min = arr[i];  }  **return** min;  }  **int** getMax(**int** arr[], **int** n) {  **int** max = arr[0];  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] > max)  max = arr[i];  }  **return** max;  }  } |

# Array Elements Are Consecutive Method2

|  |
| --- |
| /\*Mark visited array elements as negative Time Complexity: O(n) Extra Space: O(1)\*/  **public** **class** ArrayElementsAreConsecutive\_Method\_2 {  **public** **static** **void** main(String[] args) {  ArrayElementsAreConsecutive\_Method\_2 consecutive = **new** ArrayElementsAreConsecutive\_Method\_2();  **int** arr[] = { 2, 1, 0, -3, -1, -2 };  **int** n = arr.length;  **if** (consecutive.areConsecutive(arr, n) == **true**)  System.***out***.println("Array elements are consecutive");  **else**  System.***out***.println("Array elements are not consecutive");  }  **boolean** areConsecutive(**int** arr[], **int** n) {  **if** (n < 1)  **return** **false**;  **int** min = getMin(arr, n);  **int** max = getMax(arr, n);  **if** (max - min + 1 == n) {  **int** i;  **for** (i = 0; i < n; i++) {  **int** j;  **if** (arr[i] < 0)  j = -arr[i] - min;  **else**  j = arr[i] - min;  **if** (arr[j] > 0)  arr[j] = -arr[j];  **else**  **return** **false**;  }  **return** **true**;  }  **return** **false**; // if (max-min+1 != n)  }  **int** getMin(**int** arr[], **int** n) {  **int** min = arr[0];  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] < min)  min = arr[i];  }  **return** min;  }  **int** getMax(**int** arr[], **int** n) {  **int** max = arr[0];  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] > max)  max = arr[i];  }  **return** max;  }  } |

# Array Is Subset of another Array Using Hashing: - Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Using Hashing

|  |
| --- |
| public class ArrayIsSubsetOfAnotherArray\_Hashing {  public static void main(String[] args) {  int arr1[] = { 11, 1, 13, 21, 3, 7 };  int arr2[] = { 11, 3, 7, 1 };  int m = arr1.length;  int n = arr2.length;  if (isSubset(arr1, arr2, m, n))  System.out.println("arr2 is a subset of arr1");  else  System.out.println("arr2 is not a subset of arr1");  }  static boolean isSubset(int arr1[], int arr2[], int m, int n) {  HashSet<Integer> hset = new HashSet<>();  for (int i = 0; i < m; i++) {  if (!hset.contains(arr1[i]))  hset.add(arr1[i]);  }  for (int i = 0; i < n; i++) {  if (!hset.contains(arr2[i]))  return false;  }  return true;  }  } |

# Array Is Subset of another Array: - Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Simple solution

|  |
| --- |
| Input: arr1[] = {11, 1, 13, 21, 3, 7}, arr2[] = {11, 3, 7, 1} , Output: arr2[] is a subset of arr1[]  Input: arr1[] = {1, 2, 3, 4, 5, 6}, arr2[] = {1, 2, 4} Output: arr2[] is a subset of arr1[]  Input: arr1[] = {10, 5, 2, 23, 19}, arr2[] = {19, 5, 3} Output: arr2[] is not a subset of arr1[]  Time Complexity: O(m\*n)  public class ArrayIsSubsetOfAnotherArray\_Simple {  public static void main(String[] args) {  int arr1[] = { 11, 1, 13, 21, 3, 7 };  int arr2[] = { 11, 3, 7, 1 };  int m = arr1.length;  int n = arr2.length;  if (isSubset(arr1, arr2, m, n))  System.out.print("arr2[] is subset of arr1[] ");  else  System.out.print("arr2[] is not a subset of arr1[]");  }  static boolean isSubset(int arr1[], int arr2[], int m, int n) {  int i = 0;  int j = 0;  for (i = 0; i < n; i++) {  for (j = 0; j < m; j++) {  if (arr2[i] == arr1[j])  break;  }  if (j == m)  return false;  }  return true;  }  } |

# Array Is Subset of another Array Using Sorting and Binary Search: - Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Using Sorting and Binary Search

|  |
| --- |
| //Time Complexity: O (mLogm + nLogm).  public class ArrayIsSubsetOfAnotherArray\_SortingandBinarySearch {  public static void main(String[] args) {  int arr1[] = { 11, 1, 13, 21, 3, 7 };  int arr2[] = { 11, 3, 7, 1 };  int m = arr1.length;  int n = arr2.length;  if (isSubset(arr1, arr2, m, n))  System.out.print("arr2[] is subset of arr1[] ");  else  System.out.print("arr2[] is not a subset of arr1[]");  }  static boolean isSubset(int arr1[], int arr2[], int m, int n) {  int i = 0;  sort(arr1, 0, m - 1);  for (i = 0; i < n; i++) {  if (binarySearch(arr1, 0, m - 1, arr2[i]) == -1)  return false;  }  return true;  }  static int binarySearch(int arr[], int low, int high, int x) {  if (high >= low) {  int mid = (low + high) / 2;  if ((mid == 0 || x > arr[mid - 1]) && (arr[mid] == x))  return mid;  else if (x > arr[mid])  return binarySearch(arr, (mid + 1), high, x);  else  return binarySearch(arr, low, (mid - 1), x);  }  return -1;  }  static void sort(int arr[], int low, int high) {  if (low < high) {  int pi = partition(arr, low, high);  sort(arr, low, pi - 1);  sort(arr, pi + 1, high);  }  }  static int partition(int arr[], int low, int high) {  int pivot = arr[high];  int i = (low - 1);  for (int j = low; j < high; j++) {  if (arr[j] <= pivot) {  i++;  int temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  int temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  return i + 1;  }  } |

# Array Is Subset of another Array Using Sorting and Merging: - Find whether an array is subset of another array.Given two arrays: arr1 [0…m-1] and arr2 [0…n-1]. Find whether arr2 [] is subset of arr1 [] or not. Both the array are not in sorted order. Using Sorting and Merging

|  |
| --- |
| //Time Complexity: O(mLogm + nLogn)  public class ArrayIsSubsetOfAnotherArray\_SortingAndMerging {  public static void main(String[] args) {  int arr1[] = { 11, 1, 13, 21, 3, 7 };  int arr2[] = { 11, 3, 7, 1 };  int m = arr1.length;  int n = arr2.length;  if (isSubset(arr1, arr2, m, n))  System.out.println("arr2 is a subset of arr1");  else  System.out.println("arr2 is not a subset of arr1");  }  static boolean isSubset(int arr1[], int arr2[], int m, int n) {  int i = 0, j = 0;  if (m < n)  return false;  Arrays.sort(arr1); // sorts arr1  Arrays.sort(arr2); // sorts arr2  while (i < n && j < m) {  if (arr1[j] < arr2[i])  j++;  else if (arr1[j] == arr2[i]) {  j++;  i++;  } else if (arr1[j] > arr2[i])  return false;  }  if (i < n)  return false;  else  return true;  }  } |

# Asynchronous Callback :- An asynchronous call do not block the program from the code execution.When the call returns from the event the call returns back to the callback function.So in the context of java we have to create a new thread invoke the callback method inside that thread.Callback may be invoked from a thread but is not requirement.A callback may also start a new thread thus making themselves asynchronous.When the tasks are not dependent on each other and may take some time for execution we should use asynchronous callbacks. For e.g. when you orded your food other people can also order their food in the restaurant.They don’t have to wait for your order to finish, if you’re downloading a song from internet.

|  |
| --- |
| Getting an API response.  interface OnGeekEventListener1 {  void onGeekEvent();  }  class B implements OnGeekEventListener1 {  @Override  public void onGeekEvent() {  System.out.println("Performing callback after Asynchronous Task");  }  }  public class Asynchronous\_Callback {  private OnGeekEventListener1 mListener;  public static void main(String[] args) {  Asynchronous\_Callback obj = new Asynchronous\_Callback();  OnGeekEventListener1 mListener = new B();  obj.registerOnGeekEventListener1(mListener);  obj.doGeekStuff();  }  public void registerOnGeekEventListener1(OnGeekEventListener1 mListener) {  this.mListener = mListener;  }  public void doGeekStuff() {  new Thread(new Runnable() {  public void run() {  System.out.println("Performing operation in Asynchronous Task");  if (mListener != null) {  mListener.onGeekEvent();  }  }  }).start();  }  } |

# BigDecimal class:- The Big Decimal class provides operations on double numbers for arithmetic,scale handling,rounding,comparison,format conversion and hasing.It can handle very large and very small floating point numbers with great precision but compensating with the time complexity a bit .A BigDecimal consists of a random precision interger unscaled value and a 32-bit integer scale.If grater than or equal to zero, the scale is the number of digit to the right of the decimal point.If less than zero,the unscaled value of the number is mulyiplied by 10^(-scale).

|  |
| --- |
| **Need Of BigDecimal**  1. The two java primitive types (double and float) are floating point numbers, which is stored as a binary representation of a Fraction and an exponent.  2. Other primitive types (except boolean) are fixed-point numbers. Unlike fixed point numbers, floating point numbers will most of the times return an answer with a small error (around 10^-19) This is the reason why we end up with 0.009999999999999998 as the result of 0.04-0.03 in the above example.\*/  public class BigDecimalClass {  public static void main(String[] args) {  BigDecimal bd1 = new BigDecimal("124567890.0987654321");  BigDecimal bd2 = new BigDecimal("987654321.123456789");  // Addition of two BigDecimals  bd1 = bd1.add(bd2);  System.out.println("BigDecimal1 = " + bd1);  // Multiplication of two BigDecimals  bd1 = bd1.multiply(bd2);  System.out.println("BigDecimal1 = " + bd1);  // Subtraction of two BigDecimals  bd1 = bd1.subtract(bd2);  System.out.println("BigDecimal1 = " + bd1);  // Division of two BigDecimals  bd1 = bd1.divide(bd2);  System.out.println("BigDecimal1 = " + bd1);  // BigDecima1 raised to the power of 2  bd1 = bd1.pow(2);  System.out.println("BigDecimal1 = " + bd1);  // Negate value of BigDecimal1  bd1 = bd1.negate();  System.out.println("BigDecimal1 = " + bd1);  }  } |

# Calendar After:- java.util.Calender.after() is a method in Calendar class of java.util package.The method returns true if the time represented by this Calender is after the time represented by when object.If it is not the case,flase is returned.

|  |
| --- |
| java.util.Calendar.after () is a method in Calendar class of java.util package. The method returns true if the time represented by this Calendar is after the time represented by when Object. If it is not the case, false is returned.\*/  **public** **class** Calendar\_After {  **public** **static** **void** main(String[] args) {  Calendar cal\_obj1 = Calendar.*getInstance*();  System.***out***.println("Time 1 : " + cal\_obj1.getTime());// printing current date  Calendar cal\_obj2 = Calendar.*getInstance*();// creating Calendar object  System.***out***.println("Time 2 : " + cal\_obj2.getTime()); // printing current date  System.***out***.println(cal\_obj1.after(cal\_obj2)); // checking if 1st date is after 2nd date and printing the result  cal\_obj2.set(Calendar.***YEAR***, 2010); // changing year in cal\_obj2 calendar  System.***out***.println("Result : " + cal\_obj1.after(cal\_obj2)); // check if calendar date is after current date  }  } |

# Calendar Before:- java.util.Calender.before() is a method in Calender class of java.util package.The method returns true if the time represented by this Calender is before the time represented by when object.If it is not the case , flase is returned.

|  |
| --- |
| **public** **class** Calendar\_Before {  **public** **static** **void** main(String[] args) **throws** InterruptedException {  Calendar cal\_obj1 = Calendar.*getInstance*();  System.***out***.println("Time 1 : " + cal\_obj1.getTime()); // printing current date  Thread.*sleep*(3000); // sleep for 3 seconds  Calendar cal\_obj2 = Calendar.*getInstance*(); // creating Calendar object  System.***out***.println("Time 2 : " + cal\_obj2.getTime()); // printing current date  System.***out***.println(cal\_obj1.before(cal\_obj2)); // checking if 1st date is before 2nd date and printing the result  cal\_obj2.set(Calendar.***YEAR***, 2010); // changing year in cal\_obj2 calendar  System.***out***.println("Year is " + cal\_obj2.get(Calendar.***YEAR***)); // displaying the year  System.***out***.println("Result : " + cal\_obj1.before(cal\_obj2)); // check if calendar date is before current date  cal\_obj2.set(Calendar.***YEAR***, 2025); // changing year in cal\_obj2 calendar  System.***out***.println("Year is " + cal\_obj2.get(Calendar.***YEAR***)); // displaying the year  System.***out***.println("Result : " + cal\_obj1.before(cal\_obj2)); // check if calendar date is before current date  }  } |

# Check for majority element in a sorted array using binary search

|  |
| --- |
| **public** **class** Check\_for\_Majority\_Element\_in\_a\_sorted\_array\_Using\_Binary\_Search {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 3, 3, 3, 10 };  **int** n = arr.length;  **int** x = 3;  **if** (*isMajority*(arr, n, x) == **true**)  System.***out***.println(x + " appears more than " + n / 2 + " times in arr[]");  **else**  System.***out***.println(x + " does not appear more than " + n / 2 + " times in arr[]");  }  **static** **boolean** isMajority(**int** arr[], **int** n, **int** x) {  **int** i = *\_binarySearch*(arr, 0, n - 1, x);  **if** (i == -1)  **return** **false**;  **if** (((i + n / 2) <= (n - 1)) && arr[i + n / 2] == x)  **return** **true**;  **else**  **return** **false**;  }  **static** **int** \_binarySearch(**int** arr[], **int** low, **int** high, **int** x) {  **if** (high >= low) {  **int** mid = (low + high) / 2; /\* low + (high - low)/2; \*/  **if** ((mid == 0 || x > arr[mid - 1]) && (arr[mid] == x))  **return** mid;  **else** **if** (x > arr[mid])  **return** *\_binarySearch*(arr, (mid + 1), high, x);  **else**  **return** *\_binarySearch*(arr, low, (mid - 1), x);  }  **return** -1;  }  } |

# Check for majority element in a sorted array using liner search

|  |
| --- |
| Input: arr[] = {1, 2, 3, 3, 3, 3, 10}, x = 3 Output: True (x appears more than n/2 times in the given array)  Input: arr[] = {1, 1, 2, 4, 4, 4, 6, 6}, x = 4 Output: False (x doesn't appear more than n/2 times in the given array)  **public** **class** Check\_for\_Majority\_Element\_in\_a\_sorted\_array\_Using\_Linear\_Search {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 4, 4, 4 };  **int** n = arr.length;  **int** x = 4;  **if** (*isMajority*(arr, n, x) == **true**)  System.***out***.println(x + " appears more than " + n / 2 + " times in arr[]");  **else**  System.***out***.println(x + " does not appear more than " + n / 2 + " times in arr[]");  }  **static** **boolean** isMajority(**int** arr[], **int** n, **int** x) {  **int** i, last\_index = 0;  last\_index = (n % 2 == 0)? n / 2 : n / 2 + 1;  **for** (i = 0; i < last\_index; i++) {  **if** (arr[i] == x && arr[i + n / 2] == x)  **return** **true**;  }  **return** **false**;  }  } |

# Check for majority element in an array using HashMap

|  |
| --- |
| **public** **class** Check\_for\_Majority\_Element\_in\_an\_array\_Using\_HashMap {  **public** **static** **void** main(String[] args) {  **int** a[] = **new** **int**[] { 2, 2, 2, 2, 5, 5, 2, 3, 3 };  *findMajority*(a);  }  **private** **static** **void** findMajority(**int**[] arr) {  HashMap<Integer, Integer> map = **new** HashMap<Integer, Integer>();  **for** (**int** i = 0; i < arr.length; i++) {  **if** (map.containsKey(arr[i])) {  **int** count = map.get(arr[i]) + 1;  **if** (count > arr.length / 2) {  System.***out***.println("Majority found :- " + arr[i]);  **return**;  } **else**  map.put(arr[i], count);  } **else**  map.put(arr[i], 1);  }  System.***out***.println(" No Majority element");  }  } |

# Check for majority element in an array using Moore’s voting algo

|  |
| --- |
| **public** **class** Check\_for\_Majority\_Element\_in\_an\_array\_Using\_Moore\_Voting\_Algorithm {  **public** **static** **void** main(String[] args) {  Check\_for\_Majority\_Element\_in\_an\_array\_Using\_Moore\_Voting\_Algorithm majorelement = **new** Check\_for\_Majority\_Element\_in\_an\_array\_Using\_Moore\_Voting\_Algorithm();  **int** a[] = **new** **int**[] { 1, 3, 3, 1, 2 };  **int** size = a.length;  majorelement.printMajority(a, size);  }  **void** printMajority(**int** a[], **int** size) {  **int** cand = findCandidate(a, size);  **if** (isMajority(a, size, cand))  System.***out***.println(" " + cand + " ");  **else**  System.***out***.println("No Majority Element");  }  **int** findCandidate(**int** a[], **int** size) {  **int** maj\_index = 0, count = 1;  **int** i;  **for** (i = 1; i < size; i++) {  **if** (a[maj\_index] == a[i])  count++;  **else**  count--;  **if** (count == 0) {  maj\_index = i;  count = 1;  }  }  **return** a[maj\_index];  }  **boolean** isMajority(**int** a[], **int** size, **int** cand) {  **int** i, count = 0;  **for** (i = 0; i < size; i++) {  **if** (a[i] == cand)  count++;  }  **if** (count > size / 2)  **return** **true**;  **else**  **return** **false**;  }  } |

# Check for pair in array [] with sum as x using Hash Map

|  |
| --- |
| **public** **class** Check\_for\_pair\_in\_Array\_with\_sum\_as\_x\_using\_HashMap {  **private** **static** **final** **int** ***MAX*** = 100000;  **public** **static** **void** main(String[] args) {  **int** A[] = { 1, 4, 45, 6, 10, 8 };  **int** n = 16;  *printpairs*(A, n);  }  **static** **void** printpairs(**int** arr[], **int** sum) {  **boolean**[] binmap = **new** **boolean**[***MAX***];  **for** (**int** i = 0; i < arr.length; ++i) {  **int** temp = sum - arr[i];  **if** (temp >= 0 && binmap[temp]) {  System.***out***.println("Pair with given sum " + sum + " is (" + arr[i] + ", " + temp + ")");  }  binmap[arr[i]] = **true**;  }  }  } |

# Check for pair in array [] with sum as x using Quick Sort

|  |
| --- |
| **public** **class** Check\_for\_pair\_in\_Array\_with\_sum\_as\_x\_using\_QuickSort {  **public** **static** **void** main(String[] args) {  **int** A[] = { 1, 4, 45, 6, 10, -8 };  **int** n = 16;  **int** arr\_size = 6;  **if** (*hasArrayTwoCandidates*(A, arr\_size, n))  System.***out***.println("Array has two " + "elements with sum 16");  **else**  System.***out***.println("Array doesn't have " + "two elements with sum 16 ");  }  **static** **boolean** hasArrayTwoCandidates(**int** A[], **int** arr\_size, **int** sum) {  **int** l, r;  *sort*(A, 0, arr\_size - 1);  l = 0;  r = arr\_size - 1;  **while** (l < r) {  **if** (A[l] + A[r] == sum)  **return** **true**;  **else** **if** (A[l] + A[r] < sum)  l++;  **else**  r--;  }  **return** **false**;  }  **static** **void** sort(**int** arr[], **int** low, **int** high) {  **if** (low < high) {  **int** pi = *partition*(arr, low, high);  *sort*(arr, low, pi - 1);  *sort*(arr, pi + 1, high);  }  }  **static** **int** partition(**int** arr[], **int** low, **int** high) {  **int** pivot = arr[high];  **int** i = (low - 1);  **for** (**int** j = low; j <= high - 1; j++) {  **if** (arr[j] <= pivot) {  i++;  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  **int** temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  **return** i + 1;  }  } |

# Check if it is possible to transform one string to another: - given two strings s1 and s2. Check if it is poosible to convert s1 and s2 by performing following operations. 1. Make some lowercase letters uppercase. 2. Delete all lowercare letters.

|  |
| --- |
| Input : s1 = daBcd s2 = ABC Output : yes  Explanation: daBcd -> dABCd -> ABC Covert a and b at index 1 and 3 to upper case, delete the rest those are lowercase. We get the string s2.  Input : s1 = ABcd s2= BCD Output : NO\*/  **public** **class** Check\_if\_it\_is\_possible\_to\_transform\_one\_string\_to\_another {  **public** **static** **void** main(String[] args) {  String s1 = "daBcd";  String s2 = "ABC";  **if** (*check*(s1, s2))  System.***out***.println("YES");  **else**  System.***out***.println("NO");  }  **static** **boolean** check(String s1, String s2) {  **int** n = s1.length();  **int** m = s2.length();  **boolean** dp[][] = **new** **boolean**[n + 1][m + 1];  **for** (**int** i = 0; i <= n; i++) {  **for** (**int** j = 0; j <= m; j++) {  dp[i][j] = **false**;  }  }  dp[0][0] = **true**;  **for** (**int** i = 0; i < s1.length(); i++) {  **for** (**int** j = 0; j <= s2.length(); j++) {  **if** (dp[i][j]) {  **if** (j < s2.length() && (Character.*toUpperCase*(s1.charAt(i)) == s2.charAt(j)))  dp[i + 1][j + 1] = **true**;  **if** (!Character.*isUpperCase*(s1.charAt(i)))  dp[i + 1][j] = **true**;  }  }  }  **return** (dp[n][m]);  }  } |

# Check plus Perfect Number: - given an ‘n’ digit number x, check if it is a plus perfect number or not.A number is plus perfect number. If it is equal to the sum of its degits raised to the nth power.

|  |
| --- |
| Input : x = 371 Output : Yes Explanation : Number of digits n = 3 (3\*3\*3) + (7\*7\*7) + (1\*1\*1) = 371  Input : x = 9473 Output : No Explanation : Number of digits n = 4 (9\*9\*9\*9) + (4\*4\*4\*4) + (7\*7\*7\*7) + (3\*3\*3\*3) != 9474\*/  **public** **class** Check\_Plus\_Perfect\_Number {  **public** **static** **void** main(String[] args) {  **int** x = 9474;  **if** (*checkplusperfect*(x))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** **boolean** checkplusperfect(**int** x) {  **int** temp = x;  **int** n = 0;  **while** (x != 0) {  x /= 10;  n++;  }  x = temp;  **int** sum = 0;  **while** (x != 0) {  sum += Math.*pow*(x % 10, n);  x /= 10;  }  **return** (sum == temp);  }  } |

# Check Valid Mobile Number: - Program to check valid mobile number.Mobile number validation criteria: 1. the first digit ahould contains number between 7 to 9. 2. The rest 9 digit can contain any number between 0 to 9. 3. The mobile number can have 11 digits also by including 0 at starting. 4. The mobile number can be of 12 digit also by including 91 at the starting.

|  |
| --- |
| Input : Enter Mobile Number: 7873923408 Output :Valid Mobile Number  Input : Enter Mobile Number: 5678729234 Output :Invalid Mobile Number\*/  **public** **class** Check\_Valid\_Mobile\_Number {  **public** **static** **void** main(String[] args) {  String s = "7873923408";  **if** (*isValid*(s))  System.***out***.println("Valid Number");  **else**  System.***out***.println("Invalid Number");  }  **public** **static** **boolean** isValid(String s) {  Pattern p = Pattern.*compile*("(0/91)?[7-9][0-9]{9}");  Matcher m = p.matcher(s);  **return** (m.find() && m.group().equals(s));  }  } |

# Deep Cloning

|  |
| --- |
| **public** **class** CloneDeep **implements** Cloneable {  **public** CloneDeep(String name, Map<Integer, Integer> map) {  **this**.name = name;  **this**.map = map;  }  **private** String name;  **private** Map<Integer, Integer> map;  @Override  **public** CloneDeep clone() {  System.***out***.println("Doing deep copy");  Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();  Iterator<Integer> it = **this**.map.keySet().iterator();  **while** (it.hasNext()) {  Integer key = it.next();  map.put(key, **this**.map.get(key));  }  CloneDeep cloneDetailedDeep = **new** CloneDeep(**new** String(name), map);  **return** cloneDetailedDeep;  }  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();  map.put(1, 11);  CloneDeep obj1 = **new** CloneDeep("sam", map);  CloneDeep obj2 = (CloneDeep) obj1.clone();  System.***out***.println(obj1 == obj2); // false  System.***out***.println(obj1.name == obj2.name); // false  System.***out***.println(obj1.map == obj2.map); // false  }  } |

# Shallow Cloning

|  |
| --- |
| **public** **class** CloneShallow **implements** Cloneable{  **private** String name;  **private** Map<Integer, Integer> map;  **public** CloneShallow(String name, Map<Integer, Integer> map) {  **this**.name = name;  **this**.map = map;  }  @Override  **public** Object clone() {  System.***out***.println("Doing shallow copy");  **try** {  **return** **super**.clone();  } **catch** (CloneNotSupportedException e) {  **return** **null**;  }  }  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();  map.put(11, 11);  CloneShallow obj = **new** CloneShallow("ankit", map);  CloneShallow clonedObj = (CloneShallow) obj.clone();  System.***out***.println(obj == clonedObj); // false  System.***out***.println(obj.name == clonedObj.name); // true  System.***out***.println(obj.map == clonedObj.map); // true  }  } |

# Clone Using Serialization: - program for deep copy using Serialization and Deserialization.

|  |
| --- |
| **public** **class** CloneUsingSerialization {  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();  map.put(1, 11);  Employee emp = **new** Employee("ankit", map);  **try** {  OutputStream fout = **new** FileOutputStream("ser.txt");  ObjectOutput oout = **new** ObjectOutputStream(fout);  System.***out***.println ("Serialization process has started, “+ "serializing employee object...");  oout.writeObject(emp);  fout.close();  oout.close();  System.***out***.println ("employee Serialization completed.");  // DeSerialization process >  InputStream fin = **new** FileInputStream("ser.txt");  ObjectInput oin = **new** ObjectInputStream(fin);  System.***out***.println ("\nDeSerialization process has started, " + "deSerializing employee object...");  Employee deSerializedEmp = (Employee) oin.readObject();  fin.close();  oin.close();  System.***out***.println ("employee DeSerialization completed.");  System.***out***.println(emp == deSerializedEmp); // false  System.***out***.println(emp.getName() == deSerializedEmp.getName()); // false  System.***out***.println(emp.getMap() == deSerializedEmp.getMap()); // false  } **catch** (IOException | ClassNotFoundException e) {  e.printStackTrace();  }  }  }  **class** Employee **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** String name;  **private** Map<Integer, Integer> map;  **public** Employee(String name, Map<Integer, Integer> map) {  **this**.name = name;  **this**.map = map;  }  @Override  **public** String toString() {  **return** "Employee [name=" + name + "]";  }  **public** String getName() {  **return** name;  }  **public** Map<Integer, Integer> getMap() {  **return** map;  }  } |

# Closest number in array: - given an array of sorted integers.We nee to find the closest value to the given number.Array may contain duplicate values and negative numbers.

|  |
| --- |
| Input:arr[]={1,2,4,5,6,6,8,9} Target number=11 Output:9 9 is closest to 11 in given array  Input:arr[]={2,5,6,7,8,8,9}; Target number=4 Output:5  **public** **class** Closestnumberinarray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 4, 5, 6, 6, 8, 9 };  **int** target = 11;  System.***out***.println(*findClosest*(arr, target));  }  **public** **static** **int** findClosest(**int** arr[], **int** target) {  **int** n = arr.length;  **if** (target <= arr[0])  **return** arr[0];  **if** (target >= arr[n - 1])  **return** arr[n - 1];  **int** i = 0, j = n, mid = 0;  **while** (i < j) {  mid = (i + j) / 2;  **if** (arr[mid] == target)  **return** arr[mid];  **if** (target < arr[mid]) {  **if** (mid > 0 && target > arr[mid - 1])  **return** *getClosest*(arr[mid - 1], arr[mid], target);  j = mid;  }  **else** {  **if** (mid < n - 1 && target < arr[mid + 1])  **return** *getClosest*(arr[mid], arr[mid + 1], target);  i = mid + 1;  }  }  **return** arr[mid];  }  **public** **static** **int** getClosest(**int** val1, **int** val2, **int** target) {  **if** (target - val1 >= val2 - target)  **return** val2;  **else**  **return** val2;  }  } |

# Closest numbers from list: - closest numbers from a list of unsorted integers.

|  |
| --- |
| Input : arr[] = {10, 50, 12, 100} Output : (10, 12) The closest elements are 10 and 12  Input : arr[] = {5, 4, 3, 2} Output : (2, 3), (3, 4), (4, 5)  **public** **class** Closestnumbersfromlist {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 3, 2, 4, 1 };  **int** n = arr.length;  *printMinDiffPairs*(arr, n);  }  **static** **void** printMinDiffPairs(**int** arr[], **int** n) {  **if** (n <= 1)  **return**;  Arrays.*sort*(arr);  **int** minDiff = arr[1] - arr[0];  **for** (**int** i = 2; i < n; i++)  minDiff = Math.*min*(minDiff, arr[i] - arr[i - 1]);  **for** (**int** i = 1; i < n; i++) {  **if** ((arr[i] - arr[i - 1]) == minDiff) {  System.***out***.print("(" + arr[i - 1] + ", " + arr[i] + "),");  }  }  }  } |

# Closest Pair: - find the closed pair from two sorted arrays.given two sorted arrays and a number x.find the pair whose sum is closest to x and the pair has an element from each array.

|  |
| --- |
| Input: ar1[] = {1, 4, 5, 7}; ar2[] = {10, 20, 30, 40}; x = 32 Output: 1 and 30  Input: ar1[] = {1, 4, 5, 7}; ar2[] = {10, 20, 30, 40}; x = 50 Output: 7 and 40  **public** **class** ClosestPair {  **public** **static** **void** main(String[] args) {  ClosestPair ob = **new** ClosestPair();  **int** ar1[] = { 1, 4, 5, 7 };  **int** ar2[] = { 10, 20, 30, 40 };  **int** m = ar1.length;  **int** n = ar2.length;  **int** x = 38;  ob.printClosest(ar1, ar2, m, n, x);  }  **void** printClosest(**int** ar1[], **int** ar2[], **int** m, **int** n, **int** x) {  **int** diff = Integer.***MAX\_VALUE***;  **int** res\_l = 0, res\_r = 0;  **int** l = 0, r = n - 1;  **while** (l < m && r >= 0) {  **if** (Math.*abs*(ar1[l] + ar2[r] - x) < diff) {  res\_l = l;  res\_r = r;  diff = Math.*abs*(ar1[l] + ar2[r] - x);  }  **if** (ar1[l] + ar2[r] > x)  r--;  **else**  l++;  }  System.***out***.print("The closest pair is [" + ar1[res\_l] + ", " + ar2[res\_r] + "]");  }  } |

# Coefficient of Variation

|  |
| --- |
| Input : arr[] = {60.25, 62.38, 65.32, 61.41, 63.23} Output : 0.0307144  Input : arr[] = {15, 36, 53.67, 25.45, 67.8, 56, 78.09} Output : 0.48177  **public** **class** CoefficientOfVariation {  **public** **static** **void** main(String[] args) {  **double** arr[] = { 15, 36, 53.67, 25.45, 67.8, 56, 78.09 };  **int** n = arr.length;  System.***out***.println(*coefficientOfVariation*(arr, n));  }  **static** **double** coefficientOfVariation(**double** arr[], **int** n) {  **return** (*standardDeviation*(arr, n) / *mean*(arr, n));  }  **static** **double** standardDeviation(**double** arr[], **int** n) {  **double** sum = 0;  **for** (**int** i = 0; i < n; i++)  sum = sum + (arr[i] - *mean*(arr, n)) \* (arr[i] - *mean*(arr, n));  **return** Math.*sqrt*(sum / (n - 1));  }  **static** **double** mean(**double** arr[], **int** n) {  **double** sum = 0;  **for** (**int** i = 0; i < n; i++)  sum = sum + arr[i];  **return** sum / n;  }  } |

# Cohesion: - Cohesion refers all about how a single class is designed.Cohesion is the object oriented principle most closely associated with making sure that a class is designed with a single.well focused purpose.The more fpopuced a class is the cohensiveness of thet class is more.The advantage of high cohension is that such classes are much easier to maintain tan classes with low cohesion. Another benefits of high cohesion is that classes with a well-fouced purpose tend to be more reusable than other classes.for e.g. Suppose we have a class that multiply two numbers, but the same class creates a pop up window displaying the result. This is the example of low cohesive class because the window and the multiplication operation don’t have much in common.To make it high cohesive, we would have to create a class Display and a class multiply. The Display will call multiply’s method to get the result and display it. This way to develop a high cohesive solution.

|  |
| --- |
| **public** **class** Cohesion {  **public** **static** **void** main(String[] args) {  Multiply m = **new** Multiply();  System.***out***.println(m.mul(5, 5));  }  }  **class** Multiply {  **int** a = 5;  **int** b = 5;  **public** **int** mul(**int** a, **int** b) {  **this**.a = a;  **this**.b = b;  **return** a \* b;  }  } |

# Common Elements in Three Sorted Arrays: - find common elements in three sorted arrays.

|  |
| --- |
| ar1[] = {1, 5, 10, 20, 40, 80} ar2[] = {6, 7, 20, 80, 100} ar3[] = {3, 4, 15, 20, 30, 70, 80, 120} Output: 20, 80  ar1[] = {1, 5, 5} ar2[] = {3, 4, 5, 5, 10} ar3[] = {5, 5, 10, 20} Output: 5, 5  Time complexity is O (n1 + n2 + n3).  **public** **class** CommonElementsInThreeSortedArrays {  **public** **static** **void** main(String[] args) {  CommonElementsInThreeSortedArrays ob = **new** CommonElementsInThreeSortedArrays();  **int** ar1[] = { 1, 5, 10, 20, 40, 80 };  **int** ar2[] = { 6, 7, 20, 80, 100 };  **int** ar3[] = { 3, 4, 15, 20, 30, 70, 80, 120 };  System.***out***.print("Common elements are ");  ob.findCommon(ar1, ar2, ar3);  }  **void** findCommon(**int** ar1[], **int** ar2[], **int** ar3[]) {  **int** i = 0, j = 0, k = 0;  **while** (i < ar1.length && j < ar2.length && k < ar3.length) {  **if** (ar1[i] == ar2[j] && ar2[j] == ar3[k]) {  System.***out***.print(ar1[i] + " ");  i++;  j++;  k++;  }  **else** **if** (ar1[i] < ar2[j])  i++;  **else** **if** (ar2[j] < ar3[k])  j++;  **else**  k++;  }  }  } |

# Compare Two Strings using library function: - the method compareTo () is used for comparing teo strings lexicographically in java.Each character of both the atrings is converted into Unicode value for comparison.

|  |
| --- |
| int compareTo(String str) : It returns the following values:   * if (string1 > string2) it returns a positive value. * if both the strings are equal lexicographically i.e.(string1 == string2) it returns 0. * if (string1 < string2) it returns a negative value.   **public** **class** Compare\_Two\_Strings\_using\_library\_function {  **public** **static** **void** main(String[] args) {  String s1 = "Ram";  String s2 = "Ram";  String s3 = "Shyam";  String s4 = "ABC";  System.***out***.println(" Comparing strings with compareto:");  System.***out***.println(s1.compareTo(s2)); //0  System.***out***.println(s1.compareTo(s3)); //-1  System.***out***.println(s1.compareTo(s4)); //17  }  } |

# Compare Two Strings without library function: - java program to compare two strings lexicographically.

|  |
| --- |
| **public** **class** Compare\_Two\_Strings\_without\_library\_function {  **public** **static** **void** main(String[] args) {  String string1 = **new** String("Geeks");  String string2 = **new** String("Practice");  String string3 = **new** String("Geeks");  System.***out***.println(*stringCompare*(string1, string2)); //-9  System.***out***.println(*stringCompare*(string1, string3)); //0  System.***out***.println(*stringCompare*(string2, string1)); //9  }  **public** **static** **int** stringCompare(String str1, String str2) {  **for** (**int** i = 0; i < str1.length() && i < str2.length(); i++) {  **if** ((**int**) str1.charAt(i) == (**int**) str2.charAt(i)) {  **continue**;  } **else** {  **return** (**int**) str1.charAt(i) - (**int**) str2.charAt(i);  }  }  **if** (str1.length() < str2.length()) {  **return** (**int**) str1.charAt(str2.length());  } **else** **if** (str1.length() > str2.length()) {  **return** (**int**) str1.charAt(str2.length());  }  **else** {  **return** 0;  }  }  } |

# Comparing Enum Members: - An enum can contain constants, member etc.enum can be defined as a group of named constant.There are two ways for making comparison of enum members. 1. By using == operator 2. By using equals () method.

|  |
| --- |
| 1. == operator never throws NullPointerException whereas .equals () method can throw NullPointerException.  2. == is responsible for type compatibility check at compile time whereas. Equals () method will never worry about the types of both the arguments.  **public** **class** Comparing\_Enum\_Members {  **public** **enum** Day {  ***MON***, ***TUE***, ***WED***, ***THU***, ***FRI***, ***SAT***, ***SUN***  }  **public** **static** **void** main(String[] args) {  Day d = **null**;  System.***out***.println(d == Day.***MON***); //false  System.***out***.println(d.equals(Day.***MON***)); // Null pointer exception  }  } |

# Comparision of Double and Float: - size of mantissa for float is 24 and 53 for double.

|  |
| --- |
| **public** **class** Comparision\_Of\_Double\_and\_Float {  **public** **static** **void** main(String[] args) {  **float** f = 5.25f;  **double** d = 5.25;  System.***out***.println(f == d); //true  **float** f1 = 5.1f;  **double** d1 = 5.1;  System.***out***.println(f1 == d1); //false  }  } |

# Concurrent Map Interface

|  |
| --- |
| **public** **class** ConcurrentMapInterface {  **public** **static** **void** main(String[] args) {  ConcurrentHashMap m = **new** ConcurrentHashMap();  m.put(100, "Geeks");  m.put(101, "For");  m.put(102, "Geeks");  m.putIfAbsent(101, "Hello");  m.remove(101, "For");  m.putIfAbsent(101, "Hello");  m.replace(101, "Hello", "For");  System.***out***.println(m);  }  } |

# Convert First Character Uppercase in a Sentence: - java program to convert first character uppercase in a sentence.

|  |
| --- |
| Input : gEEKs Output :Geeks  Input :GFG Output :Gfg  **public** **class** Convert\_First\_Character\_Uppercase\_In\_A\_Sentence {  **public** **static** **void** main(String[] args) {  String str = "gEEks fOr GeeKs";  System.***out***.println(*convert*(str));  }  **static** String convert(String str) {  **char** ch[] = str.toCharArray();  **for** (**int** i = 0; i < str.length(); i++) {  **if** (i == 0 && ch[i] != ' ' || ch[i] != ' ' && ch[i - 1] == ' ') {  **if** (ch[i] >= 'a' && ch[i] <= 'z') {  ch[i] = (**char**) (ch[i] - 'a' + 'A');  }  }  **else** **if** (ch[i] >= 'A' && ch[i] <= 'Z')  ch[i] = (**char**) (ch[i] + 'a' - 'A');  }  String st = **new** String(ch);  **return** st;  }  } |

# Convert string into palindrome: - Convert the string into palindrom string by changing only one character.Given a string str. Check if it is poosible to convert the string into palindrom string by changing only one character.

|  |
| --- |
| Input : str = "abccaa" Output : Yes  Input : str = "abbcca" Output : No  **public** **class** Convert\_string\_into\_palindrome {  **public** **static** **void** main(String[] args) {  String str = "abccaa";  **if** (*checkPalindrome*(str))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** **boolean** checkPalindrome(String str) {  **int** n = str.length();  **int** count = 0;  **for** (**int** i = 0; i < n / 2; ++i)  **if** (str.charAt(i) != str.charAt(n - i - 1))  ++count;  **return** (count <= 1);  }  } |

# Copy Demo Using Array List

|  |
| --- |
| public class CopyDemo\_Using\_ArrayList {  public static void main(String[] args) {  ArrayList l = new ArrayList();  l.add("A");  l.add("B");  l.add("C");  Iterator itr = l.iterator();  while (itr.hasNext()) {  String s = (String) itr.next();  if (s.equals("B")) {  itr.remove();  }  }  System.out.println(l);  }  } |

# Copy Demo Using CopyOnWriteArrayList

|  |
| --- |
| public class CopyDemo\_Using\_CopyOnWriteArrayList extends Thread {  static CopyOnWriteArrayList l = new CopyOnWriteArrayList();  public static void main(String[] args) throws InterruptedException {  l.add("A");  l.add("B");  l.add("C");  Iterator itr = l.iterator();  while (itr.hasNext()) {  String s = (String) itr.next();  System.out.println(s);  if (s.equals("B")) {  // Throws RuntimeException  itr.remove();  }  Thread.sleep(1000);  }  System.out.println(l);  }  } |

# Copy on Write Array Set Demo

|  |
| --- |
| public class CopyOnWriteArraySetDemo extends Thread {  static CopyOnWriteArraySet l = new CopyOnWriteArraySet();  public void run() {  l.add("D");  }  public static void main(String[] args) {  l.add("A");  l.add("B");  l.add("C");  CopyOnWriteArraySetDemo t = new CopyOnWriteArraySetDemo();  t.start();  try {  Thread.sleep(2000);  } catch (InterruptedException e) {  System.out.println("child going to add element");  }  System.out.println(l);  Iterator itr = l.iterator();  while (itr.hasNext()) {  String s = (String) itr.next();  System.out.println(s);  if (s.equals("C")) {  // Here we will get  // RuntimeException  itr.remove();  }  }  }  } |

# Count characters: - Count characters at same position as in English alphabets.

|  |
| --- |
| Input: ABcED Output : 3  Input: geeksforgeeks Output : 1  Input : alphabetical Output : 3  **public** **class** Count\_characters {  **public** **static** **void** main(String[] args) {  String str = "AbgdiF";  System.***out***.println(*findCount*(str));  }  **static** **int** findCount(String str) {  **int** result = 0;  **char**[] c = str.toCharArray();  **for** (**int** i = 0; i <c.length; i++)  **if** (i == (c[i]- 'a') || i == (c[i] - 'A'))  result++;  **return** result;  }  } |

# Count Occurrences of Each Character

|  |
| --- |
| **public** **class** Count\_Occurrences\_Of\_Each\_Character {  **static** **final** **int** ***MAX\_CHAR*** = 256;  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  String str = "javaprogramming";  *getOccuringChar*(str);  }  **static** **void** getOccuringChar(String str) {  **int** count[] = **new** **int**[***MAX\_CHAR***];  **int** len = str.length();  **for** (**int** i = 0; i < len; i++)  count[str.charAt(i)]++;  **char** ch[] = **new** **char**[str.length()];  **for** (**int** i = 0; i < len; i++) {  ch[i] = str.charAt(i);  **int** find = 0;  **for** (**int** j = 0; j <= i; j++) {  **if** (str.charAt(i) == ch[j])  find++;  }  **if** (find == 1)  System.***out***.println("Number of Occurrence of " + str.charAt(i) + " is:" + count[str.charAt(i)]);  }  }  } |

# Count Window Distinct Efficient

|  |
| --- |
| //Time complexity is O(n).  **public** **class** Count\_Window\_Distinct\_Efficient {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 1, 3, 4, 2, 3 };  **int** k = 4;  *countDistinct*(arr, k);  }  **static** **void** countDistinct(**int** arr[], **int** k) {  HashMap<Integer, Integer> hM = **new** HashMap<Integer, Integer>();  **int** dist\_count = 0;  **for** (**int** i = 0; i < k; i++) {  **if** (hM.get(arr[i]) == **null**) {  hM.put(arr[i], 1);  dist\_count++;  } **else** {  **int** count = hM.get(arr[i]);  hM.put(arr[i], count + 1);  }  }  System.***out***.println(dist\_count);  **for** (**int** i = k; i < arr.length; i++) {  **if** (hM.get(arr[i - k]) == 1) {  hM.remove(arr[i - k]);  dist\_count--;  } **else** {  **int** count = hM.get(arr[i - k]);  hM.put(arr[i - k], count - 1);  }  **if** (hM.get(arr[i]) == **null**) {  hM.put(arr[i], 1);  dist\_count++;  } **else** {  **int** count = hM.get(arr[i]);  hM.put(arr[i], count + 1);  }  System.***out***.println(dist\_count);  }  }  } |

# Count Window Distinct Simple: - Count distinct elements in every window of size K. given an array size n and integer k, return the count of distinct numbers in all windowa of size K.

|  |
| --- |
| Input: arr[] = {1, 2, 1, 3, 4, 2, 3}; k = 4 Output: 3 4 4 3 Time complexity is O (nk2).\*/  **public** **class** Count\_Window\_Distinct\_Simple {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 1, 3, 4, 2, 3 }, k = 4;  *countDistinct*(arr, arr.length, k);  }  **static** **void** countDistinct(**int** arr[], **int** n, **int** k) {  **for** (**int** i = 0; i <= n - k; i++)  System.***out***.println(*countWindowDistinct*(Arrays.*copyOfRange*(arr, i, arr.length), k));  }  **static** **int** countWindowDistinct(**int** win[], **int** k) {  **int** dist\_count = 0;  **for** (**int** i = 0; i < k; i++) {  **int** j;  **for** (j = 0; j < i; j++)  **if** (win[i] == win[j])  **break**;  **if** (j == i)  dist\_count++;  }  **return** dist\_count;  }  } |

# Count 1’s in Sorted Binary Array

|  |
| --- |
| Input: arr[] = {1, 1, 0, 0, 0, 0, 0} Output: 2  Input: arr[] = {1, 1, 1, 1, 1, 1, 1} Output: 7  Input: arr[] = {0, 0, 0, 0, 0, 0, 0} Output: 0  Time complexity is O(Logn)  **public** **class** Count1InSortedBinaryArray {  **public** **static** **void** main(String[] args) {  Count1InSortedBinaryArray ob = **new** Count1InSortedBinaryArray();  **int** arr[] = { 1, 1, 1, 1, 0, 0, 0 };  **int** n = arr.length;  System.***out***.println("Count of 1's in given array is " + ob.countOnes(arr, 0, n - 1));  }  **int** countOnes(**int** arr[], **int** low, **int** high) {  **if** (high >= low) {  **int** mid = low + (high - low) / 2;  **if** ((mid == high || arr[mid + 1] == 0) && (arr[mid] == 1))  **return** mid + 1;  **if** (arr[mid] == 1)  **return** countOnes(arr, (mid + 1), high);  **return** countOnes(arr, low, (mid - 1));  }  **return** 0;  }  } |

# Count Down Latch Demo

|  |
| --- |
| **class** Worker **extends** Thread {  **private** **int** delay;  **private** CountDownLatch latch;  **public** Worker(**int** delay, CountDownLatch latch, String name) {  **super**(name);  **this**.delay = delay;  **this**.latch = latch;  }  @Override  **public** **void** run() {  **try** {  Thread.*sleep*(delay);  latch.countDown();  System.***out***.println(Thread.*currentThread*().getName() + " finished");  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  **public** **class** CountDownLatchDemo {  **public** **static** **void** main(String[] args)**throws** InterruptedException {  CountDownLatch latch = **new** CountDownLatch(4);  Worker first = **new** Worker(1000, latch, "WORKER-1");  Worker second = **new** Worker(2000, latch, "WORKER-2");  Worker third = **new** Worker(3000, latch, "WORKER-3");  Worker fourth = **new** Worker(4000, latch, "WORKER-4");  first.start();  second.start();  third.start();  fourth.start();  latch.await();  System.***out***.println(Thread.*currentThread*().getName() + " has finished");  }  } |

# Count of different ways: - count of different ways to express N as the sum of 1, 3, and 4.

|  |
| --- |
| Input : N = 4 Output : 4  Input : N = 5 Output : 6  Time Complexity : O(n) Auxiliary Space : O(n)  **public** **class** Countofdifferentways {  **public** **static** **void** main(String[] args) {  **int** n = 10;  System.***out***.println(*countWays*(n));  }  **static** **int** countWays(**int** n) {  **int** DP[] = **new** **int**[n + 1];  DP[0] = DP[1] = DP[2] = 1;  DP[3] = 2;  **for** (**int** i = 4; i <= n; i++)  DP[i] = DP[i - 1] + DP[i - 3] + DP[i - 4];  **return** DP[n];  }  } |

# Count smaller elements on right side:- write a function to count number of smaller elemants on right of each element in an array.given an unsorted array arr[] of distinct integers,construct another array countSmaller[] such that countSmaller[i] conatins count of smaller elemants on right side of each element arr[i] in array.

|  |
| --- |
| Input: arr[] = {12, 1, 2, 3, 0, 11, 4} Output: countSmaller[] = {6, 1, 1, 1, 0, 1, 0}  Input: arr[] = {1, 2, 3, 4, 5} Output: countSmaller[] = {0, 0, 0, 0, 0}  Time Complexity: O(n^2) Auxiliary Space: O(1)  **public** **class** Countsmallerelementsonrightside {  **public** **static** **void** main(String[] args) {  Countsmallerelementsonrightside small = **new** Countsmallerelementsonrightside();  **int** arr[] = { 12, 10, 5, 4, 2, 20, 6, 1, 0, 2 };  **int** n = arr.length;  **int** low[] = **new** **int**[n];  small.constructLowerArray(arr, low, n);  small.printArray(low, n);  }  **void** constructLowerArray(**int** arr[], **int** countSmaller[], **int** n) {  **int** i, j;  **for** (i = 0; i < n; i++)  countSmaller[i] = 0;  **for** (i = 0; i < n; i++) {  **for** (j = i + 1; j < n; j++) {  **if** (arr[j] < arr[i])  countSmaller[i]++;  }  }  }  **void** printArray(**int** arr[], **int** size) {  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Count Ways to Divide Circle:- count ways to divide circle using N non-intersecting chords.given a number N, find the number of ways you can draw N chords in a circle with 2\*N points such that no 2 chords intersect.

|  |
| --- |
| Input : N = 2 Output : 2 Explanation: If points are numbered 1 to 4 in clockwise direction, then different ways to draw chords are: {(1-2), (3-4)} and {(1-4), (2-3)}  **public** **class** CountWaysToDivideCircle {  **public** **static** **void** main(String[] args) {  **int** N;  N = 2;  System.***out***.println(*chordCnt*(N));  N = 1;  System.***out***.println(*chordCnt*(N));  N = 4;  System.***out***.println(*chordCnt*(N));  }  **static** **int** chordCnt(**int** A) {  **int** n = 2 \* A;  **int**[] dpArray = **new** **int**[n + 1];  dpArray[0] = 1;  dpArray[2] = 1;  **for** (**int** i = 4; i <= n; i += 2) {  **for** (**int** j = 0; j < i - 1; j += 2) {  dpArray[i] += (dpArray[j] \* dpArray[i - 2 - j]);  }  }  **return** dpArray[n];  }  } |

# Loose coupling: - In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to over come from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling.Example : If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. The examples of loose coupling are Interface, JMS.

|  |
| --- |
| **public** **class** Coupling\_Loose {  **public** **static** **void** main(String[] args) {  Box1 b = **new** Box1(5, 5, 5);  System.***out***.println(b.getVolume());  }  }  **final** **class** Box1 {  **private** **int** volume;  Box1(**int** length, **int** width, **int** height) {  **this**.volume = length \* width \* height;  }  **public** **int** getVolume() {  **return** volume;  }  } |

# Tight coupling: - In general, tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.Example: If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI (Remote Method Invocation).

|  |
| --- |
| **public** **class** Coupling\_Tight {  **public** **static** **void** main(String[] args) {  Box b = **new** Box(5, 5, 5);  System.***out***.println(b.volume);  }  }  **class** Box {  **public** **int** volume;  Box(**int** length, **int** width, **int** height) {  **this**.volume = length \* width \* height;  }  } |

# Which is better tight coupling or loose coupling? In general, Tight Coupling is bad in but most of the time, because it reduces flexibility and re-usability of code, it makes changes much more difficult, it impedes test ability etc. loose coupling is a better choice because A loosely coupled will help you when your application need to change or grow. If you design with loosely coupled architecture, only a few parts of the application should be affected when requirements change.

# Difference between tight coupling and loose coupling:-

* Tight coupling is not good at the test-ability. But loose coupling improves the test ability.
* Tight coupling does not provide the concept of interface. But loose coupling helps us follow the GOF principle of program to interfaces, not implementations.
* In Tight coupling, it is not easy to swap the codes between two classes. But it’s much easier to swap other pieces of code/modules/objects/components in loose coupling.
* Tight coupling does not have the changing capability. But loose coupling is highly changeable.

# Cutting Ropes: - ropes left after every removal fo smallest.

|  |
| --- |
| Time Complexity : O(n long (n)) Space complexity : O(1)  Input : Ropes[] = { 5, 1, 1, 2, 3, 5 } Output : 4 3 2  Input : Ropes[] = { 5, 1, 6, 9, 8, 11, 2, 2, 6, 5 } Output : 9 7 5 3 2 1  **public** **class** CuttingRopes {  **public** **static** **void** main(String[] args) {  **int**[] Ropes = { 5, 1, 1, 2, 3, 5 };  **int** n = Ropes.length;  *cuttringRopes*(Ropes, n);  }  **public** **static** **void** cuttringRopes(**int** Ropes[], **int** n) {  Arrays.*sort*(Ropes);  **int** singleOperation = 0;  **int** cuttingLenght = Ropes[0];  **for** (**int** i = 1; i < n; i++) {  **if** (Ropes[i] - cuttingLenght > 0) {  System.***out***.print(n - i + " ");  cuttingLenght = Ropes[i];  singleOperation++;  }  }  **if** (singleOperation == 0)  System.***out***.print("0");  }  } |

# Data Conversion using valueOf () method

|  |
| --- |
| **public** **class** Data\_Conversion\_Using\_valueOf {  **public** **static** **void** main(String[] args) {  **int** iNum = 30;  **double** fNum = 4.56789;  String s = "91";  String sample = String.*valueOf*(iNum);  System.***out***.println(sample);  System.***out***.println(sample + s);  sample = String.*valueOf*(fNum);  System.***out***.println(sample);  System.***out***.println(s + sample);  **char**[] data = { 'G', 'E', 'E', 'K', 'S', ' ', 'F', 'O', 'R', ' ', 'G', 'E', 'E', 'K', 'S' };  sample = String.*valueOf*(data, 0, 15);  System.***out***.println(sample);  String data1 = "Geeks for Geeks";  **boolean** bool = data1.contains("Geeks");  System.***out***.println(String.*valueOf*(bool));  }  } |

# Deep Copy:-

* Whenever we need own copy not to use default implementation we call it as deep copy, whenever we need deep copy of the object we need to implement according to our need.
* So for deep copy we need to ensure all the member class also implement the Cloneable interface and override the clone () method of the object class.

|  |
| --- |
| **public** **class** DeepCopy {  **public** **static** **void** main(String[] args) {  **int**[] vals = { 3, 7, 9 };  Ex1 e = **new** Ex1(vals);  e.showData(); // prints out [3, 7, 9]  vals[0] = 13;  e.showData(); // prints out [3, 7, 9]  }  }  **class** Ex1 {  **private** **int**[] data;  **public** Ex1(**int**[] values) {  data = **new** **int**[values.length];  **for** (**int** i = 0; i < data.length; i++) {  data[i] = values[i];  }  }  **public** **void** showData() {  System.***out***.println(Arrays.*toString*(data));  }  } |

# Shallow Copy:-

* Whenever we use default implementation of clone method we get shallow copy of object means it creates new instance and copies all the field of object to that new instance and returns it as object type, we need to explicitly cast it back to our original object. This is shallow copy of the object.
* Clone () method of the object class support shallow copy of the object. If the object contains primitive as well as nonprimitive or reference type variable in shallow copy, the cloned object also refers to the same object to which the original object refers as only the object references gets copied and not the referred objects themselves.
* That’s why the name shallow copy or shallow cloning in Java. If only primitive type fields or Immutable objects are there then there is no difference between shallow and deep copy in Java.

|  |
| --- |
| **public** **class** ShallowCopy {  **public** **static** **void** main(String[] args) {  **int**[] vals = { 3, 7, 9 };  Ex e = **new** Ex(vals);  e.showData(); // prints out [3, 7, 9]  vals[0] = 13;  e.showData(); // prints out [13, 7, 9]  }  }  **class** Ex {  **private** **int**[] data;  **public** Ex(**int**[] values) {  data = values;  }  **public** **void** showData() {  System.***out***.println(Arrays.*toString*(data));  }  } |

# Delegation: - Delegation is simply passing a duty off to someone/something else.

* Delegation can be an alternative to inheritance.
* Delegation means that you use an object of another class as an instance variable, and forward messages to the instance.
* It is better than inheritance for many cases because it makes you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn’t force you to accept all the methods of the super class: you can provide only the methods that really make sense.
* Delegation can be viewed as a relationship between objects where one object forwards certain method calls to another object, called its delegate.
* The primary advantage of delegation is run-time flexibility – the delegate can easily be changed at run-time. But unlike inheritance, delegation is not directly supported by most popular object-oriented languages, and it doesn’t facilitate dynamic polymorphism.

|  |
| --- |
| **public** **class** Delegation {  **public** **static** **void** main(String[] args) {  Printer printer = **new** Printer();  printer.print();  }  }  **class** RealPrinter {  **void** print() {  System.***out***.println("The Delegate");  }  }  **class** Printer {  RealPrinter p = **new** RealPrinter();  **void** print() {  p.print();  }  } |

# Delete Element from Array Using One Traversals: - move from right to left

|  |
| --- |
| **public** **class** DeleteElementFromArrayUsingOneTraversals {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 11, 15, 6, 8, 9, 10 };  **int** n = arr.length;  **int** x = 6;  n = *deleteElement*(arr, n, x);  System.***out***.println("Modified array is");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **int** deleteElement(**int** arr[], **int** n, **int** x) {  **if** (arr[n - 1] == x) **return** (n - 1);  **int** prev = arr[n - 1], i;  **for** (i = n - 2; i >= 0 && arr[i] != x; i--) {  **int** curr = arr[i];  arr[i] = prev;  prev = curr;  }  **if** (i < 0) **return** 0;  arr[i] = prev;  **return** (n - 1);  }  } |

# Delete Element from Array Using TwoTraversals: - search x in array and move left to right.

|  |
| --- |
| **public** **class** DeleteElementFromArrayUsingTwoTraversals {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 11, 15, 6, 8, 9, 10 };  **int** n = arr.length;  **int** x = 6;  n = *deleteElement*(arr, n, x);  System.***out***.println("Modified array is");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **int** deleteElement(**int** arr[], **int** n, **int** x) {  **int** i;  **for** (i = 0; i < n; i++)  **if** (arr[i] == x)  **break**;  **if** (i < n) {  n = n - 1;  **for** (**int** j = i; j < n; j++)  arr[j] = arr[j + 1];  }  **return** n;  }  } |

# Deadlock: - Deadlock is a situation when two threads are waiting for each other and the waiting is never ends. Here both threads cant completes their tasks.

|  |
| --- |
| **class** MyDeadLock{  String s1 = "Java";  String s2 = "Linux";  Thread t1 = **new** Thread("myThread1"){  **public** **void** run(){  **while**(**true**){  **synchronized**(s1){  **synchronized**(s2){  System.***out***.println(s1+s2);  }  }  }  }  };  Thread t2 = **new** Thread("myThread1"){  **public** **void** run(){  **while**(**true**){  **synchronized**(s1){  **synchronized**(s2){  System.***out***.println(s1+s2);  }  }  }  }  };  }  **public** **class** DeadLockExample {  **public** **static** **void** main(String[] args) {  MyDeadLock md = **new** MyDeadLock();  md.t1.start();  md.t2.start();  }  } |

# Download Web Page

|  |
| --- |
| public class Download\_web\_page {  public static void DownloadWebPage(String webpage) {  try {  URL url = new URL(webpage);  BufferedReader readr = new BufferedReader(new InputStreamReader(url.openStream()));  BufferedWriter writer = new BufferedWriter(new FileWriter("Download.html"));  String line;  while ((line = readr.readLine()) != null) {  writer.write(line);  }  readr.close();  writer.close();  System.out.println ("Successfully Downloaded.");  }  catch (MalformedURLException mue) {  System.out.println("Malformed URL Exception raised");  } catch (IOException ie) {  System.out.println("IOException raised");  }  }  public static void main(String args[]) throws IOException {  String url = "https://www.geeksforgeeks.org/";  DownloadWebPage(url);  }  } |

# Expand a String if range is given

|  |
| --- |
| /\*Input : string x = "1-5, 8, 11-14, 18, 20, 26-29" Output : string y = "1, 2, 3, 4, 5, 8, 11, 12, 13, 14, 18, 20, 26, 27, 28, 29" \*/  **public** **class** Expand\_a\_String\_if\_range\_is\_given {  **public** **static** **void** main(String[] args) {  String s = "1-5, 8, 11-14, 18, 20, 26-29";  *expand*(s);  }  **public** **static** **void** expand(String s) {  String p = s;  String[] arr = p.split("\\-");  String k = "";  **for** (**int** i = 0; i < arr.length; i++) {  **if** (i != arr.length - 1) {  String[] arr1 = arr[i].split(", ");  String[] arr2 = arr[i + 1].split(", ");  **int** a = Integer.*parseInt*(arr1[arr1.length - 1]);  **int** b = Integer.*parseInt*(arr2[0]);  **for** (**int** j = a + 1; j < b; j++) {  arr[i] = arr[i] + ", " + j;  }  }  **if** (k != "")  k = k + ", " + arr[i];  **else**  k = k + arr[i];  }  System.***out***.println(k);  }  **static** String[] split(String st) {  **int** wc = *countWords*(st);  String w[] = **new** String[wc];  **char**[] c = st.toCharArray();  **int** k = 0;  **for** (**int** i = 0; i < c.length; i++) {  String s = "";  **while** (i < c.length && c[i] != ' ') {  s = s + c[i];  i++;  }  **if** (s.length() != 0) {  w[k] = s;  k++;  }  }  **return** w;  }  **static** **int** countWords(String str) {  **int** count = 0;  **for** (**int** i = 0; i < str.length(); i++) {  **if** (i == 0 && str.charAt(i) != ' ' || str.charAt(i) != ' ' && str.charAt(i - 1) == ' ') {  count++;  }  }  **return** count;  }  } |

# Equals IgnoreCase Method:- the equalsIgnoreCase() method compares two strings irrespective of the case(lower or upper) of the string.This methos returns if the arguments is not null and it represents an equivalent string ignoring case, else false.

|  |
| --- |
| Input : str1 = "pAwAn"; str2 = "PAWan" str2.equalsIgnoreCase(str1); Output :true  Input : str1 = "powAn"; str2 = "PAWan" str2.equalsIgnoreCase(str1); Output :false  Explenation: powan and pawan are different strings.  **public** **class** equalsIgnoreCase\_Method {  **public** **static** **void** main(String[] args) {  String str1 = "GeeKS FOr gEEks";  String str2 = "geeKs foR gEEKs";  String str3 = "ksgee orF geeks";  // if we ignore the cases both the strings are equal.  **boolean** result1 = str2.equalsIgnoreCase(str1);  System.***out***.println("str2 is equal to str1 = " + result1);  // even if ignoring the cases both the strings are not equal.  **boolean** result2 = str2.equalsIgnoreCase(str3);  System.***out***.println("str2 is equal to str3 = " + result2);  }  } |

# Equilibrium Index Efficient

|  |
| --- |
| **public** **class** EquilibriumIndex\_Efficient {  **public** **static** **void** main(String[] args) {  EquilibriumIndex\_Efficient equi = **new** EquilibriumIndex\_Efficient();  **int** arr[] = { -7, 1, 5, 2, -4, 3, 0 };  **int** arr\_size = arr.length;  System.***out***.println("First equilibrium index is " + equi.equilibrium(arr, arr\_size));//First equilibrium index is 3  }  **int** equilibrium(**int** arr[], **int** n) {  **int** sum = 0;  **int** leftsum = 0;  **for** (**int** i = 0; i < n; ++i)  sum += arr[i];  **for** (**int** i = 0; i < n; ++i) {  sum -= arr[i];  **if** (leftsum == sum)  **return** i;  leftsum += arr[i];  }  **return** -1;  }  } |

# Equilibrium Index Simple: - Equilibrium index of an array is an index such that sum of elements at lower indexes is equal to the sum of elements at higher indexes.

|  |
| --- |
| elements at higher indexes. A[0] = -7, A[1] = 1, A[2] = 5, A[3] = 2, A[4] = -4, A[5] = 3, A[6]=0  3 is an equilibrium index, because: A[0] + A[1] + A[2] = A[4] + A[5] + A[6]  6 is also an equilibrium index, because sum of zero elements is zero, i.e., A[0] + A[1] + A[2] + A[3] + A[4] + A[5]=0  7 is not an equilibrium index, because it is not a valid index of array A. Time complexity of this solution is O (n^2).  **public** **class** EquilibriumIndex\_Simple {  **public** **static** **void** main(String[] args) {  EquilibriumIndex\_Simple equi = **new** EquilibriumIndex\_Simple();  **int** arr[] = { -7, 1, 5, 2, -4, 3, 0 };  **int** arr\_size = arr.length;  System.***out***.println(equi.equilibrium(arr, arr\_size)); //3  }  **int** equilibrium(**int** arr[], **int** n) {  **int** i, j;  **int** leftsum, rightsum;  **for** (i = 0; i < n; ++i) {  leftsum = 0;  rightsum = 0;  **for** (j = 0; j < i; j++)  leftsum += arr[j];  **for** (j = i + 1; j < n; j++)  rightsum += arr[j];  **if** (leftsum == rightsum)  **return** i;  }  **return** -1;  }  } |

# Externalizable Interface: - Externalizable interface present in java.io, is used for Externalization which extends Serialization interface. It consistes of two methods which we have to override to write/read object into/from stream which are: - 1. To read object from stream: - void readExternal (ObjectInput in) 2. To write object from stream: - void writeExternal (ObjectOutput out)

|  |
| --- |
| **Key differences between Serializable and Externalizable**   * **Implementation:** Unlike Serializable interface which will serialize the variables in object with just by implementing interface, here we have to explicitly mention what fields or variables you want to serialize. * **Methods:** Serializable is marker interface without any methods. Externalizable interface contains two methods:   + WriteExternal () and readExternal (). * **Process:** Default Serialization process will take place for classes implementing Serializable interface. Programmer defined Serialization process for classes implementing Externalizable interface. * **Backward Compatibility and Control:** If you have to support multiple versions, you can have full control with Externalizable interface. You can support different versions of your object. If you implement Externalizable, it’s your responsibility to serialize super class. * **public No-arg constructor:** Serializable uses reflection to construct object and does not require no arg constructor. But Externalizable requires public no-arg constructor.   **public** **class** Externalizable\_Interface {  **public** **static** **void** main(String[] args) {  Car car = **new** Car("Shubham", 1995);  Car newcar = **null**;  // Serialize the car  **try** {  FileOutputStream fo = **new** FileOutputStream("gfg.txt");  ObjectOutputStream so = **new** ObjectOutputStream(fo);  so.writeObject(car);  so.flush();  } **catch** (Exception e) {  System.***out***.println(e);  }  // Deserializa the car  **try** {  FileInputStream fi = **new** FileInputStream("gfg.txt");  ObjectInputStream si = **new** ObjectInputStream(fi);  newcar = (Car) si.readObject();  } **catch** (Exception e) {  System.***out***.println(e);  }  System.***out***.println("The original car is:\n" + car); //Name: Shubham , Year: 1995 , Age: 1995  System.***out***.println("The new car is:\n" + newcar); //Name: Shubham , Year: 10 , Age: 1995  }  }  **class** Car **implements** Externalizable {  **static** **int** *age*;  String name;  **int** year;  **public** Car() {  System.***out***.println("Default Constructor called");  }  Car(String n, **int** y) {  name = n;  year = y;  *age* = 10;  }  **public** **void** writeExternal(ObjectOutput out) **throws** IOException {  out.writeObject(name);  out.writeInt(*age*);  out.writeInt(year);  }  **public** **void** readExternal(ObjectInput in) **throws** IOException, ClassNotFoundException {  name = (String) in.readObject();  year = in.readInt();  *age* = in.readInt();  }  **public** String toString() {  **return** ("Name: " + name + " , Year: " + year + " , Age: " + *age*);  }  }  /\*output  Default Constructor called The original car is: Name: Shubham , Year: 1995 , Age: 1995  The new car is: Name: Shubham , Year: 10 , Age: 1995\*/ |

# Fail Fast Iterators:-Fail-Fast iterators immediately throw ConcurrentModificationException if there is structural modification of the collection.Structural modification means adding, removing or updating any element from collection while a thread is iterating over that collection. Iterator on ArrayList, HashMap classes

|  |
| --- |
| **public** **class** Fail\_Fast\_Iterators {  **public** **static** **void** main(String[] args) {  Map<String, String> cityCode = **new** HashMap<String, String>();  cityCode.put("Delhi", "India");  cityCode.put("Moscow", "Russia");  cityCode.put("New York", "USA");  Iterator iterator = cityCode.keySet().iterator();  **while** (iterator.hasNext()) {  System.***out***.println(cityCode.get(iterator.next()));  cityCode.put("Istanbul", "Turkey");  }  }  }  /\*output India Exception in thread "main" java.util.ConcurrentModificationException\*/ |

# How Fail Fast Iterator works? To know whether the collection is structurally modified or not, fail-fast iterators use an internal flag called modCount which is updated each time a collection is modified.Fail-fast iterators checks the modCount flag whenever it gets the next value (i.e. using next () method), and if it finds that the modCount has been modified after this iterator has been created, it throws ConcurrentModificationException.

# Important points of fail-fast iterators: These iterators throw ConcurrentModificationException if a collection is modified while iterating over it. They use original collection to traverse over the elements of the collection. These iterators don’t require extra memory. Ex: Iterators returned by ArrayList, Vector, HashMap.

# Fail Safe Iterators: - Fail-Safe iterators don’t throw any exceptions if a collection is structurally modified while iterating over it. This is because, they operate on the clone of the collection, not on the original collection and that’s why they are called fail-safe iterators. Iterator on CopyOnWriteArrayList, ConcurrentHashMap classes

|  |
| --- |
| **public** **class** Fail\_Safe\_Iterators {  **public** **static** **void** main(String[] args) {  ConcurrentHashMap<String, Integer> map = **new** ConcurrentHashMap<String, Integer>();  map.put("ONE", 1);  map.put("TWO", 2);  map.put("THREE", 3);  map.put("FOUR", 4);  Iterator it = map.keySet().iterator();  **while** (it.hasNext()) {  String key = (String) it.next();  System.***out***.println(key + " : " + map.get(key));  map.put("SEVEN", 7);  }  }  } |

# Fibonacci Series

|  |
| --- |
| **public** **class** FibonacciSeries {  **public** **static** **void** main(String[] args) {  **int** n =10;  **int** i =0;  **int** j =1;  **int** temp;  System.***out***.println("Fibonacci series " + i + " " + j + " ");  **for**(**int** k = 0 ; k<n ; k++){  temp = i +j;  i = j;  j = temp;  System.***out***.println(temp + " ");  }  }  } |

# Fill Array With1:- fill array with 1’s using minimum iterations of filling neighbors.given an array of 0s and 1s, in how many iterations the whole array can be filled with 1s if in a single iteration immediate neighbors of 1s can be filled. Note: if we cannot fill array with 1s, then print “-1”.

|  |
| --- |
| Input : arr[] = {1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1} Output : 1  Input : arr[] = {0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1} Output : 2 Time Complexity : O(n)  **public** **class** FillArrayWith1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0 };  System.***out***.println(*countIterations*(arr, arr.length));  }  **static** **int** countIterations(**int** arr[], **int** n) {  **boolean** oneFound = **false**;  **int** res = 0;  **for** (**int** i = 0; i < n;) {  **if** (arr[i] == 1) oneFound = **true**;  **while** (i < n && arr[i] == 1)  i++;  **int** count\_zero = 0;  **while** (i < n && arr[i] == 0) {  count\_zero++;  i++;  }  **if** (oneFound == **false** && i == n) **return** -1;  **int** curr\_count;  **if** (i < n && oneFound == **true**) {  **if** ((count\_zero & 1) == 0)  curr\_count = count\_zero / 2;  **else**  curr\_count = (count\_zero + 1) / 2;  count\_zero = 0;  }**else** {  curr\_count = count\_zero;  count\_zero = 0;  }  res = Math.*max*(res, curr\_count);  }  **return** res;  }  } |

# Final Example

|  |
| --- |
| **public** **class** Final\_Example {  **public** **static** **void** main(String[] args) {  **final** StringBuffer sb = **new** StringBuffer("Hello");  sb.append("GFG");  System.***out***.println(sb); //HelloGFG  sb = **new** StringBuffer("Hello World"); //Unresolved compilation problem.The final local variable sb cannot be assigned  System.***out***.println(sb);  }  } |

# Find all the patterns Regular Expression Approach: - find all the patterns of “1(0+) 1” in a given string using regular expression approach.

|  |
| --- |
| Input : 1101001 Output : 2  Input : 100001abc101 Output : 2  **public** **class** Find\_all\_the\_patterns\_Regular\_Expression\_Approach {  **public** **static** **void** main(String[] args) {  String str = "1001ab010abc01001";  System.***out***.println(*patternCount*(str));  }  **static** **int** patternCount(String str) {  String regex = "10+1";  Pattern p = Pattern.*compile*(regex);  Matcher m = p.matcher(str);  **int** counter = 0;  **while** (m.find()) {  m.region(m.end() - 1, str.length());  counter++;  }  **return** counter;  }  } |

# Find all the patterns Simple: - count of occurrences of a “1(0+) 1” pattern in a string.

|  |
| --- |
| Input : 1001010001 Output : 3  Input : 1001ab010abc01001 Output : 2  **public** **class** Find\_all\_the\_patterns\_Simple {  **public** **static** **void** main(String[] args) {  String str = "100001abc101";  System.***out***.println(*countPattern*(str));  }  **static** **int** countPattern(String str) {  **int** len = str.length();  **boolean** oneSeen = **false**;  **int** count = 0;  **for** (**int** i = 0; i < len; i++) {  **char** getChar = str.charAt(i);  **if** (getChar == '1' && oneSeen == **false**)  oneSeen = **true**;  **else** **if** (getChar != '0' && str.charAt(i) != '1')  oneSeen = **false**;  **else** **if** (getChar == '1' && oneSeen == **true**) {  **if** (str.charAt(i - 1) == '0')  count++;  }  }  **return** count;  }  } |

# Find GCD of more than two (or array) numbers

|  |
| --- |
| Input : arr[] = {1, 2, 3} Output : 1  Input : arr[] = {2, 4, 6, 8} Output : 2  **public** **class** find\_GCD\_of\_more\_than\_two\_numbers {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 4, 6, 8, 16 };  **int** n = arr.length;  System.***out***.println(*findGCD*(arr, n));  }  **static** **int** findGCD(**int** arr[], **int** n) {  **int** result = arr[0];  **for** (**int** i = 1; i < n; i++)  result = *gcd*(arr[i], result);  **return** result;  }  **static** **int** gcd(**int** a, **int** b) {  **if** (a == 0)  **return** b;  **return** *gcd*(b % a, a);  }  } |

# Program to find GCD or HCF of two numbers: - GCD or HCF of two numbers is the largest number that divides both of them.

|  |
| --- |
| For example GCD of 20 and 28 is 4 and GCD of 98 and 56 is 14.  **public** **class** find\_GCD\_or\_HCF {  **public** **static** **void** main(String[] args) {  **int** a = 98, b = 56;  System.***out***.println("GCD of " + a + " and " + b + " is " + *gcd*(a, b));  }  **static** **int** gcd(**int** a, **int** b) {  **if** (a == 0 || b == 0)  **return** 0;  **if** (a == b)  **return** a;  **if** (a > b)  **return** *gcd*(a - b, b);  **return** *gcd*(a, b - a);  }  } |

# Given a list of tickets, find itinerary in order using the given list.

|  |
| --- |
| Input:  "Chennai" -> "Banglore"  "Bombay" -> "Delhi"  "Goa" -> "Chennai"  "Delhi" -> "Goa"  Output: Bombay->Delhi, Delhi->Goa, Goa->Chennai, Chennai->Banglore,  time complexity is O(n)  **public** **class** Find\_Itinerary\_From\_Given\_List\_Of\_Tickets {  **public** **static** **void** main(String[] args) {  Map<String, String> dataSet = **new** HashMap<String, String>();  dataSet.put("Chennai", "Banglore");  dataSet.put("Bombay", "Delhi");  dataSet.put("Goa", "Chennai");  dataSet.put("Delhi", "Goa");  *printResult*(dataSet);  }  **private** **static** **void** printResult(Map<String, String> dataSet) {  Map<String, String> reverseMap = **new** HashMap<String, String>();  **for** (Map.Entry<String, String> entry : dataSet.entrySet())  reverseMap.put(entry.getValue(), entry.getKey());  String start = **null**;  **for** (Map.Entry<String, String> entry : dataSet.entrySet()) {  **if** (!reverseMap.containsKey(entry.getKey())) {  start = entry.getKey();  **break**;  }  }  **if** (start == **null**) {  System.***out***.println("Invalid Input");  **return**;  }  String to = dataSet.get(start);  **while** (to != **null**) {  System.***out***.print(start + "->" + to + ", ");  start = to;  to = dataSet.get(to);  }  }  } |

# LCM of given array elements

|  |
| --- |
| Input : {1, 2, 8, 3} Output : 24  Input : {2, 7, 3, 9, 4} Output : 252  **public** **class** Find\_LCM\_of\_given\_array\_elements {  **public** **static** **void** main(String[] args) {  **int**[] element\_array = { 2, 7, 3, 9, 4 };  System.***out***.println(*lcm\_of\_array\_elements*(element\_array));  }  **public** **static** **long** lcm\_of\_array\_elements(**int**[] element\_array) {  **long** lcm\_of\_array\_elements = 1;  **int** divisor = 2;  **while** (**true**) {  **int** counter = 0;  **boolean** divisible = **false**;  **for** (**int** i = 0; i < element\_array.length; i++) {  **if** (element\_array[i] == 0) {  **return** 0;  } **else** **if** (element\_array[i] < 0) {  element\_array[i] = element\_array[i] \* (-1);  }  **if** (element\_array[i] == 1) {  counter++;  }  **if** (element\_array[i] % divisor == 0) {  divisible = **true**;  element\_array[i] = element\_array[i] / divisor;  }  }  **if** (divisible) {  lcm\_of\_array\_elements = lcm\_of\_array\_elements \* divisor;  } **else** {  divisor++;  }  **if** (counter == element\_array.length) {  **return** lcm\_of\_array\_elements;  }  }  }  } |

# Find missing elements of range using Hashing

|  |
| --- |
| Time complexity is O (n + (high-low+1)).  **public** **class** Find\_missing\_elements\_of\_range\_using\_Hashing {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 3, 5, 4 };  **int** low = 1, high = 10;  *printMissing*(arr, low, high);  }  **static** **void** printMissing(**int** ar[], **int** low, **int** high) {  HashSet<Integer> hs = **new** HashSet<>();  **for** (**int** i = 0; i < ar.length; i++)  hs.add(ar[i]);  **for** (**int** i = low; i <= high; i++) {  **if** (!hs.contains(i)) {  System.***out***.print(i + " ");  }  }  }  } |

# Given an array arr [0...n-1] of distinct elements and a range [low, high], find all numbers that are in range, but not in array. The missing elements should be printed in sorted order.

|  |
| --- |
| Input: arr[] = {10, 12, 11, 15}, low = 10, hight = 15 Output: 13, 14  Input: arr[] = {1, 14, 11, 51, 15}, low = 50, hight = 55 Output: 50, 52, 53, 54  Time complexity is O(nLogn + k) where k is number of missing elements  **public** **class** Find\_missing\_elements\_of\_range\_using\_sorting {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 3, 5, 4 };  **int** low = 1, high = 10;  *printMissing*(arr, low, high);  }  **static** **void** printMissing(**int** ar[], **int** low, **int** high) {  Arrays.*sort*(ar);  **int** index = *ceilindex*(ar, low, 0, ar.length - 1);  **int** x = low;  **while** (index < ar.length && x <= high) {  **if** (ar[index] != x) {  System.***out***.print(x + " ");  }  **else**  index++;  x++;  }  **while** (x <= high) {  System.***out***.print(x + " ");  x++;  }  }  **static** **int** ceilindex(**int** ar[], **int** val, **int** low, **int** high) {  **if** (val < ar[0]) **return** 0;  **if** (val > ar[ar.length - 1]) **return** ar.length;  **int** mid = (low + high) / 2;  **if** (ar[mid] == val) **return** mid;  **if** (ar[mid] < val) {  **if** (mid + 1 < high && ar[mid + 1] >= val)  **return** mid + 1;  **return** *ceilindex*(ar, val, mid + 1, high);  } **else** {  **if** (mid - 1 >= low && ar[mid - 1] < val)  **return** mid;  **return** *ceilindex*(ar, val, low, mid - 1);  }  }  } |

# Given a mathematical series as 3, 9, 21, 41, 71… For a given integer n, you have to find the nth number of this series.

|  |
| --- |
| Input : n = 4 Output : 41  Input : n = 2 Output : 9  **public** **class** Find\_nth\_term\_of\_series {  **public** **static** **void** main(String[] args) {  **int** n = 8;  System.***out***.println(*seriesFunc*(n));  n = 13;  System.***out***.println(*seriesFunc*(13));  }  **static** **int** seriesFunc(**int** n) {  **int** sumSquare = (n \* (n + 1) \* (2 \* n + 1)) / 6;  **int** sumNatural = (n \* (n + 1) / 2);  **return** (sumSquare + sumNatural + 1);  }  } |

# Given a dictionary that contains mapping of employee and his manager as a number of (employee, manager) pairs like below.

|  |
| --- |
| { "A", "C" },{ "B", "C" },{ "C", "F" },{ "D", "E" },{ "E", "F" },{ "F", "F" }  In this example C is manager of A, C is also manager of B, F is manager of C and so on.  **public** **class** Find\_number\_of\_Employees\_Under\_every\_Employee {  **static** Map<String, Integer> *result* = **new** HashMap<String, Integer>();  **public** **static** **void** main(String[] args) {  Map<String, String> dataSet = **new** HashMap<String, String>();  dataSet.put("A", "C");  dataSet.put("B", "C");  dataSet.put("C", "F");  dataSet.put("D", "E");  dataSet.put("E", "F");  dataSet.put("F", "F");  *populateResult*(dataSet);  System.***out***.println("result = " + *result*);  }  **private** **static** **void** populateResult(Map<String, String> dataSet) {  Map<String, List<String>> mngrEmpMap = **new** HashMap<String, List<String>>();  **for** (Map.Entry<String, String> entry : dataSet.entrySet()) {  String emp = entry.getKey();  String mngr = entry.getValue();  **if** (!emp.equals(mngr)) {  List<String> directReportList = mngrEmpMap.get(mngr);  **if** (directReportList == **null**)  directReportList = **new** ArrayList<String>();  directReportList.add(emp);  mngrEmpMap.put(mngr, directReportList);  }  }  **for** (String mngr : dataSet.keySet())  *populateResultUtil*(mngr, mngrEmpMap);  }  **private** **static** **int** populateResultUtil(String mngr, Map<String, List<String>> mngrEmpMap) {  **int** count = 0;  **if** (!mngrEmpMap.containsKey(mngr)) {  *result*.put(mngr, 0);  **return** 0;  }  **else** **if** (*result*.containsKey(mngr))  count = *result*.get(mngr);  **else** {  List<String> directReportEmpList = mngrEmpMap.get(mngr);  count = directReportEmpList.size();  **for** (String directReportEmp : directReportEmpList)  count += *populateResultUtil*(directReportEmp, mngrEmpMap);  *result*.put(mngr, count);  }  **return** count;  }  } |

# Find an element using Space Efficient

|  |
| --- |
| **public** **class** Findanelement\_UsingSpaceEfficient {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 4, 1, 4, 5 };  **int** size = arr.length;  System.***out***.println(*findElement*(arr, size));  }  **static** **int** findElement(**int** arr[], **int** size) {  **int** right\_sum = 0, left\_sum = 0;  **for** (**int** i = 1; i < size; i++)  right\_sum += arr[i];  **for** (**int** i = 0, j = 1; j < size; i++, j++) {  right\_sum -= arr[j];  left\_sum += arr[i];  **if** (left\_sum == right\_sum)  **return** arr[i + 1];  }  **return** -1;  }  } |

# Find an element in array such that sum of left array is equal to sum of right array

|  |
| --- |
| Input : 1 4 2 5 Output : 2 Explanation : If 2 is the partition, subarrays are : {1, 4} and {5}  Input : 2 3 4 1 4 5 Output : 1 Explanation : If 1 is the partition, Subarrays are : {2, 3, 4} and {4, 5}  **public** **class** Findanelement\_UsingSuffixPrefix {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 4, 2, 5 };  **int** n = arr.length;  System.***out***.println(*findElement*(arr, n));  }  **static** **int** findElement(**int** arr[], **int** n) {  **int**[] prefixSum = **new** **int**[n];  prefixSum[0] = arr[0];  **for** (**int** i = 1; i < n; i++)  prefixSum[i] = prefixSum[i - 1] + arr[i];  **int**[] suffixSum = **new** **int**[n];  suffixSum[n - 1] = arr[n - 1];  **for** (**int** i = n - 2; i >= 0; i--)  suffixSum[i] = suffixSum[i + 1] + arr[i];  **for** (**int** i = 1; i < n - 1; i++)  **if** (prefixSum[i] == suffixSum[i])  **return** arr[i];  **return** -1;  }  } |

# Find duplicates in O (n) time and O (1) extra space | Set 1 :- let n be 7 and array be {1,2,3,1,3,6,6} the answaer should be 1,3 and 6.

|  |
| --- |
| Time Complexity: O(n) Auxiliary Space: O(1)  **public** **class** FindDuplicates\_Set1 {  **public** **static** **void** main(String[] args) {  FindDuplicates\_Set1 duplicate = **new** FindDuplicates\_Set1();  **int** arr[] = { 4, 2, 4, 5, 2, 3, 1 };  **int** arr\_size = arr.length;  duplicate.printRepeating(arr, arr\_size);  }  **void** printRepeating(**int** arr[], **int** size) {  **int** i;  System.***out***.println("The repeating elements are : ");  **for** (i = 0; i < size; i++) {  **if** (arr[Math.*abs*(arr[i])] >= 0)  arr[Math.*abs*(arr[i])] = -arr[Math.*abs*(arr[i])];  **else**  System.***out***.print(Math.*abs*(arr[i]) + " ");  }  }  } |

# Find the element that appears once in an array where every other element appears twice

|  |
| --- |
| Input: ar[] = {7, 3, 5, 4, 5, 3, 4}; Output: 7  **public** **class** FindElementThatAppearsOnceInAnArray {  **public** **static** **void** main(String[] args) {  **int** ar[] = { 2, 3, 5, 4, 5, 3, 4 };  **int** n = ar.length;  System.***out***.println("Element occurring once is " + *findSingle*(ar, n) + " ");  }  **static** **int** findSingle(**int** ar[], **int** ar\_size) {  **int** res = ar[0];  **for** (**int** i = 1; i < ar\_size; i++)  res = res ^ ar[i];  **return** res;  }  } |

# Find first and last positions Efficient

|  |
| --- |
| use binary search Time Complexity : O(log n) Auxiliary Space : O(1)  **public** **class** Findfirstandlastpositions\_Efficient {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };  **int** n = arr.length;  **int** x = 8;  System.***out***.println("First Occurrence = " + *first*(arr, 0, n - 1, x, n));  System.***out***.println("Last Occurrence = " + *last*(arr, 0, n - 1, x, n));  }  **public** **static** **int** first(**int** arr[], **int** low, **int** high, **int** x, **int** n) {  **if** (high >= low) {  **int** mid = low + (high - low) / 2;  **if** ((mid == 0 || x > arr[mid - 1]) && arr[mid] == x)  **return** mid;  **else** **if** (x > arr[mid])  **return** *first*(arr, (mid + 1), high, x, n);  **else**  **return** *first*(arr, low, (mid - 1), x, n);  }  **return** -1;  }  **public** **static** **int** last(**int** arr[], **int** low, **int** high, **int** x, **int** n) {  **if** (high >= low) {  **int** mid = low + (high - low) / 2;  **if** ((mid == n - 1 || x < arr[mid + 1]) && arr[mid] == x)  **return** mid;  **else** **if** (x < arr[mid])  **return** *last*(arr, low, (mid - 1), x, n);  **else**  **return** *last*(arr, (mid + 1), high, x, n);  }  **return** -1;  }  } |

# Find first and last positions of an element in a sorted array

|  |
| --- |
| Input : arr[] = {1, 3, 5, 5, 5, 5 ,67, 123, 125} x = 5 Output : First Occurrence = 2 Last Occurrence = 5  Input : arr[] = {1, 3, 5, 5, 5, 5 ,7, 123 ,125 } x = 7 Output : First Occurrence = 6 Last Occurrence = 6  Time Complexity : O(n) Auxiliary Space : O(1)  **public** **class** Findfirstandlastpositions\_Simple {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };  **int** x = 8;  *findFirstAndLast*(arr, x);  }  **public** **static** **void** findFirstAndLast(**int** arr[], **int** x) {  **int** n = arr.length;  **int** first = -1, last = -1;  **for** (**int** i = 0; i < n; i++) {  **if** (x != arr[i])  **continue**;  **if** (first == -1)  first = i;  last = i;  }  **if** (first != -1) {  System.***out***.println("First Occurrence = " + first);  System.***out***.println("Last Occurrence = " + last);  } **else**  System.***out***.println("Not Found");  }  } |

# Find Largest Subarray with 0 Sum Hashing

|  |
| --- |
| **public** **class** FindLargestSubarrayWith0SumHashing {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 15, -2, 2, -8, 1, 7, 10, 23 };  System.***out***.println("Length of the longest 0 sum subarray is " + *maxLen*(arr));  }  **static** **int** maxLen(**int** arr[]) {  HashMap<Integer, Integer> hM = **new** HashMap<Integer, Integer>();  **int** sum = 0;  **int** max\_len = 0;  **for** (**int** i = 0; i < arr.length; i++) {  sum += arr[i];  **if** (arr[i] == 0 && max\_len == 0)  max\_len = 1;  **if** (sum == 0)  max\_len = i + 1;  Integer prev\_i = hM.get(sum);  **if** (prev\_i != **null**)  max\_len = Math.*max*(max\_len, i - prev\_i);  **else**  hM.put(sum, i);  }  **return** max\_len;  }  } |

# Given an array of integers, find length of the largest subarray with sum equals to 0.

|  |
| --- |
| Input: arr[] = {15, -2, 2, -8, 1, 7, 10, 23}; Output: 5 The largest subarray with 0 sum is -2, 2, -8, 1, 7  Input: arr[] = {1, 2, 3} Output: 0 There is no subarray with 0 sum  Input: arr[] = {1, 0, 3} Output: 1 Time complexity is O(n2)  **public** **class** FindLargestSubarrayWith0SumSimple {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 15, -2, 2, -8, 1, 7, 10, 23 };  **int** n = arr.length;  System.***out***.println("Length of the longest 0 sum " + "subarray is " + *maxLen*(arr, n));  }  **static** **int** maxLen(**int** arr[], **int** n) {  **int** max\_len = 0;  **for** (**int** i = 0; i < n; i++) {  **int** curr\_sum = 0;  **for** (**int** j = i; j < n; j++) {  curr\_sum += arr[j];  **if** (curr\_sum == 0)  max\_len = Math.*max*(max\_len, j - i + 1);  }  }  **return** max\_len;  }  } |

# Find four elements a, b, c and d in an array such that a+b = c+d: - given an array of distinct intergers, find if there are two pairs (a, b) and (c, d) such that a+b = c+d, and a, b, cand d are distinct elements.If there are multiple answer, then print any of them.

|  |
| --- |
| Input: {3, 4, 7, 1, 2, 9, 8} Output: (3, 8) and (4, 7) Explanation: 3+8 = 4+7  Input: {65, 30, 7, 90, 1, 9, 8}; Output: No pairs found Time complexity is O(n2 Log n)  **public** **class** FindPair {  **class** pair {  **int** first, second;  pair(**int** f, **int** s) {  first = f;  second = s;  }  };  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 4, 7, 1, 2, 9, 8 };  FindPair a = **new** FindPair();  a.findPairs(arr);  }  **boolean** findPairs(**int** arr[]) {  HashMap<Integer, pair> map = **new** HashMap<Integer, pair>();  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i) {  **for** (**int** j = i + 1; j < n; ++j) {  **int** sum = arr[i] + arr[j];  **if** (!map.containsKey(sum))  map.put(sum, **new** pair(i, j));  **else** {  pair p = map.get(sum);  System.***out***.println(  "(" + arr[p.first] + ", " + arr[p.second] + ") and (" + arr[i] + ", " + arr[j] + ")");  **return** **true**;  }  }  }  **return** **false**;  }  } |

# Find position of an element in a sorted array of infinite numbers

|  |
| --- |
| time complexity is 2\*O(Log p)  **public** **class** Findpositionofanelement {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 3, 5, 7, 9, 10, 90, 100, 130, 140, 160, 170 };  **int** ans = *findPos*(arr, 10);  **if** (ans == -1)  System.***out***.println("Element not found");  **else**  System.***out***.println("Element found at index " + ans);  }  **static** **int** findPos(**int** arr[], **int** key) {  **int** l = 0, h = 1;  **int** val = arr[0];  **while** (val < key) {  l = h;  h = 2 \* h;  val = arr[h];  }  **return** *binarySearch*(arr, l, h, key);  }  **static** **int** binarySearch(**int** arr[], **int** l, **int** r, **int** x) {  **if** (r >= l) {  **int** mid = l + (r - l) / 2;  **if** (arr[mid] == x)  **return** mid;  **if** (arr[mid] > x)  **return** *binarySearch*(arr, l, mid - 1, x);  **return** *binarySearch*(arr, mid + 1, r, x);  }  **return** -1;  }  } |

# Find Second Most Frequent Character

|  |
| --- |
| Input: str = "aabababa"; Output: Second most frequent character is 'b'  Input: str = "geeksforgeeks"; Output: Second most frequent character is 'g'  Input: str = "abcd"; Output: No Second most frequent character  //Time complexity of this solution is O (n).  **public** **class** FindSecondMostFrequentCharacter {  **static** **final** **int** ***NO\_OF\_CHARS*** = 256;  **public** **static** **void** main(String[] args) {  String str = "geeksforgeeks";  **char** res = *getSecondMostFreq*(str);  **if** (res != '\0')  System.***out***.println("Second most frequent char" + " is " + res);  **else**  System.***out***.println("No second most frequent" + "character");  }  **static** **char** getSecondMostFreq(String str) {  **int**[] count = **new** **int**[***NO\_OF\_CHARS***];  **int** i;  **for** (i = 0; i < str.length(); i++)  (count[str.charAt(i)])++;  **int** first = 0, second = 0;  **for** (i = 0; i < ***NO\_OF\_CHARS***; i++) {  **if** (count[i] > count[first]) {  second = first;  first = i;  }  **else** **if** (count[i] > count[second] && count[i] != count[first])  second = i;  }  **return** (**char**) second;  }  } |

# Given an array of positive and negative numbers, find if there is a subarray (of size at-least one) with 0 sum.

|  |
| --- |
| Input: {4, 2, -3, 1, 6} Output: true There is a subarray with zero sum from index 1 to 3.  Input: {4, 2, 0, 1, 6} Output: true There is a subarray with zero sum from index 2 to 2.  Input: {-3, 2, 3, 1, 6} Output: false There is no subarray with zero sum.  Time Complexity is O(n)  **public** **class** FindSubarrayWith0Sum {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 4, 2, -3, 1, 6 };  **if** (*printZeroSumSubarray*(arr))  System.***out***.println("Found a subarray with 0 sum");  **else**  System.***out***.println("No Subarray with 0 sum");  }  **static** Boolean printZeroSumSubarray(**int** arr[]) {  HashMap<Integer, Integer> hM = **new** HashMap<Integer, Integer>();  **int** sum = 0;  **for** (**int** i = 0; i < arr.length; i++) {  sum += arr[i];  **if** (arr[i] == 0 || sum == 0 || hM.get(sum) != **null**)  **return** **true**;  hM.put(sum, i);  }  **return** **false**;  }  } |

# Find the index of first 1 in an infinite sorted array of 0s and 1s :- given an infinite sorted array consisting 0s and 1s.the problem is to find the index of first ‘1’ in that array.as the array is infinite,therefore it is guaranteed that number’1’ will be present in the array.

|  |
| --- |
| Input : arr[] = {0, 0, 1, 1, 1, 1} Output : 2  Input : arr[] = {1, 1, 1, 1,, 1, 1} Output : 0 time complexity is 2\*O(Log p)  **public** **class** FindTheIndex {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 0, 0, 1, 1, 1, 1 };  System.***out***.println("Index = " + *posOfFirstOne*(arr));  }  **public** **static** **int** posOfFirstOne(**int** arr[]) {  **int** l = 0, h = 1;  **while** (arr[h] == 0) {  l = h;  h = 2 \* h;  }  **return** *indexOfFirstOne*(arr, l, h);  }  **public** **static** **int** indexOfFirstOne(**int** arr[], **int** low, **int** high) {  **int** mid = 0;  **while** (low <= high) {  mid = (low + high) / 2;  **if** (arr[mid] == 1 && (mid == 0 || arr[mid - 1] == 0))  **break**;  **else** **if** (arr[mid] == 1)  high = mid - 1;  **else**  low = mid + 1;  }  **return** mid;  }  } |

# Find Triplet Use Sorting

|  |
| --- |
| **public** **class** FindTripletUseSorting {  **public** **static** **void** main(String[] args) {  FindTripletUseSorting triplet = **new** FindTripletUseSorting();  **int** A[] = { 1, 4, 45, 6, 10, 8 };  **int** sum = 22;  **int** arr\_size = A.length;  triplet.find3Numbers(A, arr\_size, sum);  }  **boolean** find3Numbers(**int** A[], **int** arr\_size, **int** sum) {  **int** l, r;  quickSort(A, 0, arr\_size - 1);  **for** (**int** i = 0; i < arr\_size - 2; i++) {  l = i + 1;  r = arr\_size - 1;  **while** (l < r) {  **if** (A[i] + A[l] + A[r] == sum) {  System.***out***.print("Triplet is " + A[i] + " ," + A[l] + " ," + A[r]);  **return** **true**;  } **else** **if** (A[i] + A[l] + A[r] < sum)  l++;  **else**  r--;  }  }  **return** **false**;  }  **void** quickSort(**int** A[], **int** si, **int** ei) {  **int** pi;  **if** (si < ei) {  pi = partition(A, si, ei);  quickSort(A, si, pi - 1);  quickSort(A, pi + 1, ei);  }  }  **int** partition(**int** A[], **int** si, **int** ei) {  **int** x = A[ei];  **int** i = (si - 1);  **int** j;  **for** (j = si; j <= ei - 1; j++) {  **if** (A[j] <= x) {  i++;  **int** temp = A[i];  A[i] = A[j];  A[j] = temp;  }  }  **int** temp = A[i + 1];  A[i + 1] = A[ei];  A[ei] = temp;  **return** (i + 1);  }  } |

# Find a triplet that sum to a given value: - given an array and a value, find if there is a triplet in array whose sum is equal to the given value.if there is such a triplet present in array, then print the triplet and return true.else return false. For e.g. if the given array is {12, 3, 4, 1, 6, 9} and given sum is 24, then there is a triplet (12, 3, and 9) present in array whose sum is 24.

|  |
| --- |
| Time Complexity: O(n^3)  **public** **class** FindTripletUsingSimple {  **public** **static** **void** main(String[] args) {  FindTripletUsingSimple triplet = **new** FindTripletUsingSimple();  **int** A[] = { 1, 4, 45, 6, 10, 8 };  **int** sum = 22;  **int** arr\_size = A.length;  triplet.find3Numbers(A, arr\_size, sum);  }  **boolean** find3Numbers(**int** A[], **int** arr\_size, **int** sum) {  **int** l, r;  **for** (**int** i = 0; i < arr\_size - 2; i++) {  **for** (**int** j = i + 1; j < arr\_size - 1; j++) {  **for** (**int** k = j + 1; k < arr\_size; k++) {  **if** (A[i] + A[j] + A[k] == sum) {  System.***out***.print("Triplet is " + A[i] + " ," + A[j] + " ," + A[k]);  **return** **true**;  }  }  }  }  **return** **false**;  }  } |

# First Repeating Element in an Array

|  |
| --- |
| Input: arr[] = {10, 5, 3, 4, 3, 5, 6} Output: 5 [5 is the first element that repeats]  Input: arr[] = {6, 10, 5, 4, 9, 120, 4, 6, 10} Output: 6 [6 is the first element that repeats]  //Time Complexity of this solution is O(n)  **public** **class** FirstRepeatingElementInAnArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 5, 3, 4, 3, 5, 6 };  *printFirstRepeating*(arr);  }  **static** **void** printFirstRepeating(**int** arr[]) {  **int** min = -1;  HashSet<Integer> set = **new** HashSet<>();  **for** (**int** i = arr.length - 1; i >= 0; i--) {  **if** (set.contains(arr[i]))  min = i;  **else**  set.add(arr[i]);  }  **if** (min != -1)  System.***out***.println("The first repeating element is " + arr[min]);  **else**  System.***out***.println("There are no repeating elements");  }  } |

# Frequency of Sorted Array

|  |
| --- |
| **public** **class** FrequencyOfSortedArray {  **static** HashMap<Integer, Integer> *hm* = **new** HashMap<Integer, Integer>();  **public** **static** **void** main(String[] args) {  **int** a[] = { 1, 3, 2, 4, 2, 1 };  **int** n = a.length;  *countFreq*(a, n);  System.***out***.println(*query*(2));  System.***out***.println(*query*(3));  System.***out***.println(*query*(5));  }  **static** **int** query(**int** x) {  **if** (*hm*.containsKey(x))  **return** *hm*.get(x);  **return** 0;  }  **static** **void** countFreq(**int** a[], **int** n) {  **for** (**int** i = 0; i < n; i++)  **if** (*hm*.containsKey(a[i]))  *hm*.put(a[i], *hm*.get(a[i]) + 1);  **else**  *hm*.put(a[i], 1);  }  } |

# Frequency of Unsorted Array

|  |
| --- |
| Input : a[] = {0, 5, 5, 5, 4} x = 5 Output : 3  Input : a[] = {1, 2, 3} x = 4 Output : 0  Time Complexity : O(n) Auxiliary Space : O(1)  **public** **class** FrequencyOfUnsortedArray {  **public** **static** **void** main(String[] args) {  **int** a[] = { 0, 5, 5, 5, 4 };  **int** x = 5;  **int** n = a.length;  System.***out***.println(*frequency*(a, n, x));  }  **static** **int** frequency(**int** a[], **int** n, **int** x) {  **int** count = 0;  **for** (**int** i = 0; i < n; i++)  **if** (a[i] == x)  count++;  **return** count;  }  } |

# Generics Type Erasure

|  |
| --- |
| **public** **class** GenericsTypeErasure {  **public** **static** **void** main(String[] args) {  List<String> list = **new** ArrayList<String>();  list.add("Hello");  Iterator<String> iter = list.iterator();  **while** (iter.hasNext()) {  String s = iter.next();  System.***out***.println(s);  }  }  } |

# Get list of files from FTPserver

|  |
| --- |
| **public** **class** Get\_list\_of\_files\_from\_FTPserver {  **public** **static** **void** main(String[] args) {  FTPClient ftp = **new** FTPClient();  **try** {  ftp.connect("ftp.test.com");  **boolean** isSuccess = ftp.login("user", "password");  **if** (isSuccess) {  String[] filesFTP = ftp.listNames();  **int** count = 1;  **for** (String file : filesFTP) {  System.***out***.println("File " + count + " :" + file);  count++;  }  }  ftp.logout();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  ftp.disconnect();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# gOOGLE cASE of given sentence efficient

|  |
| --- |
| **public** **class** gOOGLE\_cASE\_of\_given\_sentence\_efficient {  **public** **static** **void** main(String[] args) {  String str = "I get intern at geeksforgeeks";  System.***out***.println(*convert*(str));  }  **static** String convert(String s) {  **int** n = s.length();  String s1 = "";  s1 = s1 + Character.*toLowerCase*(s.charAt(0));  **for** (**int** i = 1; i < n; i++) {  **if** (s.charAt(i) == ' ' && i < n) {  s1 = s1 + " " + Character.*toLowerCase*(s.charAt(i + 1));  i++;  }  **else**  s1 = s1 + Character.*toUpperCase*(s.charAt(i));  }  **return** s1;  }  } |

# Given a sentence, task is to rewrite in Google Case. It is a style of writing where we replace all lower case letters into upper case letters leaving the initial of all the words.

|  |
| --- |
| Input : gEEks fOr GeeKs Output : gEEKS fOR gEEKS  Input : I got intern at geeksforgeeks Output : i gOT iNTERN aT gEEKSFORGEEKS  **public** **class** gOOGLE\_cASE\_of\_given\_sentence\_simple {  **public** **static** **void** main(String[] args) {  String str = "I Got Intern At Geeksforgeeks";  System.***out***.println(*convert*(str));  }  **static** String convert(String str) {  String w = "", z = "";  str = str.toUpperCase() + " ";  **for** (**int** i = 0; i < str.length(); i++) {  **char** ch = str.charAt(i);  **if** (ch != ' ')  w = w + ch;  **else** {  z = z + (Character.*toLowerCase*(w.charAt(0))) + w.substring(1) + " ";  w = "";  }  }  **return** z;  }  } |

# Given a list of words with lower cases. Implement a function to find all Words that have the same unique character set.

|  |
| --- |
| Input: words[] = { "may", "student", "students", "dog","studentssess", "god", "cat", "act", "tab", "bat", "flow", "wolf", "lambs","amy", "yam", "balms", "looped", "poodle"};  Output : looped, poodle, lambs, balms, flow, wolf, tab, bat, may, amy, yam, student, students, studentssess, dog, god, cat, act, Time complexity : O(n\*k)  **public** **class** Group\_words\_with\_same\_set\_of\_characters {  **static** **final** **int** ***MAX\_CHAR*** = 26;  **public** **static** **void** main(String[] args) {  String words[] = { "may", "student", "students", "dog", "studentssess", "god", "cat", "act", "tab", "bat",  "flow", "wolf", "lambs", "amy", "yam", "balms", "looped", "poodle" };  **int** n = words.length;  *wordsWithSameCharSet*(words, n);  }  **static** **void** wordsWithSameCharSet(String words[], **int** n) {  HashMap<String, ArrayList<Integer>> Hash = **new** HashMap<>();  **for** (**int** i = 0; i < n; i++) {  String key = *getKey*(words[i]);  **if** (Hash.containsKey(key)) {  ArrayList<Integer> get\_al = Hash.get(key);  get\_al.add(i);  Hash.put(key, get\_al);  }  **else** {  ArrayList<Integer> new\_al = **new** ArrayList<>();  new\_al.add(i);  Hash.put(key, new\_al);  }  }  **for** (Entry<String, ArrayList<Integer>> it : Hash.entrySet()) {  ArrayList<Integer> get = it.getValue();  **for** (Integer v : get)  System.***out***.print(words[v] + ", ");  System.***out***.println();  }  }  **static** String getKey(String str) {  **boolean**[] visited = **new** **boolean**[***MAX\_CHAR***];  Arrays.*fill*(visited, **false**);  **for** (**int** j = 0; j < str.length(); j++)  visited[str.charAt(j) - 'a'] = **true**;  String key = "";  **for** (**int** j = 0; j < ***MAX\_CHAR***; j++)  **if** (visited[j])  key = key + (**char**) ('a' + j);  **return** key;  }  } |

# Group all occurrences of characters according to first appearance: - given a string of lowercase, the task is to print the string in a manner such that a character comes first in string displays fisrt with all its accurrences in string.

|  |
| --- |
| Input : str = "geeksforgeeks" Output: ggeeeekkssfor  Explanation: In the given string 'g' comes first and occurs 2 times so it is printed first Then 'e' comes in this string and 4 times so it gets printed. Similarly remaining string is printed.  Input : str = "occurrence" output : occcurreen  Input : str = "cdab" Output : cdab  **public** **class** GroupAllOccurrencesOfCharacters {  **static** **final** **int** ***MAX\_CHAR*** = 26;  **public** **static** **void** main(String[] args) {  String str = **new** String("geeksforgeeks");  *printGrouped*(str);  }  **static** **void** printGrouped(String str) {  **int** n = str.length();  **int** count[] = **new** **int**[***MAX\_CHAR***];  **for** (**int** i = 0; i < n; i++)  count[str.charAt(i) - 'a']++;  **for** (**int** i = 0; i < n; i++) {  **while** (count[str.charAt(i) - 'a'] != 0) {  System.***out***.print(str.charAt(i));  count[str.charAt(i) - 'a']--;  }  count[str.charAt(i) - 'a'] = 0;  }  }  } |

# Group multiple occurrence of array elements ordered by first occurrence: - given an unsorted array with repetitions, the task is to group multiple occurrence of individual elements.the grouping should happen in a way that the order of first occurrences of all elements is maintained.

|  |
| --- |
| Input: arr[] = {5, 3, 5, 1, 3, 3} Output: {5, 5, 3, 3, 3, 1}  Input: arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4} Output: {4, 4, 4, 6, 6, 9, 9, 2, 3, 10}  Time Complexity is Θ(n)  **public** **class** GroupMultipleOccurrenceOfArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 5, 3, 10, 10, 4, 1, 3 };  *orderedGroup*(arr);  }  **static** **void** orderedGroup(**int** arr[]) {  HashMap<Integer, Integer> hM = **new** HashMap<Integer, Integer>();  **for** (**int** i = 0; i < arr.length; i++) {  Integer prevCount = hM.get(arr[i]);  **if** (prevCount == **null**)  prevCount = 0;  hM.put(arr[i], prevCount + 1);  }  **for** (**int** i = 0; i < arr.length; i++) {  Integer count = hM.get(arr[i]);  **if** (count != **null**) {  **for** (**int** j = 0; j < count; j++)  System.***out***.print(arr[i] + " ");  hM.remove(arr[i]);  }  }  }  } |

# Hashcode Method in Java with HashCode

|  |
| --- |
| **public** **class** Hashcode\_Method\_In\_Java\_With\_HashCode {  **private** **final** String first, last;  **public** **static** **void** main(String[] args) {  Set<Hashcode\_Method\_In\_Java\_With\_HashCode> s = **new** HashSet<Hashcode\_Method\_In\_Java\_With\_HashCode>();  s.add(**new** Hashcode\_Method\_In\_Java\_With\_HashCode("Shubham", "Juneja"));  System.***out***.println(s.contains(**new** Hashcode\_Method\_In\_Java\_With\_HashCode("Shubham", "Juneja")));  }  **public** Hashcode\_Method\_In\_Java\_With\_HashCode(String first, String last) {  **this**.first = first;  **this**.last = last;  }  **public** **boolean** equals(Object o) {  **if** (!(o **instanceof** Hashcode\_Method\_In\_Java\_With\_HashCode))  **return** **false**;  Hashcode\_Method\_In\_Java\_With\_HashCode n = (Hashcode\_Method\_In\_Java\_With\_HashCode) o;  **return** n.first.equals(first) && n.last.equals(last);  }  **public** **int** hashCode() {  **return** 63 \* first.hashCode() + last.hashCode();  }  } |

# Hashcode Method in Java without HashCode

|  |
| --- |
| Importance of Hashcode method in Java: - HashMap and HashSet use hashing to manipulate data. They use hashCode () method to check hash values. The default implementation of hashCode () in Object class returns distinct integers for different objects.  **public** **class** Hashcode\_Method\_In\_Java\_Without\_HashCode {  **private** **final** String first, last;  **public** **static** **void** main(String[] args) {  Set<Hashcode\_Method\_In\_Java\_Without\_HashCode> s = **new** HashSet<Hashcode\_Method\_In\_Java\_Without\_HashCode>();  s.add(**new** Hashcode\_Method\_In\_Java\_Without\_HashCode("Shubham", "Juneja"));  System.***out***.println(s.contains(**new** Hashcode\_Method\_In\_Java\_Without\_HashCode("Shubham", "Juneja")));  }  **public** Hashcode\_Method\_In\_Java\_Without\_HashCode(String first, String last) {  **this**.first = first;  **this**.last = last;  }  **public** **boolean** equals(Object o) {  **if** (!(o **instanceof** Hashcode\_Method\_In\_Java\_Without\_HashCode))  **return** **false**;  Hashcode\_Method\_In\_Java\_Without\_HashCode n = (Hashcode\_Method\_In\_Java\_Without\_HashCode) o;  **return** n.first.equals(first) && n.last.equals(last);  }  } |

# Hash Map Demo Using Concurrent Hash Map

|  |
| --- |
| **public** **class** HashMapDemoUsing\_ConcurrentHashMap **extends** Thread {  **static** ConcurrentHashMap<Integer, String> *l* = **new** ConcurrentHashMap<Integer, String>();  **public** **void** run() {  *l*.put(103, "D");  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  System.***out***.println("Child Thread going to add element");  }  }  **public** **static** **void** main(String[] args) **throws** InterruptedException {  *l*.put(100, "A");  *l*.put(101, "B");  *l*.put(102, "C");  HashMapDemoUsing\_ConcurrentHashMap t = **new** HashMapDemoUsing\_ConcurrentHashMap();  t.start();  **for** (Object o : *l*.entrySet()) {  Object s = o;  System.***out***.println(s);  Thread.*sleep*(1000);  }  System.***out***.println(*l*);  }  } |

# Hash Map Demo Using Hash Map

|  |
| --- |
| **public** **class** HashMapDemoUsing\_HashMap **extends** Thread {  **public** **static** **void** main(String[] args) **throws** InterruptedException {  *l*.put(100, "A");  *l*.put(101, "B");  *l*.put(102, "C");  HashMapDemoUsing\_HashMap t = **new** HashMapDemoUsing\_HashMap();  t.start();  **for** (Object o : *l*.entrySet()) {  Object s = o;  System.***out***.println(s);  Thread.*sleep*(1000);  }  System.***out***.println(*l*);  }  **static** HashMap<Integer, String> *l* = **new** HashMap<Integer, String>();  **public** **void** run() {  *l*.put(103, "D");  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException e) {  System.***out***.println("Child Thread going to add element");  }  }  } |

# Create a set and using .add () method we add the elements into the set

|  |
| --- |
| **public** **class** HashSet\_Using\_add {  **public** **static** **void** main(String[] args) {  Set<Integer> set = **new** HashSet<Integer>();  set.add(1);  set.add(2);  set.add(3);  set.add(4);  set.add(5);  set.add(6);  set.add(7);  set.add(8);  set.add(3);  System.***out***.println(set); // [1, 2, 3, 4, 5, 6, 7, 8]  }  } |

# Collections class consists of several methods that operate on collections.

|  |
| --- |
| a) Collection.addAll (): adds all the specified elements to the specified collection of the specified type.  b) Collections.unmodifiableSet (): adds the elements and returns an unmodifiable view of the specified set.  **public** **class** HashSet\_Using\_Collections {  **public** **static** **void** main(String[] args) {  Integer arr[] = { 5, 6, 7, 8, 1, 2, 3, 4, 3 };  Set<Integer> set = Collections.<Integer>*emptySet*();  Collections.*addAll*(set = **new** HashSet<Integer>(Arrays.*asList*(arr)));  Set<Integer> set2 = Collections.*unmodifiableSet*(**new** HashSet<Integer>(Arrays.*asList*(arr)));  System.***out***.println(set); // [1, 2, 3, 4, 5, 6, 7, 8]  }  } |

# HashSet Using Constructor: - in this method first we create an array then convert it to a list and then pass it to the HashSet constructor that accepts another collection.

|  |
| --- |
| **public** **class** HashSet\_Using\_Constructor {  **public** **static** **void** main(String[] args) {  Integer arr[] = { 5, 6, 7, 8, 1, 2, 3, 4, 3 };  Set<Integer> set = **new** HashSet<>(Arrays.*asList*(arr));  System.***out***.println(set); // [1, 2, 3, 4, 5, 6, 7, 8]  }  } |

# Helper Class: - Helper class is a java class which includes basic error handling. helper class contains functions that help in assisting the progem .this class intends to give quick implementation of basic functions such that programmers do not have to implement again and again.it is easy to access as all the member functions are static that is it can be accessed from anywhere.it implements the most commonly used functions excluding the functions that are already present in java library.

|  |
| --- |
| **public** **class** HelperClass {  **public** **static** **void** main(String[] args) {  **int** n = -5;  **if** (Test.*isValidInteger*(n))  System.***out***.println("True");  **else**  System.***out***.println("False");  String str = "madam";  **if** (Test.*isPalindrome*(str))  System.***out***.println("True");  **else**  System.***out***.println("False");  }  }  **class** Test {  **public** **static** **boolean** isValidInteger(**int** test) {  **return** (test >= 0);  }  **public** **static** **boolean** isValidInteger(**int** test, **int** low, **int** high) {  **return** (test >= low && test <= high);  }  **public** **static** **int** getInRange(**int** low) {  Scanner sc = **new** Scanner(System.***in***);  **int** test;  **do** {  test = sc.nextInt();  } **while** (test < low);  **return** test;  }  **public** **static** **int** getInRange(**int** low, **int** high) {  Scanner sc = **new** Scanner(System.***in***);  **int** test;  **do** {  test = sc.nextInt();  } **while** (test < low || test > high);  **return** test;  }  **public** **static** **boolean** validatePositiveArray(**int**[] array, **int** n) {  **for** (**int** i = 0; i < n; i++)  **if** (array[i] < 0)  **return** **false**;  **return** **true**;  }  **public** **static** **boolean** validateNegativeArray(**int**[] array, **int** n) {  **for** (**int** i = 0; i < n; i++)  **if** (array[i] > 0)  **return** **false**;  **return** **true**;  }  **public** **static** **boolean** checkRangeArray(**int**[] array, **int** n, **int** low) {  **for** (**int** i = 0; i < n; i++)  **if** (array[i] < low)  **return** **false**;  **return** **true**;  }  **public** **static** **boolean** checkRangeArray(**int**[] array, **int** n, **int** low, **int** high) {  **for** (**int** i = 0; i < n; i++)  **if** (array[i] < low || array[i] > high)  **return** **false**;  **return** **true**;  }  **public** **static** **boolean** isEqualSets(**int**[] array1, **int** n, **int**[] array2, **int** m) {  **if** (n != m)  **return** **false**;  HashMap<Integer, Integer> Map = **new** HashMap<Integer, Integer>();  **for** (**int** i = 0; i < n; i++)  Map.put(**new** Integer(array1[i]), **new** Integer(1));  **for** (**int** i = 0; i < m; i++)  Map.put(**new** Integer(array2[i]), **new** Integer(0));  **for** (**int** i = 0; i < n; i++)  **if** (Map.get(array1[i]) == 1)  **return** **false**;  **return** **true**;  }  **public** **static** String factorial(**int** n) {  String fact = **new** String("");  **int** res[] = **new** **int**[500];  res[0] = 1;  **int** res\_size = 1;  **for** (**int** x = 2; x <= n; x++)  res\_size = *multiply*(x, res, res\_size);  **for** (**int** i = res\_size - 1; i >= 0; i--)  fact += Integer.*toString*(res[i]);  **return** fact;  }  **public** **static** **int** multiply(**int** x, **int** res[], **int** res\_size) {  **int** carry = 0;  **for** (**int** i = 0; i < res\_size; i++) {  **int** prod = res[i] \* x + carry;  res[i] = prod % 10;  carry = prod / 10;  }  **while** (carry != 0) {  res[res\_size] = carry % 10;  carry = carry / 10;  res\_size++;  }  **return** res\_size;  }  **public** **static** **boolean** isPrime(**int** n) {  **if** (n == 2)  **return** **true**;  **int** squareRoot = (**int**) Math.*sqrt*(n);  **for** (**int** i = 1; i <= squareRoot; i++)  **if** (n % i == 0 && i != 1)  **return** **false**;  **return** **true**;  }  **public** **static** **int** nthPrimeNumber(**int** n) {  **int** k = 0;  **for** (**int** i = 2;; i++) {  **if** (*isPrime*(i))  k++;  **if** (k == n)  **return** i;  }  }  **public** **static** **boolean** isPalindrome(String test) {  **int** length = test.length();  **for** (**int** i = 0; i <= (test.length()) / 2; i++)  **if** (test.charAt(i) != test.charAt(length - i - 1))  **return** **false**;  **return** **true**;  }  **public** **static** **boolean** isAnagram(String s1, String s2) {  String copyOfs1 = s1.replaceAll("\\s", "");  String copyOfs2 = s2.replaceAll("\\s", "");  **if** (copyOfs1.length() != copyOfs2.length())  **return** **false**;  **char**[] s1Array = copyOfs1.toLowerCase().toCharArray();  **char**[] s2Array = copyOfs2.toLowerCase().toCharArray();  Arrays.*sort*(s1Array);  Arrays.*sort*(s2Array);  **return** (Arrays.*equals*(s1Array, s2Array));  }  } |

# Initialization blocks and Constructors

|  |
| --- |
| **Instance Initialization Blocks:** IIB are used to initialize instance variables. IIBs are executed before constructors.  They run each time when object of the class is created.  **Initializer block:** contains the code that is always executed whenever an instance is created. It is used to declare/initialize the common part of various constructors of a class.  **Constructors:** are used to initialize the object’s state. Like methods, a constructor also contains collection of statements (i.e. instructions) that are executed at time of Object creation.  **Order of execution of Initialization blocks and constructor in Java**  1.Static initialization blocks will run whenever the class is loaded first time in JVM  2. Initialization blocks run in the same order in which they appear in the program.  3. Instance Initialization blocks are executed whenever the class is initialized and before constructors are invoked. They are typically placed above the constructors within braces.  **public** **class** Initialization\_blocks\_and\_Constructors {  **public** **static** **void** main(String[] args) {  **new** GFG();  **new** GFG(8);  }  }  **class** GFG {  GFG(**int** x) {  System.***out***.println("ONE argument constructor");  }  GFG() {  System.***out***.println("No argument constructor");  }  **static** {  System.***out***.println("1st static init");  }  {  System.***out***.println("1st instance init");  }  {  System.***out***.println("2nd instance init");  }  **static** {  System.***out***.println("2nd static init");  }  } |

# Instance and Static Variables

|  |
| --- |
| **Instance variables:** Instance variables are declared in a class, but outside a method, constructor or any block.When a space is allocated for an object in the heap, a slot for each instance variable value is created. Instance variables are created when an object is created with the use of the keyword ‘new’ and destroyed when the object is destroyed. They are property of an object so they can be accessed using object only.  **Static variables:** Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block. There would only be one copy of each class variable per class, regardless of how many objects are created from it. They are property of a class not of an object so they can be used directly using class name as well as using object.  **public** **class** Instance\_And\_Static\_Variables {  String name;  **int** salary;  **static** String *company*;  **public** **static** **void** main(String[] args) {  Instance\_And\_Static\_Variables.*company* = "GeeksForGeeks";  Instance\_And\_Static\_Variables g = **new** Instance\_And\_Static\_Variables();  g.name = "Shubham";  g.salary = 100000;  Instance\_And\_Static\_Variables sp = **new** Instance\_And\_Static\_Variables();  sp.name = "Chirag";  sp.salary = 200000;  g.printDetails();  sp.printDetails();  g.*company* = "Google";  g.salary = 200000;  System.***out***.println("\nAfter change\n");  g.printDetails();  sp.printDetails();  }  **public** **void** printDetails() {  System.***out***.println("Name: " + name);  System.***out***.println("Company: " + *company*);  System.***out***.println("Salary: " + salary);  }  } |

# Intersaction Using Hashing

|  |
| --- |
| **public** **class** IntersactionUsingHashing {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 7, 1, 5, 2, 3, 6 };  **int** arr2[] = { 3, 8, 6, 20, 7 };  System.***out***.println("Intersection of two arrays is : ");  *printIntersection*(arr1, arr2);  }  **static** **void** printIntersection(**int** arr1[], **int** arr2[]) {  HashSet<Integer> hs = **new** HashSet<>();  HashSet<Integer> hs1 = **new** HashSet<>();  **for** (**int** i = 0; i < arr1.length; i++)  hs.add(arr1[i]);  **for** (**int** i = 0; i < arr2.length; i++)  **if** (hs.contains(arr2[i]))  System.***out***.print(arr2[i] + " ");  }  } |

# Intersection of two sorted arrays

|  |
| --- |
| **public** **class** Intersectionoftwosortedarrays {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 1, 2, 4, 5, 6 };  **int** arr2[] = { 2, 3, 5, 7 };  **int** m = arr1.length;  **int** n = arr2.length;  *printIntersection*(arr1, arr2, m, n);  }  **static** **void** printIntersection(**int** arr1[], **int** arr2[], **int** m, **int** n) {  **int** i = 0, j = 0;  **while** (i < m && j < n) {  **if** (arr1[i] < arr2[j])  i++;  **else** **if** (arr2[j] < arr1[i])  j++;  **else** {  System.***out***.print(arr2[j++] + " ");  i++;  }  }  }  } |

# Intersection of two unsorted arrays using sorting and searching

|  |
| --- |
| **public** **class** Intersectionoftwounsortedarrays {  **public** **static** **void** main(String[] args) {  Intersectionoftwounsortedarrays u\_i = **new** Intersectionoftwounsortedarrays();  **int** arr1[] = { 7, 1, 5, 2, 3, 6 };  **int** arr2[] = { 3, 8, 6, 20, 7 };  **int** m = arr1.length;  **int** n = arr2.length;  System.***out***.println("Intersection of two arrays is ");  u\_i.printIntersection(arr1, arr2, m, n);  }  **void** printIntersection(**int** arr1[], **int** arr2[], **int** m, **int** n) {  **if** (m > n) {  **int** tempp[] = arr1;  arr1 = arr2;  arr2 = tempp;  **int** temp = m;  m = n;  n = temp;  }  Arrays.*sort*(arr1);  **for** (**int** i = 0; i < n; i++) {  **if** (binarySearch(arr1, 0, m - 1, arr2[i]) != -1)  System.***out***.print(arr2[i] + " ");  }  }  **int** binarySearch(**int** arr[], **int** l, **int** r, **int** x) {  **if** (r >= l) {  **int** mid = l + (r - l) / 2;  **if** (arr[mid] == x)  **return** mid;  **if** (arr[mid] > x)  **return** binarySearch(arr, l, mid - 1, x);  **return** binarySearch(arr, mid + 1, r, x);  }  **return** -1;  }  } |

# Iterating Over ArrayLists Using Foreach

|  |
| --- |
| **public** **class** Iterating\_Over\_ArrayLists\_Using\_Foreach {  **public** **static** **void** main(String[] args) {  List<Integer> numbers = Arrays.*asList*(1, 2, 3, 4, 5, 6, 7, 8);  **for** (Integer i : numbers)  System.***out***.print(i + " ");  }  } |

# Iterating Over ArrayLists Using Iterator

|  |
| --- |
| **public** **class** Iterating\_Over\_ArrayLists\_Using\_Iterator {  **public** **static** **void** main(String[] args) {  List<Integer> numbers = Arrays.*asList*(1, 2, 3, 4, 5, 6, 7, 8);  Iterator it = numbers.iterator();  **while** (it.hasNext())  System.***out***.print(it.next() + " ");  }  } |

# Iterating Over ArrayLists Using Loop

|  |
| --- |
| **public** **class** Iterating\_Over\_ArrayLists\_Using\_Loop {  **public** **static** **void** main(String[] args) {  List<Integer> numbers = Arrays.*asList*(1, 2, 3, 4, 5, 6, 7, 8);  **for** (**int** i = 0; i < numbers.size(); i++)  System.***out***.print(numbers.get(i) + " ");  }  } |

# Iterative Reverse Array

|  |
| --- |
| **public** **class** IterativeReverseArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = {1, 2, 3, 4, 5, 6};  *printArray*(arr, 6);  *rvereseArray*(arr, 0, 5);  System.***out***.println("Reversed array is ");  *printArray*(arr, 6);  }  **static** **void** rvereseArray(**int** arr[], **int** start, **int** end){  **int** temp;  **if** (start >= end)  **return**;  temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  *rvereseArray*(arr, start+1, end-1);  }  **static** **void** printArray(**int** arr[], **int** size){  **int** i;  **for** (i=0; i < size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Kth Largest Element Sorting

|  |
| --- |
| Time complexity: O(nlogn)  **public** **class** KthLargestElement\_Sorting {  **public** **static** **void** main(String[] args) {  Integer arr[] = **new** Integer[] { 1, 23, 12, 9, 30, 2, 50 };  **int** k = 3;  *kLargest*(arr, k);  }  **public** **static** **void** kLargest(Integer[] arr, **int** k) {  Arrays.*sort*(arr, Collections.*reverseOrder*());  **for** (**int** i = 0; i < k; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Kth Smallest OR Largest Element Quick

|  |
| --- |
| time complexity is O(n2)  **public** **class** KthSmallestORLargestElement\_Quick {  **public** **static** **void** main(String[] args) {  Integer arr[] = **new** Integer[] { 12, 3, 5, 7, 4, 19, 26 };  **int** k = 3;  System.***out***.print("K'th smallest element is " + *kthSmallest*(arr, 0, arr.length - 1, k));  }  **public** **static** **int** kthSmallest(Integer[] arr, **int** l, **int** r, **int** k) {  **if** (k > 0 && k <= r - l + 1) {  **int** pos = *partition*(arr, l, r);  **if** (pos - l == k - 1)  **return** arr[pos];  **if** (pos - l > k - 1)  **return** *kthSmallest*(arr, l, pos - 1, k);  **return** *kthSmallest*(arr, pos + 1, r, k - pos + l - 1);  }  **return** Integer.***MAX\_VALUE***;  }  **public** **static** **int** partition(Integer[] arr, **int** l, **int** r) {  **int** x = arr[r], i = l;  **for** (**int** j = l; j <= r - 1; j++) {  **if** (arr[j] <= x) {  *swap*(arr[i], arr[j]);  i++;  }  }  *swap*(arr[i], arr[r]);  **return** i;  }  **public** **static** **void** swap(**int** a, **int** b) {  **int** temp = a;  a = b;  b = temp;  }  } |

# Kth Smallest OR Largest Element QuickSelect

|  |
| --- |
| time complexity is O(n2)  **public** **class** KthSmallestORLargestElement\_QuickSelect {  **public** **static** **void** main(String[] args) {  KthSmallestORLargestElement\_QuickSelect ob = **new** KthSmallestORLargestElement\_QuickSelect();  **int** arr[] = { 12, 3, 5, 7, 4, 19, 26 };  **int** n = arr.length, k = 3;  System.***out***.println("K'th smallest element is " + ob.kthSmallest(arr, 0, n - 1, k));  }  **int** kthSmallest(**int** arr[], **int** l, **int** r, **int** k) {  **if** (k > 0 && k <= r - l + 1) {  **int** pos = randomPartition(arr, l, r);  **if** (pos - l == k - 1)  **return** arr[pos];  **if** (pos - l > k - 1)  **return** kthSmallest(arr, l, pos - 1, k);  **return** kthSmallest(arr, pos + 1, r, k - pos + l - 1);  }  **return** Integer.***MAX\_VALUE***;  }  **int** randomPartition(**int** arr[], **int** l, **int** r) {  **int** n = r - l + 1;  **int** pivot = (**int**) (Math.*random*()) % n;  swap(arr, l + pivot, r);  **return** partition(arr, l, r);  }  **int** partition(**int** arr[], **int** l, **int** r) {  **int** x = arr[r], i = l;  **for** (**int** j = l; j <= r - 1; j++) {  **if** (arr[j] <= x) {  swap(arr, i, j);  i++;  }  }  swap(arr, i, r);  **return** i;  }  **void** swap(**int** arr[], **int** i, **int** j) {  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  } |

# K’th Smallest/Largest Element in Unsorted Array:- given an array and a number K where K is smaller than size of array, we need to find the Kth smallest element in the given array.it is given that all array elements are distinct.

|  |
| --- |
| Input: arr[] = {7, 10, 4, 3, 20, 15} k = 3 Output: 7  Input: arr[] = {7, 10, 4, 3, 20, 15} k = 4 Output: 10  Time Complexity is O (nLogn).  **public** **class** KthSmallestORLargestElement\_Simple {  **public** **static** **void** main(String[] args) {  Integer arr[] = **new** Integer[] { 12, 3, 5, 7, 19 };  **int** k = 2;  System.***out***.print("K'th smallest element is " + *kthSmallest*(arr, k));  }  **public** **static** **int** kthSmallest(Integer[] arr, **int** k) {  Arrays.*sort*(arr);  **return** arr[k - 1];  }  } |

# Largest Rectangular Area in a Histogram: - consider following histrogram with 7 bars of heights {6, 2, 5, 4, 5, 1, and 6}. The largest possible rectangle possible is 12.

|  |
| --- |
| Time complexity is O(n).  **public** **class** LargestRectangularArea {  **public** **static** **void** main(String[] args) {  **int** hist[] = { 6, 2, 5, 4, 5, 1, 6 };  System.***out***.println("Maximum area is " + *getMaxArea*(hist, hist.length));  }  **static** **int** getMaxArea(**int** hist[], **int** n) {  Stack<Integer> s = **new** Stack<>();  **int** max\_area = 0;  **int** tp;  **int** area\_with\_top;  **int** i = 0;  **while** (i < n) {  **if** (s.empty() || hist[s.peek()] <= hist[i])  s.push(i++);  **else** {  tp = s.peek();  s.pop();  area\_with\_top = hist[tp] \* (s.empty() ? i : i - s.peek() - 1);  **if** (max\_area < area\_with\_top)  max\_area = area\_with\_top;  }  }  **while** (s.empty() == **false**) {  tp = s.peek();  s.pop();  area\_with\_top = hist[tp] \* (s.empty() ? i : i - s.peek() - 1);  **if** (max\_area < area\_with\_top)  max\_area = area\_with\_top;  }  **return** max\_area;  }  } |

# Largest Number with one swap

|  |
| --- |
| **public** **class** Largest\_number\_with\_one\_swap {  **static** String swap(String str, **int** i, **int** j) {  **char** ch[] = str.toCharArray();  **char** temp = ch[i];  ch[i] = ch[j];  ch[j] = temp;  String c = String.*valueOf*(ch);  **return** c;  }  **static** **int** largestNum(**int** num) {  String num\_in\_str = "" + num;  String temp = num\_in\_str;  **for** (**int** i = 0; i < num\_in\_str.length(); i++) {  **for** (**int** j = i + 1; j < num\_in\_str.length(); j++) {  num\_in\_str = *swap*(num\_in\_str, i, j);  **if** (temp.compareTo(num\_in\_str) < 0)  temp = num\_in\_str;  num\_in\_str = *swap*(num\_in\_str, i, j);  }  }  **return** Integer.*parseInt*(temp);  }  **public** **static** **void** main(String[] s) {  **int** num = 423;  System.***out***.println(*largestNum*(num));  num = 2736;  System.***out***.println(*largestNum*(num));  num = 4596;  System.***out***.println(*largestNum*(num));  }  } |

# Largest subarray with equal number of 0s and 1s

|  |
| --- |
| Input: arr[] = {1, 0, 1, 1, 1, 0, 0} Output: 1 to 6 (Starting and Ending indexes of output subarray)  Input: arr[] = {1, 1, 1, 1} Output: No such subarray  Input: arr[] = {0, 0, 1, 1, 0} Output: 0 to 3 Or 1 to 4  Time Complexity: O(n^2) Auxiliary Space: O(1)  **public** **class** LargestSubarraySimple {  **public** **static** **void** main(String[] args) {  LargestSubarraySimple sub;  sub = **new** LargestSubarraySimple();  **int** arr[] = { 1, 0, 0, 1, 0, 1, 1 };  **int** size = arr.length;  sub.findSubArray(arr, size);  }  **int** findSubArray(**int** arr[], **int** n) {  **int** sum = 0;  **int** maxsize = -1, startindex = 0;  **int** endindex = 0;  **for** (**int** i = 0; i < n - 1; i++) {  sum = (arr[i] == 0) ? -1 : 1;  **for** (**int** j = i + 1; j < n; j++) {  **if** (arr[j] == 0)  sum += -1;  **else**  sum += 1;  **if** (sum == 0 && maxsize < j - i + 1) {  maxsize = j - i + 1;  startindex = i;  }  }  }  endindex = startindex + maxsize - 1;  **if** (maxsize == -1)  System.***out***.println("No such subarray");  **else**  System.***out***.println(startindex + " to " + endindex);  **return** maxsize;  }  } |

# Largest Subarray Tricky

|  |
| --- |
| Time Complexity: O(n) Auxiliary Space: O(n)  **public** **class** LargestSubarrayTricky {  **public** **static** **void** main(String[] args) {  LargestSubarrayTricky sub = **new** LargestSubarrayTricky();  **int** arr[] = { 1, 0, 0, 1, 0, 1, 1 };  **int** n = arr.length;  sub.maxLen(arr, n);  }  **int** maxLen(**int** arr[], **int** n) {  HashMap<Integer, Integer> hM = **new** HashMap<Integer, Integer>();  **int** sum = 0;  **int** max\_len = 0;  **int** ending\_index = -1;  **int** start\_index = 0;  **for** (**int** i = 0; i < n; i++) {  arr[i] = (arr[i] == 0) ? -1 : 1;  }  **for** (**int** i = 0; i < n; i++) {  sum += arr[i];  **if** (sum == 0) {  max\_len = i + 1;  ending\_index = i;  }  **if** (hM.containsKey(sum)) {  **if** (max\_len < i - hM.get(sum + n)) {  max\_len = i - hM.get(sum + n);  ending\_index = i;  }  } **else**  hM.put(sum + n, i);  }  **for** (**int** i = 0; i < n; i++) {  arr[i] = (arr[i] == -1) ? 0 : 1;  }  **int** end = ending\_index - max\_len + 1;  System.***out***.println(end + " to " + ending\_index);  **return** max\_len;  }  } |

# Find last k digits in product of an array numbers

|  |
| --- |
| Input : a[] = {22, 31, 44, 27, 37, 43} Output : 56  Input : a[] = {24, 7, 144, 77, 29, 19} Output : 84  **public** **class** LastkDigitsInProduct {  **public** **static** **void** main(String[] args) {  **int** a[] = { 22, 31, 44, 27, 37, 43 };  **int** k = 2;  **int** n = a.length;  System.***out***.println(*lastKDigits*(a, n, k));  }  **static** **int** lastKDigits(**int** a[], **int** n, **int** k) {  **int** num = (**int**) (Math.*pow*(10, k));  **int** mul = a[0] % num;  **for** (**int** i = 1; i < n; i++) {  a[i] = a[i] % num;  mul = (a[i] \* mul) % num;  }  **return** mul;  }  } |

# Length of the largest subarray with contiguous elements

|  |
| --- |
| Time complexity is O(n2)  Input: arr[] = {10, 12, 11}; Output: Length of the longest contiguous subarray is 3  Input: arr[] = {10, 12, 12, 10, 10, 11, 10}; Output: Length of the longest contiguous subarray is 2  **public** **class** LengthOfLargestSubarray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 12, 12, 10, 10, 11, 10 };  System.***out***.println("Length of the longest contiguous subarray is " + *findLength*(arr));  }  **static** **int** findLength(**int** arr[]) {  **int** n = arr.length;  **int** max\_len = 1;  **for** (**int** i = 0; i < n - 1; i++) {  HashSet<Integer> set = **new** HashSet<>();  set.add(arr[i]);  **int** mn = arr[i], mx = arr[i];  **for** (**int** j = i + 1; j < n; j++) {  **if** (set.contains(arr[j]))  **break**;  set.add(arr[j]);  mn = Math.*min*(mn, arr[j]);  mx = Math.*max*(mx, arr[j]);  **if** (mx - mn == j - i)  max\_len = Math.*max*(max\_len, mx - mn + 1);  }  }  **return** max\_len;  }  } |

# Object level lock

|  |
| --- |
| **Synchronization:** Synchronization is a modifier which is used for method and block only. With the help of synchronized modifier we can restrict a shared resource to be accessed only by one thread. When two or more threads need access to shared resources, there is some loss of data i.e. data inconsistency. The process by which we can achieve data consistency between multiple threads it is called Synchronization.  **Why do you need Synchronization?** Let us assume if you have two threads that are reading and writing to the same ‘resource’. Suppose there is a variable named as geek, and you want that at one time only one thread should access the variable (atomic way). But Without the synchronized keyword, your thread 1 may not see the changes thread 2 made to geek, or worse, it may only be half changed that cause the data inconsistency problem. This would not be what you logically expect. The tool needed to prevent these errors is synchronization.  **In synchronization, there are two types of locks on threads:**  **1. Object level lock: -** Every object in java has a unique lock. Whenever we are using synchronized keyword, then only lock concept will come in the picture. If a thread wants to execute synchronized method on the given object. First, it has to get lock of that object.  Once thread got the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock. Acquiring and release lock internally is taken care by JVM and programmer is not responsible for these activities.  public class Lock\_Object\_Level implements Runnable {  public static void main(String[] args) {  Lock\_Object\_Level g = new Lock\_Object\_Level();  Thread t1 = new Thread(g);  Thread t2 = new Thread(g);  Lock\_Object\_Level g1 = new Lock\_Object\_Level();  Thread t3 = new Thread(g1);  t1.setName("t1");  t2.setName("t2");  t3.setName("t3");  t1.start();  t2.start();  t3.start();  }  public void run() {  Lock();  }  public void Lock() {  System.out.println(Thread.currentThread().getName());  synchronized (this) {  System.out.println("in block " + Thread.currentThread().getName());  System.out.println("in block " + Thread.currentThread().getName() + " end");  }  }  } |

# Class level lock

|  |
| --- |
| **2. Class level lock: -** Every lock in java has a unique lock which is nothing but class level lock. If a thread wants to execute a static synchronized method, then thread requires class level lock. Once a thread got the class level lock, then it is allowed to execute any static synchronized method of that class. Once method execution completes automatically thread releases the lock.  **public** **class** Lock\_Class\_Level **implements** Runnable {  **public** **static** **void** main(String[] args) {  Lock\_Class\_Level g1 = **new** Lock\_Class\_Level();  Thread t1 = **new** Thread(g1);  Thread t2 = **new** Thread(g1);  Lock\_Class\_Level g2 = **new** Lock\_Class\_Level();  Thread t3 = **new** Thread(g2);  t1.setName("t1");  t2.setName("t2");  t3.setName("t3");  t1.start();  t2.start();  t3.start();  }  **public** **void** run() {  Lock();  }  **public** **void** Lock() {  System.***out***.println(Thread.*currentThread*().getName());  **synchronized** (Lock\_Class\_Level.**class**) {  System.***out***.println("in block " + Thread.*currentThread*().getName());  System.***out***.println("in block " + Thread.*currentThread*().getName() + " end");  }  }  } |

# Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

|  |
| --- |
| Input: arr[] = {1, 9, 3, 10, 4, 20, 2}; Output: 4 The subsequence 1, 3, 4, 2 is the longest subsequence of consecutive elements  Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42} Output: 5 The subsequence 36, 35, 33, 34, 32 is the longest subsequence of consecutive elements.  **public** **class** Longest\_Consecutive\_Subsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 9, 3, 10, 4, 20, 2 };  **int** n = arr.length;  System.***out***.println("Length of the Longest consecutive subsequence is " + *findLongestConseqSubseq*(arr, n));  }  **static** **int** findLongestConseqSubseq(**int** arr[], **int** n) {  HashSet<Integer> S = **new** HashSet<Integer>();  **int** ans = 0;  **for** (**int** i = 0; i < n; ++i)  S.add(arr[i]);  **for** (**int** i = 0; i < n; ++i) {  **if** (!S.contains(arr[i] - 1)) {  **int** j = arr[i];  **while** (S.contains(j))  j++;  **if** (ans < j - arr[i])  ans = j - arr[i];  }  }  **return** ans;  }  } |

# Longest increasing subarray

|  |
| --- |
| Input : arr[] = {5, 6, 3, 5, 7, 8, 9, 1, 2} Output : 5 The subarray is {3, 5, 7, 8, 9}  Input : arr[] = {12, 13, 1, 5, 4, 7, 8, 10, 10, 11} Output : 4 The subarray is {4, 7, 8, 10}  **public** **class** Longestincreasingsubarray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 6, 3, 5, 7, 8, 9, 1, 2 };  **int** n = arr.length;  System.***out***.println("Length of longest increasing subarray = " + *lenOfLongIncSubArr*(arr, n));  System.***out***.print("print Longest increasing subarray = ");  *printLogestIncSubArr*(arr, n);  }  **public** **static** **int** lenOfLongIncSubArr(**int** arr[], **int** n) {  **int** max = 1, len = 1;  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] > arr[i - 1])  len++;  **else** {  **if** (max < len)  max = len;  len = 1;  }  }  **if** (max < len)  max = len;  **return** max;  }  **public** **static** **void** printLogestIncSubArr(**int** arr[], **int** n) {  **int** max = 1, len = 1, maxIndex = 0;  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] > arr[i - 1])  len++;  **else** {  **if** (max < len) {  max = len;  maxIndex = i - max;  }  len = 1;  }  }  **if** (max < len) {  max = len;  maxIndex = n - max;  }  **for** (**int** i = maxIndex; i < max + maxIndex; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Longest Increasing Subsequence: - the longest increasing subsequence problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. for e.g. the length of LIS for {10,22,9,33,21,50,41,60,80} is 6 and LIS is {10,22,33,50,60,80}

|  |
| --- |
| Input : arr[] = {3, 10, 2, 1, 20} Output : Length of LIS = 3 The longest increasing subsequence is 3, 10, 20  public class LongestIncreasingSubsequence {  public static void main(String[] args) {  int A[] = { 2, 5, 3, 7, 11, 8, 10, 13, 6 };  int n = A.length;  System.out.println("Length of Longest Increasing Subsequence is " + LongestIncreasingSubsequenceLength(A, n));  }  static int LongestIncreasingSubsequenceLength(int A[], int size) {  int[] tailTable = new int[size];  int len;  tailTable[0] = A[0];  len = 1;  for (int i = 1; i < size; i++) {  if (A[i] < tailTable[0])  tailTable[0] = A[i];  else if (A[i] > tailTable[len - 1])  tailTable[len++] = A[i];  else  tailTable[CeilIndex(tailTable, -1, len - 1, A[i])] = A[i];  }  return len;  }  static int CeilIndex(int A[], int l, int r, int key) {  while (r - l > 1) {  int m = l + (r - l) / 2;  if (A[m] >= key)  r = m;  else  l = m;  }  return r;  }  } |

# Longest palindrome subsequence with O (n) space

|  |
| --- |
| Input : abbaab Output : 4  Input : geeksforgeeks Output : 5 Time Complexity : O(n\*n) Auxiliary Space : O(n)  **public** **class** LongestPalindromeSubsequence {  **public** **static** **void** main(String[] args) {  String str = "GEEKSFORGEEKS";  System.***out***.println(*lps*(str));  }  **static** **int** lps(String s) {  **int** n = s.length();  **int** a[] = **new** **int**[n];  **for** (**int** i = n - 1; i >= 0; i--) {  **int** back\_up = 0;  **for** (**int** j = i; j < n; j++) {  **if** (j == i)  a[j] = 1;  **else** **if** (s.charAt(i) == s.charAt(j)) {  **int** temp = a[j];  a[j] = back\_up + 2;  back\_up = temp;  }  **else** {  back\_up = a[j];  a[j] = Math.*max*(a[j - 1], a[j]);  }  }  }  **return** a[n - 1];  }  } |

# Longest Span Sum Auxiliary

|  |
| --- |
| **public** **class** LongestSpanSumAuxiliary {  **static** **int** *arr1*[] = **new** **int**[] { 0, 1, 0, 1, 1, 1, 1 };  **static** **int** *arr2*[] = **new** **int**[] { 1, 1, 1, 1, 1, 0, 1 };  **public** **static** **void** main(String[] args) {  System.***out***.print("Length of the longest common span with same sum is ");  System.***out***.println(*longestCommonSum*(*arr1*.length));  }  **static** **int** longestCommonSum(**int** n) {  **int** maxLen = 0;  **int** preSum1 = 0, preSum2 = 0;  **int** diff[] = **new** **int**[2 \* n + 1];  **for** (**int** i = 0; i < diff.length; i++) {  diff[i] = -1;  }  **for** (**int** i = 0; i < n; i++) {  preSum1 += *arr1*[i];  preSum2 += *arr2*[i];  **int** curr\_diff = preSum1 - preSum2;  **int** diffIndex = n + curr\_diff;  **if** (curr\_diff == 0)  maxLen = i + 1;  **else** **if** (diff[diffIndex] == -1)  diff[diffIndex] = i;  **else** {  **int** len = i - diff[diffIndex];  **if** (len > maxLen)  maxLen = len;  }  }  **return** maxLen;  }  } |

# Longest Span with same Sum in two Binary arrays:-given two binary arrays arr1 [] and arr2 [] of same size n. find length of the longest common span(i,j) where j>=I such that arr1[i]+arr1[i+1]+…+arr1[j]= arr2[i]+arr2[i+1]+…+arr2[j]

|  |
| --- |
| Time Complexity: O(n2) Auxiliary Space: O(1)  Input: arr1[] = {0, 1, 0, 0, 0, 0}; arr2[] = {1, 0, 1, 0, 0, 1}; Output: 4  **public** **class** LongestSpanSumSimple {  **static** **int** *arr1*[] = **new** **int**[] { 0, 1, 0, 1, 1, 1, 1 };  **static** **int** *arr2*[] = **new** **int**[] { 1, 1, 1, 1, 1, 0, 1 };  **public** **static** **void** main(String[] args) {  System.***out***.print("Length of the longest common span with same sum is ");  System.***out***.println(*longestCommonSum*(*arr1*.length));  }  **static** **int** longestCommonSum(**int** n) {  **int** maxLen = 0;  **for** (**int** i = 0; i < n; i++) {  **int** sum1 = 0, sum2 = 0;  **for** (**int** j = i; j < n; j++) {  sum1 += *arr1*[j];  sum2 += *arr2*[j];  **if** (sum1 == sum2) {  **int** len = j - i + 1;  **if** (len > maxLen)  maxLen = len;  }  }  }  **return** maxLen;  }  } |

# Longest Uncommon Subsequence BruteForce

|  |
| --- |
| Input: "abcd", "abc" Output: 4 The longest subsequence is 4 because "abcd" is a subsequence of first string, but not a subsequence of second string.  Time complexity: O (2x + 2y), where x and y are the lengths of two strings. Auxiliary Space: O (2x + 2y).  **public** **class** LongestUncommonSubsequence\_BruteForce {  **public** **static** **void** main(String[] args) {  String a = "abcdabcd", b = "abcabc";  System.***out***.println(*findLUSlength*(a, b));  }  **static** **int** findLUSlength(String a, String b) {  HashMap<String, Integer> map = **new** HashMap<String, Integer>();  Vector<String> strArr = **new** Vector<String>();  strArr.add(a);  strArr.add(b);  **for** (String s : strArr) {  **for** (**int** i = 0; i < (1 << s.length()); i++) {  String t = "";  **for** (**int** j = 0; j < s.length(); j++) {  **if** (((i >> j) & 1) != 0)  t += s.charAt(j);  }  **if** (map.containsKey(t))  map.put(t, map.get(t) + 1);  **else**  map.put(t, 1);  }  }  **int** res = 0;  **for** (HashMap.Entry<String, Integer> entry : map.entrySet()) {  **if** (entry.getValue() == 1)  res = Math.*max*(res, entry.getKey().length());  }  **return** res;  }  } |

# Longest Uncommon Subsequence Efficient

|  |
| --- |
| **public** **class** LongestUncommonSubsequence\_Efficient {  **public** **static** **void** main(String[] args) {  String a = "abcdabcd", b = "abcabc";  System.***out***.println(*findLUSlength*(a, b));  }  **static** **int** findLUSlength(String a, String b) {  **if** (a.equals(b) == **true**)  **return** 0;  **return** Math.*max*(a.length(), b.length());  }  } |

# Leaders in an Array Method1

|  |
| --- |
| **public** **class** LeadersInAnArray\_Method1 {  **public** **static** **void** main(String[] args) {  LeadersInAnArray\_Method1 lead = **new** LeadersInAnArray\_Method1();  **int** arr[] = **new** **int**[] { 16, 17, 4, 3, 5, 2 };  **int** n = arr.length;  lead.printLeaders(arr, n);  }  **void** printLeaders(**int** arr[], **int** size) {  **for** (**int** i = 0; i < size; i++) {  **int** j;  **for** (j = i + 1; j < size; j++) {  **if** (arr[i] <= arr[j])  **break**;  }  **if** (j == size) // the loop didn't break  System.***out***.println(arr[i] + " ");  }  }  } |

# Leaders in an Array Method2

|  |
| --- |
| **public** **class** LeadersInAnArray\_Method2 {  **public** **static** **void** main(String[] args) {  LeadersInAnArray\_Method2 lead = **new** LeadersInAnArray\_Method2();  **int** arr[] = **new** **int**[] { 16, 17, 4, 3, 5, 2 };  **int** n = arr.length;  lead.printLeaders(arr, n);  }  **void** printLeaders(**int** arr[], **int** size) {  **int** max\_from\_right = arr[size - 1];  System.***out***.println(max\_from\_right + " ");  **for** (**int** i = size - 2; i >= 0; i--) {  **if** (max\_from\_right < arr[i]) {  max\_from\_right = arr[i];  System.***out***.println(max\_from\_right + " ");  }  }  }  } |

# Live Lock

|  |
| --- |
| **public** **class** Criminal {  **private** **boolean** hostageReleased = **false**;  **public** **void** releaseHostage(Police police) {  **while** (!police.isMoneySent()) {  System.***out***.println("Criminal: waiting police to give ransom");  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException ex) {  ex.printStackTrace();  }  }  System.***out***.println("Criminal: released hostage");  **this**.hostageReleased = **true**;  }  **public** **boolean** isHostageReleased() {  **return** **this**.hostageReleased;  }  }  **public** **class** Police {  **private** **boolean** moneySent = **false**;  **public** **void** giveRansom(Criminal criminal) {  **while** (!criminal.isHostageReleased()) {  System.***out***.println("Police: waiting criminal to release hostage");  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException ex) {  ex.printStackTrace();  }  }  System.***out***.println("Police: sent money");  **this**.moneySent = **true**;  }  **public** **boolean** isMoneySent() {  **return** **this**.moneySent;  }  }  **public** **class** HostageRescueLiveLock {  **static** **final** Police ***p*** = **new** Police();  **static** **final** Criminal ***c*** = **new** Criminal();  **public** **static** **void** main(String[] args) {  Thread t1 = **new** Thread(**new** Runnable() {  **public** **void** run() {  ***p***.giveRansom(***criminal***);  }  });  t1.start();  Thread t2 = **new** Thread(**new** Runnable() {  **public** **void** run() {  ***c***.releaseHostage(***police***);  }  });  t2.start();  }  } |

# Longest Palindrome String

|  |
| --- |
| **public** **class** StringPalindrome\_Longest\_Method {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.print("Enter a string: ");  String str = r.next();  String out = longestPalindrom(str);  System.***out***.println(out);  }  **public** **static** String longestPalindrom(String str) {  **if**(str == **null**)  **return** **null**;  String longest = str.substring(0,1);  **for**(**int** i =0 ; i<str.length()-1; i++){  // odd case like 121  String palindrome = *intermediatePalindrome*(str,i,i);  **if**(palindrome.length() > longest.length()){  longest = palindrome;  }  // even case like 1221  palindrome = *intermediatePalindrome*(str,i,i+1);  **if**(palindrome.length() > longest.length()){  longest = palindrome;  }  }  **return** longest;  }  **public** **static** String intermediatePalindrome(String str , **int** left , **int** right){  **if**(left > right)  **return** **null**;  **while**(left >= 0 && right <str.length() && str.charAt(left) == str.charAt(right)){  left--;  right++;  }  **return** str.substring(left+1, right);  }  } |

# Making Elements Distinct1

|  |
| --- |
| Input : arr[] = { 2, 2, 3, 5, 6 } ; Output : 20 Explanation : We make the array as {2, 3, 4, 5, 6}. Sum becomes 2 + 3 + 4 + 5 + 6 = 20  Time Complexity : O(n^2)  **public** **class** MakingElementsDistinct1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 2, 3, 5, 6 };  **int** n = arr.length;  System.***out***.println(*minSum*(arr, n));  }  **static** **int** minSum(**int** arr[], **int** n) {  **int** sum = arr[0];  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] == arr[i - 1]) {  **int** j = i;  **while** (j < n && arr[j] <= arr[j - 1]) {  arr[j] = arr[j] + 1;  j++;  }  }  sum = sum + arr[i];  }  **return** sum;  }  } |

# Making Elements Distinct2

|  |
| --- |
| Time Complexity : O(n)  **public** **class** MakingElementsDistinct2 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 2, 3, 5, 6 };  **int** n = arr.length;  System.***out***.println(*minSum*(arr, n));  }  **static** **int** minSum(**int** arr[], **int** n) {  **int** sum = arr[0], prev = arr[0];  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] <= prev) {  prev = prev + 1;  sum = sum + prev;  }  **else** {  sum = sum + arr[i];  prev = arr[i];  }  }  **return** sum;  }  } |

# Maximize sum of consecutive differences in a circular array

|  |
| --- |
| Input : arr[] = { 4, 2, 1, 8 } Output : 18 Rearrange given array as : { 1, 8, 2, 4 } Sum of difference between consecutive element = |1 - 8| + |8 - 2| + |2 - 4| + |4 - 1| = 7 + 6 + 2 + 3 = 18.  Time Complexity: O (nlogn). Auxiliary Space : O(1)  **public** **class** MaximizeSumOfConsecutiveDifferences {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 4, 2, 1, 8 };  **int** n = arr.length;  System.***out***.println(*maxSum*(arr, n));  }  **static** **int** maxSum(**int** arr[], **int** n) {  **int** sum = 0;  Arrays.*sort*(arr);  **for** (**int** i = 0; i < n / 2; i++) {  sum -= (2 \* arr[i]);  sum += (2 \* arr[n - i - 1]);  }  **return** sum;  }  } |

# Maximum difference of zeros and ones in binary string:- given a binary string of 0s and 1s.the task is to find the length of substring which is having a maximum difference of number of 0s and number of 1s(no of 0s- no of 1s) In case of all print -1.

|  |
| --- |
| Input: S = "11000010001" Output: 6 From index 2 to index 9, there are 7 0s and 1 1s, so number of 0s – number of 1s is 6.  Time Complexity : O(n) Space complexity : O(1)  **public** **class** Maximum\_difference\_of\_zeros\_and\_ones {  **public** **static** **void** main(String[] args) {  String str = "11000010001";  **int** n = str.length();  System.***out***.println(*findLength*(str, n));  }  **public** **static** **int** findLength(String str, **int** n) {  **int** current\_sum = 0;  **int** max\_sum = 0;  **for** (**int** i = 0; i < n; i++) {  current\_sum += (str.charAt(i) == '0' ? 1 : -1);  **if** (current\_sum < 0)  current\_sum = 0;  max\_sum = Math.*max*(current\_sum, max\_sum);  }  **return** max\_sum == 0 ? -1 : max\_sum;  }  } |

# Maximum sum such that no two elements are adjacent

|  |
| --- |
| Input : arr[] = {5, 5, 10, 100, 10, 5} Output : 110  Input : arr[] = {1, 2, 3} Output : 4  Input : arr[] = {1, 20, 3} Output : 20  **public** **class** Maximum\_sum\_such\_that\_no\_two\_elements\_are\_adjacent {  **public** **static** **void** main(String[] args) {  Maximum\_sum\_such\_that\_no\_two\_elements\_are\_adjacent sum = **new** Maximum\_sum\_such\_that\_no\_two\_elements\_are\_adjacent();  **int** arr[] = **new** **int**[] { 5, 5, 10, 100, 10, 5 };  System.***out***.println(sum.FindMaxSum(arr, arr.length));  }  **int** FindMaxSum(**int** arr[], **int** n) {  **int** incl = arr[0];  **int** excl = 0;  **int** excl\_new;  **int** i;  **for** (i = 1; i < n; i++) {  excl\_new = (incl > excl) ? incl : excl;  incl = excl + arr[i];  excl = excl\_new;  }  **return** ((incl > excl) ? incl : excl);  }  } |

# Maximum Difference Efficient

|  |
| --- |
| Time Complexity: O(n) Auxiliary Space: O(1)  **public** **class** MaximumDifferenceEfficient {  **public** **static** **void** main(String[] args) {  MaximumDifferenceEfficient maxdif = **new** MaximumDifferenceEfficient();  **int** arr[] = { 1, 2, 90, 10, 110 };  **int** size = arr.length;  System.***out***.println("MaximumDifference is " + maxdif.maxDiff(arr, size));  }  **int** maxDiff(**int** arr[], **int** arr\_size) {  **int** max\_diff = arr[1] - arr[0];  **int** min\_element = arr[0];  **int** i;  **for** (i = 1; i < arr\_size; i++) {  **if** (arr[i] - min\_element > max\_diff)  max\_diff = arr[i] - min\_element;  **if** (arr[i] < min\_element)  min\_element = arr[i];  }  **return** max\_diff;  }  } |

# Maximum difference between two elements such that larger element appears after the smaller number: - given an array arr [] of interger, find out the difference between any two elements such that larger element appears after the smaller number in arr []. For e.g. if arr is {2, 3, 10, 6, 4, 8, 1} then returned value should be 8 (diff between 10 and 2). If array is [7, 9, 5, 6, 3, 2] then returned value should be 2 (diff between 7 and 9).

|  |
| --- |
| Time Complexity: O(n^2) Auxiliary Space: O(1)  **public** **class** MaximumDifferenceSimple {  **public** **static** **void** main(String[] args) {  MaximumDifferenceSimple maxdif = **new** MaximumDifferenceSimple();  **int** arr[] = { 1, 2, 90, 10, 110 };  System.***out***.println("Maximum differnce is " + maxdif.maxDiff(arr, 5));  }  **int** maxDiff(**int** arr[], **int** arr\_size) {  **int** max\_diff = arr[1] - arr[0];  **int** i, j;  **for** (i = 0; i < arr\_size; i++) {  **for** (j = i + 1; j < arr\_size; j++) {  **if** (arr[j] - arr[i] > max\_diff)  max\_diff = arr[j] - arr[i];  }  }  **return** max\_diff;  }  } |

# Maximum Difference Tricky

|  |
| --- |
| Time Complexity: O(n) Auxiliary Space: O(n)  **public** **class** MaximumDifferenceTricky {  **public** **static** **void** main(String[] args) {  MaximumDifferenceTricky mxdif = **new** MaximumDifferenceTricky();  **int** arr[] = { 80, 2, 6, 3, 100 };  **int** size = arr.length;  System.***out***.println(mxdif.maxDiff(arr, size));  }  **int** maxDiff(**int** arr[], **int** n) {  **int** diff[] = **new** **int**[n - 1];  **for** (**int** i = 0; i < n - 1; i++)  diff[i] = arr[i + 1] - arr[i];  **int** max\_diff = diff[0];  **for** (**int** i = 1; i < n - 1; i++) {  **if** (diff[i - 1] > 0)  diff[i] += diff[i - 1];  **if** (max\_diff < diff[i])  max\_diff = diff[i];  }  **return** max\_diff;  }  } |

# Maximum Elements of Array

|  |
| --- |
| Input: arr [] = {10 50 30 60 15} k = 2 Output : 50 60 The top 2 elements are printed as per their appearance in original array.  Input : arr[] = {50 8 45 12 25 40 84} k = 3 Output : 50 45 84  Time Complexity: O (n Log n) for sorting. Auxiliary Space : O(n)  **public** **class** MaximumElementsOfArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 50, 8, 45, 12, 25, 40, 84 };  **int** n = arr.length;  **int** k = 3;  *printMax*(arr, k, n);  }  **public** **static** **void** printMax(**int** arr[], **int** k, **int** n) {  Integer[] brr = **new** Integer[n];  **for** (**int** i = 0; i < n; i++)  brr[i] = arr[i];  Arrays.*sort*(brr, Collections.*reverseOrder*());  **for** (**int** i = 0; i < n; ++i)  **if** (Arrays.*binarySearch*(brr, arr[i], Collections.*reverseOrder*()) >= 0  && Arrays.*binarySearch*(brr, arr[i], Collections.*reverseOrder*()) < k)  System.***out***.print(arr[i] + " ");  }  } |

# Maximum Index Difference Efficient

|  |
| --- |
| Time Complexity: O(n) Auxiliary Space: O(n)  **public** **class** MaximumIndexDifferenceEfficient {  **public** **static** **void** main(String[] args) {  MaximumIndexDifferenceEfficient max = **new** MaximumIndexDifferenceEfficient();  **int** arr[] = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };  **int** n = arr.length;  **int** maxDiff = max.maxIndexDiff(arr, n);  System.***out***.println(maxDiff);  }  **int** maxIndexDiff(**int** arr[], **int** n) {  **int** maxDiff;  **int** i, j;  **int** RMax[] = **new** **int**[n];  **int** LMin[] = **new** **int**[n];  LMin[0] = arr[0];  **for** (i = 1; i < n; ++i)  LMin[i] = min(arr[i], LMin[i - 1]);  RMax[n - 1] = arr[n - 1];  **for** (j = n - 2; j >= 0; --j)  RMax[j] = max(arr[j], RMax[j + 1]);  i = 0;  j = 0;  maxDiff = -1;  **while** (j < n && i < n) {  **if** (LMin[i] < RMax[j]) {  maxDiff = max(maxDiff, j - i);  j = j + 1;  } **else**  i = i + 1;  }  **return** maxDiff;  }  **int** max(**int** x, **int** y) {  **return** x > y ? x : y;  }  **int** min(**int** x, **int** y) {  **return** x < y ? x : y;  }  } |

# Given an array arr[], find the maximum j – i such that arr[j] > arr[i]

|  |
| --- |
| Input: {34, 8, 10, 3, 2, 80, 30, 33, 1} Output: 6 (j = 7, i = 1)  Input: {6, 5, 4, 3, 2, 1} Output: -1 Time Complexity: O(n^2)  **public** **class** MaximumIndexDifferenceSimple {  **public** **static** **void** main(String[] args) {  MaximumIndexDifferenceSimple max = **new** MaximumIndexDifferenceSimple();  **int** arr[] = { 9, 2, 3, 4, 5, 6, 7, 8, 18, 0 };  **int** n = arr.length;  **int** maxDiff = max.maxIndexDiff(arr, n);  System.***out***.println(maxDiff);  }  **int** maxIndexDiff(**int** arr[], **int** n) {  **int** maxDiff = -1;  **int** i, j;  **for** (i = 0; i < n; ++i) {  **for** (j = n - 1; j > i; --j) {  **if** (arr[j] > arr[i] && maxDiff < (j - i))  maxDiff = j - i;  }  }  **return** maxDiff;  }  } |

# Maximum Length Bitonic Subarray | Set 1 (O(n) tine and O(n) space)

|  |
| --- |
| 1) A[] = {12, 4, 78, 90, 45, 23}, the maximum length bitonic subarray is {4, 78, 90, 45, 23} which is of length 5.  2) A[] = {20, 4, 1, 2, 3, 4, 2, 10}, the maximum length bitonic subarray is {1, 2, 3, 4, 2} which is of length 5.  3) A[] = {10}, the single element is bitnoic, so output is 1.  4) A[] = {10, 20, 30, 40}, the complete array itself is bitonic, so output is 4.  5) A[] = {40, 30, 20, 10}, the complete array itself is bitonic, so output is 4.  **public** **class** MaximumLengthBitonicSubarray\_Set1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 4, 78, 90, 45, 23 };  **int** n = arr.length;  System.***out***.println("Length of max length Bitnoic Subarray is " + *bitonic*(arr, n));  }  **static** **int** bitonic(**int** arr[], **int** n) {  **int**[] inc = **new** **int**[n];  **int**[] dec = **new** **int**[n];  **int** max;  inc[0] = 1;  dec[n - 1] = 1;  **for** (**int** i = 1; i < n; i++)  inc[i] = (arr[i] >= arr[i - 1]) ? inc[i - 1] + 1 : 1;  **for** (**int** i = n - 2; i >= 0; i--)  dec[i] = (arr[i] >= arr[i + 1]) ? dec[i + 1] + 1 : 1;  max = inc[0] + dec[0] - 1;  **for** (**int** i = 1; i < n; i++)  **if** (inc[i] + dec[i] - 1 > max)  max = inc[i] + dec[i] - 1;  **return** max;  }  } |

# Maximum Length Bitonic Subarray | Set 2 (O(n) time and O(1) Space)

|  |
| --- |
| **public** **class** MaximumLengthBitonicSubarray\_Set2 {  **public** **static** **void** main(String[] args) {  **int** A[] = { 12, 4, 78, 90, 45, 23 };  System.***out***.println("Length of maximal length bitonic " + "subarray is " + *maxLenBitonic*(A, A.length));  }  **static** **int** maxLenBitonic(**int**[] A, **int** n) {  **if** (n == 0) **return** 0;  **int** maxLen = 1;  **int** start = 0;  **int** nextStart = 0;  **int** j = 0;  **while** (j < n - 1) {  **while** (j < n - 1 && A[j] <= A[j + 1])  j++;  **while** (j < n - 1 && A[j] >= A[j + 1]) {  **if** (j < n - 1 && A[j] > A[j + 1])  nextStart = j + 1;  j++;  }  maxLen = Math.*max*(maxLen, j - (start - 1));  start = nextStart;  }  **return** maxLen;  }  } |

# Maximum Occurring Character

|  |
| --- |
| Time Complexity: O(n) Space Complexity: O(1) — Because we are using fixed space (Hash array) irrespective of input string size.  **public** **class** MaximumOccurringCharacter {  **static** **final** **int** ***ASCII\_SIZE*** = 256;  **public** **static** **void** main(String[] args) {  String str = "sample string";  System.***out***.println("Max occurring character is " + *getMaxOccuringChar*(str));  }  **static** **char** getMaxOccuringChar(String str) {  **int** count[] = **new** **int**[***ASCII\_SIZE***];  **int** len = str.length();  **for** (**int** i = 0; i < len; i++)  count[str.charAt(i)]++;  **int** max = -1;  **char** result = ' ';  **for** (**int** i = 0; i < len; i++) {  **if** (max < count[str.charAt(i)]) {  max = count[str.charAt(i)];  result = str.charAt(i);  }  }  **return** result;  }  } |

# Find the maximum repeating number in O(n) time and O(1) extra space

|  |
| --- |
| **public** **class** MaximumRepeatingNumber {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 3, 5, 3, 4, 1, 7 };  **int** n = arr.length;  **int** k = 8;  System.***out***.println("Maximum repeating element is: " + *maxRepeating*(arr, n, k));  }  **static** **int** maxRepeating(**int** arr[], **int** n, **int** k) {  **for** (**int** i = 0; i < n; i++)  arr[(arr[i] % k)] += k;  **int** max = arr[0], result = 0;  **for** (**int** i = 1; i < n; i++) {  **if** (arr[i] > max) {  max = arr[i];  result = i;  }  }  **return** result;  }  } |

# Maximum Sum

|  |
| --- |
| **class** MaximumSum {  **public** **static** **void** main(String[] args) {  MaximumSum sum = **new** MaximumSum();  **int** arr[] = **new** **int**[] { 5, 5, 10, 100, 10, 5 };  System.***out***.println(sum.FindMaxSum(arr, arr.length));  }  **int** FindMaxSum(**int** arr[], **int** n) {  **int** incl = arr[0];  **int** excl = 0;  **int** excl\_new;  **int** i;  **for** (i = 1; i < n; i++) {  excl\_new = (incl > excl)? incl : excl;  incl = excl + arr[i];  excl = excl\_new;  }  **return** ((incl > excl) ? incl : excl);  }  } |

# Given two sorted arrays such the arrays may have some common elements: - find the sum of the maximum sum path to reach from beginning of any array to end of any to the two arrays.

|  |
| --- |
| Input: ar1 [] = {2, 3, 7, 10, 12}, ar2 [] = {1, 5, 7, 8} Output: 35 35 is sum of 1 + 5 + 7 + 10 + 12. We start from first element of arr2 which is 1, then we move to 5, then 7. From 7, we switch to ar1 (7 is common) and traverse 10 and 12.  Input: ar1 [] = {10, 12}, ar2 = {5, 7, 9} Output: 22 22 is sum of 10 and 12. Since there is no common element, we need to take all elements from the array with more sum.  Time complexity is O (m+n).  **public** **class** MaximumSumPath {  **public** **static** **void** main(String[] args) {  MaximumSumPath sumpath = **new** MaximumSumPath();  **int** ar1[] = { 2, 3, 7, 10, 12, 15, 30, 34 };  **int** ar2[] = { 1, 5, 7, 8, 10, 15, 16, 19 };  **int** m = ar1.length;  **int** n = ar2.length;  System.***out***.println("Maximum sum path is :" + sumpath.maxPathSum(ar1, ar2, m, n));  }  **int** maxPathSum(**int** ar1[], **int** ar2[], **int** m, **int** n) {  **int** i = 0, j = 0;  **int** result = 0, sum1 = 0, sum2 = 0;  **while** (i < m && j < n) {  **if** (ar1[i] < ar2[j])  sum1 += ar1[i++];  **else** **if** (ar1[i] > ar2[j])  sum2 += ar2[j++];  **else** {  result += max(sum1, sum2);  sum1 = 0;  sum2 = 0;  **while** (i < m && j < n && ar1[i] == ar2[j]) {  result = result + ar1[i++];  j++;  }  }  }  **while** (i < m)  sum1 += ar1[i++];  **while** (j < n)  sum2 += ar2[j++];  result += max(sum1, sum2);  **return** result;  }  **int** max(**int** x, **int** y) {  **return** (x > y) ? x : y;  }  } |

# Find maximum sum possible equal sum of three stacks: - given three stack of the positive numbers, the task is to find the possible equal maximum sum of the stacks with removal of top elements allowed.stacks are represented as array, and the first index of the array represent the top element of the stack.

|  |
| --- |
| Input : stack1[] = { 3, 10} stack2[] = { 4, 5 } stack3[] = { 2, 1 } Output : 0 Sum can only be equal after removing all elements from all stacks. Time Complexity : O(n1 + n2 + n3)  **public** **class** MaximumSumPossibleEqualSum {  **public** **static** **void** main(String[] args) {  **int** stack1[] = { 3, 2, 1, 1, 1 };  **int** stack2[] = { 4, 3, 2 };  **int** stack3[] = { 1, 1, 4, 1 };  **int** n1 = stack1.length;  **int** n2 = stack2.length;  **int** n3 = stack3.length;  System.***out***.println(*maxSum*(stack1, stack2, stack3, n1, n2, n3));  }  **public** **static** **int** maxSum(**int** stack1[], **int** stack2[], **int** stack3[], **int** n1, **int** n2, **int** n3) {  **int** sum1 = 0, sum2 = 0, sum3 = 0;  **for** (**int** i = 0; i < n1; i++)  sum1 += stack1[i];  **for** (**int** i = 0; i < n2; i++)  sum2 += stack2[i];  **for** (**int** i = 0; i < n3; i++)  sum3 += stack3[i];  **int** top1 = 0, top2 = 0, top3 = 0;  **int** ans = 0;  **while** (**true**) {  **if** (top1 == n1 || top2 == n2 || top3 == n3) **return** 0;  **if** (sum1 == sum2 && sum2 == sum3) **return** sum1;  **if** (sum1 >= sum2 && sum1 >= sum3) sum1 -= stack1[top1++];  **else** **if** (sum2 >= sum3 && sum2 >= sum3) sum2 -= stack2[top2++];  **else** **if** (sum3 >= sum2 && sum3 >= sum1) sum3 -= stack3[top3++];  }  }  } |

# Maximum sum subarray having sum less than or equal to given sum

|  |
| --- |
| Input : arr[] = { 1, 2, 3, 4, 5 } sum = 11 Output : 10 Subarray having maximum sum is { 1, 2, 3, 4 }  Input : arr[] = { 2, 4, 6, 8, 10 } sum = 7 Output : 6 Subarray having maximum sum is { 2, 4 } or { 6 }  Time complecxity will be O (N\*N).  **public** **class** MaximumSumSubarray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 5 };  **int** n = arr.length;  **int** sum = 11;  System.***out***.println(*findMaxSubarraySum*(arr, n, sum));  }  **static** **int** findMaxSubarraySum(**int** arr[], **int** n, **int** sum) {  **int** curr\_sum = arr[0], max\_sum = 0, start = 0;  **for** (**int** i = 1; i < n; i++) {  **while** (curr\_sum > sum && start < i) {  curr\_sum -= arr[start];  start++;  }  max\_sum = Math.*max*(max\_sum, curr\_sum);  curr\_sum += arr[i];  }  **if** (curr\_sum <= sum)  max\_sum = Math.*max*(max\_sum, curr\_sum);  **return** max\_sum;  }  } |

# Maximum subarray sum in an array created after repeated concatenation

|  |
| --- |
| Input: arr [] = {-1, 10, 20}, k = 2 Output: 59 After concatenating array twice, we get {-1, 10, 20, -1, 10, 20} which has maximum subarray sum as 59.  Input : arr[] = {-1, -2, -3}, k = 3 Output : -1  **public** **class** MaximumSumSubarrayRepeated {  **public** **static** **void** main(String[] args) {  **int** a[] = { 10, 20, -30, -1 };  **int** n = a.length;  **int** k = 3;  System.***out***.println("Maximum contiguous sum is " + *maxSubArraySumRepeated*(a, n, k));  }  **static** **int** maxSubArraySumRepeated(**int** a[], **int** n, **int** k) {  **int** max\_so\_far = 0;  **int** INT\_MIN, max\_ending\_here = 0;  **for** (**int** i = 0; i < n \* k; i++) {  max\_ending\_here = max\_ending\_here + a[i % n];  **if** (max\_so\_far < max\_ending\_here)  max\_so\_far = max\_ending\_here;  **if** (max\_ending\_here < 0)  max\_ending\_here = 0;  }  **return** max\_so\_far;  }  } |

# Maximum Sum Subarray Using Dynamic Programming\_2

|  |
| --- |
| **public** **class** MaximumSumSubarrayUsingDynamicProgramming\_2 {  **public** **static** **void** main(String[] args) {  **int** a[] = { -2, -3, 4, -1, -2, 1, 5, -3 };  **int** n = a.length;  *maxSubArraySum*(a, n);  }  **static** **void** maxSubArraySum(**int** a[], **int** size) {  **int** max\_so\_far = Integer.***MIN\_VALUE***, max\_ending\_here = 0, start = 0, end = 0, s = 0;  **for** (**int** i = 0; i < size; i++) {  max\_ending\_here += a[i];  **if** (max\_so\_far < max\_ending\_here) {  max\_so\_far = max\_ending\_here;  start = s;  end = i;  }  **if** (max\_ending\_here < 0) {  max\_ending\_here = 0;  s = i + 1;  }  }  System.***out***.println("Maximum contiguous sum is " + max\_so\_far);  System.***out***.println("Starting index " + start);  System.***out***.println("Ending index " + end);  }  } |

# Maximum Sum Subarray Using Dynamic Programming

|  |
| --- |
| **public** **class** MaximumSumSubarrayUsingDynamicProgramming {  **public** **static** **void** main(String[] args) {  **int** a[] = { -2, -3, 4, -1, -2, 1, 5, -3 };  **int** n = a.length;  **int** max\_sum = *maxSubArraySum*(a, n);  System.***out***.println("Maximum contiguous sum is " + max\_sum);  }  **static** **int** maxSubArraySum(**int** a[], **int** size) {  **int** max\_so\_far = a[0];  **int** curr\_max = a[0];  **for** (**int** i = 1; i < size; i++) {  curr\_max = Math.*max*(a[i], curr\_max + a[i]);  max\_so\_far = Math.*max*(max\_so\_far, curr\_max);  }  **return** max\_so\_far;  }  } |

# Maximum Sum Subarray Using Kadane Algo

|  |
| --- |
| //Time Complexity: O(n)  **public** **class** MaximumSumSubarrayUsingKadaneAlgo {  **public** **static** **void** main(String[] args) {  **int**[] a = { -2, -3, 4, -1, -2, 1, 5, -3 };  System.***out***.println("Maximum contiguous sum is " + *maxSubArraySum*(a));  }  **static** **int** maxSubArraySum(**int** a[]) {  **int** size = a.length;  **int** max\_so\_far = Integer.***MIN\_VALUE***, max\_ending\_here = 0;  **for** (**int** i = 0; i < size; i++) {  max\_ending\_here = max\_ending\_here + a[i];  **if** (max\_so\_far < max\_ending\_here)  max\_so\_far = max\_ending\_here;  **if** (max\_ending\_here < 0)  max\_ending\_here = 0;  }  **return** max\_so\_far;  }  } |

# Maximum Sum Subarray Using Prefix Sum

|  |
| --- |
| Input1 : arr = {-2, -3, 4, -1, -2, 1, 5, -3} Output1 : 7  Input2 : arr = {4, -8, 9, -4, 1, -8, -1, 6} Output2 : 9  Time Complexity: O (n).  **public** **class** MaximumSumSubarrayUsingPrefixSum {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { -2, -3, 4, -1, -2, 1, 5, -3 };  **int** n1 = arr1.length;  System.***out***.println(*maximumSumSubarray*(arr1, n1));  **int** arr2[] = { 4, -8, 9, -4, 1, -8, -1, 6 };  **int** n2 = arr2.length;  System.***out***.println(*maximumSumSubarray*(arr2, n2));  }  **static** **int** maximumSumSubarray(**int** arr[], **int** n) {  **int** min\_prefix\_sum = 0;  **int** res = Integer.***MIN\_VALUE***;  **int** prefix\_sum[] = **new** **int**[n];  prefix\_sum[0] = arr[0];  **for** (**int** i = 1; i < n; i++)  prefix\_sum[i] = prefix\_sum[i - 1] + arr[i];  **for** (**int** i = 0; i < n; i++) {  res = Math.*max*(res, prefix\_sum[i] - min\_prefix\_sum);  min\_prefix\_sum = Math.*min*(min\_prefix\_sum, prefix\_sum[i]);  }  **return** res;  }  } |

# Largest sum subarray with at-least k numbers

|  |
| --- |
| Input : arr[] = {-4, -2, 1, -3} k = 2 Output : -1 The sub array is {-2, 1}  Input : arr[] = {1, 1, 1, 1, 1, 1} k = 2 Output : 6 The sub array is {1, 1, 1, 1, 1, 1} Time Complexity: O(n)  **public** **class** MaximumSumSubarrayWithK {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, -10, -3 };  **int** k = 4;  System.***out***.println(*maxSumWithK*(arr, arr.length, k));  }  **static** **int** maxSumWithK(**int** a[], **int** n, **int** k) {  **int** maxSum[] = **new** **int**[n];  maxSum[0] = a[0];  **int** curr\_max = a[0];  **for** (**int** i = 1; i < n; i++) {  curr\_max = Math.*max*(a[i], curr\_max + a[i]);  maxSum[i] = curr\_max;  }  **int** sum = 0;  **for** (**int** i = 0; i < k; i++)  sum += a[i];  **int** result = sum;  **for** (**int** i = k; i < n; i++) {  sum = sum + a[i] - a[i - k];  result = Math.*max*(result, sum);  result = Math.*max*(result, sum + maxSum[i - k]);  }  **return** result;  }  } |

# Maximum sum such that no two elements are adjacent

|  |
| --- |
| /\*Input : arr[] = {5, 5, 10, 100, 10, 5} Output : 110  Input : arr[] = {1, 2, 3} Output : 4  Input : arr[] = {1, 20, 3} Output : 20  **public** **class** Maximum\_sum\_such\_that\_no\_two\_elements\_are\_adjacent {  **public** **static** **void** main(String[] args) {  Maximum\_sum\_such\_that\_no\_two\_elements\_are\_adjacent sum = **new** Maximum\_sum\_such\_that\_no\_two\_elements\_are\_adjacent();  **int** arr[] = **new** **int**[] { 5, 5, 10, 100, 10, 5 };  System.***out***.println(sum.FindMaxSum(arr, arr.length));  }  **int** FindMaxSum(**int** arr[], **int** n) {  **int** incl = arr[0];  **int** excl = 0;  **int** excl\_new;  **int** i;  **for** (i = 1; i < n; i++) {  excl\_new = (incl > excl)? incl : excl;  incl = excl + arr[i];  excl = excl\_new;  }  **return** ((incl > excl) ? incl : excl);  }  } |

# Measure time Taken by a Function

|  |
| --- |
| **public** **class** Measure\_time\_taken\_by\_a\_function {  **public** **static** **void** main(String[] args) {  **long** start = System.*currentTimeMillis*();  *count\_function*(10000000);  **long** end = System.*currentTimeMillis*();  System.***out***.println("Counting to 10000000 takes " + (end - start) + "ms");  }  **public** **static** **void** count\_function(**long** x) {  System.***out***.println("Loop starts");  **for** (**long** i = 0; i < x; i++)  ;  System.***out***.println("Loop ends");  }  } |

# Memory Leaks

|  |
| --- |
| **public** **class** Memory\_leaks {  **public** **static** **void** main(String[] args) {  Vector v = **new** Vector(214444);  Vector v1 = **new** Vector(214744444);  Vector v2 = **new** Vector(2144444449);  System.***out***.println("Memory Leaks");  }  } |

# Merge Array: - merge array n [] of size n into mPlusN [] of size m+n

|  |
| --- |
| **public** **class** MergeArrays {  **public** **static** **void** main(String[] args) {  MergeArrays mergearray = **new** MergeArrays();  **int** mPlusN[] = {2, 8, -1, -1, -1, 13, -1, 15, 20};  **int** N[] = {5, 7, 9, 25};  **int** n = N.length;  **int** m = mPlusN.length - n;  mergearray.moveToEnd(mPlusN, m + n);  mergearray.merge(mPlusN, N, m, n);  mergearray.printArray(mPlusN, m + n);  }  **void** moveToEnd(**int** mPlusN[], **int** size){  **int** i, j = size - 1;  **for** (i = size - 1; i >= 0; i--){  **if** (mPlusN[i] != -1){  mPlusN[j] = mPlusN[i];  j--;  }  }  }  **void** merge(**int** mPlusN[], **int** N[], **int** m, **int** n){  **int** i = n;  **int** j = 0;  **int** k = 0;  **while** (k < (m + n)){  **if** ((i < (m + n) && mPlusN[i] <= N[j]) || (j == n)){  mPlusN[k] = mPlusN[i];  k++;  i++;  }**else** { // Otherwise take element from N[]  mPlusN[k] = N[j];  k++;  j++;  }  }  }  **void** printArray(**int** arr[], **int** size){  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Minimum sum of multiplications of n numbers

|  |
| --- |
| Input : 40 60 20 Output : 2400  Input : 5 6 Output : 30 Time Complexity: O(n^3) Auxiliary Space: O(n^2)  **public** **class** Minimum\_sum\_of\_multiplications {  **static** **long** *dp*[][] = **new** **long**[1000][1000];  **public** **static** **void** main(String[] args) {  **int** a[] = { 40, 60, 20 };  **int** n = a.length;  *intialize*(n);  System.***out***.println(*solve*(a, 0, n - 1));  }  **static** **void** intialize(**int** n) {  **for** (**int** i = 0; i <= n; i++)  **for** (**int** j = 0; j <= n; j++)  *dp*[i][j] = -1;  }  **static** **long** solve(**int** a[], **int** i, **int** j) {  **if** (i == j) **return** 0;  **if** (*dp*[i][j] != -1) **return** *dp*[i][j];  *dp*[i][j] = 100000000;  **for** (**int** k = i; k < j; k++) {  *dp*[i][j] = Math.*min*(*dp*[i][j], (*solve*(a, i, k) + *solve*(a, k + 1, j) + (*sum*(a, i, k) \* *sum*(a, k + 1, j))));  }  **return** *dp*[i][j];  }  **static** **long** sum(**int** a[], **int** i, **int** j) {  **long** ans = 0;  **for** (**int** m = i; m <= j; m++)  ans = (ans + a[m]) % 100;  **return** ans;  }  } |

# Minimum absolute difference of XOR values of two subarrays

|  |
| --- |
| Input : arr[] = {12, 6, 20, 14, 38, 6} Output : 16  The two subarrays are: {12, 6, 20} = 12 ^ 6 ^ 20 = 30 {14, 38, 6} = 14 ^ 38 ^ 6 = 46  Absolute difference = abs(30-46) = 16 Time Complexity: O(n)  **public** **class** MinimumAbsoluteDifference {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 6, 20, 14, 38, 6 };  **int** n = 6;  System.***out***.println("Minimum Absolute Difference = " + *minDiffBtwXorValues*(arr, n));  }  **public** **static** **int** minDiffBtwXorValues(**int** arr[], **int** n) {  **int** tot\_xor = 0;  **for** (**int** i = 0; i < n; i++)  tot\_xor ^= arr[i];  **int** part\_xor = 0, min = Integer.***MAX\_VALUE***;  **for** (**int** i = 0; i < n - 1; i++) {  tot\_xor ^= arr[i];  part\_xor ^= arr[i];  **if** (Math.*abs*(tot\_xor - part\_xor) < min)  min = Math.*abs*(tot\_xor - part\_xor);  }  **return** min;  }  } |

# Find minimum difference between any two elements

|  |
| --- |
| Input : {1, 5, 3, 19, 18, 25}; Output : 1 Minimum difference is between 18 and 19  Input : {30, 5, 20, 9}; Output : 4 Minimum difference is between 5 and 9  Simple: O(n2)  **public** **class** MinimumDifferenceBetweenTwoElements {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 1, 5, 3, 19, 18, 25 };  System.***out***.println("Minimum difference is " + *findMinDiff*(arr, arr.length));  }  **static** **int** findMinDiff(**int**[] arr, **int** n) {  **int** diff = Integer.***MAX\_VALUE***;  **for** (**int** i = 0; i < n - 1; i++)  **for** (**int** j = i + 1; j < n; j++)  **if** (Math.*abs*((arr[i] - arr[j])) < diff)  diff = Math.*abs*((arr[i] - arr[j]));  **return** diff;  }  } |

# Minimum Difference between Two Elements Efficient

|  |
| --- |
| //O(n) time  **public** **class** MinimumDifferenceBetweenTwoElementsEfficient {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 1, 5, 3, 19, 18, 25 };  System.***out***.println("Minimum difference is " + *findMinDiff*(arr, arr.length));  }  **static** **int** findMinDiff(**int**[] arr, **int** n) {  Arrays.*sort*(arr);  **int** diff = Integer.***MAX\_VALUE***;  **for** (**int** i = 0; i < n - 1; i++)  **if** (arr[i + 1] - arr[i] < diff)  diff = arr[i + 1] - arr[i];  **return** diff;  }  } |

# Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted

|  |
| --- |
| 1) If the input array is [10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60], find that the subarray lies between the indexes 3 and 8.  2) If the input array is [0, 1, 15, 25, 6, 7, 30, 40, 50], find that the subarray lies between the indexes 2 and 5.  Time Complexity: O(n)  **public** **class** MinimumLengthUnsortedSubarray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60 };  **int** arr\_size = arr.length;  *printUnsorted*(arr, arr\_size);  }  **static** **void** printUnsorted(**int** arr[], **int** n) {  **int** s = 0, e = n - 1, i, max, min;  **for** (s = 0; s < n - 1; s++) {  **if** (arr[s] > arr[s + 1])  **break**;  }  **if** (s == n - 1) {  System.***out***.println("The complete array is sorted");  **return**;  }  **for** (e = n - 1; e > 0; e--) {  **if** (arr[e] < arr[e - 1])  **break**;  }  max = arr[s];  min = arr[s];  **for** (i = s + 1; i <= e; i++) {  **if** (arr[i] > max)  max = arr[i];  **if** (arr[i] < min)  min = arr[i];  }  **for** (i = 0; i < s; i++) {  **if** (arr[i] > min) {  s = i;  **break**;  }  }  **for** (i = n - 1; i >= e + 1; i--) {  **if** (arr[i] < max) {  e = i;  **break**;  }  }  System.***out***.println(" The unsorted subarray which" + " makes the given array sorted lies"+ " between the indices " + s + " and " + e);  **return**;  }  } |

# Find the smallest positive number missing from an unsorted array

|  |
| --- |
| Input: {2, 3, 7, 6, 8, -1, -10, 15} Output: 1  Input: { 2, 3, -7, 6, 8, 1, -10, 15 } Output: 4  **public** **class** MissingNumber\_SmallestPositive {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 0, 10, 2, -10, -20 };  **int** arr\_size = arr.length;  **int** missing = *findMissing*(arr, arr\_size);  System.***out***.println("The smallest positive missing number is " + missing);  }  **static** **int** findMissing(**int** arr[], **int** size) {  **int** shift = *segregate*(arr, size);  **int** arr2[] = **new** **int**[size - shift];  **int** j = 0;  **for** (**int** i = shift; i < size; i++) {  arr2[j] = arr[i];  j++;  }  **return** *findMissingPositive*(arr2, j);  }  **static** **int** segregate(**int** arr[], **int** size) {  **int** j = 0, i;  **for** (i = 0; i < size; i++) {  **if** (arr[i] <= 0) {  **int** temp;  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  j++;  }  }  **return** j;  }  **static** **int** findMissingPositive(**int** arr[], **int** size) {  **int** i;  **for** (i = 0; i < size; i++) {  **if** (Math.*abs*(arr[i]) - 1 < size && arr[Math.*abs*(arr[i]) - 1] > 0)  arr[Math.*abs*(arr[i]) - 1] = -arr[Math.*abs*(arr[i]) - 1];  }  **for** (i = 0; i < size; i++)  **if** (arr[i] > 0)  **return** i + 1;  **return** size + 1;  }  } |

# Missing Number Using List

|  |
| --- |
| **public** **class** MissingNumber\_UsingList {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 13, 12, 11, 15 };  **int** n = arr.length;  System.***out***.println(*missingNum*(arr, n));  }  **static** **int** missingNum(**int** arr[], **int** n) {  List<Integer> list = **new** ArrayList<>(arr.length);  **for** (**int** i : arr) {  list.add(Integer.*valueOf*(i));  }  **int** minvalue = Collections.*min*(list);  ;  **int** xornum = 0;  **for** (**int** i = 0; i < n; i++) {  xornum ^= (minvalue) ^ arr[i];  minvalue++;  }  **return** xornum ^ minvalue;  }  } |

# Missing Number Using Sum Formula

|  |
| --- |
| **public** **class** MissingNumber\_UseSumFormula {  **public** **static** **void** main(String[] args) {  **int** a[] = {1,2,4,5,6};  **int** miss = *getMissingNo*(a,5);  System.***out***.println(miss);  }  **static** **int** getMissingNo (**int** a[], **int** n){  **int** i, total;  total = (n+1)\*(n+2)/2;  **for** ( i = 0; i< n; i++)  total -= a[i];  **return** total;  }  } |

# Missing Number Using Total Number

|  |
| --- |
| **public** **class** MissingNumber\_UseTotalNumber {  **public** **static** **void** main(String[] args) {  **int** n = 8;  **int** a [] = {1,4,5,3,7,8,6};  **int** sumOfNumbers = n \* (n+1)/2;  **int** sumOfElements = 0;  **for** (**int** i =0; i <a.length; i++){  sumOfElements = sumOfElements + a[i];  }  **int** missingNumber = sumOfNumbers - sumOfElements;  System.***out***.println(missingNumber);  }  } |

# Missing Number Using XOR

|  |
| --- |
| **public** **class** MissingNumber\_UsingXOR {  **public** **static** **void** main(String[] args) {  **int** a[] = { 1, 2, 4, 5, 6 };  **int** miss = *getMissingNo*(a, 5);  System.***out***.println(miss);  }  **static** **int** getMissingNo(**int** a[], **int** n) {  **int** i;  **int** x1 = a[0];  **int** x2 = 1;  **for** (i = 1; i < n; i++)  x1 = x1 ^ a[i];  **for** (i = 2; i <= n + 1; i++)  x2 = x2 ^ i;  **return** (x1 ^ x2);  }  } |

# Move All Zeroes To End

|  |
| --- |
| Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0}; Output : arr[] = {1, 2, 4, 3, 5, 0, 0};  Input : arr[] = {1, 2, 0, 0, 0, 3, 6}; Output : arr[] = {1, 2, 3, 6, 0, 0, 0};  Time complexity is O (n) and extra space is O (1).  **public** **class** MoveAllZeroesToEnd {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0, 9 };  **int** n = arr.length;  *pushZerosToEnd*(arr, n);  System.***out***.println("Array after pushing zeros to the back: ");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **void** pushZerosToEnd(**int** arr[], **int** n) {  **int** count = 0;  **for** (**int** i = 0; i < n; i++)  **if** (arr[i] != 0)  arr[count++] = arr[i];  **while** (count < n)  arr[count++] = 0;  }  } |

# Find if neat arrangement of cups and shelves can be made:- given three different types of cups(a[]) and saucers(b[],and n number of shelves,find if neat arrangement of cups and shelves can be made.Arrangement of the cups and saucers will be neat if it follows the below rules:- 1. No shelf can contain both cups and saucers. 2. There can be no more than 5 cups in any shelf. 3. There can be no more than 10 saucers in any shelf.

|  |
| --- |
| Input : a[] = {3, 2, 6} b[] = {4, 8, 9} n = 10 Output : Yes  Explanation : Total cups = 11, shelves required = 3 Total saucers = 21, shelves required = 3  Total required shelves = 3 + 3 = 6, which is less than given number of shelves n. So, output is Yes.  Input : a[] = {4, 7, 4} b[] = {3, 9, 10} n = 2 Output : No  **public** **class** NeatArrangementOfCups {  **public** **static** **void** main(String[] args) {  **int** a[] = { 3, 2, 6 };  **int** b[] = { 4, 8, 9 };  **int** n = 10;  *canArrange*(a, b, n);  }  **public** **static** **void** canArrange(**int** a[], **int** b[], **int** n) {  **int** suma = 0, sumb = 0;  **for** (**int** i = 0; i < 2; i++)  suma += a[i];  **for** (**int** i = 0; i < 2; i++)  sumb += b[i];  **int** na = (suma + 5 - 1) / 5;  **int** nb = (sumb + 10 - 1) / 10;  **if** (na + nb <= n)  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  } |

# Number Format Demo I18N

|  |
| --- |
| NumberFormat is an abstract base class for all number formats. This class provides the interface for formatting and parsing numbers. NumberFormat also provides methods for determining which locales (US, India, Italy, etc) have number formats, and what their names are. NumberFormat helps you to format and parse numbers for any locale.**Some important points about NumberFormat class:**   * NumberFormat class is present in java.text package and it is an abstract class. * NumberFormat class implements Serializable, Cloneable. * NumberFormat is the direct child class of Format class. * Number formats are generally not synchronized. It is recommended to create separate format instances for each thread. If multiple threads access a format concurrently, it must be synchronized externally.   **Methods present in NumberFormat class:**   * **public static NumberFormat getInstance();:** To get the NumberFormat object for default Locale. * **public static NumberFormat getCurrencyInstance();:** To get the NumberFormat object for default Locale to represent in specific Currency. * **public static NumberFormat getPercentInstance();:** * **public static NumberFormat getInstance(Locale l);:** To get the NumberFormat object for the specified Locale object. * **public static format(long l);:**To convert java number to locale object.   public class NumberFormatDemo\_I18N {  public static void main(String[] args) {  double d = 123456.789;  NumberFormat nf = NumberFormat.getInstance(Locale.ITALY);  NumberFormat nf1 = NumberFormat.getInstance(Locale.US);  NumberFormat nf2 = NumberFormat.getInstance(Locale.CHINA);  System.out.println("ITALY representation of " + d + " : " + nf.format(d));  System.out.println("US representation of " + d + " : " + nf1.format(d));  System.out.println("CHINA representation of " + d + " : " + nf2.format(d));  }  } |

# Number of horizontal or vertical line segments to connect 3 points

|  |
| --- |
| Input : A = {-1, -1}, B = {-1, 3}, C = {4, 3} Output : 2  Input :A = {1, 1}, B = {2, 3} C = {3, 2} Output : 3  **public** **class** NumberOfHorizontalOrVerticalLine {  **public** **static** **void** main(String[] args) {  **int** x[] = **new** **int**[3], y[] = **new** **int**[3];  x[0] = -1;  y[0] = -1;  x[1] = -1;  y[1] = 3;  x[2] = 4;  y[2] = 3;  System.***out***.println(*countLineSegments*(x, y));  }  **static** **int** countLineSegments(**int** x[], **int** y[]) {  **if** ((x[0] == x[1] && x[1] == x[2]) || (y[0] == y[1] && y[1] == y[2]))  **return** 1;  **else** **if** (*canJoin*(x, y, 0, 1, 2) || *canJoin*(x, y, 0, 2, 1) || *canJoin*(x, y, 1, 2, 0))  **return** 2;  **else**  **return** 3;  }  **static** **boolean** canJoin(**int** x[], **int** y[], **int** i, **int** j, **int** k) {  **return** (x[k] == x[i] || x[k] == x[j]) && *isBetween*(y[i], y[j], y[k])  || (y[k] == y[i] || y[k] == y[j]) && *isBetween*(x[i], x[j], x[k]);  }  **static** **boolean** isBetween(**int** a, **int** b, **int** c) {  **return** (Math.*min*(a, b) <= c && c <= Math.*max*(a, b));  }  } |

# Find number of pairs in an array such that their XOR is 0

|  |
| --- |
| Input : A[] = {1, 3, 4, 1, 4} Output : 2 Explanation : Index (0, 3) and (2, 4)  Input : A[] = {2, 2, 2} Output : 3 Time Complexity : O(N)  **public** **class** NumberOfPairsInAnArray {  **public** **static** **void** main(String[] args) {  **int** a[] = { 1, 2, 1, 2, 4 };  **int** n = a.length;  System.***out***.println(*calculate*(a, n));  }  **static** **int** calculate(**int** a[], **int** n) {  Arrays.*sort*(a);  **int** count = 1;  **int** answer = 0;  **for** (**int** i = 1; i < n; i++) {  **if** (a[i] == a[i - 1]) {  count += 1;  } **else** {  answer = answer + (count \* (count - 1)) / 2;  count = 1;  }  }  answer = answer + (count \* (count - 1)) / 2;  **return** answer;  }  } |

# Numbers with prime frequencies

|  |
| --- |
| Input : int[] arr = { 11, 11, 11, 23, 11, 37, 51, 37, 37, 51, 51, 51, 51 }; k = 2 Output : 37, 51  Explanation: 11's count is 4, 23 count 1, 37 count 3, 51 count 5. 37 and 51 are two number that appear prime number of time and frequencies greater than or equal to k.  Input : int[] arr = { 11, 22, 33 } min Occurrence = 1 Output : -1 None of the count is prime number of times  **public** **class** Numberswithprimefrequencies {  **public** **static** **void** main(String[] args) {  **int**[] arr = { 11, 11, 11, 23, 11, 37, 37, 51, 51, 51, 51, 51 };  **int** k = 2;  *primeOccurences*(arr, k);  }  **static** **void** primeOccurences(**int**[] arr, **int** k) {  Map<Integer, Integer> map = **new** HashMap<>();  **for** (**int** i = 0; i < arr.length; i++) {  **int** val = arr[i];  **int** freq;  **if** (map.containsKey(val)) {  freq = map.get(val);  freq++;  } **else**  freq = 1;  map.put(val, freq);  }  **for** (Map.Entry<Integer, Integer> entry : map.entrySet()) {  **int** value = entry.getValue();  **if** (*isPrime*(value) && value >= k)  System.***out***.println(entry.getKey());  }  }  **private** **static** **boolean** isPrime(**int** n) {  **if** ((n > 2 && n % 2 == 0) || n == 1)  **return** **false**;  **for** (**int** i = 3; i <= (**int**) Math.*sqrt*(n); i += 2) {  **if** (n % i == 0)  **return** **false**;  }  **return** **true**;  }  } |

# Number or Not

|  |
| --- |
| **public** **class** NumberOrNot {  **public** **static** **void** main(String[] args) {  Scanner reader = **new** Scanner(System.***in***);  System.***out***.print("Enter a data: ");  String input = reader.next();  **boolean** out = *isNumberOrNot*(input);  System.***out***.println(out);  }  **public** **static** **boolean** isNumberOrNot(String input){  **try**{  Integer.*parseInt*(input);  }**catch**(NumberFormatException e){  **return** **false**;  }  **return** **true**;  }  } |

# Occurring Odd Number with Bitwise XOR

|  |
| --- |
| Time Complexity: O(n)  **public** **class** OccurringOddNumberWithBitwiseXOR {  **public** **static** **void** main(String[] args) {  OccurringOddNumberWithBitwiseXOR occur = **new** OccurringOddNumberWithBitwiseXOR();  **int** ar[] = **new** **int**[] { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };  **int** n = ar.length;  System.***out***.println(occur.getOddOccurrence(ar, n));  }  **int** getOddOccurrence(**int** ar[], **int** ar\_size) {  **int** i;  **int** res = 0;  **for** (i = 0; i < ar\_size; i++) {  res = res ^ ar[i];  }  **return** res;  }  } |

# Occurring Odd Number with Hashing

|  |
| --- |
| //Time complexity of this solution is O(n)  **public** **class** OccurringOddNumberWithHashing {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };  **int** n = arr.length;  System.***out***.println(*getOddOccurrence*(arr, n));  }  **static** **int** getOddOccurrence(**int** arr[], **int** n) {  HashMap<Integer, Integer> hmap = **new** HashMap<>();  **for** (**int** i = 0; i < n; i++) {  **if** (hmap.containsKey(arr[i])) {  **int** val = hmap.get(arr[i]);  hmap.put(arr[i], val + 1);  } **else**  hmap.put(arr[i], 1);  }  **for** (Integer a : hmap.keySet()) {  **if** (hmap.get(a) % 2 != 0)  **return** a;  }  **return** -1;  }  } |

# Occurring Odd Number without Hashing

|  |
| --- |
| //Time complexity of this solution is O (n2).  **public** **class** OccurringOddNumberWithoutHashing {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2 };  **int** n = arr.length;  System.***out***.println(*getOddOccurrence*(arr, n));  }  **static** **int** getOddOccurrence(**int** arr[], **int** arr\_size) {  **int** i;  **for** (i = 0; i < arr\_size; i++) {  **int** count = 0;  **for** (**int** j = 0; j < arr\_size; j++) {  **if** (arr[i] == arr[j])  count++;  }  **if** (count % 2 != 0)  **return** arr[i];  }  **return** -1;  }  } |

# Find the two numbers with odd occurrences in an unsorted array

|  |
| --- |
| Time Complexity: O(n) Auxiliary Space: O(1)  **public** **class** OccurringOddNumberWithXOR {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 4, 2, 4, 5, 2, 3, 3, 1 };  **int** arr\_size = arr.length;  *printTwoOdd*(arr, arr\_size);  }  **static** **void** printTwoOdd(**int** arr[], **int** size) {  **int** xor2 = arr[0];  **int** set\_bit\_no;  **int** i;  **int** n = size - 2;  **int** x = 0, y = 0;  **for** (i = 1; i < size; i++)  xor2 = xor2 ^ arr[i];  set\_bit\_no = xor2 & ~(xor2 - 1);  **for** (i = 0; i < size; i++) {  **if** ((arr[i] & set\_bit\_no) > 0)  x = x ^ arr[i];  **else**  y = y ^ arr[i];  }  System.***out***.println("The two ODD elements are " + x + " & " + y);  }  } |

# Optional Class | isPresent (): - the isPresent () function in Optional class is used to evaluate whether the value if assigned to variable is present or not.

|  |
| --- |
| Input : Optional value1 = Optional.ofNullable(10); value1.isPresent() Output : True  Input : Optional value2 = Optional.ofNullable(null); value2.isPresent() Output : False  **public** **class** OptionalClass {  **public** **static** **void** main(String[] args) {  Optional<Integer> value1 = Optional.*ofNullable*(10);  Optional<Integer> value2 = Optional.*ofNullable*(**null**);  System.***out***.println("First parameter is present:" + value1.isPresent());  System.***out***.println("Second parameter is present:" + value2.isPresent());  }  } |

# Overloading Of Thread Class Run Method

|  |
| --- |
| **public** **class** Overloading\_Of\_Thread\_Class\_Run\_Method {  **public** **static** **void** main(String[] args) {  G t = **new** G();  t.start();  }  }  **class** G **extends** Thread {  **public** **void** run() {  System.***out***.println("GeeksforGeeks"); // GeeksforGeeks  }  **public** **void** run(**int** i) {  System.***out***.println("Bishal");  }  } |

# Overloading Of Thread Class Start Method

|  |
| --- |
| **public** **class** Overloading\_Of\_Thread\_Class\_Start\_Method {  **public** **static** **void** main(String[] args) {  Bishal thread = **new** Bishal();  thread.start();  System.***out***.println("Main Method");  }  }  **class** Bishal **extends** Thread {  **public** **void** start() {  System.***out***.println("Start Method");  }  **public** **void** run() {  System.***out***.println("Run Method");  }  } |

# Overriding Default Method

|  |
| --- |
| // Hello.java  package a;  public class Hello {  void printMessage() {  System.out.println("Hello");  }  }  // World.java  package b;  import a.Hello;  public class World extends Hello {  void printMessage() {  System.out.println("World");  }  public static void main(String[] args) {  Hello gfg = new World();  gfg.printMessage();  }  }  //output :- error: printMessage() is not public in Hello; cannot be accessed from outside package\*/ |

# Overriding Private Method

|  |
| --- |
| How JVM find which method to call? When we run a java program,   * JVM checks the runtime class of the object. * JVM checks whether the object’s runtime class has overridden the method of the declared class. * If so, that’s the method called. Otherwise, declared class’s method is called.   Hello.java  package a;  public class Hello {  private void printMessage() {  System.out.println("Hello");  }  public void fun() {  printMessage();  }  }  World.java  package b;  import a.Hello;  public class World extends Hello {  private void printMessage() {  System.out.println("World");  }  public static void main(String[] args) {  Hello gfg = new World();  gfg.fun();  }  }  //Output: Hello\*/ |

# Overriding Public Method

|  |
| --- |
| //Hello.java  package a;  public class Hello {  public void printMessage() {  System.out.println("Hello");  }  }  //World.java  package b;  import a.Hello;  public class World extends Hello {  public void printMessage() {  System.out.println("World");  }  public static void main(String[] args) {  Hello gfg = new World();  gfg.printMessage();  }  }  //Output: World\*/ |

# Given a sorted array and a number x, find the pair in array whose sum is closest to x

|  |
| --- |
| Input: arr[] = {10, 22, 28, 29, 30, 40}, x = 54 Output: 22 and 30  Input: arr[] = {1, 3, 4, 7, 10}, x = 15 Output: 4 and 10 Time complexity is O(n2)  **public** **class** PairInArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 22, 28, 29, 30, 40 }, x = 54;  **int** n = arr.length;  *printClosest*(arr, n, x);  }  **static** **void** printClosest(**int** arr[], **int** n, **int** x) {  **int** res\_l = 0, res\_r = 0;  **int** l = 0, r = n - 1, diff = Integer.***MAX\_VALUE***;  **while** (r > l) {  **if** (Math.*abs*(arr[l] + arr[r] - x) < diff) {  res\_l = l;  res\_r = r;  diff = Math.*abs*(arr[l] + arr[r] - x);  }  **if** (arr[l] + arr[r] > x)  r--;  **else**  l++;  }  System.***out***.println(" The closest pair is " + arr[res\_l] + " and " + arr[res\_r]);  }  } |

# Position of Element after Stable Sort: - given an array of integers which may contain duplicate elements, an element of this array is given to us, we need to tell the final position of this element in the array, if a stable sort algorithm is applied.

|  |
| --- |
| Input : arr[] = [3, 4, 3, 5, 2, 3, 4, 3, 1, 5], index = 5 Output : 4  **public** **class** PositionOfElementAfterStableSort {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 4, 3, 5, 2, 3, 4, 3, 1, 5 };  **int** n = arr.length;  **int** idxOfEle = 5;  System.***out***.println(*getIndexInSortedArray*(arr, n, idxOfEle));  }  **static** **int** getIndexInSortedArray(**int** arr[], **int** n, **int** idx) {  **int** result = 0;  **for** (**int** i = 0; i < n; i++) {  **if** (arr[i] < arr[idx])  result++;  **if** (arr[i] == arr[idx] && i < idx)  result++;  }  **return** result;  }  } |

# Prefix Sum Array – Implementation and Applications in Competitive Programming

|  |
| --- |
| Input : arr[] = {10, 20, 10, 5, 15} Output : prefixSum[] = {10, 30, 40, 45, 60}  Explanation: While traversing the array, update the element by adding it with its previous element.  prefixSum [0] = 10, prefixSum[1] = prefixSum[0] + arr[1] = 30, prefixSum[2] = prefixSum[1] + arr[2] = 40 and so on.\*/  **public** **class** PrefixSumArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 4, 16, 20 };  **int** n = arr.length;  **int** prefixSum[] = **new** **int**[n];  *fillPrefixSum*(arr, n, prefixSum);  **for** (**int** i = 0; i < n; i++)  System.***out***.print(prefixSum[i] + " ");  System.***out***.println("");  }  **static** **void** fillPrefixSum(**int** arr[], **int** n, **int** prefixSum[]) {  prefixSum[0] = arr[0];  **for** (**int** i = 1; i < n; ++i)  prefixSum[i] = prefixSum[i - 1] + arr[i];  }  } |

# Prime Number

|  |
| --- |
| **public** **class** PrimeNumber {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.println("Enter a integer: ");  **int** n = r.nextInt();  **boolean** out = *isPrimeNumber*(n);  System.***out***.println(out);  }  **public** **static** **boolean** isPrimeNumber(**int** n) {  **for** (**int** i = 2; i <= Math.*sqrt*(n); i++) {  **if** (n % i == 0) {  **return** **false**;  }  }  **return** **true**;  }  } |

# Print 2DArray foreach

|  |
| --- |
| **public** **class** Print\_2DArray\_foreach {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };  *print2D*(mat);  }  **public** **static** **void** print2D(**int** mat[][]) {  **for** (**int**[] row : mat)  **for** (**int** x : row)  System.***out***.print(x + " ");  }  } |

# Print 2DArray Simple

|  |
| --- |
| **public** **class** Print\_2DArray\_Simple {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };  *print2D*(mat);  }  **public** **static** **void** print2D(**int** mat[][]) {  **for** (**int** i = 0; i < mat.length; i++)  **for** (**int** j = 0; j < mat[i].length; j++)  System.***out***.print(mat[i][j] + " ");  }  } |

# Print 2DArray toString

|  |
| --- |
| **public** **class** Print\_2DArray\_toString {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 } };  *print2D*(mat);  }  **public** **static** **void** print2D(**int** mat[][]) {  **for** (**int**[] row : mat)  System.***out***.println(Arrays.*toString*(row));  }  } |

# Print all combinations of factors (Ways to factorize):- write a program to print all the combinations of factors of given number n.

|  |
| --- |
| Input : 16  Output :2 2 2 2  2 2 4  2 8  4 4  Input : 12  Output : 2 2 3  2 6  3 4  **public** **class** Print\_all\_combinations\_of\_factors {  **public** **static** **void** main(String[] args) {  **int** n = 16;  List<List<Integer>> resultant = *factComb*(n);  **for** (List<Integer> i : resultant) {  **for** (**int** j : i) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  }  **public** **static** List<List<Integer>> factComb(**int** n) {  List<List<Integer>> result\_list = **new** ArrayList<List<Integer>>();  List<Integer> list = **new** ArrayList<Integer>();  *factorsListFunc*(2, 1, n, result\_list, list);  **return** result\_list;  }  **public** **static** **void** factorsListFunc(**int** first, **int** each\_prod, **int** n, List<List<Integer>> result\_list,  List<Integer> single\_result\_list) {  **if** (first > n || each\_prod > n)  **return**;  **if** (each\_prod == n) {  ArrayList<Integer> t = **new** ArrayList<Integer>(single\_result\_list);  result\_list.add(t);  **return**;  }  **for** (**int** i = first; i < n; i++) {  **if** (i \* each\_prod > n)  **break**;  **if** (n % i == 0) {  single\_result\_list.add(i);  *factorsListFunc*(i, i \* each\_prod, n, result\_list, single\_result\_list);  single\_result\_list.remove(single\_result\_list.size() - 1);  }  }  }  } |

# Efficient program to print all prime factors of a given number: - if the input number is 12, then output should be “2 2 3” and if the input number is 315, then output should be “3 3 5 7”.

|  |
| --- |
| **public** **class** print\_all\_prime\_factors {  **public** **static** **void** main(String[] args) {  **int** n = 315;  *primeFactors*(n);  }  **public** **static** **void** primeFactors(**int** n) {  **while** (n % 2 == 0) {  System.***out***.print(2 + " ");  n /= 2;  }  **for** (**int** i = 3; i <= Math.*sqrt*(n); i += 2) {  **while** (n % i == 0) {  System.***out***.print(i + " ");  n /= i;  }  }  **if** (n > 2)  System.***out***.print(n);  }  } |

# Print Intermediate value in an array

|  |
| --- |
| /\*Input : arr[] = { 4, 2, 7, 5};  Output : Intermediate elements between 4 and 2 :- 2 3 4  Intermediate elements between 2 and 7 :- 2 3 4 5 6 7  Intermediate elements between 7 and 5 :- 5 6 7\*/  **public** **class** Print\_intermediate\_values\_in\_an\_array {  **public** **static** **void** main(String[] args) {  **int**[] arr = { 4, 2, 7, 5 };  *inter*(arr);  }  **static** **void** inter(**int**[] arr) {  **for** (**int** l = 0; l < arr.length - 1; l++) {  **int** i = arr[l];  **int** j = arr[l + 1];  **int** big = i > j ? i : j;  **int** sml = i < j ? i : j;  System.***out***.println("Intermediate elements between " + i + " and " + j);  **for** (**int** k = sml; k <= big; k++)  System.***out***.print(k + " ");  System.***out***.println();  }  }  } |

# Print number of words, vowels and frequency of each character

|  |
| --- |
| Input: How Good GOD Is.  Output : Number of words = 4 Number of vowels = 5 Number of upper case characters = 6  Character = Frequency = 3 Character =. Frequency = 1 Character = D Frequency = 1 Character = G Frequency = 2  Character = H Frequency = 1 Character = I Frequency = 1 Character = O Frequency = 1 Character = d Frequency = 1  Character = o Frequency = 3 Character = s Frequency = 1 Character = w Frequency = 1  Time Complexity: O (n), where n is the number of characters in the string. Auxiliary Space: O (1).  **public** **class** Print\_Number\_Of\_Words\_Vowels\_And\_Frequency {  String str = "Geeks for Geeks.”;  **public** **static** **void** main(String[] args) {  Print\_Number\_Of\_Words\_Vowels\_And\_Frequency obj = **new** Print\_Number\_Of\_Words\_Vowels\_And\_Frequency();  obj.words();  obj.frequency();  }  **void** words() {  **int** wCount = 0, uCount = 0, vCount = 0;  **for** (**int** i = 0; i < str.length(); i++) {  **char** c = str.charAt(i);  **switch** (c) {  **case** ' ':  **case** '.':  wCount++;  }  **switch** (c) {  **case** 'A':  **case** 'E':  **case** 'I':  **case** 'O':  **case** 'U':  **case** 'a':  **case** 'e':  **case** 'i':  **case** 'o':  **case** 'u':  vCount++;  }  **if** (c >= 65 && c <= 90) {  uCount++;  }  }  System.***out***.println("Number of words = " + wCount);  System.***out***.println("Number of vowels = " + vCount);  System.***out***.println("Number of upper case characters = " + uCount);  }  **void** frequency() {  TreeMap<Character, Integer> hmap = **new** TreeMap<Character, Integer>();  **for** (**int** i = 0; i < str.length(); i++) {  Integer c = hmap.get(str.charAt(i));  **if** (hmap.get(str.charAt(i)) == **null**)  hmap.put(str.charAt(i), 1);  **else**  hmap.put(str.charAt(i), ++c);  }  **for** (Map.Entry m : hmap.entrySet())  System.***out***.println("Character = " + m.getKey() + " Frequency = " + m.getValue());  }  } |

# Print All Distinct Elements of a given integer array: - given an integer array, print all distinct elements in array.the given array may contain duplicates and the output should print every element only once. The given array is not sorted.

|  |
| --- |
| Input: arr[] = {12, 10, 9, 45, 2, 10, 10, 45} Output: 12, 10, 9, 45, 2  Input: arr[] = {1, 2, 3, 4, 5} Output: 1, 2, 3, 4, 5  Input: arr[] = {1, 1, 1, 1, 1} Output: 1  **public** **class** PrintAllDistinctElements {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 5, 3, 4, 3, 5, 6 };  *printDistinct*(arr);  }  **static** **void** printDistinct(**int** arr[]) {  HashSet<Integer> set = **new** HashSet<>();  **for** (**int** i = 0; i < arr.length; i++) {  **if** (!set.contains(arr[i])) {  set.add(arr[i]);  System.***out***.print(arr[i] + " ");  }  }  }  } |

# Print All Duplicates

|  |
| --- |
| //Time Complexity: O(n)  **public** **class** PrintAllDuplicates {  **static** **final** **int** ***NO\_OF\_CHARS*** = 256;  **public** **static** **void** main(String[] args) {  String str = "test string";  *printDups*(str);  }  **static** **void** printDups(String str) {  **int** count[] = **new** **int**[***NO\_OF\_CHARS***];  *fillCharCounts*(str, count);  **for** (**int** i = 0; i < ***NO\_OF\_CHARS***; i++)  **if** (count[i] > 1)  System.***out***.printf("%c, count = %d \n", i, count[i]);  }  **static** **void** fillCharCounts(String str, **int**[] count) {  **for** (**int** i = 0; i < str.length(); i++)  count[str.charAt(i)]++;  }  } |

# Print Frequencies of All Elements Method1

|  |
| --- |
| **public** **class** PrintFrequenciesOfAllElements\_Method\_1 {  **public** **static** **void** main(String[] args) {  PrintFrequenciesOfAllElements\_Method\_1 count = **new** PrintFrequenciesOfAllElements\_Method\_1();  **int** arr[] = { 2, 3, 3, 2, 5 };  count.findCounts(arr, arr.length);  **int** arr1[] = { 1 };  count.findCounts(arr1, arr1.length);  **int** arr3[] = { 4, 4, 4, 4 };  count.findCounts(arr3, arr3.length);  **int** arr2[] = { 1, 3, 5, 7, 9, 1, 3, 5, 7, 9, 1 };  count.findCounts(arr2, arr2.length);  }  **void** findCounts(**int** arr[], **int** n) {  **int** i = 0;  **while** (i < n) {  **if** (arr[i] <= 0) {  i++;  **continue**;  }  **int** elementIndex = arr[i] - 1;  **if** (arr[elementIndex] > 0) {  arr[i] = arr[elementIndex];  arr[elementIndex] = -1;  } **else** {  arr[elementIndex]--;  arr[i] = 0;  i++;  }  }  System.***out***.println("Below are counts of all elements");  **for** (**int** j = 0; j < n; j++)  System.***out***.println(j + 1 + "->" + Math.*abs*(arr[j]));  }  } |

# Print Frequencies of All Elements Method2 by adding n to keep track of counts.

|  |
| --- |
| **public** **class** PrintFrequenciesOfAllElements\_Method\_2 {  **public** **static** **void** main(String[] args) {  PrintFrequenciesOfAllElements\_Method\_2 count = **new** PrintFrequenciesOfAllElements\_Method\_2();  **int** arr[] = { 2, 3, 3, 2, 5 };  **int** n = arr.length;  count.printfrequency(arr, n);  }  **void** printfrequency(**int** arr[], **int** n) {  **for** (**int** j = 0; j < n; j++)  arr[j] = arr[j] - 1;  **for** (**int** i = 0; i < n; i++)  arr[arr[i] % n] = arr[arr[i] % n] + n;  **for** (**int** i = 0; i < n; i++)  System.***out***.println(i + 1 + "->" + arr[i] / n);  }  } |

# Probability of reaching a point with 2 or 3 steps at a time: - A person starts walking from position X=0, find the probability to each exactly on X=N if she can only take either 2 steps or 3 steps.Probability from step length 2 is given i.e. P, probability for step length 3 is 1-P.

|  |
| --- |
| Input : N = 5, P = 0.20 Output : 0.32 Explanation: - There are two ways to reach 5. 2+3 with probability = 0.2 \* 0.8 = 0.16 3+2 with probability = 0.8 \* 0.2 = 0.16 So, total probability = 0.32.  **public** **class** Probability\_of\_reaching\_a\_point {  **public** **static** **void** main(String[] args) {  **int** n = 5;  **float** p = 0.2f;  System.***out***.printf("%.2f", *find\_prob*(n, p));  }  **static** **float** find\_prob(**int** N, **float** P) {  **double** dp[] = **new** **double**[N + 1];  dp[0] = 1;  dp[1] = 0;  dp[2] = P;  dp[3] = 1 - P;  **for** (**int** i = 4; i <= N; ++i)  dp[i] = (P) \* dp[i - 2] + (1 - P) \* dp[i - 3];  **return** ((**float**) (dp[N]));  }  } |

# Producer Consumer Problem

|  |
| --- |
| **public** **class** Producer\_Consumer\_Problem {  **public** **static** **void** main(String[] args) **throws** InterruptedException {  **final** PC pc = **new** PC();  Thread t1 = **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **try** {  pc.produce();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  });  Thread t2 = **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **try** {  pc.consume();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  });  t1.start();  t2.start();  t1.join();  t2.join();  }  **public** **static** **class** PC {  LinkedList<Integer> list = **new** LinkedList<>();  **int** capacity = 2;  **public** **void** produce() **throws** InterruptedException {  **int** value = 0;  **while** (**true**) {  **synchronized** (**this**) {  **while** (list.size() == capacity)  wait();  System.***out***.println("Producer produced-" + value);  list.add(value++);  notify();  Thread.*sleep*(1000);  }  }  }  **public** **void** consume() **throws** InterruptedException {  **while** (**true**) {  **synchronized** (**this**) {  **while** (list.size() == 0)  wait();  **int** val = list.removeFirst();  System.***out***.println("Consumer consumed-" + val);  notify();  Thread.*sleep*(1000);  }  }  }  }  } |

# Random nextInt

|  |
| --- |
| **Java.util.Random.nextInt() in Java :-**  **1.** **Java.util.Random.nextInt ():** The nextInt () is used to get the next random integer value from this random number generator’s sequence.  Declaration : public int nextInt()  Parameters : NA  Return Value : The method call returns the next integer number from the sequence  Exception : NA  **2. Java.util.Random.nextInt (int n):** The nextInt (int n) is used to get a random number between 0(inclusive) and the number passed in this argument(n), exclusive.  Declaration : public int nextInt(int n)  Parameters: n: This is the bound on the random number to be returned. Must be positive.  Return Value: Returns a random number. Between 0 (inclusive) and n (exclusive).  Exception: IllegalArgumentException: This is thrown if n is not positive.  **public** **class** Random\_nextInt {  **public** **static** **void** main(String[] args) {  Random ran = **new** Random();  **int** nxt = ran.nextInt();  System.***out***.println("The Randomly generated integer is : " + nxt);  **int** nxt1 = ran.nextInt(10);  System.***out***.println("Random number between 0 and 10 is : " + nxt1);  }  } |

# Read Mobile Numbers File

|  |
| --- |
| **public** **class** Read\_Mobile\_Numbers\_File {  **public** **static** **void** main(String[] args) **throws** IOException {  PrintWriter pw = **new** PrintWriter("C:/Users/136342/workspace/Programming\_Projects/output.txt");  Pattern p = Pattern.*compile*("(0/91)?[7-9][0-9]{9}");  BufferedReader br = **new** BufferedReader(  **new** FileReader("C:/Users/136342/workspace/Programming\_Projects/input.txt"));  String line = br.readLine();  **while** (line != **null**) {  Matcher m = p.matcher(line);  **while** (m.find()) {  pw.println(m.group());  }  line = br.readLine();  }  pw.flush();  }  } |

# Rearrange an array such that ‘arr[j]’ becomes ‘i’ if ‘arr[i]’ is ‘j’:- given an array of size n where all elements are in range from 0 to n-1, change contents of arr [] so that arr[i] =j is changed to arr[j] = i.

|  |
| --- |
| Input: arr[] = {1, 3, 0, 2}; Output: arr[] = {2, 0, 3, 1};  Input: arr[] = {2, 0, 1, 4, 5, 3}; Output: arr[] = {1, 2, 0, 5, 3, 4}; Time complexity is O(n) and auxiliary space needed is O(n).  **public** **class** RearrangeAnArray\_Set2 {  **public** **static** **void** main(String[] args) {  RearrangeAnArray\_Set2 arrange = **new** RearrangeAnArray\_Set2();  **int** arr[] = { 1, 3, 0, 2 };  **int** n = arr.length;  System.***out***.println("Given array is ");  arrange.printArray(arr, n);  arrange.rearrangeNaive(arr, n);  System.***out***.println("Modified array is ");  arrange.printArray(arr, n);  }  **void** rearrangeNaive(**int** arr[], **int** n) {  **int** temp[] = **new** **int**[n];  **int** i;  **for** (i = 0; i < n; i++)  temp[arr[i]] = i;  **for** (i = 0; i < n; i++)  arr[i] = temp[i];  }  **void** printArray(**int** arr[], **int** n) {  **int** i;  **for** (i = 0; i < n; i++) {  System.***out***.print(arr[i] + " ");  }  System.***out***.println("");  }  } |

# Rearrange an Array Set3

|  |
| --- |
| **public** **class** RearrangeAnArray\_Set3 {  **public** **static** **void** main(String[] args) {  RearrangeAnArray\_Set3 arrange = **new** RearrangeAnArray\_Set3();  **int** arr[] = { 2, 0, 1, 4, 5, 3 };  **int** n = arr.length;  System.***out***.println("Given array is ");  arrange.printArray(arr, n);  arrange.rearrange(arr, n);  System.***out***.println("Modified array is ");  arrange.printArray(arr, n);  }  **void** rearrange(**int** arr[], **int** n) {  **int** i;  **for** (i = 0; i < n; i++)  arr[i]++;  **for** (i = 0; i < n; i++) {  **if** (arr[i] > 0)  rearrangeUtil(arr, n, i);  }  **for** (i = 0; i < n; i++)  arr[i] = (-arr[i]) - 1;  }  **void** rearrangeUtil(**int** arr[], **int** n, **int** i) {  **int** val = -(i + 1);  i = arr[i] - 1;  **while** (arr[i] > 0) {  **int** new\_i = arr[i] - 1;  arr[i] = val;  val = -(i + 1);  i = new\_i;  }  }  **void** printArray(**int** arr[], **int** n) {  **int** i;  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Rearrange array in alternating positive & negative items with O (1) extra space

|  |
| --- |
| Input: arr[] = {1, 2, 3, -4, -1, 4} Output: arr[] = {-4, 1, -1, 2, 3, 4}  Input: arr[] = {-5, -2, 5, 2, 4, 7, 1, 8, 0, -8} output: arr[] = {-5, 5, -2, 2, -8, 4, 7, 1, 8, 0}  **public** **class** RearrangeArray\_Set1 {  **public** **static** **void** main(String[] args) {  RearrangeArray\_Set1 rearrange = **new** RearrangeArray\_Set1();  **int** arr[] = { -5, -2, 5, 2, 4, 7, 1, 8, 0, -8 };  **int** n = arr.length;  System.***out***.println("Given array is ");  rearrange.printArray(arr, n);  rearrange.rearrange(arr, n);  System.***out***.println("RearrangeD array is ");  rearrange.printArray(arr, n);  }  **void** rearrange(**int** arr[], **int** n) {  **int** outofplace = -1;  **for** (**int** index = 0; index < n; index++) {  **if** (outofplace >= 0) {  **if** (((arr[index] >= 0) && (arr[outofplace] < 0)) || ((arr[index] < 0) && (arr[outofplace] >= 0))) {  rightrotate(arr, n, outofplace, index);  **if** (index - outofplace > 2)  outofplace = outofplace + 2;  **else**  outofplace = -1;  }  }  **if** (outofplace == -1) {  **if** (((arr[index] >= 0) && ((index & 0x01) == 0)) || ((arr[index] < 0) && (index & 0x01) == 1))  outofplace = index;  }  }  }  **void** printArray(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  **void** rightrotate(**int** arr[], **int** n, **int** outofplace, **int** cur) {  **int** tmp = arr[cur];  **for** (**int** i = cur; i > outofplace; i--)  arr[i] = arr[i - 1];  arr[outofplace] = tmp;  }  } |

# Rearrange Array in Maximum Minimum Set1:- given a sorted array of positive integers, rearrange the array alternately i.e. first element should be maximum value,second minimum value, third second mas, fourth second min and so on.

|  |
| --- |
| Input : arr[] = {1, 2, 3, 4, 5, 6, 7} Output : arr[] = {7, 1, 6, 2, 5, 3, 4}  Input : arr[] = {1, 2, 3, 4, 5, 6} Output : arr[] = {6, 1, 5, 2, 4, 3}  Time Complexity : O(n) Auxiliary Space : O(n)  **public** **class** RearrangeArrayInMaximumMinimum\_Set1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  System.***out***.println("Original Array ");  System.***out***.println(Arrays.*toString*(arr));  *rearrange*(arr, arr.length);  }  **static** **void** rearrange(**int**[] arr, **int** n) {  **int** temp[] = **new** **int**[n];  **int** small = 0, large = n - 1;  **boolean** flag = **true**;  **for** (**int** i = 0; i < n; i++) {  **if** (flag)  temp[i] = arr[large--];  **else**  temp[i] = arr[small++];  flag = !flag;  }  System.***out***.println("Modified Array ");  System.***out***.println(Arrays.*toString*(temp));  }  } |

# Rearrange Array In Maximum Minimum Set2

|  |
| --- |
| // O(n) time and O(1) extra space  **import** java.util.Arrays;  **public** **class** RearrangeArrayInMaximumMinimum\_Set2 {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  System.***out***.println("Original Array ");  System.***out***.println(Arrays.*toString*(arr));  *rearrange*(arr, arr.length);  System.***out***.println("Modified Array ");  System.***out***.println(Arrays.*toString*(arr));  }  **public** **static** **void** rearrange(**int** arr[], **int** n) {  **int** max\_idx = n - 1, min\_idx = 0;  **int** max\_elem = arr[n - 1] + 1;  **for** (**int** i = 0; i < n; i++) {  **if** (i % 2 == 0) {  arr[i] += (arr[max\_idx] % max\_elem) \* max\_elem;  max\_idx--;  } **else** {  arr[i] += (arr[min\_idx] % max\_elem) \* max\_elem;  min\_idx++;  }  }  **for** (**int** i = 0; i < n; i++)  arr[i] = arr[i] / max\_elem;  }  } |

# Rearrange positive and negative numbers in O (n) time and O (1) extra space

|  |
| --- |
| **public** **class** Rearrangepositiveandnegative {  **public** **static** **void** main(String[] args) {  **int** arr[] = { -1, 2, -3, 4, 5, 6, -7, 8, 9 };  **int** n = arr.length;  *rearrange*(arr, n);  System.***out***.println("Array after rearranging: ");  *printArray*(arr, n);  }  **static** **void** rearrange(**int** arr[], **int** n) {  **int** i = -1, temp = 0;  **for** (**int** j = 0; j < n; j++) {  **if** (arr[j] < 0) {  i++;  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  **int** pos = i + 1, neg = 0;  **while** (pos < n && neg < pos && arr[neg] < 0) {  temp = arr[neg];  arr[neg] = arr[pos];  arr[pos] = temp;  pos++;  neg += 2;  }  }  **static** **void** printArray(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Remove Duplicates Using Hashing

|  |
| --- |
| //Time Complexity: O(n)  **public** **class** RemoveDuplicatesUsingHashing {  **public** **static** **void** main(String[] args) {  String str = "geeksforgeeks";  RemoveDuplicatesUsingHashing r = **new** RemoveDuplicatesUsingHashing();  r.removeDuplicates(str);  }  **void** removeDuplicates(String str) {  LinkedHashSet<Character> lhs = **new** LinkedHashSet<>();  **for** (**int** i = 0; i < str.length(); i++)  lhs.add(str.charAt(i));  **for** (Character ch : lhs)  System.***out***.print(ch);  }  } |

# Remove Duplicates Using Sorting

|  |
| --- |
| Input string: geeksforgeeks  1) Sort the characters eeeefggkkorss  2) Remove duplicates efgkorskkorss  3) Remove extra characters efgkors  //Time Complexity: O(nlogn)  **public** **class** RemoveDuplicatesUsingSorting {  **public** **static** **void** main(String[] args) {  String str = "geeksforgeeks";  System.***out***.println(*removeDups*(str));  }  **static** String removeDupsSorted(String str) {  **int** res\_ind = 1, ip\_ind = 1;  **char** arr[] = str.toCharArray();  **while** (ip\_ind != arr.length) {  **if** (arr[ip\_ind] != arr[ip\_ind - 1]) {  arr[res\_ind] = arr[ip\_ind];  res\_ind++;  }  ip\_ind++;  }  str = **new** String(arr);  **return** str.substring(0, res\_ind);  }  **static** String removeDups(String str) {  **char** temp[] = str.toCharArray();  Arrays.*sort*(temp);  str = **new** String(temp);  **return** *removeDupsSorted*(str);  }  } |

# Remove the forbidden strings

|  |
| --- |
| Input : n = 3 s1 = "etr" s2 = "ed" s3 = "ied" W = "PEtrUnited" letter = "d" Output : PDddUnitda  Input : n = 1 s1 = "PetrsDreamOh" W = "PetrsDreamOh" letter = h Output : HhhhhHhhhhHa  **public** **class** Removetheforbiddenstrings {  **public** **static** **int** *n*;  **public** **static** String *z*;  **public** **static** String *s*[] = **new** String[100];  **public** **static** **char** *w*[];  **public** **static** **char** *letter*;  **public** **static** **boolean** *pre*[] = **new** **boolean**[100];  **public** **static** **void** main(String[] args) {  *n* = 3;  *s*[0] = "etr";  *s*[1] = "ed";  *s*[2] = "ied";  *z* = "PEtrUnited";  *solve*();  }  **public** **static** **void** solve() {  *w* = *z*.toCharArray();  *letter* = 'd';  **int** l = *z*.length();  **int** p = *letter* - 'a';  **for** (**int** i = 0; i < 100; i++)  *pre*[i] = **false**;  **for** (**int** i = 0; i < l; i++) {  **for** (**int** j = 0; j < *n*; j++)  *verify*(i, j);  }  **for** (**int** i = 0; i < l; i++) {  **if** (*pre*[i] == **true**) {  **if** (*w*[i] == *letter*)  *w*[i] = (*letter* == 'a') ? 'b' : 'a';  **else** **if** (*w*[i] == (**char**) ((**int**) 'A' + p))  *w*[i] = (*letter* == 'a') ? 'B' : 'A';  **else** **if** (*w*[i] >= 'a' && *w*[i] <= 'z')  *w*[i] = *letter*;  **else** **if** (*w*[i] >= 'A' && *w*[i] <= 'Z')  *w*[i] = (**char**) ((**int**) 'A' + p);  }  }  System.***out***.println(*w*);  }  **public** **static** **void** verify(**int** position, **int** index) {  **int** l = *z*.length();  **int** k = *s*[index].length();  **if** (position + k > l)  **return**;  **boolean** same = **true**;  **for** (**int** i = position; i < position + k; i++) {  **int** n, n1;  **char** ch = *w*[i];  **char** ch1 = *s*[index].charAt(i - position);  **if** (ch >= 'a' && ch <= 'z')  n = ch - 'a';  **else**  n = ch - 'A';  **if** (ch1 >= 'a' && ch1 <= 'z')  n1 = ch1 - 'a';  **else**  n1 = ch1 - 'A';  **if** (n != n1)  same = **false**;  }  **if** (same == **true**) {  **for** (**int** i = position; i < position + k; i++)  *pre*[i] = **true**;  **return**;  }  }  } |

# Itrative Reverse Array

|  |
| --- |
| **public** **class** ReverseArrayItrative {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 5, 6 };  *printArray*(arr, 6);  *rvereseArray*(arr, 0, 5);  System.***out***.println("Reversed array is ");  *printArray*(arr, 6);  }  **static** **void** rvereseArray(**int** arr[], **int** start, **int** end) {  **int** temp;  **if** (start >= end)  **return**;  temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  *rvereseArray*(arr, start + 1, end - 1);  }  **static** **void** printArray(**int** arr[], **int** size) {  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Recursive Reverse Array

|  |
| --- |
| **public** **class** RecursiveReverseArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = {1, 2, 3, 4, 5, 6};  *printArray*(arr, 6);  *rvereseArray*(arr, 0, 5);  System.***out***.println("Reversed array is ");  *printArray*(arr, 6);  }  **static** **void** rvereseArray(**int** arr[], **int** start, **int** end){  **int** temp;  **if** (start >= end)  **return**;  temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  *rvereseArray*(arr, start+1, end-1);  }  **static** **void** printArray(**int** arr[], **int** size){  **for** (**int** i=0; i < size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Reversal algorithm for right rotation of an array

|  |
| --- |
| /\*Input: arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} k = 3 Output: 8 9 10 1 2 3 4 5 6 7  Input: arr[] = {121, 232, 33, 43 ,5} k = 2 Output: 43 5 121 232 33\*/  **public** **class** Reversal\_algorithm\_for\_right\_rotation\_of\_an\_array {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  **int** n = arr.length;  **int** k = 3;  *rightRotate*(arr, k, n);  *printArray*(arr, n);  }  **static** **void** rightRotate(**int** arr[], **int** d, **int** n) {  *reverseArray*(arr, 0, n - 1);  *reverseArray*(arr, 0, d - 1);  *reverseArray*(arr, d, n - 1);  }  **static** **void** reverseArray(**int** arr[], **int** start, **int** end) {  **while** (start < end) {  **int** temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  start++;  end--;  }  }  **static** **void** printArray(**int** arr[], **int** size) {  **for** (**int** i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Reverse Number

|  |
| --- |
| **public** **class** ReverseNumber {  **public** **static** **void** main(String[] args) {  **int** num = 12345;  System.***out***.println("original number " +num);  System.***out***.println("reverse number " +*reverseNumber*(num));  }  **public** **static** **int** reverseNumber(**int** number){  **int** reverse=0;  **int** remainder;  **while**(number>0){  remainder=number%10;  number=number/10;  reverse=reverse\*10+remainder;  }  **return** reverse;  }  } |

# Rotational Array Method1

|  |
| --- |
| **public** **class** RotationalArrayMethod\_1 {  **public** **static** **void** main(String[] args) {  RotationalArrayMethod\_1 rotate = **new** RotationalArrayMethod\_1();  **int** arr[] = {1, 2, 3, 4, 5, 6, 7};  rotate.leftRotate(arr, 2, 7);  rotate.printArray(arr, 7);  }  **void** leftRotate(**int** arr[], **int** d, **int** n){  **int** i;  **for** (i = 0; i < d; i++)  leftRotatebyOne(arr, n);  }  **void** leftRotatebyOne(**int** arr[], **int** n){  **int** i, temp;  temp = arr[0];  **for** (i = 0; i < n - 1; i++)  arr[i] = arr[i + 1];  arr[i] = temp;  }  **void** printArray(**int** arr[], **int** size){  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Rotational Array Method2

|  |
| --- |
| **public** **class** RotationalArrayMethod\_2 {  **public** **static** **void** main(String[] args) {  RotationalArrayMethod\_2 rotate = **new** RotationalArrayMethod\_2();  **int** arr[] = {1, 2, 3, 4, 5, 6, 7};  rotate.leftRotate(arr, 2, 7);  rotate.printArray(arr, 7);  }  **void** leftRotate(**int** arr[], **int** d, **int** n){  **int** i, j, k, temp;  **for** (i = 0; i < gcd(d, n); i++){  temp = arr[i];  j = i;  **while** (1 != 0){  k = j + d;  **if** (k >= n)  k = k - n;  **if** (k == i)  **break**;  arr[j] = arr[k];  j = k;  }  arr[j] = temp;  }  }  **int** gcd(**int** a, **int** b){  **if** (b == 0)  **return** a;  **else**  **return** gcd(b, a % b);  }  **void** printArray(**int** arr[], **int** size){  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Rotational Array Reversal Algo

|  |
| --- |
| **public** **class** RotationalArrayReversalAlgo {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 5, 6, 7 };  *leftRotate*(arr, 2); // Rotate array by 2  *printArray*(arr);  }  **static** **void** leftRotate(**int** arr[], **int** d) {  **int** n = arr.length;  *rvereseArray*(arr, 0, d - 1);  *rvereseArray*(arr, d, n - 1);  *rvereseArray*(arr, 0, n - 1);  }  **static** **void** rvereseArray(**int** arr[], **int** start, **int** end) {  **int** temp;  **while** (start < end) {  temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  start++;  end--;  }  }  **static** **void** printArray(**int** arr[]) {  **for** (**int** i = 0; i < arr.length; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Segregate 0s and 1s in an array using two indexes to traverse

|  |
| --- |
| **public** **class** Segregate\_0s\_and\_1s\_in\_an\_array\_Using\_two\_indexes\_to\_traverse {  **public** **static** **void** main(String[] args) {  Segregate\_0s\_and\_1s\_in\_an\_array\_Using\_two\_indexes\_to\_traverse seg = **new** Segregate\_0s\_and\_1s\_in\_an\_array\_Using\_two\_indexes\_to\_traverse();  **int** arr[] = **new** **int**[] { 0, 1, 0, 1, 1, 1 };  **int** i, arr\_size = arr.length;  seg.segregate0and1(arr, arr\_size);  System.***out***.print("Array after segregation is ");  **for** (i = 0; i < 6; i++)  System.***out***.print(arr[i] + " ");  }  **void** segregate0and1(**int** arr[], **int** size) {  **int** left = 0, right = size - 1;  **while** (left < right) {  **while** (arr[left] == 0 && left < right)  left++;  **while** (arr[right] == 1 && left < right)  right--;  **if** (left < right) {  arr[left] = 0;  arr[right] = 1;  left++;  right--;  }  }  }  } |

# Segregate 0s and 1s in an array using Count 0s or 1s

|  |
| --- |
| **public** **class** Segregate\_0s\_and\_1s\_in\_an\_array\_UsingCount0s\_or\_1s {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 0, 1, 0, 1, 1, 1 };  **int** n = arr.length;  *segregate0and1*(arr, n);  *print*(arr, n);  }  **static** **void** segregate0and1(**int** arr[], **int** n) {  **int** count = 0;  **for** (**int** i = 0; i < n; i++) {  **if** (arr[i] == 0)  count++;  }  **for** (**int** i = 0; i < count; i++)  arr[i] = 0;  **for** (**int** i = count; i < n; i++)  arr[i] = 1;  }  **static** **void** print(**int** arr[], **int** n) {  System.***out***.print("Array after segregation is ");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  } |

# Segregate Even and Odd Nodes in Linked List

|  |
| --- |
| **class** Segregate\_Even\_And\_Odd\_Nodes {  Node head;  **class** Node {  **int** data;  Node next;  Node(**int** d) {  data = d;  next = **null**;  }  }  **public** **void** segregateEvenOdd() {  Node evenStart = **null**;  Node evenEnd = **null**;  Node oddStart = **null**;  Node oddEnd = **null**;  Node currentNode = head;  **while** (currentNode != **null**) {  **int** element = currentNode.data;  **if** (element % 2 == 0) {  **if** (evenStart == **null**) {  evenStart = currentNode;  evenEnd = evenStart;  } **else** {  evenEnd.next = currentNode;  evenEnd = evenEnd.next;  }  } **else** {  **if** (oddStart == **null**) {  oddStart = currentNode;  oddEnd = oddStart;  } **else** {  oddEnd.next = currentNode;  oddEnd = oddEnd.next;  }  }  currentNode = currentNode.next;  }  **if** (oddStart == **null** || evenStart == **null**) {  **return**;  }  evenEnd.next = oddStart;  oddEnd.next = **null**;  head = evenStart;  }  **void** push(**int** new\_data) {  Node new\_node = **new** Node(new\_data);  new\_node.next = head;  head = new\_node;  }  **void** printList() {  Node temp = head;  **while** (temp != **null**) {  System.***out***.print(temp.data + " ");  temp = temp.next;  }  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  Segregate\_Even\_And\_Odd\_Nodes llist = **new** Segregate\_Even\_And\_Odd\_Nodes();  llist.push(11);  llist.push(10);  llist.push(9);  llist.push(6);  llist.push(4);  llist.push(1);  llist.push(0);  System.***out***.println("Original Linked List");  llist.printList();  llist.segregateEvenOdd();  System.***out***.println("Modified Linked List");  llist.printList();  }  } |

# Segregate Even and Odd numbers

|  |
| --- |
| /\*Input = {12, 34, 45, 9, 8, 90, 3} Output = {12, 34, 8, 90, 45, 9, 3} \*/  **public** **class** SegregateEvenandOddnumbers {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 34, 45, 9, 8, 90, 3 };  *segregateEvenOdd*(arr);  System.***out***.print("Array after segregation ");  **for** (**int** i = 0; i < arr.length; i++)  System.***out***.print(arr[i] + " ");  }  **static** **void** segregateEvenOdd(**int** arr[]) {  **int** left = 0, right = arr.length - 1;  **while** (left < right) {  **while** (arr[left] % 2 == 0 && left < right)  left++;  **while** (arr[right] % 2 == 1 && left < right)  right--;  **if** (left < right) {  **int** temp = arr[left];  arr[left] = arr[right];  arr[right] = temp;  left++;  right--;  }  }  }  } |

# Semaphore Demo

|  |
| --- |
| **class** Shared {  **static** **int** *count* = 0;  }  **class** MyThread **extends** Thread {  Semaphore sem;  String threadName;  **public** MyThread(Semaphore sem, String threadName) {  **super**(threadName);  **this**.sem = sem;  **this**.threadName = threadName;  }  @Override  **public** **void** run() {  **if** (**this**.getName().equals("A")) {  System.***out***.println("Starting " + threadName);  **try** {  System.***out***.println (threadName + “is waiting for a permit.");  sem.acquire();  System.***out***.println (threadName + “gets a permit.");  **for** (**int** i = 0; i < 5; i++) {  Shared.*count*++;  System.***out***.println(threadName + ": " + Shared.*count*);  Thread.*sleep*(10);  }  } **catch** (InterruptedException exc) {  System.***out***.println(exc);  }  System.***out***.println (threadName + “releases the permit.");  sem.release();  } **else** {  System.***out***.println("Starting " + threadName);  **try** {  System.***out***.println (threadName + “is waiting for a permit.");  sem.acquire();  System.***out***.println (threadName + “gets a permit.");  **for** (**int** i = 0; i < 5; i++) {  Shared.*count*--;  System.***out***.println(threadName + ": " + Shared.*count*);  Thread.*sleep*(10);  }  } **catch** (InterruptedException exc) {  System.***out***.println(exc);  }  System.***out***.println (threadName + “releases the permit.");  sem.release();  }  }  }  **public** **class** Semaphore\_Demo {  **public** **static** **void** main(String[] args) **throws** InterruptedException{  Semaphore sem = **new** Semaphore(1);  MyThread mt1 = **new** MyThread(sem, "A");  MyThread mt2 = **new** MyThread(sem, "B");  mt1.start();  mt2.start();  mt1.join();  mt2.join();  System.***out***.println("count: " + Shared.*count*);  }  } |

# Serialization and Deserialization SerialVersionUID Example

|  |
| --- |
| **public** **class** SerialVersionUID\_Example {  **private** **static** **final** **long** ***SerialVersionUID*** = 10l;  **int** i = 10;  **int** j = 20;  }  // Serialization  **class** Sender {  **public** **static** **void** main(String[] args) **throws** IOException {  SerialVersionUID\_Example g = **new** SerialVersionUID\_Example();  FileOutputStream fos = **new** FileOutputStream("C://Users//136342//workspace//Programming\_Projects//xyz.txt");  ObjectOutputStream oos = **new** ObjectOutputStream(fos);  oos.writeObject(g);  }  }  // Deserialization  **class** Receiver {  **public** **static** **void** main(String[] args) **throws** IOException, ClassNotFoundException {  FileInputStream fis = **new** FileInputStream("C://Users//136342//workspace//Programming\_Projects//xyz.txt");  ObjectInputStream ois = **new** ObjectInputStream(fis);  SerialVersionUID\_Example g1 = (SerialVersionUID\_Example) ois.readObject();  System.***out***.println("Deserialized Object Value:" + g1.i + "..." + g1.j);  }  } |

# Given a set (HashSet or TreeSet) of strings in Java, convert it into an array of strings.

|  |
| --- |
| Input : Set hash\_Set = new HashSet(); hash\_Set.add("Geeks"); hash\_Set.add("For"); Output : String arr[] = {"Geeks", "for"}  **public** **class** Set\_To\_Array\_Simple {  **public** **static** **void** main(String[] args) {  Set<String> s = **new** HashSet<String>();  s.add("Geeks");  s.add("for");  **int** n = s.size();  String arr[] = **new** String[n];  **int** i = 0;  **for** (String x : s)  arr[i++] = x;  **for** (String x : arr)  System.***out***.println(x);  }  } |

# Set to Array Stream

|  |
| --- |
| **public** **class** Set\_To\_Array\_Stream {  **public** **static** **void** main(String[] args) {  Set<String> s = **new** HashSet<String>();  s.add("Geeks");  s.add("for");  **int** n = s.size();  String[] arr = s.stream().toArray(String[]::**new**);  **for** (String x : arr)  System.***out***.println(x);  }  } |

# Set to Array toArray Function

|  |
| --- |
| **public** **class** Set\_To\_Array\_toArrayFun {  **public** **static** **void** main(String[] args) {  Set<String> s = **new** HashSet<String>();  s.add("Geeks");  s.add("for");  **int** n = s.size();  String arr[] = **new** String[n];  arr = s.toArray(arr);  **for** (String x : arr)  System.***out***.println(x);  }  } |

# Size of file on the Internet using Java: - to get the size of file from server first you need to connect to the server using URL and HttpURLConnection class. To get the size of file we use getContentLength() method.As the size of file can be too large we use BigInteger class.You cannot use integer datatype as it can generate an error in case the size of file is greater than 2GB.

|  |
| --- |
| **public** **class** Size\_Of\_File {  **public** **static** **void** main(String[] args) **throws** Exception {  BigInteger size = **new** BigInteger("1");  URL url = **new** URL("https://contribute.geeksforgeeks.org/wp-content/uploads/GATE.pdf");  HttpURLConnection conn;  **try** {  conn = (HttpURLConnection) url.openConnection();  conn.setRequestMethod("HEAD");  conn.getInputStream();  size = BigInteger.*valueOf*(conn.getContentLength());  System.***out***.println("The Size of file is:" + size + " bytes");  conn.getInputStream().close();  } **catch** (Exception e) {  System.***out***.println("Connection failed");  }  }  } |

# Smallest Missing Number

|  |
| --- |
| Input: {0, 1, 2, 6, 9}, n = 5, m = 10 Output: 3  Input: {0, 1, 2, 3, 4, 5, 6, 7, 10}, n = 9, m = 11 Output: 8  Use Binary Search Time Complexity: O(m log n)  Linear Search Time Complexity: O(n)  Modified Binary Search Time Complexity: O(Logn)  **public** **class** SmallestMissingNumber {  **public** **static** **void** main(String[] args) {  SmallestMissingNumber small = **new** SmallestMissingNumber();  **int** arr[] = { 0, 1, 2, 3, 4, 5, 6, 7, 10 };  **int** n = arr.length;  System.***out***.println("First Missing element is : " + small.findFirstMissing(arr, 0, n - 1));  }  **int** findFirstMissing(**int** array[], **int** start, **int** end) {  **if** (start > end)  **return** end + 1;  **if** (start != array[start])  **return** start;  **int** mid = (start + end) / 2;  **if** (array[mid] == mid)  **return** findFirstMissing(array, mid + 1, end);  **return** findFirstMissing(array, start, mid);  }  } |

# Find the smallest positive integer value that cannot be represented as sum of any subset of a given array

|  |
| --- |
| Input: arr[] = {1, 2, 3, 4, 5, 6}; Output: 22 Time Complexity is O(n).\*/  **public** **class** SmallestPositiveIntegerValue {  **public** **static** **void** main(String[] args) {  SmallestPositiveIntegerValue small = **new** SmallestPositiveIntegerValue();  **int** arr1[] = { 1, 3, 4, 5 };  **int** n1 = arr1.length;  System.***out***.println(small.findSmallest(arr1, n1));  }  **int** findSmallest(**int** arr[], **int** n) {  **int** res = 1;  **for** (**int** i = 0; i < n && arr[i] <= res; i++)  res = res + arr[i];  **return** res;  }  } |

# Smallest subarray with product divisible by k

|  |
| --- |
| Input: arr [] = {1, 9, 16, 5, 4, 3, 2} k = 720 Output: 3 The smallest subarray is {9, 16, 5} whose product is 720.  Input : arr[] = {1, 2, 4, 5, 6} K = 96 Output : No such subarray exists Time Complexity = O(n^2)  **public** **class** SmallestSubarrayWithProduct {  **public** **static** **void** main(String[] args) {  **int** array[] = { 1, 9, 16, 5, 4, 3, 2 };  **int** k = 720;  **int** answer = *findsubArray*(array, k);  **if** (answer != 0)  System.***out***.println(answer);  **else**  System.***out***.println ("No Such subarray exists.");  }  **public** **static** **int** findsubArray(**int** arr[], **int** k) {  **int** n = arr.length;  **int** res = n + 1;  **for** (**int** i = 0; i < n; i++) {  **int** curr\_prod = 1;  **for** (**int** j = i; j < n; j++) {  curr\_prod = curr\_prod \* arr[j];  **if** (curr\_prod % k == 0 && res > (j - i + 1)) {  res = Math.*min*(res, j - i + 1);  **break**;  }  }  }  **return** (res == n + 1) ? 0 : res;  }  } |

# Smallest Subarray with Sum Efficient

|  |
| --- |
| //Time complexity is O(n)  **public** **class** SmallestSubarrayWithSumEfficient {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 1, 4, 45, 6, 10, 19 };  **int** x = 51;  **int** n1 = arr1.length;  **int** res1 = *smallestSubWithSum*(arr1, n1, x);  **if** (res1 == n1 + 1)  System.***out***.println("Not Possible");  **else**  System.***out***.println(res1);  }  **static** **int** smallestSubWithSum(**int** arr[], **int** n, **int** x) {  **int** curr\_sum = 0, min\_len = n + 1;  **int** start = 0, end = 0;  **while** (end < n) {  **while** (curr\_sum <= x && end < n)  curr\_sum += arr[end++];  **while** (curr\_sum > x && start < n) {  **if** (end - start < min\_len)  min\_len = end - start;  curr\_sum -= arr[start++];  }  }  **return** min\_len;  }  } |

# Smallest subarray with sum greater than a given value

|  |
| --- |
| arr[] = {1, 4, 45, 6, 0, 19} x = 51 Output: 3 Minimum length subarray is {4, 45, 6}  arr[] = {1, 2, 4} x = 8 Output : Not Possible Whole array sum is smaller than 8. Time complexity is O(n2).  **public** **class** SmallestSubarrayWithSumSimple {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 1, 4, 45, 6, 10, 19 };  **int** x = 51;  **int** n1 = arr1.length;  **int** res1 = *smallestSubWithSum*(arr1, n1, x);  **if** (res1 == n1 + 1)  System.***out***.println("Not Possible");  **else**  System.***out***.println(res1);  }  **static** **int** smallestSubWithSum(**int** arr[], **int** n, **int** x) {  **int** min\_len = n + 1;  **for** (**int** start = 0; start < n; start++) {  **int** curr\_sum = arr[start];  **if** (curr\_sum > x)  **return** 1;  **for** (**int** end = start + 1; end < n; end++) {  curr\_sum += arr[end];  **if** (curr\_sum > x && (end - start + 1) < min\_len)  min\_len = (end - start + 1);  }  }  **return** min\_len;  }  } |

# Find the smallest window in a string containing all characters of another string

|  |
| --- |
| Input : string = "this is a test string" pattern = "tist" Output: Minimum window is "t stri" Explanation: "t stri" contains all the characters of pattern.  Input : string = "geeksforgeeks" pattern = "ork" Output : Minimum window is "ksfor"  **public** **class** SmallestWindowInString {  **static** **final** **int** ***no\_of\_chars*** = 256;  **public** **static** **void** main(String[] args) {  String str = "this is a test string";  String pat = "tist";  System.***out***.print("Smallest window is : n" + *findSubString*(str, pat));  }  **static** String findSubString(String str, String pat) {  **int** len1 = str.length();  **int** len2 = pat.length();  **if** (len1 < len2) {  System.***out***.println("No such window exists");  **return** "";  }  **int** hash\_pat[] = **new** **int**[***no\_of\_chars***];  **int** hash\_str[] = **new** **int**[***no\_of\_chars***];  **for** (**int** i = 0; i < len2; i++)  hash\_pat[pat.charAt(i)]++;  **int** start = 0, start\_index = -1, min\_len = Integer.***MAX\_VALUE***;  **int** count = 0;  **for** (**int** j = 0; j < len1; j++) {  hash\_str[str.charAt(j)]++;  **if** (hash\_pat[str.charAt(j)] != 0 && hash\_str[str.charAt(j)] <= hash\_pat[str.charAt(j)])  count++;  **if** (count == len2) {  **while** (hash\_str[str.charAt(start)] > hash\_pat[str.charAt(start)] || hash\_pat[str.charAt(start)] == 0) {  **if** (hash\_str[str.charAt(start)] > hash\_pat[str.charAt(start)])  hash\_str[str.charAt(start)]--;  start++;  }  **int** len\_window = j - start + 1;  **if** (min\_len > len\_window) {  min\_len = len\_window;  start\_index = start;  }  }  }  **if** (start\_index == -1) {  System.***out***.println("No such window exists");  **return** "";  }  **return** str.substring(start\_index, start\_index + min\_len);  }  } |

# Given an array A [] consisting 0s, 1s and 2s, write a function that sorts A []. : - the function should put all 0s first then all 1s and all 2s in last.

|  |
| --- |
| Input : {0, 1, 2, 0, 1, 2} Output : {0, 0, 1, 1, 2, 2}  Input : {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1} Output : {0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2} Time Complexity: O(n)  **public** **class** SortAnArrayOf0Aand1sAnd2s {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };  **int** arr\_size = arr.length;  *sort012*(arr, arr\_size);  System.***out***.println("Array after seggregation ");  *printArray*(arr, arr\_size);  }  **static** **void** sort012(**int** a[], **int** arr\_size) {  **int** lo = 0;  **int** hi = arr\_size - 1;  **int** mid = 0, temp = 0;  **while** (mid <= hi) {  **switch** (a[mid]) {  **case** 0: {  temp = a[lo];  a[lo] = a[mid];  a[mid] = temp;  lo++;  mid++;  **break**;  }  **case** 1:  mid++;  **break**;  **case** 2: {  temp = a[mid];  a[mid] = a[hi];  a[hi] = temp;  hi--;  **break**;  }  }  }  }  **static** **void** printArray(**int** arr[], **int** arr\_size) {  **int** i;  **for** (i = 0; i < arr\_size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  } |

# Sort an Array of 0s and 1s and 2s Simple Counting

|  |
| --- |
| //Time complexity will be O(n).  **public** **class** SortAnArrayOf0sAnd1sAnd2s\_SimpleCounting {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };  **int** n = 12;  *sort012*(arr, n);  }  **public** **static** **void** sort012(**int** arr[], **int** n) {  **int** count0 = 0, count1 = 0;  **int** count2 = 0;  **for** (**int** i = 0; i < n; i++) {  **if** (arr[i] == 0)  count0++;  **if** (arr[i] == 1)  count1++;  **if** (arr[i] == 2)  count2++;  }  **for** (**int** i = 0; i < count0; i++)  arr[i] = 0;  **for** (**int** i = count0; i < (count0 + count1); i++)  arr[i] = 1;  **for** (**int** i = (count0 + count1); i < n; i++)  arr[i] = 2;  *printArray*(arr, n);  }  **public** **static** **void** printArray(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  } |

# Sorted and Rotated Array

|  |
| --- |
| **public** **class** Sorted\_And\_Rotated\_array {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 5, 6, 7, 8, 9, 10, 1, 2, 3 };  **int** n = arr1.length;  **int** key = 3;  System.***out***.println("Index of the element is: " + *pivotedBinarySearch*(arr1, n, key));  }  **static** **int** pivotedBinarySearch(**int** arr[], **int** n, **int** key) {  **int** pivot = *findPivot*(arr, 0, n - 1);  **if** (pivot == -1)  **return** *binarySearch*(arr, 0, n - 1, key);  **if** (arr[pivot] == key)  **return** pivot;  **if** (arr[0] <= key)  **return** *binarySearch*(arr, 0, pivot - 1, key);  **return** *binarySearch*(arr, pivot + 1, n - 1, key);  }  **static** **int** findPivot(**int** arr[], **int** low, **int** high) {  **if** (high < low)  **return** -1;  **if** (high == low)  **return** low;  **int** mid = (low + high) / 2;  **if** (mid < high && arr[mid] > arr[mid + 1])  **return** mid;  **if** (mid > low && arr[mid] < arr[mid - 1])  **return** (mid - 1);  **if** (arr[low] >= arr[mid])  **return** *findPivot*(arr, low, mid - 1);  **return** *findPivot*(arr, mid + 1, high);  }  **static** **int** binarySearch(**int** arr[], **int** low, **int** high, **int** key) {  **if** (high < low)  **return** -1;  **int** mid = (low + high) / 2;  **if** (key == arr[mid])  **return** mid;  **if** (key > arr[mid])  **return** *binarySearch*(arr, (mid + 1), high, key);  **return** *binarySearch*(arr, low, (mid - 1), key);  }  } |

# Sorted Array Delete

|  |
| --- |
| //Delete: O(n)  **public** **class** Sorted\_Array\_Delete {  **public** **static** **void** main(String[] args) {  **int** i;  **int** arr[] = { 10, 20, 30, 40, 50 };  **int** n = arr.length;  **int** key = 30;  System.***out***.print("Array before deletion:\n");  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  n = *deleteElement*(arr, n, key);  System.***out***.print("\n\nArray after deletion:\n");  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **int** deleteElement(**int** arr[], **int** n, **int** key) {  **int** pos = *binarySearch*(arr, 0, n - 1, key);  **if** (pos == -1) {  System.***out***.println("Element not found");  **return** n;  }  **int** i;  **for** (i = pos; i < n - 1; i++)  arr[i] = arr[i + 1];  **return** n - 1;  }  **static** **int** binarySearch(**int** arr[], **int** low, **int** high, **int** key) {  **if** (high < low)  **return** -1;  **int** mid = (low + high) / 2;  **if** (key == arr[mid])  **return** mid;  **if** (key > arr[mid])  **return** *binarySearch*(arr, (mid + 1), high, key);  **return** *binarySearch*(arr, low, (mid - 1), key);  }  } |

# Sorted Array Insert

|  |
| --- |
| //Insert: O(n)  **public** **class** Sorted\_Array\_Insert {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[20];  arr[0] = 12;  arr[1] = 16;  arr[2] = 20;  arr[3] = 40;  arr[4] = 50;  arr[5] = 70;  **int** capacity = arr.length;  **int** n = 6;  **int** key = 26;  System.***out***.print("\nBefore Insertion: ");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  n = *insertSorted*(arr, n, key, capacity);  System.***out***.print("\nAfter Insertion: ");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **int** insertSorted(**int** arr[], **int** n, **int** key, **int** capacity) {  **if** (n >= capacity)  **return** n;  **int** i;  **for** (i = n - 1; (i >= 0 && arr[i] > key); i--)  arr[i + 1] = arr[i];  arr[i + 1] = key;  **return** (n + 1);  }  } |

# Sorted Array Search

|  |
| --- |
| //Search: O(Log n)  **public** **class** Sorted\_Array\_Search {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 6, 7, 8, 9, 10 };  **int** n, key;  n = arr.length;  key = 10;  System.***out***.println("Index: " + *binarySearch*(arr, 0, n, key));  }  **static** **int** binarySearch(**int** arr[], **int** low, **int** high, **int** key) {  **if** (high < low)  **return** -1;  **int** mid = (low + high) / 2;  **if** (key == arr[mid])  **return** mid;  **if** (key > arr[mid])  **return** *binarySearch*(arr, (mid + 1), high, key);  **return** *binarySearch*(arr, low, (mid - 1), key);  }  } |

# Find a sorted subsequence of size 3 in linear time

|  |
| --- |
| Input: arr[] = {12, 11, 10, 5, 6, 2, 30} Output: 5, 6, 30  Input: arr[] = {1, 2, 3, 4} Output: 1, 2, 3 OR 1, 2, 4 OR 2, 3, 4  Input: arr[] = {4, 3, 2, 1} Output: No such triplet  Time Complexity: O(n) Auxliary Space: O(n)  **public** **class** SortedSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 11, 10, 5, 6, 2, 30 };  *find3Numbers*(arr);  }  **static** **void** find3Numbers(**int** arr[]) {  **int** n = arr.length;  **int** max = n - 1;  **int** min = 0;  **int** i;  **int**[] smaller = **new** **int**[n];  smaller[0] = -1;  **for** (i = 1; i < n; i++) {  **if** (arr[i] <= arr[min]) {  min = i;  smaller[i] = -1;  } **else**  smaller[i] = min;  }  **int**[] greater = **new** **int**[n];  greater[n - 1] = -1;  **for** (i = n - 2; i >= 0; i--) {  **if** (arr[i] >= arr[max]) {  max = i;  greater[i] = -1;  } **else**  greater[i] = max;  }  **for** (i = 0; i < n; i++) {  **if** (smaller[i] != -1 && greater[i] != -1) {  System.***out***.print(arr[smaller[i]] + " " + arr[i] + " " + arr[greater[i]]);  **return**;  }  }  System.***out***.println("No such triplet found");  **return**;  }  } |

# Sorting a 2D Array according to values in any given column in Java

|  |
| --- |
| Input : If our 2D array is given as (Order 4X4)  39 27 11 42  10 93 91 90  54 78 56 89  24 64 20 65  Sorting it by values in column 3  Output : 39 27 11 42  24 64 20 65  54 78 56 89  10 93 91 90  **public** **class** Sorting\_2DArray\_Values {  **public** **static** **void** main(String[] args) {  **int** matrix[][] = { { 39, 27, 11, 42 }, { 10, 93, 91, 90 }, { 54, 78, 56, 89 }, { 24, 64, 20, 65 } };  **int** col = 3;  *sortbyColumn*(matrix, col - 1);  **for** (**int** i = 0; i < matrix.length; i++) {  **for** (**int** j = 0; j < matrix[i].length; j++)  System.***out***.print(matrix[i][j] + " ");  System.***out***.println();  }  }  **public** **static** **void** sortbyColumn(**int** arr[][], **int** col) {  Arrays.*sort*(arr, **new** Comparator<**int**[]>() {  @Override  **public** **int** compare(**final** **int**[] entry1, **final** **int**[] entry2) {  **if** (entry1[col] > entry2[col])  **return** 1;  **else**  **return** -1;  }  });  }  } |

# Sorting HashMap keys ArrayList

|  |
| --- |
| **public** **class** Sorting\_HashMap\_keys\_ArrayList {  **static** Map<String, Integer> *map* = **new** HashMap<>();  **public** **static** **void** main(String[] args) {  *map*.put("Jayant", 80);  *map*.put("Abhishek", 90);  *map*.put("Anushka", 80);  *map*.put("Amit", 75);  *map*.put("Danish", 40);  *sortbykey*();  }  **public** **static** **void** sortbykey() {  ArrayList<String> sortedKeys = **new** ArrayList<String>(*map*.keySet());  Collections.*sort*(sortedKeys);  **for** (String x : sortedKeys)  System.***out***.println("Key = " + x + ", Value = " + *map*.get(x));  }  } |

# Sorting HashMap keys constructor

|  |
| --- |
| **public** **class** Sorting\_HashMap\_keys\_constructor {  **static** Map<String, Integer> *map* = **new** HashMap<>();  **public** **static** **void** main(String[] args) {  *map*.put("Jayant", 80);  *map*.put("Abhishek", 90);  *map*.put("Anushka", 80);  *map*.put("Amit", 75);  *map*.put("Danish", 40);  *sortbykey*();  }  **public** **static** **void** sortbykey() {  TreeMap<String, Integer> sorted = **new** TreeMap<>(*map*);  **for** (Map.Entry<String, Integer> entry : sorted.entrySet())  System.***out***.println("Key = " + entry.getKey() + ", Value = " + entry.getValue());  }  } |

# Sorting HashMap keys putAll

|  |
| --- |
| Input : Key = Jayant, Value = 80  Key = Anushka, Value = 80  Key = Amit, Value = 75  Key = Abhishek, Value = 90  Key = Danish, Value = 40  Output : Sorted Map according to Names:  Key = Abhishek, Value = 90  Key = Amit, Value = 75  Key = Anushka, Value = 80  Key = Danish, Value = 40  Key = Jayant, Value = 80  **public** **class** Sorting\_HashMap\_keys\_putAll {  **static** Map<String, Integer> *map* = **new** HashMap<>();  **public** **static** **void** main(String[] args) {  *map*.put("Jayant", 80);  *map*.put("Abhishek", 90);  *map*.put("Anushka", 80);  *map*.put("Amit", 75);  *map*.put("Danish", 40);  *sortbykey*();  }  **public** **static** **void** sortbykey() {  TreeMap<String, Integer> sorted = **new** TreeMap<>();  sorted.putAll(*map*);  **for** (Map.Entry<String, Integer> entry : sorted.entrySet())  System.***out***.println("Key = " + entry.getKey() + ", Value = " + entry.getValue());  }  } |

# Splitting Numeric String: - given a numeric string (length <= 32), split it into two or more integers (if possible), such that: - 1. Difference between current and previous number is 1. 2. No number contains leading zeroes. If it is possible to separate a given numeric string then print “Possible” followed by the first number of the increasing sequence, else print “Not Possible”.

|  |
| --- |
| Input : 1234 Output : Possible 1 Explanation: String can be split as "1", "2", "3", "4"  Input : 99100 Output :Possible 99 Explanation: String can be split as "99","100"  Input : 101103 Output : Not Possible Explanation: It is not possible to split this string under given constraint.  **public** **class** Splitting\_Numeric\_String {  **public** **static** **void** main(String[] args) {  Scanner in = **new** Scanner(System.***in***);  String str = "99100";  *split*(str);  }  **public** **static** **void** split(String str) {  **int** len = str.length();  **if** (len == 1) {  System.***out***.println("Not Possible");  **return**;  }  String s1 = "", s2 = "";  **long** num1, num2;  **for** (**int** i = 0; i <= len / 2; i++) {  **int** flag = 0;  s1 = str.substring(0, i + 1);  num1 = Long.*parseLong*((s1));  num2 = num1 + 1;  s2 = Long.*toString*(num2);  **int** k = i + 1;  **while** (flag == 0) {  **int** l = s2.length();  **if** (k + l > len) {  flag = 1;  **break**;  }  **if** ((str.substring(k, k + l).equals(s2))) {  flag = 0;  num2++;  k = k + l;  **if** (k == len)  **break**;  s2 = Long.*toString*(num2);  l = s2.length();  **if** (k + 1 > len) {  flag = 1;  **break**;  }  }  **else**  flag = 1;  }  **if** (flag == 0) {  System.***out***.println("Possible" + " " + s1);  **break**;  }  **else** **if** (flag == 1 && i > len / 2 - 1) {  System.***out***.println("Not Possible");  **break**;  }  }  }  } |

# Standard Normal Distribution

|  |
| --- |
| The standard normal distribution is a special case of the normal distribution. It occurs when a normal random variable has a mean of 0 and a standard deviation of 1. The normal random variable of a standard normal distribution is called a standard score or a z score. Z = (X - u) / s  where:  Z = value on the standard normal distribution  X = value on the original distribution  u = mean of the original distribution  s = standard deviation of the original distribution  The nextGaussian () method is used to get the next random, Normally distributed double value with mean 0.0 and standard deviation 1.0.  **public** **class** Standard\_Normal\_Distribution {  **public** **static** **void** main(String[] args) {  **double** Z, X, s, u;  X = 26;  u = 50;  s = 10;  Z = (X - u) / s;  System.***out***.println("the Z-value obtained is: " + Z);  Random ran = **new** Random();  **double** nxt = ran.nextGaussian();  System.***out***.println("The next Gaussian value generated is : " + nxt);  }  } |

# Star Pattern

|  |
| --- |
| Input :5  Output :  \*\*\*\*\*\*\*\*1\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*2\*2\*\*\*\*\*\*\*  \*\*\*\*\*\*3\*3\*3\*\*\*\*\*\*  \*\*\*\*\*4\*4\*4\*4\*\*\*\*\*  \*\*\*\*5\*5\*5\*5\*5\*\*\*\*  **public** **class** StarPattern {  **public** **static** **void** main(String[] args) {  **int** height = 7;  *StarPattern1*(height);  }  **public** **static** **void** StarPattern1(**int** height) {  **for** (**int** i = 0; i < height; i++) {  **for** (**int** j = height - 1; j > i; j--) {  System.***out***.print("\*");  }  **boolean** printChar = **false**;  **for** (**int** j = 0; j < ((i \* 2) + 1); j++) {  **if** (printChar) {  System.***out***.print("\*");  } **else** {  System.***out***.print(i + 1);  }  printChar = !printChar;  }  **for** (**int** j = height - 1; j > i; j--) {  System.***out***.print("\*");  }  System.***out***.println();  }  }  } |

# Static blank final variable

|  |
| --- |
| **Blank final variable:** A final variable declared but not assigned is known as a blank final variable. It can be initialized within a constructor only. It raises compilation error if it is not initialized because it should be given a value somewhere in the program and that too from a constructor only.  **Static blank final variable:** It is a blank final variable declared as static. That is, a final static variable declared but not given a value or not initialized is known as static blank final variable.It can be initialized through a static block only.  **public** **class** Static\_blank\_final\_variable {  **private** **static** **final** **int** ***a***;  **private** **final** **int** b;  **static** {  ***a*** = 1;  }  Static\_blank\_final\_variable(**int** c) {  b = c;  }  **public** **static** **void** main(String[] args) {  Static\_blank\_final\_variable g1 = **new** Static\_blank\_final\_variable(10);  Static\_blank\_final\_variable g2 = **new** Static\_blank\_final\_variable(20);  System.***out***.println(Static\_blank\_final\_variable.***a***); // 1  System.***out***.println(g1.b); // 10  System.***out***.println(g1.b); // 10  }  } |

# String Interning Method

|  |
| --- |
| String Interning is a method of storing only one copy of each distinct String Value, which must be immutable.By applying **String.intern ()** on a couple of strings will ensure that all strings having same contents share same memory.  **intern () method :** In Java, when we perform any operation using intern() method, it returns a canonical representation forthe string object. A pool is managed by String class.  **When the intern() method** is executed then it checks whether the String equals to this String Object is in the pool or not.If it is available, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.It follows that for any two strings s and t, s.intern() == t.intern() is true if and only if s.equals(t) is true.In the java programming language, intern() is a method present in String class which is present in java.lang package. By the help of intern () method, we can get the reference of corresponding String constant pool object of an original object.  **public** **class** String\_Interning\_Method {  **public** **static** **void** main(String[] args) {  String s1 = **new** String("GFG"); // Line-1  String s2 = s1.intern(); // Line-2  System.***out***.println(s1 == s2); // false  String s3 = "GFG"; // Line-3  System.***out***.println(s2 == s3); // true  }  } |

# String Literal Vs String Object

|  |
| --- |
| **String Literal: -** **String str = “GeeksForGeeks**”; This method references internal pool of string objects. If there already exists a string value “GeeksForGeeks”, then str will reference of that string and no new String object will be created.  **String Object :-** **String str = new String(“GeeksForGeeks”);** In this method JVM is forced to create a new string reference, even if “GeeksForGeeks” is in the reference pool. if we compare performance of string literal and string object, string object will always take more time to execute than string literal because it will construct a new string every time it is executed  **public** **class** String\_Literal\_Vs\_String\_Object {  **public** **static** **void** main(String[] args) {  **long** start1 = System.*currentTimeMillis*();  **for** (**int** i = 0; i < 10000; i++) {  String s1 = "GeeksForGeeks";  String s2 = "Welcome";  }  **long** end1 = System.*currentTimeMillis*();  **long** total\_time = end1 - start1;  System.***out***.println("Time taken to execute" + " string literal = " + total\_time);  **long** start2 = System.*currentTimeMillis*();  **for** (**int** i = 0; i < 10000; i++) {  String s3 = **new** String("GeeksForGeeks");  String s4 = **new** String("Welcome");  }  **long** end2 = System.*currentTimeMillis*();  **long** total\_time1 = end2 - start2;  System.***out***.println("Time taken to execute" + " string object=" + total\_time1);  }  } |

# Subarray With Given Sum Efficient

|  |
| --- |
| //time complexity is O(n)  **public** **class** SubarrayWithGivenSum\_Efficient {  **public** **static** **void** main(String[] args) {  SubarrayWithGivenSum\_Efficient arraysum = **new** SubarrayWithGivenSum\_Efficient();  **int** arr[] = { 15, 2, 4, 8, 9, 5, 10, 23 };  **int** n = arr.length;  **int** sum = 23;  arraysum.subArraySum(arr, n, sum);  }  **int** subArraySum(**int** arr[], **int** n, **int** sum) {  **int** curr\_sum = arr[0], start = 0, i;  **for** (i = 1; i <= n; i++) {  **while** (curr\_sum > sum && start < i - 1) {  curr\_sum = curr\_sum - arr[start];  start++;  }  **if** (curr\_sum == sum) {  **int** p = i - 1;  System.***out***.println("Sum found between indexes " + start + " and " + p);  **return** 1;  }  **if** (i < n)  curr\_sum = curr\_sum + arr[i];  }  System.***out***.println("No subarray found");  **return** 0;  }  } |

# Subarray With Given Sum Simple

|  |
| --- |
| Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33 Ouptut: Sum found between indexes 2 and 4  Input: arr[] = {1, 4}, sum = 0 Output: No subarray found  Time Complexity: O (n^2) in worst case.  **public** **class** SubarrayWithGivenSum\_Simple {  **public** **static** **void** main(String[] args) {  SubarrayWithGivenSum\_Simple arraysum = **new** SubarrayWithGivenSum\_Simple();  **int** arr[] = { 15, 2, 4, 8, 9, 5, 10, 23 };  **int** n = arr.length;  **int** sum = 23;  arraysum.subArraySum(arr, n, sum);  }  **int** subArraySum(**int** arr[], **int** n, **int** sum) {  **int** curr\_sum, i, j;  **for** (i = 0; i < n; i++) {  curr\_sum = arr[i];  **for** (j = i + 1; j <= n; j++) {  **if** (curr\_sum == sum) {  **int** p = j - 1;  System.***out***.println("Sum found between indexes " + i + " and " + p);  **return** 1;  }  **if** (curr\_sum > sum || j == n)  **break**;  curr\_sum = curr\_sum + arr[j];  }  }  System.***out***.println("No subarray found");  **return** 0;  }  } |

# Find the sum up to n terms of the sequence: 5 + 55 + 555 + … up to n.

|  |
| --- |
| Input : 2 Output: 60  Input : 3 Output: 595  **public** **class** Sum\_of\_the\_series {  **public** **static** **void** main(String[] args) {  **int** n = 2;  System.***out***.println(*sumOfSeries*(n));  }  **static** **int** sumOfSeries(**int** n) {  **return** (**int**) (0.6172 \* (Math.*pow*(10, n) - 1) - 0.55 \* n);  }  } |

# Sum of Bitwise and of all pairs

|  |
| --- |
| Input: arr[] = {5, 10, 15} Output: 15  Required Value = (5 & 10) + (5 & 15) + (10 & 15) = 0 + 5 + 10 = 15  Time complexity is O (n2).  **public** **class** SumofBitwiseAndofallpairs {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 9, 7, 6 };  **int** n = arr.length;  System.***out***.println(*pairAndSum*(arr, arr.length));  }  **static** **int** pairAndSum(**int** arr[], **int** n) {  **int** ans = 0;  **for** (**int** i = 0; i < n; i++)  **for** (**int** j = i + 1; j < n; j++)  ans += arr[i] & arr[j];  **return** ans;  }  } |

# Sum of f (a[i], a[j]) over all pairs in an array of n integers

|  |
| --- |
| Input : 6 6 4 4 Output : -8 Explanation: All pairs are: (6 - 6) + (6 - 6) +(6 - 6) + (4 - 6) + (4 - 6) + (4 - 6) + (4 - 6) + (4 - 4) + (4 - 4) = -8  Input: 1 2 3 1 3 Output: 4 Explanation: the pairs that add up are: (3, 1), (3, 1) to give 4, rest all pairs according to condition gives 0.  Time Complexity : O(n)  **public** **class** SumofOverAllPairs {  **public** **static** **void** main(String[] args) {  **int** a[] = { 1, 2, 3, 1, 3 };  **int** n = a.length;  System.***out***.println(*sum*(a, n));  }  **public** **static** **int** sum(**int** a[], **int** n) {  Map<Integer, Integer> cnt = **new** HashMap<Integer, Integer>();  **int** ans = 0, pre\_sum = 0;  **for** (**int** i = 0; i < n; i++) {  ans += (i \* a[i]) - pre\_sum;  pre\_sum += a[i];  **if** (cnt.containsKey(a[i] - 1))  ans -= cnt.get(a[i] - 1);  **if** (cnt.containsKey(a[i] + 1))  ans += cnt.get(a[i] + 1);  **if** (cnt.containsKey(a[i])) {  cnt.put(a[i], cnt.get(a[i]) + 1);  } **else** {  cnt.put(a[i], 1);  }  }  **return** ans;  }  } |

# Sum of the products of all possible Subsets

|  |
| --- |
| Input : arr[] = {1, 2, 3} Output : 23 Possible Subset are: 1, 2, 3, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}  Products of elements in above subsets : 1, 2, 3, 2, 3, 6, 6  Sum of all products = 1 + 2 + 3 + 2 + 3 + 6 + 6 = 23  Time Complexity: O(n) Auxiliary Space: O(1)\*/  **public** **class** Sumoftheproducts {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4 };  **int** n = arr.length;  System.***out***.println(*productOfSubsetSums*(arr, n));  }  **static** **int** productOfSubsetSums(**int** arr[], **int** n) {  **int** ans = 1;  **for** (**int** i = 0; i < n; ++i)  ans = ans \* (arr[i] + 1);  **return** ans - 1;  }  } |

# Sum of XOR of all pairs in an array

|  |
| --- |
| Input : arr[] = {5, 9, 7, 6} Output : 47  5 ^ 9 = 12 9 ^ 7 = 14 7 ^ 6 = 1 5 ^ 7 = 2 5 ^ 6 = 3 9 ^ 6 = 15 Sum = 12 + 14 + 1 + 2 + 3 + 15 = 47  Time complexity is O (n2).  **public** **class** SumofXORofallpairsinanarray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 5, 9, 7, 6 };  **int** n = arr.length;  System.***out***.println(*pairORSum*(arr, arr.length));  }  **static** **int** pairORSum(**int** arr[], **int** n) {  **int** ans = 0;  **for** (**int** i = 0; i < n; i++)  **for** (**int** j = i + 1; j < n; j++)  ans += arr[i] ^ arr[j];  **return** ans;  }  } |

# Super Example

|  |
| --- |
| super () is use to call Base class’s(Parent class’s) constructor.super() should be first statement inside any constructor. It can be used only inside constructor and nowhere else.super () is used to refer only parent class’s (super class’s) constructor.  **public** **class** SuperExample **extends** Parent {  SuperExample() {  **super**();  System.***out***.println("Flow comes back from " + "Parent class no arg const");  }  **public** **static** **void** main(String[] args) {  **new** SuperExample();  System.***out***.println("Inside Main");  }  }  **class** Parent {  Parent() {  System.***out***.println("Parent class's No " + " arg constructor");  }  } |

# Write a Java program to take an input string and exchange the first and last word and revers the middle word.

|  |
| --- |
| Input : Hello World GFG Welcomes You Output :You semocleW GFG dlroW Hello  **public** **class** Swap\_corner\_words\_and\_reverse\_middle\_characters {  **public** **static** **void** main(String[] args) {  String s = "Hello World GFG Welcomes You";  *print*(s);  }  **static** **void** print(String s) {  String fst = "";  **int** i = 0;  **for** (i = 0; i < s.length();) {  **while** (s.charAt(i) != ' ') {  fst = fst + s.charAt(i);  i++;  }  **break**;  }  String last = "";  **int** j = 0;  **for** (j = s.length() - 1; j >= i;) {  **while** (s.charAt(j) != ' ') {  last = s.charAt(j) + last;  j--;  }  **break**;  }  System.***out***.print(last);  **for** (**int** m = j; m >= i; m--) {  System.***out***.print(s.charAt(m));  }  System.***out***.println(fst);  }  } |

# Symmetric difference of two sorted array

|  |
| --- |
| Input: arr1 [] = {1, 3, 5, 8, 15, 27, 35}. arr2 [] = {5, 7, 8, 11, 15, 18, 35}. Output : 1 3 7 11 18 27  arr1 [] - arr2 [] = {1, 3, 27}.  arr [2] - arr1[] = {7, 11, 18}.  SymmDiff = (arr1 [] - arr2 []) UNION (arr2 [] - arr1 []). = {1, 3, 7, 11, 18, 27}.  **public** **class** Symmetricdifferenceoftwosortedarray {  **public** **static** **void** main(String[] args) {  **int**[] arr1 = { 2, 4, 5, 7, 8, 10, 12, 15 };  **int**[] arr2 = { 5, 8, 11, 12, 14, 15 };  **int** n = arr1.length;  **int** m = arr2.length;  *symmDiff*(arr1, arr2, n, m);  }  **static** **void** symmDiff(**int**[] arr1, **int**[] arr2, **int** n, **int** m) {  **int** i = 0, j = 0;  **while** (i < n && j < m) {  **if** (arr1[i] < arr2[j]) {  System.***out***.print(arr1[i] + " ");  i++;  } **else** **if** (arr2[j] < arr1[i]) {  System.***out***.print(arr2[j] + " ");  j++;  }  **else** {  i++;  j++;  }  }  }  } |

# Given an array of pairs, find all symmetric pairs in it: - Two pairs (a, b) and (c, d) are said to be symmetric if c is equal to b and a, is equal to c and d. For e.g. (10, 20) and (20, 10) are symmetric

|  |
| --- |
| Input: arr[] = {{11, 20}, {30, 40}, {5, 10}, {40, 30}, {10, 5}}  Output: Following pairs have symmetric pairs (30, 40) (5, 10)  Time Complexity is O(n)  **public** **class** SymmetricPairs {  **public** **static** **void** main(String[] args) {  **int** arr[][] = **new** **int**[5][2];  arr[0][0] = 11;  arr[0][1] = 20;  arr[1][0] = 30;  arr[1][1] = 40;  arr[2][0] = 5;  arr[2][1] = 10;  arr[3][0] = 40;  arr[3][1] = 30;  arr[4][0] = 10;  arr[4][1] = 5;  *findSymPairs*(arr);  }  **static** **void** findSymPairs(**int** arr[][]) {  HashMap<Integer, Integer> hM = **new** HashMap<Integer, Integer>();  **for** (**int** i = 0; i < arr.length; i++) {  **int** first = arr[i][0];  **int** sec = arr[i][1];  Integer val = hM.get(sec);  **if** (val != **null** && val == first)  System.***out***.println("(" + sec + ", " + first + ")");  **else**  hM.put(first, sec);  }  }  } |

# Synchronous Callback: - the code execution will block or wait for the event before continuing. Until your event returns a response your program will not execute any further.so basically the callback performs all its work before returning to the call statement. Then problem with synchronous callback is that they appear to lag. Any process having multiple tasks where the tasks must be executed in sequence and doesn’t occupy much time should usr Synchronous Callbacks. For e.g. yor are in a movie queue for ticket you can’t get one until everyone in frony of yoe gets one.

|  |
| --- |
| **interface** OnGeekEventListener {  **void** onGeekEvent();  }  **class** A **implements** OnGeekEventListener {  @Override  **public** **void** onGeekEvent() {  System.***out***.println("Performing callback after synchronous Task");  }  }  **public** **class** Synchronous\_Callback {  **private** OnGeekEventListener mListener;  **public** **static** **void** main(String[] args) {  Synchronous\_Callback obj = **new** Synchronous\_Callback();  OnGeekEventListener mListener = **new** A();  obj.registerOnGeekEventListener(mListener);  obj.doGeekStuff();  }  **public** **void** registerOnGeekEventListener(OnGeekEventListener mListener) {  **this**.mListener = mListener;  }  **public** **void** doGeekStuff() {  System.***out***.println("Performing callback before synchronous Task");  **if** (**this**.mListener != **null**) {  mListener.onGeekEvent();  }  }  } |

# Java Program to take Screenshots

|  |
| --- |
| **Java.awt.Robot** class to capture pixels of screen. It provides method like createScreenCapture which captures the current screen. This method returns captured image as BufferedImage object which can be saved as a file. It also uses ImageIO to save it as PNG image format. Toolkit.getDefaultToolkit ().getSize () method is used to get the size of screen.The **serialVersionUID** is universal version identifier for Serializable class. Thread is used so that after executing the program we can switch to the screen we want to take screenshot of. 120s is the time in seconds i.e. 2 mins.  **public** **class** Take\_Screenshots {  **public** **static** **final** **long** ***serialVersionUID*** = 1L;  **public** **static** **void** main(String[] args) {  **try** {  Thread.*sleep*(120);  Robot r = **new** Robot();  String path = "C://Users//136342//workspace//Programming\_Projects// Shot.jpg";  Rectangle capture = **new** Rectangle(Toolkit.*getDefaultToolkit*().getScreenSize());  BufferedImage Image = r.createScreenCapture(capture);  ImageIO.*write*(Image, "jpg", **new** File(path));  System.***out***.println("Screenshot saved");  } **catch** (AWTException | IOException | InterruptedException ex) {  System.***out***.println(ex);  }  }  } |

# This Example

|  |
| --- |
| this () is used to call current class’s constructor.this() should be first statement inside any constructor. It can be used only inside constructor and nowhere else. this () is use to refer only current class’s constructor.  **public** **class** ThisExample {  ThisExample() {  **this**(10);  System.***out***.println("Flow comes back from " + "RR class's 1 arg const");  }  ThisExample(**int** a) {  System.***out***.println("RR class's 1 arg const");  }  **public** **static** **void** main(String[] args) {  **new** ThisExample();  System.***out***.println("Inside Main");  }  } |

# Starvation: - In Starvation, threads are also waiting for each other. But here waiting time is not infinite after some interval of time, waiting thread always gets the resources whatever is required to execute thread run() method.

|  |
| --- |
| **class** Worker{  **public** **synchronized** **void** work(){  String name = Thread.*currentThread*().getName();  String filename = name +".txt";    **try**(BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(filename))){  bw.write("thread "+ name+ "wrote this msg");  }**catch**(IOException e){  e.printStackTrace();  }  **while**(**true**){  System.***out***.println(name +" is working");  }  }  }  **public** **class** StarvationDemo {  **public** **static** **void** main(String[] args) {  Worker w = **new** Worker();  **for**(**int** i=0 ; i<10; i++){  **new** Thread(**new** Runnable(){  **public** **void** run(){  w.work();  }  }).start();  }  }  } |

# String Palindrome Method1

|  |
| --- |
| **public** **class** StringPalindrom\_Method\_1 {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.print("Enter a string: ");  String str = r.next();  String revStr = "";  **for** (**int** i = str.length()-1 ; i>=0 ; i--){  revStr = revStr + str.charAt(i);  }  **if**(revStr.equalsIgnoreCase(str)){  System.***out***.println("Palindrome");  }**else**{  System.***out***.println("Not Palindrome");  }  }  } |

# String Palindrome Method2

|  |
| --- |
| **public** **class** StringPalindrome\_Method\_2 {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.print("Enter a string: ");  String str = r.next();  **boolean** out = *isPalindrom*(str);  System.***out***.println(out);  }  **public** **static** **boolean** isPalindrom(String str){  **if**(str == **null**)  **return** **false**;  StringBuilder sb = **new** StringBuilder(str);  sb.reverse();  **return** sb.toString().equals(str);  }  } |

# String Palindrome Method3

|  |
| --- |
| **public** **class** StringPalindrome\_Method\_3 {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.print("Enter a string: ");  String str = r.next();  **boolean** out = *isPalindrom*(str);  System.***out***.println(out);  }  **public** **static** **boolean** isPalindrom(String str){  **if**(str == **null**)  **return** **false**;  **int** length = str.length();  **for**(**int** i =0 ; i<length/2 ; i++){  **if**(str.charAt(i) != str.charAt(length-i-1))  **return** **false**;  }  **return** **true**;  }  } |

# String Palindrome Recursion Method

|  |
| --- |
| **public** **class** StringPalindrome\_Method\_Recursion {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.print("Enter a string: ");  String str = r.next();  **boolean** out = *isPalindrom*(str);  System.***out***.println(out);  }  **public** **static** **boolean** isPalindrom(String str){  **if**(str.length() == 0 || str.length() == 1)  **return** **true**;  **if**(str.charAt(0) == str.charAt(str.length()-1)){  **return** *isPalindrom*(str.substring(1, str.length()-1));  }  **return** **false**;  }  } |

# String Palindrome Rotational Method

|  |
| --- |
| **public** **class** StringPalindrome\_Method\_Rotational {  **public** **static** **void** main(String[] args) {  Scanner r = **new** Scanner(System.***in***);  System.***out***.print("Enter a string: ");  String str = r.next();  **boolean** out = *isRotationalPalindrom*(str);  System.***out***.println(out);  }  **public** **static** **boolean** isRotationalPalindrom(String str) {  **char** a[] = str.toCharArray();  **for** (**int** i = 0; i < a.length; i++) {  *rotateString*(a);  **if** (*isPalindrom*(a)) {  System.***out***.println(str + " " + String.*valueOf*(a));  **return** **true**;  }  }  **return** **false**;  }  **public** **static** **void** rotateString(**char**[] arr) {  **char** temp = arr[0];  **int** i = 0;  **for** (i = 0; i < arr.length - 1; i++) {  arr[i] = arr[i + 1];  }  arr[i] = temp;  }  **public** **static** **boolean** isPalindrom(**char**[] arr) {  **for** (**int** i = 0, j = arr.length - 1; i < arr.length / 2; i++, j--) {  **if** (arr[i] != arr[j]) {  **return** **false**;  }  }  **return** **true**;  }  } |

# Differences between extending Thread class and implementing Runnable interface

|  |
| --- |
| 1. When we extend Thread class, we can’t extend any other class even we require and When we implement Runnable, we can save a space for our class to extend any other class in future or now.  2. When we extend Thread class, each of our thread creates unique object and associate with it. When we implements Runnable, it shares the same object to multiple threads.  **public** **class** Thread\_Extending **extends** Thread {  **public** **static** **void** main(String[] args) {  Thread\_Extending t = **new** Thread\_Extending();  t.start();  System.***out***.println("Main method executed by main thread");  }  **public** **void** run() {  System.***out***.println("Run method executed by child Thread");  }  }  output  Main method executed by main thread  Run method executed by child Thread |

# Thread Runnable

|  |
| --- |
| **public** **class** Thread\_Runnable **extends** T **implements** Runnable {  **public** **static** **void** main(String[] args) {  Thread\_Runnable t = **new** Thread\_Runnable();  t.*m1*();  Thread t1 = **new** Thread(t);  t1.start();  System.***out***.println("Main method executed by main thread");  }  **public** **void** run() {  System.***out***.println("Run method executed by child Thread");  }  }  **class** T {  **public** **static** **void** m1() {  System.***out***.println("Hello Visitors");  }  }  output  Hello Visitors  Main method executed by main thread  Run method executed by child Thread |

# Thread interrupt method1

|  |
| --- |
| interrupt () method : If any thread is in sleeping or waiting state then using interrupt() method, we can interrupt the execution of that thread by showing InterruptedException. A thread which is in the sleeping or waiting state can be interrupted with the help of interrupt () method of Thread class.  Example : Suppose there are two threads and If one of the threads is blocked in an invocation of the wait(), wait(long), or wait(long, int) methods of the Object class, or of the join(), join(long), join(long, int), sleep(long), or sleep(long, int), methods of this class, then its interrupt status will be cleared and it will receive an InterruptedException, which gives the chance to another thread to execute the corresponding run() method of another thread which results into high performance and reduces the waiting time of the threads.  scenarios where we can interrupt a thread:-  **1. Interrupting a thread that doesn’t stop working:** In the program, we handle the InterruptedException using try and catch block, so whenever any thread interrupt currently executing thread it will comes out from the sleeping state but it will not stop working.  **public** **class** Thread\_interrupt\_method1 {  **public** **static** **void** main(String[] args) **throws** InterruptedException {  MyClass thread = **new** MyClass();  thread.start();  thread.interrupt();  System.***out***.println("Main thread execution completes");  }  }  **class** MyClass **extends** Thread {  **public** **void** run() {  **try** {  **for** (**int** i = 0; i < 5; i++) {  System.***out***.println("Child Thread executing");  Thread.*sleep*(1000);  }  } **catch** (InterruptedException e) {  System.***out***.println("InterruptedException occur");  }  }  }  //output  //Main thread execution completes  //Child Thread executing  //InterruptedException occur |

# Thread interrupt method2

|  |
| --- |
| **2. Interrupting a thread that stops working:** In the program, after interrupting currently executing thread, we are throwing a new exception in the catch block so it will stop working.  **public** **class** Thread\_interrupt\_method2 {  **public** **static** **void** main(String[] args) {  Geeks t1 = **new** Geeks();  t1.start();  **try** {  t1.interrupt();  } **catch** (Exception e) {  System.***out***.println("Exception handled");  }  }  }  **class** Geeks **extends** Thread {  **public** **void** run() {  **try** {  Thread.*sleep*(2000);  System.***out***.println("Geeksforgeeks");  } **catch** (InterruptedException e) {  **throw** **new** RuntimeException("Thread " + "interrupted");  }  }  }  //output  //Exception in thread "Thread-0" java.lang.RuntimeException: Thread interrupted |

# Thread interrupt method3

|  |
| --- |
| **3. Interrupting a thread that works normally**: In the program, there is no exception occurred during the execution of thread.Here, interrupt only sets the interrupted flag to true, which can be used by java programmer later.  **public** **class** Thread\_interrupt\_method3 {  **public** **static** **void** main(String[] args) {  Geeks1 t1 = **new** Geeks1();  t1.start();  t1.interrupt();  }  }  **class** Geeks1 **extends** Thread {  **public** **void** run() {  **for** (**int** i = 0; i < 5; i++)  System.***out***.println(i); //0 1 2 3 4  }  } |

# Thread isAlive Method: - how can one thread know when another thread has ended?

|  |
| --- |
| Java multi-threading provides two ways to find that :- isAlive() and join() methods  **isAlive () :** It tests if this thread is alive. A thread is alive if it has been started and has not yet died. There is a transitional period from when a thread is running to when a thread is not running. After the run () method returns, there is a short period of time before the thread stops. If we want to know if the start method of the thread has been called or if thread has been terminated, we must use isAlive () method. This method is used to find out if a thread has actually been started and has yet not terminated.  **Syntax** :- final boolean isAlive( )  **Return Value:** returns true if the thread upon which it is called is still running. It returns false otherwise.\*/  **public** **class** Thread\_isAlive\_Method **extends** Thread{  **public** **static** **void** main(String[] args) {  Thread\_isAlive\_Method c1 = **new** Thread\_isAlive\_Method();  Thread\_isAlive\_Method c2 = **new** Thread\_isAlive\_Method();  c1.start();  c2.start();  System.***out***.println(c1.isAlive());  System.***out***.println(c2.isAlive());  }  **public** **void** run() {  System.***out***.println("geeks ");  **try** {  Thread.*sleep*(300);  } **catch** (InterruptedException ie) {  }  System.***out***.println("forgeeks ");  }  } |

# Thread Join Method

|  |
| --- |
| **join() :** When the join() method is called, the current thread will simply wait until the thread it is joining with is no longer alive.Or we can say the method that you will more commonly use to wait for a thread to finish is called join( ).  This method waits until the thread on which it is called terminates. Its name comes from the concept of the calling thread waiting until the specified thread joins it. Additional forms of join ( ) allow you to specify a maximum amount of time thatyou want to wait for the specified thread to terminate.  **Syntax :** final void join( ) throws InterruptedException  **public** **class** Thread\_Join\_Method **extends** Thread {  **public** **static** **void** main(String[] args) {  Thread\_Join\_Method c1 = **new** Thread\_Join\_Method();  Thread\_Join\_Method c2 = **new** Thread\_Join\_Method();  c1.start();  **try** {  c1.join();  } **catch** (InterruptedException ie) {  }  c2.start();  }  **public** **void** run() {  System.***out***.println("geeks ");  **try** {  Thread.*sleep*(300);  } **catch** (InterruptedException ie) {  }  System.***out***.println("forgeeks ");  }  } |

# Thread without Join Method

|  |
| --- |
| **public** **class** Thread\_WithoutJoin\_Method **extends** Thread {  **public** **static** **void** main(String[] args) {  Thread\_WithoutJoin\_Method c1 = **new** Thread\_WithoutJoin\_Method();  Thread\_WithoutJoin\_Method c2 = **new** Thread\_WithoutJoin\_Method();  c1.start();  c2.start();  System.***out***.println(c1.isAlive());  System.***out***.println(c2.isAlive());  }  **public** **void** run() {  System.***out***.println("geeks ");  **try** {  Thread.*sleep*(300);  } **catch** (InterruptedException ie) {  }  System.***out***.println("forgeeks ");  }  } |

# Three way partitioning of an array around a given range: - given an array and a range [lowval, highval], partition the array around the range such that array is divided in three parts: - 1. All elements smaller than lowval come first. 2. All elements in range lowval to highval come next. 3. All elements of three sets can appear in any order. The individual elements of three sets can appear in any end.

|  |
| --- |
| Input: arr[] = {1, 14, 5, 20, 4, 2, 54, 20, 87, 98, 3, 1, 32} lowVal = 14, highVal = 20  Output: arr[] = {1, 5, 4, 2, 1, 3, 14, 20, 20, 98, 87, 32, 54} Time Complexity : O(n) Auxiliary Space : O(1)  **public** **class** ThreeWayPartitioning {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 14, 5, 20, 4, 2, 54, 20, 87, 98, 3, 1, 32 };  *threeWayPartition*(arr, 10, 20);  System.***out***.println("Modified array ");  **for** (**int** i = 0; i < arr.length; i++) {  System.***out***.print(arr[i] + " ");  }  }  **public** **static** **void** threeWayPartition(**int**[] arr, **int** lowVal, **int** highVal) {  **int** n = arr.length;  **int** start = 0, end = n - 1;  **for** (**int** i = 0; i < end;) {  **if** (arr[i] < lowVal) {  **int** temp = arr[start];  arr[start] = arr[i];  arr[i] = temp;  start++;  i++;  }  **else** **if** (arr[i] > highVal) {  **int** temp = arr[end];  arr[end] = arr[i];  arr[i] = temp;  end--;  }  **else**  i++;  }  }  } |

# Thread Pool

|  |
| --- |
| **class** Task **implements** Runnable {  **private** String name;  **public** Task(String s) {  name = s;  }  **public** **void** run() {  **try** {  **for** (**int** i = 0; i <= 5; i++) {  **if** (i == 0) {  Date d = **new** Date();  SimpleDateFormat ft = **new** SimpleDateFormat("hh:mm:ss");  System.***out***.println("Initialization Time for" + " task name - " + name + " = " + ft.format(d));  } **else** {  Date d = **new** Date();  SimpleDateFormat ft = **new** SimpleDateFormat("hh:mm:ss");  System.***out***.println("Executing Time for task name - " + name + " = " + ft.format(d));  }  Thread.*sleep*(1000);  }  System.***out***.println(name + " complete");  }  **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  **public** **class** Thread\_Pool {  **static** **final** **int** ***MAX\_T*** = 3;  **public** **static** **void** main(String[] args) {  Runnable r1 = **new** Task("task 1");  Runnable r2 = **new** Task("task 2");  Runnable r3 = **new** Task("task 3");  Runnable r4 = **new** Task("task 4");  Runnable r5 = **new** Task("task 5");  ExecutorService pool = Executors.*newFixedThreadPool*(***MAX\_T***);  pool.execute(r1);  pool.execute(r2);  pool.execute(r3);  pool.execute(r4);  pool.execute(r5);  pool.shutdown();  }  } |

# Two Elements Whose Sum Is Closest To Zero Method1

|  |
| --- |
| **public** **class** TwoElementsWhoseSumIsClosestToZero\_Method1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 60, -10, 70, -80, 85 };  *minAbsSumPair*(arr, 6);  }  **static** **void** minAbsSumPair(**int** arr[], **int** arr\_size) {  **int** inv\_count = 0;  **int** l, r, min\_sum, sum, min\_l, min\_r;  **if** (arr\_size < 2) {  System.***out***.println("Invalid Input");  **return**;  }  min\_l = 0;  min\_r = 1;  min\_sum = arr[0] + arr[1];  **for** (l = 0; l < arr\_size - 1; l++) {  **for** (r = l + 1; r < arr\_size; r++) {  sum = arr[l] + arr[r];  **if** (Math.*abs*(min\_sum) > Math.*abs*(sum)) {  min\_sum = sum;  min\_l = l;  min\_r = r;  }  }  }  System.***out***.println(" The two elements whose " + "sum is minimum are " + arr[min\_l] + " and " + arr[min\_r]);  }  } |

# Two elements whose sum is closest to zero using sorting

|  |
| --- |
| **public** **class** TwoElementsWhoseSumIsClosestToZero\_UseSorting {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 60, -10, 70, -80, 85 };  **int** n = arr.length;  *minAbsSumPair*(arr, n);  }  **static** **void** minAbsSumPair(**int** arr[], **int** n) {  **int** sum, min\_sum = 999999;  **int** l = 0, r = n - 1;  **int** min\_l = l, min\_r = n - 1;  **if** (n < 2) {  System.***out***.println("Invalid Input");  **return**;  }  *sort*(arr, l, r);  **while** (l < r) {  sum = arr[l] + arr[r];  **if** (Math.*abs*(sum) < Math.*abs*(min\_sum)) {  min\_sum = sum;  min\_l = l;  min\_r = r;  }  **if** (sum < 0)  l++;  **else**  r--;  }  System.***out***.println(" The two elements whose " + "sum is minimum are " + arr[min\_l] + " and " + arr[min\_r]);  }  **static** **void** sort(**int** arr[], **int** low, **int** high) {  **if** (low < high) {  **int** pi = *partition*(arr, low, high);  *sort*(arr, low, pi - 1);  *sort*(arr, pi + 1, high);  }  }  **static** **int** partition(**int** arr[], **int** low, **int** high) {  **int** pivot = arr[high];  **int** i = (low - 1); // index of smaller element  **for** (**int** j = low; j < high; j++) {  **if** (arr[j] <= pivot) {  i++;  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  **int** temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  **return** i + 1;  }  } |

# Two Given Sets Are Disjoint Simple

|  |
| --- |
| Input: set1[] = {12, 34, 11, 9, 3} set2[] = {2, 1, 3, 5} Output: Not Disjoint 3 is common in two sets.  Input: set1[] = {12, 34, 11, 9, 3} set2[] = {7, 2, 1, 5} Output: Yes, Disjoint  Time complexity of this method is O(mn).  **public** **class** TwoGivenSetsAreDisjointSimple {  **public** **static** **void** main(String[] args) {  TwoGivenSetsAreDisjointSimple dis = **new** TwoGivenSetsAreDisjointSimple();  **int** set1[] = { 12, 34, 11, 9, 3 };  **int** set2[] = { 7, 2, 1, 5 };  **boolean** result = dis.aredisjoint(set1, set2);  **if** (result)  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **boolean** aredisjoint(**int** set1[], **int** set2[]) {  **for** (**int** i = 0; i < set1.length; i++) {  **for** (**int** j = 0; j < set2.length; j++) {  **if** (set1[i] == set2[j])  **return** **false**;  }  }  **return** **true**;  }  } |

# Two Given Sets Are Disjoint Use Hashing

|  |
| --- |
| //Time complexity is O(m+n)  **public** **class** TwoGivenSetsAreDisjointUseHashing {  **public** **static** **void** main(String[] args) {  **int** set1[] = { 10, 5, 3, 4, 6 };  **int** set2[] = { 8, 7, 9, 3 };  **if** (*areDisjoint*(set1, set2))  System.***out***.println("Yes");  **else**  System.***out***.println("No");  }  **static** **boolean** areDisjoint(**int** set1[], **int** set2[]) {  HashSet<Integer> set = **new** HashSet<>();  **for** (**int** i = 0; i < set1.length; i++)  set.add(set1[i]);  **for** (**int** i = 0; i < set2.length; i++)  **if** (set.contains(set2[i]))  **return** **false**;  **return** **true**;  }  } |

# Two Given Sets Are Disjoint Use Sorting And Merging

|  |
| --- |
| //Time complexity is O (mLogm + nLogn).  **public** **class** TwoGivenSetsAreDisjointUseSortingAndMerging {  **public** **static** **void** main(String[] args) {  TwoGivenSetsAreDisjointUseSortingAndMerging dis = **new** TwoGivenSetsAreDisjointUseSortingAndMerging();  **int** set1[] = { 12, 34, 11, 9, 3 };  **int** set2[] = { 7, 2, 1, 5 };  **boolean** result = dis.aredisjoint(set1, set2);  **if** (result)  System.***out***.println("YES");  **else**  System.***out***.println("NO");  }  **boolean** aredisjoint(**int** set1[], **int** set2[]) {  **int** i = 0, j = 0;  Arrays.*sort*(set1);  Arrays.*sort*(set2);  **while** (i < set1.length && j < set2.length) {  **if** (set1[i] < set2[j])  i++;  **else** **if** (set1[i] > set2[j])  j++;  **else**  **return** **false**;  }  **return** **true**;  }  } |

# Two repeating elements in given array using array elements as index

|  |
| --- |
| **public** **class** TwoRepeatingElementsInGivenArray\_UsingArrayElementsAsIndex {  **public** **static** **void** main(String[] args) {  TwoRepeatingElementsInGivenArray\_UsingArrayElementsAsIndex repeat = **new** TwoRepeatingElementsInGivenArray\_UsingArrayElementsAsIndex();  **int** arr[] = { 4, 2, 4, 5, 2, 3, 1 };  **int** arr\_size = arr.length;  repeat.printRepeating(arr, arr\_size);  }  **void** printRepeating(**int** arr[], **int** size) {  **int** i;  System.***out***.println("The repeating elements are : ");  **for** (i = 0; i < size; i++) {  **if** (arr[Math.*abs*(arr[i])] > 0)  arr[Math.*abs*(arr[i])] = -arr[Math.*abs*(arr[i])];  **else**  System.***out***.print(Math.*abs*(arr[i]) + " ");  }  }  } |

# Two repeating elements in given array using count array

|  |
| --- |
| **public** **class** TwoRepeatingElementsInGivenArray\_UseCountArray {  **public** **static** **void** main(String[] args) {  TwoRepeatingElementsInGivenArray\_UseCountArray repeat = **new** TwoRepeatingElementsInGivenArray\_UseCountArray();  **int** arr[] = { 4, 2, 4, 5, 2, 3, 1 };  **int** arr\_size = arr.length;  repeat.printRepeating(arr, arr\_size);  }  **void** printRepeating(**int** arr[], **int** size) {  **int** count[] = **new** **int**[size];  **int** i;  System.***out***.println("Repeated elements are : ");  **for** (i = 0; i < size; i++) {  **if** (count[arr[i]] == 1)  System.***out***.print(arr[i] + " ");  **else**  count[arr[i]]++;  }  }  } |

# Two repeating elements in given array using make two equations

|  |
| --- |
| **public** **class** TwoRepeatingElementsInGivenArray\_MakeTwoEquations {  **public** **static** **void** main(String[] args) {  TwoRepeatingElementsInGivenArray\_MakeTwoEquations repeat = **new** TwoRepeatingElementsInGivenArray\_MakeTwoEquations();  **int** arr[] = { 4, 2, 4, 5, 2, 3, 1 };  **int** arr\_size = arr.length;  repeat.printRepeating(arr, arr\_size);  }  **void** printRepeating(**int** arr[], **int** size) {  **int** S = 0;  **int** P = 1;  **int** x, y;  **int** D;  **int** n = size - 2, i;  **for** (i = 0; i < size; i++) {  S = S + arr[i];  P = P \* arr[i];  }  S = S - n \* (n + 1) / 2;  P = P / fact(n);  D = (**int**) Math.*sqrt*(S \* S - 4 \* P);  x = (D + S) / 2;  y = (S - D) / 2;  System.***out***.println("The two repeating elements are :");  System.***out***.print(x + " " + y);  }  **int** fact(**int** n) {  **return** (n == 0) ? 1 : n \* fact(n - 1);  }  } |

# Two Repeating Elements In Given Array Method1

|  |
| --- |
| **public** **class** TwoRepeatingElementsInGivenArray\_Method1 {  **public** **static** **void** main(String[] args) {  TwoRepeatingElementsInGivenArray\_Method1 repeat = **new** TwoRepeatingElementsInGivenArray\_Method1();  **int** arr[] = { 4, 2, 4, 5, 2, 3, 1 };  **int** arr\_size = arr.length;  repeat.printRepeating(arr, arr\_size);  }  **void** printRepeating(**int** arr[], **int** size) {  **int** i, j;  System.***out***.println("Repeated Elements are :");  **for** (i = 0; i < size; i++) {  **for** (j = i + 1; j < size; j++) {  **if** (arr[i] == arr[j])  System.***out***.print(arr[i] + " ");  }  }  }  } |

# Two repeating elements in given array using XOR

|  |
| --- |
| **public** **class** TwoRepeatingElementsInGivenArray\_UsingXOR {  **public** **static** **void** main(String[] args) {  TwoRepeatingElementsInGivenArray\_UsingXOR repeat = **new** TwoRepeatingElementsInGivenArray\_UsingXOR();  **int** arr[] = { 4, 2, 4, 5, 2, 3, 1 };  **int** arr\_size = arr.length;  repeat.printRepeating(arr, arr\_size);  }  **void** printRepeating(**int** arr[], **int** size) {  **int** xor = arr[0];  **int** set\_bit\_no;  **int** i;  **int** n = size - 2;  **int** x = 0, y = 0;  **for** (i = 1; i < size; i++)  xor ^= arr[i];  **for** (i = 1; i <= n; i++)  set\_bit\_no = (xor & ~(xor - 1));  **for** (i = 0; i < size; i++) {  **int** a = arr[i] & set\_bit\_no;  **if** (a != 0)  x = x ^ arr[i]; /\* XOR of first set in arr[] \*/  **else**  y = y ^ arr[i]; /\* XOR of second set in arr[] \*/  }  **for** (i = 1; i <= n; i++) {  **int** a = i & set\_bit\_no;  **if** (a != 0)  x = x ^ i; /\* XOR of first set in arr[] and {1, 2, ...n } \*/  **else**  y = y ^ i; /\* XOR of second set in arr[] and {1, 2, ...n } \*/  }  System.***out***.println("The two repeated elements are :");  System.***out***.println(x + " " + y);  }  } |

# Two Way Sorting Using Negative Multiplication

|  |
| --- |
| Time complexity: O(n log n) space complexity: O(1)  **public** **class** TwoWaySorting\_UsingNegativeMultiplication {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 3, 2, 7, 5, 4 };  *twoWaySort*(arr, arr.length);  System.***out***.println(Arrays.*toString*(arr));  }  **static** **void** twoWaySort(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++)  **if** ((arr[i] & 1) != 0) // Check for odd  arr[i] \*= -1;  Arrays.*sort*(arr);  **for** (**int** i = 0; i < n; i++)  **if** ((arr[i] & 1) != 0)  arr[i] \*= -1;  }  } |

# Sort all even numbers in ascending order and then sort all odd numbers in descending order

|  |
| --- |
| Input : arr[] = {1, 2, 3, 5, 4, 7, 10} Output : arr[] = {7, 5, 3, 1, 2, 4, 10} Time complexity: O(n log n) space complexity: O(1)  **public** **class** TwoWaySorting\_UsingPartition {  **public** **static** **void** main(String[] args) {  Integer arr[] = { 1, 3, 2, 7, 5, 4 };  *twoWaySort*(arr, arr.length);  System.***out***.println(Arrays.*toString*(arr));  }  **static** **void** twoWaySort(Integer arr[], **int** n) {  **int** l = 0, r = n - 1;  **int** k = 0;  **while** (l < r) {  **while** (arr[l] % 2 != 0) {  l++;  k++;  }  **while** (arr[r] % 2 == 0 && l < r)  r--;  **if** (l < r) {  **int** temp = arr[l];  arr[l] = arr[r];  arr[r] = temp;  }  }  Arrays.*sort*(arr, 0, k, Collections.*reverseOrder*());  Arrays.*sort*(arr, k, n);  }  } |

# Unchecked Exceptions

|  |
| --- |
| **public** **class** Unchecked\_Exceptions {  **public** **static** **void** main(String[] args) {  Unchecked\_Exceptions obj = **new** Unchecked\_Exceptions();  obj.p();  System.***out***.println ("Normal flow...");  }  **void** p() {  **try** {  n();  } **catch** (Exception e) {  System.***out***.println("Exception handled");  }  }  **void** n() {  m();  }  **void** m() {  **int** data = 50 / 0;  }  } |

# Union of two sorted arrays

|  |
| --- |
| Input: arr1[] = {1, 3, 4, 5, 7} arr2[] = {2, 3, 5, 6} Union : {1, 2, 3, 4, 5, 6, 7}  **public** **class** Unionoftwosortedarrays {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 1, 2, 4, 5, 6 };  **int** arr2[] = { 2, 3, 5, 7 };  **int** m = arr1.length;  **int** n = arr2.length;  *printUnion*(arr1, arr2, m, n);  }  **static** **int** printUnion(**int** arr1[], **int** arr2[], **int** m, **int** n) {  **int** i = 0, j = 0;  **while** (i < m && j < n) {  **if** (arr1[i] < arr2[j])  System.***out***.print(arr1[i++] + " ");  **else** **if** (arr2[j] < arr1[i])  System.***out***.print(arr2[j++] + " ");  **else** {  System.***out***.print(arr2[j++] + " ");  i++;  }  }  **while** (i < m)  System.***out***.print(arr1[i++] + " ");  **while** (j < n)  System.***out***.print(arr2[j++] + " ");  **return** 0;  }  } |

# Union of two unsorted arrays using sorting and searching

|  |
| --- |
| **public** **class** Unionoftwounsortedarrays {  **public** **static** **void** main(String[] args) {  Unionoftwounsortedarrays u\_i = **new** Unionoftwounsortedarrays();  **int** arr1[] = { 7, 1, 5, 2, 3, 6 };  **int** arr2[] = { 3, 8, 6, 20, 7 };  **int** m = arr1.length;  **int** n = arr2.length;  System.***out***.println("Union of two arrays is ");  u\_i.printUnion(arr1, arr2, m, n);  }  **void** printUnion(**int** arr1[], **int** arr2[], **int** m, **int** n) {  **if** (m > n) {  **int** tempp[] = arr1;  arr1 = arr2;  arr2 = tempp;  **int** temp = m;  m = n;  n = temp;  }  Arrays.*sort*(arr1);  **for** (**int** i = 0; i < m; i++)  System.***out***.print(arr1[i] + " ");  **for** (**int** i = 0; i < n; i++) {  **if** (binarySearch(arr1, 0, m - 1, arr2[i]) == -1)  System.***out***.print(arr2[i] + " ");  }  }  **int** binarySearch(**int** arr[], **int** l, **int** r, **int** x) {  **if** (r >= l) {  **int** mid = l + (r - l) / 2;  **if** (arr[mid] == x)  **return** mid;  **if** (arr[mid] > x)  **return** binarySearch(arr, l, mid - 1, x);  **return** binarySearch(arr, mid + 1, r, x);  }  **return** -1;  }  } |

# Union Using Hashing

|  |
| --- |
| **public** **class** UnionUsingHashing {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 7, 1, 5, 2, 3, 6 };  **int** arr2[] = { 3, 8, 6, 20, 7 };  System.***out***.println("Union of two arrays is : ");  *printUnion*(arr1, arr2);  }  **static** **void** printUnion(**int** arr1[], **int** arr2[]) {  HashSet<Integer> hs = **new** HashSet<>();  **for** (**int** i = 0; i < arr1.length; i++)  hs.add(arr1[i]);  **for** (**int** i = 0; i < arr2.length; i++)  hs.add(arr2[i]);  System.***out***.println(hs);  }  } |

# Unsorted Array Delete

|  |
| --- |
| //Time complexities: O(n)  **public** **class** Unsorted\_Array\_Delete {  **public** **static** **void** main(String[] args) {  **int** i;  **int** arr[] = { 10, 50, 30, 40, 20 };  **int** n = arr.length;  **int** key = 30;  System.***out***.println("Array before deletion");  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  n = *deleteElement*(arr, n, key);  System.***out***.println("\n\nArray after deletion");  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **int** deleteElement(**int** arr[], **int** n, **int** key) {  **int** pos = *findElement*(arr, n, key);  **if** (pos == -1) {  System.***out***.println("Element not found");  **return** n;  }  **int** i;  **for** (i = pos; i < n - 1; i++)  arr[i] = arr[i + 1];  **return** n - 1;  }  **static** **int** findElement(**int** arr[], **int** n, **int** key) {  **int** i;  **for** (i = 0; i < n; i++)  **if** (arr[i] == key)  **return** i;  **return** -1;  }  } |

# Unsorted Array Insert

|  |
| --- |
| //Time complexities: O(1)  **public** **class** Unsorted\_Array\_Insert {  **public** **static** **void** main(String[] args) {  **int**[] arr = **new** **int**[20];  arr[0] = 12;  arr[1] = 16;  arr[2] = 20;  arr[3] = 40;  arr[4] = 50;  arr[5] = 70;  **int** capacity = 20;  **int** n = 6;  **int** i, key = 26;  System.***out***.print("Before Insertion: ");  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  n = *insertSorted*(arr, n, key, capacity);  System.***out***.print("\nAfter Insertion: ");  **for** (i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **static** **int** insertSorted(**int** arr[], **int** n, **int** key, **int** capacity) {  **if** (n >= capacity)  **return** n;  arr[n] = key;  **return** (n + 1);  }  } |

# Unsorted Array Search

|  |
| --- |
| //Time complexities: O(n)  **public** **class** Unsorted\_Array\_Search {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 34, 10, 6, 40 };  **int** n = arr.length;  **int** key = 40;  **int** position = *findElement*(arr, n, key);  **if** (position == -1)  System.***out***.println("Element not found");  **else**  System.***out***.println("Element Found at Position: " + (position + 1));  }  **static** **int** findElement(**int** arr[], **int** n, **int** key) {  **for** (**int** i = 0; i < n; i++)  **if** (arr[i] == key)  **return** i;  **return** -1;  }  } |

# Verification In Java

|  |
| --- |
| **public** **class** Verification\_In\_Java {  **private** **float** bal;  **public** **static** **void** main(String[] args) {  Verification\_In\_Java obj = **new** Verification\_In\_Java();  System.***out***.println(obj.depositBalance(4000));  }  **float** depositBalance(**int** bal) {  **int** myBal = bal;  **this**.bal += myBal;  **return** **this**.bal;  }  } |

# Wrapper classes

|  |
| --- |
| **Utility methods of Wrapper classes:-**  **1. valueOf () method:-**We can use valueOf() method to create Wrapper object for given primitive or String.  There are 3 types of valueOf() methods:  **a.Wrapper valueOf(String s):-**Every wrapper class except Character class contains a static valueOf() method to create Wrapper class object for given String. **Syntax: public static Wrapper valueOf(String s);**  **b.Wrapper valueOf(String s, int radix):-**Every Integral Wrapper class Byte, Short, Integer, Long) contains the following valueOf() method to create a Wrapper object for the given String with specified radix. The range of the radix is 2 to 36.  **Syntax: public static Wrapper valueOf(String s, int radix)**  **c.Wrapper valueOf(primitive p):-** Every Wrapper class including Character class contains the following method to create a Wrapper object for the given primitive type.**Syntax:public static Wrapper valueOf(primitive p);**  **2. xxxValue() method:-** We can use xxxValue() methods to get the primitive for the given Wrapper Object.Every number type  Wrapper class( Byte, Short, Integer, Long, Float, Double) contains the following 6 methods to get primitive for the given  Wrapper object:   * + **public byte byteValue()**   + **public short shortValue()**   + **public int intValue()**   + **public long longValue()**   + **public float floatValue()**   + **public float doubleValue()**   **3. parseXxx() method:-** We can use parseXxx() methods to convert String to primitive. There are two types of parseXxx()  methods:  **a.primitive parseXxx(String s):-** Every Wrapper class except character class contains the following parseXxx() method to find primitive for the given String object. **Syntax: public static primitive parseXxx(String s);**  **b.parseXxx(String s, int radix)-** Every Integral type Wrapper class (Byte, Short, Integer, Long) contains the following parseXxx() method to convert specified radix String to primitive.**Syntax:public static primitive parseXxx(String s, int radix);**  **4. toString() method:-** We can use toString() method to convert Wrapper object or primitive to String.  There are few forms of toString() method:  **a.public String toString():-**Every wrapper class contains the following toString() method to convert Wrapper Object to String type. **Syntax:public String toString();**  **b.toString(primitive p):-**Every Wrapper class including Character class contains the following static toString() method  to convert primitive to String.**Syntax:public static String toString(primitive p);**  **c.toString(primitive p, int radix):-**Integer and Long classes contains the following toString() method to convert primitve to specified radix String.**Syntax:public static String toString(primitive p, int radix);**  **public** **class** WrapperClasses {  **public** **static** **void** main(String[] args) {  Integer I = Integer.*valueOf*("10"); // Wrapper valueOf(String s)  System.***out***.println(I);  //Integer I1 = Integer.valueOf("ten"); //Wrapper valueOf(String s)  //System.out.println(I1); // output :- Exception in thread "main" java.lang.NumberFormatException: For input string: "ten"  Integer I2 = Integer.*valueOf*("1111", 2); //Wrapper valueOf(String s, int radix)  System.***out***.println(I2);  Integer I3 = Integer.*valueOf*(10); //Wrapper valueOf(primitive p)    System.***out***.println(I3);  Integer I4 = **new** Integer(130); //xxxValue() method  System.***out***.println(I4.byteValue());    **int** i = Integer.*parseInt*("10"); //primitive parseXxx(String s)  System.***out***.println(i);  **double** d = Double.*parseDouble*("10.5"); //primitive parseXxx(String s)  System.***out***.println(d);  **boolean** b = Boolean.*parseBoolean*("true"); //primitive parseXxx(String s)  System.***out***.println(b);    **int** i1 = Integer.*parseInt*("1000", 2); //parseXxx(String s, int radix)  System.***out***.println(i1);  **long** l = Long.*parseLong*("1111", 4); //parseXxx(String s, int radix)  System.***out***.println(l);    Integer I5 = **new** Integer(10); //String toString()  String s = I5.toString(); //String toString()  System.***out***.println(s);    String s1 = Integer.*toString*(10); //toString(primitive p)  System.***out***.println(s1);  String s2 = Character.*toString*('a'); //toString(primitive p)  System.***out***.println(s2);    String s3 = Integer.*toString*(15, 2); //toString(primitive p, int radix)  System.***out***.println(s3);  String s4 = Long.*toString*(11110000, 4); //toString(primitive p, int radix)  System.***out***.println(s4);  }  } |

# Sort Employee list on basis of name in ascending order by implementing Comparable interface and overriding its compareTo method in java

|  |
| --- |
| **public** **class** ComparableUsageExample {  **public** **static** **void** main(String[] args) {  Employee emp1 = **new** Employee("sam", "4");  Employee emp2 = **new** Employee("amy", "2");  Employee emp3 = **new** Employee("brad", "1");  ArrayList<Employee> list = **new** ArrayList<Employee>();  list.add(emp1);  list.add(emp2);  list.add(emp3);  System.***out***.println("list Before sorting : \n" + list);  Collections.*sort*(list);  System.***out***.println("\nlist after sorting on basis of name(ascending order) : \n" + list);  }  }  **class** Employee **implements** Comparable<Employee> {  String name;  String id;  **public** Employee(String name, String id) {  **this**.name = name;  **this**.id = id;  }  @Override  **public** **int** compareTo(Employee otherEmployee) {  // sort Employee on basis of name(ascending order)  **return** **this**.name.compareTo(otherEmployee.name);  }  @Override  **public** String toString() {  **return** "Employee{" + "name=" + name + ", id=" + id + '}';  }  } |

# Sort Employee list on basis of name in ascending order by implementing Comparator interface and overriding its compare method in java

|  |
| --- |
| **public** **class** ComparatorUsageExample {  **public** **static** **void** main(String[] args) {  Employee1 emp1 = **new** Employee1("sam", "4");  Employee1 emp2 = **new** Employee1("amy", "2");  Employee1 emp3 = **new** Employee1("brad", "1");  ArrayList<Employee1> list = **new** ArrayList<Employee1>();  list.add(emp1);  list.add(emp2);  list.add(emp3);  System.***out***.println("list Before sorting : \n" + list);  Collections.*sort*(list, **new** Employee1());  System.***out***.println("\nlist after sorting on basis of name(ascending order) : \n" + list);  }  }  **class** Employee1 **implements** Comparator<Employee1> {  String name;  String id;  **public** Employee1() {  }  **public** Employee1(String name, String id) {  **this**.name = name;  **this**.id = id;  }  @Override  **public** **int** compare(Employee1 obj1, Employee1 obj2) {  // sort Employee on basis of name(ascending order)  **return** obj1.name.compareTo(obj2.name);  }  @Override  **public** String toString() {  **return** "Employee{" + "name=" + name + ", id=" + id + '}';  }  } |

# Sort Employee list on basis of name and id in ascending order by implementing Comparator interface and overriding its compare method in java

|  |
| --- |
| **public** **class** ComparatorUsageExample2 {  **public** **static** **void** main(String[] args) {  Employee2 emp1 = **new** Employee2("sam", "4");  Employee2 emp2 = **new** Employee2("amy", "2");  Employee2 emp3 = **new** Employee2("brad", "1");  ArrayList<Employee2> list = **new** ArrayList<Employee2>();  list.add(emp1);  list.add(emp2);  list.add(emp3);  System.***out***.println("list Before sorting : \n" + list);  Collections.*sort*(list, **new** ComparatorName());  System.***out***.println("\nlist after sorting on basis of name(ascending order) : \n" + list);  Collections.*sort*(list, **new** ComparatorId());  System.***out***.println("\nlist after sorting on basis of id(ascending order) : \n" + list);  }  }  **class** Employee2 {  String name;  String id;  **public** Employee2() {  }  **public** Employee2(String name, String id) {  **this**.name = name;  **this**.id = id;  }  @Override  **public** String toString() {  **return** "Employee2{" + "name=" + name + ", id=" + id + '}';  }  }  **class** ComparatorName **implements** Comparator<Employee2> {  @Override  **public** **int** compare(Employee2 obj1, Employee2 obj2) {  // sort Employee2 on basis of name(ascending order)  **return** obj1.name.compareTo(obj2.name);  }  }  **class** ComparatorId **implements** Comparator<Employee2> {  @Override  **public** **int** compare(Employee2 obj1, Employee2 obj2) {  // sort Employee2 on basis of id(ascending order)  **return** obj1.id.compareTo(obj2.id);  }  } |

# Sort Employee list on basis of name and id in ascending order by implementing Comparator interface in inner and static nested class and overriding its compare method in java

|  |
| --- |
| **public** **class** ComparatorUsageExample3 {  **public** **static** **void** main(String[] args) {  Employee3 emp1 = **new** Employee3("sam", "4");  Employee3 emp2 = **new** Employee3("amy", "2");  Employee3 emp3 = **new** Employee3("brad", "1");  ArrayList<Employee3> list = **new** ArrayList<Employee3>();  list.add(emp1);  list.add(emp2);  list.add(emp3);  System.***out***.println("list Before sorting : \n" + list);  Collections.*sort*(list, **new** Employee3().**new** ComparatorName());  System.***out***.println("\nlist after sorting on basis of name(ascending order), " + "using inner class : \n" + list);  Collections.*sort*(list, **new** Employee3.ComparatorId());  System.***out***.println( "\nlist after sorting on basis of id(ascending order), " + "using static nested class : \n" + list);  }  }  **class** Employee3 {  String name;  String id;  **public** Employee3() {  }  **public** Employee3(String name, String id) {  **this**.name = name;  **this**.id = id;  }  @Override  **public** String toString() {  **return** "Employee3{" + "name=" + name + ", id=" + id + '}';  }  // Inner class  **class** ComparatorName **implements** Comparator<Employee3> {  @Override  **public** **int** compare(Employee3 obj1, Employee3 obj2) {  // sort Employee3 on basis of name(ascending order)  **return** obj1.name.compareTo(obj2.name);  }  }  // static nested class  **static** **class** ComparatorId **implements** Comparator<Employee3> {  @Override  **public** **int** compare(Employee3 obj1, Employee3 obj2) {  // sort Employee3 on basis of id(ascending order)  **return** obj1.id.compareTo(obj2.id);  }  }  } |

# Arithmetic Exception

|  |
| --- |
| **public** **class** Arithmetic\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  **int** a = 30, b = 0;  **int** c = a / b; // cannot divide by zero  System.***out***.println("Result = " + c);  } **catch** (ArithmeticException e) {  System.***out***.println("Can't divide a number by 0");  }  }  } |

# ArrayIndexOutOfBounds Exception: - It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to size of the array.

|  |
| --- |
| **public** **class** ArrayIndexOutOfBound\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  **int** a[] = **new** **int**[5];  a[6] = 9; // accessing 7th element in an array of size 5  } **catch** (ArrayIndexOutOfBoundsException e) {  System.***out***.println("Array Index is Out Of Bounds");  }  }  } |

# Assertion Error: - Exception rises explicitly by the programmer or by API developer to indicate that assert statement fails.

|  |
| --- |
| **public** **class** AssertionError {  **public** **static** **void** main(String[] args) {  **assert**(x >= 10);  }  } |

# Checked Exceptions

|  |
| --- |
| **public** **class** Checked\_Exceptions {  **public** **static** **void** main(String[] args) {  Checked\_Exceptions obj = **new** Checked\_Exceptions();  obj.p();  System.***out***.println ("normal flow...");  }  **void** p() {  **try** {  n();  } **catch** (Exception e) {  System.***out***.println("exception handled");  }  }  **void** n() **throws** IOException {  m();  }  **void** m() **throws** IOException {  **throw** **new** IOException("device error");  }  } |

# Class Cast Exception

|  |
| --- |
| **public** **class** ClassCast\_Exception {  **public** **static** **void** main(String[] args) {  String s = **new** String("Geeks");  Object o = (Object) s;  Object o1 = **new** Object();  String s1 = (String) o1;  }  } |

# ClassNotFound Exception: - ClassNotFoundException occurs when you try to load a class at runtime using Class.forName () or loadClass () methods and requested classes are not found in Classpath. Most of the time this exception will occur when you try to run application without updating Classpath with JAR files. This exception is checked exception derived from java.lang.Exception class and you need to provide explicit handling for it.This exception also occurs when you have two classes loaders and if a ClassLoader tries to access a class which is loaded by another classLoader in java. Java classLoader is a part of java runtime environment that dynamically loads java classes in JVM. The java runtime system does not need to know about files and files syatem because of classloader.

|  |
| --- |
| **public** **class** ClassNotFound\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  Class.*forName*("GeeksForGeeks");  } **catch** (ClassNotFoundException ex) {  ex.printStackTrace();  }  }  } |

# ClassNotFoundException Vs NoClassDefFoundError

|  |
| --- |
| As the name suggests, ClassNotFoundException is an exception while NoClassDefFoundError is an error.  ClassNotFoundException occurs when classpath is does not get updated with required JAR files while error occurs when required class definition is not present at runtime. |

# Concurrent Modification Exception

|  |
| --- |
| **public** **class** ConcurrentModification\_Exception **extends** Thread {  **static** ArrayList *l* = **new** ArrayList();  **public** **static** **void** main(String[] args) **throws** InterruptedException {  *l*.add("A");  *l*.add("B");  *l*.add("c");  ConcurrentModification\_Exception t = **new** ConcurrentModification\_Exception();  t.start();  Iterator itr = *l*.iterator();  **while** (itr.hasNext()) {  String s = (String) itr.next();  System.***out***.println(s);  Thread.*sleep*(6000);  }  System.***out***.println(*l*);  }  **public** **void** run() {  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  System.***out***.println("Child Thread" + " going to add element");  }  *l*.add("D");  }  } |

# Exception InInitializer Error: - Exception occurs whenever while executing static variable assignment and static block if any Exception occurs.

|  |
| --- |
| **Method 1:-**  **public** **class** ExceptionInInitializerError {  **static** {  String s = **null**;  System.***out***.println(s.length());  }  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  }  }  **Method 2:-**  **public** **class** ExceptionInInitializer\_Error {  **static** **int** *x* = 10 / 0;  **public** **static** **void** main(String[] args) {  }  } |

# FileNotFoundException: - this exception is raised when a file is not accessible or does not open.

|  |
| --- |
| **public** **class** FileNotFound\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  // Following file does not exist  File file = **new** File("E:// file.txt");  FileReader fr = **new** FileReader(file);  } **catch** (FileNotFoundException e) {  System.***out***.println("File does not exist");  }  }  } |

# IllegalArgument Exception: - this occurs when the argument passed is not positive. The exception occurs explicitly either by programmer or by API developer to indicate that method has been invoked with illegal argument.

|  |
| --- |
| **Method 1:-**  **public** **class** IllegalArgument\_Exception {  **public** **static** **void** main(String[] args) {  Random ran = **new** Random();  // generating number btween 0 and -12345 Raises Runtime error, as n is negative.  **int** nxt = ran.nextInt(-12345);  System.***out***.println("Generated Random number is : " + nxt);  }  }  **Method 2:-**  **public** **class** IllegalArgumentException {  **public** **static** **void** main(String[] args) {  Thread t = **new** Thread();  Thread t1 = **new** Thread();  t.setPriority(7); // Correct  t1.setPriority(17); // Exception  }  } |

# IndexOutOfBound Exception

|  |
| --- |
| **public** **class** IndexOutOfBound\_Exception {  **public** **static** **void** main(String[] args) {  LinkedList list = **new** LinkedList();  // adding elements using add()  list.add("Geeks");  list.add(4);  list.add("Geeks");  list.add("8");  // Trying to get element at index 7 throws exception  System.***out***.println("The element at index 7 is : " + list.get(7));  }  } |

# InterruptedException: - it is thrown when a thread is waiting, sleeping or doing some processing and it is interrupted.

|  |
| --- |
| **public** **class** Interrupted\_Exception {  **public** **static** **void** main(String[] args) {  Thread t = **new** Thread();  **try** {  t.*sleep*(100);  } **catch** (InterruptedException e) {  System.***out***.println("InterruptedException");  }  }  } |

# IOException: - it is thrown when an io/op operation failed or interrupted.

|  |
| --- |
| **public** **class** IO\_Exception {  **public** **static** **void** main(String[] args) {  FileInputStream f = **null**;  **try** {  f = **new** FileInputStream("abc.txt");  **int** i;  **while** ((i = f.read()) != -1) {  System.***out***.print((**char**) i);  }  } **catch** (IOException e) {  // f.close();  System.***out***.println("IOException");  }  }  } |

# NoClassDefFoundError: - NoClassDefFoundError occur when class present during complie time and program was compiled and linked successfully but class was not present during runtime.it is error which is derived from LinkageError. LinkageError occurs when a class has some dependencies on another class and letter class changes after compilation of former class. NoClassDefFoundError is the result of implicit loading of class because of calling a method or accessing a variable from that class. This error is more difficult to debug and find the reason why this error occurred, so in this case you should always check the classes which are dependent on this class.

|  |
| --- |
| **public** **class** NoClassDefFoundError\_Example {  **public** **static** **void** main(String[] args) {  GeeksForGeeks geeks = **new** geeksForGeeks();  geeks.greeting();  }  }  **class** GeeksForGeeks {  **void** greeting() {  System.***out***.println ("hello!");  }  } |

# NoSuchElement Exception

|  |
| --- |
| **public** **class** NoSuchElement\_Exception {  **public** **static** **void** main(String[] args) {  LinkedList list = **new** LinkedList();  // Trying to get first element at index 7 throws exception  System.***out***.println("The first element of list is : " + list.getFirst());  }  } |

# NoSuchMethodException: - it is thrown when accessing a method which is not found.

|  |
| --- |
| **public** **class** NoSuchMethod\_Exception {  **public** **static** **void** main(String[] args) {  **new** NoSuchMethod\_Exception();  }  **public** NoSuchMethod\_Exception() {  Class i;  **try** {  i = Class.*forName*("java.lang.String");  **try** {  Class[] p = **new** Class[5];  } **catch** (SecurityException e) {  System.***out***.println("SecurityException");  } **catch** (NoSuchMethodException e) {  System.***out***.println("NoSuchMethodException");  }  } **catch** (ClassNotFoundException e) {  System.***out***.println("ClassNotFoundException");  }  }  } |

# NullPointerException: - this exception is raised when referring to the members of null object.Null represents nothing.

|  |
| --- |
| **public** **class** NullPointer\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  String a = **null**; // null value  System.***out***.println(a.charAt(0));  } **catch** (NullPointerException e) {  System.***out***.println ("NullPointerException...”);  }  }  } |

# NumberFormatException: - this exception is raised when a method could not convert a string into a numeric format.

|  |
| --- |
| **public** **class** NumberFormat\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  // "akki" is not a number  **int** num = Integer.*parseInt*("akki");  System.***out***.println(num);  } **catch** (NumberFormatException e) {  System.***out***.println("Number format exception");  }  }  } |

# StackOverflow Error

|  |
| --- |
| **public** **class** StackOverflow\_Error {  **public** **static** **void** main(String[] args) {  *m1*();  }  **public** **static** **void** m1() {  *m2*();  }  **public** **static** **void** m2() {  *m1*();  }  } |

# StringIndexOutOfBoundsException: - it is thrown by String class methos to indicate that an index is either negative than the size of the string.

|  |
| --- |
| **public** **class** StringIndexOutOfBounds\_Exception {  **public** **static** **void** main(String[] args) {  **try** {  String a = "This is like chipping "; // length is 22  **char** c = a.charAt(24); // accessing 25th element  System.***out***.println(c);  } **catch** (StringIndexOutOfBoundsException e) {  System.***out***.println("StringIndexOutOfBoundsException");  }  }  } |

# TreeMap putAll NullPointerException: - occurs if the map contains a null key or it is null in putAll ().

|  |
| --- |
| **public** **class** TreeMap\_putAll\_NullPointerException {  **public** **static** **void** main(String[] args) {  TreeMap<String, Integer> tmp1 = **new** TreeMap<String, Integer>();  TreeMap<String, Integer> tmp2 = **new** TreeMap<String, Integer>();  tmp1.put("two", 3);  tmp1.put("one", 1);  // assigning in 2nd TreeMap  tmp2.put("three", 3);  tmp2.put(**null**, 2);  System.***out***.println("First treemap values are : " + tmp1);  // use of putAll() Putting 2nd map in 1st map  tmp1.putAll(tmp2);  System.***out***.println("Values after modifying 1st treemap : " + tmp1);  }  } |

# Best ranked person in given list, or can be used to assign winner in game in which person with lowest time to finish a task wins.

|  |
| --- |
| **public** **class** Best\_Ranked\_Person {  **public** **static** **void** main(String[] args) {  TreeMap<Float, String> time = **new** TreeMap<Float, String>();  **// assigning the time taken to complete task** **using put()**  time.put(2.32f, "Astha");  time.put(7.43f, "Manjeet");  time.put(1.3f, "Shambhavi");  time.put(5.63f, "Nikhil");  time.put(6.26f, "Vaishnavi");  **// use of firstEntry()** **printing person with least time**  System.***out***.println("Winner with lowest time is : " + time.firstEntry());  }  } |

# BrainFuck Interpreter

|  |
| --- |
| Input: -- [+++++++>-->+>+>+<<<->---.>--..>+.<<<.+>->>.+++[.<] Output: Hello World!  Input: ++++++++++[>+++++++>++++++++>+++<+++.>++..<+. Output : GEEKS FOR GEEKS  **public** **class** BrainFuck\_Interpreter {  **private** **static** Scanner *ob* = **new** Scanner(System.***in***);  **private** **static** **int** *ptr*;  **private** **static** **int** *length* = 65535;  **private** **static** **byte** *memory*[] = **new** **byte**[*length*];  **public** **static** **void** main(String[] args) {  System.***out***.println("Enter the code:");  String code = *ob*.nextLine();  System.***out***.println("Output:");  *interpret*(code);  }  **private** **static** **void** interpret(String s) {  **int** c = 0;  **for** (**int** i = 0; i < s.length(); i++) {  **if** (s.charAt(i) == '>') {  **if** (*ptr* == *length* - 1)  *ptr* = 0;  **else**  *ptr*++;  }  **else** **if** (s.charAt(i) == '<') {  **if** (*ptr* == 0)  *ptr* = *length* - 1;  **else**  *ptr*--;  }  **else** **if** (s.charAt(i) == '+')  *memory*[*ptr*]++;  **else** **if** (s.charAt(i) == '-')  *memory*[*ptr*]--;  **else** **if** (s.charAt(i) == '.')  System.***out***.print((**char**) (*memory*[*ptr*]));  **else** **if** (s.charAt(i) == ',')  *memory*[*ptr*] = (**byte**) (*ob*.next().charAt(0));  **else** **if** (s.charAt(i) == '[') {  **if** (*memory*[*ptr*] == 0) {  i++;  **while** (c > 0 || s.charAt(i) != ']') {  **if** (s.charAt(i) == '[')  c++;  **else** **if** (s.charAt(i) == ']')  c--;  i++;  }  }  }  **else** **if** (s.charAt(i) == ']') {  **if** (*memory*[*ptr*] != 0) {  i--;  **while** (c > 0 || s.charAt(i) != '[') {  **if** (s.charAt(i) == ']')  c++;  **else** **if** (s.charAt(i) == '[')  c--;  i--;  }  i--;  }  }  }  }  } |

# Implemented by Cricket data to communicate with observers OR Java program to demonstrate working of onserver pattern OR Building Cricket Score APP

|  |
| --- |
| **interface** Subject {  **public** **void** registerObserver(Observer o);  **public** **void** unregisterObserver(Observer o);  **public** **void** notifyObservers();  }  **class** CricketData **implements** Subject {  **int** runs;  **int** wickets;  **float** overs;  ArrayList<Observer> observerList;  **public** CricketData() {  observerList = **new** ArrayList<Observer>();  }  @Override  **public** **void** registerObserver(Observer o) {  observerList.add(o);  }  @Override  **public** **void** unregisterObserver(Observer o) {  observerList.remove(observerList.indexOf(o));  }  @Override  **public** **void** notifyObservers() {  **for** (Iterator<Observer> it = observerList.iterator(); it.hasNext();) {  Observer o = it.next();  o.update(runs, wickets, overs);  }  }  **private** **int** getLatestRuns() {  **return** 90;  }  **private** **int** getLatestWickets() {  **return** 2;  }  **private** **float** getLatestOvers() {  **return** (**float**) 10.2;  }  **public** **void** dataChanged() {  runs = getLatestRuns();  wickets = getLatestWickets();  overs = getLatestOvers();  notifyObservers();  }  }  **interface** Observer {  **public** **void** update(**int** runs, **int** wickets, **float** overs);  }  **class** AverageScoreDisplay **implements** Observer {  **private** **float** runRate;  **private** **int** predictedScore;  **public** **void** update(**int** runs, **int** wickets, **float** overs) {  **this**.runRate = (**float**) runs / overs;  **this**.predictedScore = (**int**) (**this**.runRate \* 50);  display();  }  **public** **void** display() {  System.***out***.println( "\nAverage Score Display: \n" + "Run Rate: " + runRate + "\nPredictedScore: " + predictedScore);  }  }  **class** CurrentScoreDisplay **implements** Observer {  **private** **int** runs, wickets;  **private** **float** overs;  **public** **void** update(**int** runs, **int** wickets, **float** overs) {  **this**.runs = runs;  **this**.wickets = wickets;  **this**.overs = overs;  display();  }  **public** **void** display() {  System.***out***.println("\nCurrent Score Display:\n" + "Runs: " + runs + "\nWickets:" + wickets + "\nOvers: " + overs);  }  }  **public** **class** Building\_Cricket\_Score\_APP {  **public** **static** **void** main(String args[]) {  AverageScoreDisplay averageScoreDisplay = **new** AverageScoreDisplay();  CurrentScoreDisplay currentScoreDisplay = **new** CurrentScoreDisplay();  CricketData cricketData = **new** CricketData();  cricketData.registerObserver(averageScoreDisplay);  cricketData.registerObserver(currentScoreDisplay);  cricketData.dataChanged();  cricketData.unregisterObserver(averageScoreDisplay);  cricketData.dataChanged();  }  } |

# **Game of Brussels sprouts: -** given n number of spots, and two players. Players need to connect any two apots without intersecting any of the drawn line, the player who cannot give a move, loses. Determine who will win player1 or player2.

|  |
| --- |
| /\*Input : n = 2 Output : player 2  Input : n = 3 Output : player 1\*/  **public** **class** GameofBrusselsSprouts {  **public** **static** **void** main(String[] args) {  **int** n1 = 2;  *winner*(n1); //Player 2  **int** n2 = 3;  *winner*(n2); //Player 1  }  **static** **void** winner(**int** n) {  **int** m = 5 \* n - 2;  **if** (m % 2 == 1)  System.***out***.println("Player 1");  **else**  System.***out***.println("Player 2");  }  } |

# **Three N plus One (3n+1) Problem**

|  |
| --- |
| **public** **class** ThreeNPlusOneProblem {  **public** **static** **int**[] *cache* = **new** **int**[3000000];  **public** **static** **int** length(**int** n) {  **if** (n < *cache*.length) {  **if** (*cache*[n] == 0) {  **if** (n % 2 == 0) {  *cache*[n] = 1 + *length*(n / 2);  } **else** {  *cache*[n] = 1 + *length*(3 \* n + 1);  }  }  **return** *cache*[n];  } **else** {  **if** (n % 2 == 0) {  **return** 1 + *length*(n / 2);  } **else** {  **return** 1 + *length*(3 \* n + 1);  }  }  }  **public** **static** **void** main(String[] args) **throws** IOException {  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***));  String s;  *cache*[1] = 1;  **while** ((s = br.readLine()) != **null**) {  StringTokenizer st = **new** StringTokenizer(s);  **int** in1 = Integer.*parseInt*(st.nextToken());  **int** in2 = Integer.*parseInt*(st.nextToken());  **int** min = Math.*min*(in1, in2);  **int** max = Math.*max*(in1, in2);  **int** maxCycle = 0;  **for** (**int** i = min; i <= max; i++) {  maxCycle = Math.*max*(*length*(i), maxCycle);  }  System.***out***.println(in1 + " " + in2 + " " + maxCycle);  }  }  } |

# **Circular Queue Example: -** implementation with generics in java, insert and remove element from Circuler Queues in java.

|  |
| --- |
| Complexity of Circular Queues in java  enqueue - O(1) [as we insert element at Rear in Circular Queue]  dequeue - O(1) [as we remove element from front in Circular Queue]\*/  **public** **class** CircularQueueExample {  **public** **static** **void** main(String[] args) {  CircularQueue<Integer> circularQueue = **new** CircularQueue<Integer>(5);  circularQueue.enqueue(11);  circularQueue.enqueue(21);  circularQueue.enqueue(31);  circularQueue.enqueue(51);  circularQueue.enqueue(61);  System.***out***.print("Elements deQueued from circular Queue: ");  System.***out***.print(circularQueue.dequeue() + " ");  System.***out***.print(circularQueue.dequeue() + " ");  System.***out***.print(circularQueue.dequeue() + " ");  System.***out***.print(circularQueue.dequeue() + " ");  System.***out***.print(circularQueue.dequeue() + " ");  }  }  **class** CircularQueue<E> {  **private** E[] circularQueueAr;  **private** **int** maxSize;  **private** **int** rear;  **private** **int** front;  **private** **int** number;  **public** CircularQueue(**int** maxSize) {  **this**.maxSize = maxSize;  circularQueueAr = (E[]) **new** Object[**this**.maxSize];  number = 0;  front = 0;  rear = 0;  }  **public** **void** enqueue(E item) **throws** QueueFullException {  **if** (isFull()) {  **throw** **new** QueueFullException("Circular Queue is full");  } **else** {  circularQueueAr[rear] = item;  rear = (rear + 1) % circularQueueAr.length;  number++;  }  }  **public** E dequeue() **throws** QueueEmptyException {  E deQueuedElement;  **if** (isEmpty()) {  **throw** **new** QueueEmptyException("Circular Queue is empty");  } **else** {  deQueuedElement = circularQueueAr[front];  circularQueueAr[front] = **null**;  front = (front + 1) % circularQueueAr.length;  number--;  }  **return** deQueuedElement;  }  **public** **boolean** isFull() {  **return** (number == circularQueueAr.length);  }  **public** **boolean** isEmpty() {  **return** (number == 0);  }  }  **class** QueueFullException **extends** RuntimeException {  **public** QueueFullException() {  **super**();  }  **public** QueueFullException(String message) {  **super**(message);  }  }  **class** QueueEmptyException **extends** RuntimeException {  **public** QueueEmptyException() {  **super**();  }  **public** QueueEmptyException(String message) {  **super**(message);  }  } |

# **Priority Queue1 with Default Comaparator**

|  |
| --- |
| **public** **class** PriorityQueue1 {  **public** **static** **void** main(String[] args) {  PriorityQueue<Integer> pQueue = **new** PriorityQueue<Integer>();  pQueue.offer(21);  pQueue.offer(17);  pQueue.offer(37);  pQueue.offer(41);  pQueue.offer(9);  pQueue.offer(67);  pQueue.offer(31);  System.***out***.println(pQueue.poll()); // Output : 9  System.***out***.println(pQueue.poll()); // Output : 17  System.***out***.println(pQueue.poll()); // Output : 21  System.***out***.println(pQueue.poll()); // Output : 31  System.***out***.println(pQueue.poll()); // Output : 37  System.***out***.println(pQueue.poll()); // Output : 41  System.***out***.println(pQueue.poll()); // Output : 67  }  } |

# **Priority Queue2 with customized comaparator**

|  |
| --- |
| **public** **class** PriorityQueue2 {  **public** **static** **void** main(String[] args) {  MyComparator comparator = **new** MyComparator();  PriorityQueue<Employee3> pQueue = **new** PriorityQueue<Employee3>(7, comparator);  pQueue.offer(**new** Employee3("AAA", 15000));  pQueue.offer(**new** Employee3("BBB", 12000));  pQueue.offer(**new** Employee3("CCC", 7500));  pQueue.offer(**new** Employee3("DDD", 17500));  pQueue.offer(**new** Employee3("EEE", 21500));  pQueue.offer(**new** Employee3("FFF", 29000));  pQueue.offer(**new** Employee3("GGG", 14300));  System.***out***.println(pQueue.poll()); // Output --> CCC : 7500  System.***out***.println(pQueue.poll()); // Output --> BBB : 12000  System.***out***.println(pQueue.poll()); // Output --> GGG : 14300  System.***out***.println(pQueue.poll()); // Output --> AAA : 15000  System.***out***.println(pQueue.poll()); // Output --> DDD : 17500  System.***out***.println(pQueue.poll()); // Output --> EEE : 21500  System.***out***.println(pQueue.poll()); // Output --> FFF : 29000  }  }  **class** Employee3 {  String name;  **int** salary;  **public** Employee3(String name, **int** salary) {  **this**.name = name;  **this**.salary = salary;  }  @Override  **public** String toString() {  **return** name + " : " + salary;  }  }  **class** MyComparator **implements** Comparator<Employee3> {  @Override  **public** **int** compare(Employee3 e1, Employee3 e2) {  **return** e1.salary - e2.salary;  }  } |

# **Cyclic Barrier Custom**

|  |
| --- |
| **class** CyclicBarrierCustom {  **int** initialParties; // total parties  **int** partiesAwait; // parties yet to arrive  Runnable cyclicBarrrierEvent;  **public** CyclicBarrierCustom(**int** parties, Runnable cyclicBarrrierEvent) {  initialParties = parties;  partiesAwait = parties;  **this**.cyclicBarrrierEvent = cyclicBarrrierEvent;  }  **public** **synchronized** **void** await() **throws** InterruptedException {  partiesAwait--;  **if** (partiesAwait > 0) {  **this**.wait();  }  **else** {  partiesAwait = initialParties;  notifyAll(); // notify all waiting threads  cyclicBarrrierEvent.run(); // launch event  }  }  }  **public** **class** CyclicBarrierCustomTest {  **public** **static** **void** main(String[] args) {  CyclicBarrierCustom cyclicBarrierCustom = **new** CyclicBarrierCustom(3, **new** CyclicBarrrierEvent());  System.***out***.println("CountDownLatch has been created with parties=3,"  + " when all 3 parties will reach common barrier point " + "CyclicBarrrierEvent will be triggered");  MyRunnable3 myRunnable1 = **new** MyRunnable3(cyclicBarrierCustom);  **new** Thread(myRunnable1, "Thread-1").start();  **new** Thread(myRunnable1, "Thread-2").start();  **new** Thread(myRunnable1, "Thread-3").start();  }  }  **class** MyRunnable3 **implements** Runnable {  CyclicBarrierCustom cyclicBarrierCustom;  MyRunnable3(CyclicBarrierCustom cyclicBarrierCustom) {  **this**.cyclicBarrierCustom = cyclicBarrierCustom;  }  @Override  **public** **void** run() {  System.***out***.println(  Thread.*currentThread*().getName() + " is waiting for all other threads to reach common barrier point");  **try** {  Thread.*sleep*(1000);  cyclicBarrierCustom.await();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println("As all threads have reached common barrier point " + Thread.*currentThread*().getName()  + " has been released");  }  }  **class** CyclicBarrrierEvent **implements** Runnable {  **public** **void** run() {  System.***out***.println(  "As all threads have reached common barrier point " + ", CyclicBarrrierEvent has been triggered");  }  }  /\* OUTPUT  \* CountDownLatch has been created with parties=3, when all 3 parties will reach  \* common barrier point CyclicBarrrierEvent will be triggered Thread-1 is  \* waiting for all other threads to reach common barrier point Thread-2 is  \* waiting for all other threads to reach common barrier point Thread-3 is  \* waiting for all other threads to reach common barrier point As all threads  \* have reached common barrier point , CyclicBarrrierEvent has been triggered As  \* all threads have reached common barrier point Thread-1 has been released As  \* all threads have reached common barrier point Thread-3 has been released As  \* all threads have reached common barrier point Thread-2 has been released \*/ |

# **Read Lock Test: -** Program to demonstrate usage of radlock () method of ReadWriteLock in java. Void readLock () more than one threads can acquire readlock at a time, provided no other thread is acquiring writeLock at same time in java.

|  |
| --- |
| **public** **class** ReadLockTest {  **public** **static** **void** main(String[] args) {  ReadWriteLock readWriteLock = **new** ReentrantReadWriteLock();  MyRunnable myRunnable = **new** MyRunnable(readWriteLock);  **new** Thread(myRunnable, "Thread-1").start();  **new** Thread(myRunnable, "Thread-2").start();  }  }  **class** MyRunnable **implements** Runnable {  ReadWriteLock readWriteLock;  **public** MyRunnable(ReadWriteLock readWriteLock) {  **this**.readWriteLock = readWriteLock;  }  **public** **void** run() {  readWriteLock.readLock().lock();  System.***out***.println (Thread.*currentThread* ().getName () + “has acquired read lock.");  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println (Thread.*currentThread* ().getName () + “has released read lock.");  readWriteLock.readLock().unlock();  }  }  /\* OUTPUT  \* Thread-1 has acquired read lock. Thread-2 has acquired read lock. Thread-1  \* has released read lock. Thread-2 has released read lock. \* / |

# **Write Lock Test: -** Program to demonstrate usage of WriteLock () method of ReadWriteLock in java. Void writeLock () only one threads can acquire writeLock at a time. Means writeLocK can only be optained if no other thread is acquiring read or writelock at a time in java.

|  |
| --- |
| **public** **class** WriteLockTest {  **public** **static** **void** main(String[] args) {  ReadWriteLock readWriteLock = **new** ReentrantReadWriteLock();  MyRunnable1 myRunnable = **new** MyRunnable1(readWriteLock);  **new** Thread(myRunnable, "Thread-1").start();  **new** Thread(myRunnable, "Thread-2").start();  }  }  **class** MyRunnable1 **implements** Runnable {  ReadWriteLock readWriteLock;  **public** MyRunnable1(ReadWriteLock readWriteLock) {  **this**.readWriteLock = readWriteLock;  }  **public** **void** run() {  readWriteLock.writeLock().lock();  System.***out***.println (Thread.*currentThread* ().getName () + “has acquired write lock.");  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println (Thread.*currentThread* ().getName () + “has released write lock.");  readWriteLock.writeLock().unlock();  }  }  /\*OUTPUT  Thread-1 has acquired write lock.  Thread-1 has released write lock.  Thread-2 has acquired write lock.  Thread-2 has released write lock. \*/ |

# **Reentrant Lock Custom Test**

|  |
| --- |
| **interface** LockCustom{  **void** lock();  **void** unlock();  **boolean** tryLock();  }  **class** ReentrantLockCustom **implements** LockCustom {  **int** lockHoldCount;  **long** IdOfThreadCurrentlyHoldingLock;  ReentrantLockCustom(){  lockHoldCount=0;  }  **public** **synchronized** **void** lock() {  **if**(lockHoldCount==0){  lockHoldCount++;  IdOfThreadCurrentlyHoldingLock=Thread.*currentThread*().getId();  }  **else** **if**(lockHoldCount>0  && IdOfThreadCurrentlyHoldingLock==Thread.*currentThread*().getId()){  lockHoldCount++;  }  **else**{  **try** {  wait();  lockHoldCount++;  IdOfThreadCurrentlyHoldingLock=Thread.*currentThread*().getId();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  **public** **synchronized** **void** unlock() {  **if**(lockHoldCount==0)  **throw** **new** IllegalMonitorStateException();  lockHoldCount--;  **if**(lockHoldCount==0)  notify();  }  **public** **synchronized** **boolean** tryLock(){  **if**(lockHoldCount==0){  lock();  **return** **true**;  }  **else**  **return** **false**;  }  }  **public** **class** ReentrantLockCustomTest {  **public** **static** **void** main(String[] args) {  LockCustom LockCustom=**new** ReentrantLockCustom();  MyRunnable2 myRunnable=**new** MyRunnable2(LockCustom);  **new** Thread(myRunnable,"Thread-1").start();  **new** Thread(myRunnable,"Thread-2").start();  }  }  **class** MyRunnable2 **implements** Runnable{  LockCustom lockCustom;  **public** MyRunnable2(LockCustom LockCustom) {  **this**.lockCustom=LockCustom;  }  **public** **void** run(){  System.***out***.println(Thread.*currentThread*().getName()  +" is Waiting to acquire LockCustom");  lockCustom.lock();  System.***out***.println(Thread.*currentThread*().getName()  +" has acquired LockCustom.");  **try** {  Thread.*sleep*(5000);  System.***out***.println(Thread.*currentThread*().getName()  +" is sleeping.");  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println(Thread.*currentThread*().getName()  +" has released LockCustom.");  lockCustom.unlock();  }  }  /\*OUTPUT  Thread-1 is Waiting to acquire LockCustom  Thread-2 is Waiting to acquire LockCustom  Thread-1 has acquired LockCustom.  Thread-1 is sleeping.  Thread-1 has released LockCustom.  Thread-2 has acquired LockCustom.  Thread-2 is sleeping.  Thread-2 has released LockCustom. \*/ |

# **Semaphore Custom Consumer Producer**

|  |
| --- |
| **public** **class** SemaphoreCustomConsumerProducer {  **public** **static** **void** main(String[] args) {  SemaphoreCustom semaphoreProducer = **new** SemaphoreCustom(1);  SemaphoreCustom semaphoreConsumer = **new** SemaphoreCustom(0);  System.***out***.println("semaphoreProducer permit=1 | semaphoreConsumer permit=0");  Producer producer = **new** Producer(semaphoreProducer, semaphoreConsumer);  Consumer consumer = **new** Consumer(semaphoreConsumer, semaphoreProducer);  Thread producerThread = **new** Thread(producer, "ProducerThread");  Thread consumerThread = **new** Thread(consumer, "ConsumerThread");  producerThread.start();  consumerThread.start();  }  }  **class** SemaphoreCustom {  **private** **int** permits;  **public** SemaphoreCustom(**int** permits) {  **this**.permits = permits;  }  **public** **synchronized** **void** acquire() **throws** InterruptedException {  **if** (permits > 0) {  permits--;  }  **else** {  **this**.wait();  permits--;  }  }  **public** **synchronized** **void** release() {  permits++;  **if** (permits > 0)  **this**.notifyAll();  }  }  **class** Producer **implements** Runnable {  SemaphoreCustom semaphoreProducer;  SemaphoreCustom semaphoreConsumer;  **public** Producer(SemaphoreCustom semaphoreProducer, SemaphoreCustom semaphoreConsumer) {  **this**.semaphoreProducer = semaphoreProducer;  **this**.semaphoreConsumer = semaphoreConsumer;  }  **public** **void** run() {  **for** (**int** i = 1; i <= 5; i++) {  **try** {  semaphoreProducer.acquire();  System.***out***.println("Produced : " + i);  semaphoreConsumer.release();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }  **class** Consumer **implements** Runnable {  SemaphoreCustom semaphoreConsumer;  SemaphoreCustom semaphoreProducer;  **public** Consumer(SemaphoreCustom semaphoreConsumer, SemaphoreCustom semaphoreProducer) {  **this**.semaphoreConsumer = semaphoreConsumer;  **this**.semaphoreProducer = semaphoreProducer;  }  **public** **void** run() {  **for** (**int** i = 1; i <= 5; i++) {  **try** {  semaphoreConsumer.acquire();  System.***out***.println("Consumed : " + i);  semaphoreProducer.release();  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }  /\* OUTPUT  \* semaphoreProducer permit=1 | semaphoreConsumer permit=0 Produced : 1 Consumed  \* : 1 Produced : 2 Consumed : 2 Produced : 3 Consumed : 3 Produced : 4 Consumed  \* : 4 Produced : 5 Consumed : 5 \*/ |

# **Semaphore Custom Test**

|  |
| --- |
| **public** **class** SemaphoreCustomTest {  **static** **int** *SharedValue* = 0;  **public** **static** **void** main(String[] args) {  SemaphoreCustom1 semaphore = **new** SemaphoreCustom1(1);  System.***out***.println("Semaphore with 1 permit has been created");  IncrementThread incrementThread = **new** IncrementThread(semaphore);  **new** Thread(incrementThread, "incrementThread").start();  DecrementThread decrementThread = **new** DecrementThread(semaphore);  **new** Thread(decrementThread, "decrementThread").start();  }  }  **class** IncrementThread **implements** Runnable {  SemaphoreCustom1 semaphoreCustom;  **public** IncrementThread(SemaphoreCustom1 semaphoreCustom) {  **this**.semaphoreCustom = semaphoreCustom;  }  **public** **void** run() {  System.***out***.println(Thread.*currentThread*().getName() + " is waiting for permit");  **try** {  semaphoreCustom.acquire();  System.***out***.println(Thread.*currentThread*().getName() + " has got permit");  **for** (**int** i = 0; i < 5; i++) {  Thread.*sleep*(1000);  System.***out***.println(Thread.*currentThread*().getName() + " > " + SemaphoreCustomTest.*SharedValue*++);  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println(Thread.*currentThread*().getName() + " has released permit");  semaphoreCustom.release();  }  }  **class** DecrementThread **implements** Runnable {  SemaphoreCustom1 semaphoreCustom;  **public** DecrementThread(SemaphoreCustom1 semaphoreCustom) {  **this**.semaphoreCustom = semaphoreCustom;  }  **public** **void** run() {  System.***out***.println(Thread.*currentThread*().getName() + " is waiting for permit");  **try** {  semaphoreCustom.acquire();  System.***out***.println(Thread.*currentThread*().getName() + " has got permit");  **for** (**int** i = 0; i < 5; i++) {  Thread.*sleep*(1000);  System.***out***.println(Thread.*currentThread*().getName() + " >" + SemaphoreCustomTest.*SharedValue*--);  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println(Thread.*currentThread*().getName() + " has released permit");  semaphoreCustom.release();  }  }  **class** SemaphoreCustom1 {  **private** **int** permits;  **public** SemaphoreCustom1(**int** permits) {  **this**.permits = permits;  }  **public** **synchronized** **void** acquire() **throws** InterruptedException {  **if** (permits > 0) {  permits--;  }  **else** {  **this**.wait();  permits--;  }  }  **public** **synchronized** **void** release() {  permits++;  **if** (permits > 0)  **this**.notifyAll();  }  }  /\*OUTPUT  Semaphore with 1 permit has been created  incrementThread is waiting for permit  incrementThread has got permit  decrementThread is waiting for permit  incrementThread > 0  incrementThread > 1  incrementThread > 2  incrementThread > 3  incrementThread > 4  incrementThread has released permit  decrementThread has got permit  decrementThread >5  decrementThread >4  decrementThread >3  decrementThread >2  decrementThread >1  decrementThread has released permit \*/ |

# **Thread Death Demo: -** calling stop method on thread throws ThreadDeath error in java.

|  |
| --- |
| **public** **class** ThreadDeathDemo {  **public** **static** **void** main(String[] args) {  Thread thread1 = **new** Thread("thread-1") {  **public** **void** run() {  **try** {  System.***out***.println(Thread.*currentThread*().getName() + " has started");  stop();  } **catch** (ThreadDeath e) {  System.***out***.println(Thread.*currentThread*().getName() + " has died");  }  }  };  thread1.start();  }  }  /\*OUTPUT  thread-1 has started  thread-1 has died \*/ |

# **Add Element at Specific Position**

|  |
| --- |
| **public** **class** AddElementAtSpecificPosition {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> list = **new** LinkedList<Integer>();  list.add(10);  list.add(20);  list.add(30);  list.add(40);  System.***out***.println(list); // Output : [10, 20, 30, 40]  list.add(2, 9999);  System.***out***.println(list); // Output : [10, 20, 9999, 30, 40]  LinkedList<Integer> list1 = **new** LinkedList<Integer>();  list1.addFirst(111);  list1.addFirst(222);  list1.addFirst(333);  System.***out***.println(list1); // Output : [333, 222, 111]  list.addAll(3, list1);  System.***out***.println(list); // Output : [10, 20, 9999, 333, 222, 111, 30, 40]  }  } |

# **Clone of Linked List**

|  |
| --- |
| **public** **class** CloneOfLinkedList {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> linkedList1 = **new** LinkedList<Integer>();  linkedList1.add(10);  linkedList1.add(20);  linkedList1.add(30);  linkedList1.add(40);  linkedList1.add(50);  System.***out***.println(linkedList1); // Output : [10, 20, 30, 40, 50]  LinkedList<Integer> linkedList2 = **new** LinkedList<Integer>();  linkedList2 = (LinkedList<Integer>) linkedList1.clone();  System.***out***.println(linkedList2); // Output : [10, 20, 30, 40, 50]  }  } |

# **Doubly Linked List at First**

|  |
| --- |
| /\*Displaying in forward direction [first--->last] : 39 14 59 21 11  Displaying in backward direction [last-->first] : 11 21 59 14 39  Deleted Nodes: 39 14  Displaying in forward direction [first--->last] : 59 21 11  Displaying in backward direction [last-->first] : 11 21 59 \*/  **public** **class** DoublyLinkedListAtFirst {  **public** **static** **void** main(String[] args) {  LinkedList4 linkedList = **new** LinkedList4();  linkedList.insertFirst(11);  linkedList.insertFirst(21);  linkedList.insertFirst(59);  linkedList.insertFirst(14);  linkedList.insertFirst(39);  linkedList.displayFrwd();  linkedList.displayBckwrd();  System.***out***.print("Deleted Nodes: ");  Node4 deletedNode = linkedList.deleteFirst();  deletedNode.displayNode ();  deletedNode = linkedList.deleteFirst();  deletedNode.displayNode ();  System.***out***.println();  linkedList.displayFrwd();  linkedList.displayBckwrd();  }  }  **class** LinkedList4 {  **private** Node4 first;  **private** Node4 last;  **public** LinkedList4(){  first = **null**;  }  **public** **void** insertFirst(**int** data){  Node4 newNode = **new** Node4(data);  **if** (first == **null**)  last = newNode;  **else**  first.previous = newNode;  newNode.next = first;  first = newNode;  }  **public** Node4 deleteFirst() {  **if**(first==**null**){  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node4 tempNode = first;  **if** (first.next == **null**)  last = **null**;  **else**  first.next.previous = **null**;  first = first.next;  **return** tempNode;  }  **public** **void** displayFrwd() {  System.***out***.print("Displaying in forward direction [first--->last] : ");  Node4 tempDisplay = first;  **while** (tempDisplay != **null**){  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println("");  }  **public** **void** displayBckwrd() {  System.***out***.print("Displaying in backward direction [last-->first] : ");  Node4 tempDisplay = last;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.previous;  }  System.***out***.println("");  }  }  **class** Node4 {  **public** **int** data;.  **public** Node4 next;  **public** Node4 previous;  **public** Node4(**int** data){  **this**.data = data;  }  **public** **void** displayNode() {  System.***out***.print( data + " ");  }  } |

# **Element Exist in Linked List: -** given an element, how do you find out whether that element exist in the linkedlist or not. If it exist retrieve the position of that element.

|  |
| --- |
| **public** **class** ElementExistInLinkedList {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("JAVA");  list.add("J2EE");  list.add("JSP");  list.add("SERVLETS");  list.add("JDBC");  System.***out***.println(list); // Output : [JAVA, J2EE, JSP, SERVLETS, JDBC]  String s = "JSP";  **boolean** contains = list.contains(s);  **if** (contains) {  System.***out***.println(list.indexOf(s)); // Output : 2  }  s = "STRUTS";  contains = list.contains("STRUTS");  **if** (contains) {  System.***out***.println(list.indexOf(s));  }  }  } |

# **Find Circular Singly Linked List Intersection Point Example**

|  |
| --- |
| /\*LinkedList has been successfully converted into CircularLinkedList Displaying LinkedList [first--->last]: 11 22 33 44 55 66 33 44 55 66 33 44 LinkedList is circular at Node: 33\*/  **public** **class** FindCircularSinglyLinkedListInterscetionPointExample {  **public** **static** **void** main(String[] args) {  LinkedList8 linkedList=**new** LinkedList8();  linkedList.insert(66);  linkedList.insert(55);  linkedList.insert(44);  linkedList.insert(33);  linkedList.insert(22);  linkedList.insert(11);  linkedList.makeLinkedListCircular();  linkedList.displayLinkedList();  linkedList.findIntersectionPointOfCircularLikedList();  }  }  **class** LinkedList8{  Node first=**null**;  Node circularPoint1; //points used to make LinkedList circular.  Node circularPoint2;  Node slowPointer; //will step over LinkedList by 1 Node.  Node fastPointer; //will step over LinkedList by 2 Node.  **public** **void** insert(**int** data){  Node newNode=**new** Node(data);  newNode.next=first;  first=newNode;  **if**(data==33)  circularPoint1=newNode;  **if**(data==66)  circularPoint2=newNode;  }  **public** **void** makeLinkedListCircular(){  circularPoint2.next=circularPoint1;  System.***out***.println("LinkedList has been successfully converted into CircularLinkedList");  }  **public** **void** findIntersectionPointOfCircularLikedList(){  slowPointer=first;  fastPointer=first;  **while**( (slowPointer!=fastPointer || slowPointer==first) && fastPointer.next!=**null** && fastPointer.next.next!=**null**){  slowPointer =slowPointer.next; // step over LinkedList by 1 Node.  fastPointer =fastPointer.next.next; // step over LinkedList by 2 Node.  }  slowPointer=first;  **while**(slowPointer!=fastPointer){  slowPointer =slowPointer.next; // step over LinkedList by 1 Node.  fastPointer =fastPointer.next; // step over LinkedList by 1 Node.  }  System.***out***.println("LinkedList is circular at Node: "+slowPointer.data);  }  **public** **void** displayLinkedList(){  Node tempDisplay=first;  **int** displayLimiterCtr=0;  System.***out***.print("Displaying LinkedList [first--->last]: ");  **while**(tempDisplay!=**null**){  tempDisplay.displayNode();  tempDisplay=tempDisplay.next;  **if** (++displayLimiterCtr >= 12) //stops displaying after 12 Nodes.  **break**;  }  System.***out***.println();  }  } |

# **Generic Single Linked List Insert Delete First Example**

|  |
| --- |
| /\*Displaying LinkedList [first--->last]: 39 14 59 21 11 Deleted Nodes: 39 14 Displaying LinkedList [first--->last]: 59 21 11 \*/  **public** **class** GenericSingleLinkedListInsertDeleteFirstExample {  **public** **static** **void** main(String[] args) {  LinkedList10<Integer> linkedList = **new** LinkedList10<Integer>();  linkedList.insertFirst(11);  linkedList.insertFirst(21);  linkedList.insertFirst(59);  linkedList.insertFirst(14);  linkedList.insertFirst(39);  linkedList.displayLinkedList();  System.***out***.print("Deleted Nodes: ");  Node10<Integer> deletedNode = linkedList.deleteFirst();  deletedNode.displayNode();  deletedNode = linkedList.deleteFirst();  deletedNode.displayNode();  System.***out***.println();  linkedList.displayLinkedList();  }  }  **class** LinkedList10<T> {  **private** Node10<T> first;  **public** LinkedList10(){  first = **null**;  }  **public** **void** insertFirst(T data) {  Node10<T> newNode = **new** Node10<T>(data);  newNode.next = first;  first = newNode;  }  **public** Node10<T> deleteFirst() {  **if**(first==**null**){  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node10<T> tempNode = first;  first = first.next;  **return** tempNode;  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList [first--->last]: ");  Node10<T> tempDisplay = first;  **while** (tempDisplay != **null**){  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println();  }  }  **class** Node10<T> {  **public** T data;  **public** Node10<T> next;  **public** Node10(T data){  **this**.data = data;  }  **public** **void** displayNode() {  System.***out***.print( data + " ");  }  } |

# **Get position of last occurrence of a given element in a linkedlist.**

|  |
| --- |
| **public** **class** Get\_position\_of\_last\_occurrence {  **public** **static** **void** main(String[] args) {  LinkedList<String> linkedList = **new** LinkedList<String>();  linkedList.add("AAA");  linkedList.add("BBB");  linkedList.add("CCC");  linkedList.add("BBB");  linkedList.add("FFF");  linkedList.add("BBB");  System.***out***.println(linkedList); // Output : [AAA, BBB, CCC, BBB, FFF, BBB]  System.***out***.println(linkedList.lastIndexOf("BBB")); // Output : 5  }  } |

# **Get Number of Elements**

|  |
| --- |
| **public** **class** GetNumberOfElements {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("JAVA");  list.add("J2EE");  list.add("JSP");  list.add("SERVLETS");  list.add("JDBC");  System.***out***.println(list); // Output : [JAVA, J2EE, JSP, SERVLETS, JDBC]  System.***out***.println(list.size()); // Output : 5  }  } |

# **InOrder Traversal of BinaryTree**

|  |
| --- |
| /\* Output In order Traversal of Binary Tree in java :4 2 5 1 6 3 7 \*/  /\*1. What happens in InOrder traversal of Binary Tree in java: - In InOrder traversal of Binary Tree in java first left subtree is traversed then node is processed and then right subtree is traversed.  2. Steps in InOrder Traversal of Binary Tree in java -   * Traverse the left subtree * Visit the node. * Traverse the right subtree.\*/   public class InOrderTraversalOfBinaryTree {  public static void main(String[] args) {  InOrderTraversalOfBinaryTree inOrderTraversalOfBinaryTree = new InOrderTraversalOfBinaryTree();  BinaryTreeNode binaryTreeNode1 = new BinaryTreeNode(1);  BinaryTreeNode node2 = new BinaryTreeNode(2);  binaryTreeNode1.leftNode = node2;  BinaryTreeNode node3 = new BinaryTreeNode(3);  binaryTreeNode1.rightNode = node3;  BinaryTreeNode node4 = new BinaryTreeNode(4);  node2.leftNode = node4;  BinaryTreeNode node5 = new BinaryTreeNode(5);  node2.rightNode = node5;  BinaryTreeNode node6 = new BinaryTreeNode(6);  node3.leftNode = node6;  BinaryTreeNode node7 = new BinaryTreeNode(7);  node3.rightNode = node7;  System.out.println("In order Traversal of Binary Tree in java : ");  inOrderTraversalOfBinaryTree.inorderTraversalMethod(binaryTreeNode1);  }  public static class BinaryTreeNode {  BinaryTreeNode leftNode;  BinaryTreeNode rightNode;  int data;  BinaryTreeNode(int data) {  this.data = data;  }  }  public void inorderTraversalMethod(BinaryTreeNode binaryTreeNode) {  if (binaryTreeNode != null) {  inorderTraversalMethod(binaryTreeNode.leftNode);  System.out.print(binaryTreeNode.data + " ");  inorderTraversalMethod(binaryTreeNode.rightNode);  }  }  } |

# **PreOrder Traversal of BinaryTree**

|  |
| --- |
| /\* Output Pre order Traversal of Binary Tree in java : 1 2 4 5 3 6 7 \*/  /\*1. What happens in PreOrder traversal of Binary Tree in java: - In PreOrder traversal of Binary Tree in java each node is processed before traversing its subtree.  2. Steps in PreOrder Traversal of Binary Tree in java -   * Visit the node. * Traverse the left subtree. * Traverse the right subtree.\*/   public class PreOrderTraversalOfBinaryTree {  public static void main(String[] args) {  PreOrderTraversalOfBinaryTree preOrderTraversalOfBinaryTree = new PreOrderTraversalOfBinaryTree();  BinaryTreeNode binaryTreeNode1 = new BinaryTreeNode(1);  BinaryTreeNode node2 = new BinaryTreeNode(2);  binaryTreeNode1.leftNode = node2;  BinaryTreeNode node3 = new BinaryTreeNode(3);  binaryTreeNode1.rightNode = node3;  BinaryTreeNode node4 = new BinaryTreeNode(4);  node2.leftNode = node4;  BinaryTreeNode node5 = new BinaryTreeNode(5);  node2.rightNode = node5;  BinaryTreeNode node6 = new BinaryTreeNode(6);  node3.leftNode = node6;  BinaryTreeNode node7 = new BinaryTreeNode(7);  node3.rightNode = node7;  System.out.println("Pre order Traversal of Binary Tree in java : ");  preOrderTraversalOfBinaryTree.preorderTraversalMethod(binaryTreeNode1);  }  public static class BinaryTreeNode {  BinaryTreeNode leftNode;  BinaryTreeNode rightNode;  int data;  BinaryTreeNode(int data) {  this.data = data;  }  }  public void preorderTraversalMethod(BinaryTreeNode binaryTreeNode) {  if (binaryTreeNode != null) {  System.out.print(binaryTreeNode.data + " ");  preorderTraversalMethod(binaryTreeNode.leftNode);  preorderTraversalMethod(binaryTreeNode.rightNode);  }  }  } |

# **PostOrder Traversal of BinaryTree**

|  |
| --- |
| /\*1. What happens in PostOrder traversal of Binary Tree in java :- In PostOrder traversal of Binary Tree in java each node is processed after traversing its subtree.  2. Steps in PostOrder Traversal of Binary Tree in java -   * traverse left subtree * traverse the right subtree. * Visit the node.\*/   /\* Output Post order Traversal of Binary Tree in java :4 5 2 6 7 3 1 \*/  public class PostOrderTraversalOfBinaryTree {  public static void main(String[] args) {  PostOrderTraversalOfBinaryTree postOrderTraversalOfBinaryTree = new PostOrderTraversalOfBinaryTree();  BinaryTreeNode binaryTreeNode1 = new BinaryTreeNode(1);  BinaryTreeNode node2 = new BinaryTreeNode(2);  binaryTreeNode1.leftNode = node2;  BinaryTreeNode node3 = new BinaryTreeNode(3);  binaryTreeNode1.rightNode = node3;  BinaryTreeNode node4 = new BinaryTreeNode(4);  node2.leftNode = node4;  BinaryTreeNode node5 = new BinaryTreeNode(5);  node2.rightNode = node5;  BinaryTreeNode node6 = new BinaryTreeNode(6);  node3.leftNode = node6;  BinaryTreeNode node7 = new BinaryTreeNode(7);  node3.rightNode = node7;  System.out.println("Post order Traversal of Binary Tree in java : ");  postOrderTraversalOfBinaryTree.postorderTraversalMethod(binaryTreeNode1);  }  public static class BinaryTreeNode {  BinaryTreeNode leftNode;  BinaryTreeNode rightNode;  int data;  BinaryTreeNode(int data) {  this.data = data;  }  }  public void postorderTraversalMethod(BinaryTreeNode binaryTreeNode) {  if (binaryTreeNode != null) { // Visit the node-Printing the node data  postorderTraversalMethod(binaryTreeNode.leftNode);  postorderTraversalMethod(binaryTreeNode.rightNode);  System.out.print(binaryTreeNode.data + " ");  }  }  } |

# **Insert an Element at the head and tail of a linkedlist**

|  |
| --- |
| **public** **class** InsertAnElement {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> list = **new** LinkedList<Integer>();  list.add(10);  list.addLast(20);  list.offer(30);  list.offerLast(40);  System.***out***.println(list); // Output : [10, 20, 30, 40]  list.offerFirst(1);  list.addFirst(2);  System.***out***.println(list); // Output : [2, 1, 10, 20, 30, 40]  }  } |

# **Join ArrayList at End of the linkedlist**

|  |
| --- |
| public class JoinArrayListAtEnd {  public static void main(String[] args) {  LinkedList<String> linkedList = new LinkedList<String>();  linkedList.add("ONE");  linkedList.add("TWO");  linkedList.add("THREE");  linkedList.add("FOUR");  linkedList.add("FIVE");  System.out.println(linkedList); // Output : [ONE, TWO, THREE, FOUR, FIVE]  ArrayList<String> arrayList = new ArrayList<String>();  arrayList.add("SIX");  arrayList.add("SEVEN");  arrayList.add("EIGHT");  arrayList.add("NINE");  System.out.println(arrayList); // Output : [SIX, SEVEN, EIGHT, NINE]  linkedList.addAll(arrayList);  System.out.println(linkedList); // Output : [ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE]  }  } |

# **LinkedList as Queue**

|  |
| --- |
| **public** **class** LinkedListAsQueue {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> queue = **new** LinkedList<Integer>();  queue.offer(10);  queue.offer(20);  queue.offer(30);  queue.offer(40);  System.***out***.println(queue); // Output : [10, 20, 30, 40]  System.***out***.println(queue.poll()); // Output : 10  System.***out***.println(queue.poll()); // Output : 20  }  } |

# **LinkedList as Stack**

|  |
| --- |
| **public** **class** LinkedListAsStack {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> stack = **new** LinkedList<Integer>();  stack.push(10);  stack.push(20);  stack.push(30);  stack.push(40);  System.***out***.println(stack); // Output : [40, 30, 20, 10]  System.***out***.println(stack.pop()); // Output : 40  System.***out***.println(stack.pop()); // Output : 30  }  } |

# **LinkedList Circular Calculate Nodes**

|  |
| --- |
| /\*LinkedList has been successfully converted into CircularLinkedList Displaying LinkedList [first--->last]: 11 22 33 44 55 66 33 44 55 66 33 44 LinkedList is circular at Node: 33 Total number of nodes in LinkedList are: 6\*/  **public** **class** LinkedListCircularCalculateNodes {  **public** **static** **void** main(String[] args) {  LinkedList9 linkedList=**new** LinkedList9();  linkedList.insert(66);  linkedList.insert(55);  linkedList.insert(44);  linkedList.insert(33);  linkedList.insert(22);  linkedList.insert(11);  linkedList.makeLinkedListCircular();  linkedList.displayLinkedList();  linkedList.calculateNodesInCircularLikedList();  }  }  **class** LinkedList9{  Node first=**null**;  Node circularPoint1; //points used to make LinkedList circular.  Node circularPoint2;  Node slowPointer; //will step over LinkedList by 1 Node.  Node fastPointer; //will step over LinkedList by 2 Node.  **public** **void** insert(**int** data){  Node newNode=**new** Node(data);  newNode.next=first;  first=newNode;  **if**(data==33)  circularPoint1=newNode;  **if**(data==66)  circularPoint2=newNode;  }  **public** **void** makeLinkedListCircular(){  circularPoint2.next=circularPoint1;  System.***out***.println("LinkedList has been successfully converted into CircularLinkedList");  }  **public** **void** calculateNodesInCircularLikedList(){  slowPointer=first;  fastPointer=first;  **while**( (slowPointer!=fastPointer || slowPointer==first)  && fastPointer.next!=**null**  && fastPointer.next.next!=**null**){  slowPointer =slowPointer.next; // step over LinkedList by 1 Node.  fastPointer =fastPointer.next.next; // step over LinkedList by 2 Node.  }  **int** noOfNodes=0;  slowPointer=first;  **while**(slowPointer!=fastPointer){  slowPointer=slowPointer.next; // step over LinkedList by 1 Node.  fastPointer=fastPointer.next; // step over LinkedList by 1 Node.  noOfNodes++;  }  System.***out***.println("LinkedList is circular at Node: "+slowPointer.data);  **boolean** pointersJustmet=**true**;  **while**(slowPointer!=fastPointer || pointersJustmet){  slowPointer=slowPointer.next; // step over LinkedList by 1 Node.  fastPointer=fastPointer.next.next; // step over LinkedList by 2 Node.  noOfNodes++;  pointersJustmet=**false**;  }  System.***out***.println("Total number of nodes in LinkedList are: "+noOfNodes);  }  **public** **void** displayLinkedList(){  Node tempDisplay=first;  **int** displayLimiterCtr=0;  System.***out***.print("Displaying LinkedList [first--->last]: ");  **while**(tempDisplay!=**null**){  tempDisplay.displayNode();  tempDisplay=tempDisplay.next;  **if**(++displayLimiterCtr >= 12) //stops displaying after 12 Nodes.  **break**;  }  System.***out***.println();  }  } |

# **Traverse the element of LinkedList in Reverse direction**

|  |
| --- |
| public class LinkedListInReverse {  public static void main(String[] args) {  LinkedList<String> list = new LinkedList<String>();  list.add("JAVA");  list.add("J2EE");  list.add("JSP");  list.add("SERVLETS");  list.add("JDBC");  System.out.println(list); // Output : [JAVA, J2EE, JSP, SERVLETS, JDBC]  Iterator<String> it = list.descendingIterator();  while (it.hasNext()) {  System.out.println(it.next());  }  }  } |

# **LinkedList Intersection Point Example**

|  |
| --- |
| //Lists are merging at :19  **public** **class** LinkedListIntersectionPointExample {  **public** **static** **void** main(String[] args) {  List<Integer> l1 = **new** LinkedList<Integer>();  List<Integer> l2 = **new** LinkedList<Integer>();  l1.add(11);  l1.add(13);  l1.add(16);  l1.add(19);  l1.add(22);  l1.add(23);  l2.add(12);  l2.add(14);  l2.add(19);  l2.add(22);  l2.add(23);  *twoListsAreMergingOrNot*(l1, l2);  }  **public** **static** **void** twoListsAreMergingOrNot(List<Integer> l1, List<Integer> l2) {  List<Integer> smallLinkedList = l1;  List<Integer> largeLinkedList = l2;  Integer diffInSize = l2.size() - l1.size();  **int** ctr = 0;  **if** (diffInSize < 0) {.  smallLinkedList = l2;  largeLinkedList = l1;  diffInSize = Math.*abs*(diffInSize);  }  Iterator<Integer> smallListIterator = smallLinkedList.iterator();  Iterator<Integer> largeListIterator = largeLinkedList.iterator();  **while** (largeListIterator.hasNext()) {  **int** listValue = largeListIterator.next();  **if** (ctr < diffInSize) {  ctr++;  **continue**;  }  **if** (listValue == smallListIterator.next()) {  System.***out***.println("Lists are merging at :" + listValue);  **return**;  }  }  }  } |

# **Queue Implementation Using LinkedList**

|  |
| --- |
| **public** **class** QueueImplementationUsingLinkedList {  **public** **static** **void** main(String[] args) {  QueueLinkedList queueLinkedList = **new** QueueLinkedList();  System.***out***.println ("INSERTING AT LAST (REAR) IN QUEUE IMPLEMENTED USING LINKED LIST "); queueLinkedList.insert (11); // insert node.  queueLinkedList.displayStack();  queueLinkedList.insert (71); // insert node.  queueLinkedList.displayStack();  queueLinkedList.insert (39); // insert node.  queueLinkedList.displayStack();  queueLinkedList.insert (31); // insert node.  queueLinkedList.displayStack();  System.***out***.println("\deleting FROM FIRST (FRONT) OF QUEUE IMPLEMENTED USING LINKED LIST ");  queueLinkedList.remove(); // remove Node  queueLinkedList.displayStack();  queueLinkedList.remove(); // remove Node  queueLinkedList.displayStack();  queueLinkedList.remove(); // remove Node  queueLinkedList.displayStack();  queueLinkedList.remove(); // remove Node  queueLinkedList.displayStack();  }  }  **class** QueueLinkedList {  LinkedList13 linkedList = **new** LinkedList13(); // creation of Linked List  **public** **void** insert(**int** value) {  linkedList.insertLast(value);  }  **public** **void** remove() **throws** QueueEmptyException {  **try** {  linkedList.deleteFirst();  } **catch** (LinkedListEmptyException llee) {  **throw** **new** QueueEmptyException();  }  }  **public** **void** displayStack() {  System.***out***.print("Displaying Queue> Front to Rear: ");  linkedList.displayLinkedList();  }  }  **class** LinkedList13 {  **private** Node first; // ref to first link on list  **public** LinkedList13() {  first = **null**;  **public** **void** insertLast(**int** data) {  Node newNode = **new** Node (data); // Creation of New Node.  **if** (first == **null**) {  first = newNode; // first ---> newNode  **return**;  }  Node tempNode = first;  **while** (tempNode.next != **null**) {  tempNode = tempNode.next; // move to next Node.  }  tempNode.next = newNode; // make last's Node next point to new Node  }  **public** Node deleteFirst() {  **if** (first == **null**) {  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node tempNode = first;  first = first.next;  **return** tempNode;  }  **public** **void** displayLinkedList() {  Node tempDisplay = first; // start at the beginning of linkedList  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next; // move to next Node  }  System.***out***.println();  }  }  **class** QueueEmptyException **extends** RuntimeException {  **public** QueueEmptyException() {  **super**();  }  **public** QueueEmptyException(String message) {  **super**(message);  }  }  /\* OUTPUT  \* INSERTING AT LAST (REAR) IN QUEUE IMPLEMENTED USING LINKED LIST Displaying  \* Queue> Front to Rear: 11 Displaying Queue> Front to Rear: 11 71 Displaying  \* Queue> Front to Rear: 11 71 39 Displaying Queue> Front to Rear: 11 71 39 31  \* DELETING FROM FIRST (FRONT) OF QUEUE IMPLEMENTED USING LINKED LIST Displaying  \* Queue> Front to Rear: 71 39 31 Displaying Queue> Front to Rear: 39 31  \* Displaying Queue> Front to Rear: 31 Displaying Queue> Front to Rear: \*/ |

# **Remove first and last occurrence of s given element in the linkedlist**

|  |
| --- |
| **public** **class** Remove\_first\_and\_last\_occurrence {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("JAVA");  list.add("J2EE");  list.add("JSP");  list.add("J2EE");  list.add("JDBC");  System.***out***.println(list); // Output : [JAVA, J2EE, JSP, J2EE, JDBC]  list.removeFirstOccurrence("J2EE");  System.***out***.println(list); // Output : [JAVA, JSP, J2EE, JDBC]  list.removeLastOccurrence("J2EE");  System.***out***.println(list); // Output : [JAVA, JSP, JDBC]  }  } |

# **Remove All Elements from linkedlist**

|  |
| --- |
| **public** **class** RemoveAllElements {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> linkedList = **new** LinkedList<Integer>();  linkedList.add(10);  linkedList.add(20);  linkedList.add(30);  linkedList.add(40);  linkedList.add(50);  System.***out***.println(linkedList); // Output : [10, 20, 30, 40, 50]  linkedList.clear();  System.***out***.println(linkedList); // Output : []  }  } |

# **Remove Elements of linkedlist from both the ends**

|  |
| --- |
| **public** **class** RemoveElements {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("ONE");  list.add("TWO");  list.add("THREE");  list.add("FOUR");  list.add("FIVE");  list.add("SIX");  list.add("SEVEN");  System.***out***.println(list); // Output : [ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN]  list.poll();  list.pollFirst();  list.remove();  list.removeFirst();  System.***out***.println(list); // Output : [FIVE, SIX, SEVEN]  list.pollLast();  list.removeLast();  System.***out***.println(list); // Output : [FIVE]  }  } |

# **Replace Element at specified position of linkedlist with the given element.**

|  |
| --- |
| **public** **class** ReplaceElement {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("ONE");  list.add("TWO");  list.add("THREE");  list.add("FOUR");  System.***out***.println(list); // Output : [ONE, TWO, THREE, FOUR]  list.set(2, "ZERO");  System.***out***.println(list); // Output : [ONE, TWO, ZERO, FOUR]  }  } |

# **Retrieve but Not Remove the elements of linkedlist from both the ends.**

|  |
| --- |
| **public** **class** Retrieve\_But\_Not\_Remove {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("FIRST");  list.add("SECOND");  list.add("THIRD");  list.add("FOURTH");  list.add("FIFTH");  System.***out***.println(list); // Output : [FIRST, SECOND, THIRD, FOURTH, FIFTH]  System.***out***.println(list.element()); // Output : FIRST  System.***out***.println(list.getFirst()); // Output : FIRST  System.***out***.println(list.peek()); // Output : FIRST  System.***out***.println(list.peekFirst()); // Output : FIRST  System.***out***.println(list.peekLast()); // Output : FIFTH  System.***out***.println(list.getLast()); // Output : FIFTH  }  } |

# **Retrieve and Remove and only retrieve an element from specified position of a linkedlist**

|  |
| --- |
| **public** **class** RetrieveAndRemove {  **public** **static** **void** main(String[] args) {  LinkedList<String> list = **new** LinkedList<String>();  list.add("JAVA");  list.add("J2EE");  list.add("JSP");  list.add("SERVLETS");  list.add("JDBC");  System.***out***.println(list); // Output : [JAVA, J2EE, JSP, SERVLETS, JDBC]  System.***out***.println(list.remove(2)); // Output : JSP  System.***out***.println(list); // Output : [JAVA, J2EE, SERVLETS, JDBC]  System.***out***.println(list.get(2)); // Output : SERVLETS  }  } |

# **Reversing Singly LinkedList**

|  |
| --- |
| /\*Displaying LinkedList [first--->last]: 1 2 3 4  LinkedList has been reversed successfully.  Displaying LinkedList [first--->last]: 4 3 2 1\*/  **public** **class** ReversingSinglyLinkedList {  **public** **static** **void** main(String[] args) {  LinkedList6 linkedList = **new** LinkedList6();  linkedList.insertFirst(4);  linkedList.insertFirst(3);  linkedList.insertFirst(2);  linkedList.insertFirst(1);  linkedList.displayLinkedList();  linkedList.reverseLinkedList(); //REVERSE LinkedList  linkedList.displayLinkedList();  }  }  **class** LinkedList6 {  **private** Node first; // ref to first link on list  **public** LinkedList6(){  first = **null**;  }  **public** **void** reverseLinkedList() {  Node previousNode=**null**;  Node currentNode=first;  Node nextNode=first;  **while**(nextNode!=**null**){  nextNode =nextNode.next; //make nextNode point to next node.  currentNode.next=previousNode; //make current node's next point to previous node.  previousNode =currentNode; //make previousNode point to currentNode.  currentNode =nextNode; //make currentNode point to nextNode.  }  First =previousNode; //now make first point to previous node(i.e. last node).  System.***out***.println ("LinkedList has been reversed successfully.");  }  **public** **void** insertFirst(**int** data) {  Node newNode = **new** Node(data); //Creation of New Node.  newNode.next = first; //newLink ---> old first  first = newNode; //first ---> newNode  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList [first--->last]: ");  Node tempDisplay = first; // start at the beginning of linkedList  **while** (tempDisplay != **null**){ // Executes until we don't find end of list.  tempDisplay.displayNode();  tempDisplay = tempDisplay.next; // move to next Node  }  System.***out***.println();  }  } |

# **Singly LinkedList at First**

|  |
| --- |
| **public** **class** SinglyLinkedListatFirst {  **public** **static** **void** main(String[] args) {  LinkedList linkedList = **new** LinkedList(); // creation of Linked List  linkedList.insertFirst(11);  linkedList.insertFirst(21);  linkedList.insertFirst(59);  linkedList.insertFirst(14);  linkedList.insertFirst(39);  linkedList.displayLinkedList(); // display LinkedList  System.***out***.print("Deleted Nodes: ");  Node deletedNode = linkedList.deleteFirst(); // delete Node  deletedNode.displayNode (); // display deleted Node.  deletedNode = linkedList.deleteFirst(); // delete Node.  deletedNode.displayNode (); // display deleted Node.  System.***out***.println();// sysout used to format output  linkedList.displayLinkedList(); // Again display LinkedList  }  }  **class** LinkedList {  **private** Node first; // ref to first link on list  **public** LinkedList() {  first = **null**;  }  **public** **void** insertFirst(**int** data) {  Node newNode = **new** Node (data); // Creation of New Node.  newNode.next = first; // newLink ---> old first  first = newNode; // first ---> newNode  }  **public** Node deleteFirst() {  **if** (first == **null**) { // means LinkedList in empty, throw exception.  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node tempNode = first;  first = first.next;  **return** tempNode; // return tempNode (i.e. deleted Node)  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList [first--->last]: ");  Node tempDisplay = first; // start at the beginning of linkedList  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next; // move to next Node  }  System.***out***.println();  }  }  **class** Node {  **public** **int** data; // data in Node.  **public** Node next; // points to next Node in list.  **public** Node(**int** data) {  **this**.data = data;  }  **public** **void** displayNode() {  System.***out***.print(data + " ");  }  }  **class** LinkedListEmptyException **extends** RuntimeException {  **public** LinkedListEmptyException() {  **super**();  }  **public** LinkedListEmptyException(String message) {  **super**(message);  }  }  /\*Displaying LinkedList [first--->last]: 39 14 59 21 11  Deleted Nodes: 39 14  Displaying LinkedList [first--->last]: 59 21 11 \*/ |

# **Singly LinkedList at Last**

|  |
| --- |
| **public** **class** SinglyLinkedListatLast {  **public** **static** **void** main(String[] args) {  LinkedList1 linkedList = **new** LinkedList1(); // creation of Linked List  linkedList.insertLast(11);  linkedList.insertLast(21);  linkedList.insertLast(59);  linkedList.insertLast(14);  linkedList.insertLast(39);  linkedList.displayLinkedList(); // display LinkedList  System.***out***.print("Deleted Nodes: ");  Node1 deletedNode = linkedList.deleteLast(); // delete Node  deletedNode.displayNode (); // display deleted Node.  deletedNode = linkedList.deleteLast(); // delete Node  deletedNode.displayNode (); // display deleted Node.  System.***out***.println();// sysout used to format output  linkedList.displayLinkedList(); // Again display LinkedList  }  }  **class** LinkedList1 {  **private** Node1 first; // ref to first link on list  **public** LinkedList1() {  first = **null**;  }  **public** **void** insertLast(**int** data) {  Node1 newNode = **new** Node1(data); // Creation of New Node.  **if** (first == **null**) {  first = newNode; // first ---> newNode  **return**;  }  Node1 tempNode = first;  **while** (tempNode.next != **null**) {  tempNode = tempNode.next; // move to next Node.  }  tempNode.next = newNode; // make last's Node next point to new Node  }  **public** Node1 deleteLast() {  **if** (first == **null**) {  **throw** **new** LinkedListEmptyException1("LinkedList doesn't contain any Nodes.");  }  **if** (first.next == **null**) {  Node1 tempNode = first;  first = first.next;  **return** tempNode;  }  Node1 previous = **null**;  Node1 current = first;  **while** (current.next != **null**)  previous = current;  current = current.next;  }  previous.next = **null**;  **return** current;  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList [first--->last]: ");  Node1 tempDisplay = first;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println();  }  }  **class** Node1 {  **public** **int** data; // data in Node.  **public** Node1 next; // points to next Node in list.  **public** Node1(**int** data) {  **this**.data = data;  }  **public** **void** displayNode() {  System.***out***.print(data + " ");  }  }  **class** LinkedListEmptyException1 **extends** RuntimeException {  **public** LinkedListEmptyException1() {  **super**();  }  **public** LinkedListEmptyException1(String message) {  **super**(message);  }  }  /\*Displaying LinkedList [first--->last]: 11 21 59 14 39  Deleted Nodes: 39 14  Displaying LinkedList [first--->last]: 11 21 59 \*/ |

# **Singly LinkedList Circular or Not**

|  |
| --- |
| /\*Displaying LinkedList [first--->last]: 11 22 33 44 55 66 LinkedList is NOT CIRCULAR  LinkedList has been successfully converted into CircularLinkedList  Displaying LinkedList [first--->last]: 11 22 33 44 55 66 33 44 55 66 33 44 LinkedList is CIRCULAR\*/  **public** **class** SinglyLinkedListCircularOrNot {  **public** **static** **void** main(String[] args) {  LinkedList7 linkedList = **new** LinkedList7();  linkedList.insert(66);  linkedList.insert(55);  linkedList.insert(44);  linkedList.insert(33);  linkedList.insert(22);  linkedList.insert(11);  linkedList.displayLinkedList();  linkedList.findCircularOrNot();  System.***out***.println();  linkedList.makeLinkedListCircular();  linkedList.displayLinkedList();  linkedList.findCircularOrNot();  }  }  **class** LinkedList7 {  Node first = **null**;  Node circularPoint1;  Node circularPoint2;  Node slowPointer;  Node fastPointer;  **public** **void** insert(**int** data) {  Node newNode = **new** Node(data);  newNode.next = first;  first = newNode;  **if** (data == 33)  circularPoint1 = newNode;  **if** (data == 66)  circularPoint2 = newNode;  }  **public** **void** makeLinkedListCircular() {  circularPoint2.next = circularPoint1;  System.***out***.println("LinkedList has been successfully converted into CircularLinkedList");  }  **public** **void** findCircularOrNot() {  slowPointer = first;  fastPointer = first;  **while** ((slowPointer != fastPointer || slowPointer == first)  && fastPointer.next != **null**  && fastPointer.next.next != **null**) {  slowPointer = slowPointer.next;  fastPointer = fastPointer.next.next;  }  **if** (slowPointer == fastPointer)  System.***out***.println("LinkedList is CIRCULAR");  **else**  System.***out***.println("LinkedList is NOT CIRCULAR");  }  **public** **void** displayLinkedList() {  Node tempDisplay = first;  **int** displayLimiterCtr = 0;  System.***out***.print("Displaying LinkedList [first--->last]: ");  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  **if** (++displayLimiterCtr >= 12)  **break**;  }  System.***out***.println();  }  } |

# **Singly LinkedList Delete Specific Node**

|  |
| --- |
| /\*Displaying LinkedList [first--->last]: 29 19 20 92. Node with data=29 was found on first and has been deleted. Node with data=11 wasn't found for deletion. Displaying LinkedList [first--->last]: 19 20 92 \*/  **public** **class** SinglyLinkedListDeleteSpecificNode {  **public** **static** **void** main(String[] args) {  LinkedList3 linkedList = **new** LinkedList3();  linkedList.insertFirst(92);  linkedList.insertFirst(20);  linkedList.insertFirst(19);  linkedList.insertFirst(29);  linkedList.displayLinkedList();  linkedList.deleteSpecificNode(29);  linkedList.deleteSpecificNode(11);  linkedList.displayLinkedList();  }  }  **class** LinkedList3 {  **private** Node first;  **public** LinkedList3() {  first = **null**;  }  **public** **void** insertFirst(**int** data) {  Node newNode = **new** Node(data);  newNode.next = first;  first = newNode;  }  **public** Node deleteSpecificNode(**int** deleteKey) {  **if** (first == **null**) {  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  **if** (first.data == deleteKey) {  Node tempNode = first;  first = first.next;  System.***out***.println ("Node with data=" + tempNode.data + “was found on first and has been deleted.");  **return** tempNode;  }  Node previous = **null**;  Node current = first;  **while** (current != **null**) {  **if** (current.data == deleteKey) {  System.***out***.println ("Node with data=" + current.data + “has been deleted.");  previous.next = current.next;  **return** current;  } **else** {  **if** (current.next == **null**) {  System.***out***.println ("Node with data=" + deleteKey + “wasn’t found for deletion.");  **return** **null**;  }  previous = current;  current = current.next;  }  }  **return** **null**;  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList [first--->last]: ");  Node tempDisplay = first;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println();  }  } |

# **Singly LinkedList Generic Example**

|  |
| --- |
| /\*Displaying LinkedList [first--->last]: Employee [id=39, name=pat] Employee [id=14, name=sai] Employee [id=59, name=katy] Employee [id=21, name=amy] Employee [id=11, name=sam]  Deleted Nodes: Employee [id=39, name=pat] Employee [id=14, name=sai]  Displaying LinkedList [first--->last]: Employee [id=59, name=katy] Employee [id=21, name=amy] Employee [id=11, name=sam] \*/  **public** **class** SinglyLinkedListGenericExample {  **public** **static** **void** main(String[] args) {  LinkedList11<Employee11> linkedList = **new** LinkedList11<Employee11>();  linkedList.insertFirst(**new** Employee11("11", "sam"));  linkedList.insertFirst(**new** Employee11("21", "amy"));  linkedList.insertFirst(**new** Employee11("59", "katy"));  linkedList.insertFirst(**new** Employee11("14", "sai"));  linkedList.insertFirst(**new** Employee11("39", "pat"));  linkedList.displayLinkedList();  System.***out***.print("Deleted Nodes: ");  Node11<Employee11> deletedNode = linkedList.deleteFirst();  deletedNode.displayNode();  deletedNode = linkedList.deleteFirst();  deletedNode.displayNode();  System.***out***.println();  linkedList.displayLinkedList();  }  }  **class** LinkedList11<T> {  **private** Node11<T> first;  **public** LinkedList11() {  first = **null**;  }  **public** **void** insertFirst(T data) {  Node11<T> newNode = **new** Node11<T>(data);  newNode.next = first;  first = newNode;  }  **public** Node11<T> deleteFirst() {  **if** (first == **null**) {  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node11<T> tempNode = first;  first = first.next;  **return** tempNode;  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList [first--->last]: ");  Node11<T> tempDisplay = first;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println();  }  }  **class** Node11<T> {  **public** T data;  **public** Node11<T> next;  **public** Node11(T data) {  **this**.data = data;  }  **public** **void** displayNode() {  System.***out***.print(data + " ");  }  }  **class** Employee11 {  **private** String id;  **private** String name;  **public** Employee11(String id, String name) {  **this**.id = id;  **this**.name = name;  }  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", name=" + name + "] ";  }  } |

# **Singly LinkedList Insert Node In Between**

|  |
| --- |
| /\*Displaying LinkedList (first--->last):  Node with data=92 inserted at first.  Node with data=20 inserted at first Node, because newNode is smallest of existing Nodes.  Node with data=19 inserted at first Node, because newNode is smallest of existing Nodes.  Node with data=29 inserted successfully in middle of LinkedList.  Node with data=99 inserted successfully at last of LinkedList.  Displaying LinkedList (first--->last): 19 20 29 92 99 \*/  **public** **class** SinglyLinkedListInsertNodeInBetween {  **public** **static** **void** main(String[] args) {  LinkedList2 linkedList = **new** LinkedList2();  linkedList.displayLinkedList();  linkedList.insertNodeInSortedLinkedList(92);  linkedList.insertNodeInSortedLinkedList(20);  linkedList.insertNodeInSortedLinkedList(19);  linkedList.insertNodeInSortedLinkedList(29);  linkedList.insertNodeInSortedLinkedList(99);  linkedList.displayLinkedList();  }  }  **class** LinkedList2 {  **private** Node first;  **public** LinkedList2() {  first = **null**;  }  **public** **void** insertNodeInSortedLinkedList(**int** data) {  Node newNode = **new** Node(data);  **if** (first == **null**) {  first = newNode;  System.***out***.println ("Node with data=" + newNode.Data + “inserted at first.");  **return**;  }  **if** (first.data >= newNode. Data) {  newNode.next = first;  first = newNode;  System.***out***.println("Node with data=" + newNode.data  + “inserted at first Node, because newNode is smallest of existing Nodes.");  **return**;  }  Node current = first;  Node previous = **null**;  **while** (current != **null**) {  **if** (current.data < newNode.data) {  **if** (current.next == **null**) {  current.next = newNode;  System.***out***.println(  "Node with data=" + newNode.data + “inserted successfully at last of LinkedList.");  **return**;  }  previous = current;  current = current.next;  } **else** {  newNode.next = previous.next;  previous.next = newNode;  System.**out**.println ("Node with data=" + newNode.data + “inserted successfully in middle of LinkedList.");  **return**;  }  }  }  **public** **void** displayLinkedList() {  System.***out***.print("Displaying LinkedList (first--->last): ");  Node tempDisplay = first;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println();  }  } |

# **Sorted Doubly LinkedList**

|  |
| --- |
| **public** **class** SortedDoublyLinkedList {  **public** **static** **void** main(String[] args) {  LinkedList5 linkedList = **new** LinkedList5();  linkedList.insertSorted(11);  linkedList.insertSorted(21);  linkedList.insertSorted(59);  linkedList.insertSorted(14);  linkedList.insertSorted(39);  linkedList.insertSorted(66);  linkedList.insertSorted(33);  linkedList.displayFrwd();  linkedList.displayBckwrd();  System.***out***.println();  linkedList.deleteSpecificNode(11);  linkedList.deleteSpecificNode(21);  linkedList.deleteSpecificNode(29);  linkedList.deleteSpecificNode(59);  linkedList.displayFrwd();  linkedList.displayBckwrd();  }  }  **class** LinkedList5 {  **private** Node4 first;  **private** Node4 last;  **public** LinkedList5() {  first = **null**;  }  **public** **void** insertFirst(**int** data) {  Node4 newNode = **new** Node4(data);  **if** (first == **null**)  last = newNode;  **else**  first.previous = newNode;  newNode.next = first;  first = newNode;  }  **public** Node4 deleteFirst() {  Node4 tempNode = first;  **if** (first.next == **null**)  last = **null**;  **else**  first.next.previous = **null**;  first = first.next;  **return** tempNode;  }  **public** **void** insertSorted(**int** newKey) {  Node4 newNode = **new** Node4(newKey);  **if** (first == **null**) {  first = last = newNode;  System.***out***.println ("Node with data=" + newNode.data + “inserted at first.");  **return**;  }  Node4 current = first;  **if** (current.data >= newNode.data) {  newNode.next = first;  first.previous = newNode;  first = newNode;  System.***out***.println("Node with data=" + newNode.data  + “inserted at first Node, because newNode is smallest of existing Nodes.");  **return**;  }  **while** (**true**) {  **if** (newNode.data > current.data) {  **if** (current.next == **null**) {  last.next = newNode;  newNode.previous = last;  last = newNode;  System.***out***.println(  "Node with data=" + newNode.data + “inserted successfully at last of LinkedList.");  **return**;  }  current = current.next;  } **else** {  current = current.previous;  newNode.next = current.next;  current.next.previous = newNode;  newNode.previous = current;  current.next = newNode;  System.***out***.println ("Node with data=" + newNode.data + “inserted successfully in middle of LinkedList.");  **return**;  }  }  }  **public** **void** deleteSpecificNode(**int** deleteKey) {  **if** (first == **null**) {  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node4 current = first;  **while** (current.data != deleteKey) {  **if** (current.next == **null**) {  System.***out***.println ("Node with data=" + deleteKey + “wasn’t found for deletion.");  **return**;  }  current = current.next;  }  **if** (current == first) {  System.***out***.println ("Node with data=" + current.data + “was found on first and has been deleted.");  first = first.next;  first.previous = **null**;  } **else** **if** (current == last) {  System.***out***.println ("Node with data=" + current.data + " was found on last has been deleted.");  last = last.previous;  last.next = **null**;  } **else** {  current.previous.next = current.next;  current.next.previous = current.previous;  System.***out***.println ("Node with data=" + current.data + “has been deleted.");  }  }  **public** **void** displayFrwd() {  System.***out***.print("Displaying in forward direction [first--->last] : ");  Node4 tempDisplay = first;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println("");  }  **public** **void** displayBckwrd() {  System.***out***.print("Displaying in backward direction [last-->first] : ");  Node4 tempDisplay = last;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.previous;  }  System.***out***.println("");  }  } |

# **Stack Example Generic**

|  |
| --- |
| /\* OUTPUT Popped items: Employee [id=11, name=sam] Employee [id=44, name=sam] Employee  \* [id=33, name=sam] Employee [id=22, name=sam] Employee [id=11, name=sam] \*/  **public** **class** StackExampleGeneric {  **public** **static** **void** main(String[] args) {  Stack<Employee> stack = **new** Stack<Employee>(10);  stack.push(**new** Employee("11", "sam"));  stack.push(**new** Employee("22", "sam"));  stack.push(**new** Employee("33", "sam"));  stack.push(**new** Employee("44", "sam"));  stack.push(**new** Employee("11", "sam"));  System.***out***.print("Popped items: ");  System.***out***.print(stack.pop() + " ");  System.***out***.print(stack.pop() + " ");  System.***out***.print(stack.pop() + " ");  System.***out***.print(stack.pop() + " ");  System.***out***.print(stack.pop() + " ");  }  }  **class** Employee {  **private** String id;  **private** String name;  **public** Employee(String id, String name) {  **this**.id = id;  **this**.name = name;  }  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", name=" + name + "] ";  }  }  **class** StackFullException **extends** RuntimeException {  **public** StackFullException() {  **super**();  }  **public** StackFullException(String message) {  **super**(message);  }  }  **class** StackEmptyException **extends** RuntimeException {  **public** StackEmptyException() {  **super**();  }  **public** StackEmptyException(String message) {  **super**(message);  }  }  **class** Stack<T> {  **private** **int** size;  **private** T[] stackAr;  **private** **int** top;  @SuppressWarnings("unchecked")  **public** Stack(**int** size) {  **this**.size = size;  stackAr = (T[]) **new** Object[size];  top = -1;  }  **public** **void** push(T value) {  **if** (isFull()) {  **throw** **new** StackFullException("Cannot push " + value + ", Stack is full");  }  stackAr[++top] = value;  }  **public** T pop() {  **if** (isEmpty()) {  **throw** **new** StackEmptyException("Stack is empty");  }  **return** stackAr[top--];  }  **public** **boolean** isEmpty() {  **return** (top == -1);  }  **public** **boolean** isFull() {  **return** (top == size - 1);  }  } |

# **Stack Implementation Using LinkedList**

|  |
| --- |
| **public** **class** StackImplementationUsingLinkedList {  **public** **static** **void** main(String[] args) {  StackLinkedList stackLinkedList = **new** StackLinkedList();  System.***out***.println("INSERTING AT FIRST (TOP) IN STACK IMPLEMENTED USING LINKED LIST ");  stackLinkedList.push(39);  stackLinkedList.displayStack();  stackLinkedList.push(71);  stackLinkedList.displayStack();  stackLinkedList.push(11);  stackLinkedList.displayStack();  stackLinkedList.push(76);  stackLinkedList.displayStack();  System.***out***.println("\deleting FROM FIRST (TOP) FROM STACK IMPLEMENTED USING LINKED LIST ");  stackLinkedList.pop();  stackLinkedList.displayStack();  stackLinkedList.pop();  stackLinkedList.displayStack();  stackLinkedList.pop();  stackLinkedList.displayStack();  stackLinkedList.pop();  stackLinkedList.displayStack();  }  }  **class** StackLinkedList {  LinkedList12 linkedList = **new** LinkedList12();  **public** **void** push(**int** value) {  linkedList.insertFirst(value);  }  **public** **void** pop() **throws** StackEmptyException {  **try** {  linkedList.deleteFirst();  } **catch** (LinkedListEmptyException llee) {  **throw** **new** StackEmptyException();  }  }  **public** **void** displayStack() {  System.***out***.print("Displaying Stack > Top to Bottom : ");  linkedList.displayLinkedList();  }  }  **class** LinkedList12 {  **private** Node first;  **public** LinkedList12() {  first = **null**;  }  **public** **void** insertFirst(**int** data) {  Node newNode = **new** Node(data);  newNode.next = first;  first = newNode;  }  **public** Node deleteFirst() {  **if** (first == **null**) {  **throw** **new** LinkedListEmptyException("LinkedList doesn't contain any Nodes.");  }  Node tempNode = first;  first = first.next;  **return** tempNode;  }  **public** **void** displayLinkedList() {  Node tempDisplay = first;  **while** (tempDisplay != **null**) {  tempDisplay.displayNode();  tempDisplay = tempDisplay.next;  }  System.***out***.println();  }  } |

# **ArrayList Custom App**

|  |
| --- |
| /\*implementing custom ArrayList in java Displaying list : 1 2 3 4 1 2  element at index in custom ArrayList > 1 = 2  element removed from index 1 = 2  let's display custom ArrayList again after removal at index 1 Displaying list : 1 3 4 1 2 \*/  **public** **class** ArrayListCustomApp {  **public** **static** **void** main(String[] args) {  ArrayListCustom<Integer> list = **new** ArrayListCustom<Integer>();  list.add(1);  list.add(2);  list.add(3);  list.add(4);  list.add(1);  list.add(2);  list.display();  System.***out***.println("\element at index in custom ArrayList > " + 1 + " = " + list.get(1));  System.***out***.println("element removed from index " + 1 + " = " + list.remove(1));  System.***out***.println("\let’s display custom ArrayList again after removal at index 1");  list.display();  }  }  **class** ArrayListCustom<E> {  **private** **static** **final** **int** ***INITIAL\_CAPACITY*** = 10;  **private** **int** size = 0;  **private** Object elementData[] = {};  **public** ArrayListCustom() {  elementData = **new** Object[***INITIAL\_CAPACITY***];  }  **public** **void** add(E e) {  **if** (size == elementData.length) {  ensureCapacity();  }  elementData[size++] = e;  }  @SuppressWarnings("unchecked")  **public** E get(**int** index) {  **if** (index < 0 || index >= size) {  **throw** **new** IndexOutOfBoundsException("Index: " + index + ", Size " + index);  }  **return** (E) elementData[index];  }  **public** Object remove(**int** index) {  **if** (index < 0 || index >= size) {  **throw** **new** IndexOutOfBoundsException("Index: " + index + ", Size " + index);  }  Object removedElement = elementData[index];  **for** (**int** i = index; i < size - 1; i++) {  elementData[i] = elementData[i + 1];  }  size--;  **return** removedElement;  }  **private** **void** ensureCapacity() {  **int** newIncreasedCapacity = elementData.length \* 2;  elementData = Arrays.*copyOf*(elementData, newIncreasedCapacity);  }  **public** **void** display() {  System.***out***.print("Displaying list : ");  **for** (**int** i = 0; i < size; i++) {  System.***out***.print(elementData[i] + " ");  }  }  } |

# **ArrayList Synchronization Fails Example**

|  |
| --- |
| /\*OUTPUT Exception in thread "thread-2" java.util.ConcurrentModificationException \*/  public class ArrayListSynchronizationFailsExample {  public static void main(String[] args) {  final List<Integer> arrayList = new ArrayList<Integer>();  final List<Integer> synchronizedArrayList;  synchronizedArrayList = Collections.synchronizedList(arrayList);  Thread t1 = new Thread(new Runnable() {  public void run() {  for (int i = 0; i <= 3; i++) {  synchronizedArrayList.add(i);  try {  Thread.sleep(100);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  }, "thread-1");  t1.start();  Thread t2 = new Thread(new Runnable() {  public void run() {  Iterator<Integer> it = synchronizedArrayList.iterator();  while (it.hasNext()) {  try {  Thread.sleep(100);  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println(it.next());  }  }  }, "thread-2");  t2.start();  }  } |

# **ArrayList Synchronization Succeeds Example**

|  |
| --- |
| /\*output 0 \*/  **public** **class** ArrayListSynchronizationSucceedsExample {  **public** **static** **void** main(String[] args) {  **final** List<Integer> arrayList = **new** ArrayList<Integer>();  **final** List<Integer> synchronizedArrayList;  synchronizedArrayList = Collections.*synchronizedList*(arrayList);  Thread t1 = **new** Thread(**new** Runnable() {  **public** **void** run() {  **for** (**int** i = 0; i <= 3; i++) {  synchronizedArrayList.add(i);  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }, "thread-1");  t1.start();  Thread t2 = **new** Thread(**new** Runnable() {  **public** **void** run() {  Iterator<Integer> it = synchronizedArrayList.iterator();  **synchronized** (synchronizedArrayList) {  **while** (it.hasNext()) {  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println(it.next());  }  }  }  }, "thread-2");  t2.start();  }  } |

# **Arrays Sort: -** Program to understand comparatoe of superclass can be used by subclasses

|  |
| --- |
| **public** **class** ArraysSort {  **public** **static** **void** main(String[] args) {  System.***out***.println("-----Sorting Lion's height -- using AnimalComparator-----");  Lion lion1 = **new** Lion(3);  Lion lion2 = **new** Lion(1);  Lion lion3 = **new** Lion(2);  Lion lionArray[] = { lion1, lion2, lion3 };  System.***out***.print(" lionArray before sorting : \n");  **for** (Animal animal : lionArray) {  System.***out***.print(animal + " ");  }  Arrays.*sort*(lionArray, **new** AnimalComparator());  System.***out***.println("\n\n lionArray after sorting on basis of " + "height(ascending order) : ");  **for** (Animal animal : lionArray) {  System.***out***.print(animal + " ");  }  System.***out***.println("\n\n\n------Sorting Dog's height -- using AnimalComparator----");  Dog dog1 = **new** Dog(2);  Dog dog2 = **new** Dog(1);  Dog dog3 = **new** Dog(3);  Dog dogArray[] = { dog1, dog2, dog3 };  System.***out***.print(" dogArray before sorting : \n");  **for** (Animal animal : dogArray) {  System.***out***.print(animal + " ");  }  Arrays.*sort*(dogArray, **new** AnimalComparator());  System.***out***.println("\n\n dogArray after sorting on basis of " + "height(ascending order) : ");  **for** (Animal animal : dogArray) {  System.***out***.print(animal + " ");  }  }  }  **class** Animal {  Integer height;  }  **class** Lion **extends** Animal {  Lion(Integer height) {  **this**.height = height;  }  @Override  **public** String toString() {  **return** "Lion [height=" + height + "]";  }  }  **class** Dog **extends** Animal {  Dog(Integer height) {  **this**.height = height;  }  @Override  **public** String toString() {  **return** "Dog [height=" + height + "]";  }  }  **class** AnimalComparator **implements** Comparator<Animal> {  @Override  **public** **int** compare(Animal o1, Animal o2) {  **return** o1.height.compareTo(o2.height);  }  } |

# **Concurrent HashMap Example: -** program to use put, get, containsKey, containsValue, remove, size methods

|  |
| --- |
| **public** **class** ConcurrentHashMapExample {  **public** **static** **void** main(String[] args) {  System.***out***.println ("-----1. create Map---");  Map<Integer, String> concurrentHashMap = **new** ConcurrentHashMap<Integer, String>();  System.***out***.println ("\n-----2. put key-value pair in Map ---");  concurrentHashMap.put(11, "ankit");  concurrentHashMap.put(21, "mittal");  concurrentHashMap.put(31, "javaMadeSoEasy");  System.***out***.println ("\n-----3. get method returns value corresponding to key ---");  System.***out***.println("concurrentHashMap.get(11) : " + concurrentHashMap.get(11));  System.***out***.println(  "\n-----4. containsKey method returns true if map " + "contains a mapping for the specified key ---"); System.***out***.println("concurrentHashMap.containsKey(11) : " + concurrentHashMap.containsKey(11));  System.***out***.println ("\n-----5. containsValue method returns true if map maps one "  + "or more keys to the specified value. ---");  System.***out***.println("concurrentHashMap.containsValue(\"ankit\") : " + concurrentHashMap.containsValue("ankit"));  System.***out***.println("concurrentHashMap.remove(11) : " + concurrentHashMap.remove(11));  System.***out***.println ("\n-----6. size ---");  System.***out***.println("concurrentHashMap.size() : " + concurrentHashMap.size());  }  } |

# **Concurrent Skip List Map Example: -** program to use put, get, containsKey, containsValue, remove, size methods

|  |
| --- |
| **public** **class** ConcurrentSkipListMapExample {  **public** **static** **void** main(String[] args) {  System.***out***.println ("-----1. create Map---");  Map<Integer, String> concurrentSkipListMap = **new** ConcurrentSkipListMap<Integer, String>();  System.***out***.println ("\n-----2. put key-value pair in Map ---");  concurrentSkipListMap.put(11, "ankit");  concurrentSkipListMap.put(21, "mittal");  concurrentSkipListMap.put(31, "javaMadeSoEasy");  System.***out***.println ("\n-----3. get method returns value corresponding to key ---");  System.***out***.println("concurrentSkipListMap.get(11) : " + concurrentSkipListMap.get(11));  System.***out***.println(  "\n-----4. containsKey method returns true if map " + "contains a mapping for the specified key ---");  System.***out***.println("concurrentSkipListMap.containsKey(11) : " + concurrentSkipListMap.containsKey(11));  System.***out***.println ("\n-----5. containsValue method returns true if map maps one "  + "or more keys to the specified value. ---");  System.***out***.println(  "concurrentSkipListMap.containsValue(\"ankit\") : " + concurrentSkipListMap.containsValue("ankit"));  System.***out***.println("concurrentSkipListMap.remove(11) : " + concurrentSkipListMap.remove(11));  System.***out***.println ("\n-----6. size ---");  System.***out***.println("concurrentSkipListMap.size() : " + concurrentSkipListMap.size());  }  } |

# **Concurrent Skip List Map Example to make map unmodifiable**

|  |
| --- |
| **public** **class** ConcurrentSkipListMapUnmodifiableExample {  **public** **static** **void** main(String[] args) {  Map<Integer, String> concurrentSkipListMap = **new** ConcurrentSkipListMap<Integer, String>();  concurrentSkipListMap.put(11, "ankit");  concurrentSkipListMap.put(21, "mittal");  concurrentSkipListMap.put(31, "javaMadeSoEasy");  Map<Integer, String> unmodifiableMap = Collections.*unmodifiableMap*(concurrentSkipListMap);  unmodifiableMap.put(41, "java");  }  }  /\*OUTPUT Exception in thread "main" java.lang.UnsupportedOperationException \*/ |

# **CopyOnWriteArrayList Synchronization Succeeds Example**

|  |
| --- |
| /\*output 0 \*/  **public** **class** CopyOnWriteArrayListSynchronizationSucceedsExample {  **public** **static** **void** main(String[] args) {  **final** List<Integer> copyOnWriteArrayList = **new** CopyOnWriteArrayList<Integer>();  Thread t1 = **new** Thread(**new** Runnable() {  **public** **void** run() {  **for** (**int** i = 0; i <= 3; i++) {  copyOnWriteArrayList.add(i);  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  }, "thread-1");  t1.start();  Thread t2 = **new** Thread(**new** Runnable() {  **public** **void** run() {  Iterator<Integer> it = copyOnWriteArrayList.iterator();  **while** (it.hasNext()) {  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println(it.next());  }  }  }, "thread-2");  t2.start();  }  } |

# **HashCode\_and\_equal\_1:-** create a map which contains address of each employee and uses employee object as a key. Store address of some employees in this map. Now create a method that accepts map and employee object as parameters and return address of this employee. **Approach 1 (without overriding hashcode and equals method).**

|  |
| --- |
| **public** **class** HashCode\_and\_equal\_1 {  **public** **static** **void** main(String[] args) {  Employee emp1 = **new** Employee(110, "Sajid Ali Khan");  Address adrs1 = **new** Address(304, "Marol Mahrisi", "Mumbai", 400069);  Employee emp2 = **new** Employee(111, "Jaspreet Singh");  Address adrs2 = **new** Address(203, "Seepz", "Mumbai", 400093);  Map<Employee, Address> map = **new** HashMap<>();  map.put(emp1, adrs1);  map.put(emp2, adrs2);  System.***out***.println(HashCode\_and\_equal\_1.*getAddress*(map, **new** Employee(110, "Sajid Ali Khan")));  }  **public** **static** String getAddress(Map<Employee, Address> map, Employee emp) {  Address adrs = map.get(emp);  **return** adrs.getAddress();  }  }  **class** Employee {  **private** **int** empId;  **private** String name;  **public** Employee(**int** empId, String name) {  **this**.empId = empId;  **this**.name = name;  }  }  **class** Address {  **private** **int** houseNo;  **private** String streetName;  **private** String city;  **private** **int** pinCode;  **public** Address(**int** houseNo, String streetName, String city, **int** pinCode) {  **this**.houseNo = houseNo;  **this**.streetName = streetName;  **this**.city = city;  **this**.pinCode = pinCode;  }  **public** String getAddress() {  **return** houseNo + ", " + streetName + ", " + city + ", " + pinCode;  }  } |

# **HashCode\_and\_equal\_2 Approach2 (Overriding hashcode and equals method)**

|  |
| --- |
| **public** **class** HashCode\_and\_equal\_2 {  **public** **static** **void** main(String[] args) {  Employee1 emp1 = **new** Employee1(110, "Sajid Ali Khan");  Address1 adrs1 = **new** Address1(304, "Marol Mahrisi", "Mumbai", 400069);  Employee1 emp2 = **new** Employee1(111, "Jaspreet Singh");  Address1 adrs2 = **new** Address1(203, "Seepz", "Mumbai", 400093);  Map<Employee1, Address1> map = **new** HashMap<>();  map.put(emp1, adrs1);  map.put(emp2, adrs2);  System.***out***.println(HashCode\_and\_equal\_2.*getAddress1*(map, **new** Employee1(110, "Sajid Ali Khan")));  }  **public** **static** String getAddress1(Map<Employee1, Address1> map, Employee1 emp) {  Address1 adrs = map.get(emp);  **return** adrs.getAddress1();  }  }  **class** Employee1 {  **private** **int** empId;  **private** String name;  **public** Employee1(**int** empId, String name) {  **this**.empId = empId;  **this**.name = name;  }  @Override  **public** **int** hashCode() {  **final** **int** prime = 31;  **int** result = 1;  result = prime \* result + empId;  result = prime \* result + ((name == **null**) ? 0 : name.hashCode());  **return** result;  }  @Override  **public** **boolean** equals(Object obj) {  **if** (**this** == obj) **return** **true**;  **if** (obj == **null**) **return** **false**;  **if** (getClass() != obj.getClass()) **return** **false**;  Employee1 other = (Employee1) obj;  **if** (empId != other.empId) **return** **false**;  **if** (name == **null**) {  **if** (other.name != **null**)  **return** **false**;  } **else** **if** (!name.equals(other.name))  **return** **false**;  **return** **true**;  }  }  **class** Address1 {  **private** **int** houseNo;  **private** String streetName;  **private** String city;  **private** **int** pinCode;  **public** Address1(**int** houseNo, String streetName, String city, **int** pinCode) {  **this**.houseNo = houseNo;  **this**.streetName = streetName;  **this**.city = city;  **this**.pinCode = pinCode;  }  **public** String getAddress1() {  **return** houseNo + ", " + streetName + ", " + city + ", " + pinCode;  }  } |

# **HashMap Custom App**

|  |
| --- |
| //implementing custom HashMap in java  value corresponding to key 21=12  value corresponding to key 51=null  Displaying : {21=12} {25=121} {33=15} {30=151} {35=89}  value corresponding to key 21 removed: true  value corresponding to key 51 removed: false  Displaying : {25=121} {33=15} {30=151} {35=89}\*/  **public** **class** HashMapCustomApp {  **public** **static** **void** main(String[] args) {  HashMapCustom<Integer, Integer> hashMapCustom = **new** HashMapCustom<Integer, Integer>();  hashMapCustom.put(21, 12);  hashMapCustom.put(25, 121);  hashMapCustom.put(30, 151);  hashMapCustom.put(33, 15);  hashMapCustom.put(35, 89);  System.***out***.println("value corresponding to key 21=" + hashMapCustom.get(21));  System.***out***.println("value corresponding to key 51=" + hashMapCustom.get(51));  System.***out***.print("Displaying : ");  hashMapCustom.display();  System.***out***.println("\n\nvalue corresponding to key 21 removed: " + hashMapCustom.remove(21));  System.***out***.println("value corresponding to key 51 removed: " + hashMapCustom.remove(51));  System.***out***.print("Displaying : ");  hashMapCustom.display();  }  }  **class** HashMapCustom<K, V> {  **private** Entry<K, V>[] table;  **private** **int** capacity = 4;  **static** **class** Entry<K, V> {  K key;  V value;  Entry<K, V> next;  **public** Entry(K key, V value, Entry<K, V> next) {  **this**.key = key;  **this**.value = value;  **this**.next = next;  }  }  @SuppressWarnings("unchecked")  **public** HashMapCustom() {  table = **new** Entry[capacity];  }  **public** **void** put(K newKey, V data) {  **if** (newKey == **null**)  **return**;  **int** hash = hash(newKey);  Entry<K, V> newEntry = **new** Entry<K, V>(newKey, data, **null**);  **if** (table[hash] == **null**) {  table[hash] = newEntry;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(newKey)) {  **if** (previous == **null**) {  newEntry.next = current.next;  table[hash] = newEntry;  **return**;  } **else** {  newEntry.next = current.next;  previous.next = newEntry;  **return**;  }  }  previous = current;  current = current.next;  }  previous.next = newEntry;  }  }  **public** V get(K key) {  **int** hash = hash(key);  **if** (table[hash] == **null**) {  **return** **null**;  } **else** {  Entry<K, V> temp = table[hash];  **while** (temp != **null**) {  **if** (temp.key.equals(key))  **return** temp.value;  temp = temp.next;  }  **return** **null**;  }  }  **public** **boolean** remove(K deleteKey) {  **int** hash = hash(deleteKey);  **if** (table[hash] == **null**) {  **return** **false**;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(deleteKey)) {  **if** (previous == **null**) {  table[hash] = table[hash].next;  **return** **true**;  } **else** {  previous.next = current.next;  **return** **true**;  }  }  previous = current;  current = current.next;  }  **return** **false**;  }  }  **public** **void** display() {  **for** (**int** i = 0; i < capacity; i++) {  **if** (table[i] != **null**) {  Entry<K, V> entry = table[i];  **while** (entry != **null**) {  System.***out***.print("{" + entry.key + "=" + entry.value + "}" + " ");  entry = entry.next;  }  }  }  }  **private** **int** hash(K key) {  **return** Math.*abs*(key.hashCode()) % capacity;  }  } |

# **HashMap Test: -** program to create method that provides functionality smiliar to putIfAbsent () method of ConcurrentHashMap and to be used with HashMap.

|  |
| --- |
| hashMap : {1=javaMadeSoEasy}  functionalityOfPutIfAbsent method >> javaMadeSoEasy hashMap : {1=javaMadeSoEasy}  functionalityOfPutIfAbsent method >> null hashMap : {1=javaMadeSoEasy, 2=audi}\*/  **public** **class** HashMapTest {  **static** Map<Integer, String> *map* = **new** HashMap<Integer, String>();  **public** **static** **void** main(String[] args) {  *map*.put(1, "javaMadeSoEasy");  System.***out***.println("hashMap : " + *map*);  System.***out***.println("\n functionalityOfPutIfAbsent method >> " + *functionalityOfPutIfAbsent*(1, "ankit")); System.***out***.println("hashMap : " + *map*);  System.***out***.println("\n functionalityOfPutIfAbsent method >> " + *functionalityOfPutIfAbsent*(2, "audi")); System.***out***.println("hashMap : " + *map*);  }  **public** **static** **synchronized** String functionalityOfPutIfAbsent(Integer key, String value) {  **if** (!*map*.containsKey(key))  **return** *map*.put(key, value);  **else**  **return** *map*.get(key);  }  } |

# **HashMap to ArrayList**

|  |
| --- |
| **public** **class** HashMapToArrayList {  **public** **static** **void** main(String[] args) {  HashMap<String, String> studentPerformanceMap = **new** HashMap<String, String>();  studentPerformanceMap.put("John Kevin", "Average");  studentPerformanceMap.put("Rakesh Sharma", "Good");  studentPerformanceMap.put("Prachi D", "Very Good");  studentPerformanceMap.put("Ivan Jose", "Very Bad");  studentPerformanceMap.put("Smith Jacob", "Very Good");  studentPerformanceMap.put("Anjali N", "Bad");  // HashMap Keys Into ArrayList  Set<String> keySet = studentPerformanceMap.keySet();  ArrayList<String> listOfKeys = **new** ArrayList<String>(keySet);  System.***out***.println("ArrayList Of Keys :");  **for** (String key : listOfKeys) {  System.***out***.println(key);  }  System.***out***.println("--------------------------");  // HashMap Values Into ArrayList  Collection<String> values = studentPerformanceMap.values();  ArrayList<String> listOfValues = **new** ArrayList<String>(values);  System.***out***.println("ArrayList Of Values :");  **for** (String value : listOfValues) {  System.***out***.println(value);  }  System.***out***.println("--------------------------");  // HashMap’s Key-Value Pairs Into ArrayList  Set<Entry<String, String>> entrySet = studentPerformanceMap.entrySet();  ArrayList<Entry<String, String>> listOfEntry = **new** ArrayList<Entry<String, String>>(entrySet);  System.***out***.println("ArrayList of Key-Values :");  **for** (Entry<String, String> entry : listOfEntry) {  System.***out***.println(entry.getKey() + " : " + entry.getValue());  }  }  } |

# **HashSet Custom App**

|  |
| --- |
| HashSetCustom contains 21 =true  HashSetCustom contains 51 =false  Displaying HashSetCustom: 21 25 33 30 35  21 removed: true  22 removed: false  Displaying HashSetCustom: 25 33 30 35 \*/  **public** **class** HashSetCustomApp {  **public** **static** **void** main(String[] args) {  HashSetCustom<Integer> hashSetCustom = **new** HashSetCustom<Integer>();  hashSetCustom.add(21);  hashSetCustom.add(25);  hashSetCustom.add(30);  hashSetCustom.add(33);  hashSetCustom.add(35);  System.***out***.println("HashSetCustom contains 21 =" + hashSetCustom.contains(21));  System.***out***.println("HashSetCustom contains 51 =" + hashSetCustom.contains(51));  System.***out***.print("Displaying HashSetCustom: ");  hashSetCustom.display();  System.***out***.println("\n\n21 removed: " + hashSetCustom.remove(21));  System.***out***.println("22 removed: " + hashSetCustom.remove(22));  System.***out***.print("Displaying HashSetCustom: ");  hashSetCustom.display();  }  }  **class** HashSetCustom<E> {  **private** HashMapCustom1<E, Object> hashMapCustom;  **public** HashSetCustom() {  hashMapCustom = **new** HashMapCustom1<>();  }  **public** **void** add(E value) {  hashMapCustom.put(value, **null**);  }  **public** **boolean** contains(E obj) {  **return** hashMapCustom.contains(obj) != **null** ? **true** : **false**;  }  **public** **void** display() {  hashMapCustom.displaySet();  }  **public** **boolean** remove(E obj) {  **return** hashMapCustom.remove(obj);  }  }  **class** HashMapCustom1<K, V> {  **private** Entry<K, V>[] table;  **private** **int** capacity = 4;  **static** **class** Entry<K, V> {  K key;  V value;  Entry<K, V> next;  **public** Entry(K key, V value, Entry<K, V> next) {  **this**.key = key;  **this**.value = value;  **this**.next = next;  }  }  @SuppressWarnings("unchecked")  **public** HashMapCustom1() {  table = **new** Entry[capacity];  }  **public** **void** put(K newKey, V data) {  **if** (newKey == **null**) **return**;  **int** hash = hash(newKey);  Entry<K, V> newEntry = **new** Entry<K, V>(newKey, data, **null**);  **if** (table[hash] == **null**) {  table[hash] = newEntry;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(newKey)) {  **if** (previous == **null**) {  newEntry.next = current.next;  table[hash] = newEntry;  **return**;  } **else** {  newEntry.next = current.next;  previous.next = newEntry;  **return**;  }  }  previous = current;  current = current.next;  }  previous.next = newEntry;  }  }  **public** V get(K key) {  **int** hash = hash(key);  **if** (table[hash] == **null**) {  **return** **null**;  } **else** {  Entry<K, V> temp = table[hash];  **while** (temp != **null**) {  **if** (temp.key.equals(key))  **return** temp.value;  temp = temp.next;  }  **return** **null**;  }  }  **public** **boolean** remove(K deleteKey) {  **int** hash = hash(deleteKey);  **if** (table[hash] == **null**) {  **return** **false**;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(deleteKey)) {  **if** (previous == **null**) {  table[hash] = table[hash].next;  **return** **true**;  } **else** {  previous.next = current.next;  **return** **true**;  }  }  previous = current;  current = current.next;  }  **return** **false**;  }  }  **public** **void** display() {  **for** (**int** i = 0; i < capacity; i++) {  **if** (table[i] != **null**) {  Entry<K, V> entry = table[i];  **while** (entry != **null**) {  System.***out***.print("{" + entry.key + "=" + entry.value + "}" + " ");  entry = entry.next;  }  }  }  }  **public** K contains(K key) {  **int** hash = hash(key);  **if** (table[hash] == **null**) {  **return** **null**;  } **else** {  Entry<K, V> temp = table[hash];  **while** (temp != **null**) {  **if** (temp.key.equals(key))  **return** key;  temp = temp.next;  }  **return** **null**;  }  }  **public** **void** displaySet() {  **for** (**int** i = 0; i < capacity; i++) {  **if** (table[i] != **null**) {  Entry<K, V> entry = table[i];  **while** (entry != **null**) {  System.***out***.print(entry.key + " ");  entry = entry.next;  }  }  }  }  **private** **int** hash(K key) {  **return** Math.*abs*(key.hashCode()) % capacity;  }  } |

# **Hashtable Test1 making map unmodifiable using Collections.unmodifiableMap**

|  |
| --- |
| **public** **class** HashtableTest1 {  **public** **static** **void** main(String[] args) {  Map<Integer, String> hashtable = **new** Hashtable<Integer, String>();  hashtable.put(11, "ankit");  hashtable.put(21, "mittal");  hashtable.put(31, "javaMadeSoEasy");  Map<Integer, String> unmodifiableMap = Collections.*unmodifiableMap*(hashtable);  unmodifiableMap.put(41, "java");  }  }  /\* OUTPUT Exception in thread "main" java.lang.UnsupportedOperationException at |

# **Hashtable Test2 Iterator on KeySet, values and entrySet is fail-fast.**

|  |
| --- |
| **public** **class** HashtableTest2 {  **public** **static** **void** main(String[] args) {  Map<Integer, String> hashtable = **new** Hashtable<Integer, String>();  hashtable.put(11, "ankit");  hashtable.put(21, "javaMadeSoEasy");  System.***out***.println ("\n---1. Iterate on keys, by obtaining iterator on keySet---");  // fail-fast  Iterator<Integer> keyIterator = hashtable.keySet().iterator();  **while** (keyIterator.hasNext()) {  hashtable.put(4, "newEle1");  System.***out***.println(keyIterator.next());  }  System.***out***.println ("\n---2. Iterate on values, by obtaining iterator on values---");  // fail-fast  Iterator<String> valueIterator = hashtable.values().iterator();  **while** (valueIterator.hasNext()) {  hashtable.put(4, "newEle1");  System.***out***.println(valueIterator.next());  }  System.***out***.println ("\n---3. Iterate on entry, by obtaining iterator on entrySet---");  // fail-fast  Iterator<Entry<Integer, String>> entryIterator = hashtable.entrySet().iterator();  **while** (entryIterator.hasNext()) {  hashtable.put(4, "newEle1");  System.***out***.println(entryIterator.next());  }  }  }  /\* OUTPUT  \* ---1. Iterate on keys, by obtaining iterator on keySet--- Exception in thread  \* "main" java.util.ConcurrentModificationException at  \* java.util.Hashtable$Enumerator.next(Unknown Source) at  \* hashtable\_iterator\_fail.HashtableTest.main(HashtableTest.java:24) \*/ |

# **Hashtable Test3:-** Iterate on keys by obtaining keyset, Iterate on values by obtaining values, Iterate on entry by obtaining entrySet

|  |
| --- |
| **public** **class** HashtableTest3 {  **public** **static** **void** main(String[] args) {  Map<Integer, String> hashtable = **new** Hashtable<Integer, String>();  hashtable.put(11, "ankit");  hashtable.put(21, "javaMadeSoEasy");  System.***out***.println ("\n---1. Iterate on keys, by obtaining iterator on keySet---");  Iterator<Integer> keyIterator = hashtable.keySet().iterator();  **while** (keyIterator.hasNext()) {  System.***out***.println(keyIterator.next());  }  System.***out***.println("\n---obtain keySet from Map---");  Set<Integer> keySet = hashtable.keySet();  System.***out***.println("---now iterate on keySet using enhanced for loop---");  **for** (Integer key : keySet) {  System.***out***.println(key);  }  System.***out***.println ("\n---2. Iterate on values, by obtaining iterator on values---");  Iterator<String> valueIterator = hashtable.values().iterator();  **while** (valueIterator.hasNext()) {  System.***out***.println(valueIterator.next());  }  System.***out***.println("\n---obtain values from Map---");  Collection<String> collection = hashtable.values();  System.***out***.println("---now iterate on values using enhanced for loop---");  **for** (String value : collection) {  System.***out***.println(value);  }  System.***out***.println ("\n---3. Iterate on entry, by obtaining iterator on entrySet---");  Iterator<Entry<Integer, String>> entryIterator = hashtable.entrySet().iterator();  **while** (entryIterator.hasNext()) {  System.***out***.println(entryIterator.next());  }  System.***out***.println("\n---obtain entrySet from Map---");  Set<Entry<Integer, String>> entrySet = hashtable.entrySet();  System.***out***.println("---now iterate on entrySet using enhanced for loop---");  **for** (Entry<Integer, String> entry : entrySet) {  System.***out***.println(entry);  }  }  } |

# **Hashtable Test4 put, get, containsKey, conatinsValue, remove ans size method**

|  |
| --- |
| **public** **class** HashtableTest4 {  **public** **static** **void** main(String[] args) {  System.***out***.println ("-----1. create Map---");  Map<Integer, String> hashtable = **new** Hashtable<Integer, String>();  System.***out***.println ("\n-----2. put key-value pair in Map ---");  hashtable.put(11, "ankit");  hashtable.put(21, "mittal");  hashtable.put(31, "javaMadeSoEasy");  System.***out***.println ("\n-----3. get method returns value corresponding to key ---");  System.***out***.println("hashtable.get(11) : " + hashtable.get(11));  System.***out***.println(  "\n-----4. containsKey method returns true if map " + "contains a mapping for the specified key ---");  System.***out***.println("hashtable.containsKey(11) : " + hashtable.containsKey(11));  System.***out***.println ("\n-----5. containsValue method returns true if map maps one "  + "or more keys to the specified value. ---");  System.***out***.println("hashtable.containsValue(\"ankit\") : " + hashtable.containsValue("ankit"));  System.***out***.println("hashtable.remove(11) : " + hashtable.remove(11));  System.***out***.println ("\n-----6. size ---");  System.***out***.println("hashtable.size() : " + hashtable.size());  }  } |

# **Linked HashMap Custom App display values corresponding to keys**

|  |
| --- |
| value corresponding to key 21=12 value corresponding to key 51=null  Displaying : {21=12} {25=121} {30=151} {33=15} {35=89}  value corresponding to key 21 removed: true value corresponding to key 22 removed: false  Displaying : {25=121} {30=151} {33=15} {35=89} \*/  **public** **class** LinkedHashMapCustomApp {  **public** **static** **void** main(String[] args) {  LinkedHashMapCustom<Integer, Integer> linkedHashMapCustom = **new** LinkedHashMapCustom<Integer, Integer>();  linkedHashMapCustom.put(21, 12);  linkedHashMapCustom.put(25, 121);  linkedHashMapCustom.put(30, 151);  linkedHashMapCustom.put(33, 15);  linkedHashMapCustom.put(35, 89);  System.***out***.println("Display values corresponding to keys>");  System.***out***.println("value corresponding to key 21=" + linkedHashMapCustom.get(21));  System.***out***.println("value corresponding to key 51=" + linkedHashMapCustom.get(51));  System.***out***.print("Displaying : ");  linkedHashMapCustom.display();  System.***out***.println("\n\nvalue corresponding to key 21 removed: " + linkedHashMapCustom.remove(21));  System.***out***.println("value corresponding to key 22 removed: " + linkedHashMapCustom.remove(22));  System.***out***.print("Displaying : ");  linkedHashMapCustom.display();  }  }  **class** LinkedHashMapCustom<K, V> {  **private** Entry<K, V>[] table;  **private** **int** capacity = 4;  **private** Entry<K, V> header;  **private** Entry<K, V> last;  **static** **class** Entry<K, V> {  K key;  V value;  Entry<K, V> next;  Entry<K, V> before, after;  **public** Entry(K key, V value, Entry<K, V> next) {  **this**.key = key;  **this**.value = value;  **this**.next = next;  }  }  @SuppressWarnings("unchecked")  **public** LinkedHashMapCustom() {  table = **new** Entry[capacity];  }  **public** **void** put(K newKey, V data) {  **if** (newKey == **null**) **return**;  **int** hash = hash(newKey);  Entry<K, V> newEntry = **new** Entry<K, V>(newKey, data, **null**);  maintainOrderAfterInsert(newEntry);  **if** (table[hash] == **null**) {  table[hash] = newEntry;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(newKey)) {  **if** (previous == **null**) {  newEntry.next = current.next;  table[hash] = newEntry;  **return**;  } **else** {  newEntry.next = current.next;  previous.next = newEntry;  **return**;  }  }  previous = current;  current = current.next;  }  previous.next = newEntry;  }  }  **private** **void** maintainOrderAfterInsert(Entry<K, V> newEntry) {  **if** (header == **null**) {  header = newEntry;  last = newEntry;  **return**;  }  **if** (header.key.equals(newEntry.key)) {  deleteFirst();  insertFirst(newEntry);  **return**;  }  **if** (last.key.equals(newEntry.key)) {  deleteLast();  insertLast(newEntry);  **return**;  }  Entry<K, V> beforeDeleteEntry = deleteSpecificEntry(newEntry);  **if** (beforeDeleteEntry == **null**) {  insertLast(newEntry);  } **else** {  insertAfter(beforeDeleteEntry, newEntry);  }  }  **private** **void** maintainOrderAfterDeletion(Entry<K, V> deleteEntry) {  **if** (header.key.equals(deleteEntry.key)) {  deleteFirst();  **return**;  }  **if** (last.key.equals(deleteEntry.key)) {  deleteLast();  **return**;  }  deleteSpecificEntry(deleteEntry);  }  **private** **void** insertAfter(Entry<K, V> beforeDeleteEntry, Entry<K, V> newEntry) {  Entry<K, V> current = header;  **while** (current != beforeDeleteEntry) {  current = current.after;  }  newEntry.after = beforeDeleteEntry.after;  beforeDeleteEntry.after.before = newEntry;  newEntry.before = beforeDeleteEntry;  beforeDeleteEntry.after = newEntry;  }  **private** **void** deleteFirst() {  **if** (header == last) {  header = last = **null**;  **return**;  }  header = header.after;  header.before = **null**;  }  **private** **void** insertFirst(Entry<K, V> newEntry) {  **if** (header == **null**) {  header = newEntry;  last = newEntry;  **return**;  }  newEntry.after = header;  header.before = newEntry;  header = newEntry;  }  **private** **void** insertLast(Entry<K, V> newEntry) {  **if** (header == **null**) {  header = newEntry;  last = newEntry;  **return**;  }  last.after = newEntry;  newEntry.before = last;  last = newEntry;  }  **private** **void** deleteLast() {  **if** (header == last) {  header = last = **null**;  **return**;  }  last = last.before;  last.after = **null**;  }  **private** Entry<K, V> deleteSpecificEntry(Entry<K, V> newEntry) {  Entry<K, V> current = header;  **while** (!current.key.equals(newEntry.key)) {  **if** (current.after == **null**) {  **return** **null**;  }  current = current.after;  }  Entry<K, V> beforeDeleteEntry = current.before;  current.before.after = current.after;  current.after.before = current.before;  **return** beforeDeleteEntry;  }  **public** V get(K key) {  **int** hash = hash(key);  **if** (table[hash] == **null**) {  **return** **null**;  } **else** {  Entry<K, V> temp = table[hash];  **while** (temp != **null**) {  **if** (temp.key.equals(key))  **return** temp.value;  temp = temp.next;  }  **return** **null**;  }  }  **public** **boolean** remove(K deleteKey) {  **int** hash = hash(deleteKey);  **if** (table[hash] == **null**) {  **return** **false**;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(deleteKey)) {  maintainOrderAfterDeletion(current);  **if** (previous == **null**) {  table[hash] = table[hash].next;  **return** **true**;  } **else** {  previous.next = current.next;  **return** **true**;  }  }  previous = current;  current = current.next;  }  **return** **false**;  }  }  **public** **void** display() {  Entry<K, V> currentEntry = header;  **while** (currentEntry != **null**) {  System.***out***.print("{" + currentEntry.key + "=" + currentEntry.value + "}" + " ");  currentEntry = currentEntry.after;  }  }  **private** **int** hash(K key) {  **return** Math.*abs*(key.hashCode()) % capacity;  }  } |

# **LinkedHashMap Test1:-** Iterate on keys by obtaining KeySet, iterate on values by obtaining values and iterate on entry by obtaining entrySet

|  |
| --- |
| **public** **class** LinkedHashMapTest1 {  **public** **static** **void** main(String[] args) {  Map<Integer, String> linkedHashMap = **new** LinkedHashMap<Integer, String>();  linkedHashMap.put(11, "ankit");  linkedHashMap.put(21, "javaMadeSoEasy");  System.***out***.println ("\n---1. Iterate on keys, by obtaining iterator on keySet---");  Iterator<Integer> keyIterator = linkedHashMap.keySet().iterator();  **while** (keyIterator.hasNext()) {  System.***out***.println(keyIterator.next());  }  System.***out***.println("\n---obtain keySet from Map---");  Set<Integer> keySet = linkedHashMap.keySet();  System.***out***.println("---now iterate on keySet using enhanced for loop---");  **for** (Integer key : keySet) {  System.***out***.println(key);  }  System.***out***.println ("\n---2. Iterate on values, by obtaining iterator on values---");  Iterator<String> valueIterator = linkedHashMap.values().iterator();  **while** (valueIterator.hasNext()) {  System.***out***.println(valueIterator.next());  }  System.***out***.println("\n---obtain values from Map---");  Collection<String> collection = linkedHashMap.values();  System.***out***.println("---now iterate on values using enhanced for loop---");  **for** (String value : collection) {  System.***out***.println(value);  }  System.***out***.println ("\n---3. Iterate on entry, by obtaining iterator on entrySet---");  Iterator<Entry<Integer, String>> entryIterator = linkedHashMap.entrySet().iterator();  **while** (entryIterator.hasNext()) {  System.***out***.println(entryIterator.next());  }  System.***out***.println("\n---obtain entrySet from Map---");  Set<Entry<Integer, String>> entrySet = linkedHashMap.entrySet();  System.***out***.println("---now iterate on entrySet using enhanced for loop---");  **for** (Entry<Integer, String> entry : entrySet) {  System.***out***.println(entry);  }  }  } |

# **LinkedHashMap Test2:-** iterate on keyset, values and entryset is fail-fast or fail-safe

|  |
| --- |
| **public** **class** LinkedHashMapTest2 {  **public** **static** **void** main(String[] args) {  Map<Integer, String> linkedHashMap = **new** LinkedHashMap<Integer, String>();  linkedHashMap.put(11, "ankit");  linkedHashMap.put(21, "javaMadeSoEasy");  System.***out***.println ("\n---1. Iterate on keys, by obtaining iterator on keySet---");  // fail-fast  Iterator<Integer> keyIterator = linkedHashMap.keySet().iterator();  **while** (keyIterator.hasNext()) {  linkedHashMap.put(4, "newEle1");  System.***out***.println(keyIterator.next());  }  System.***out***.println ("\n---2. Iterate on values, by obtaining iterator on values---");  // fail-fast  Iterator<String> valueIterator = linkedHashMap.values().iterator();  **while** (valueIterator.hasNext()) {  linkedHashMap.put(4, "newEle1");  System.***out***.println(valueIterator.next());  }  System.***out***.println ("\n---3. Iterate on entry, by obtaining iterator on entrySet---");  // fail-fast  Iterator<Entry<Integer, String>> entryIterator = linkedHashMap.entrySet().iterator();  **while** (entryIterator.hasNext()) {  linkedHashMap.put(4, "newEle1");  System.***out***.println(entryIterator.next());  }  }  } |

# **LinkedHashMap Test3 put, get, conatinskey, conatinsvalue, remove and size method**

|  |
| --- |
| **public** **class** LinkedHashMapTest3 {  **public** **static** **void** main(String[] args) {  System.***out***.println ("-----1. create Map---");  Map<Integer, String> linkedHashMap = **new** LinkedHashMap<Integer, String>();  System.***out***.println ("\n-----2. put key-value pair in Map ---");  linkedHashMap.put(11, "ankit");  linkedHashMap.put(21, "mittal");  linkedHashMap.put(31, "javaMadeSoEasy");  System.***out***.println ("\n-----3. get method returns value corresponding to key ---");  System.***out***.println("linkedHashMap.get(11) : " + linkedHashMap.get(11));  System.***out***.println(  "\n-----4. containsKey method returns true if map " + "contains a mapping for the specified key ---");  System.***out***.println("linkedHashMap.containsKey(11) : " + linkedHashMap.containsKey(11));  System.***out***.println ("\n-----5. containsValue method returns true if map maps one "  + "or more keys to the specified value. ---");  System.***out***.println("linkedHashMap.containsValue(\"ankit\") : " + linkedHashMap.containsValue("ankit"));  System.***out***.println("linkedHashMap.remove(11) : " + linkedHashMap.remove(11));  System.***out***.println ("\n-----6. size ---");  System.***out***.println("linkedHashMap.size() : " + linkedHashMap.size());  }  } |

# **LinkedHashMap Example making it unmodifiable**

|  |
| --- |
| **public** **class** LinkedHashMapUnmodifiableExample {  **public** **static** **void** main(String[] args) {  Map<Integer, String> linkedHashMap = **new** LinkedHashMap<Integer, String>();  linkedHashMap.put(11, "ankit");  linkedHashMap.put(21, "mittal");  linkedHashMap.put(31, "javaMadeSoEasy");  Map<Integer, String> unmodifiableMap = Collections.*unmodifiableMap*(linkedHashMap);  unmodifiableMap.put(41, "java");  }  }  /\* OUTPUT Exception in thread "main" java.lang.UnsupportedOperationException at |

# **LinkedHashSet Custom App**

|  |
| --- |
| LinkedHashSetCustom contains 21 =true LinkedHashSetCustom contains 51 =false  Displaying LinkedHashSetCustom: 21 25 30 33 35  21 removed: true  22 removed: false  Displaying LinkedHashSetCustom: 25 30 33 35 \*/  **public** **class** LinkedHashSetCustomApp {  **public** **static** **void** main(String[] args) {  LinkedHashSetCustom<Integer> linkedHashSetCustom = **new** LinkedHashSetCustom<Integer>();  linkedHashSetCustom.add(21);  linkedHashSetCustom.add(25);  linkedHashSetCustom.add(30);  linkedHashSetCustom.add(33);  linkedHashSetCustom.add(35);  System.***out***.println("LinkedHashSetCustom contains 21 =" + linkedHashSetCustom.contains(21));  System.***out***.println("LinkedHashSetCustom contains 51 =" + linkedHashSetCustom.contains(51));  System.***out***.print("Displaying LinkedHashSetCustom: ");  linkedHashSetCustom.display();  System.***out***.println("\n\n21 removed: " + linkedHashSetCustom.remove(21));  System.***out***.println("22 removed: " + linkedHashSetCustom.remove(22));  System.***out***.print("Displaying LinkedHashSetCustom: ");  linkedHashSetCustom.display();  }  }  **class** LinkedHashSetCustom<E> {  **private** LinkedHashMapCustom1<E, Object> linkedHashMapCustom;  **public** LinkedHashSetCustom() {  linkedHashMapCustom = **new** LinkedHashMapCustom1<>();  }  **public** **void** add(E value) {  linkedHashMapCustom.put(value, **null**);  }  **public** **boolean** contains(E obj) {  **return** linkedHashMapCustom.contains(obj) != **null** ? **true** : **false**;  }  **public** **void** display() {  linkedHashMapCustom.displaySet();  }  **public** **boolean** remove(E obj) {  **return** linkedHashMapCustom.remove(obj);  }  }  **class** LinkedHashMapCustom1<K, V> {  **private** Entry<K, V>[] table;  **private** **int** capacity = 4;  **private** Entry<K, V> header;  **private** Entry<K, V> last;  **static** **class** Entry<K, V> {  K key;  V value;  Entry<K, V> next;  Entry<K, V> before;  Entry<K, V> after;  **public** Entry(K key, V value, Entry<K, V> next) {  **this**.key = key;  **this**.value = value;  **this**.next = next;  }  }  @SuppressWarnings("unchecked")  **public** LinkedHashMapCustom1() {  table = **new** Entry[capacity];  }  **public** **void** put(K newKey, V data) {  **if** (newKey == **null**)  **return**;  **int** hash = hash(newKey);  Entry<K, V> newEntry = **new** Entry<K, V>(newKey, data, **null**);  maintainOrderAfterInsert(newEntry);  **if** (table[hash] == **null**) {  table[hash] = newEntry;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(newKey)) {  **if** (previous == **null**) {  newEntry.next = current.next;  table[hash] = newEntry;  **return**;  } **else** {  newEntry.next = current.next;  previous.next = newEntry;  **return**;  }  }  previous = current;  current = current.next;  }  previous.next = newEntry;  }  }  **private** **void** maintainOrderAfterInsert(Entry<K, V> newEntry) {  **if** (header == **null**) {  header = newEntry;  last = newEntry;  **return**;  }  **if** (header.key.equals(newEntry.key)) {  deleteFirst();  insertFirst(newEntry);  **return**;  }  **if** (last.key.equals(newEntry.key)) {  deleteLast();  insertLast(newEntry);  **return**;  }  Entry<K, V> beforeDeleteEntry = deleteSpecificEntry(newEntry);  **if** (beforeDeleteEntry == **null**) {  insertLast(newEntry);  } **else** {  insertAfter(beforeDeleteEntry, newEntry);  }  }  **private** **void** maintainOrderAfterDeletion(Entry<K, V> deleteEntry) {  **if** (header.key.equals(deleteEntry.key)) {  deleteFirst();  **return**;  }  **if** (last.key.equals(deleteEntry.key)) {  deleteLast();  **return**;  }  deleteSpecificEntry(deleteEntry);  }  **private** **void** insertAfter(Entry<K, V> beforeDeleteEntry, Entry<K, V> newEntry) {  Entry<K, V> current = header;  **while** (current != beforeDeleteEntry) {  current = current.after;  }  newEntry.after = beforeDeleteEntry.after;  beforeDeleteEntry.after.before = newEntry;  newEntry.before = beforeDeleteEntry;  beforeDeleteEntry.after = newEntry;  }  **void** deleteFirst() {  **if** (header == last) {  header = last = **null**;  **return**;  }  header = header.after;  header.before = **null**;  }  **void** insertFirst(Entry<K, V> newEntry) {  **if** (header == **null**) {  header = newEntry;  last = newEntry;  **return**;  }  newEntry.after = header;  header.before = newEntry;  header = newEntry;  }  **void** insertLast(Entry<K, V> newEntry) {  **if** (header == **null**) {  header = newEntry;  last = newEntry;  **return**;  }  last.after = newEntry;  newEntry.before = last;  last = newEntry;  }  **void** deleteLast() {  **if** (header == last) {  header = last = **null**;  **return**;  }  last = last.before;  last.after = **null**;  }  **private** Entry<K, V> deleteSpecificEntry(Entry<K, V> newEntry) {  Entry<K, V> current = header;  **while** (!current.key.equals(newEntry.key)) {  **if** (current.after == **null**) {  **return** **null**;  }  current = current.after;  }  Entry<K, V> beforeDeleteEntry = current.before;  current.before.after = current.after;  current.after.before = current.before;  **return** beforeDeleteEntry;  }  **public** V get(K key) {  **int** hash = hash(key);  **if** (table[hash] == **null**) {  **return** **null**;  } **else** {  Entry<K, V> temp = table[hash];  **while** (temp != **null**) {  **if** (temp.key.equals(key))  **return** temp.value;  temp = temp.next;  }  **return** **null**;  }  }  **public** **boolean** remove(K deleteKey) {  **int** hash = hash(deleteKey);  **if** (table[hash] == **null**) {  **return** **false**;  } **else** {  Entry<K, V> previous = **null**;  Entry<K, V> current = table[hash];  **while** (current != **null**) {  **if** (current.key.equals(deleteKey)) {  maintainOrderAfterDeletion(current);  **if** (previous == **null**) {  table[hash] = table[hash].next;  **return** **true**;  } **else** {  previous.next = current.next;  **return** **true**;  }  }  previous = current;  current = current.next;  }  **return** **false**;  }  }  **public** **void** display() {  Entry<K, V> currentEntry = header;  **while** (currentEntry != **null**) {  System.***out***.print("{" + currentEntry.key + "=" + currentEntry.value + "}" + " ");  currentEntry = currentEntry.after;  }  }  **private** **int** hash(K key) {  **return** Math.*abs*(key.hashCode()) % capacity;  }  **public** K contains(K key) {  **int** hash = hash(key);  **if** (table[hash] == **null**) {  **return** **null**;  } **else** {  Entry<K, V> temp = table[hash];  **while** (temp != **null**) {  **if** (temp.key.equals(key))  **return** key;  temp = temp.next;  }  **return** **null**;  }  }  **public** **void** displaySet() {  Entry<K, V> currentEntry = header;  **while** (currentEntry != **null**) {  System.***out***.print(currentEntry.key + " ");  currentEntry = currentEntry.after;  }  }  } |

# **Linked Hash Set Example isEmpty, size and clear method**

|  |
| --- |
| **public** **class** LinkedHashSetExample {  **public** **static** **void** main(String[] args) {  Set<String> linkedLinkedHashSet = **new** LinkedHashSet<String>();  System.***out***.println("--------add element ");  linkedLinkedHashSet.add("ankit");  linkedLinkedHashSet.add("mittal");  linkedLinkedHashSet.add("javaMadeSoEasy");  System.***out***.println("linkedLinkedHashSet = " + linkedLinkedHashSet);  System.***out***.println("linkedLinkedHashSet.isEmpty() = " + linkedLinkedHashSet.isEmpty());  System.***out***.println("\n--------clear list - remove all elements from set");  linkedLinkedHashSet.clear();  System.***out***.println("linkedLinkedHashSet.isEmpty() = " + linkedLinkedHashSet.isEmpty());  System.***out***.println("linkedLinkedHashSet = " + linkedLinkedHashSet);  }  } |

# **Linked HashSet Iterator Example iterate using iterator, enumeration and enhanced for loop**

|  |
| --- |
| **public** **class** LinkedHashSetIteratorExample {  **public** **static** **void** main(String[] args) {  Set<String> linkedLinkedHashSet = **new** LinkedHashSet<String>();  linkedLinkedHashSet.add("ankit");  linkedLinkedHashSet.add("javaMadeSoEasy");  System.***out***.println("-------use iterator-------");  Iterator<String> iterator = linkedLinkedHashSet.iterator();  **while** (iterator.hasNext()) {  System.***out***.println(iterator.next());  }  System.***out***.println("-------use Enumeration-------");  Enumeration<String> listEnum = Collections.*enumeration*(linkedLinkedHashSet);  **while** (listEnum.hasMoreElements()) {  System.***out***.println(listEnum.nextElement());  }  System.***out***.println("-------use enhanced for loop-------");  **for** (String string : linkedLinkedHashSet) {  System.***out***.println(string);  }  }  } |

# **Linked Hash Set Iterator Fail Fast Example using iterator, enumeration and enhanced for loop**

|  |
| --- |
| **public** **class** LinkedHashSetIteratorFailFastExample {  **public** **static** **void** main(String[] args) {  Set<String> linkedLinkedHashSet = **new** LinkedHashSet<String>();  linkedLinkedHashSet.add("ankit");  linkedLinkedHashSet.add("javaMadeSoEasy");  System.***out***.println("-------use iterator-------");  // fail-fast  Iterator<String> iterator = linkedLinkedHashSet.iterator();  **while** (iterator.hasNext()) {  linkedLinkedHashSet.add("newElement1");  System.***out***.println(iterator.next());  }  System.***out***.println("-------use Enumeration-------");  // fail-fast  Enumeration<String> listEnum = Collections.*enumeration*(linkedLinkedHashSet);  **while** (listEnum.hasMoreElements()) {  linkedLinkedHashSet.add("newElement2");  System.***out***.println(listEnum.nextElement());  }  System.***out***.println("-------use enhanced for loop-------");  // enhanced for loop is fail-fast  **for** (String string : linkedLinkedHashSet) {  linkedLinkedHashSet.add("newElement3");  System.***out***.println(string);  }  }  } |

# **Linked Hash Set Synchronizing Example using Collections.synchronizedSet**

|  |
| --- |
| **public** **class** LinkedHashSetSynchronizingExample {  **public** **static** **void** main(String[] args) {  Set<String> linkedLinkedHashSet = **new** LinkedHashSet<String>();  linkedLinkedHashSet.add("ankit");  linkedLinkedHashSet.add("javaMadeSoEasy");  Set<String> synchronizedSet = Collections.*synchronizedSet*(linkedLinkedHashSet);  System.***out***.println("------- iterating on synchronized set -------");  **synchronized** (synchronizedSet) {  Iterator<String> iterator = synchronizedSet.iterator();  **while** (iterator.hasNext()) {  System.***out***.println(iterator.next());  }  }  }  } |

# **Linked Hash Set Example making set unmodifiable using Collections.unmodifiableSet**

|  |
| --- |
| **public** **class** LinkedHashSetUnmodifiableExample {  **public** **static** **void** main(String[] args) {  Set<String> linkedLinkedHashSet = **new** LinkedHashSet<String>();  linkedLinkedHashSet.add("ankit");  linkedLinkedHashSet.add("javaMadeSoEasy");  Set<String> unmodifiableSet = Collections.*unmodifiableSet*(linkedLinkedHashSet);  unmodifiableSet.add("mittal");  }  } |

# **Removing Duplicate Elements Using HashSet**

|  |
| --- |
| **public** **class** RemovingDuplicateElementsUsingHashSet {  **public** **static** **void** main(String[] args) {  ArrayList<String> listWithDuplicateElements = **new** ArrayList<String>();  listWithDuplicateElements.add("JAVA");  listWithDuplicateElements.add("J2EE");  listWithDuplicateElements.add("JSP");  listWithDuplicateElements.add("SERVLETS");  listWithDuplicateElements.add("JAVA");  listWithDuplicateElements.add("STRUTS");  listWithDuplicateElements.add("JSP");  System.***out***.print("ArrayList With Duplicate Elements :");  System.***out***.println(listWithDuplicateElements);  HashSet<String> set = **new** HashSet<String>(listWithDuplicateElements);  ArrayList<String> listWithoutDuplicateElements = **new** ArrayList<String>(set);  System.***out***.print("ArrayList After Removing Duplicate Elements :");  System.***out***.println(listWithoutDuplicateElements);  }  }  /\*ArrayList With Duplicate Elements :[JAVA, J2EE, JSP, SERVLETS, JAVA, STRUTS, JSP]  ArrayList After Removing Duplicate Elements :[JAVA, SERVLETS, JSP, J2EE, STRUTS]\*/ |

# **Removing Duplicate Elements Using LinkedHashSet**

|  |
| --- |
| **public** **class** RemovingDuplicateElementsUsingLinkedHashSet {  **public** **static** **void** main(String[] args) {  ArrayList<String> listWithDuplicateElements = **new** ArrayList<String>();  listWithDuplicateElements.add("JAVA");  listWithDuplicateElements.add("J2EE");  listWithDuplicateElements.add("JSP");  listWithDuplicateElements.add("SERVLETS");  listWithDuplicateElements.add("JAVA");  listWithDuplicateElements.add("STRUTS");  listWithDuplicateElements.add("JSP");  System.***out***.print("ArrayList With Duplicate Elements :");  System.***out***.println(listWithDuplicateElements);  LinkedHashSet<String> set = **new** LinkedHashSet<String>(listWithDuplicateElements);  ArrayList<String> listWithoutDuplicateElements = **new** ArrayList<String>(set);  System.***out***.print("ArrayList After Removing Duplicate Elements :");  System.***out***.println(listWithoutDuplicateElements);  }  }  /\*ArrayList With Duplicate Elements :[JAVA, J2EE, JSP, SERVLETS, JAVA, STRUTS, JSP]  ArrayList After Removing Duplicate Elements :[JAVA, J2EE, JSP, SERVLETS, STRUTS]\*/ |

# **Reverse ArrayList**

|  |
| --- |
| **public** **class** ReverseArrayList {  **public** **static** **void** main(String[] args) {  ArrayList<String> list = **new** ArrayList<String>();  list.add("Gold");  list.add("Iron");  list.add("Copper");  list.add("Silver");  list.add("Nickel");  list.add("Cobalt");  list.add("Zinc");  System.***out***.println("ArrayList Before Reverse :");  System.***out***.println(list);  Collections.*reverse*(list);  System.***out***.println("ArrayList After Reverse :");  System.***out***.println(list);  }  } |

# **Reverse LinkedList**

|  |
| --- |
| **public** **class** ReverseLinkedList {  **public** **static** **void** main(String[] args) {  LinkedList<Integer> list = **new** LinkedList<Integer>();  list.add(56);  list.add(67);  list.add(81);  list.add(41);  list.add(63);  list.add(21);  list.add(96);  System.***out***.println("LinkedList Before Reverse :");  System.***out***.println(list);  Collections.*reverse*(list);  System.***out***.println("LinkedList After Reverse :");  System.***out***.println(list);  }  } |

# **SortMap by Key**

|  |
| --- |
| //TreeMap’s constructor to sort map on basis of key OUTPUT treeMap : {2=1, 3=1, 4=1}  **public** **class** SortMapByKey {  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> hashMap = **new** HashMap<Integer, Integer>();  hashMap.put(4, 1);  hashMap.put(2, 1);  hashMap.put(3, 1);  Map<Integer, Integer> treeMap = **new** TreeMap<Integer, Integer>(hashMap);  System.***out***.println("treeMap : " + treeMap);  }  } |

# **SortMap by Key descending**

|  |
| --- |
| OUTPUT treeMap : {5=1, 4=1, 3=1, 2=1}  **public** **class** SortMapByKeyAscending {  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> treeMap = **new** TreeMap<Integer, Integer>(**new** Comparator<Integer>() {  @Override  **public** **int** compare(Integer o1, Integer o2) {  // using Comparator to sort map in descending order of keys.  **return** o2.compareTo(o1);  }  });  treeMap.put(4, 1);  treeMap.put(2, 1);  treeMap.put(3, 1);  treeMap.put(5, 1);  System.***out***.println("treeMap : " + treeMap);  }  } |

# **SortMap by Key Ascending**

|  |
| --- |
| Before sorting by key : 4=1 2=1 3=1 5=1  After sorting by key(ascending): 2=1 3=1 4=1 5=1  **public** **class** SortMapByKeyAscending1 {  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> map = **new** LinkedHashMap<Integer, Integer>();  map.put(4, 1);  map.put(2, 1);  map.put(3, 1);  map.put(5, 1);  Set<Entry<Integer, Integer>> entrySet = map.entrySet();  List<Entry<Integer, Integer>> listOfentrySet = **new** ArrayList<Entry<Integer, Integer>>(entrySet);  System.***out***.print("Before sorting by key : ");  **for** (Map.Entry<Integer, Integer> entry : listOfentrySet) {  System.***out***.print(entry.getKey() + "=" + entry.getValue() + " ");  }  Collections.*sort*(listOfentrySet, **new** SortByKeyAscending());  System.***out***.print("\nAfter sorting by key(ascending): ");  **for** (Map.Entry<Integer, Integer> entry : listOfentrySet)  System.***out***.print(entry.getKey() + "=" + entry.getValue() + " ");  }  }  **class** SortByKeyAscending **implements** Comparator<Map.Entry<Integer, Integer>> {  @Override  **public** **int** compare(Map.Entry<Integer, Integer> entry1, Map.Entry<Integer, Integer> entry2) {  **return** (entry1.getKey()).compareTo(entry2.getKey());  }  } |

# **SortMap by Value Ascending by implementing Comparator interface and overriding its compare () method**

|  |
| --- |
| Before sorting by value: 1=2 2=1 3=9 4=8  After sorting by value(ascending): 2=1 1=2 4=8 3=9  **public** **class** SortMapByValueAscending {  **public** **static** **void** main(String[] args) {  Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();  map.put(1, 2);  map.put(2, 1);  map.put(3, 9);  map.put(4, 8);  Set<Entry<Integer, Integer>> entrySet = map.entrySet();  List<Entry<Integer, Integer>> listOfentrySet = **new** ArrayList<Entry<Integer, Integer>>(entrySet);  System.***out***.print("Before sorting by value: ");  **for** (Map.Entry<Integer, Integer> entry : listOfentrySet) {  System.***out***.print(entry.getKey() + "=" + entry.getValue() + " ");  }  Collections.*sort*(listOfentrySet, **new** SortByValue());  System.***out***.print("\nAfter sorting by value(ascending): ");  **for** (Map.Entry<Integer, Integer> entry : listOfentrySet)  System.***out***.print(entry.getKey() + "=" + entry.getValue() + " ");  }  }  **class** SortByValue **implements** Comparator<Map.Entry<Integer, Integer>> {  @Override  **public** **int** compare(Map.Entry<Integer, Integer> entry1, Map.Entry<Integer, Integer> entry2) {  **return** (entry1.getValue()).compareTo(entry2.getValue());  }  } |

# **Program to sort set using treeMap’s constructor**

|  |
| --- |
| OUTPUT treeSet : [1, 2, 3]  **public** **class** SortSet {  **public** **static** **void** main(String[] args) {  Collection<Integer> collection = **new** HashSet<Integer>();  collection.add(3);  collection.add(1);  collection.add(2);  // TreeSet's constructor  Set<Integer> treeSet = **new** TreeSet<Integer>(collection);  System.***out***.println("treeSet : " + treeSet);  }  } |

# **Program to sort set in Asecnding order by implementing TreeSet and implementing Comaparator interface**

|  |
| --- |
| /\*OUTPUT treeSet : [Employee{name=amy, id=2}, Employee{name=brad, id=1}, Employee{name=sam, id=4}]  **public** **class** SortSet1 {  **public** **static** **void** main(String[] args) {  Employee11 emp1 = **new** Employee11("sam", "4");  Employee11 emp2 = **new** Employee11("amy", "2");  Employee11 emp3 = **new** Employee11("brad", "1");  Set<Employee11> treeSet = **new** TreeSet<Employee11>(**new** Comparator<Employee11>() {  @Override  **public** **int** compare(Employee11 o1, Employee11 o2) {  **return** o1.name.compareTo(o2.name);  }  });  treeSet.add(emp1);  treeSet.add(emp2);  treeSet.add(emp3);  System.***out***.println("treeSet : " + treeSet);  }  }  **class** Employee11 {  String name;  String id;  **public** Employee11(String name, String id) {  **this**.name = name;  **this**.id = id;  }  @Override  **public** String toString() {  **return** "Employee{" + "name=" + name + ", id=" + id + '}';  }  } |

# **Synchronized HashMap**

|  |
| --- |
| **public** **class** SynchronizedHashMap {  **public** **static** **void** main(String[] args) {  HashMap<String, Integer> map = **new** HashMap<String, Integer>();  map.put("ONE", 1);  map.put("TWO", 2);  map.put("THREE", 3);  map.put("FOUR", 4);  map.put("FIVE", 5);  Map<String, Integer> synchronizedMap = Collections.*synchronizedMap*(map);  Set<String> keySet = synchronizedMap.keySet();  System.***out***.println ("Keys.............");  **synchronized** (synchronizedMap) {  Iterator<String> it = keySet.iterator();  **while** (it.hasNext()) {  System.***out***.println(it.next());  }  }  Collection<Integer> values = synchronizedMap.values();  System.***out***.println ("Values.............");  **synchronized** (synchronizedMap) {  Iterator<Integer> it = values.iterator();  **while** (it.hasNext()) {  System.***out***.println(it.next());  }  }  }  } |

# **Synchronized HashSet**

|  |
| --- |
| **public** **class** SynchronizedHashSet {  **public** **static** **void** main(String[] args) {  HashSet<String> set = **new** HashSet<String>();  set.add("JAVA");  set.add("STRUTS");  set.add("JSP");  set.add("SERVLETS");  set.add("JSF");  Set<String> synchronizedSet = Collections.*synchronizedSet*(set);  **synchronized** (synchronizedSet) {  Iterator<String> it = synchronizedSet.iterator();  **while** (it.hasNext()) {  System.***out***.println(it.next());  }  }  }  } |

# **Synchronized List**

|  |
| --- |
| /\*To synchronize ArrayList, we use Collections.synchronizedList () method. This method returns synchronized list backed by the specified list. There is an advice from javadocs that while iterating over the synchronized list, you must use it in a synchronized block. Failed to do so may result in non-deterministic behavior.\*/  **public** **class** SynchronizedList {  **public** **static** **void** main(String[] args) {  ArrayList<String> list = **new** ArrayList<String>();  list.add("JAVA");  list.add("STRUTS");  list.add("JSP");  list.add("SERVLETS");  list.add("JSF");  List<String> synchronizedList = Collections.*synchronizedList*(list);  **synchronized** (synchronizedList) {  Iterator<String> it = synchronizedList.iterator();  **while** (it.hasNext()) {  System.***out***.println(it.next());  }  }  }  } |

# **TreeMap Example using put, get, conatinskey, conatinsvalue, remove and size method.**

|  |
| --- |
| **public** **class** TreeMapExample {  **public** **static** **void** main(String[] args) {  System.***out***.println ("-----1. create Map---");  Map<Integer, String> treeMap = **new** TreeMap<Integer, String>();  System.***out***.println ("\n-----2. put key-value pair in Map ---");  treeMap.put(11, "ankit");  treeMap.put(21, "mittal");  treeMap.put(31, "javaMadeSoEasy");  System.***out***.println ("\n-----3. get method returns value corresponding to key ---");  System.***out***.println("treeMap.get(11) : " + treeMap.get(11));  System.***out***.println(  "\n-----4. containsKey method returns true if map " + "contains a mapping for the specified key ---");  System.***out***.println("treeMap.containsKey(11) : " + treeMap.containsKey(11));  System.***out***.println ("\n-----5. containsValue method returns true if map maps one "  + "or more keys to the specified value. ---");  System.***out***.println("treeMap.containsValue(\"ankit\") : " + treeMap.containsValue("ankit"));  System.***out***.println("treeMap.remove(11) : " + treeMap.remove(11));  System.***out***.println ("\n-----6. size ---");  System.***out***.println("treeMap.size() : " + treeMap.size());  }  } |

# **TreeMap Fail Fast Example**

|  |
| --- |
| **public** **class** TreeMapFailFastExample {  **public** **static** **void** main(String[] args) {  Map<Integer, String> treeMap = **new** TreeMap<Integer, String>();  treeMap.put(11, "ankit");  treeMap.put(21, "javaMadeSoEasy");  System.***out***.println ("\n---1. Iterate on keys, by obtaining iterator on keySet---");  // fail-fast  Iterator<Integer> keyIterator = treeMap.keySet().iterator();  **while** (keyIterator.hasNext()) {  treeMap.put(4, "newEle1");  System.***out***.println(keyIterator.next());  }  System.***out***.println ("\n---2. Iterate on values, by obtaining iterator on values---");  // fail-fast  Iterator<String> valueIterator = treeMap.values().iterator();  **while** (valueIterator.hasNext()) {  treeMap.put(4, "newEle1");  System.***out***.println(valueIterator.next());  }  System.***out***.println ("\n---3. Iterate on entry, by obtaining iterator on entrySet---");  // fail-fast  Iterator<Entry<Integer, String>> entryIterator = treeMap.entrySet().iterator();  **while** (entryIterator.hasNext()) {  treeMap.put(4, "newEle1");  System.***out***.println(entryIterator.next());  }  }  }  /\* OUTPUT 1. Iterate on keys, by obtaining iterator on keySet--- Exception in thread  \* "main" java.util.ConcurrentModificationException at \*/ |

# **TreeMap Iterator Example**

|  |
| --- |
| **public** **class** TreeMapIteratorExample {  **public** **static** **void** main(String[] args) {  Map<Integer, String> treeMap = **new** TreeMap<Integer, String>();  treeMap.put(21, "javaMadeSoEasy");  treeMap.put(11, "ankit");  System.***out***.println ("\n---1. Iterate on keys, by obtaining iterator on keySet---");  Iterator<Integer> keyIterator = treeMap.keySet().iterator();  **while** (keyIterator.hasNext()) {  System.***out***.println(keyIterator.next());  }  System.***out***.println("\n---obtain keySet from Map---");  Set<Integer> keySet = treeMap.keySet();  System.***out***.println("---now iterate on keySet using enhanced for loop---");  **for** (Integer key : keySet) {  System.***out***.println(key);  }  System.***out***.println ("\n---2. Iterate on values, by obtaining iterator on values---");  Iterator<String> valueIterator = treeMap.values().iterator();  **while** (valueIterator.hasNext()) {  System.***out***.println(valueIterator.next());  }  System.***out***.println("\n---obtain values from Map---");  Collection<String> collection = treeMap.values();  System.***out***.println("---now iterate on values using enhanced for loop---");  **for** (String value : collection) {  System.***out***.println(value);  }  System.***out***.println ("\n---3. Iterate on entry, by obtaining iterator on entrySet---");  Iterator<Entry<Integer, String>> entryIterator = treeMap.entrySet().iterator();  **while** (entryIterator.hasNext()) {  System.***out***.println(entryIterator.next());  }  System.***out***.println("\n---obtain entrySet from Map---");  Set<Entry<Integer, String>> entrySet = treeMap.entrySet();  System.***out***.println("---now iterate on entrySet using enhanced for loop---");  **for** (Entry<Integer, String> entry : entrySet) {  System.***out***.println(entry);  }  }  } |

# **TreeMap Example to make it unmodifiable**

|  |
| --- |
| **public** **class** TreeMapUnmodifiableExample {  **public** **static** **void** main(String[] args) {  Map<Integer, String> treeMap = **new** TreeMap<Integer, String>();  treeMap.put(11, "ankit");  treeMap.put(21, "mittal");  treeMap.put(31, "javaMadeSoEasy");  Map<Integer, String> unmodifiableMap = Collections.*unmodifiableMap*(treeMap);  unmodifiableMap.put(41, "java");  }  }  /\* OUTPUT Exception in thread "main" java.lang.UnsupportedOperationException at |

# **Vector Custom Example**

|  |
| --- |
| //show advantage of using Vector in multithreading environment in java element at index 0 = 54  **public** **class** VectorCustomExample {  **public** **static** **void** main(String[] args) **throws** InterruptedException {  **final** VectorCustom<Integer> list = **new** VectorCustom<Integer>();  **new** Thread() {  **public** **void** run() {  list.add(54);  }  }.start();  Thread.*sleep*(10);  **new** Thread() {  **public** **void** run() {  System.***out***.println("element at index " + 0 + " = " + list.get(0));  }  }.start();  }  }  **class** VectorCustom<E> {  **private** **static** **final** **int** ***INITIAL\_CAPACITY*** = 10;  **private** Object elementData[] = {};  **private** **int** size = 0;  **public** VectorCustom() {  elementData = **new** Object[***INITIAL\_CAPACITY***];  }  **public** **synchronized** **void** add(E e) {  **if** (size == elementData.length) {  ensureCapacity();  }  **try** {  Thread.*sleep*(1000);  } **catch** (InterruptedException ex) {  ex.printStackTrace();  }  elementData[size++] = e;  }  @SuppressWarnings("unchecked")  **public** **synchronized** E get(**int** index) {  **if** (index < 0 || index >= size) {  **throw** **new** IndexOutOfBoundsException("Index: " + index + ", Size " + index);  }  **return** (E) elementData[index];  }  **public** **synchronized** Object remove(**int** index) {  **if** (index < 0 || index >= size) {  **throw** **new** IndexOutOfBoundsException("Index: " + index + ", Size " + index);  }  Object removedElement = elementData[index];  **for** (**int** i = index; i < size; i++) {  elementData[i] = elementData[i + 1];  }  size--;  **return** removedElement;  }  **private** **void** ensureCapacity() {  **int** newIncreasedCapacity = elementData.length \* 2;  elementData = Arrays.*copyOf*(elementData, newIncreasedCapacity);  }  **public** **void** display() {  System.***out***.print("Displaying list : ");  **for** (**int** i = 0; i < size; i++) {  System.***out***.print(elementData[i] + " ");  }  }  } |

# **Arithmetic Progression: -** count of AP subsequences in an array

|  |
| --- |
| Input : arr[] = { 1, 2, 3 } Output : 8  Arithmetic Progression subsequence from the given array are: {}, { 1 }, { 2 }, { 3 }, { 1, 2 },{ 2, 3 }, { 1, 3 }, { 1, 2, 3 }.  **public** **class** ArithmeticProgression {  **static** **final** **int** ***MAX*** = 1000001;  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3 };  **int** n = arr.length;  System.***out***.println(*numofAP*(arr, n)); //8  }  **static** **int** numofAP(**int** a[], **int** n) {  **int** minarr = +2147483647;  **int** maxarr = -2147483648;  **for** (**int** i = 0; i < n; i++) {  minarr = Math.*min*(minarr, a[i]);  maxarr = Math.*max*(maxarr, a[i]);  }  **int** dp[] = **new** **int**[n];  **int** sum[] = **new** **int**[***MAX***];  **int** ans = n + 1;  **for** (**int** d = (minarr - maxarr); d <= (maxarr - minarr); d++) {  Arrays.*fill*(sum, 0);  **for** (**int** i = 0; i < n; i++) {  dp[i] = 1;  **if** (a[i] - d >= 1 && a[i] - d <= 1000000)  dp[i] += sum[a[i] - d];  ans += dp[i] - 1;  sum[a[i]] += dp[i];  }  }  **return** ans;  }  } |

# **Bellman Ford Algorithm:** - works for Graphs with negative weight edges. And suites well for distributed systems

|  |
| --- |
| **public** **class** BellmanFordAlgorithm {  **class** Edge {  **int** src, dest, weight;  Edge() {  src = dest = weight = 0;  }  };  **int** V, E;  Edge edge[];  BellmanFordAlgorithm(**int** v, **int** e) {  V = v;  E = e;  edge = **new** Edge[e];  **for** (**int** i = 0; i < e; ++i)  edge[i] = **new** Edge();  }  **public** **static** **void** main(String[] args) {  **int** V = 5; // Number of vertices in graph  **int** E = 8; // Number of edges in graph  BellmanFordAlgorithm graph = **new** BellmanFordAlgorithm(V, E);  graph.edge[0].src = 0;  graph.edge[0].dest = 1;  graph.edge[0].weight = -1;  graph.edge[1].src = 0;  graph.edge[1].dest = 2;  graph.edge[1].weight = 4;  graph.edge[2].src = 1;  graph.edge[2].dest = 2;  graph.edge[2].weight = 3;  graph.edge[3].src = 1;  graph.edge[3].dest = 3;  graph.edge[3].weight = 2;  graph.edge[4].src = 1;  graph.edge[4].dest = 4;  graph.edge[4].weight = 2;  graph.edge[5].src = 3;  graph.edge[5].dest = 2;  graph.edge[5].weight = 5;  graph.edge[6].src = 3;  graph.edge[6].dest = 1;  graph.edge[6].weight = 1;  graph.edge[7].src = 4;  graph.edge[7].dest = 3;  graph.edge[7].weight = -3;  graph.BellmanFord(graph, 0);  }  **void** BellmanFord(BellmanFordAlgorithm graph, **int** src) {  **int** V = graph.V, E = graph.E;  **int** dist[] = **new** **int**[V];  **for** (**int** i = 0; i < V; ++i)  dist[i] = Integer.***MAX\_VALUE***;  dist[src] = 0;  **for** (**int** i = 1; i < V; ++i) {  **for** (**int** j = 0; j < E; ++j) {  **int** u = graph.edge[j].src;  **int** v = graph.edge[j].dest;  **int** weight = graph.edge[j].weight;  **if** (dist[u] != Integer.***MAX\_VALUE*** && dist[u] + weight < dist[v])  dist[v] = dist[u] + weight;  }  }  **for** (**int** j = 0; j < E; ++j) {  **int** u = graph.edge[j].src;  **int** v = graph.edge[j].dest;  **int** weight = graph.edge[j].weight;  **if** (dist[u] != Integer.***MAX\_VALUE*** && dist[u] + weight < dist[v])  System.***out***.println("Graph contains negative weight cycle");  }  printArr(dist, V);  }  **void** printArr(**int** dist[], **int** V) {  System.***out***.println("Vertex Distance from Source");  **for** (**int** i = 0; i < V; ++i)  System.***out***.println(i + "\t\t" + dist[i]);  }  }  /\*Vertex Distance from Source  0 0  1 -1  2 2  3 -2  4 1\*/ |

# **Bitmasking: -** count ways to assign unique cap to every person. There 100 different types of caps each having a unique id from 1 to 100. Also there ‘n’ persons each having a collection of variable number of caps. One day all of these persons decide to go in a party wearing a cap but to look unique they decided that none them will wear the same type of caps. So count the total number of arrangements or ways such that none of them is wearing same type of cap.

|  |
| --- |
| Input:  3  5 100 1 // Collection of first person.  2 // Collection of second person.  5 100 // Collection of second person.  Output:4  Explanation: All valid possible ways are (5, 2, 100), (100, 2, 5), (1, 2, 5) and (1, 2, 100)\*/  **public** **class** Bitmasking {  **static** **final** **int** ***MOD*** = 1000000007;  **static** BufferedReader *br* = **new** BufferedReader(**new** InputStreamReader(System.***in***));  **static** Vector<Integer> *capList*[] = **new** Vector[101];  **static** **int** *dp*[][] = **new** **int**[1025][101];  **static** **int** *allmask*;  **public** **static** **void** main(String[] args) **throws** Exception {  **int** n;  **for** (**int** i = 0; i < *capList*.length; i++)  *capList*[i] = **new** Vector<>();  n = Integer.*parseInt*(*br*.readLine());  *countWays*(n);  }  **static** **void** countWays(**int** n) **throws** Exception {  String str;  String split[];  **int** x;  **for** (**int** i = 0; i < n; i++) {  str = *br*.readLine();  split = str.split(" ");  **for** (**int** j = 0; j < split.length; j++) {  x = Integer.*parseInt*(split[j]);  *capList*[x].add(i);  }  }  *allmask* = (1 << n) - 1;  **for** (**int**[] is : *dp*) {  **for** (**int** i = 0; i < is.length; i++) {  is[i] = -1;  }  }  System.***out***.println(*countWaysUtil*(0, 1));  }  **static** **long** countWaysUtil(**int** mask, **int** i) {  **if** (mask == *allmask*)  **return** 1;  **if** (i > 100)  **return** 0;  **if** (*dp*[mask][i] != -1)  **return** *dp*[mask][i];  **long** ways = *countWaysUtil*(mask, i + 1);  **int** size = *capList*[i].size();  **for** (**int** j = 0; j < size; j++) {  **if** ((mask & (1 << *capList*[i].get(j))) != 0)  **continue**;  **else**  ways += *countWaysUtil*(mask | (1 << *capList*[i].get(j)), i + 1);  ways %= ***MOD***;  }  **return** *dp*[mask][i] = (**int**) ways;  }  } |

# **BoxStacking:-** you are given a set of n types of rectangular-3d boxes, where the i^th box has height h(i) , width w(i) and depth d(i) ( all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2d base of the lower box are each strictly larger than those of the 2d base of higer box.

|  |
| --- |
| **public** **class** BoxStacking {  **public** **static** **void** main(String[] args) {  Box[] arr = **new** Box[4];  arr[0] = **new** Box(4, 6, 7);  arr[1] = **new** Box(1, 2, 3);  arr[2] = **new** Box(4, 5, 6);  arr[3] = **new** Box(10, 12, 32);  System.***out***.println("The maximum possible " + "height of stack is " + *maxStackHeight*(arr, 4)); 60  }  **static** **int** maxStackHeight(Box arr[], **int** n) {  Box[] rot = **new** Box[n \* 3];  **for** (**int** i = 0; i < n; i++) {  Box box = arr[i];  rot[3 \* i] = **new** Box(box.h, Math.*max*(box.w, box.d), Math.*min*(box.w, box.d));  rot[3 \* i + 1] = **new** Box(box.w, Math.*max*(box.h, box.d), Math.*min*(box.h, box.d));  rot[3 \* i + 2] = **new** Box(box.d, Math.*max*(box.w, box.h), Math.*min*(box.w, box.h));  }  **for** (**int** i = 0; i < rot.length; i++)  rot[i].area = rot[i].w \* rot[i].d;  Arrays.*sort*(rot);  **int** count = 3 \* n;  **int**[] msh = **new** **int**[count];  **for** (**int** i = 0; i < count; i++)  msh[i] = rot[i].h;  **for** (**int** i = 0; i < count; i++) {  msh[i] = 0;  Box box = rot[i];  **int** val = 0;  **for** (**int** j = 0; j < i; j++) {  Box prevBox = rot[j];  **if** (box.w < prevBox.w && box.d < prevBox.d) {  val = Math.*max*(val, msh[j]);  }  }  msh[i] = val + box.h;  }  **int** max = -1;  **for** (**int** i = 0; i < count; i++) {  max = Math.*max*(max, msh[i]);  }  **return** max;  }  **static** **class** Box **implements** Comparable<Box> {  **int** h, w, d, area;  **public** Box(**int** h, **int** w, **int** d) {  **this**.h = h;  **this**.w = w;  **this**.d = d;  }  @Override  **public** **int** compareTo(Box o) {  **return** o.area - **this**.area;  }  }  } |

# **Clustering Partitioning an Array such that sum of square differences is minimum**

|  |
| --- |
| Input : arr[] = {1, 5, 8, 10} k = 2 Output : 20 Time Complexity of O(2^n)\*/  **public** **class** ClusteringPartitioningArray {  **static** **int** *inf* = 1000000000;  **static** **int** *ans* = *inf*;  **public** **static** **void** main(String[] args) {  **int** k = 2;  **int** a[] = { 1, 5, 8, 10 };  **int** n = a.length;  *solve*(-1, 0, a, n, k, 0);  System.***out***.println(*ans*); // 20  }  **static** **void** solve(**int** i, **int** par, **int** a[], **int** n, **int** k, **int** current\_ans) {  **if** (par > k)  **return**;  **if** (par == k && i == n - 1) {  *ans* = Math.*min*(*ans*, current\_ans);  **return**;  }  **for** (**int** j = i + 1; j < n; j++)  *solve*(j, par + 1, a, n, k, current\_ans + (a[j] - a[i + 1]) \* (a[j] - a[i + 1]));  }  } |

# **Clustering Partitioning Array1 using dynamic programming**

|  |
| --- |
| Time Complexity: Having the three simple loops, the complexity of the above algorithm is (n^2k).  **public** **class** ClusteringPartitioningArray1 {  **static** **int** *inf* = 1000000000;  **public** **static** **void** main(String[] args) {  **int** k = 2;  **int** a[] = { 1, 5, 8, 10 };  **int** n = a.length;  System.***out***.println(*minCost*(a, n, k)); //20  }  **static** **int** minCost(**int** a[], **int** n, **int** k) {  **int** dp[][] = **new** **int**[n + 1][k + 1];  **for** (**int** i = 0; i <= n; i++)  **for** (**int** j = 0; j <= k; j++)  dp[i][j] = *inf*;  dp[0][0] = 0;  **for** (**int** i = 1; i <= n; i++)  **for** (**int** j = 1; j <= k; j++)  **for** (**int** m = i - 1; m >= 0; m--)  dp[i][j] = Math.*min*(dp[i][j], dp[m][j - 1] + (a[i - 1] - a[m]) \* (a[i - 1] - a[m]));  **return** dp[n][k];  }  } |

# **Compute Sum of Digits in all numbers from 1 to n**

|  |
| --- |
| /\*Input: n = 5 Output: Sum of digits in numbers from 1 to 5 = 15  Input: n = 12 Output: Sum of digits in numbers from 1 to 12 = 51\*/  **public** **class** ComputeSumOfDigits {  **public** **static** **void** main(String[] args) {  **int** n = 328;  System.***out***.println("Sum of digits in numbers" + " from 1 to " + n + " is " + *sumOfDigitsFrom1ToN*(n));//Sum of digits in numbers from 1 to 328 is 3241  }  **static** **int** sumOfDigitsFrom1ToN(**int** n) {  **int** result = 0;  **for** (**int** x = 1; x <= n; x++)  result += *sumOfDigits*(x);  **return** result;  }  **static** **int** sumOfDigits(**int** x) {  **int** sum = 0;  **while** (x != 0) {  sum += x % 10;  x = x / 10;  }  **return** sum;  }  } |

# **Compute Sum of Digits1**

|  |
| --- |
| **public** **class** ComputeSumOfDigits1 {  **public** **static** **void** main(String[] args) {  **int** n = 328;  System.***out***.println("Sum of digits in numbers " + "from 1 to " + n + " is " + *sumOfDigitsFrom1ToN*(n));//3241  }  **static** **int** sumOfDigitsFrom1ToN(**int** n) {  **if** (n < 10)  **return** (n \* (n + 1) / 2);  **int** d = (**int**) (Math.*log10*(n));  **int** a[] = **new** **int**[d + 1];  a[0] = 0;  a[1] = 45;  **for** (**int** i = 2; i <= d; i++)  a[i] = a[i - 1] \* 10 + 45 \* (**int**) (Math.*ceil*(Math.*pow*(10, i - 1)));  **int** p = (**int**) (Math.*ceil*(Math.*pow*(10, d)));  **int** msd = n / p;  **return** (msd \* a[d] + (msd \* (msd - 1) / 2) \* p + msd \* (1 + n % p) + *sumOfDigitsFrom1ToN*(n % p));  }  } |

# **Count number of binary strings without consecutive 1’s. Given a positive integer N, count all possible distinct binary strings of length N such that there are no consecutive 1’s**

|  |
| --- |
| //Input: N = 2 Output: 3 The 3 strings are 00, 01, 10  **public** **class** consecutive1 {  **public** **static** **void** main(String[] args) {  System.***out***.println(*countStrings*(3));  }  **static** **int** countStrings(**int** n) {  **int** a[] = **new** **int**[n];  **int** b[] = **new** **int**[n];  a[0] = b[0] = 1;  **for** (**int** i = 1; i < n; i++) {  a[i] = a[i - 1] + b[i - 1];  b[i] = a[i - 1];  }  **return** a[n - 1] + b[n - 1];  }  } |

# **Count of N Digit Numbers whose sum of digits equals to given sum**

|  |
| --- |
| Input: n = 2, sum = 2 Output: 2 Explanation: Numbers are 11 and 20  Input: n = 2, sum = 5 Output: 5 Explanation: Numbers are 14, 23, 32, 41 and 50\*/  **public** **class** CounNDigitNumbers {  **public** **static** **void** main(String[] args) {  **int** n = 2, sum = 5;  System.***out***.println(*finalCount*(n, sum));//5  }  **static** **int** countRec(**int** n, **int** sum) {  **if** (n == 0) **return** sum == 0 ? 1 : 0;  **if** (sum == 0) **return** 1;  **int** ans = 0;  **for** (**int** i = 0; i <= 9; i++)  **if** (sum - i >= 0)  ans += *countRec*(n - 1, sum - i);  **return** ans;  }  **static** **int** finalCount(**int** n, **int** sum) {  **int** ans = 0;  **for** (**int** i = 1; i <= 9; i++)  **if** (sum - i >= 0)  ans += *countRec*(n - 1, sum - i);  **return** ans;  }  } |

# **Count of N Digit Numbers1 whose sum of digits equals to given sum using dynamic programming**

|  |
| --- |
| **public** **class** CounNDigitNumbers1 {  **static** **int** *lookup*[][] = **new** **int**[101][501];  **public** **static** **void** main(String[] args) {  **int** n = 2, sum = 5;  System.***out***.println(*finalCount*(n, sum));//5  }  **static** **int** finalCount(**int** n, **int** sum) {  **for** (**int** i = 0; i <= 100; ++i) {  **for** (**int** j = 0; j <= 500; ++j) {  *lookup*[i][j] = -1;  }  }  **int** ans = 0;  **for** (**int** i = 1; i <= 9; i++)  **if** (sum - i >= 0)  ans += *countRec*(n - 1, sum - i);  **return** ans;  }  **static** **int** countRec(**int** n, **int** sum) {  **if** (n == 0) **return** sum == 0 ? 1 : 0;  **if** (*lookup*[n][sum] != -1) **return** *lookup*[n][sum];  **int** ans = 0;  **for** (**int** i = 0; i < 10; i++)  **if** (sum - i >= 0)  ans += *countRec*(n - 1, sum - i);  **return** *lookup*[n][sum] = ans;  }  } |

# **Count All Palindromic Subsequence in a given string. Find how many palindromic subsequence (need not necessarily br distinct) can be formed in a given string. Note that the empty string is not considered as a polindrom.**

|  |
| --- |
| Input : str = "abcd" Output : 4 Explanation :- palindromic subsequence are : "a" ,"b", "c" ,"d" Time Complexity : O(N2)  **public** **class** CountAllPalindromicSubsequence {  **public** **static** **void** main(String[] args) {  String str = "abcb";  System.***out***.println("Total palindromic " + "subsequence are : " + *countPS*(str)); //Total palindromic subsequence are : 6  }  **static** **int** countPS(String str) {  **int** N = str.length();  **int**[][] cps = **new** **int**[N + 1][N + 1];  **for** (**int** i = 0; i < N; i++)  cps[i][i] = 1;  **for** (**int** L = 2; L <= N; L++) {  **for** (**int** i = 0; i < N; i++) {  **int** k = L + i - 1;  **if** (k < N) {  **if** (str.charAt(i) == str.charAt(k))  cps[i][k] = cps[i][k - 1] + cps[i + 1][k] + 1;  **else**  cps[i][k] = cps[i][k - 1] + cps[i + 1][k] - cps[i + 1][k - 1];  }  }  }  **return** cps[0][N - 1];  }  } |

# **Count All Possible Walks from a source to a distination with exactly K edges using simple solution.**

|  |
| --- |
| **public** **class** CountAllPossibleWalks {  **static** **final** **int** ***V*** = 4;  **public** **static** **void** main(String[] args) **throws** java.lang.Exception {  **int** graph[][] = **new** **int**[][] { { 0, 1, 1, 1 }, { 0, 0, 0, 1 }, { 0, 0, 0, 1 }, { 0, 0, 0, 0 } };  **int** u = 0, v = 3, k = 2;  CountAllPossibleWalks p = **new** CountAllPossibleWalks();  System.***out***.println(p.countwalks(graph, u, v, k)); //2  }  **int** countwalks(**int** graph[][], **int** u, **int** v, **int** k) {  **if** (k == 0 && u == v)  **return** 1;  **if** (k == 1 && graph[u][v] == 1)  **return** 1;  **if** (k <= 0)  **return** 0;  **int** count = 0;  **for** (**int** i = 0; i < ***V***; i++)  **if** (graph[u][i] == 1)  count += countwalks(graph, i, v, k - 1);  **return** count;  }  } |

# **Count All Possible Walks1 using dynamic pogramming**

|  |
| --- |
| **public** **class** CountAllPossibleWalks1 {  **static** **final** **int** ***V*** = 4;  **public** **static** **void** main(String[] args) {  **int** graph[][] = **new** **int**[][] { { 0, 1, 1, 1 }, { 0, 0, 0, 1 }, { 0, 0, 0, 1 }, { 0, 0, 0, 0 } };  **int** u = 0, v = 3, k = 2;  CountAllPossibleWalks1 p = **new** CountAllPossibleWalks1();  System.***out***.println(p.countwalks(graph, u, v, k)); //2  }  **int** countwalks(**int** graph[][], **int** u, **int** v, **int** k) {  **int** count[][][] = **new** **int**[***V***][***V***][k + 1];  **for** (**int** e = 0; e <= k; e++) {  **for** (**int** i = 0; i < ***V***; i++) {  **for** (**int** j = 0; j < ***V***; j++) {  count[i][j][e] = 0;  **if** (e == 0 && i == j)  count[i][j][e] = 1;  **if** (e == 1 && graph[i][j] != 0)  count[i][j][e] = 1;  **if** (e > 1) {  **for** (**int** a = 0; a < ***V***; a++)  **if** (graph[i][a] != 0)  count[i][j][e] += count[a][j][e - 1];  }  }  }  }  **return** count[u][v][k];  }  } |

# **Given a teo strings S and T, find count of distinct occurrences of T in S as a subsequence**

|  |
| --- |
| **Input**: S = banana, T = ban **Output**: 3 T appears in S as below three subsequences. [ban], [ba n], [b an]  **Input** : S = geeksforgeeks, T = ge **Output** : 6 T appears in S as below three subsequences.  [ge], [ ge], [g e], [g e] [g e] and [ g e]  Time Complexity : O(m\*n) Auxiliary Space : O(m\*n)  **public** **class** CountDistinctOccurrencesSubsequence {  **public** **static** **void** main(String[] args) {  String T = "ge";  String S = "geeksforgeeks";  System.***out***.println(*findSubsequenceCount*(S, T));//6  }  **static** **int** findSubsequenceCount(String S, String T) {  **int** m = T.length();  **int** n = S.length();  **if** (m > n)  **return** 0;  **int** mat[][] = **new** **int**[m + 1][n + 1];  **for** (**int** i = 1; i <= m; i++)  mat[i][0] = 0;  **for** (**int** j = 0; j <= n; j++)  mat[0][j] = 1;  **for** (**int** i = 1; i <= m; i++) {  **for** (**int** j = 1; j <= n; j++) {  **if** (T.charAt(i - 1) != S.charAt(j - 1))  mat[i][j] = mat[i][j - 1];  **else**  mat[i][j] = mat[i][j - 1] + mat[i - 1][j - 1];  }  }  **return** mat[m][n];  }  } |

# **Recursive java program to find n-th Rencontres number (counting partial derangements)**

|  |
| --- |
| Input : n = 3, k = 0 Output : 2  Since k = 0, no point needs to be on its original position. So derangements are {3, 1, 2} and {2, 3, 1}  **public** **class** CountingPartialDerangements {  **public** **static** **void** main(String[] args) {  **int** n = 7, m = 2;  System.***out***.println(*RencontresNumber*(n, m)); //924  }  **static** **int** RencontresNumber(**int** n, **int** m) {  **if** (n == 0 && m == 0) **return** 1;  **if** (n == 1 && m == 0) **return** 0;  **if** (m == 0)  **return** (n - 1) \* (*RencontresNumber*(n - 1, 0) + *RencontresNumber*(n - 2, 0));  **return** *binomialCoeff*(n, m) \* *RencontresNumber*(n - m, 0);  }  **static** **int** binomialCoeff(**int** n, **int** k) {  **if** (k == 0 || k == n) **return** 1;  **return** *binomialCoeff*(n - 1, k - 1) + *binomialCoeff*(n - 1, k);  }  } |

# **Recursive java program to find n-th Rencontres number using dynamic programming**

|  |
| --- |
| **public** **class** CountingPartialDerangements1 {  **static** **int** *MAX* = 100;  **public** **static** **void** main(String[] args) {  **int** n = 7, m = 2;  **int** C[][] = **new** **int**[*MAX*][*MAX*];  *binomialCoeff*(C, n, m);  System.***out***.println(*RencontresNumber*(C, n, m));//924  }  **static** **void** binomialCoeff(**int** C[][], **int** n, **int** k) {  **for** (**int** i = 0; i <= n; i++) {  **for** (**int** j = 0; j <= Math.*min*(i, k); j++) {  **if** (j == 0 || j == i)  C[i][j] = 1;  **else**  C[i][j] = C[i - 1][j - 1] + C[i - 1][j];  }  }  }  **static** **int** RencontresNumber(**int** C[][], **int** n, **int** m) {  **int** dp[][] = **new** **int**[n + 1][m + 1];  **for** (**int** i = 0; i <= n; i++) {  **for** (**int** j = 0; j <= m; j++) {  **if** (j <= i) {  **if** (i == 0 && j == 0)  dp[i][j] = 1;  **else** **if** (i == 1 && j == 0)  dp[i][j] = 0;  **else** **if** (j == 0)  dp[i][j] = (i - 1) \* (dp[i - 1][0] + dp[i - 2][0]);  **else**  dp[i][j] = C[i][j] \* dp[i - j][0];  }  }  }  **return** dp[n][m];  }  } |

# **Count Number of Ways to reach destination in a Maze.** Given a maze with obstacles, count number of paths to reach rightmost-bottommost cell from tomost-leftmost cell. A cell in given maze has value -1 if it is a blockage or dead end, else 0. From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only

|  |
| --- |
| Input: maze[R][C] = {{0, 0, 0, 0},  {0, -1, 0, 0},  {-1, 0, 0, 0},  {0, 0, 0, 0}};  Output: 4  Time Complexity : O(R x C)\*/  **public** **class** CountNumberOfWays {  **static** **int** *R* = 4;  **static** **int** *C* = 4;  **public** **static** **void** main(String[] args) {  **int** maze[][] = { { 0, 0, 0, 0 }, { 0, -1, 0, 0 }, { -1, 0, 0, 0 }, { 0, 0, 0, 0 } };  System.***out***.println(*countPaths*(maze)); //4  }  **static** **int** countPaths(**int** maze[][]) {  **if** (maze[0][0] == -1)  **return** 0;  **for** (**int** i = 0; i < *R*; i++) {  **if** (maze[i][0] == 0)  maze[i][0] = 1;  **else**  **break**;  }  **for** (**int** i = 1; i < *C*; i++) {  **if** (maze[0][i] == 0)  maze[0][i] = 1;  **else**  **break**;  }  **for** (**int** i = 1; i < *R*; i++) {  **for** (**int** j = 1; j < *C*; j++) {  **if** (maze[i][j] == -1)  **continue**;  **if** (maze[i - 1][j] > 0)  maze[i][j] = (maze[i][j] + maze[i - 1][j]);  **if** (maze[i][j - 1] > 0)  maze[i][j] = (maze[i][j] + maze[i][j - 1]);  }  }  **return** (maze[*R* - 1][*C* - 1] > 0) ? maze[*R* - 1][*C* - 1] : 0;  }  } |

# **Count Number of Ways1 to jump to reach end:** - given an array of numbers where each element represents the max number of jumps that can be made forward from that element. For each array element, count number of ways jumps can be made from that element to reach the end of the array. If an element is 0, then move cannot be made through that element. The element that cannot reach to the end should have a count “-1”.

|  |
| --- |
| Input : {3, 2, 0, 1} Output : 2 1 -1 0 Time Complexity: O (n2) in worst case.  **public** **class** CountNumberOfWays1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 3, 5, 8, 9, 1, 0, 7, 6, 8, 9 };  **int** n = arr.length;  *countWaysToJump*(arr, n);  }  **static** **void** countWaysToJump(**int** arr[], **int** n) {  **int** count\_jump[] = **new** **int**[n];  Arrays.*fill*(count\_jump, 0);  **for** (**int** i = n - 2; i >= 0; i--) {  **if** (arr[i] >= n - i - 1)  count\_jump[i]++;  **for** (**int** j = i + 1; j < n - 1 && j <= arr[i] + i; j++)  **if** (count\_jump[j] != -1)  count\_jump[i] += count\_jump[j];  **if** (count\_jump[i] == 0)  count\_jump[i] = -1;  }  **for** (**int** i = 0; i < n; i++)  System.***out***.print(count\_jump[i] + " "); //52 52 28 16 8 -1 -1 4 2 1 0  }  } |

# **Count Possible Decodings of a given digit sequence.** Let 1 represent ‘A’ , 2 represent ‘B’ etc. given a digit sequence , count the number of possible decoding of the given digit sequence.

|  |
| --- |
| Input: digits[] = "121" Output: 3 The possible Decodings are "ABA", "AU", "LA"  **public** **class** CountPossibleDecodings {  **public** **static** **void** main(String[] args) {  **char** digits[] = { '1', '2', '3', '4' };  **int** n = digits.length;  System.***out***.print("Count is %d", *countDecoding*(digits, n)); //Count is 3  }  **static** **int** countDecoding(**char**[] digits, **int** n) {  **if** (n == 0 || n == 1) **return** 1;  **int** count = 0;  **if** (digits[n - 1] > '0')  count = *countDecoding*(digits, n - 1);  **if** (digits[n - 2] == '1' || (digits[n - 2] == '2' && digits[n - 1] < '7'))  count += *countDecoding*(digits, n - 2);  **return** count;  }  } |

# **Count Possible Ways to construct buildings: -** given an input number of sections and each section has 2 plots on either sides of the road. Find all possible ways to construct building in the lopts such that there is a space between any 2 buildings.

|  |
| --- |
| Time complexity: O(N) Auxiliary Space: O(1)  N = 1 Output = 4  N = 3 Output = 25  **public** **class** CountPossibleWays {  **public** **static** **void** main(String[] args) {  **int** N = 3;  System.***out***.println("Count of ways for " + N + " sections is " + *countWays*(N)); //Count of ways for 3 sections is 25  }  **static** **int** countWays(**int** N) {  **if** (N == 1) **return** 4; // 2 for one side and 4 for two sides  **int** countB = 1, countS = 1, prev\_countB, prev\_countS;  **for** (**int** i = 2; i <= N; i++) {  prev\_countB = countB;  prev\_countS = countS;  countS = prev\_countB + prev\_countS;  countB = prev\_countS;  }  **int** result = countS + countB;  **return** (result \* result);  }  } |

# **Egg Dropping: -** the following is the description of the instance of this famous puzzle involving n=2 egg and a building with k = 36 floors.

|  |
| --- |
| Time Complexity: O(nk^2) Auxiliary Space: O(nk) Minimum number of trials in worst case with 2 eggs and 10 floors is 4  **public** **class** EggDropping {  **public** **static** **void** main(String[] args) {  **int** n = 2, k = 10;  System.***out***.println(  "Minimum number of trials in worst case with " + n + " eggs and " + k + " floors is " + *eggDrop*(n, k));  }  **static** **int** eggDrop(**int** n, **int** k) {  **int** eggFloor[][] = **new** **int**[n + 1][k + 1];  **int** res;  **int** i, j, x;  **for** (i = 1; i <= n; i++) {  eggFloor[i][1] = 1;  eggFloor[i][0] = 0;  }  **for** (j = 1; j <= k; j++)  eggFloor[1][j] = j;  **for** (i = 2; i <= n; i++) {  **for** (j = 2; j <= k; j++) {  eggFloor[i][j] = Integer.***MAX\_VALUE***;  **for** (x = 1; x <= j; x++) {  res = 1 + *max*(eggFloor[i - 1][x - 1], eggFloor[i][j - x]);  **if** (res < eggFloor[i][j])  eggFloor[i][j] = res;  }  }  }  **return** eggFloor[n][k];  }  **static** **int** max(**int** a, **int** b) {  **return** (a > b) ? a : b;  }  } |

# **Find All Distinct Subset sum of an array: -** given a set of intergers, find distinct sum that can be generated from the susets of a given sets and print them in an increasing order. It is given that sum of array elements is small.

|  |
| --- |
| Input : arr[] = {1, 2, 3} Output : 0 1 2 3 4 5 6 The naive solution  **public** **class** FindAllDistinctSubset {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 4, 5, 6 };  **int** n = arr.length;  *printDistSum*(arr, n);  }  **static** **void** printDistSum(**int** arr[], **int** n) {  HashSet<Integer> s = **new** HashSet<>();  *distSumRec*(arr, n, 0, 0, s);  **for** (**int** i : s)  System.***out***.print(i + " "); //0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20  }  **static** **void** distSumRec(**int** arr[], **int** n, **int** sum, **int** currindex, HashSet<Integer> s) {  **if** (currindex > n)  **return**;  **if** (currindex == n) {  s.add(sum);  **return**;  }  *distSumRec*(arr, n, sum + arr[currindex], currindex + 1, s);  *distSumRec*(arr, n, sum, currindex + 1, s);  }  } |

# **Find All Distinct Subset1 using dynamic programming**

|  |
| --- |
| //using Dynamic Programming  **public** **class** FindAllDistinctSubset1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 4, 5, 6 };  **int** n = arr.length;  *printDistSum*(arr, n);  }  **static** **void** printDistSum(**int** arr[], **int** n) {  **int** sum = 0;  **for** (**int** i = 0; i < n; i++)  sum += arr[i];  **boolean**[][] dp = **new** **boolean**[n + 1][sum + 1];  **for** (**int** i = 0; i <= n; i++)  dp[i][0] = **true**;  **for** (**int** i = 1; i <= n; i++) {  dp[i][arr[i - 1]] = **true**;  **for** (**int** j = 1; j <= sum; j++) {  **if** (dp[i - 1][j] == **true**) {  dp[i][j] = **true**;  dp[i][j + arr[i - 1]] = **true**;  }  }  }  **for** (**int** j = 0; j <= sum; j++)  **if** (dp[n][j] == **true**)  System.***out***.print(j + " "); //0 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20  }  } |

# **Find Maximum Dot Product of two arrays with insertion of 0’s.** Given two arrays of positive integers of size m and n where m>n. we need to maximize the dot product by inserting 0’s in the second array but we cannot disturb the order of elements.

|  |
| --- |
| Input : A[] = {2, 3 , 1, 7, 8} B[] = {3, 6, 7} Output : 107 Time Complexity : O(nm)  **public** **class** FindMaximumDotProduct {  **public** **static** **void** main(String[] args) {  **int** A[] = { 2, 3, 1, 7, 8 };  **int** B[] = { 3, 6, 7 };  **int** m = A.length;  **int** n = B.length;  System.***out***.print(*MaxDotProduct*(A, B, m, n));//107  }  **static** **int** MaxDotProduct(**int** A[], **int** B[], **int** m, **int** n) {  **int** dp[][] = **new** **int**[n + 1][m + 1];  **for** (**int**[] row : dp)  Arrays.*fill*(row, 0);  **for** (**int** i = 1; i <= n; i++)  **for** (**int** j = i; j <= m; j++)  dp[i][j] = Math.*max*((dp[i - 1][j - 1] + (A[j - 1] \* B[i - 1])), dp[i][j - 1]);  **return** dp[n][m];  }  } |

# **Find Minimum Sum such that one of every three consecutive elements is taken given an array of n non-negative numbers, the task is to find the minimum sum of elements such that at least one element is picked out of every 3 consecutive elements in the array.**

|  |
| --- |
| Input : arr[] = {1, 2, 3} Output : 1 Time Complexity : O(n) Auxiliary Space : O(n)  **public** **class** FindMinimumSum {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 20, 2, 10, 1 };  **int** n = arr.length;  System.***out***.println("Min Sum is " + *findMinSum*(arr, n)); //Min Sum is 4  }  **static** **int** findMinSum(**int** arr[], **int** n) {  **int** sum[] = **new** **int**[n];  sum[0] = arr[0];  sum[1] = arr[1];  sum[2] = arr[2];  **for** (**int** i = 3; i < n; i++)  sum[i] = arr[i] + *minimum*(sum[i - 3], sum[i - 2], sum[i - 1]);  **return** *minimum*(sum[n - 1], sum[n - 2], sum[n - 3]);  }  **static** **int** minimum(**int** a, **int** b, **int** c) {  **return** Math.*min*(Math.*min*(a, b), c);  }  } |

# **Find Number Endless Points: -** give a binary NxN matrix, we need to find the total number of matrix positions from which there is an endless path.

|  |
| --- |
| Input : 0 1 0  1 1 1  0 1 1  Output : 4  **public** **class** FindNumberEndlessPoints {  **static** **final** **int** ***MAX*** = 100;  **public** **static** **void** main(String[] args) {  **boolean** input[][] = { { **true**, **false**, **true**, **true** }, { **false**, **true**, **true**, **true** }, { **true**, **true**, **true**, **true** },  { **false**, **true**, **true**, **false** } };  **int** n = 4;  System.***out***.print(*countEndless*(input, n)); //5  }  **static** **int** countEndless(**boolean** input[][], **int** n) {  **boolean** row[][] = **new** **boolean**[n][n];  **boolean** col[][] = **new** **boolean**[n][n];  **for** (**int** j = 0; j < n; j++) {  **boolean** isEndless = **true**;  **for** (**int** i = n - 1; i >= 0; i--) {  **if** (input[i][j] == **false**)  isEndless = **false**;  col[i][j] = isEndless;  }  }  **for** (**int** i = 0; i < n; i++) {  **boolean** isEndless = **true**;  **for** (**int** j = n - 1; j >= 0; j--) {  **if** (input[i][j] == **false**)  isEndless = **false**;  row[i][j] = isEndless;  }  }  **int** ans = 0;  **for** (**int** i = 0; i < n; i++)  **for** (**int** j = 1; j < n; j++)  **if** (row[i][j] && col[i][j])  ans++;  **return** ans;  }  } |

# **Floyd Warshall Algorithm is for solving the all pairs shortest path problem.** The problem is to find shortest distances between every pair of vertices in a given dege weighted directed graph.

|  |
| --- |
| Time Complexity: O(V^3)  Input: graph[][] = { {0, 5, INF, 10},  {INF, 0, 3, INF},  {INF, INF, 0, 1},  {INF, INF, INF, 0} }  Output: Shortest distance matrix  0 5 8 9  INF 0 3 4  INF INF 0 1  INF INF INF 0  Let us create the following weighted graph  > 10  > (0)------->(3)  > | /|\  > 5 | |  > | | 1  > \|/ |  > (1)------->(2)  > 3  public class FloydWarshallAlgorithm {  final static int INF = 99999, V = 4;  public static void main(String[] args) {  int graph[][] = { { 0, 5, INF, 10 }, { INF, 0, 3, INF }, { INF, INF, 0, 1 }, { INF, INF, INF, 0 } };  FloydWarshallAlgorithm a = new FloydWarshallAlgorithm();  a.floydWarshall(graph);  }  void floydWarshall(int graph[][]) {  int dist[][] = new int[V][V];  int i, j, k;  for (i = 0; i < V; i++)  for (j = 0; j < V; j++)  dist[i][j] = graph[i][j];  for (k = 0; k < V; k++) {  for (i = 0; i < V; i++) {  for (j = 0; j < V; j++) {  if (dist[i][k] + dist[k][j] < dist[i][j])  dist[i][j] = dist[i][k] + dist[k][j];  }  }  }  printSolution(dist);  }  void printSolution(int dist[][]) {  System.out.println("Following matrix shows the shortest " + "distances between every pair of vertices");  for (int i = 0; i < V; ++i) {  for (int j = 0; j < V; ++j) {  if (dist[i][j] == INF)  System.out.print("INF ");  else  System.out.print(dist[i][j] + " ");  }  System.out.println();  }  }  }  /\*Following matrix shows the shortest distances between every pair of vertices  0 5 8 9  INF 0 3 4  INF INF 0 1  INF INF INF 0 \*/ |

# **Friends Pairing Problem: -** given n friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out total number of ways in which friends can remain single or can be paired up.

|  |
| --- |
| Input : n = 3 Output : 4 Time Complexity : O(n) Auxiliary Space : O(n)  **public** **class** FriendsPairingProblem {  **public** **static** **void** main(String[] args) {  **int** n = 4;  System.***out***.println(*countFriendsPairings*(n));  }  **static** **int** countFriendsPairings(**int** n) {  **int** dp[] = **new** **int**[n + 1];  **for** (**int** i = 0; i <= n; i++) {  **if** (i <= 2)  dp[i] = i;  **else**  dp[i] = dp[i - 1] + (i - 1) \* dp[i - 2];  }  **return** dp[n];  }  } |

# **Gold Mine Problem: -** given a gold mine of n\*m dimensions. Each field in this mine conatins a positive integer which is the amount of gold in tons. Initially the miner is at first column but can be at any row. He can move only (right->, right up/, right down\) that is from a given cell, the miner can move to cell diagonally up towards the right or diagonally down towards the right. Find the maximum amount of gold he can collect.

|  |
| --- |
| Input : mat[][] = {{1, 3, 3},  {2, 1, 4},  {0, 6, 4}};  Output : 12 {(1,0)->(2,1)->(2,2)} Time Complexity :O(m\*n) Space Complexity :O(m\*n)  **public** **class** GoldMineProblem {  **static** **final** **int** ***MAX*** = 100;  **public** **static** **void** main(String[] args) {  **int** gold[][] = { { 1, 3, 1, 5 }, { 2, 2, 4, 1 }, { 5, 0, 2, 3 }, { 0, 6, 1, 2 } };  **int** m = 4, n = 4;  System.***out***.print(*getMaxGold*(gold, m, n)); //16  }  **static** **int** getMaxGold(**int** gold[][], **int** m, **int** n) {  **int** goldTable[][] = **new** **int**[m][n];  **for** (**int**[] rows : goldTable)  Arrays.*fill*(rows, 0);  **for** (**int** col = n - 1; col >= 0; col--) {  **for** (**int** row = 0; row < m; row++) {  **int** right = (col == n - 1) ? 0 : goldTable[row][col + 1];  **int** right\_up = (row == 0 || col == n - 1) ? 0 : goldTable[row - 1][col + 1];  **int** right\_down = (row == m - 1 || col == n - 1) ? 0 : goldTable[row + 1][col + 1];  goldTable[row][col] = gold[row][col] + Math.*max*(right, Math.*max*(right\_up, right\_down));  ;  }  }  **int** res = goldTable[0][0];  **for** (**int** i = 1; i < m; i++)  res = Math.*max*(res, goldTable[i][0]);  **return** res;  }  } |

# **Knapsack Problem: -** given wights and values of n items, put these items in a knapsack of capacity W to get the maximum toral value in the knapsack. Overlapping subproblems.

|  |
| --- |
| **public** **class** KnapsackProblem {  **public** **static** **void** main(String[] args) {  **int** val[] = **new** **int**[] { 60, 100, 120 };  **int** wt[] = **new** **int**[] { 10, 20, 30 };  **int** W = 50;  **int** n = val.length;  System.***out***.println(*knapSack*(W, wt, val, n)); //220  }  **static** **int** knapSack(**int** W, **int** wt[], **int** val[], **int** n) {  **if** (n == 0 || W == 0) **return** 0;  **if** (wt[n - 1] > W) **return** *knapSack*(W, wt, val, n - 1);  **else**  **return** *max*(val[n - 1] + *knapSack*(W - wt[n - 1], wt, val, n - 1), *knapSack*(W, wt, val, n - 1));  }  **static** **int** max(**int** a, **int** b) {  **return** (a > b) ? a : b;  }  } |

# **Knapsack Problem1 dynamic programming based implementation**

|  |
| --- |
| Time Complexity: O (nW) where n is the number of items and W is the capacity of knapsack.  **public** **class** KnapsackProblem1 {  **public** **static** **void** main(String[] args) {  **int** val[] = **new** **int**[] { 60, 100, 120 };  **int** wt[] = **new** **int**[] { 10, 20, 30 };  **int** W = 50;  **int** n = val.length;  System.***out***.println(*knapSack*(W, wt, val, n)); //220  }  **static** **int** knapSack(**int** W, **int** wt[], **int** val[], **int** n) {  **int** i, w;  **int** K[][] = **new** **int**[n + 1][W + 1];  **for** (i = 0; i <= n; i++) {  **for** (w = 0; w <= W; w++) {  **if** (i == 0 || w == 0)  K[i][w] = 0;  **else** **if** (wt[i - 1] <= w)  K[i][w] = *max*(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);  **else**  K[i][w] = K[i - 1][w];  }  }  **return** K[n][W];  }  **static** **int** max(**int** a, **int** b) {  **return** (a > b) ? a : b;  }  } |

# **Knight Remain Chessboard: -** probability of knight to remain in the chessboard.given an NxN chessboard and a knight at position(x, y). The knoght has to take exactly K steps, where at each step it chooses any of the 8 directions uniformly at random.what is the probability that the knight remains in the chessboard after taking K steps, with the condition that it can’t enter the board again once it leaves it.

|  |
| --- |
| Let's take: 8x8 chessboard, initial position of the knight: (0, 0), number of steps: 1 At each step, the Knight has 8 different positions to choose from. If it starts from (0, 0), after taking one step it will lie inside the board only at 2 out of 8 positions, and will lie outside at other positions. So, the probability is 2/8 = 0.25  Time Complexity: O(NxNxKx8) Space Complexity: O(NxNxK)  **public** **class** KnightRemainChessboard {  **static** **final** **int** ***N*** = 8;  **static** **int** *dx*[] = { 1, 2, 2, 1, -1, -2, -2, -1 };  **static** **int** *dy*[] = { 2, 1, -1, -2, -2, -1, 1, 2 };  **public** **static** **void** main(String[] args) {  **int** K = 3;  System.***out***.println(*findProb*(0, 0, K)); //0.125  }  **static** **double** findProb(**int** start\_x, **int** start\_y, **int** steps) {  **double** dp1[][][] = **new** **double**[***N***][***N***][***N***];  **for** (**int** i = 0; i < ***N***; ++i)  **for** (**int** j = 0; j < ***N***; ++j)  dp1[i][j][0] = 1;  **for** (**int** s = 1; s <= steps; ++s) {  **for** (**int** x = 0; x < ***N***; ++x) {  **for** (**int** y = 0; y < ***N***; ++y) {  **double** prob = 0.0;  **for** (**int** i = 0; i < 8; ++i) {  **int** nx = x + *dx*[i];  **int** ny = y + *dy*[i];  **if** (*inside*(nx, ny))  prob += dp1[nx][ny][s - 1] / 8.0;  }  dp1[x][y][s] = prob;  }  }  }  **return** dp1[start\_x][start\_y][steps];  }  **static** **boolean** inside(**int** x, **int** y) {  **return** (x >= 0 && x < ***N*** && y >= 0 && y < ***N***);  }  } |

# **Largest Independent Set: -** given a binary tree, find size of the largest independent set in it. A suset of all tree nodes is an independent set if there is no edge between any two nodes of the subset.

|  |
| --- |
| Time Complexity: O (n) where n is the number of nodes in given Binary tree. For example, consider the following binary tree. The largest independent set (LIS) is {10, 40, 60, 70, and 80} and size of the LIS is 5.  **public** **class** LargestIndependentSet {  **static** **class** node {  **int** data, liss;  node left, right;  **public** node(**int** data) {  **this**.data = data;  **this**.liss = 0;  }  }  **public** **static** **void** main(String[] args) {  node root = **new** node(20);  root.left = **new** node(8);  root.left.left = **new** node(4);  root.left.right = **new** node(12);  root.left.right.left = **new** node(10);  root.left.right.right = **new** node(14);  root.right = **new** node(22);  root.right.right = **new** node(25);  System.***out***.println("Size of the Largest Independent Set is " + *liss*(root)); //Size of the Largest Independent Set is 5  }  **static** **int** liss(node root) {  **if** (root == **null**)  **return** 0;  **if** (root.liss != 0)  **return** root.liss;  **if** (root.left == **null** && root.right == **null**)  **return** root.liss = 1;  **int** liss\_excl = *liss*(root.left) + *liss*(root.right);  **int** liss\_incl = 1;  **if** (root.left != **null**) {  liss\_incl += (*liss*(root.left.left) + *liss*(root.left.right));  }  **if** (root.right != **null**) {  liss\_incl += (*liss*(root.right.left) + *liss*(root.right.right));  }  **return** root.liss = Math.*max*(liss\_excl, liss\_incl);  }  } |

# **Length of Longest Substring without repeating characters.** Given a sting, find the length of the longest substring without repeating characters.

|  |
| --- |
| For example, the longest substrings without repeating characters for “ABDEFGABEF” are “BDEFGA” and “DEFGAB”, with length 6. Time Complexity: O(n + d) Auxiliary Space: O(d)  **public** **class** LengthLongestSubstring {  **static** **final** **int** ***NO\_OF\_CHARS*** = 256;  **public** **static** **void** main(String[] args) {  String str = "ABDEFGABEF";  System.***out***.println("The input string is " + str); //ABDEFGABEF  **int** len = *longestUniqueSubsttr*(str);  System.***out***.println("The length of " + "the longest non repeating character is " + len); //6  }  **static** **int** longestUniqueSubsttr(String str) {  **int** n = str.length();  **int** cur\_len = 1;  **int** max\_len = 1;  **int** prev\_index;  **int** i;  **int** visited[] = **new** **int**[***NO\_OF\_CHARS***];  **for** (i = 0; i < ***NO\_OF\_CHARS***; i++) {  visited[i] = -1;  }  visited[str.charAt(0)] = 0;  **for** (i = 1; i < n; i++) {  prev\_index = visited[str.charAt(i)];  **if** (prev\_index == -1 || i - cur\_len > prev\_index)  cur\_len++;  **else** {  **if** (cur\_len > max\_len)  max\_len = cur\_len;  cur\_len = i - prev\_index;  }  visited[str.charAt(i)] = i;  }  **if** (cur\_len > max\_len)  max\_len = cur\_len;  **return** max\_len;  }  } |

# **Longest Alternating Subsequence**

|  |
| --- |
| Input: arr[] = {1, 5, 4} Output: 3 The whole arrays is of the form x1 < x2 > x3  Input: arr [] = {10, 22, 9, 33, 49, 50, 31, 60} Output: 6 The subsequences {10, 22, 9, 33, 31, 60} or {10, 22, 9, 49, 31, 60} or {10, 22, 9, 50, 31, 60} are longest subsequence of length 6.  Time Complexity: O(n2) Auxiliary Space: O(n)  **public** **class** LongestAlternatingSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 22, 9, 33, 49, 50, 31, 60 };  **int** n = arr.length;  System.***out***.println("Length of Longest " + "alternating subsequence is " + *zzis*(arr, n)); 6  }  **static** **int** zzis(**int** arr[], **int** n) {  **int** las[][] = **new** **int**[n][2];  **for** (**int** i = 0; i < n; i++)  las[i][0] = las[i][1] = 1;  **int** res = 1;  **for** (**int** i = 1; i < n; i++) {  **for** (**int** j = 0; j < i; j++) {  **if** (arr[j] < arr[i] && las[i][0] < las[j][1] + 1)  las[i][0] = las[j][1] + 1;  **if** (arr[j] > arr[i] && las[i][1] < las[j][0] + 1)  las[i][1] = las[j][0] + 1;  }  **if** (res < Math.*max*(las[i][0], las[i][1]))  res = Math.*max*(las[i][0], las[i][1]);  }  **return** res;  }  } |

# **Longest Arithmetic Progression: -** given a set of numbers, find the length of the longest arithmetic progession in it.

|  |
| --- |
| set[] = {1, 7, 10, 15, 27, 29} output = 3 The longest arithmetic progression is {1, 15, 29}  set[] = {5, 10, 15, 20, 25, 30} output = 6 The whole set is in AP  Time Complexity: O(n2) Auxiliary Space: O(n2)  **public** **class** LongestArithmeticProgression {  **public** **static** **void** main(String[] args) {  **int** set1[] = { 1, 7, 10, 13, 14, 19 };  **int** n1 = set1.length;  System.***out***.println(*lenghtOfLongestAP*(set1, n1)); //4  **int** set2[] = { 1, 7, 10, 15, 27, 29 };  **int** n2 = set2.length;  System.***out***.println(*lenghtOfLongestAP*(set2, n2)); //3  **int** set3[] = { 2, 4, 6, 8, 10 };  **int** n3 = set3.length;  System.***out***.println(*lenghtOfLongestAP*(set3, n3)); //5  }  **static** **int** lenghtOfLongestAP(**int** set[], **int** n) {  **if** (n <= 2) **return** n;  **int** L[][] = **new** **int**[n][n];  **int** llap = 2;  **for** (**int** i = 0; i < n; i++)  L[i][n - 1] = 2;  **for** (**int** j = n - 2; j >= 1; j--) {  **int** i = j - 1, k = j + 1;  **while** (i >= 0 && k <= n - 1) {  **if** (set[i] + set[k] < 2 \* set[j])  k++;  **else** **if** (set[i] + set[k] > 2 \* set[j]) {  L[i][j] = 2;  i--;  }  **else** {  L[i][j] = L[j][k] + 1;  llap = Math.*max*(llap, L[i][j]);  i--;  k++;  }  }  **while** (i >= 0) {  L[i][j] = 2;  i--;  }  }  **return** llap;  }  } |

# **Longest Bitonic Subsequence: -** given an array arr [0…n-1] containing n positive integers, a subsequence of arr [] is called Bitnonic if it is first increasing, then decreasing. Write a function that takes an array as argument and return the length of the longest bitonic subsequence.

|  |
| --- |
| Input arr[] = {1, 11, 2, 10, 4, 5, 2, 1}; Output: 6 (A Longest Bitonic Subsequence of length 6 is 1, 2, 10, 4, 2, 1)  Time Complexity: O(n^2) Auxiliary Space: O(n)  **public** **class** LongestBitonicSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15 };  **int** n = arr.length;  System.***out***.println("Length of LBS is " + *lbs*(arr, n)); //Length of LBS is 7  }  **static** **int** lbs(**int** arr[], **int** n) {  **int** i, j;  **int**[] lis = **new** **int**[n];  **for** (i = 0; i < n; i++)  lis[i] = 1;  **for** (i = 1; i < n; i++)  **for** (j = 0; j < i; j++)  **if** (arr[i] > arr[j] && lis[i] < lis[j] + 1)  lis[i] = lis[j] + 1;  **int**[] lds = **new** **int**[n];  **for** (i = 0; i < n; i++)  lds[i] = 1;  **for** (i = n - 2; i >= 0; i--)  **for** (j = n - 1; j > i; j--)  **if** (arr[i] > arr[j] && lds[i] < lds[j] + 1)  lds[i] = lds[j] + 1;  **int** max = lis[0] + lds[0] - 1;  **for** (i = 1; i < n; i++)  **if** (lis[i] + lds[i] - 1 > max)  max = lis[i] + lds[i] - 1;  **return** max;  }  } |

# **Longest Common Increasing Subsequence: -** given two array, find the length of the longest common increasing subsequence and print one of such sequences.

|  |
| --- |
| we consider two arrays – arr1[] = {3, 4, 9, 1} and arr2[] = {5, 3, 8, 9, 10, 2, 1} Our answer would be {3, 9}  **public** **class** LongestCommonIncreasingSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 3, 4, 9, 1 };  **int** arr2[] = { 5, 3, 8, 9, 10, 2, 1 };  **int** n = arr1.length;  **int** m = arr2.length;  System.***out***.println("Length of LCIS is " + *LCIS*(arr1, n, arr2, m)); //Length of LCIS is 2  }  **static** **int** LCIS(**int** arr1[], **int** n, **int** arr2[], **int** m) {  **int** table[] = **new** **int**[m];  **for** (**int** j = 0; j < m; j++)  table[j] = 0;  **for** (**int** i = 0; i < n; i++) {  **int** current = 0;  **for** (**int** j = 0; j < m; j++) {  **if** (arr1[i] == arr2[j])  **if** (current + 1 > table[j])  table[j] = current + 1;  **if** (arr1[i] > arr2[j])  **if** (table[j] > current)  current = table[j];  }  }  **int** result = 0;  **for** (**int** i = 0; i < m; i++)  **if** (table[i] > result)  result = table[i];  **return** result;  }  } |

# **Longest Common Increasing Subsequence1**

|  |
| --- |
| Time Complexity : O(m\*n) Auxiliary Space : O(m)  The LCIS is : 3 9 Length of LCIS is 2  public class LongestCommonIncreasingSubsequence1 {  public static void main(String[] args) {  int arr1[] = { 3, 4, 9, 1 };  int arr2[] = { 5, 3, 8, 9, 10, 2, 1 };  int n = arr1.length;  int m = arr2.length;  System.out.println("\nLength of LCIS is " + LCIS(arr1, n, arr2, m));  }  static int LCIS(int arr1[], int n, int arr2[], int m) {  int table[] = new int[m];  int parent[] = new int[m];  for (int j = 0; j < m; j++)  table[j] = 0;  for (int i = 0; i < n; i++) {  int current = 0, last = -1;  for (int j = 0; j < m; j++) {  if (arr1[i] == arr2[j]) {  if (current + 1 > table[j]) {  table[j] = current + 1;  parent[j] = last;  }  }  if (arr1[i] > arr2[j]) {  if (table[j] > current) {  current = table[j];  last = j;  }  }  }  }  int result = 0, index = -1;  for (int i = 0; i < m; i++) {  if (table[i] > result) {  result = table[i];  index = i;  }  }  int lcis[] = new int[result];  for (int i = 0; index != -1; i++) {  lcis[i] = arr2[index];  index = parent[index];  }  System.out.print("The LCIS is : ");  for (int i = result - 1; i >= 0; i--)  System.out.print(lcis[i] + " ");  return result;  }  } |

# **Longest Common Subsequence of three strings:** - given 3 strings of all having length<100, the task is to find the longest common subsequence in all the three given sequences.

|  |
| --- |
| Input : str1 = "geeks" str2 = "geeksfor" str3 = "geeksforgeeks" Output : 5 Longest common subsequence is "geeks"i.e., length = 5  **public** **class** LongestCommonSubsequence {  **public** **static** **void** main(String[] args) {  String X = "AGGT12";  String Y = "12TXAYB";  String Z = "12XBA";  **int** m = X.length();  **int** n = Y.length();  **int** o = Z.length();  System.***out***.println("Length of LCS is " + *lcsOf3*(X, Y, Z, m, n, o)); //2  }  **static** **int** lcsOf3(String X, String Y, String Z, **int** m, **int** n, **int** o) {  **int**[][][] L = **new** **int**[m + 1][n + 1][o + 1];  **for** (**int** i = 0; i <= m; i++) {  **for** (**int** j = 0; j <= n; j++) {  **for** (**int** k = 0; k <= o; k++) {  **if** (i == 0 || j == 0 || k == 0)  L[i][j][k] = 0;  **else** **if** (X.charAt(i - 1) == Y.charAt(j - 1) && X.charAt(i - 1) == Z.charAt(k - 1))  L[i][j][k] = L[i - 1][j - 1][k - 1] + 1;  **else**  L[i][j][k] = Math.*max*(Math.*max*(L[i - 1][j][k], L[i][j - 1][k]), L[i][j][k - 1]);  }  }  }  **return** L[m][n][o];  }  } |

# **Longest Consecutive Path:-** find length of the longest consecutive path from a given starting character.given a matrix of characters.find length of the longest path from a given character,such that all characters in the path are consecutive to each other,i.e.,every character in the path is next to previous in alphabetical order. It is allowed to move in all 8 directions from a cell.

|  |
| --- |
| Input: mat[][] = { {a, c, d},  {h, b, e},  {i, g, f}}  Starting Point = 'e' Output: 5  **public** **class** LongestConsecutivePath {  **static** **int** *x*[] = {0, 1, 1, -1, 1, 0, -1, -1};  **static** **int** *y*[] = { 1, 0, 1, 1, -1, -1, 0, -1 };  **static** **int** *R* = 3;  **static** **int** *C* = 3;  **static** **int** *dp*[][] = **new** **int**[*R*][*C*];  **public** **static** **void** main(String[] args) {  **char** mat[][] = { { 'a', 'c', 'd' }, { 'h', 'b', 'a' }, { 'i', 'g', 'f' } };  System.***out***.println(*getLen*(mat, 'a')); //4  System.***out***.println(*getLen*(mat, 'e')); //0  System.***out***.println(*getLen*(mat, 'b')); //3  System.***out***.println(*getLen*(mat, 'f')); //4  }  **static** **int** getLen(**char** mat[][], **char** s) {  **for** (**int** i = 0; i < *R*; ++i)  **for** (**int** j = 0; j < *C*; ++j)  *dp*[i][j] = -1;  **int** ans = 0;  **for** (**int** i = 0; i < *R*; i++) {  **for** (**int** j = 0; j < *C*; j++) {  **if** (mat[i][j] == s) {  **for** (**int** k = 0; k < 8; k++)  ans = Math.*max*(ans, 1 + *getLenUtil*(mat, i + *x*[k], j + *y*[k], s));  }  }  }  **return** ans;  }  **static** **int** getLenUtil(**char** mat[][], **int** i, **int** j, **char** prev) {  **if** (!*isvalid*(i, j) || !*isadjacent*(prev, mat[i][j]))  **return** 0;  **if** (*dp*[i][j] != -1)  **return** *dp*[i][j];  **int** ans = 0; // Initialize answer  **for** (**int** k = 0; k < 8; k++)  ans = Math.*max*(ans, 1 + *getLenUtil*(mat, i + *x*[k], j + *y*[k], mat[i][j]));  **return** *dp*[i][j] = ans;  }  **static** **boolean** isadjacent(**char** prev, **char** curr) {  **return** ((curr - prev) == 1);  }  **static** **boolean** isvalid(**int** i, **int** j) {  **if** (i < 0 || j < 0 || i >= *R* || j >= *C*)  **return** **false**;  **return** **true**;  }  } |

# **Longest Even Length Substring** :- such that sum of first and second half is same.given a string’str’ of digits, find length of the longest substring of ‘str’ , such that the length of the substring is 2k digit and sum of left k digits is equal to the sum of the right k digits.

|  |
| --- |
| Input: str = "123123" Output: 6  Input: str = "1538023" Output: 4  **public** **class** LongestEvenLengthSubstring {  **public** **static** **void** main(String[] args) {  String str = "1538023";  System.***out***.println("Length of the substring is " + *findLength*(str)); // 4  }  **static** **int** findLength(String str) {  **int** n = str.length();  **int** maxlen = 0;  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = i + 1; j < n; j += 2) {  **int** length = j - i + 1;  **int** leftsum = 0, rightsum = 0;  **for** (**int** k = 0; k < length / 2; k++) {  leftsum += (str.charAt(i + k) - '0');  rightsum += (str.charAt(i + k + length / 2) - '0');  }  **if** (leftsum == rightsum && maxlen < length)  maxlen = length;  }  }  **return** maxlen;  }  } |

# **Longest Even Length Substring1 using dynamic programming [O (n2) and O (n2) extra space]**

|  |
| --- |
| **public** **class** LongestEvenLengthSubstring1 {  **public** **static** **void** main(String[] args) {  String str = "153803";  System.***out***.println("Length of the substring is " + *findLength*(str)); // 4  }  **static** **int** findLength(String str) {  **int** n = str.length();  **int** maxlen = 0;  **int** sum[][] = **new** **int**[n][n];  **for** (**int** i = 0; i < n; i++)  sum[i][i] = str.charAt(i) - '0';  **for** (**int** len = 2; len <= n; len++) {  **for** (**int** i = 0; i < n - len + 1; i++) {  **int** j = i + len - 1;  **int** k = len / 2;  sum[i][j] = sum[i][j - k] + sum[j - k + 1][j];  **if** (len % 2 == 0 && sum[i][j - k] == sum[(j - k + 1)][j] && len > maxlen)  maxlen = len;  }  }  **return** maxlen;  }  } |

# **Longest Even Length Substring2 using dynamic programming [A O (n2) and O (n) extra space]**

|  |
| --- |
| **public** **class** LongestEvenLengthSubstring2 {  **public** **static** **void** main(String[] args) {  String str = "123123";  System.***out***.println("Length of the substring is " + *findLength*(str, str.length())); 6  }  **static** **int** findLength(String str, **int** n) {  **int** sum[] = **new** **int**[n + 1];  sum[0] = 0;  **for** (**int** i = 1; i <= n; i++)  sum[i] = (sum[i - 1] + str.charAt(i - 1) - '0');  **int** ans = 0;  **for** (**int** len = 2; len <= n; len += 2) {  **for** (**int** i = 0; i <= n - len; i++) {  **int** j = i + len - 1;  **if** (sum[i + len / 2] - sum[i] == sum[i + len] - sum[i + len / 2])  ans = Math.*max*(ans, len);  }  }  **return** ans;  }  } |

# **Longest Palindromic Subsequence: -** given a sequence, find the length of the longest palindromic subsequence in it.

|  |
| --- |
| If the given sequence is “BBABCBCAB”, then the output should be 7 as “BABCBAB” is the longest palindromic subsequence in it.“BBBBB” and “BBCBB” are also palindromic subsequences of the given sequence, but not the longest ones. Time Complexity is O(n^2)  **public** **class** LongestPalindromicSubsequence {  **public** **static** **void** main(String[] args) {  String seq = "GEEKSFORGEEKS";  **int** n = seq.length();  System.***out***.println("The length of the lps is " + *lps*(seq)); //5  }  **static** **int** lps(String seq) {  **int** n = seq.length();  **int** i, j, cl;  **int** L[][] = **new** **int**[n][n];  **for** (i = 0; i < n; i++)  L[i][i] = 1;  **for** (cl = 2; cl <= n; cl++) {  **for** (i = 0; i < n - cl + 1; i++) {  j = i + cl - 1;  **if** (seq.charAt(i) == seq.charAt(j) && cl == 2)  L[i][j] = 2;  **else** **if** (seq.charAt(i) == seq.charAt(j))  L[i][j] = L[i + 1][j - 1] + 2;  **else**  L[i][j] = *max*(L[i][j - 1], L[i + 1][j]);  }  }  **return** L[0][n - 1];  }  **static** **int** max(**int** x, **int** y) {  **return** (x > y) ? x : y;  }  } |

# **Longest Path Matrix**

|  |
| --- |
| Find the longest path in a matrix with given constraints. Given an n\*n matrix where all numbers are distinct, find the maximum length path (starting from any cell) such that all cells along the path are in increasing order with a difference of 1. Time complexity of the above solution is O (n2).  Input: mat[][] = {{1, 2, 9}  {5, 3, 8}  {4, 6, 7}}  Output: 4  The longest path is 6-7-8-9.  **public** **class** LongestPathMatrix {  **public** **static** **int** *n* = 3;  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 2, 9 }, { 5, 3, 8 }, { 4, 6, 7 } };  System.***out***.println("Length of the longest path is " + *finLongestOverAll*(mat)); //4  }  **static** **int** finLongestOverAll(**int** mat[][]) {  **int** result = 1;  **int**[][] dp = **new** **int**[*n*][*n*];  **for** (**int** i = 0; i < *n*; i++)  **for** (**int** j = 0; j < *n*; j++)  dp[i][j] = -1;  **for** (**int** i = 0; i < *n*; i++) {  **for** (**int** j = 0; j < *n*; j++) {  **if** (dp[i][j] == -1)  *findLongestFromACell*(i, j, mat, dp);  result = Math.*max*(result, dp[i][j]);  }  }  **return** result;  }  **static** **int** findLongestFromACell(**int** i, **int** j, **int** mat[][], **int** dp[][]) {  **if** (i < 0 || i >= *n* || j < 0 || j >= *n*)  **return** 0;  **if** (dp[i][j] != -1)  **return** dp[i][j];  **if** (j < *n* - 1 && ((mat[i][j] + 1) == mat[i][j + 1]))  **return** dp[i][j] = 1 + *findLongestFromACell*(i, j + 1, mat, dp);  **if** (j > 0 && (mat[i][j] + 1 == mat[i][j - 1]))  **return** dp[i][j] = 1 + *findLongestFromACell*(i, j - 1, mat, dp);  **if** (i > 0 && (mat[i][j] + 1 == mat[i - 1][j]))  **return** dp[i][j] = 1 + *findLongestFromACell*(i - 1, j, mat, dp);  **if** (i < *n* - 1 && (mat[i][j] + 1 == mat[i + 1][j]))  **return** dp[i][j] = 1 + *findLongestFromACell*(i + 1, j, mat, dp);  **return** dp[i][j] = 1;  }  } |

# **Longest Repeating Subsequence**

|  |
| --- |
| Given a string, find length of the longest repeating subsequence such that the two subsequence don’t have same string character at same position, i.e., any i’th character in the two subsequences shouldn’t have the same index in the original string. Input: str = "abc" Output: 0 There is no repeating subsequence  Input: str = "aab" Output: 1\*/  **public** **class** LongestRepeatingSubsequence {  **public** **static** **void** main(String[] args) {  String str = "aabb";  System.***out***.println("The length of the largest subsequence that" + " repeats itself is : "  + *findLongestRepeatingSubSeq*(str)); //2  }  **static** **int** findLongestRepeatingSubSeq(String str) {  **int** n = str.length();  **int**[][] dp = **new** **int**[n + 1][n + 1];  **for** (**int** i = 1; i <= n; i++) {  **for** (**int** j = 1; j <= n; j++) {  **if** (str.charAt(i - 1) == str.charAt(j - 1) && i != j)  dp[i][j] = 1 + dp[i - 1][j - 1];  **else**  dp[i][j] = Math.*max*(dp[i][j - 1], dp[i - 1][j]);  }  }  **return** dp[n][n];  }  } |

# **Longest Subsequence**

|  |
| --- |
| Longest subsequence such that difference between adjacent is one Given an array of n size, the task is to find the longest subsequence such that difference between adjacent is one.  Input : arr[] = {10, 9, 4, 5, 4, 8, 6} Output : 3 As longest subsequences with difference 1 are, "10, 9, 8", "4, 5, 4" and "4, 5, 6"  Time Complexity: O(n2) Auxiliary Space: O(n)  **public** **class** LongestSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 5, 3, 2 };  **int** n = arr.length;  System.***out***.println(*longestSubseqWithDiffOne*(arr, n)); //6  }  **static** **int** longestSubseqWithDiffOne(**int** arr[], **int** n) {  **int** dp[] = **new** **int**[n];  **for** (**int** i = 0; i < n; i++)  dp[i] = 1;  **for** (**int** i = 1; i < n; i++) {  **for** (**int** j = 0; j < i; j++) {  **if** ((arr[i] == arr[j] + 1) || (arr[i] == arr[j] - 1))  dp[i] = Math.*max*(dp[i], dp[j] + 1);  }  }  **int** result = 1;  **for** (**int** i = 0; i < n; i++)  **if** (result < dp[i])  result = dp[i];  **return** result;  }  } |

# **Longest Zig Zag Subsequence**

|  |
| --- |
| The longest Zig-Zag subsequence problem is to find length of the longest subsequence of given sequence such that all elements of this are alternating.  Input: arr[] = {10, 22, 9, 33, 49, 50, 31, 60} Output: 6 The subsequences {10, 22, 9, 33, 31, 60} or  {10, 22, 9, 49, 31, 60} or {10, 22, 9, 50, 31, 60} are longest Zig-Zag of length 6.  Time Complexity: O(n2) Auxiliary Space: O(n)  **public** **class** LongestZigZagSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 22, 9, 33, 49, 50, 31, 60 };  **int** n = arr.length;  System.***out***.println("Length of Longest " + "Zig-Zag subsequence is " + *zzis*(arr, n)); 6  }  **static** **int** zzis(**int** arr[], **int** n) {  **int** Z[][] = **new** **int**[n][2];  **for** (**int** i = 0; i < n; i++)  Z[i][0] = Z[i][1] = 1;  **int** res = 1;  **for** (**int** i = 1; i < n; i++) {  **for** (**int** j = 0; j < i; j++) {  **if** (arr[j] < arr[i] && Z[i][0] < Z[j][1] + 1)  Z[i][0] = Z[j][1] + 1;  **if** (arr[j] > arr[i] && Z[i][1] < Z[j][0] + 1)  Z[i][1] = Z[j][0] + 1;  }  **if** (res < Math.*max*(Z[i][0], Z[i][1]))  res = Math.*max*(Z[i][0], Z[i][1]);  }  **return** res;  }  } |

# **Matrix Chain Multiplication**

|  |
| --- |
| Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications. Overlapping Subproblems. Minimum number of multiplications is 30  **public** **class** MatrixChainMultiplication {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 1, 2, 3, 4, 3 };  **int** n = arr.length;  System.***out***.println("Minimum number of multiplications is " + *MatrixChainOrder*(arr, 1, n - 1));  }  **static** **int** MatrixChainOrder(**int** p[], **int** i, **int** j) {  **if** (i == j) **return** 0;  **int** min = Integer.***MAX\_VALUE***;  **for** (**int** k = i; k < j; k++) {  **int** count = *MatrixChainOrder*(p, i, k) + *MatrixChainOrder*(p, k + 1, j) + p[i - 1] \* p[k] \* p[j];  **if** (count < min)  min = count;  }  **return** min;  }  } |

# **Matrix Chain Multiplication1**

|  |
| --- |
| Dynamic Programming Solution .Minimum number of multiplications is 18 .Time Complexity: O(n^3)  Auxiliary Space: O(n^2)  **public** **class** MatrixChainMultiplication1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = **new** **int**[] { 1, 2, 3, 4 };  **int** size = arr.length;  System.***out***.println("Minimum number of multiplications is " + *MatrixChainOrder*(arr, size));  }  **static** **int** MatrixChainOrder(**int** p[], **int** n) {  **int** m[][] = **new** **int**[n][n];  **int** i, j, k, L, q;  **for** (i = 1; i < n; i++)  m[i][i] = 0;  **for** (L = 2; L < n; L++) {  **for** (i = 1; i < n - L + 1; i++) {  j = i + L - 1;  **if** (j == n) **continue**;  m[i][j] = Integer.***MAX\_VALUE***;  **for** (k = i; k <= j - 1; k++) {  q = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];  **if** (q < m[i][j])  m[i][j] = q;  }  }  }  **return** m[1][n - 1];  }  } |

# **Maximize Sum Differences**

|  |
| --- |
| Modify array to maximize sum of adjacent differences. Given an array, we need to modify values of this array in such a way that sum of absolute differences between two consecutive elements is maximized. If the value of an array element is X, then we can change it to either 1 or X.  Input : arr[] = [3, 2, 1, 4, 5] Output : 8 We can modify above array as, Modified arr[] = [3, 1, 1, 4, 1]  Sum of differences = |1-3| + |1-1| + |4-1| + |1-4| = 8  Time Complexity : O(N) Auxiliary Space : O(N)  **public** **class** MaximizeSumDifferences {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 2, 1, 4, 5 };  **int** N = arr.length;  System.***out***.println(*maximumDifferenceSum*(arr, N)); //8  }  **static** **int** maximumDifferenceSum(**int** arr[], **int** N) {  **int** dp[][] = **new** **int**[N][2];  **for** (**int** i = 0; i < N; i++)  dp[i][0] = dp[i][1] = 0;  **for** (**int** i = 0; i < (N - 1); i++) {  dp[i + 1][0] = Math.*max*(dp[i][0], dp[i][1] + Math.*abs*(1 - arr[i]));  dp[i + 1][1] = Math.*max*(dp[i][0] + Math.*abs*(arr[i + 1] - 1), dp[i][1] + Math.*abs*(arr[i + 1] - arr[i]));  }  **return** Math.*max*(dp[N - 1][0], dp[N - 1][1]);  }  } |

# **Maximum Path Sum**

|  |
| --- |
| Maximum path sum that starting with any cell of 0-th row and ending with any cell of (N-1)-th row  Input : mat[4][4] = { {4, 2, 3, 4},  {2, 9, 1, 10},  {15, 1, 3, 0},  {16, 92, 41, 44} };  Output :120 path : 4 + 9 + 15 + 92 = 120 Time complexity : O(N2)  **public** **class** MaximumPathSum {  **static** **int** *N* = 4;  **public** **static** **void** main(String[] args) {  **int** Mat[][] = { { 4, 2, 3, 4 }, { 2, 9, 1, 10 }, { 15, 1, 3, 0 }, { 16, 92, 41, 44 } };  System.***out***.println(*MaximumPath*(Mat)); //120  }  **static** **int** MaximumPath(**int** Mat[][]) {  **int** result = 0;  **int** dp[][] = **new** **int**[*N*][*N* + 2];  **for** (**int**[] rows : dp)  Arrays.*fill*(rows, 0);  **for** (**int** i = 0; i < *N*; i++)  dp[0][i + 1] = Mat[0][i];  **for** (**int** i = 1; i < *N*; i++)  **for** (**int** j = 1; j <= *N*; j++)  dp[i][j] = Math.*max*(dp[i - 1][j - 1], Math.*max*(dp[i - 1][j], dp[i - 1][j + 1])) + Mat[i][j - 1];  **for** (**int** i = 0; i <= *N*; i++)  result = Math.*max*(result, dp[*N* - 1][i]);  **return** result;  }  } |

# **Maximum Path Sum Position**

|  |
| --- |
| Maximum path sum for each position with jumps under divisibility condition. Given an array of n positive integers. Initially we are at first position. We can jump to position y (1 <= x <= n) from position x (1 <= x <= n) if x divides y and x < y. The task is to print maximum sum path ending at every position x. Note: Since first element is at position 1, we can jump to any position from here as 1 divides all other position numbers.  Input : arr[] = {2, 3, 1, 4, 6, 5} Output : 2 5 3 9 8 10 Time Complexity: O(n\*sqrt(n)).  **public** **class** MaximumPathSumPosition {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 1, 4, 6, 5 };  **int** n = arr.length;  *printMaxSum*(arr, n);  }  **static** **void** printMaxSum(**int** arr[], **int** n) {  **int** dp[] = **new** **int**[n];  Arrays.*fill*(dp, 0);  **for** (**int** i = 0; i < n; i++) {  dp[i] = arr[i];  **int** maxi = 0;  **for** (**int** j = 1; j <= Math.*sqrt*(i + 1); j++) {  **if** (((i + 1) % j == 0) && (i + 1) != j) {  **if** (dp[j - 1] > maxi)  maxi = dp[j - 1];  **if** (dp[(i + 1) / j - 1] > maxi && j != 1)  maxi = dp[(i + 1) / j - 1];  }  }  dp[i] += maxi;  }  **for** (**int** i = 0; i < n; i++)  System.***out***.print(dp[i] + " ");//2 5 3 9 8 10  }  } |

# **Maximum Path Sum Triangle**

|  |
| --- |
| We have given numbers in form of triangle, by starting at the top of the triangle and moving to adjacent numbers on the row below, find the maximum total from top to bottom.  Input : 3  7 4  2 4 6  8 5 9 3  Output : 23  Explanation : 3 + 7 + 4 + 9 = 23  Input :  8  -4 4  2 2 6  1 1 1 1  Output : 19  Explanation : 8 + 4 + 6 + 1 = 19  **public** **class** MaximumPathSumTriangle {  **static** **int** *N* = 3;  **public** **static** **void** main(String[] args) {  **int** tri[][] = { { 1, 0, 0 }, { 4, 8, 0 }, { 1, 5, 3 } };  System.***out***.println(*maxPathSum*(tri, 2, 2));//14  }  **static** **int** maxPathSum(**int** tri[][], **int** m, **int** n) {  **for** (**int** i = m - 1; i >= 0; i--) {  **for** (**int** j = 0; j <= i; j++) {  **if** (tri[i + 1][j] > tri[i + 1][j + 1])  tri[i][j] += tri[i + 1][j];  **else**  tri[i][j] += tri[i + 1][j + 1];  }  }  **return** tri[0][0];  }  } |

1. **Maximum Product Cutting: -** Given a rope of length n meters, cut the rope in different parts of integer lengths in a way that maximizes product of lengths of all parts. You must make at least one cut. Assume that the length of rope is more than 2 meters.

|  |
| --- |
| Input: n = 2 Output: 1 (Maximum obtainable product is 1\*1)  Input: n = 3 Output: 2 (Maximum obtainable product is 1\*2)  Overlapping Subproblems  **public** **class** MaximumProductCutting {  **public** **static** **void** main(String[] args) {  System.***out***.println("Maximum Product is " + *maxProd*(10)); //Maximum Product is 36  }  **static** **int** maxProd(**int** n) {  **if** (n == 0 || n == 1) **return** 0;  **int** max\_val = 0;  **for** (**int** i = 1; i < n; i++)  max\_val = Math.*max*(max\_val, Math.*max*(i \* (n - i), *maxProd*(n - i) \* i));  **return** max\_val;  }  } |

# **Maximum Product Cutting1**

|  |
| --- |
| //A Tricky Solution:  **public** **class** MaximumProductCutting1 {  **public** **static** **void** main(String[] args) {  System.***out***.println("Maximum Product is " + *maxProd*(10)); //Maximum Product is 36  }  **static** **int** maxProd(**int** n) {  **if** (n == 2 || n == 3) **return** (n - 1);  **int** res = 1;  **while** (n > 4) {  n -= 3;  res \*= 3;  }  **return** (n \* res);  }  } |

# **Maximum Profit: -** Maximum profit by buying and selling a share at most twice.In a daily share trading, a buyer buys shares in the morning and sells it on same day. If the trader is allowed to make at most 2 transactions in a day, whereas second transaction can only start after first one is complete (Sell->buy->sell->buy).Given stock prices throughout day, find out maximum profit that a share trader could have made.

|  |
| --- |
| Time complexity of the above solution is O (n).  Input: price[] = {10, 22, 5, 75, 65, 80} Output: 87  **public** **class** MaximumProfit {  **public** **static** **void** main(String[] args) {  **int** price[] = { 2, 30, 15, 10, 8, 25, 80 };  **int** n = price.length;  System.***out***.println("Maximum Profit = " + *maxProfit*(price, n)); //Maximum Profit = 100  }  **static** **int** maxProfit(**int** price[], **int** n) {  **int** profit[] = **new** **int**[n];  **for** (**int** i = 0; i < n; i++)  profit[i] = 0;  **int** max\_price = price[n - 1];  **for** (**int** i = n - 2; i >= 0; i--) {  **if** (price[i] > max\_price)  max\_price = price[i];  profit[i] = Math.*max*(profit[i + 1], max\_price - price[i]);  }  **int** min\_price = price[0];  **for** (**int** i = 1; i < n; i++) {  **if** (price[i] < min\_price)  min\_price = price[i];  profit[i] = Math.*max*(profit[i - 1], profit[i] + (price[i] - min\_price));  }  **int** result = profit[n - 1];  **return** result;  }  } |

# **Maximum Subsequence Sum: -** Maximum subsequence sum such that no three are consecutive. Given a sequence of positive numbers, find the maximum sum that can be formed which has no three consecutive elements present.

|  |
| --- |
| Input: arr[] = {1, 2, 3} Output: 5 We can't take three of them, so answer is 2 + 3 = 5  Input: arr[] = {3000, 2000, 1000, 3, 10} Output: 5013  Time Complexity : O(n) Auxiliary Space : O(n)  **public** **class** MaximumSubsequenceSum {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 100, 1000, 100, 1000, 1 };  **int** n = arr.length;  System.***out***.println(*maxSumWO3Consec*(arr, n)); //2101  }  **static** **int** maxSumWO3Consec(**int** arr[], **int** n) {  **int** sum[] = **new** **int**[n];  sum[0] = arr[0];  sum[1] = arr[0] + arr[1];  sum[2] = Math.*max*(sum[1], Math.*max*(arr[1] + arr[2], arr[0] + arr[2]));  **for** (**int** i = 3; i < n; i++)  sum[i] = Math.*max*(Math.*max*(sum[i - 1], sum[i - 2] + arr[i]), arr[i] + arr[i - 1] + sum[i - 3]);  **return** sum[n - 1];  }  } |

# **Maximum Sum Alternating Subsequence: -** Given an array, the task is to find sum of maximum sum alternating subsequence starting with first elements. Here alternating sebsequence means decreasing, then increasing then decreasing. For e.g. 10, 5,14,3 is alternating sequence.

|  |
| --- |
| Input : arr[] = {4, 3, 8, 5, 3, 8} Output : 28  Explanation: The alternating subsequence (starting with first element) that has above maximum sum is {4, 3, 8, 5, 8}  Time Complexity : O(n2) Auxiliary Space : O(n)  **public** **class** MaximumSumAlternatingSubsequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 8, 2, 3, 5, 7, 9, 10 };  System.***out***.println("Maximum sum = " + *maxAlternateSum*(arr, arr.length));//Maximum sum = 25  }  **static** **int** maxAlternateSum(**int** arr[], **int** n) {  **if** (n == 1) **return** arr[0];  **int** dec[] = **new** **int**[n];  **int** inc[] = **new** **int**[n];  dec[0] = inc[0] = arr[0];  **int** flag = 0;  **for** (**int** i = 1; i < n; i++) {  **for** (**int** j = 0; j < i; j++) {  **if** (arr[j] > arr[i]) {  dec[i] = Math.*max*(dec[i], inc[j] + arr[i]);  flag = 1;  }  **else** **if** (arr[j] < arr[i] && flag == 1)  inc[i] = Math.*max*(inc[i], dec[j] + arr[i]);  }  }  **int** result = Integer.***MIN\_VALUE***;  **for** (**int** i = 0; i < n; i++) {  **if** (result < inc[i])  result = inc[i];  **if** (result < dec[i])  result = dec[i];  }  **return** result;  }  } |

# **Maximum Sum BiTonic SubSequence: -** Given an array of intergers. A subsequence of arr [] is called Bitonic if it is first increasing, then decreasing.

|  |
| --- |
| Time complexity : O(n2) Input : arr[] = {1, 15, 51, 45, 33, 100, 12, 18, 9} Output : 194  Maximum sum Bi-tonic sub-sequence is 1 + 15 + 51 + 100 + 18 + 9 = 194  **public** **class** MaximumSumBiTonicSubSequence {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 15, 51, 45, 33, 100, 12, 18, 9 };  **int** n = arr.length;  System.***out***.println("Maximum Sum : " + *MaxSumBS*(arr, n));//194  }  **static** **int** MaxSumBS(**int** arr[], **int** n) {  **int** max\_sum = Integer.***MIN\_VALUE***;  **int** MSIBS[] = **new** **int**[n];  **int** MSDBS[] = **new** **int**[n];  **for** (**int** i = 0; i < n; i++) {  MSDBS[i] = arr[i];  MSIBS[i] = arr[i];  }  **for** (**int** i = 1; i < n; i++)  **for** (**int** j = 0; j < i; j++)  **if** (arr[i] > arr[j] && MSIBS[i] < MSIBS[j] + arr[i])  MSIBS[i] = MSIBS[j] + arr[i];  **for** (**int** i = n - 2; i >= 0; i--)  **for** (**int** j = n - 1; j > i; j--)  **if** (arr[i] > arr[j] && MSDBS[i] < MSDBS[j] + arr[i])  MSDBS[i] = MSDBS[j] + arr[i];  **for** (**int** i = 0; i < n; i++)  max\_sum = Math.*max*(max\_sum, (MSDBS[i] + MSIBS[i] - arr[i]));  **return** max\_sum;  }  } |

# **Maximum Sum of Pairs**

|  |
| --- |
| Maximum sum of pairs with specific difference. Given an array of integers and a number k. We can pair two number of array if difference between them is strictly less than k. The task is to find maximum possible sum of disjoint pairs. Sum of P pairs is sum of all 2P numbers of pairs.  Input : arr[] = {3, 5, 10, 15, 17, 12, 9}, K = 4 Output : 62  Time complexity : O(N Log N) Auxiliary Space : O(N)  **public** **class** MaximumSumOfPairs {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 5, 10, 15, 17, 12, 9 };  **int** N = arr.length;  **int** K = 4;  System.***out***.println(*maxSumPairWithDifferenceLessThanK*(arr, N, K)); //62  }  **static** **int** maxSumPairWithDifferenceLessThanK(**int** arr[], **int** N, **int** K) {  Arrays.*sort*(arr);  **int** dp[] = **new** **int**[N];  dp[0] = 0;  **for** (**int** i = 1; i < N; i++) {  dp[i] = dp[i - 1];  **if** (arr[i] - arr[i - 1] < K) {  **if** (i >= 2)  dp[i] = Math.*max*(dp[i], dp[i - 2] + arr[i] + arr[i - 1]);  **else**  dp[i] = Math.*max*(dp[i], arr[i] + arr[i - 1]);  }  }  **return** dp[N - 1];  }  } |

# **Maximum Sum of Pairs1**

|  |
| --- |
| Time complexity : O(N Log N) Auxiliary Space : O(1)  **public** **class** MaximumSumOfPairs1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 5, 10, 15, 17, 12, 9 };  **int** N = arr.length;  **int** K = 4;  System.***out***.println(*maxSumPairWithDifferenceLessThanK*(arr, N, K));//62  }  **static** **int** maxSumPairWithDifferenceLessThanK(**int** arr[], **int** N, **int** k) {  **int** maxSum = 0;  Arrays.*sort*(arr);  **for** (**int** i = N - 1; i > 0; --i) {  **if** (arr[i] - arr[i - 1] < k) {  maxSum += arr[i];  maxSum += arr[i - 1];  --i;  }  }  **return** maxSum;  }  } |

# **Maximum Sum of Path**

|  |
| --- |
| Maximum sum of a path in a Right Number Triangle. Given a right triangle of numbers, find the largest of the sum of numbers that appear on the paths starting from the top towards the base, so that on each path the next number is located directly below or below-and-one-place-to-the-right.  Input : 1  1 2  4 1 2  2 3 1 1  Output : 9 Explanation : 1 + 1 + 4 + 3  **public** **class** MaximumSumOfPath {  **public** **static** **void** main(String[] args) {  **int** tri[][] = { { 1 }, { 2, 1 }, { 3, 3, 2 } };  System.***out***.println(*maxSum*(tri, 3)); //6  }  **static** **int** maxSum(**int** tri[][], **int** n) {  **if** (n > 1)  tri[1][1] = tri[1][1] + tri[0][0];  tri[1][0] = tri[1][0] + tri[0][0];  **for** (**int** i = 2; i < n; i++) {  tri[i][0] = tri[i][0] + tri[i - 1][0];  tri[i][i] = tri[i][i] + tri[i - 1][i - 1];  **for** (**int** j = 1; j < i; j++) {  **if** (tri[i][j] + tri[i - 1][j - 1] >= tri[i][j] + tri[i - 1][j])  tri[i][j] = tri[i][j] + tri[i - 1][j - 1];  **else**  tri[i][j] = tri[i][j] + tri[i - 1][j];  }  }  **int** max = tri[n - 1][0];  **for** (**int** i = 1; i < n; i++) {  **if** (max < tri[n - 1][i])  max = tri[n - 1][i];  }  **return** max;  }  } |

# **Maximum Sum Rectangle**

|  |
| --- |
| Kadane’s algorithm: - Given a 2D array, find the maximum sum subarray in it. For example, in the following 2D array, the maximum sum subarray is highlighted with blue rectangle and sum of this subarray is 29.  time complexity to O(n^3) MaxSum: 29, range: [(1, 1)(3, 3)]  **public** **class** MaximumSumRectangle {  **public** **static** **void** main(String[] args) **throws** java.lang.Exception {  *findMaxSubMatrix*(  **new** **int**[][] { { 1, 2, -1, -4, -20 }, { -8, -3, 4, 2, 1 }, { 3, 8, 10, 1, 3 }, { -4, -1, 1, 7, -6 } });  }  **public** **static** **void** findMaxSubMatrix(**int**[][] a) {  **int** cols = a[0].length;  **int** rows = a.length;  **int**[] currentResult;  **int** maxSum = Integer.***MIN\_VALUE***;  **int** left = 0;  **int** top = 0;  **int** right = 0;  **int** bottom = 0;  **for** (**int** leftCol = 0; leftCol < cols; leftCol++) {  **int**[] tmp = **new** **int**[rows];  **for** (**int** rightCol = leftCol; rightCol < cols; rightCol++) {  **for** (**int** i = 0; i < rows; i++) {  tmp[i] += a[i][rightCol];  }  currentResult = *kadane*(tmp);  **if** (currentResult[0] > maxSum) {  maxSum = currentResult[0];  left = leftCol;  top = currentResult[1];  right = rightCol;  bottom = currentResult[2];  }  }  }  System.***out***.println("MaxSum: " + maxSum + ", range: [(left " + left + ", top " + top + ")( right " + right + ", bottom " + bottom + ")]");  }  **public** **static** **int**[] kadane(**int**[] a) {  **int**[] result = **new** **int**[] { Integer.***MIN\_VALUE***, 0, -1 };  **int** currentSum = 0;  **int** localStart = 0;  **for** (**int** i = 0; i < a.length; i++) {  currentSum += a[i];  **if** (currentSum < 0) {  currentSum = 0;  localStart = i + 1;  } **else** **if** (currentSum > result[0]) {  result[0] = currentSum;  result[1] = localStart;  result[2] = i;  }  }  **if** (result[2] == -1) {  result[0] = 0;  **for** (**int** i = 0; i < a.length; i++) {  **if** (a[i] > result[0]) {  result[0] = a[i];  result[1] = i;  result[2] = i;  }  }  }  **return** result;  }  } |

# **Maximum Weight Path**

|  |
| --- |
| Maximum weight path ending at any element of last row in a matrix: - Time complexity : O(N\*N) .Given a matrix of integers where every element represents weight of the cell. Find the path having the maximum weight in matrix [N X N]. Path Traversal Rules are:  1) It should begin from top left element.  2) The path can end at any element of last row.  3) We can move to following two cells from a cell (i, j).  a) Down Move : (i+1, j)  b) Diagonal Move : (i+1, j+1)  Input : N = 5  mat[5][5] = {{ 4, 2 ,3 ,4 ,1 },  { 2 , 9 ,1 ,10 ,5 },  {15, 1 ,3 , 0 ,20 },  {16 ,92, 41, 44 ,1},  {8, 142, 6, 4, 8} };  Output : 255  Path with max weight : 4 + 2 +15 + 92 + 142 = 255  **public** **class** MaximumWeightPath {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 4, 1, 5, 6, 1 }, { 2, 9, 2, 11, 10 }, { 15, 1, 3, 15, 2 }, { 16, 92, 41, 4, 3 },{ 8, 142, 6, 4, 8 } };  **int** N = 5;  System.***out***.println("Maximum Path Sum : " + *maxCost*(mat, N)); //Maximum Path Sum : 255  }  **public** **static** **int** maxCost(**int** mat[][], **int** N) {  **int** dp[][] = **new** **int**[N][N];  dp[0][0] = mat[0][0];  **for** (**int** i = 1; i < N; i++)  dp[i][0] = mat[i][0] + dp[i - 1][0];  **for** (**int** i = 1; i < N; i++)  **for** (**int** j = 1; j < i + 1 && j < N; j++)  dp[i][j] = mat[i][j] + Math.*max*(dp[i - 1][j - 1], dp[i - 1][j]);  **int** result = 0;  **for** (**int** i = 0; i < N; i++)  **if** (result < dp[N - 1][i])  result = dp[N - 1][i];  **return** result;  }  } |

# **Minimum number of deletions to make a string palindrome**

|  |
| --- |
| Minimum number of deletions to make a string palindrome. Given a string of size ‘n’. The task is to remove or delete minimum number of characters from the string so that the resultant string is palindrome.  Input : aebcbda Output : 2 Remove characters 'e' and 'd' Resultant string will be 'abcba' which is a palindromic string  Input : geeksforgeeks Output : 8 Time Complexity: O(n2)  **public** **class** MindeletetoPalin {  **public** **static** **void** main(String[] args) {  String str = "geeksforgeeks";  System.***out***.println("Minimum number of deletions = " + *minimumNumberOfDeletions*(str)); 8  }  **static** **int** minimumNumberOfDeletions(String str) {  **int** n = str.length();  **int** len = *lps*(str);  **return** (n - len);  }  **static** **int** lps(String str) {  **int** n = str.length();  **int** L[][] = **new** **int**[n][n];  **for** (**int** i = 0; i < n; i++)  L[i][i] = 1;  **for** (**int** cl = 2; cl <= n; cl++) {  **for** (**int** i = 0; i < n - cl + 1; i++) {  **int** j = i + cl - 1;  **if** (str.charAt(i) == str.charAt(j) && cl == 2)  L[i][j] = 2;  **else** **if** (str.charAt(i) == str.charAt(j))  L[i][j] = L[i + 1][j - 1] + 2;  **else**  L[i][j] = Integer.*max*(L[i][j - 1], L[i + 1][j]);  }  }  **return** L[0][n - 1];  }  } |

# **Find minimum adjustment cost of an array**

|  |
| --- |
| Input: arr = [1, 3, 0, 3], target = 1 Output: Minimum adjustment cost is 3  Explanation: One of the possible solutions is [2, 3, 2, 3]  **public** **class** MinimumAdjustmentCost {  **public** **static** **int** *M* = 100;  **public** **static** **void** main(String[] args) {  **int** arr[] = { 55, 77, 52, 61, 39, 6, 25, 60, 49, 47 };  **int** n = arr.length;  **int** target = 10;  System.***out***.println("Minimum adjustment cost is " + *minAdjustmentCost*(arr, n, target)); 75  }  **static** **int** minAdjustmentCost(**int** A[], **int** n, **int** target) {  **int**[][] dp = **new** **int**[n][*M* + 1];  **for** (**int** j = 0; j <= *M*; j++)  dp[0][j] = Math.*abs*(j - A[0]);  **for** (**int** i = 1; i < n; i++) {  **for** (**int** j = 0; j <= *M*; j++) {  dp[i][j] = Integer.***MAX\_VALUE***;  **int** k = Math.*max*(j - target, 0);  **for** (; k <= Math.*min*(*M*, j + target); k++)  dp[i][j] = Math.*min*(dp[i][j], dp[i - 1][k] + Math.*abs*(A[i] - j));  }  }  **int** res = Integer.***MAX\_VALUE***;  **for** (**int** j = 0; j <= *M*; j++)  res = Math.*min*(res, dp[n - 1][j]);  **return** res;  }  } |

# **Find the minimum cost to reach destination using a train**

|  |
| --- |
| There are N stations on route of a train. The train goes from station 0 to N-1. The ticket cost for all pair of stations (i, j) is given where j is greater than i. Find the minimum cost to reach the destination.  Input: cost[N][N] = { {0, 15, 80, 90},  {INF, 0, 40, 50},  {INF, INF, 0, 70},  {INF, INF, INF, 0}  };  There are 4 stations and cost[i][j] indicates cost to reach j from i. The entries where j < i are meaningless.  Output: The minimum cost is 65 The minimum cost can be obtained by first going to station 1 from 0. Then from station 1 to station 3.\*/  **public** **class** MinimumCostReachDestination {  **static** **int** *INF* = Integer.***MAX\_VALUE***, *N* = 4;  **public** **static** **void** main(String[] args) {  **int** cost[][] = { { 0, 15, 80, 90 }, { *INF*, 0, 40, 50 }, { *INF*, *INF*, 0, 70 }, { *INF*, *INF*, *INF*, 0 } };  System.***out***.println("The Minimum cost to reach station " + *N* + " is " + *minCost*(cost)); 4 is 65  }  **static** **int** minCost(**int** cost[][]) {  **return** *minCostRec*(cost, 0, *N* - 1);  }  **static** **int** minCostRec(**int** cost[][], **int** s, **int** d) {  **if** (s == d || s + 1 == d) **return** cost[s][d];  **int** min = cost[s][d];  **for** (**int** i = s + 1; i < d; i++) {  **int** c = *minCostRec*(cost, s, i) + *minCostRec*(cost, i, d);  **if** (c < min)  min = c;  }  **return** min;  }  } |

# **Minimum Cost Reach Destination1**

|  |
| --- |
| **public** **class** MinimumCostReachDestination1 {  **static** **int** *INF* = Integer.***MAX\_VALUE***, *N* = 4;  **public** **static** **void** main(String[] args) {  **int** cost[][] = { { 0, 15, 80, 90 }, { *INF*, 0, 40, 50 }, { *INF*, *INF*, 0, 70 }, { *INF*, *INF*, *INF*, 0 } };  System.***out***.println("The Minimum cost to reach station " + *N* + " is " + *minCost*(cost)); 4 is 65  }  **static** **int** minCost(**int** cost[][]) {  **int** dist[] = **new** **int**[*N*];  **for** (**int** i = 0; i < *N*; i++)  dist[i] = *INF*;  dist[0] = 0;  **for** (**int** i = 0; i < *N*; i++)  **for** (**int** j = i + 1; j < *N*; j++)  **if** (dist[j] > dist[i] + cost[i][j])  dist[j] = dist[i] + cost[i][j];  **return** dist[*N* - 1];  }  } |

# **Minimum cost to fill given weight in a bag**

|  |
| --- |
| You are given a bag of size W kg and you are provided costs of packets different weights of oranges in array cost [] where cost[i] is basically cost of ‘i’ kg packet of oranges. Where cost[i] = -1 means that ‘i’ kg packet of orange is unavailable. Find the minimum total cost to buy exactly W kg oranges and if it is not possible to buy exactly W kg oranges then print -1. It may be assumed that there is infinite supply of all available packet types.  Input: W = 5, cost [] = {20, 10, 4, 50, 100} Output: 14 We can choose two oranges to minimize cost. First orange of 2Kg and cost 10. Second orange of 3Kg and cost 4.  **public** **class** MinimumCostToFill {  **public** **static** **void** main(String[] args) {  **int** cost[] = { 1, 2, 3, 4, 5 }, W = 5;  **int** n = cost.length;  System.***out***.println(*MinimumCost*(cost, n, W)); //5  }  **public** **static** **int** MinimumCost(**int** cost[], **int** n, **int** W) {  Vector<Integer> val = **new** Vector<Integer>();  Vector<Integer> wt = **new** Vector<Integer>();  **int** size = 0;  **for** (**int** i = 0; i < n; i++) {  **if** (cost[i] != -1) {  val.add(cost[i]);  wt.add(i + 1);  size++;  }  }  n = size;  **int** min\_cost[][] = **new** **int**[n + 1][W + 1];  **for** (**int** i = 0; i <= W; i++)  min\_cost[0][i] = Integer.***MAX\_VALUE***;  **for** (**int** i = 1; i <= n; i++)  min\_cost[i][0] = 0;  **for** (**int** i = 1; i <= n; i++) {  **for** (**int** j = 1; j <= W; j++) {  **if** (wt.get(i - 1) > j)  min\_cost[i][j] = min\_cost[i - 1][j];  **else**  min\_cost[i][j] = Math.*min*(min\_cost[i - 1][j], min\_cost[i][j - wt.get(i - 1)] + val.get(i - 1));  }  }  **return** (min\_cost[n][W] == Integer.***MAX\_VALUE***) ? -1 : min\_cost[n][W];  }  } |

# **Minimum Cost to Make Two Strings Identical**

|  |
| --- |
| Given two strings X and Y, and two values costX and costY. We need to find minimum cost required to make the given two strings identical. We can delete characters from both the strings. The cost of deleting a character from string X is costX and from Y is costY. Cost of removing all characters from a string is same.  Input : X = "abcd", Y = "acdb", costX = 10, costY = 20. Output: 30  For Making both strings identical we have to delete character 'b' from both the string, hence cost will be = 10 + 20 = 30. Minimum Cost to make two strings identical is = 60  **public** **class** MinimumCostToMakeTwoStringsIdentical {  **public** **static** **void** main(String[] args) {  String X = "ef";  String Y = "gh";  System.***out***.println("Minimum Cost to make two strings " + " identical is = " + *findMinCost*(X, Y, 10, 20));  }  **static** **int** findMinCost(String X, String Y, **int** costX, **int** costY) {  **int** m = X.length();  **int** n = Y.length();  **int** len\_LCS;  len\_LCS = *lcs*(X, Y, m, n);  **return** costX \* (m - len\_LCS) + costY \* (n - len\_LCS);  }  **static** **int** lcs(String X, String Y, **int** m, **int** n) {  **int** L[][] = **new** **int**[m + 1][n + 1];  **for** (**int** i = 0; i <= m; i++) {  **for** (**int** j = 0; j <= n; j++) {  **if** (i == 0 || j == 0)  L[i][j] = 0;  **else** **if** (X.charAt(i - 1) == Y.charAt(j - 1))  L[i][j] = L[i - 1][j - 1] + 1;  **else**  L[i][j] = Math.*max*(L[i - 1][j], L[i][j - 1]);  }  }  **return** L[m][n];  }  } |

# **Minimum Initial Points**

|  |
| --- |
| Minimum Initial Points to Reach Destination: - Given a grid with each cell consisting of positive, negative or no points i.e, zero points. We can move across a cell only if we have positive points (> 0). Whenever we pass through a cell, points in that cell are added to our overall points. We need to find minimum initial points to reach cell (m-1, n-1) from (0, 0).  Constraints: From a cell (i, j) we can move to (i+1, j) or (i, j+1). We cannot move from (i, j) if your overall points at (i, j) is <= 0. We have to reach at (n-1, m-1) with minimum positive points i.e., > 0.  Input: points[m][n] = { {-2, -3, 3},  {-5, -10, 1},  {10, 30, -5}  };  Output: 7 Explanation: 7 is the minimum value to reach destination with positive throughout the path. Below is the path.  (0,0) -> (0,1) -> (0,2) -> (1, 2) -> (2, 2)  **public** **class** MinimumInitialPoints {  **public** **static** **void** main(String[] args) {  **int** points[][] = { { -2, -3, 3 }, { -5, -10, 1 }, { 10, 30, -5 } };  **int** R = 3, C = 3;  System.***out***.println("Minimum Initial Points Required: " + *minInitialPoints*(points, R, C));//7  }  **static** **int** minInitialPoints(**int** points[][], **int** R, **int** C) {  **int** dp[][] = **new** **int**[R][C];  **int** m = R, n = C;  dp[m - 1][n - 1] = points[m - 1][n - 1] > 0 ? 1 : Math.*abs*(points[m - 1][n - 1]) + 1;  **for** (**int** i = m - 2; i >= 0; i--)  dp[i][n - 1] = Math.*max*(dp[i + 1][n - 1] - points[i][n - 1], 1);  **for** (**int** j = n - 2; j >= 0; j--)  dp[m - 1][j] = Math.*max*(dp[m - 1][j + 1] - points[m - 1][j], 1);  **for** (**int** i = m - 2; i >= 0; i--) {  **for** (**int** j = n - 2; j >= 0; j--) {  **int** min\_points\_on\_exit = Math.*min*(dp[i + 1][j], dp[i][j + 1]);  dp[i][j] = Math.*max*(min\_points\_on\_exit - points[i][j], 1);  }  }  **return** dp[0][0];  }  } |

# **Minimum Insertions Palindrome**

|  |
| --- |
| Given a string, find the minimum number of characters to be inserted to convert it to palindrome. Recursive Solution Before we go further, let us understand with few examples:  ab: Number of insertions required is 1. bab  aa: Number of insertions required is 0. aa  abcd: Number of insertions required is 3. dcbabcd  abcda: Number of insertions required is 2. adcbcda which is same as number of insertions in the substring bcd(Why?).  abcde: Number of insertions required is 4. edcbabcde  **public** **class** MinimumInsertionsPalindrome {  **public** **static** **void** main(String[] args) {  String str = "geeks";  System.***out***.println(*findMinInsertions*(str.toCharArray(), 0, str.length() - 1)); //3  }  **static** **int** findMinInsertions(**char** str[], **int** l, **int** h) {  **if** (l > h)  **return** Integer.***MAX\_VALUE***;  **if** (l == h)  **return** 0;  **if** (l == h - 1)  **return** (str[l] == str[h]) ? 0 : 1;  **return** (str[l] == str[h]) ? *findMinInsertions*(str, l + 1, h - 1)  : (Integer.*min*(*findMinInsertions*(str, l, h - 1), *findMinInsertions*(str, l + 1, h)) + 1);  }  } |

# **Minimum Insertions Palindrome1**

|  |
| --- |
| //Dynamic Programming based Solution Time complexity: O(N^2) Auxiliary Space: O(N^2)  **public** **class** MinimumInsertionsPalindrome1 {  **public** **static** **void** main(String[] args) {  String str = "geeks";  System.***out***.println(*findMinInsertionsDP*(str.toCharArray(), str.length())); //3  }  **static** **int** findMinInsertionsDP(**char** str[], **int** n) {  **int** table[][] = **new** **int**[n][n];  **int** l, h, gap;  **for** (gap = 1; gap < n; ++gap)  **for** (l = 0, h = gap; h < n; ++l, ++h)  table[l][h] = (str[l] == str[h]) ? table[l + 1][h - 1]  : (Integer.*min*(table[l][h - 1], table[l + 1][h]) + 1);  **return** table[0][n - 1];  }  } |

# **Minimum Insertions Palindrome2**

|  |
| --- |
| //Another Dynamic Programming Solution. Time complexity of this method is also O (n^2) and this method also requires O (n^2) extra space.  **public** **class** MinimumInsertionsPalindrome2 {  **public** **static** **void** main(String[] args) {  String str = "geeks";  System.***out***.println(*findMinInsertionsLCS*(str, str.length())); //3  }  **static** **int** findMinInsertionsLCS(String str, **int** n) {  StringBuffer sb = **new** StringBuffer(str);  sb.reverse();  String revString = sb.toString();  **return** (n - *lcs*(str, revString, n, n));  }  **static** **int** lcs(String X, String Y, **int** m, **int** n) {  **int** L[][] = **new** **int**[n + 1][n + 1];  **int** i, j;  **for** (i = 0; i <= m; i++) {  **for** (j = 0; j <= n; j++) {  **if** (i == 0 || j == 0)  L[i][j] = 0;  **else** **if** (X.charAt(i - 1) == Y.charAt(j - 1))  L[i][j] = L[i - 1][j - 1] + 1;  **else**  L[i][j] = Integer.*max*(L[i - 1][j], L[i][j - 1]);  }  }  **return** L[m][n];  }  } |

# **Minimum insertions to sort an array**

|  |
| --- |
| Given an array of integer numbers, we need to sort this array in a minimum number of steps where in one step we can insert any array element from its position to any other position.  Input : arr[] = [2, 3, 5, 1, 4, 7, 6] Output : 3  **public** **class** MinimumInsertionsSortArray {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 5, 1, 4, 7, 6 };  **int** N = arr.length;  System.***out***.println(*minInsertionStepToSortArray*(arr, N)); //3  }  **static** **int** minInsertionStepToSortArray(**int** arr[], **int** N) {  **int**[] lis = **new** **int**[N];  **for** (**int** i = 0; i < N; i++)  lis[i] = 1;  **for** (**int** i = 1; i < N; i++)  **for** (**int** j = 0; j < i; j++)  **if** (arr[i] >= arr[j] && lis[i] < lis[j] + 1)  lis[i] = lis[j] + 1;  **int** max = 0;  **for** (**int** i = 0; i < N; i++)  **if** (max < lis[i])  max = lis[i];  **return** (N - max);  }  } |

# **Find minimum number of coins that make a given value**

|  |
| --- |
| 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  **public** **class** MinimumNumberOfCoins {  **public** **static** **void** main(String[] args) {  **int** coins[] = { 9, 6, 5, 1 };  **int** m = coins.length;  **int** V = 11;  System.***out***.println("Minimum coins required is " + *minCoins*(coins, m, V)); //2  }  **static** **int** minCoins(**int** coins[], **int** m, **int** V) {  **if** (V == 0) **return** 0;  **int** res = Integer.***MAX\_VALUE***;  **for** (**int** i = 0; i < m; i++) {  **if** (coins[i] <= V) {  **int** sub\_res = *minCoins*(coins, m, V - coins[i]);  **if** (sub\_res != Integer.***MAX\_VALUE*** && sub\_res + 1 < res)  res = sub\_res + 1;  }  }  **return** res;  }  } |

# **Minimum Number of Coins1**

|  |
| --- |
| //Dynamic Programming (DP) problems, Time complexity of the above solution is O (mV).  **public** **class** MinimumNumberOfCoins1 {  **public** **static** **void** main(String[] args) {  **int** coins[] = { 9, 6, 5, 1 };  **int** m = coins.length;  **int** V = 11;  System.***out***.println("Minimum coins required is " + *minCoins*(coins, m, V));//2  }  **static** **int** minCoins(**int** coins[], **int** m, **int** V) {  **int** table[] = **new** **int**[V + 1];  table[0] = 0;  **for** (**int** i = 1; i <= V; i++)  table[i] = Integer.***MAX\_VALUE***;  **for** (**int** i = 1; i <= V; i++) {  **for** (**int** j = 0; j < m; j++)  **if** (coins[j] <= i) {  **int** sub\_res = table[i - coins[j]];  **if** (sub\_res != Integer.***MAX\_VALUE*** && sub\_res + 1 < table[i])  table[i] = sub\_res + 1;  }  }  **return** table[V];  }  } |

# **Minimum number of deletions to make a sorted sequence**

|  |
| --- |
| Given an array of n integers. The task is to remove or delete minimum number of elements from the array so that when the remaining elements are placed in the same sequence order form a sorted sequence.  Input: {5, 6, 1, 7, 4} Output: 2. Removing 1 and 4 leaves the remaining sequence order as 5 6 7 which is a sorted sequence. Time Complexity: O(n2)  **public** **class** MinimumNumberOfDeletions {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 30, 40, 2, 5, 1, 7, 45, 50, 8 };  **int** n = arr.length;  System.***out***.println("Minimum number of deletions = " + *minimumNumberOfDeletions*(arr, n)); 4  }  **static** **int** minimumNumberOfDeletions(**int** arr[], **int** n) {  **int** len = *lis*(arr, n);  **return** (n - len);  }  **static** **int** lis(**int** arr[], **int** n) {  **int** result = 0;  **int**[] lis = **new** **int**[n];  **for** (**int** i = 0; i < n; i++)  lis[i] = 1;  **for** (**int** i = 1; i < n; i++)  **for** (**int** j = 0; j < i; j++)  **if** (arr[i] > arr[j] && lis[i] < lis[j] + 1)  lis[i] = lis[j] + 1;  **for** (**int** i = 0; i < n; i++)  **if** (result < lis[i])  result = lis[i];  **return** result;  }  } |

# **Minimum number of squares whose sum equals to given number n**

|  |
| --- |
| A number can always be represented as a sum of squares of other numbers. Note that 1 is a square and we can always break a number as (1\*1 + 1\*1 + 1\*1 + …). Given a number n, find the minimum number of squares that sum to X.  Input: n = 100 Output: 1  Input: n = 6 Output: 3  **public** **class** MinimumNumberOfSquares {  **public** **static** **void** main(String[] args) {  System.***out***.println(*getMinSquares*(6)); //3  }  **static** **int** getMinSquares(**int** n) {  **if** (n <= 3) **return** n;  **int** res = n;  **for** (**int** x = 1; x <= n; x++) {  **int** temp = x \* x;  **if** (temp > n)  **break**;  **else**  res = Math.*min*(res, 1 + *getMinSquares*(n - temp));  }  **return** res;  }  } |

# **Minimum Number of Squares1**

|  |
| --- |
| **public** **class** MinimumNumberOfSquares1 {  **public** **static** **void** main(String[] args) {  System.***out***.println(*getMinSquares*(6));//3  }  **static** **int** getMinSquares(**int** n) {  **int** dp[] = **new** **int**[n + 1];  dp[0] = 0;  dp[1] = 1;  dp[2] = 2;  dp[3] = 3;  **for** (**int** i = 4; i <= n; i++) {  dp[i] = i;  **for** (**int** x = 1; x <= i; x++) {  **int** temp = x \* x;  **if** (temp > i)  **break**;  **else**  dp[i] = Math.*min*(dp[i], 1 + dp[i - temp]);  }  }  **int** res = dp[n];  **return** res;  }  } |

# **Minimum steps to delete a string after repeated deletion of palindrome substrings**

|  |
| --- |
| Given a string containing characters as integers only. We need to delete all character of this string in a minimum number of steps where in one step we can delete the substring which is a palindrome. After deleting a substring remaining parts are concatenated.  Input : s = “2553432” Output : 2  **public** **class** MinimumStepsToDelete {  **public** **static** **void** main(String[] args) {  String str = "2553432";  System.***out***.println(*minStepToDeleteString*(str)); //2  }  **static** **int** minStepToDeleteString(String str) {  **int** N = str.length();  **int**[][] dp = **new** **int**[N + 1][N + 1];  **for** (**int** i = 0; i <= N; i++)  **for** (**int** j = 0; j <= N; j++)  dp[i][j] = 0;  **for** (**int** len = 1; len <= N; len++) {  **for** (**int** i = 0, j = len - 1; j < N; i++, j++) {  **if** (len == 1)  dp[i][j] = 1;  **else** {  dp[i][j] = 1 + dp[i + 1][j];  **if** (str.charAt(i) == str.charAt(i + 1))  dp[i][j] = Math.*min*(1 + dp[i + 2][j], dp[i][j]);  **for** (**int** K = i + 2; K <= j; K++)  **if** (str.charAt(i) == str.charAt(K))  dp[i][j] = Math.*min*(dp[i + 1][K - 1] + dp[K + 1][j], dp[i][j]);  }  }  }  **return** dp[0][N - 1];  }  } |

# **Minimum Sum Path in 3D Array**

|  |
| --- |
| Time Complexity : O(l\*m\*n) Auxiliary Space : O(l\*m\*n)  Input : arr[][][]= { {{1, 2}, {3, 4}}, {{4, 8}, {5, 2}} }; Output : 9  **public** **class** MinimumSumPathIn3DArray {  **static** **int** *l* = 3;  **static** **int** *m* = 3;  **static** **int** *n* = 3;  **public** **static** **void** main(String[] args) {  **int** arr[][][] = { { { 1, 2, 4 }, { 3, 4, 5 }, { 5, 2, 1 } }, { { 4, 8, 3 }, { 5, 2, 1 }, { 3, 4, 2 } },  { { 2, 4, 1 }, { 3, 1, 4 }, { 6, 3, 8 } } };  System.***out***.println(*minPathSum*(arr));//20  }  **static** **int** minPathSum(**int** arr[][][]) {  **int** i, j, k;  **int** tSum[][][] = **new** **int**[*l*][*m*][*n*];  tSum[0][0][0] = arr[0][0][0];  **for** (i = 1; i < *l*; i++) tSum[i][0][0] = tSum[i - 1][0][0] + arr[i][0][0];  **for** (j = 1; j < *m*; j++) tSum[0][j][0] = tSum[0][j - 1][0] + arr[0][j][0];  **for** (k = 1; k < *n*; k++) tSum[0][0][k] = tSum[0][0][k - 1] + arr[0][0][k];  **for** (i = 1; i < *l*; i++)  **for** (j = 1; j < *m*; j++)  tSum[i][j][0] = *min*(tSum[i - 1][j][0], tSum[i][j - 1][0], Integer.***MAX\_VALUE***) + arr[i][j][0];  **for** (i = 1; i < *l*; i++)  **for** (k = 1; k < *n*; k++)  tSum[i][0][k] = *min*(tSum[i - 1][0][k], tSum[i][0][k - 1], Integer.***MAX\_VALUE***) + arr[i][0][k];  **for** (k = 1; k < *n*; k++)  **for** (j = 1; j < *m*; j++)  tSum[0][j][k] = *min*(tSum[0][j][k - 1], tSum[0][j - 1][k], Integer.***MAX\_VALUE***) + arr[0][j][k];  **for** (i = 1; i < *l*; i++)  **for** (j = 1; j < *m*; j++)  **for** (k = 1; k < *n*; k++)  tSum[i][j][k] = *min*(tSum[i - 1][j][k], tSum[i][j - 1][k], tSum[i][j][k - 1]) + arr[i][j][k];  **return** tSum[*l* - 1][*m* - 1][*n* - 1];  }  **static** **int** min(**int** x, **int** y, **int** z) {  **return** (x < y) ? ((x < z) ? x : z) : ((y < z) ? y : z);  }  } |

# **Minimum sum subsequence such that at least one of every four consecutive elements is picked**

|  |
| --- |
| Given an array arr [] of positive integers. The task is to find minimum sum subsequence from the array such that at least one value among all groups of four consecutive elements is picked.  Input: arr[] = {1, 2, 3, 4, 5, 6, 7, 8} Output: 6  **public** **class** MinimumSumSubsequence {  **public** **static** **void** main(String[] args) {  **int**[] arr = { 1, 2, 3, 3, 4, 5, 6, 1 };  **int** n = arr.length;  System.***out***.println(*minSum*(arr, n)); //4  }  **static** **int** minSum(**int**[] arr, **int** n) {  **int**[] dp = **new** **int**[n];  **if** (n == 1) **return** arr[0];  **if** (n == 2) **return** Math.*min*(arr[0], arr[1]);  **if** (n == 3) **return** Math.*min*(arr[0], Math.*min*(arr[1], arr[2]));  **if** (n == 4) **return** Math.*min*(Math.*min*(arr[0], arr[1]), Math.*min*(arr[2], arr[3]));  dp[0] = arr[0];  dp[1] = arr[1];  dp[2] = arr[2];  dp[3] = arr[3];  **for** (**int** i = 4; i < n; i++)  dp[i] = arr[i] + Math.*min*(Math.*min*(dp[i - 1], dp[i - 2]), Math.*min*(dp[i - 3], dp[i - 4]));  **return** Math.*min*(Math.*min*(dp[n - 1], dp[n - 2]), Math.*min*(dp[n - 4], dp[n - 3]));  }  } |

# **Minimum time to finish tasks without skipping two consecutive**

|  |
| --- |
| Given time taken by n tasks. Find the minimum time needed to finish the tasks such that skipping of tasks is allowed, but cannot skip two consecutive tasks.  Input : arr[] = {10, 30} Output : 10  Input : arr[] = {10, 5, 2, 4, 8, 6, 7, 10} Output : 22  Expected Time Complexity is O (n) and extra space is O (1).  **public** **class** MinimumTimeToFinishTasks {  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 10, 5, 2, 7, 10 };  **int** n1 = arr1.length;  System.***out***.println(*minTime*(arr1, n1)); //12  }  **static** **int** minTime(**int** arr[], **int** n) {  **if** (n <= 0) **return** 0;  **int** incl = arr[0];  **int** excl = 0;  **for** (**int** i = 1; i < n; i++) {  **int** incl\_new = arr[i] + Math.*min*(excl, incl);  **int** excl\_new = incl;  incl = incl\_new;  excl = excl\_new;  }  **return** Math.*min*(incl, excl);  }  } |

# **Number of permutation with K inversions**

|  |
| --- |
| Input : N = 3, K = 1 Output : 2  Explanation : Total Permutation of first N number, 123, 132, 213, 231, 312, 321 Permutation with 1 inversion : 132 and 213  Input : N = 4, K = 2 Output : 2  **public** **class** NumberOfPermutation {  **static** **int** *M* = 100;  **static** **int** *memo*[][] = **new** **int**[*M*][*M*];  **public** **static** **void** main(String[] args) {  **int** N = 4;  **int** K = 2;  System.***out***.println(*numberOfPermWithKInversion*(N, K));//5  }  **static** **int** numberOfPermWithKInversion(**int** N, **int** K) {  **if** (N == 0) **return** 0;  **if** (K == 0) **return** 1;  **if** (*memo*[N][K] != 0)**return** *memo*[N][K];  **int** sum = 0;  **for** (**int** i = 0; i <= K; i++) {  **if** (i <= N - 1)  sum += *numberOfPermWithKInversion*(N - 1, K - i);  }  *memo*[N][K] = sum;  **return** sum;  }  } |

# **Number of subsequences of the form a^i b^j c^k**

|  |
| --- |
| Expected Time Complexity : O(n)  Input : abbc Output : 3 Subsequences are abc, abc and abbc  Input : abcabc Output : 7 Subsequences are abc, abc, abbc, aabc abcc, abc and abc  **public** **class** NumberOfSubsequences {  **public** **static** **void** main(String[] args) {  String s = "abbc";  System.***out***.println(*countSubsequences*(s)); //3  }  **static** **int** countSubsequences(String s) {  **int** aCount = 0;  **int** bCount = 0;  **int** cCount = 0;  **for** (**int** i = 0; i < s.length(); i++) {  **if** (s.charAt(i) == 'a')  aCount = (1 + 2 \* aCount);  **else** **if** (s.charAt(i) == 'b')  bCount = (aCount + 2 \* bCount);  **else** **if** (s.charAt(i) == 'c')  cCount = (bCount + 2 \* cCount);  }  **return** cCount;  }  } |

# **Find number of times a string occurs as a subsequence in given string**

|  |
| --- |
| Given two strings, find the number of times the second string occurs in the first string, whether continuous or discontinuous. Input: string a = "GeeksforGeeks" string b = "Gks" Output: 4  **public** **class** NumberTimesStringOccursSubsequence {  **public** **static** **void** main(String[] args) {  String a = "GeeksforGeeks";  String b = "Gks";  System.***out***.println(*count*(a, b, a.length(), b.length())); //4  }  **static** **int** count(String a, String b, **int** m, **int** n) {  **if** ((m == 0 && n == 0) || n == 0) **return** 1;  **if** (m == 0) **return** 0;  **if** (a.charAt(m - 1) == b.charAt(n - 1)) **return** *count*(a, b, m - 1, n - 1) + *count*(a, b, m - 1, n);  **else**  **return** *count*(a, b, m - 1, n);  }  } |

# **Optimal Binary Search Tree**

|  |
| --- |
| Overlapping Subproblems Example 1  Input: keys[] = {10, 12}, freq[] = {34, 50}  There can be following two possible BSTs  10 12  \ /  12 10  I II  Frequency of searches of 10 and 12 are 34 and 50 respectively.  The cost of tree I is 34\*1 + 50\*2 = 134  The cost of tree II is 50\*1 + 34\*2 = 118  **public** **class** OptimalBinarySearchTree {  **public** **static** **void** main(String[] args) {  **int** keys[] = { 10, 12, 20 };  **int** freq[] = { 34, 8, 50 };  **int** n = keys.length;  System.***out***.println("Cost of Optimal BST is " + *optimalSearchTree*(keys, freq, n)); //Cost of Optimal BST is 142  }  **static** **int** optimalSearchTree(**int** keys[], **int** freq[], **int** n) {  **return** *optCost*(freq, 0, n - 1);  }  **static** **int** optCost(**int** freq[], **int** i, **int** j) {  **if** (j < i) **return** 0;  **if** (j == i) **return** freq[i];  **int** fsum = *sum*(freq, i, j);  **int** min = Integer.***MAX\_VALUE***;  **for** (**int** r = i; r <= j; ++r) {  **int** cost = *optCost*(freq, i, r - 1) + *optCost*(freq, r + 1, j);  **if** (cost < min)  min = cost;  }  **return** min + fsum;  }  **static** **int** sum(**int** freq[], **int** i, **int** j) {  **int** s = 0;  **for** (**int** k = i; k <= j; k++)  s += freq[k];  **return** s;  }  } |

# **Optimal Binary Search Tree1**

|  |
| --- |
| //Dynamic Programming Solution  **public** **class** OptimalBinarySearchTree1 {  **public** **static** **void** main(String[] args) {  **int** keys[] = { 10, 12, 20 };  **int** freq[] = { 34, 8, 50 };  **int** n = keys.length;  System.***out***.println("Cost of Optimal BST is " + *optimalSearchTree*(keys, freq, n)); //Cost of Optimal BST is 142  }  **static** **int** optimalSearchTree(**int** keys[], **int** freq[], **int** n) {  **int** cost[][] = **new** **int**[n + 1][n + 1];  **for** (**int** i = 0; i < n; i++)  cost[i][i] = freq[i];  **for** (**int** L = 2; L <= n; L++) {  **for** (**int** i = 0; i <= n - L + 1; i++) {  **int** j = i + L - 1;  cost[i][j] = Integer.***MAX\_VALUE***;  **for** (**int** r = i; r <= j; r++) {  **int** c = ((r > i) ? cost[i][r - 1] : 0) + ((r < j) ? cost[r + 1][j] : 0) + *sum*(freq, i, j);  **if** (c < cost[i][j])  cost[i][j] = c;  }  }  }  **return** cost[0][n - 1];  }  **static** **int** sum(**int** freq[], **int** i, **int** j) {  **int** s = 0;  **for** (**int** k = i; k <= j; k++) {  **if** (k >= freq.length)  **continue**;  s += freq[k];  }  **return** s;  }  } |

# **Optimal Strategy Game**

|  |
| --- |
| Problem statement: Consider a row of n coins of values v1 . . . vn, where n is even. We play a game against an opponent by alternating turns.In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin. Determine the maximum possible amount of money we can definitely win if we move first.  **public** **class** OptimalStrategyGame {  **int** max(**int** a, **int** b) {  **return** a > b ? a : b;  }  **int** min(**int** a, **int** b) {  **return** a < b ? a : b;  }  **public** **static** **void** main(String[] args) {  **int** arr1[] = { 8, 15, 3, 7 };  **int** n = arr1.length;  System.***out***.println("" + *optimalStrategyOfGame*(arr1, n)); //22  **int** arr2[] = { 2, 2, 2, 2 };  n = arr2.length;  System.***out***.println("" + *optimalStrategyOfGame*(arr2, n));//4  **int** arr3[] = { 20, 30, 2, 2, 2, 10 };  n = arr3.length;  System.***out***.println("" + *optimalStrategyOfGame*(arr3, n));//42  }  **static** **int** optimalStrategyOfGame(**int** arr[], **int** n) {  **int** table[][] = **new** **int**[n][n];  **int** gap, i, j, x, y, z;  **for** (gap = 0; gap < n; ++gap) {  **for** (i = 0, j = gap; j < n; ++i, ++j) {  x = ((i + 2) <= j)? table[i + 2][j] : 0;  y = ((i + 1) <= (j - 1))? table[i + 1][j - 1] : 0;  z = (i <= (j - 2))? table[i][j - 2] : 0;  table[i][j] = Math.*max*(arr[i] + Math.*min*(x, y), arr[j] + Math.*min*(y, z));  }  }  **return** table[0][n - 1];  }  } |

# **Palindrome Partitioning**

|  |
| --- |
| Given a string, a partitioning of the string is a palindrome partitioning if every substring of the partition is a palindrome. For example, “aba|b|bbabb|a|b|aba” is a palindrome partitioning of “ababbbabbababa”. \*/  For example, minimum 3 cuts are needed for “ababbbabbababa”. The three cuts are “a|babbbab|b|ababa”.  If a string is palindrome, then minimum 0 cuts are needed.  If a string of length n containing all different characters, then minimum n-1 cuts are needed.\*/  //Min cuts needed for Palindrome Partitioning is 3  //Time Complexity: O(n3)  public class PalindromePartitioning {  public static void main(String[] args) {  String str = "ababbbabbababa";  System.out.println("Min cuts needed for " + "Palindrome Partitioning is " + minPalPartion(str));  }  static int minPalPartion(String str) {  int n = str.length();  int[][] C = new int[n][n];  boolean[][] P = new boolean[n][n];  int i, j, k, L;  for (i = 0; i < n; i++) {  P[i][i] = true;  C[i][i] = 0;  }  for (L = 2; L <= n; L++) {  for (i = 0; i < n - L + 1; i++) {  j = i + L - 1;  if (L == 2)  P[i][j] = (str.charAt(i) == str.charAt(j));  else  P[i][j] = (str.charAt(i) == str.charAt(j)) && P[i + 1][j - 1];  if (P[i][j] == true)  C[i][j] = 0;  else {  C[i][j] = Integer.MAX\_VALUE;  for (k = i; k <= j - 1; k++)  C[i][j] = Integer.min(C[i][j], C[i][k] + C[k + 1][j] + 1);  }  }  }  return C[0][n - 1];  }  } |

# **Palindrome Partitioning1**

|  |
| --- |
| //Time Complexity: O(n2) Min cuts needed for Palindrome Partitioning is 3  **public** **class** PalindromePartitioning1 {  **public** **static** **void** main(String[] args) {  String str = "ababbbabbababa";  System.***out***.println("Min cuts needed for " + "Palindrome Partitioning" + " is " + *minPalPartion*(str));  }  **static** **int** minPalPartion(String str) {  **int** n = str.length();  **int**[] C = **new** **int**[n];  **boolean**[][] P = **new** **boolean**[n][n];  **int** i, j, k, L;  **for** (i = 0; i < n; i++) {  P[i][i] = **true**;  }  **for** (L = 2; L <= n; L++) {  **for** (i = 0; i < n - L + 1; i++) {  j = i + L - 1;  **if** (L == 2)  P[i][j] = (str.charAt(i) == str.charAt(j));  **else**  P[i][j] = (str.charAt(i) == str.charAt(j)) && P[i + 1][j - 1];  }  }  **for** (i = 0; i < n; i++) {  **if** (P[0][i] == **true**)  C[i] = 0;  **else** {  C[i] = Integer.***MAX\_VALUE***;  **for** (j = 0; j < i; j++) {  **if** (P[j + 1][i] == **true** && 1 + C[j] < C[i])  C[i] = 1 + C[j];  }  }  }  **return** C[n - 1];  }  } |

# **Partition Problem**

|  |
| --- |
| Partition problem is to determine whether a given set can be partitioned into two subsets such that the sum of elements in both subsets is same.  arr[] = {1, 5, 11, 5} Output: true The array can be partitioned as {1, 5, 5} and {11}  arr[] = {1, 5, 3} Output: false The array cannot be partitioned into equal sum sets.  Time Complexity: O(2^n) In worst case  **public** **class** PartitionProblem {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 1, 5, 9, 12 };  **int** n = arr.length;  **if** (*findPartition*(arr, n) == **true**)  System.***out***.println("Can be divided into two " + "subsets of equal sum"); //Can be divided into two subsets of equal sum  **else**  System.***out***.println("Can not be divided into " + "two subsets of equal sum");  }  **static** **boolean** isSubsetSum(**int** arr[], **int** n, **int** sum) {  **if** (sum == 0)  **return** **true**;  **if** (n == 0 && sum != 0)  **return** **false**;  **if** (arr[n - 1] > sum)  **return** *isSubsetSum*(arr, n - 1, sum);  **return** *isSubsetSum*(arr, n - 1, sum) || *isSubsetSum*(arr, n - 1, sum - arr[n - 1]);  }  **static** **boolean** findPartition(**int** arr[], **int** n) {  **int** sum = 0;  **for** (**int** i = 0; i < n; i++)  sum += arr[i];  **if** (sum % 2 != 0)  **return** **false**;  **return** *isSubsetSum*(arr, n, sum / 2);  }  } |

# **Partition Problem1**

|  |
| --- |
| Time Complexity: O(sum\*n) Auxiliary Space: O(sum\*n)  **public** **class** PartitionProblem1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 1, 1, 2, 2, 1 };  **int** n = arr.length;  **if** (*findPartition*(arr, n) == **true**)  System.***out***.println("Can be divided into two subsets of equal sum"); //Can be divided into two subsets of equal sum  **else**  System.***out***.println("Can not be divided into two subsets of equal sum");  }  **static** **boolean** findPartition(**int** arr[], **int** n) {  **int** sum = 0;  **int** i, j;  **for** (i = 0; i < n; i++)  sum += arr[i];  **if** (sum % 2 != 0)  **return** **false**;  **boolean** part[][] = **new** **boolean**[sum / 2 + 1][n + 1];  **for** (i = 0; i <= n; i++)  part[0][i] = **true**;  **for** (i = 1; i <= sum / 2; i++)  part[i][0] = **false**;  **for** (i = 1; i <= sum / 2; i++) {  **for** (j = 1; j <= n; j++) {  part[i][j] = part[i][j - 1];  **if** (i >= arr[j - 1])  part[i][j] = part[i][j] || part[i - arr[j - 1]][j - 1];  }  }  **return** part[sum / 2][n];  }  } |

# **Partition Set into Two Subsets**

|  |
| --- |
| Recursive Solution:-  Partition a set into two subsets such that the difference of subset sums is minimum  Given a set of integers, the task is to divide it into two sets S1 and S2 such that the absolute difference between their sums is minimum.  Input: arr[] = {1, 6, 11, 5} Output: 1  Explanation:  Subset1 = {1, 5, 6}, sum of Subset1 = 12  Subset2 = {11}, sum of Subset2 = 11  **public** **class** PartitionSetIntoTwoSubsets {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 1, 4, 2, 2, 1 };  **int** n = arr.length;  System.***out***.print("The minimum difference" + " between two sets is " + *findMin*(arr, n)); //1  }  **public** **static** **int** findMin(**int** arr[], **int** n) {  **int** sumTotal = 0;  **for** (**int** i = 0; i < n; i++)  sumTotal += arr[i];  **return** *findMinRec*(arr, n, 0, sumTotal);  }  **public** **static** **int** findMinRec(**int** arr[], **int** i, **int** sumCalculated, **int** sumTotal) {  **if** (i == 0)  **return** Math.*abs*((sumTotal - sumCalculated) - sumCalculated);  **return** Math.*min*(*findMinRec*(arr, i - 1, sumCalculated + arr[i - 1], sumTotal),  *findMinRec*(arr, i - 1, sumCalculated, sumTotal));  }  } |

# **Partition Set into Two Subsets1**

|  |
| --- |
| //Dynamic Programming  **public** **class** PartitionSetIntoTwoSubsets1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 3, 1, 4, 2, 2, 1 };  **int** n = arr.length;  System.***out***.println("The minimum difference between 2 sets is " + *findMin*(arr, n)); //1  }  **static** **int** findMin(**int** arr[], **int** n) {  **int** sum = 0;  **for** (**int** i = 0; i < n; i++)  sum += arr[i];  **boolean** dp[][] = **new** **boolean**[n + 1][sum + 1];  **for** (**int** i = 0; i <= n; i++)  dp[i][0] = **true**;  **for** (**int** i = 1; i <= sum; i++)  dp[0][i] = **false**;  **for** (**int** i = 1; i <= n; i++) {  **for** (**int** j = 1; j <= sum; j++) {  dp[i][j] = dp[i - 1][j];  **if** (arr[i - 1] <= j)  dp[i][j] |= dp[i - 1][j - arr[i - 1]];  }  }  **int** diff = Integer.***MAX\_VALUE***;  **for** (**int** j = sum / 2; j >= 0; j--) {  **if** (dp[n][j] == **true**) {  diff = sum - 2 \* j;  **break**;  }  }  **return** diff;  }  } |

# **Path with Maximum Average Value**

|  |
| --- |
| Given a square matrix of size N\*N, where each cell is associated with a specific cost. A path is defined as a specific sequence of cells which starts from top left cell move only right or down and ends on bottom right cell. We want to find a path with maximum average over all existing paths. Average is computed as total cost divided by number of cells visited in path.  Input : Matrix = [1, 2, 3  4, 5, 6  7, 8, 9]  Output : 5.8 Path with maximum average is, 1 -> 4 -> 7 -> 8 -> 9 Sum of the path is 29 and average is 29/5 = 5.8  **public** **class** PathWithMaximumAverageValue {  **public** **static** **void** main(String[] args) {  **int** cost[][] = { { 1, 2, 3 }, { 6, 5, 4 }, { 7, 3, 9 } };  System.***out***.println(*maxAverageOfPath*(cost, 3));//5.2  }  **public** **static** **double** maxAverageOfPath(**int** cost[][], **int** N) {  **int** dp[][] = **new** **int**[N + 1][N + 1];  dp[0][0] = cost[0][0];  **for** (**int** i = 1; i < N; i++)  dp[i][0] = dp[i - 1][0] + cost[i][0];  **for** (**int** j = 1; j < N; j++)  dp[0][j] = dp[0][j - 1] + cost[0][j];  **for** (**int** i = 1; i < N; i++)  **for** (**int** j = 1; j < N; j++)  dp[i][j] = Math.*max*(dp[i - 1][j], dp[i][j - 1]) + cost[i][j];  **return** (**double**) dp[N - 1][N - 1] / (2 \* N - 1);  }  } |

# **Perfect Sum Problem**

|  |
| --- |
| Perfect Sum Problem (Print all subsets with given sum). Given an array of integers and a sum, the task is to print all subsets of given array with sum equal to given sum.  Input : arr[] = {2, 3, 5, 6, 8, 10} sum = 10  Output : 5 2 3  2 8  10  **public** **class** PerfectSumProblem {  **static** **boolean**[][] *dp*;  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 2, 3, 4, 5 };  **int** n = arr.length;  **int** sum = 10;  *printAllSubsets*(arr, n, sum);  }  **static** **void** printAllSubsets(**int** arr[], **int** n, **int** sum) {  **if** (n == 0 || sum < 0) **return**;  *dp* = **new** **boolean**[n][sum + 1];  **for** (**int** i = 0; i < n; ++i) {  *dp*[i][0] = **true**;  }  **if** (arr[0] <= sum)  *dp*[0][arr[0]] = **true**;  **for** (**int** i = 1; i < n; ++i)  **for** (**int** j = 0; j < sum + 1; ++j)  *dp*[i][j] = (arr[i] <= j) ? (*dp*[i - 1][j] || *dp*[i - 1][j - arr[i]]) : *dp*[i - 1][j];  **if** (*dp*[n - 1][sum] == **false**) {  System.***out***.println("There are no subsets with" + " sum " + sum);  **return**;  }  ArrayList<Integer> p = **new** ArrayList<>();  *printSubsetsRec*(arr, n - 1, sum, p);  }  **static** **void** printSubsetsRec(**int** arr[], **int** i, **int** sum, ArrayList<Integer> p) {  **if** (i == 0 && sum != 0 && *dp*[0][sum]) {  p.add(arr[i]);  *display*(p);  p.clear();  **return**;  }  **if** (i == 0 && sum == 0) {  *display*(p);  p.clear();  **return**;  }  **if** (*dp*[i - 1][sum]) {  ArrayList<Integer> b = **new** ArrayList<>();  b.addAll(p);  *printSubsetsRec*(arr, i - 1, sum, b);  }  **if** (sum >= arr[i] && *dp*[i - 1][sum - arr[i]]) {  p.add(arr[i]);  *printSubsetsRec*(arr, i - 1, sum - arr[i], p);  }  }  **static** **void** display(ArrayList<Integer> v) {  System.***out***.println(v);  /\*[4, 3, 2, 1]  [5, 3, 2]  [5, 4, 1]\*/  }  } |

# **Permutation**

|  |
| --- |
| A Derangement is a permutation of n elements, such that no element appears in its original position.  For example, a derangement of {0, 1, 2, 3} is {2, 3, 1, 0}. Count Derangements (Permutation such that no element appears in its original position)  Input: n = 2 Output: 1 For two elements say {0, 1}, there is only one possible derangement {1, 0}  Time Complexity: T(n) = T(n-1) + T(n-2) which is exponential.  **public** **class** Permutation {  **public** **static** **void** main(String[] args) {  **int** n = 4;  System.***out***.println("Count of Derangements is " + *countDer*(n)); //Count of Derangements is 9  }  **static** **int** countDer(**int** n) {  **if** (n == 1) **return** 0;  **if** (n == 0) **return** 1;  **if** (n == 2) **return** 1;  // countDer(n) = (n-1)[countDer(n-1) + der(n-2)]  **return** (n - 1) \* (*countDer*(n - 1) + *countDer*(n - 2));  }  } |

# **Permutation1**

|  |
| --- |
| //A Dynamic programming based  **public** **class** Permutation1 {  **public** **static** **void** main(String[] args) {  **int** n = 4;  System.***out***.println("Count of Derangements is " + *countDer*(n)); //Count of Derangements is 9  }  **static** **int** countDer(**int** n) {  **int** der[] = **new** **int**[n + 1];  der[0] = 1;  der[1] = 0;  der[2] = 1;  **for** (**int** i = 3; i <= n; ++i)  der[i] = (i - 1) \* (der[i - 1] + der[i - 2]);  **return** der[n];  }  } |

# **Print Maximum Number**

|  |
| --- |
| How to print maximum number of A’s using given four keys. Imagine you have a special keyboard with the following keys:  Key 1: Prints 'A' on screen  Key 2: (Ctrl-A): Select screen  Key 3: (Ctrl-C): Copy selection to buffer  Key 4: (Ctrl-V): Print buffer on screen appending it after what has already been printed.  If you can only press the keyboard for N times (with the above four keys), write a program to produce maximum numbers of A's. That is to say, the input parameter is N (No. of keys that you can press), the output is M (No. of As that you can produce).  Input: N = 3 Output: 3 We can at most get 3 A's on screen by pressing following key sequence. A, A, A  Input: N = 7 Output: 9 We can at most get 9 A's on screen by pressing following key sequence.  A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V  **public** **class** PrintMaximumNumber {  **public** **static** **void** main(String[] args) {  **int** N;  **for** (N = 1; N <= 20; N++)  System.***out***.println("Maximum Number of A's with keystrokes is " + N + *findoptimal*(N));  }  **static** **int** findoptimal(**int** N) {  **if** (N <= 6) **return** N;  **int** max = 0;  **int** b;  **for** (b = N - 3; b >= 1; b--) {  **int** curr = (N - b - 1) \* *findoptimal*(b);  **if** (curr > max)  max = curr;  }  **return** max;  }  } |

# **Print Maximum Number1**

|  |
| --- |
| Input: N = 11 Output: 27 We can at most get 27 A's on screen by pressing following key sequence.  A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V, Ctrl A, Ctrl C, Ctrl V, Ctrl V  **public** **class** PrintMaximumNumber1 {  **public** **static** **void** main(String[] args) {  **int** N;  **for** (N = 1; N <= 20; N++)  System.***out***.println("Maximum Number of A's with keystrokes is " + N + *findoptimal*(N));  }  **static** **int** findoptimal(**int** N) {  **if** (N <= 6) **return** N;  **int** screen[] = **new** **int**[N];  **int** b;  **int** n;  **for** (n = 1; n <= 6; n++)  screen[n - 1] = n;  **for** (n = 7; n <= N; n++) {  screen[n - 1] = 0;  **for** (b = n - 3; b >= 1; b--) {  **int** curr = (n - b - 1) \* screen[b - 1];  **if** (curr > screen[n - 1])  screen[n - 1] = curr;  }  }  **return** screen[N - 1];  }  } |

# **Probability of getting at least K heads in N tosses of Coins**

|  |
| --- |
| Given N number of coins, the task is to find probability of getting at least K number of heads after tossing all the N coins simultaneously. Suppose we have 3 unbiased coins and we have to find the probability of getting at least 2 heads, so there are 23 = 8 ways to toss these coins, i.e.,HHH, HHT, HTH, HTT, THH, THT, TTH, TTT  Out of which there are 4 set which contain at least 2 Heads i.e., HHH, HHT, HH, THH  So the probability is 4/8 or 0.5 Time Complexity: O(n) where n < 20 Auxiliary space: O(n)  **public** **class** Probability {  **public** **static** **double** *fact*[];  **public** **static** **void** main(String[] args) {  *fact* = **new** **double**[100];  *precompute*();  System.***out***.println(*probability*(2, 3));//0.5  }  **public** **static** **void** precompute() {  *fact*[0] = *fact*[1] = 1;  **for** (**int** i = 2; i < 20; ++i)  *fact*[i] = *fact*[i - 1] \* i;  }  **public** **static** **double** probability(**int** k, **int** n) {  **double** ans = 0;  **for** (**int** i = k; i <= n; ++i)  ans += *fact*[n] / (*fact*[i] \* *fact*[n - i]);  ans = ans / (1 << n);  **return** ans;  }  } |

# **Recursively break a number in 3 parts to get maximum sum**

|  |
| --- |
| Given a number n, we can divide it in only three parts n/2, n/3 and n/4 (we will consider only integer part). The task is to find the maximum sum we can make by dividing number in three parts recursively and summing up them together.  Input : n = 12 Output : 13  //A simple recursive JAVA program to find maximum sum by recursively breaking a number in 3 parts.  **public** **class** RecursivelBreakNumber {  **public** **static** **void** main(String[] args) {  **int** n = 12;  System.***out***.println(*breakSum*(n)); //13  }  **static** **int** breakSum(**int** n) {  **if** (n == 0 || n == 1)  **return** n;  **return** Math.*max*((*breakSum*(n / 2) + *breakSum*(n / 3) + *breakSum*(n / 4)), n);  }  } |

# **Recursively break a number in 3 parts to get maximum sum using Dynamic programming. Time Complexity: O (n) Auxiliary Space: O (n)**

|  |
| --- |
| **public** **class** RecursivelBreakNumber1 {  **final** **int** MAX = 1000000;  **public** **static** **void** main(String[] args) {  **int** n = 24;  System.***out***.println(*breakSum*(n));//27  }  **static** **int** breakSum(**int** n) {  **int** dp[] = **new** **int**[n + 1];  dp[0] = 0;  dp[1] = 1;  **for** (**int** i = 2; i <= n; i++)  dp[i] = Math.*max*(dp[i / 2] + dp[i / 3] + dp[i / 4], i);  **return** dp[n];  }  } |

# **Given two strings str1 and str2, find the shortest string that has both str1 and str2 as subsequences.**

|  |
| --- |
| Input: str1 = "geek", str2 = "eke" Output: "geeke"  Input: str1 = "AGGTAB", str2 = "GXTXAYB" Output: "AGXGTXAYB"  **public** **class** ShortestCommonSupersequence {  **public** **static** **void** main(String[] args) {  String X = "AGGTAB";  String Y = "GXTXAYB";  System.***out***.println("Length of the shortest supersequence is " + *shortestSuperSequence*(X, Y)); //9  }  **static** **int** shortestSuperSequence(String X, String Y) {  **int** m = X.length();  **int** n = Y.length();  **int** l = *lcs*(X, Y, m, n); // find lcs  **return** (m + n - l);  }  **static** **int** lcs(String X, String Y, **int** m, **int** n) {  **int**[][] L = **new** **int**[m + 1][n + 1];  **int** i, j;  **for** (i = 0; i <= m; i++) {  **for** (j = 0; j <= n; j++) {  **if** (i == 0 || j == 0)  L[i][j] = 0;  **else** **if** (X.charAt(i - 1) == Y.charAt(j - 1))  L[i][j] = L[i - 1][j - 1] + 1;  **else**  L[i][j] = Math.*max*(L[i - 1][j], L[i][j - 1]);  }  }  **return** L[m][n];  }  } |

# **Shortest Common Supersequence1**

|  |
| --- |
| **public** **class** ShortestCommonSupersequence1 {  **public** **static** **void** main(String[] args) {  String X = "AGGTAB";  String Y = "GXTXAYB";  System.***out***.println("Length of the shortest supersequence is: " + *superSeq*(X, Y, X.length(), Y.length())); //9  }  **static** **int** superSeq(String X, String Y, **int** m, **int** n) {  **if** (m == 0)  **return** n;  **if** (n == 0)  **return** m;  **if** (X.charAt(m - 1) == Y.charAt(n - 1))  **return** 1 + *superSeq*(X, Y, m - 1, n - 1);  **return** 1 + Math.*min*(*superSeq*(X, Y, m - 1, n), *superSeq*(X, Y, m, n - 1));  }  } |

# **Shortest Common Supersequence2**

|  |
| --- |
| **public** **class** ShortestCommonSupersequence2 {  **public** **static** **void** main(String[] args) {  String X = "AGGTAB";  String Y = "GXTXAYB";  System.***out***.println("Length of the shortest supersequence is " + *superSeq*(X, Y, X.length(), Y.length()));//9  }  **static** **int** superSeq(String X, String Y, **int** m, **int** n) {  **int**[][] dp = **new** **int**[m + 1][n + 1];  **for** (**int** i = 0; i <= m; i++) {  **for** (**int** j = 0; j <= n; j++) {  **if** (i == 0)  dp[i][j] = j;  **else** **if** (j == 0)  dp[i][j] = i;  **else** **if** (X.charAt(i - 1) == Y.charAt(j - 1))  dp[i][j] = 1 + dp[i - 1][j - 1];  **else**  dp[i][j] = 1 + Math.*min*(dp[i - 1][j], dp[i][j - 1]);  }  }  **return** dp[m][n];  }  } |

# **Shortest path with exactly k edges in a directed and weighted graph a simple solution**

|  |
| --- |
| **public** **class** ShortestPath {  **static** **final** **int** ***V*** = 4;  **static** **final** **int** ***INF*** = Integer.***MAX\_VALUE***;  **public** **static** **void** main(String[] args) {  **int** graph[][] = **new** **int**[][] { { 0, 10, 3, 2 }, { ***INF***, 0, ***INF***, 7 }, { ***INF***, ***INF***, 0, 6 }, { ***INF***, ***INF***, ***INF***, 0 } };  ShortestPath t = **new** ShortestPath();  **int** u = 0, v = 3, k = 2;  System.***out***.println("Weight of the shortest path is " + t.shortestPath(graph, u, v, k)); // 9  }  **int** shortestPath(**int** graph[][], **int** u, **int** v, **int** k) {  **if** (k == 0 && u == v)  **return** 0;  **if** (k == 1 && graph[u][v] != ***INF***)  **return** graph[u][v];  **if** (k <= 0)  **return** ***INF***;  **int** res = ***INF***;  **for** (**int** i = 0; i < ***V***; i++) {  **if** (graph[u][i] != ***INF*** && u != i && v != i) {  **int** rec\_res = shortestPath(graph, i, v, k - 1);  **if** (rec\_res != ***INF***)  res = Math.*min*(res, graph[u][i] + rec\_res);  }  }  **return** res;  }  } |

# **Shortest path with exactly k edges in a directed and weighted graph using Dynamic Programming.**

|  |
| --- |
| **public** **class** ShortestPath1 {  **static** **final** **int** ***V*** = 4;  **static** **final** **int** ***INF*** = Integer.***MAX\_VALUE***;  **public** **static** **void** main(String[] args) {  **int** graph[][] = **new** **int**[][] { { 0, 10, 3, 2 }, { ***INF***, 0, ***INF***, 7 }, { ***INF***, ***INF***, 0, 6 }, { ***INF***, ***INF***, ***INF***, 0 } };  ShortestPath t = **new** ShortestPath();  **int** u = 0, v = 3, k = 2;  System.***out***.println("Weight of the shortest path is " + t.shortestPath(graph, u, v, k)); 9  }  **int** shortestPath(**int** graph[][], **int** u, **int** v, **int** k) {  **int** sp[][][] = **new** **int**[***V***][***V***][k + 1];  **for** (**int** e = 0; e <= k; e++) {  **for** (**int** i = 0; i < ***V***; i++) {  **for** (**int** j = 0; j < ***V***; j++) {  sp[i][j][e] = ***INF***;  **if** (e == 0 && i == j)  sp[i][j][e] = 0;  **if** (e == 1 && graph[i][j] != ***INF***)  sp[i][j][e] = graph[i][j];  **if** (e > 1) {  **for** (**int** a = 0; a < ***V***; a++) {  **if** (graph[i][a] != ***INF*** && i != a && j != a && sp[a][j][e - 1] != ***INF***)  sp[i][j][e] = Math.*min*(sp[i][j][e], graph[i][a] + sp[a][j][e - 1]);  }  }  }  }  }  **return** sp[u][v][k];  }  } |

# **Subset with sum divisible by m: -** Given a set of non-negative distinct integers, and a value m, determine if there is a subset of the given set with sum divisible by m.

|  |
| --- |
| Input Constraints Size of set i.e., n <= 1000000, m <= 1000  Input : arr[] = {3, 1, 7, 5}; m = 6; Output : YES  Time Complexity : O(m^2) Auxiliary Space : O(m)  **public** **class** SubsetWithSumDivisible {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 7 };  **int** n = arr.length;  **int** m = 5;  **if** (*modularSum*(arr, n, m))  System.***out***.print("YES\n");  **else**  System.***out***.print("NO\n"); //NO  }  **static** **boolean** modularSum(**int** arr[], **int** n, **int** m) {  **if** (n > m)  **return** **true**;  **boolean** DP[] = **new** **boolean**[m];  Arrays.*fill*(DP, **false**);  **for** (**int** i = 0; i < n; i++) {  **if** (DP[0])  **return** **true**;  **boolean** temp[] = **new** **boolean**[m];  Arrays.*fill*(temp, **false**);  **for** (**int** j = 0; j < m; j++) {  **if** (DP[j] == **true**) {  **if** (DP[(j + arr[i]) % m] == **false**)  temp[(j + arr[i]) % m] = **true**;  }  }  **for** (**int** j = 0; j < m; j++)  **if** (temp[j])  DP[j] = **true**;  DP[arr[i] % m] = **true**;  }  **return** DP[0];  }  } |

# **Sum of average of all subsets**

|  |
| --- |
| Given an array arr of N integer elements, the task is to find sum of average of all subsets of this array.  Input : arr[] = [2, 3, 5] Output : 23.33  **public** **class** SumAverageAllSubsets {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 2, 3, 5, 7 };  **int** N = arr.length;  System.***out***.println(*resultOfAllSubsets*(arr, N)); //63.75  }  **static** **double** resultOfAllSubsets(**int** arr[], **int** N) {  **double** result = 0.0;  **int** sum = 0;  **for** (**int** i = 0; i < N; i++)  sum += arr[i];  **for** (**int** n = 1; n <= N; n++)  result += (**double**) (sum \* (*nCr*(N - 1, n - 1))) / n;  **return** result;  }  **static** **int** nCr(**int** n, **int** k) {  **int** C[][] = **new** **int**[n + 1][k + 1];  **int** i, j;  **for** (i = 0; i <= n; i++) {  **for** (j = 0; j <= Math.*min*(i, k); j++) {  **if** (j == 0 || j == i)  C[i][j] = 1;  **else**  C[i][j] = C[i - 1][j - 1] + C[i - 1][j];  }  }  **return** C[n][k];  }  } |

# **Total number of non-decreasing numbers with n digits**

|  |
| --- |
| Given the number of digits n, you are required to find the count of total non-decreasing numbers with n digits.  Input: n = 1 Output: count = 10  Input: n = 2 Output: count = 55  **public** **class** TotalNumberNonDecreasing {  **public** **static** **void** main(String[] args) {  **int** n = 3;  System.***out***.println(*countNonDecreasing*(n)); //220  }  **static** **int** countNonDecreasing(**int** n) {  **int** dp[][] = **new** **int**[10][n + 1];  **for** (**int** i = 0; i < 10; i++)  dp[i][1] = 1;  **for** (**int** digit = 0; digit <= 9; digit++) {  **for** (**int** len = 2; len <= n; len++) {  **for** (**int** x = 0; x <= digit; x++)  dp[digit][len] += dp[x][len - 1];  }  }  **int** count = 0;  **for** (**int** i = 0; i < 10; i++)  count += dp[i][n];  **return** count;  }  } |

# **Unbounded Knapsack (Repetition of items allowed)**

|  |
| --- |
| Given a knapsack weight W and a set of n items with certain value vali and weight wti, we need to calculate minimum amount that could make up this quantity exactly.  Input : W = 100 val[] = {1, 30} wt[] = {1, 50} Output : 100  **public** **class** UnboundedKnapsack {  **public** **static** **void** main(String[] args) {  **int** W = 100;  **int** val[] = { 10, 30, 20 };  **int** wt[] = { 5, 10, 15 };  **int** n = val.length;  System.***out***.println(*unboundedKnapsack*(W, n, val, wt));//300  }  **private** **static** **int** unboundedKnapsack(**int** W, **int** n, **int**[] val, **int**[] wt) {  **int** dp[] = **new** **int**[W + 1];  **for** (**int** i = 0; i <= W; i++) {  **for** (**int** j = 0; j < n; j++) {  **if** (wt[j] <= i) {  dp[i] = *max*(dp[i], dp[i - wt[j]] + val[j]);  }  }  }  **return** dp[W];  }  **private** **static** **int** max(**int** i, **int** j) {  **return** (i > j) ? i : j;  }  } |

# **Ways to arrange Balls such that adjacent balls are of different types**

|  |
| --- |
| There are ‘p’ balls of type P, ‘q’ balls of type Q and ‘r’ balls of type R. Using the balls we want to create a straight line such that no two balls of same type are adjacent.  Naive Solution: Input : p = 1, q = 1, r = 0 Output : 2 There are only two arrangements PQ and QP  **public** **class** WaysToArrangeBalls {  **public** **static** **void** main(String[] args) {  **int** p = 1, q = 1, r = 1;  System.***out***.print(*countUtil*(p, q, r)); //6  }  **static** **int** countUtil(**int** p, **int** q, **int** r) {  **return** *countWays*(p, q, r, 0) + *countWays*(p, q, r, 1) + *countWays*(p, q, r, 2);  }  **static** **int** countWays(**int** p, **int** q, **int** r, **int** last) {  **if** (p < 0 || q < 0 || r < 0) **return** 0;  **if** (p == 1 && q == 0 && r == 0 && last == 0) **return** 1;  **if** (p == 0 && q == 1 && r == 0 && last == 1) **return** 1;  **if** (p == 0 && q == 0 && r == 1 && last == 2) **return** 1;  **if** (last == 0)  **return** *countWays*(p - 1, q, r, 1) + *countWays*(p - 1, q, r, 2);  **if** (last == 1)  **return** *countWays*(p, q - 1, r, 0) + *countWays*(p, q - 1, r, 2);  **if** (last == 2)  **return** *countWays*(p, q, r - 1, 0) + *countWays*(p, q, r - 1, 1);  **return** 0;  }  } |

# **Ways to arrange Balls such that adjacent balls are of different types using Dynamic Programming**

|  |
| --- |
| Input : p = 2, q = 1, r = 1 Output : 6 There are only six arrangements PQRP, QPRP,PRQP, RPQP, PRPQ and PQPR  **public** **class** WaysToArrangeBalls1 {  **static** **final** **int** ***MAX*** = 100;  **static** **int** *dp*[][][][] = **new** **int**[***MAX***][***MAX***][***MAX***][3];  **public** **static** **void** main(String[] args) {  **int** p = 1, q = 1, r = 1;  System.***out***.print(*countUtil*(p, q, r)); //6  }  **static** **int** countUtil(**int** p, **int** q, **int** r) {  **for** (**int**[][][] row : *dp*) {  **for** (**int**[][] innerRow : row) {  **for** (**int**[] innerInnerRow : innerRow) {  Arrays.*fill*(innerInnerRow, -1);  }  }  }  ;  **return** *countWays*(p, q, r, 0) + *countWays*(p, q, r, 1) + *countWays*(p, q, r, 2);  }  **static** **int** countWays(**int** p, **int** q, **int** r, **int** last) {  **if** (p < 0 || q < 0 || r < 0) **return** 0;  **if** (p == 1 && q == 0 && r == 0 && last == 0) **return** 1;  **if** (p == 0 && q == 1 && r == 0 && last == 1) **return** 1;  **if** (p == 0 && q == 0 && r == 1 && last == 2) **return** 1;  **if** (*dp*[p][q][r][last] != -1) **return** *dp*[p][q][r][last];  **if** (last == 0)  *dp*[p][q][r][last] = *countWays*(p - 1, q, r, 1) + *countWays*(p - 1, q, r, 2);  **else** **if** (last == 1)  *dp*[p][q][r][last] = *countWays*(p, q - 1, r, 0) + *countWays*(p, q - 1, r, 2);  **else**  *dp*[p][q][r][last] = *countWays*(p, q, r - 1, 0) + *countWays*(p, q, r - 1, 1);  **return** *dp*[p][q][r][last];  }  } |

# **Count number of ways to partition a set into k subsets**

|  |
| --- |
| Given two numbers n and k where n represents number of elements in a set, find number of ways to partition the set into k subsets.  Input: n = 3, k = 2 Output: 3 Explanation: Let the set be {1, 2, 3}, we can partition it into 2 subsets in following ways {{1,2}, {3}}, {{1}, {2,3}}, {{1,3}, {2}}  Input: n = 3, k = 1 Output: 1 Explanation: There is only one way {{1, 2, 3}}  **public** **class** waysToPartition {  **public** **static** **void** main(String[] args) {  System.***out***.println(*countP*(3, 2)); //3  }  **public** **static** **int** countP(**int** n, **int** k) {  **if** (n == 0 || k == 0 || k > n)  **return** 0;  **if** (k == 1 || k == n)  **return** 1;  // S(n+1, k) = k\*S(n, k) + S(n, k-1)  **return** (k \* *countP*(n - 1, k) + *countP*(n - 1, k - 1));  }  } |

# **Weighted Job Scheduling in O (n Log n) time**

|  |
| --- |
| Given N jobs where every job is represented by following three elements of it.   * Start Time * Finish Time * Profit or Value Associated   Find the maximum profit subset of jobs such that no two jobs in the subset overlap.  Input: Number of Jobs n = 4 Job Details {Start Time, Finish Time, Profit}  Job 1: {1, 2, 50}  Job 2: {3, 5, 20}  Job 3: {6, 19, 100}  Job 4: {2, 100, 200}  Output: The maximum profit is 250.  We can get the maximum profit by scheduling jobs 1 and 4.Note that there is longer schedules possible Jobs 1, 2 and 3 but the profit with this schedule is 20+50+100 which is less than 250.  **class** Job {  **int** start, finish, profit;  Job(**int** start, **int** finish, **int** profit) {  **this**.start = start;  **this**.finish = finish;  **this**.profit = profit;  }  }  **public** **class** WeightedJobScheduling {  **public** **static** **void** main(String[] args) {  Job jobs[] = { **new** Job(1, 2, 50), **new** Job(3, 5, 20), **new** Job(6, 19, 100), **new** Job(2, 100, 200) };  System.***out***.println("Optimal profit is " + *schedule*(jobs)); //Optimal profit is 250  }  **static** **public** **int** schedule(Job jobs[]) {  Arrays.*sort*(jobs, **new** JobComparator());  **int** n = jobs.length;  **int** table[] = **new** **int**[n];  table[0] = jobs[0].profit;  **for** (**int** i = 1; i < n; i++) {  **int** inclProf = jobs[i].profit;  **int** l = *binarySearch*(jobs, i);  **if** (l != -1)  inclProf += table[l];  table[i] = Math.*max*(inclProf, table[i - 1]);  }  **return** table[n - 1];  }  **static** **public** **int** binarySearch(Job jobs[], **int** index) {  **int** lo = 0, hi = index - 1;  **while** (lo <= hi) {  **int** mid = (lo + hi) / 2;  **if** (jobs[mid].finish <= jobs[index].start) {  **if** (jobs[mid + 1].finish <= jobs[index].start)  lo = mid + 1;  **else**  **return** mid;  } **else**  hi = mid - 1;  }  **return** -1;  }  }  **class** JobComparator **implements** Comparator<Job> {  **public** **int** compare(Job a, Job b) {  **return** a.finish < b.finish ? -1 : a.finish == b.finish ? 0 : 1;  }  } |

# **Given a text and a wildcard pattern, implement wildcard pattern matching algorithm that finds if wildcard pattern is matched with text**

|  |
| --- |
| The matching should cover the entire text (not partial text). The wildcard pattern can include the characters ‘?’ and ‘\*’ ‘?’ – matches any single character ‘\*’ – Matches any sequence of characters (including the empty sequence)  Text = "baaabab",  Pattern = “\*\*\*\*\*ba\*\*\*\*\*ab", output : true  Pattern = "baaa?ab", output : true  Pattern = "ba\*a?", output : true  Pattern = "a\*ab", output : false  Time complexity of above solution is O (m x n). Auxiliary space used is also O (m x n).  **public** **class** WildcardPatternMatching {  **public** **static** **void** main(String[] args) {  String str = "baaabab";  String pattern = "\*\*\*\*\*ba\*\*\*\*\*ab";  **if** (*strmatch*(str, pattern, str.length(), pattern.length()))  System.***out***.println("Yes"); //Yes  **else**  System.***out***.println("No");  }  **static** **boolean** strmatch(String str, String pattern, **int** n, **int** m) {  **if** (m == 0) **return** (n == 0);  **boolean**[][] lookup = **new** **boolean**[n + 1][m + 1];  **for** (**int** i = 0; i < n + 1; i++)  Arrays.*fill*(lookup[i], **false**);  lookup[0][0] = **true**;  **for** (**int** j = 1; j <= m; j++)  **if** (pattern.charAt(j - 1) == '\*')  lookup[0][j] = lookup[0][j - 1];  **for** (**int** i = 1; i <= n; i++) {  **for** (**int** j = 1; j <= m; j++) {  **if** (pattern.charAt(j - 1) == '\*')  lookup[i][j] = lookup[i][j - 1] || lookup[i - 1][j];  **else** **if** (pattern.charAt(j - 1) == '?' || str.charAt(i - 1) == pattern.charAt(j - 1))  lookup[i][j] = lookup[i - 1][j - 1];  **else**  lookup[i][j] = **false**;  }  }  **return** lookup[n][m];  }  } |

# **Given a sequence of words, and a limit on the number of characters that can be put in one line (line width). Put line breaks in the given sequence such that the lines are printed neatly. Assume that the length of each word is smaller than the line width.**

|  |
| --- |
| For example, consider the following string and line width M = 15 "Geeks for Geeks presents word wrap problem"  Following is the optimized arrangement of words in 3 lines Geeks for Geeks presents word wrap problem  The total extra spaces in line 1, line 2 and line 3 are 0, 2 and 3 respectively. So optimal value of total cost is 0 + 2\*2 + 3\*3 = 13  Time Complexity: O(n^2) Auxiliary Space: O(n^2)  **public** **class** WordWrapProblem {  **final** **int** MAX = Integer.***MAX\_VALUE***;  **public** **static** **void** main(String[] args) {  WordWrapProblem w = **new** WordWrapProblem();  **int** l[] = { 3, 2, 2, 5 };  **int** n = l.length;  **int** M = 6;  w.solveWordWrap(l, n, M);  }  **void** solveWordWrap(**int** l[], **int** n, **int** M) {  **int** extras[][] = **new** **int**[n + 1][n + 1];  **int** lc[][] = **new** **int**[n + 1][n + 1];  **int** c[] = **new** **int**[n + 1];  **int** p[] = **new** **int**[n + 1];  **for** (**int** i = 1; i <= n; i++) {  extras[i][i] = M - l[i - 1];  **for** (**int** j = i + 1; j <= n; j++)  extras[i][j] = extras[i][j - 1] - l[j - 1] - 1;  }  **for** (**int** i = 1; i <= n; i++) {  **for** (**int** j = i; j <= n; j++) {  **if** (extras[i][j] < 0)  lc[i][j] = MAX;  **else** **if** (j == n && extras[i][j] >= 0)  lc[i][j] = 0;  **else**  lc[i][j] = extras[i][j] \* extras[i][j];  }  }  c[0] = 0;  **for** (**int** j = 1; j <= n; j++) {  c[j] = MAX;  **for** (**int** i = 1; i <= j; i++) {  **if** (c[i - 1] != MAX && lc[i][j] != MAX && (c[i - 1] + lc[i][j] < c[j])) {  c[j] = c[i - 1] + lc[i][j];  p[j] = i;  }  }  }  printSolution(p, n);  }  **int** printSolution(**int** p[], **int** n) {  **int** k;  **if** (p[n] == 1)  k = 1;  **else**  k = printSolution(p, p[n] - 1) + 1;  System.***out***.println("Line number" + " " + k + ": " + "From word no." + " " + p[n] + " " + "to" + " " + n);  **return** k;  }  }  Line number 1: From word no. 1 to 1  Line number 2: From word no. 2 to 3  Line number 3: From word no. 4 to 4 |

# **Ways to write n as sum of two or more positive integers: -** for a given number>0 find the number of different ways in which n can be written as a sum of at two or more positive integers.

|  |
| --- |
| Input : n = 5 Output : 6  Explanation : All possible six ways are :  4 + 1  3 + 2  3 + 1 + 1  2 + 2 + 1  2 + 1 + 1 + 1  1 + 1 + 1 + 1 + 1  Time complexity O(n2)  **public** **class** WriteNSumOfTwoOrMore {  **public** **static** **void** main(String[] args) {  **int** n = 7;  System.***out***.print(*countWays*(n));//14  }  **static** **int** countWays(**int** n) {  **int** table[] = **new** **int**[n + 1];  Arrays.*fill*(table, 0);  table[0] = 1;  **for** (**int** i = 1; i < n; i++)  **for** (**int** j = i; j <= n; j++)  table[j] += table[j - i];  **return** table[n];  }  } |

# **Anagram Using HashMap**

|  |
| --- |
| **public** **class** AnagramUsingHashMap {  **public** **static** **void** main(String[] args) {  *isAnagram*("Mother In Law", "Hitler Woman");  }  **static** **void** isAnagram(String s1, String s2) {  String copyOfs1 = s1.replaceAll("\\s", "").toLowerCase();  String copyOfs2 = s2.replaceAll("\\s", "").toLowerCase();  **boolean** status = **true**;  **if** (copyOfs1.length() != copyOfs2.length()) {  status = **false**;  } **else** {  HashMap<Character, Integer> map = **new** HashMap<Character, Integer>();  **for** (**int** i = 0; i < copyOfs1.length(); i++) {  **char** charAsKey = copyOfs1.charAt(i);  **int** charCountAsValue = 0;  **if** (map.containsKey(charAsKey)) {  charCountAsValue = map.get(charAsKey);  }  map.put(charAsKey, ++charCountAsValue);  charAsKey = copyOfs2.charAt(i);  charCountAsValue = 0;  **if** (map.containsKey(charAsKey)) {  charCountAsValue = map.get(charAsKey);  }  map.put(charAsKey, --charCountAsValue);  }  **for** (**int** value : map.values()) {  **if** (value != 0) {  status = **false**;  }  }  }  **if** (status) {  System.***out***.println(s1 + " and " + s2 + " are anagrams");  } **else** {  System.***out***.println(s1 + " and " + s2 + " are not anagrams");  }  }  } |

# **Anagram Using Iterative**

|  |
| --- |
| **public** **class** AnagramUsingIterative {  **public** **static** **void** main(String[] args) {  *isAnagram*("Mother In Law", "Hitler Woman");  }  **static** **void** isAnagram(String s1, String s2) {  String copyOfs1 = s1.replaceAll("\\s", "").toLowerCase();  String copyOfs2 = s2.replaceAll("\\s", "").toLowerCase();  **boolean** status = **true**;  **if** (copyOfs1.length() != copyOfs2.length()) {  status = **false**;  } **else** {  **char**[] s1ToArray = copyOfs1.toCharArray();  **for** (**char** c : s1ToArray) {  **int** index = copyOfs2.indexOf(c);  **if** (index != -1) {  copyOfs2 = copyOfs2.substring(0, index) + copyOfs2.substring(index + 1, copyOfs2.length());  } **else** {  status = **false**;  **break**;  }  }  }  **if** (status) {  System.***out***.println(s1 + " and " + s2 + " are anagrams");  } **else** {  System.***out***.println(s1 + " and " + s2 + " are not anagrams");  }  }  } |

# **Anagram Using Sort and Equals**

|  |
| --- |
| Two strings are called anagrams if they contain same set of characters but in different order.  For example, “Dormitory – Dirty Room”, “keep – peek”, “School Master – The Classroom” are some anagrams.\*/  **public** **class** AnagramUsingSortAndEquals {  **public** **static** **void** main(String[] args) {  *isAnagram*("Mother In Law", "Hitler Woman");  }  **static** **void** isAnagram(String s1, String s2) {  String copyOfs1 = s1.replaceAll("\\s", "");  String copyOfs2 = s2.replaceAll("\\s", "");  **boolean** status = **true**;  **if** (copyOfs1.length() != copyOfs2.length()) {  status = **false**;  } **else** {  **char**[] s1Array = copyOfs1.toLowerCase().toCharArray();  **char**[] s2Array = copyOfs2.toLowerCase().toCharArray();  Arrays.*sort*(s1Array);  Arrays.*sort*(s2Array);  status = Arrays.*equals*(s1Array, s2Array);  }  **if** (status) {  System.***out***.println(s1 + " and " + s2 + " are anagrams");  } **else** {  System.***out***.println(s1 + " and " + s2 + " are not anagrams");  }  }  } |

# **Anagram Using String Builder**

|  |
| --- |
| **public** **class** AnagramUsingStringBuilder {  **public** **static** **void** main(String[] args) {  *isAnagram*("Mother In Law", "Hitler Woman");  }  **static** **void** isAnagram(String s1, String s2) {  String copyOfs1 = s1.replaceAll("\\s", "").toLowerCase();  String copyOfs2 = s2.replaceAll("\\s", "").toLowerCase();  **boolean** status = **true**;  **if** (copyOfs1.length() != copyOfs2.length()) {  status = **false**;  } **else** {  **char**[] s1Array = copyOfs1.toCharArray();  StringBuilder sb = **new** StringBuilder(copyOfs2);  **for** (**char** c : s1Array) {  **int** index = sb.indexOf("" + c);  **if** (index != -1) {  sb = sb.deleteCharAt(index);  } **else** {  status = **false**;  **break**;  }  }  }  **if** (status) {  System.***out***.println(s1 + " and " + s2 + " are anagrams");  } **else** {  System.***out***.println(s1 + " and " + s2 + " are not anagrams");  }  }  } |

# **Armstrong Number**

|  |
| --- |
| **public** **class** ArmstrongNumber {  **public** **static** **void** main(String[] args) {  *checkArmstrongNumber*(153);  }  **static** **void** checkArmstrongNumber(**int** number) {  **int** copyOfNumber = number;  **int** noOfDigits = String.*valueOf*(number).length();  **int** sum = 0;  **while** (copyOfNumber != 0) {  **int** lastDigit = copyOfNumber % 10;  **int** lastDigitToThePowerOfNoOfDigits = 1;  **for** (**int** i = 0; i < noOfDigits; i++) {  lastDigitToThePowerOfNoOfDigits = lastDigitToThePowerOfNoOfDigits \* lastDigit;  }  sum = sum + lastDigitToThePowerOfNoOfDigits;  copyOfNumber = copyOfNumber / 10;  }  **if** (sum == number) {  System.***out***.println(number + " is an Armstrong number");  } **else** {  System.***out***.println(number + " is not an Armstrong number");  }  }  } |

# **Array Element Count**

|  |
| --- |
| **public** **class** ArrayElementCount {  **public** **static** **void** main(String[] args) {  *arrayElementCount*(**new** **int**[] { 4, 5, 4, 5, 4, 6 });  }  **static** **void** arrayElementCount(**int** inputArray[]) {  HashMap<Integer, Integer> elementCountMap = **new** HashMap<Integer, Integer>();  **for** (**int** i : inputArray) {  **if** (elementCountMap.containsKey(i)) {  elementCountMap.put(i, elementCountMap.get(i) + 1);  } **else** {  elementCountMap.put(i, 1);  }  }  System.***out***.println("Input Array : " + Arrays.*toString*(inputArray));  System.***out***.println("Element Count : " + elementCountMap);  }  } |

# **ArrayList to Array**

|  |
| --- |
| **public** **class** ArrayListToArray {  **public** **static** **void** main(String[] args) {  ArrayList<String> list = **new** ArrayList<String>();  list.add("JAVA");  list.add("JSP");  list.add("ANDROID");  list.add("STRUTS");  list.add("HADOOP");  list.add("JSF");  String[] array = **new** String[list.size()];  list.toArray(array);  **for** (String s : array) {  System.***out***.println(s);  }  }  } |

# **Array to ArrayList**

|  |
| --- |
| **public** **class** ArrayToArrayList {  **public** **static** **void** main(String[] args) {  // Using Arrays.asList() Method  String[] array = **new** String[] { "ANDROID", "JSP", "JAVA", "STRUTS", "HADOOP", "JSF" };  ArrayList<String> list = **new** ArrayList<String>(Arrays.*asList*(array));  System.***out***.println(list);  // Using Collections.addAll() Method  String[] array1 = **new** String[] { "ANDROID", "JSP", "JAVA", "STRUTS", "HADOOP", "JSF" };  ArrayList<String> list1 = **new** ArrayList<String>();  Collections.*addAll*(list1, array1);  System.***out***.println(list1);  // Using ArrayList.addAll() Method  String[] array2 = **new** String[] { "ANDROID", "JSP", "JAVA", "STRUTS", "HADOOP", "JSF" };  ArrayList<String> list2 = **new** ArrayList<String>();  list2.addAll(Arrays.*asList*(array2));  System.***out***.println(list2);  // Using Streams from Java 8  String[] array3 = **new** String[] { "ANDROID", "JSP", "JAVA", "STRUTS", "HADOOP", "JSF" };  List<Object> list3 = Arrays.*stream*(array3).collect(Collectors.*toList*());  System.***out***.println(list);  }  } |

# **Array Reverse**

|  |
| --- |
| **public** **class** ArrayReverse {  **public** **static** **void** main(String[] args) {  *reverseArray*(**new** **int**[] { 4, 5, 8, 9, 10 });  }  **static** **void** reverseArray(**int** inputArray[]) {  System.***out***.println("Array Before Reverse : " + Arrays.*toString*(inputArray));  **int** temp;  **for** (**int** i = 0; i < inputArray.length / 2; i++) {  temp = inputArray[i];  inputArray[i] = inputArray[inputArray.length - 1 - i];  inputArray[inputArray.length - 1 - i] = temp;  }  System.***out***.println("Array After Reverse : " + Arrays.*toString*(inputArray));  }  } |

# **Check Mobile Number**

|  |
| --- |
| **public** **class** CheckMobileNumber {  **public** **static** **void** main(String[] args) {  System.***out***.println("Enter your mobile number");  Scanner sc = **new** Scanner(System.***in***);  String input = sc.next();  **if** (Utility.*numberOrNot*(input) && (input.length() == 10)) {  System.***out***.println("Good!!! You have entered valid mobile number");  } **else** {  System.***out***.println("Sorry!!!! You have entered invalid mobile number. Try again...");  }  }  }  **class** Utility {  **static** **boolean** numberOrNot(String input) {  **try** {  Integer.*parseInt*(input);  } **catch** (NumberFormatException ex) {  **return** **false**;  }  **return** **true**;  }  } |

# **Common Elements Using Iterative**

|  |
| --- |
| **public** **class** CommonElementsUsingIterative {  **public** **static** **void** main(String[] args) {  String[] s1 = { "ONE", "TWO", "THREE", "FOUR", "FIVE", "FOUR" };  String[] s2 = { "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "FOUR" };  HashSet<String> set = **new** HashSet<String>();  **for** (**int** i = 0; i < s1.length; i++) {  **for** (**int** j = 0; j < s2.length; j++) {  **if** (s1[i].equals(s2[j])) {  set.add(s1[i]);  }  }  }  System.***out***.println(set); // OUTPUT : [THREE, FOUR, FIVE]  }  } |

# **Common Elements Using retainAll**

|  |
| --- |
| **public** **class** CommonElementsUsingretainAll {  **public** **static** **void** main(String[] args) {  Integer[] i1 = { 1, 2, 3, 4, 5, 4 };  Integer[] i2 = { 3, 4, 5, 6, 7, 4 };  HashSet<Integer> set1 = **new** HashSet<>(Arrays.*asList*(i1));  HashSet<Integer> set2 = **new** HashSet<>(Arrays.*asList*(i2));  set1.retainAll(set2);  System.***out***.println(set1); // Output : [3, 4, 5]  }  } |

# **Compare Text Files**

|  |
| --- |
| **public** **class** CompareTextFiles {  **public** **static** **void** main(String[] args) **throws** IOException {  BufferedReader reader1 = **new** BufferedReader(**new** FileReader("C:\\file1.txt"));  BufferedReader reader2 = **new** BufferedReader(**new** FileReader("C:\\file2.txt"));  String line1 = reader1.readLine();  String line2 = reader2.readLine();  **boolean** areEqual = **true**;  **int** lineNum = 1;  **while** (line1 != **null** || line2 != **null**) {  **if** (line1 == **null** || line2 == **null**) {  areEqual = **false**;  **break**;  } **else** **if** (!line1.equalsIgnoreCase(line2)) {  areEqual = **false**;  **break**;  }  line1 = reader1.readLine();  line2 = reader2.readLine();  lineNum++;  }  **if** (areEqual) {  System.***out***.println ("Two files have same content.");  } **else** {  System.***out***.println ("Two files have different content. They differ at line " + lineNum);  System.***out***.println("File1 has " + line1 + " and File2 has " + line2 + " at line " + lineNum);  }  reader1.close();  reader2.close();  }  } |

# **Convert Decimal to Other**

|  |
| --- |
| **public** **class** ConvertDecimalToOther {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Decimal Number : ");  **int** inputNumber = sc.nextInt();  **int** copyOfInputNumber = inputNumber;  String binary = "";  **int** rem = 0;  // Convert Decimal To Binary :  **while** (inputNumber > 0) {  rem = inputNumber % 2;  binary = rem + binary;  inputNumber = inputNumber / 2;  }  System.***out***.println("Binary Equivalent of " + copyOfInputNumber + " is " + binary);  // Convert Decimal To Octal :  String octal = "";  **while** (inputNumber > 0) {  rem = inputNumber % 8;  octal = rem + octal;  inputNumber = inputNumber / 8;  }  System.***out***.println("Octal Equivalent of " + copyOfInputNumber + " is " + octal);  // Decimal To HexaDecimal :  String hexa = "";  **char** hexaDecimals[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };  **while** (inputNumber > 0) {  rem = inputNumber % 16;  hexa = hexaDecimals[rem] + hexa;  inputNumber = inputNumber / 16;  }  System.***out***.println("HexaDecimal Equivalent of " + copyOfInputNumber + " is " + hexa);  }  } |

# **Write a java program to count the total number of occurrences of a given character in a string without using any loop?**

|  |
| --- |
| **public** **class** CountCharacterOccurence {  **public** **static** **void** main(String[] args) {  String s = "Java is java again java again";  **char** c = 'a';  **int** count = s.length() - s.replace("a", "").length();  System.***out***.println("Number of occurances of 'a' in " + s + " = " + count);  }  } |

# **Java program to count the number of words in a string?**

|  |
| --- |
| **public** **class** CountTheWords1 {  **public** **static** **void** main(String[] args) {  System.***out***.println("Enter the string");  Scanner sc = **new** Scanner(System.***in***);  String s = sc.nextLine();  String[] words = s.trim().split(" ");  System.***out***.println("Number of words in the string = " + words.length);  }  } |

# **Java program to count the number of words in a string other method**

|  |
| --- |
| **public** **class** CountTheWords2 {  **public** **static** **void** main(String[] args) {  System.***out***.println("Enter the string");  Scanner sc = **new** Scanner(System.***in***);  String s = sc.nextLine();  **int** count = 1;  **for** (**int** i = 0; i < s.length() - 1; i++) {  **if** ((s.charAt(i) == ' ') && (s.charAt(i + 1) != ' ')) {  count++;  }  }  System.***out***.println("Number of words in a string = " + count);  }  } |

# **Delegation**

|  |
| --- |
| **public** **class** Delegation {  **public** **static** **void** main(String[] args) {  Printer printer = **new** Printer();  printer.print();  }  }  **class** RealPrinter {  // the "delegate"  **void** print() {  System.***out***.println("The Delegate");  }  }  **class** Printer {  // the "delegator"  RealPrinter p = **new** RealPrinter();  // create the delegate  **void** print() {  p.print(); // delegation  }  } |

# **Detect Deadlock**

|  |
| --- |
| **public** **class** DetectDeadLock {  **public** **static** **void** main(String[] args) {  **final** Shared s1 = **new** Shared();  **final** Shared s2 = **new** Shared();  Thread t1 = **new** Thread() {  **public** **void** run() {  s1.methodOne(s2);  }  };  Thread t2 = **new** Thread() {  @Override  **public** **void** run() {  s2.methodTwo(s1);  }  };  t1.start();  t2.start();  **try** {  Thread.*sleep*(5000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  ThreadMXBean bean = ManagementFactory.*getThreadMXBean*();  **long** ids[] = bean.findMonitorDeadlockedThreads();  **if** (ids != **null**) {  ThreadInfo threadInfo[] = bean.getThreadInfo(ids);  **for** (ThreadInfo threadInfo1 : threadInfo) {  System.***out***.println(threadInfo1.getThreadId()); // Prints the ID of deadlocked thread  System.***out***.println(threadInfo1.getThreadName()); // Prints the name of deadlocked thread  System.***out***.println (threadInfo1.getLockName ()); // Prints the string representation of an object for which thread has entered into deadlock.  System.***out***.println(threadInfo1.getLockOwnerId()); // Prints the ID of thread which currently owns the object lock  System.***out***.println (threadInfo1.getLockOwnerName ()); // Prints name of the thread which currently owns the object lock.  }  } **else** {  System.***out***.println("No Deadlocked Threads");  }  }  }  **class** Shared {  **synchronized** **void** methodOne(Shared s) {  Thread t = Thread.*currentThread*();  System.***out***.println (t.getName () + "is executing methodOne...");  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println (t.getName () + "is calling methodTwo...");  s.methodTwo(**this**);  System.***out***.println (t.getName () + "is finished executing methodOne...");  }  **synchronized** **void** methodTwo(Shared s) {  Thread t = Thread.*currentThread*();  System.***out***.println (t.getName () + "is executing methodTwo...");  **try** {  Thread.*sleep*(2000);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  System.***out***.println (t.getName () + "is calling methodOne...");  s.methodOne(**this**);  System.***out***.println (t.getName () + "is finished executing methodTwo...");  }  } |

# **Find Difference between Two Dates**

|  |
| --- |
| **public** **class** DiffOfTwoDates1 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter start date in dd-MM-yyyy/HH:mm:ss format :");  String s1 = sc.next();  System.***out***.println("Enter end date in dd-MM-yyyy/HH:mm:ss format :");  String s2 = sc.next();  SimpleDateFormat formatter = **new** SimpleDateFormat("dd-MM-yyyy/HH:mm:ss");  **try** {  Date startDate = formatter.parse(s1);  Date endDate = formatter.parse(s2);  **long** diffInMilliSec = endDate.getTime() - startDate.getTime();  **long** seconds = (diffInMilliSec / 1000) % 60;  **long** minutes = (diffInMilliSec / (1000 \* 60)) % 60;  **long** hours = (diffInMilliSec / (1000 \* 60 \* 60)) % 24;  **long** days = (diffInMilliSec / (1000 \* 60 \* 60 \* 24)) % 365;  **long** years = (diffInMilliSec / (1000l \* 60 \* 60 \* 24 \* 365));  System.***out***.println("Difference is ---> ");  System.***out***.println(years + " years, " + days + " days, " + hours + " hours, " + minutes + " minutes, "  + seconds + " seconds");  } **catch** (ParseException e) {  e.printStackTrace();  }  sc.close();  }  } |

# **Find Difference between Two Dates in Java Using TimeUnit**

|  |
| --- |
| **public** **class** DiffOfTwoDates2 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter start date in dd-MM-yyyy/HH:mm:ss format :");  String s1 = sc.next();  System.***out***.println("Enter end date in dd-MM-yyyy/HH:mm:ss format :");  String s2 = sc.next();  SimpleDateFormat formatter = **new** SimpleDateFormat("dd-MM-yyyy/HH:mm:ss");  **try** {  Date startDate = formatter.parse(s1);  Date endDate = formatter.parse(s2);  **long** diffInMilliSec = endDate.getTime() - startDate.getTime();  **long** seconds = TimeUnit.***MILLISECONDS***.toSeconds(diffInMilliSec) % 60;  **long** minutes = TimeUnit.***MILLISECONDS***.toMinutes(diffInMilliSec) % 60;  **long** hours = TimeUnit.***MILLISECONDS***.toHours(diffInMilliSec) % 24;  **long** days = TimeUnit.***MILLISECONDS***.toDays(diffInMilliSec) % 365;  **long** years = TimeUnit.***MILLISECONDS***.toDays(diffInMilliSec) / 365l;  System.***out***.println("Difference is ---> ");  System.***out***.println(years + " years, " + days + " days, " + hours + " hours, " + minutes + " minutes, "  + seconds + " seconds");  } **catch** (ParseException e) {  e.printStackTrace();  }  sc.close();  }  } |

# **Duplicate Characters in String**

|  |
| --- |
| **public** **class** DuplicateCharactersInString {  **public** **static** **void** main(String[] args) {  *duplicateCharCount*("JavaJ2EE");  }  **static** **void** duplicateCharCount(String inputString) {  HashMap<Character, Integer> charCountMap = **new** HashMap<Character, Integer>();  **char**[] strArray = inputString.toCharArray();  **for** (**char** c : strArray) {  **if** (charCountMap.containsKey(c)) {  charCountMap.put(c, charCountMap.get(c) + 1);  } **else** {  charCountMap.put(c, 1);  }  }  Set<Character> charsInString = charCountMap.keySet();  System.***out***.println("Duplicate Characters In " + inputString);  **for** (Character ch : charsInString) {  **if** (charCountMap.get(ch) > 1) {  System.***out***.println(ch + " : " + charCountMap.get(ch));  }  }  }  } |

# **Duplicates in Array Using Brute Force**

|  |
| --- |
| **public** **class** DuplicatesInArrayUsingBruteForce {  **public** **static** **void** main(String[] args) {  String[] strArray = { "abc", "def", "mno", "xyz", "pqr", "xyz", "def" };  **for** (**int** i = 0; i < strArray.length - 1; i++) {  **for** (**int** j = i + 1; j < strArray.length; j++) {  **if** ((strArray[i].equals(strArray[j])) && (i != j)) {  System.***out***.println("Duplicate Element is : " + strArray[j]);  }  }  }  }  } |

# **Duplicates in Array Using HashSet**

|  |
| --- |
| **public** **class** DuplicatesInArrayUsingHashSet {  **public** **static** **void** main(String[] args) {  String[] strArray = { "abc", "def", "mno", "xyz", "pqr", "xyz", "def" };  HashSet<String> set = **new** HashSet<String>();  **for** (String arrayElement : strArray) {  **if** (!set.add(arrayElement)) {  System.***out***.println("Duplicate Element is : " + arrayElement);  }  }  }  } |

# **Find the duplicate words and their number of occurrences in a string?**

|  |
| --- |
| **public** **class** duplicateWordsInString {  **public** **static** **void** main(String[] args) {  *duplicateWords*("Bread butter and bread");  }  **static** **void** duplicateWords(String inputString) {  String[] words = inputString.split(" ");  HashMap<String, Integer> wordCount = **new** HashMap<String, Integer>();  **for** (String word : words) {  **if** (wordCount.containsKey(word.toLowerCase())) {  wordCount.put(word.toLowerCase(), wordCount.get(word.toLowerCase()) + 1);  } **else** {  wordCount.put(word.toLowerCase(), 1);  }  }  Set<String> wordsInString = wordCount.keySet();  **for** (String word : wordsInString) {  **if** (wordCount.get(word) > 1) {  System.***out***.println(word + " : " + wordCount.get(word));  }  }  }  } |

# **Java Program to Count Occurrences of Each Character in String**

|  |
| --- |
| **public** **class** EachCharCountInString {  **public** **static** **void** main(String[] args) {  *characterCount*("Java J2EE Java JSP J2EE");  }  **static** **void** characterCount(String inputString) {  HashMap<Character, Integer> charCountMap = **new** HashMap<Character, Integer>();  **char**[] strArray = inputString.toCharArray();  **for** (**char** c : strArray) {  **if** (charCountMap.containsKey(c)) {  charCountMap.put(c, charCountMap.get(c) + 1);  } **else** {  charCountMap.put(c, 1);  }  }  System.***out***.println(charCountMap);  }  } |

# **Equality of Two Arrays Using deep equals**

|  |
| --- |
| **public** **class** EqualityOfTwoArraysUsingdeepEquals {  **public** **static** **void** main(String[] args) {  String[][] s1 = { { "java", "swings", "j2ee" }, { "struts", "jsp", "hibernate" } };  String[][] s2 = { { "java", "swings", "j2ee" }, { "struts", "jsp", "hibernate" } };  System.***out***.println(Arrays.*deepEquals*(s1, s2)); // Output : true  System.***out***.println(Arrays.*equals*(s1, s2)); // Output : false  }  } |

# **Equality of Two Arrays Using equals**

|  |
| --- |
| **public** **class** EqualityOfTwoArraysUsingequals {  **public** **static** **void** main(String[] args) {  String[] s1 = { "java", "j2ee", "struts", "hibernate" };  String[] s2 = { "jsp", "spring", "jdbc", "hibernate" };  String[] s3 = { "java", "j2ee", "struts", "hibernate" };  System.***out***.println(Arrays.*equals*(s1, s2)); // Output : false  System.***out***.println(Arrays.*equals*(s1, s3)); // Output : true  }  } |

# **Equality of Two Arrays Using Iterative**

|  |
| --- |
| **public** **class** EqualityOfTwoArraysUsingIterative {  **public** **static** **void** main(String[] args) {  **int**[] arrayOne = { 2, 5, 1, 7, 4 };  **int**[] arrayTwo = { 2, 5, 1, 7, 4 };  **boolean** equalOrNot = **true**;  **if** (arrayOne.length == arrayTwo.length) {  **for** (**int** i = 0; i < arrayOne.length; i++) {  **if** (arrayOne[i] != arrayTwo[i]) {  equalOrNot = **false**;  }  }  } **else** {  equalOrNot = **false**;  }  **if** (equalOrNot) {  System.***out***.println("Two Arrays Are Equal");  } **else** {  System.***out***.println("Two Arrays Are Not equal");  }  }  } |

# **Equality of Two Arrays Using Sort and Equals**

|  |
| --- |
| **public** **class** EqualityOfTwoArraysUsingSortAndEquals {  **public** **static** **void** main(String[] args) {  String[] s1 = { "java", "swings", "j2ee", "struts", "jsp", "hibernate" };  String[] s2 = { "java", "struts", "j2ee", "hibernate", "swings", "jsp" };  Arrays.*sort*(s1);  Arrays.*sort*(s2);  System.***out***.println(Arrays.*equals*(s1, s2)); // Output : true  }  } |

# **Fibonacci Series**

|  |
| --- |
| **public** **class** FibonacciSeries {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter positive number :");  **int** inputNumber = sc.nextInt();  **int** firstTerm = 0;  **int** secondTerm = 1;  **int** thirdTerm = 0;  **while** (thirdTerm < inputNumber) {  thirdTerm = firstTerm + secondTerm;  firstTerm = secondTerm;  secondTerm = thirdTerm;  }  **if** (thirdTerm == inputNumber) {  System.***out***.println("Number belongs to Fibonacci series");  } **else** {  System.***out***.println("Number doesn't belongs to Fibonacci series");  }  }  } |

# **Java Program to Append Text to an Existing File**

|  |
| --- |
| **public** **class** FileWriterExample {  **public** **static** **void** main(String[] args) {  FileWriter fileWriter = **null**;  BufferedWriter bufferedWriter = **null**;  PrintWriter printWriter = **null**;  **try** {  fileWriter = **new** FileWriter("C:\\sample.txt", **true**);  bufferedWriter = **new** BufferedWriter(fileWriter);  printWriter = **new** PrintWriter(bufferedWriter);  printWriter.println();  printWriter.println("Venkatesh : 789546");  printWriter.println("Daniel : 874566");  printWriter.println("Shankar : 789546");  System.***out***.println("Done");  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  printWriter.close();  bufferedWriter.close();  fileWriter.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# **Find Duplicate Elements1 Using Brute Force Method**

|  |
| --- |
| **public** **class** FindDuplicateElements1 {  **public** **static** **void** main(String[] args) {  String[] strArray = { "Java", "JSP", "Servlets", "Java", "Struts", "JSP", "JDBC" };  **for** (**int** i = 0; i < strArray.length - 1; i++) {  **for** (**int** j = i + 1; j < strArray.length; j++) {  **if** ((strArray[i].equals(strArray[j])) && (i != j)) {  System.***out***.println("Duplicate Element is : " + strArray[j]);  }  }  }  }  } |

# **Find Duplicate Elements2 Using HashSet**

|  |
| --- |
| **public** **class** FindDuplicateElements2 {  **public** **static** **void** main(String[] args) {  String[] strArray = { "Java", "JSP", "Servlets", "Java", "Struts", "JSP", "JDBC" };  HashSet<String> set = **new** HashSet<String>();  **for** (String arrayElement : strArray) {  **if** (!set.add(arrayElement)) {  System.***out***.println("Duplicate Element is : " + arrayElement);  }  }  }  } |

# **First Repeated NonRepeated Char**

|  |
| --- |
| **public** **class** firstRepeatedNonRepeatedChar {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter the string :");  String input = sc.next();  *firstRepeatedNonRepeated*(input);  }  **static** **void** firstRepeatedNonRepeated(String inputString) {  HashMap<Character, Integer> charCountMap = **new** HashMap<Character, Integer>();  **char**[] strArray = inputString.toCharArray();  **for** (**char** c : strArray) {  **if** (charCountMap.containsKey(c)) {  charCountMap.put(c, charCountMap.get(c) + 1);  } **else** {  charCountMap.put(c, 1);  }  }  **for** (**char** c : strArray) {  **if** (charCountMap.get(c) == 1) {  System.***out***.println("First Non-Repeated Character In '" + inputString + "' is '" + c + "'");  **break**;  }  }  **for** (**char** c : strArray) {  **if** (charCountMap.get(c) > 1) {  System.***out***.println("First Repeated Character In '" + inputString + "' is '" + c + "'");  **break**;  }  }  }  } |

# **Floyd’s triangle is the right angled triangle consists of natural numbers**

|  |
| --- |
| **public** **class** FloydsTriangle {  **public** **static** **void** main(String[] args) {  System.***out***.println ("How many rows you want in Floyd's Triangle?");  Scanner sc = **new** Scanner(System.***in***);  **int** noOfRows = sc.nextInt();  **int** value = 1;  System.***out***.println("Floyd's Triangle : ");  **for** (**int** i = 1; i <= noOfRows; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(value + "\t");  value++;  }  System.***out***.println();  }  }  }  /\*How many rows you want in Floyd's Triangle?  5  Floyd's Triangle :  1  2 3  4 5 6  7 8 9 10  11 12 13 14 15 \*/ |

# **Pattern 1: dd/MM/yyyy (Ex: 10/09/2016)**

|  |
| --- |
| **public** **class** FormatDate1 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("dd/MM/yyyy");  System.***out***.println("Today in dd/MM/yyyy format : " + formatter.format(today));  }  } |

# **Pattern 2: yyyy-MM-dd (Ex: 2016-09-10)**

|  |
| --- |
| **public** **class** FormatDate2 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("yyyy-MM-dd");  System.***out***.println("Today in yyyy-MM-dd format : " + formatter.format(today));  }  } |

# **Pattern 3: dd MMMM yyyy (Ex: 10 September 2016)**

|  |
| --- |
| **public** **class** FormatDate3 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("dd MMMM yyyy");  System.***out***.println("Today in dd MMMM yyyy format : " + formatter.format(today));  }  } |

# **Pattern 4: E, dd MMM yyyy (Ex: Sat, 10 Sep 2016)**

|  |
| --- |
| **public** **class** FormatDate4 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("E, dd MMM yyyy");  System.***out***.println("Today in E, dd MMM yyyy format : " + formatter.format(today));  }  } |

# **Pattern 5: dd-MMM-yyyy HH:mm:ss (Ex: 10-Sep-2016 18:40:47)**

|  |
| --- |
| **public** **class** FormatDate5 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("dd-MMM-yyyy HH:mm:ss");  System.***out***.println("Today in dd-MMM-yyyy HH:mm:ss format : " + formatter.format(today));  }  } |

# **Pattern 6: EEEE, MMM dd yyyy, hh:mm:ss a (Ex: Saturday, Sep 10 2016, 06:45:51 PM)**

|  |
| --- |
| **public** **class** FormatDate6 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("EEEE, MMM dd yyyy, hh:mm:ss a");  System.***out***.println("Today in EEEE, MMM dd yyyy, hh:mm:ss a format : " + formatter.format(today));  }  } |

# **Pattern 7: dd-MMM-yyyy HH:mm:ss z (Ex: 10-Sep-2016 18:49:53 IST)**

|  |
| --- |
| **public** **class** FormatDate7 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("dd-MMM-yyyy HH:mm:ss z");  System.***out***.println("Today in dd-MMM-yyyy HH:mm:ss z format : " + formatter.format(today));  }  } |

# **Pattern 8: dd-MMM-yyyy HH:mm:ss Z (Ex: 10-Sep-2016 19:01:39 +0530)**

|  |
| --- |
| **public** **class** FormatDate8 {  **public** **static** **void** main(String[] args) {  Date today = **new** Date();  System.***out***.println("Today is : " + today);  SimpleDateFormat formatter = **new** SimpleDateFormat("dd-MMM-yyyy HH:mm:ss Z");  System.***out***.println("Today in dd-MMM-yyyy HH:mm:ss Z format : " + formatter.format(today));  }  } |

# **Find Frequency of All Digits in Number Using Collection**

|  |
| --- |
| **public** **class** FrequencyOfDigits1 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter the number :");  **int** inputNumber = sc.nextInt();  HashMap<Integer, Integer> digitCountMap = **new** HashMap<Integer, Integer>();  **while** (inputNumber != 0) {  **int** lastDigit = inputNumber % 10;  **if** (digitCountMap.containsKey(lastDigit)) {  digitCountMap.put(lastDigit, digitCountMap.get(lastDigit) + 1);  } **else** {  digitCountMap.put(lastDigit, 1);  }  inputNumber = inputNumber / 10;  }  System.***out***.println("===================");  System.***out***.println("Digits : Frequency");  System.***out***.println("===================");  Set<Integer> keys = digitCountMap.keySet();  **for** (Integer key : keys) {  System.***out***.println(" " + key + " : " + digitCountMap.get(key));  }  sc.close();  }  }  /\* Enter the number : 21757132  Digits : Frequency 1 : 2 2 : 2 3 : 1 5 : 1 7 : 2 |

# **Find Frequency of All Digits in Number without Using Collection**

|  |
| --- |
| **public** **class** FrequencyOfDigits2 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter the number :");  **int** inputNumber = sc.nextInt();  **int**[] digitCount = **new** **int**[10];  **while** (inputNumber != 0) {  **int** lastDigit = inputNumber % 10;  digitCount[lastDigit]++;  inputNumber = inputNumber / 10;  }  System.***out***.println("===================");  System.***out***.println("Digits : Frequency");  System.***out***.println("===================");  **for** (**int** i = 0; i < digitCount.length; i++) {  **if** (digitCount[i] != 0) {  System.***out***.println(" " + i + " : " + digitCount[i]);  }  }  sc.close();  }  }  \* Enter the number : 21757132  Digits : Frequency 1 : 2 2 : 2 3 : 1 5 : 1 7 : 2 |

# **Harshad number or Niven number is a number which is divisible by the sum of its digits.**

|  |
| --- |
| 21 is a Harshad number because it is divisible by the sum of its digits.  21 –> sum of digits –> 2+1 = 3 and 21 is divisible by 3 –> 21/3 = 7.  **public** **class** HarshadNumber {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter any positive number : ");  **int** inputNumber = sc.nextInt();  *checkForHarshad*(inputNumber);  sc.close();  }  **static** **void** checkForHarshad(**int** inputNumber) {  **int** copyOfInputNumber = inputNumber;  **int** sum = 0;  **int** lastDigit = 0;  **while** (inputNumber != 0) {  lastDigit = inputNumber % 10;  sum = sum + lastDigit;  inputNumber = inputNumber / 10;  }  **if** ((copyOfInputNumber % sum) == 0) {  System.***out***.println(copyOfInputNumber + " is a Harshad number");  } **else** {  System.***out***.println(copyOfInputNumber + " is not a Harshad number");  }  }  } |

# **Integer to String**

|  |
| --- |
| **public** **class** IntegerToString {  **public** **static** **void** main(String[] args) {  **int** i = 2015;  // Integer.toString() Method  String s = Integer.*toString*(i);  System.***out***.println(s);  // Using String.valueOf() method  **int** i1 = 2015;  String s1 = String.*valueOf*(i1);  System.***out***.println(s1);  }  } |

# **String to Integer**

|  |
| --- |
| **public** **class** StringToInteger {  **public** **static** **void** main(String[] args) {  String s = "2015";  // Using Integer.parseInt() method  **int** i = Integer.*parseInt*(s);  System.***out***.println(i);  // Using Integer.valueOf() method  String s1 = "2015";  **int** i1 = Integer.*valueOf*(s1);  System.***out***.println(i1);  }  } |

# **Intersection Iterative Using Iterative Method**

|  |
| --- |
| **public** **class** IntersectionIterative {  **public** **static** **void** main(String[] args) {  String[] inputArray1 = { "ONE", "TWO", "THREE", "FOUR", "FIVE", "FOUR" };  String[] inputArray2 = { "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "FOUR" };  HashSet<String> set = **new** HashSet<String>();  **for** (**int** i = 0; i < inputArray1.length; i++) {  **for** (**int** j = 0; j < inputArray2.length; j++) {  **if** (inputArray1[i].equals(inputArray2[j])) {  set.add(inputArray1[i]);  }  }  }  System.***out***.println("First Array : " + Arrays.*toString*(inputArray1));  System.***out***.println("Second Array : " + Arrays.*toString*(inputArray2));  System.***out***.println("Common Elements : " + set);  }  } |

# **Intersection of Multiple Arrays Using retainAll () Method**

|  |
| --- |
| **public** **class** IntersectionOfMultipleArrays {  **public** **static** **void** main(String[] args) {  Integer[] inputArray1 = { 2, 3, 4, 7, 1 };  Integer[] inputArray2 = { 4, 1, 3, 5 };  Integer[] inputArray3 = { 8, 4, 6, 2, 1 };  Integer[] inputArray4 = { 7, 9, 4, 1 };  *intersection*(inputArray1, inputArray2, inputArray3, inputArray4);  }  **static** **void** intersection(Integer[]... inputArrays) {  System.***out***.println("Input Arrays :");  System.***out***.println("======================");  **for** (Integer[] inputArray : inputArrays) {  System.***out***.println(Arrays.*toString*(inputArray));  }  HashSet<Integer> intersectionSet = **new** HashSet<>(Arrays.*asList*(inputArrays[0]));  **for** (**int** i = 1; i < inputArrays.length; i++) {  HashSet<Integer> set = **new** HashSet<>(Arrays.*asList*(inputArrays[i]));  intersectionSet.retainAll(set);  }  System.***out***.println("===========================");  System.***out***.println("Intersection Of All Input Arrays :");  System.***out***.println("===========================");  System.***out***.println(intersectionSet);  }  } |

# **Java Program to Find Largest Number Less than Given Number and Without a Given Digit**

|  |
| --- |
| **public** **class** LargestNumber {  **public** **static** **void** main(String[] args) {  System.***out***.println(*getLLessThanN*(123, 2));  }  **static** **int** getLLessThanN(**int** number, **int** digit) {  **char** c = Integer.*toString*(digit).charAt(0);  **for** (**int** i = number; i > 0; --i) {  **if** (Integer.*toString*(i).indexOf(c) == -1) {  **return** i;  }  }  **return** -1;  }  } |

# **Launching External Apps**

|  |
| --- |
| **public** **class** LaunchingExternalApps {  **public** **static** **void** main(String[] args) {  Runtime runtime = Runtime.*getRuntime*(); // getting Runtime object  String[] s = **new** String[] { "C:\\Program Files (x86)\\Google\\Chrome\\Application\\chrome.exe",  "http://javaconceptoftheday.com/" };  **try** {  runtime.exec("notepad.exe"); // opens new notepad instance  runtime.exec(s); // OR runtime.exec("notepad");  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

# **Leaders in Array1**

|  |
| --- |
| **public** **class** LeadersInArray1 {  **public** **static** **void** main(String[] args) {  *findTheLeaders*(**new** **int**[] { 12, 9, 7, 14, 8, 6, 3 });  }  **static** **void** findTheLeaders(**int** inputArray[]) {  **int** inputArrayLength = inputArray.length;  **int** max = inputArray[inputArrayLength - 1];  System.***out***.println("The leaders in " + Arrays.*toString*(inputArray) + " are : ");  System.***out***.println(inputArray[inputArrayLength - 1]);  **for** (**int** i = inputArray.length - 2; i >= 0; i--) {  **if** (inputArray[i] > max) {  System.***out***.println(inputArray[i]);  max = inputArray[i];  }  }  }  } |

# **Leaders in Array2**

|  |
| --- |
| **public** **class** LeadersInArray2 {  **public** **static** **void** main(String[] args) {  *findTheLeaders*(**new** **int**[] { 12, 9, 7, 14, 8, 6, 3 });  }  **static** **void** findTheLeaders(**int** inputArray[]) {  System.***out***.println("The leaders in " + Arrays.*toString*(inputArray) + " are :");  **for** (**int** i = 0; i < inputArray.length; i++) {  **boolean** flag = **true**;  **for** (**int** j = i + 1; j < inputArray.length; j++) {  **if** (inputArray[j] > inputArray[i]) {  flag = **false**;  **break**;  }  }  **if** (flag) {  System.***out***.println(inputArray[i]);  }  }  }  } |

# **List of All Files1**

|  |
| --- |
| **public** **class** ListOfAllFiles1 {  **public** **static** **void** main(String[] args) {  // 1. File.list() Method Example :  File folder = **new** File("F:/Path");  String[] files = folder.list();  **for** (String file : files) {  System.***out***.println(file);  }  // 2. File.listFiles() Method Example :  File folder1 = **new** File("F:/Path");  File[] files1 = folder1.listFiles();  **for** (File file1 : files1) {  System.***out***.println(file1.getName());  }  // 3. File.listFiles(FilenameFilter filter) Method Example :  // Implementing FilenameFilter to retrieve only txt files  File folder2 = **new** File("C:/Path");  FilenameFilter txtFileFilter = **new** FilenameFilter() {  @Override  **public** **boolean** accept(File dir, String name) {  **if** (name.endsWith(".txt")) {  **return** **true**;  } **else** {  **return** **false**;  }  }  };  File[] files2 = folder.listFiles(txtFileFilter);  **for** (File file2 : files2) {  System.***out***.println(file2.getName());  }  // 4. File.listFiles(FileFilter filter) Method Example : Implementing  // FileFilter to retrieve the files smaller than 10MB  File folder3 = **new** File("C:/Path");  FileFilter sizeFilter = **new** FileFilter() {  @Override  **public** **boolean** accept(File file3) {  **if** (file3.length() < 10 \* 1024 \* 1024) {  **return** **true**;  } **else** {  **return** **false**;  }  }  };  // 5. Passing sizeFilter to listFiles() method  File[] files4 = folder.listFiles(sizeFilter);  **for** (File file4 : files4) {  System.***out***.println(file4.getName());  }  }  } |

# **We are sorting an ArrayList of strings using first form of Collections.sort () method**

|  |
| --- |
| **public** **class** ListSorting1 {  **public** **static** **void** main(String[] args) {  ArrayList<String> list = **new** ArrayList<String>();  list.add("Virat");  list.add("rohit");  list.add("Shikar");  list.add("ashwin");  list.add("ravindra");  list.add("Bhargav");  System.***out***.println("ArrayList Before Sorting :");  System.***out***.println(list);  Collections.*sort*(list);  System.***out***.println("ArrayList After Sorting :");  System.***out***.println(list);  }  } |

# **We are sorting an ArrayList of strings using first form of Collections.sort () method with two arguments.**

|  |
| --- |
| We use second form of the Collections.sort () method which takes two arguments. One is the list to be sorted and another one is the Comparator. We pass String.CASE\_INSENSITIVE\_ORDER as Comparator here. This Comparator ignores the case of the string elements.  **public** **class** ListSorting2 {  **public** **static** **void** main(String[] args) {  ArrayList<String> list = **new** ArrayList<String>();  list.add("Virat");  list.add("rohit");  list.add("Shikar");  list.add("ashwin");  list.add("ravindra");  list.add("Bhargav");  System.***out***.println("ArrayList Before Sorting :");  System.***out***.println(list);  Collections.*sort*(list, String.***CASE\_INSENSITIVE\_ORDER***);  System.***out***.println("ArrayList After Sorting :");  System.***out***.println(list);  }  } |

# **Sort an ArrayList of Custom Objects: -** we sort an arraylist of student object. To do this, student class must implement Comparable interface and override comapreTo () method.

|  |
| --- |
| **public** **class** ListSorting3 {  **public** **static** **void** main(String[] args) {  ArrayList<Student> listOfStudents = **new** ArrayList<Student>();  listOfStudents.add(**new** Student(123, "Student1", 62));  listOfStudents.add(**new** Student(231, "Student2", 81));  listOfStudents.add(**new** Student(85, "Student3", 79));  listOfStudents.add(**new** Student(478, "Student4", 94));  listOfStudents.add(**new** Student(365, "Student5", 62));  System.***out***.println("listOfStudents Before Sorting :");  System.***out***.println(listOfStudents);  Collections.*sort*(listOfStudents);  System.***out***.println("listOfStudents After Sorting :");  System.***out***.println(listOfStudents);  }  }  **class** Student **implements** Comparable<Student> {  **int** id;  String name;  **int** percentage;  **public** Student(**int** id, String name, **int** percentage) {  **this**.id = id;  **this**.name = name;  **this**.percentage = percentage;  }  @Override  **public** **int** compareTo(Student s) {  **return** **this**.id - s.id; // Sorts the objects in ascending order  // return s.id - this.id; //Sorts the objects in descending order  }  @Override  **public** String toString() {  **return** "{ID : " + id + ", Name : " + name + ", Percentage : " + percentage + "}";  }  } |

# **Sort an ArrayList of Custom Objects Using Comparator: -** we sort an arraylist of student objects by using our aown Comaparator. This comparator sorts the student objects based on theirs percentage.

|  |
| --- |
| **public** **class** ListSorting4 {  **public** **static** **void** main(String[] args) {  ArrayList<Student1> listOfStudents = **new** ArrayList<Student1>();  listOfStudents.add(**new** Student1(123, "Student1", 62));  listOfStudents.add(**new** Student1(231, "Student2", 81));  listOfStudents.add(**new** Student1(85, "Student3", 79));  listOfStudents.add(**new** Student1(478, "Student4", 94));  listOfStudents.add(**new** Student1(365, "Student5", 62));  System.***out***.println("listOfStudents Before Sorting :");  System.***out***.println(listOfStudents);  Collections.*sort*(listOfStudents, **new** OrderByPercentage());  System.***out***.println("listOfStudents After Sorting :");  System.***out***.println(listOfStudents);  }  }  **class** Student1 **implements** Comparable<Student1> {  **int** id;  String name;  **int** percentage;  **public** Student1(**int** id, String name, **int** percentage) {  **this**.id = id;  **this**.name = name;  **this**.percentage = percentage;  }  @Override  **public** **int** compareTo(Student1 s) {  **return** **this**.id - s.id; // Sorts the objects in ascending order  // return s.id - this.id; //Sorts the objects in descending order  }  @Override  **public** String toString() {  **return** "{ID : " + id + ", Name : " + name + ", Percentage : " + percentage + "}";  }  }  // Defining our own Comparator  **class** OrderByPercentage **implements** Comparator<Student1> {  **public** **int** compare(Student1 s1, Student1 s2) {  **return** s1.percentage - s2.percentage;  }  } |

# **Sort an ArrayList in the Reverse Order: -** you can sort the list in the reverse order also by passing the Comparator returned by collections.reverseOrder () as comaparator to collections.sort () method.

|  |
| --- |
| **public** **class** ListSorting5 {  **public** **static** **void** main(String[] args) {  ArrayList<Integer> list = **new** ArrayList<Integer>();  list.add(1452);  list.add(6854);  list.add(8741);  list.add(6542);  list.add(3845);  System.***out***.println("ArrayList Before Sorting :");  System.***out***.println(list);  Collections.*sort*(list, Collections.*reverseOrder*());  System.***out***.println("ArrayList Sorted In The Reverse Order :");  System.***out***.println(list);  }  } |

# **Longest Substring without Repeating Char**

|  |
| --- |
| **public** **class** LongestSubstringWithoutRepeatingChar {  **public** **static** **void** main(String[] args) {  *longestSubstring*("javaconceptoftheday");  }  **static** **void** longestSubstring(String inputString) {  **char**[] charArray = inputString.toCharArray();  String longestSubstring = **null**;  **int** longestSubstringLength = 0;  LinkedHashMap<Character, Integer> charPosMap = **new** LinkedHashMap<Character, Integer>();  **for** (**int** i = 0; i < charArray.length; i++) {  **char** ch = charArray[i];  **if** (!charPosMap.containsKey(ch)) {  charPosMap.put(ch, i);  } **else** {  i = charPosMap.get(ch);  charPosMap.clear();  }  **if** (charPosMap.size() > longestSubstringLength) {  longestSubstringLength = charPosMap.size();  longestSubstring = charPosMap.keySet().toString();  }  }  System.***out***.println("Input String : " + inputString);  System.***out***.println("The longest substring : " + longestSubstring);  System.***out***.println("The longest Substring Length : " + longestSubstringLength);  }  } |

# **Matrix Addition**

|  |
| --- |
| /\*First Matrix =  5 4  8 6  1 3  Second Matrix =  9 2  8 6  3 0  Sum =  14 6  16 12  4 3\*/  **public** **class** MatrixAddition {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Number Of Rows");  **int** row = sc.nextInt();  System.***out***.println("Enter The Number Of Columns");  **int** cols = sc.nextInt();  **int**[][] matrix1 = **new** **int**[row][cols];  **int**[][] matrix2 = **new** **int**[row][cols];  **int**[][] sum = **new** **int**[row][cols];  System.***out***.println("Enter The Data For First Matrix :");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix1[i][j] = sc.nextInt();  }  }  System.***out***.println("Enter The Data For Second Matrix :");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix2[i][j] = sc.nextInt();  }  }  System.***out***.println("First Matrix = ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix1[i][j] + "\t");  }  System.***out***.println();  }  System.***out***.println("Second Matrix = ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix2[i][j] + "\t");  }  System.***out***.println();  }  System.***out***.println("Sum = ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  sum[i][j] = matrix1[i][j] + matrix2[i][j];  System.***out***.print(sum[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Matrix Creation**

|  |
| --- |
| **public** **class** MatrixCreation {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Number Of Rows");  **int** row = sc.nextInt();  System.***out***.println("Enter The Number Of Columns");  **int** cols = sc.nextInt();  **int**[][] matrix = **new** **int**[row][cols];  System.***out***.println("Enter Matrix Data");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix[i][j] = sc.nextInt();  }  }  System.***out***.println("Your Matrix is : ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Matrix Multiplication**

|  |
| --- |
| /\*First Matrix =  7 6 1  2 3 8  Second Matrix =  4 9  1 7  3 8  Product =  37 113  35 103\*/  **public** **class** MatrixMultiplication {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Number Of Rows In First Matrix");  **int** rowsInFirst = sc.nextInt();  System.***out***.println("Enter The Number Of Columns In First Matrix / Rows In Second Matrix");  **int** colsInFirstRowsInSecond = sc.nextInt();  System.***out***.println("Enter The Number Of Columns In Second Matrix");  **int** colsInSecond = sc.nextInt();  **int**[][] matrix1 = **new** **int**[rowsInFirst][colsInFirstRowsInSecond];  **int**[][] matrix2 = **new** **int**[colsInFirstRowsInSecond][colsInSecond];  **int**[][] product = **new** **int**[rowsInFirst][colsInSecond];  System.***out***.println("Enter The Data For First Matrix :");  **for** (**int** i = 0; i < rowsInFirst; i++) {  **for** (**int** j = 0; j < colsInFirstRowsInSecond; j++) {  matrix1[i][j] = sc.nextInt();  }  }  System.***out***.println("Enter The Data For Second Matrix :");  **for** (**int** i = 0; i < colsInFirstRowsInSecond; i++) {  **for** (**int** j = 0; j < colsInSecond; j++) {  matrix2[i][j] = sc.nextInt();  }  }  System.***out***.println("First Matrix = ");  **for** (**int** i = 0; i < rowsInFirst; i++) {  **for** (**int** j = 0; j < colsInFirstRowsInSecond; j++) {  System.***out***.print(matrix1[i][j] + "\t");  }  System.***out***.println();  }  System.***out***.println("Second Matrix = ");  **for** (**int** i = 0; i < colsInFirstRowsInSecond; i++) {  **for** (**int** j = 0; j < colsInSecond; j++) {  System.***out***.print(matrix2[i][j] + "\t");  }  System.***out***.println();  }  System.***out***.println("Product = ");  **for** (**int** i = 0; i < rowsInFirst; i++) {  **for** (**int** j = 0; j < colsInSecond; j++) {  **for** (**int** k = 0; k < colsInFirstRowsInSecond; k++) {  product[i][j] += matrix1[i][k] \* matrix2[k][j];  }  }  }  **for** (**int** i = 0; i < rowsInFirst; i++) {  **for** (**int** j = 0; j < colsInSecond; j++) {  System.***out***.print(product[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Matrix Subtraction**

|  |
| --- |
| /\*First Matrix =  7 6 9  4 8 2  Second Matrix =  6 9 1  2 5 7  Subtraction =  1 -3 8  2 3 -5\*/  **public** **class** MatrixSubtraction {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Number Of Rows");  **int** row = sc.nextInt();  System.***out***.println("Enter The Number Of Columns");  **int** cols = sc.nextInt();  **int**[][] matrix1 = **new** **int**[row][cols];  **int**[][] matrix2 = **new** **int**[row][cols];  **int**[][] sub = **new** **int**[row][cols];  System.***out***.println("Enter The Data For First Matrix :");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix1[i][j] = sc.nextInt();  }  }  System.***out***.println("Enter The Data For Second Matrix :");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix2[i][j] = sc.nextInt();  }  }  System.***out***.println("First Matrix = ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix1[i][j] + "\t");  }  System.***out***.println();  }  System.***out***.println("Second Matrix = ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix2[i][j] + "\t");  }  System.***out***.println();  }  System.***out***.println("Subtraction = ");  **for** (**int** i = 0; i < row; i++) {  **for** (**int** j = 0; j < cols; j++) {  sub[i][j] = matrix1[i][j] - matrix2[i][j];  System.***out***.print(sub[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Matrix Transpose**

|  |
| --- |
| /\*Transpose of a matrix is formed by turning all rows of a matrix into columns and columns into rows.  Your Matrix is :  7 2 9  1 6 3  Transpose of Matrix is :  7 1  2 6  9 3\*/  **public** **class** MatrixTranspose {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Number Of Rows");  **int** rows = sc.nextInt();  System.***out***.println("Enter The Number Of Columns");  **int** cols = sc.nextInt();  **int**[][] matrix = **new** **int**[rows][cols];  **int**[][] transpose = **new** **int**[cols][rows];  System.***out***.println("Enter The Data For Matrix :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix[i][j] = sc.nextInt();  }  }  System.***out***.println("Your Matrix is :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix[i][j] + " ");  }  System.***out***.println();  }  System.***out***.println("Transpose of Matrix is :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  transpose[j][i] = matrix[i][j];  }  }  **for** (**int** i = 0; i < cols; i++) {  **for** (**int** j = 0; j < rows; j++) {  System.***out***.print(transpose[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Missing Number in Array**

|  |
| --- |
| **public** **class** MissingNumberInArray {  **public** **static** **void** main(String[] args) {  **int** n = 8;  **int**[] a = { 1, 4, 5, 3, 7, 8, 6 };  **int** sumOfNnumbers = *sumOfNnumbers*(n);  **int** sumOfElements = *sumOfElements*(a);  **int** missingNumber = sumOfNnumbers - sumOfElements;  System.***out***.println("Missing Number is = " + missingNumber);  }  **static** **int** sumOfNnumbers(**int** n) {  **int** sum = (n \* (n + 1)) / 2;  **return** sum;  }  **static** **int** sumOfElements(**int**[] array) {  **int** sum = 0;  **for** (**int** i = 0; i < array.length; i++) {  sum = sum + array[i];  }  **return** sum;  }  } |

# **Most Frequent Element**

|  |
| --- |
| **public** **class** MostFrequentElement {  **public** **static** **void** main(String[] args) {  *getMostFrequentElement*(**new** **int**[] { 4, 5, 8, 7, 4, 7, 6, 7 });  }  **static** **void** getMostFrequentElement(**int** inputArray[]) {  HashMap<Integer, Integer> elementCountMap = **new** HashMap<Integer, Integer>();  **for** (**int** i : inputArray) {  **if** (elementCountMap.containsKey(i)) {  elementCountMap.put(i, elementCountMap.get(i) + 1);  } **else** {  elementCountMap.put(i, 1);  }  }  **int** element = 0;  **int** frequency = 1;  Set<Entry<Integer, Integer>> entrySet = elementCountMap.entrySet();  **for** (Entry<Integer, Integer> entry : entrySet) {  **if** (entry.getValue() > frequency) {  element = entry.getKey();  frequency = entry.getValue();  }  }  **if** (frequency > 1) {  System.***out***.println("Input Array : " + Arrays.*toString*(inputArray));  System.***out***.println("The most frequent element : " + element);  System.***out***.println("Its frequency : " + frequency);  System.***out***.println("========================");  } **else** {  System.***out***.println("Input Array : " + Arrays.*toString*(inputArray));  System.***out***.println ("No frequent element. All elements are unique.");  System.***out***.println("=========================");  }  }  } |

# **Number Is Binary or Not**

|  |
| --- |
| **public** **class** NumberIsBinaryOrNot {  **public** **static** **void** main(String[] args) {  *isBinaryOrNot*(128956);  }  **static** **void** isBinaryOrNot(**int** number) {  **boolean** isBinary = **true**;  **int** copyOfNumber = number;  **while** (copyOfNumber != 0) {  **int** temp = copyOfNumber % 10;  **if** (temp > 1) {  isBinary = **false**;  **break**;  } **else** {  copyOfNumber = copyOfNumber / 10;  }  }  **if** (isBinary) {  System.***out***.println(number + " is a binary number");  } **else** {  System.***out***.println(number + " is not a binary number");  }  }  } |

# **Pairs of Elements in Array Using Alternative**

|  |
| --- |
| **public** **class** PairsOfElementsInArrayUsingAlternative {  **public** **static** **void** main(String[] args) {  *findThePairs*(**new** **int**[] { 4, 6, 5, -10, 8, 5, 20 }, 10);  }  **static** **void** findThePairs(**int** inputArray[], **int** inputNumber) {  Arrays.*sort*(inputArray);  System.***out***.println("Pairs of elements whose sum is " + inputNumber + " are : ");  **int** i = 0;  **int** j = inputArray.length - 1;  **while** (i < j) {  **if** (inputArray[i] + inputArray[j] == inputNumber) {  System.***out***.println(inputArray[i] + " + " + inputArray[j] + " = " + inputNumber);  i++;  j--;  } **else** **if** (inputArray[i] + inputArray[j] < inputNumber) {  i++;  } **else** **if** (inputArray[i] + inputArray[j] > inputNumber) {  j--;  }  }  }  } |

# **Find All Pairs Of Elements in an Array Whose Sum Is Equal to a Given Number**

|  |
| --- |
| **public** **class** PairsOfElementsInArrayUsingBruteForce {  **public** **static** **void** main(String[] args) {  *findThePairs*(**new** **int**[] { 4, 6, 5, -10, 8, 5, 20 }, 10);  }  **static** **void** findThePairs(**int** inputArray[], **int** inputNumber) {  System.***out***.println("Pairs of elements whose sum is " + inputNumber + " are : ");  **for** (**int** i = 0; i < inputArray.length; i++) {  **for** (**int** j = i + 1; j < inputArray.length; j++) {  **if** (inputArray[i] + inputArray[j] == inputNumber) {  System.***out***.println(inputArray[i] + " + " + inputArray[j] + " = " + inputNumber);  }  }  }  }  } |

# **Pattern1**

|  |
| --- |
| /\*How Many Rows You Want In Your Pyramid? 5  Here Is Your Pyramid  1  2 2  3 3 3  4 4 4 4  5 5 5 5 5 \*/  **public** **class** Pattern1 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Pyramid?");  **int** noOfRows = sc.nextInt();  **int** rowCount = 1;  System.***out***.println("Here Is Your Pyramid");  **for** (**int** i = noOfRows; i > 0; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= rowCount; j++) {  System.***out***.print(rowCount + " ");  }  System.***out***.println();  rowCount++;  }  }  } |

# **Pattern2**

|  |
| --- |
| /\*How Many Rows You Want In Your Pyramid? 5  Here Is Your Pyramid  1  1 2  1 2 3  1 2 3 4  1 2 3 4 5\*/  **public** **class** Pattern2 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Pyramid?");  **int** noOfRows = sc.nextInt();  **int** rowCount = 1;  System.***out***.println("Here Is Your Pyramid");  **for** (**int** i = noOfRows; i > 0; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= rowCount; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  rowCount++;  }  }  } |

# **Pattern3**

|  |
| --- |
| /\*How Many Rows You Want In Your Pyramid? 5  Here Is Your Pyramid  \*  \* \*  \* \* \*  \* \* \* \*  \* \* \* \* \* \*/  **public** **class** Pattern3 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Pyramid?");  **int** noOfRows = sc.nextInt();  **int** rowCount = 1;  System.***out***.println("Here Is Your Pyramid");  **for** (**int** i = noOfRows; i > 0; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= rowCount; j++) {  System.***out***.print("\* ");  }  System.***out***.println();  rowCount++;  }  }  } |

# **Pattern4**

|  |
| --- |
| /\*How Many Rows You Want In Your Pyramid? 5  Here Is Your Pyramid  1  1 2 1  1 2 3 2 1  1 2 3 4 3 2 1  1 2 3 4 5 4 3 2 1\*/  **public** **class** Pattern4 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Pyramid?");  **int** noOfRows = sc.nextInt();  **int** rowCount = 1;  System.***out***.println("Here Is Your Pyramid");  **for** (**int** i = noOfRows; i > 0; i--) {  **for** (**int** j = 1; j <= i \* 2; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= rowCount; j++) {  System.***out***.print(j + " ");  }  **for** (**int** j = rowCount - 1; j >= 1; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  rowCount++;  }  }  } |

# **Pattern5**

|  |
| --- |
| /\*How Many Rows You Want In Your Pyramid? 5  Here Is Your Pyramid  1 2 3 4 5 4 3 2 1  1 2 3 4 3 2 1  1 2 3 2 1  1 2 1  1 \*/  **public** **class** Pattern5 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Pyramid?");  **int** noOfRows = sc.nextInt();  **int** rowCount = noOfRows;  System.***out***.println("Here Is Your Pyramid");  **for** (**int** i = 0; i < noOfRows; i++) {  **for** (**int** j = 1; j <= i \* 2; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= rowCount; j++) {  System.***out***.print(j + " ");  }  **for** (**int** j = rowCount - 1; j >= 1; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  rowCount--;  }  }  } |

# **Pattern6**

|  |
| --- |
| /\*How Many Rows You Want In Your Pyramid? 5  Here Is Your Pyramid  5  4 5 4  3 4 5 4 3  2 3 4 5 4 3 2  1 2 3 4 5 4 3 2 1 \*/  **public** **class** Pattern6 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Pyramid?");  **int** noOfRows = sc.nextInt();  **int** rowCount = 1;  System.***out***.println("Here Is Your Pyramid");  **for** (**int** i = noOfRows; i >= 1; i--) {  **for** (**int** j = 1; j <= i \* 2; j++) {  System.***out***.print(" ");  }  **for** (**int** j = i; j <= noOfRows; j++) {  System.***out***.print(j + " ");  }  **for** (**int** j = noOfRows - 1; j >= i; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  rowCount++;  }  }  } |

# **Pattern7**

|  |
| --- |
| /\*Here is your pattern….!!!  1  1 2  1 2 3  1 2 3 4  1 2 3 4 5  1 2 3 4 5 6  1 2 3 4 5 6 7\*/  **public** **class** Pattern7 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern8**

|  |
| --- |
| /\*Here is your pattern….!!!  1  2 2  3 3 3  4 4 4 4  5 5 5 5 5  6 6 6 6 6 6  7 7 7 7 7 7 7\*/  **public** **class** Pattern8 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(i + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern9**

|  |
| --- |
| /\*Here is your pattern….!!!  1  1 2  1 2 3  1 2 3 4  1 2 3 4 5  1 2 3 4 5 6  1 2 3 4 5 6 7  1 2 3 4 5 6  1 2 3 4 5  1 2 3 4  1 2 3  1 2  1\*/  **public** **class** Pattern9 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  **for** (**int** i = rows - 1; i >= 1; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern10**

|  |
| --- |
| /\*Here is your pattern….!!!  1 2 3 4 5 6 7  1 2 3 4 5 6  1 2 3 4 5  1 2 3 4  1 2 3  1 2  1\*/  **public** **class** Pattern10 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = rows; i >= 1; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern11**

|  |
| --- |
| /\*Here is your pattern….!!!  7 6 5 4 3 2 1  7 6 5 4 3 2  7 6 5 4 3  7 6 5 4  7 6 5  7 6  7\*/  **public** **class** Pattern11 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = rows; j >= i; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern12**

|  |
| --- |
| /\*Here is your pattern….!!!  7  7 6  7 6 5  7 6 5 4  7 6 5 4 3  7 6 5 4 3 2  7 6 5 4 3 2 1\*/  **public** **class** Pattern12 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = rows; i >= 1; i--) {  **for** (**int** j = rows; j >= i; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern13**

|  |
| --- |
| /\*Here is your pattern….!!!  7 6 5 4 3 2 1  6 5 4 3 2 1  5 4 3 2 1  4 3 2 1  3 2 1  2 1  1\*/  **public** **class** Pattern13 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = rows; i >= 1; i--) {  **for** (**int** j = i; j >= 1; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern14**

|  |
| --- |
| /\*Here is your pattern….!!!  1 2 3 4 5 6 7  1 2 3 4 5 6  1 2 3 4 5  1 2 3 4  1 2 3  1 2  1  1 2  1 2 3  1 2 3 4  1 2 3 4 5  1 2 3 4 5 6  1 2 3 4 5 6 7\*/  **public** **class** Pattern14 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = rows; i >= 1; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  **for** (**int** i = 2; i <= rows; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern15**

|  |
| --- |
| /\*Here is your pattern….!!!  1  1 2 1  1 2 3 2 1  1 2 3 4 3 2 1  1 2 3 4 5 4 3 2 1  1 2 3 4 5 6 5 4 3 2 1  1 2 3 4 5 6 7 6 5 4 3 2 1\*/  **public** **class** Pattern15 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(j + " ");  }  **for** (**int** j = i - 1; j >= 1; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern16**

|  |
| --- |
| /\*Here is your pattern….!!!  1  2 1  3 2 1  4 3 2 1  5 4 3 2 1  6 5 4 3 2 1  7 6 5 4 3 2 1\*/  **public** **class** Pattern16 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = i; j >= 1; j--) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern17**

|  |
| --- |
| /\*Here is your pattern....!!!  1234567  234567  34567  4567  567  67  7  67  567  4567  34567  234567  1234567\*/  **public** **class** Pattern17 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j < i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = i; j <= rows; j++) {  System.***out***.print(j);  }  System.***out***.println();  }  **for** (**int** i = rows - 1; i >= 1; i--) {  **for** (**int** j = 1; j < i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = i; j <= rows; j++) {  System.***out***.print(j);  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern18**

|  |
| --- |
| /\*Here is your pattern....!!!  1 2 3 4 5 6 7  2 3 4 5 6 7  3 4 5 6 7  4 5 6 7  5 6 7  6 7  7  6 7  5 6 7  4 5 6 7  3 4 5 6 7  2 3 4 5 6 7  1 2 3 4 5 6 7\*/  **public** **class** Pattern18 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j < i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = i; j <= rows; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  **for** (**int** i = rows - 1; i >= 1; i--) {  **for** (**int** j = 1; j < i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = i; j <= rows; j++) {  System.***out***.print(j + " ");  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern19**

|  |
| --- |
| /\*Here is your pattern….!!!  1  10  101  1010  10101  101010  1010101\*/  **public** **class** Pattern19 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j <= i; j++) {  **if** (j % 2 == 0) {  System.***out***.print(0);  } **else** {  System.***out***.print(1);  }  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern20**

|  |
| --- |
| /\*Here is your pattern….!!!  1010101  0101010  1010101  0101010  1010101  0101010  1010101\*/  **public** **class** Pattern20 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **int** num;  **if** (i % 2 == 0) {  num = 0;  **for** (**int** j = 1; j <= rows; j++) {  System.***out***.print(num);  num = (num == 0) ? 1 : 0;  }  } **else** {  num = 1;  **for** (**int** j = 1; j <= rows; j++) {  System.***out***.print(num);  num = (num == 0) ? 1 : 0;  }  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern21**

|  |
| --- |
| /\*Here is your pattern….!!!  1111111  1111122  1111333  1114444  1155555  1666666  7777777\*/  **public** **class** Pattern21 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **for** (**int** j = 1; j <= rows - i; j++) {  System.***out***.print(1);  }  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(i);  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern22**

|  |
| --- |
| /\*Here is your pattern….!!!  0000000  0100000  0020000  0003000  0000400  0000050  0000006\*/  **public** **class** Pattern22 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < rows; j++) {  **if** (i == j) {  System.***out***.print(i);  } **else** {  System.***out***.print(0);  }  }  System.***out***.println();  }  sc.close();  }  } |

# **Pattern23**

|  |
| --- |
| /\*Here is your pattern….!!!  1  2 6  3 7 10  4 8 11 13  5 9 12 14 15\*/  **public** **class** Pattern23 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How many rows you want in this pattern?");  **int** rows = sc.nextInt();  System.***out***.println ("Here is your pattern....!!!");  **for** (**int** i = 1; i <= rows; i++) {  **int** num = i;  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(num + " ");  num = num + rows - j;  }  System.***out***.println();  }  sc.close();  }  } |

# **Find the Percentage of Uppercase Letters, Lowercase Letters, Digits and Other Special Characters in a String**

|  |
| --- |
| **public** **class** PercentageInString {  **public** **static** **void** main(String[] args) {  *characterPercentage* ("Tiger Runs @ The Speed Of 100 km/hour.");  }  **static** **void** characterPercentage(String inputString) {  **int** totalChars = inputString.length();  **int** upperCaseLetters = 0;  **int** lowerCaseLetters = 0;  **int** digits = 0;  **int** others = 0;  **for** (**int** i = 0; i < inputString.length(); i++) {  **char** ch = inputString.charAt(i);  **if** (Character.*isUpperCase*(ch)) {  upperCaseLetters++;  } **else** **if** (Character.*isLowerCase*(ch)) {  lowerCaseLetters++;  } **else** **if** (Character.*isDigit*(ch)) {  digits++;  } **else** {  others++;  }  }  **double** upperCaseLetterPercentage = (upperCaseLetters \* 100.0) / totalChars;  **double** lowerCaseLetterPercentage = (lowerCaseLetters \* 100.0) / totalChars;  **double** digitsPercentage = (digits \* 100.0) / totalChars;  **double** otherCharPercentage = (others \* 100.0) / totalChars;  DecimalFormat formatter = **new** DecimalFormat ("##.##");  System.***out***.println("In '" + inputString + "' : ");  System.***out***.println("Uppercase letters are " + formatter.format(upperCaseLetterPercentage) + "% ");  System.***out***.println("Lowercase letters are " + formatter.format(lowerCaseLetterPercentage) + "%");  System.***out***.println("Digits Are " + formatter.format(digitsPercentage) + "%");  System.***out***.println("Other Characters Are " + formatter.format(otherCharPercentage) + "%");  System.***out***.println("-----------------------------");  }  } |

# **Permutations of String**

|  |
| --- |
| **public** **class** PermutationsOfString {  **public** **static** **void** main(String[] args) {  *StringPermutation*("JSP");  }  **static** **public** **void** StringPermutation(String input) {  *StringPermutation*("", input);  }  **private** **static** **void** StringPermutation(String permutation, String input) {  **if** (input.length() == 0) {  System.***out***.println(permutation);  } **else** {  **for** (**int** i = 0; i < input.length(); i++) {  *StringPermutation*(permutation + input.charAt(i),  input.substring(0, i) + input.substring(i + 1, input.length()));  }  }  }  } |

# **Print Diamond1**

|  |
| --- |
| /\*Here Is Your Diamond :  \*  \* \*  \* \* \*  \* \* \* \*  \* \* \*  \* \*  \*  \*/  **public** **class** PrintDiamond1 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Diamond?");  **int** noOfRows = sc.nextInt();  **int** midRow = (noOfRows) / 2;  **int** row = 1;  System.***out***.println("Here Is Your Diamond : ");  **for** (**int** i = midRow; i > 0; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= row; j++) {  System.***out***.print("\* ");  }  System.***out***.println();  row++;  }  **for** (**int** i = 0; i <= midRow; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = row; j > 0; j--) {  System.***out***.print("\* ");  }  System.***out***.println();  row--;  }  }  } |

# **Print Diamond2**

|  |
| --- |
| /\*Here Is Your Diamond :  1  2 2  3 3 3  4 4 4 4  3 3 3  2 2  1  \*/  **public** **class** PrintDiamond2 {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("How Many Rows You Want In Your Diamond?");  **int** noOfRows = sc.nextInt();  **int** midRow = noOfRows / 2;  **int** row = 1;  System.***out***.println("Here Is Your Diamond : ");  **for** (**int** i = midRow; i > 0; i--) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = 1; j <= row; j++) {  System.***out***.print(row + " ");  }  System.***out***.println();  row++;  }  **for** (**int** i = 0; i <= midRow; i++) {  **for** (**int** j = 1; j <= i; j++) {  System.***out***.print(" ");  }  **for** (**int** j = row; j > 0; j--) {  System.***out***.print(row + " ");  }  System.***out***.println();  row--;  }  }  } |

# **Read and Write Image from Local Disc**

|  |
| --- |
| **public** **class** ReadAndWriteImageFromLocalDisc {  **public** **static** **void** main(String[] args) {  File file = **new** File("I:/input.jpg");  BufferedImage image = **null**;  **try** {  image = ImageIO.*read*(file);  ImageIO.*write*(image, "jpg", **new** File("I:/output.jpg"));  ImageIO.*write*(image, "png", **new** File("I:/output.png"));  ImageIO.*write*(image, "gif", **new** File("I:/output.gif"));  ImageIO.*write*(image, "bmp", **new** File("I:/output.bmp"));  } **catch** (IOException e) {  e.printStackTrace();  }  System.***out***.println("done");  }  } |

# **Read and Write Image from URL**

|  |
| --- |
| **public** **class** ReadAndWriteImageFromURL {  **public** **static** **void** main(String[] args) {  URL url = **null**;  BufferedImage image = **null**;  **try** {  url = **new** URL("http://javaconceptoftheday.com/wp-content/NumberPatternPrograms.png");  } **catch** (MalformedURLException e1) {  e1.printStackTrace();  }  **try** {  image = ImageIO.*read*(url);  ImageIO.*write*(image, "jpg", **new** File("I:/output.jpg"));  ImageIO.*write*(image, "png", **new** File("I:/output.png"));  ImageIO.*write*(image, "gif", **new** File("I:/output.gif"));  ImageIO.*write*(image, "bmp", **new** File("I:/output.bmp"));  } **catch** (IOException e) {  e.printStackTrace();  }  System.***out***.println("done");  }  } |

# **Remove All Vowels**

|  |
| --- |
| **public** **class** RemoveAllVowels {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println ("Enter the string...");  String inputString = sc.nextLine();  String newInputString = inputString.replaceAll("[AEIOUaeiou]", "");  System.***out***.println ("The string without vowels...");  System.***out***.println(newInputString);  sc.close();  }  } |

# **Remove Duplicates Elements**

|  |
| --- |
| **public** **class** RemoveDuplicatesElements {  **public** **static** **void** main(String[] args) {  *removeDuplicates*(**new** **int**[] { 4, 3, 2, 4, 9, 2 });  }  **static** **void** removeDuplicates(**int**[] arrayWithDuplicates) {  System.***out***.println("Array With Duplicates : ");  **for** (**int** i = 0; i < arrayWithDuplicates.length; i++) {  System.***out***.print(arrayWithDuplicates[i] + "\t");  }  **int** noOfUniqueElements = arrayWithDuplicates.length;  **for** (**int** i = 0; i < noOfUniqueElements; i++) {  **for** (**int** j = i + 1; j < noOfUniqueElements; j++) {  **if** (arrayWithDuplicates[i] == arrayWithDuplicates[j]) {  arrayWithDuplicates[j] = arrayWithDuplicates[noOfUniqueElements - 1];  noOfUniqueElements--;  j--;  }  }  }  **int**[] arrayWithoutDuplicates = Arrays.*copyOf*(arrayWithDuplicates, noOfUniqueElements);  System.***out***.println();  System.***out***.println("Array Without Duplicates : ");  **for** (**int** i = 0; i < arrayWithoutDuplicates.length; i++) {  System.***out***.print(arrayWithoutDuplicates[i] + "\t");  }  System.***out***.println();  System.***out***.println("==============================");  }  } |

# **Remove WhiteSpaces**

|  |
| --- |
| **public** **class** RemoveWhiteSpaces {  **public** **static** **void** main(String[] args) {  String str = " Core Java jsp servlets jdbc struts hibernate spring ";  // 1. Using replaceAll() Method  String strWithoutSpace = str.replaceAll("\\s", "");  System.***out***.println(strWithoutSpace); // Output : CoreJavajspservletsjdbcstrutshibernatespring  // 2. Without Using replaceAll() Method  **char**[] strArray = str.toCharArray();  StringBuffer sb = **new** StringBuffer();  **for** (**int** i = 0; i < strArray.length; i++) {  **if** ((strArray[i] != ' ') && (strArray[i] != '\t')) {  sb.append(strArray[i]);  }  }  System.***out***.println(sb); // Output : CoreJavajspservletsjdbcstrutshibernatespring  }  } |

# **To Find the Most Repeated Word in Text File**

|  |
| --- |
| **public** **class** RepeatedWordInFile1 {  **public** **static** **void** main(String[] args) {  HashMap<String, Integer> wordCountMap = **new** HashMap<String, Integer>();  BufferedReader reader = **null**;  **try** {  reader = **new** BufferedReader(**new** FileReader("C:\\sample.txt"));  String currentLine = reader.readLine();  **while** (currentLine != **null**) {  String[] words = currentLine.toLowerCase().split(" ");  **for** (String word : words) {  **if** (wordCountMap.containsKey(word)) {  wordCountMap.put(word, wordCountMap.get(word) + 1);  } **else** {  wordCountMap.put(word, 1);  }  }  currentLine = reader.readLine();  }  String mostRepeatedWord = **null**;  **int** count = 0;  Set<Entry<String, Integer>> entrySet = wordCountMap.entrySet();  **for** (Entry<String, Integer> entry : entrySet) {  **if** (entry.getValue() > count) {  mostRepeatedWord = entry.getKey();  count = entry.getValue();  }  }  System.***out***.println("The most repeated word in input file is : " + mostRepeatedWord);  System.***out***.println("Number Of Occurrences : " + count);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  reader.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# **Find All Repeated Words in Text File and Their Occurrences in Java**

|  |
| --- |
| **public** **class** RepeatedWordInFile2 {  **public** **static** **void** main(String[] args) {  HashMap<String, Integer> wordCountMap = **new** HashMap<String, Integer>();  BufferedReader reader = **null**;  **try** {  reader = **new** BufferedReader(**new** FileReader("C:\\sample.txt"));  String currentLine = reader.readLine();  **while** (currentLine != **null**) {  String[] words = currentLine.toLowerCase().split(" ");  **for** (String word : words) {  **if** (wordCountMap.containsKey(word)) {  wordCountMap.put(word, wordCountMap.get(word) + 1);  } **else** {  wordCountMap.put(word, 1);  }  }  currentLine = reader.readLine();  }  Set<Entry<String, Integer>> entrySet = wordCountMap.entrySet();  List<Entry<String, Integer>> list = **new** ArrayList<Entry<String, Integer>>(entrySet);  Collections.*sort*(list, **new** Comparator<Entry<String, Integer>>() {  @Override  **public** **int** compare(Entry<String, Integer> e1, Entry<String, Integer> e2) {  **return** (e2.getValue().compareTo(e1.getValue()));  }  });  System.***out***.println("Repeated Words In Input File Are :");  **for** (Entry<String, Integer> entry : list) {  **if** (entry.getValue() > 1) {  System.***out***.println(entry.getKey() + " : " + entry.getValue());  }  }  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  reader.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# **Java Program to Reverse and Add a Number Until You Get a Palindrome**

|  |
| --- |
| **public** **class** ReverseAddPalindrome {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter Number : ");  **int** inputNumber = sc.nextInt();  *reverseAndAdd*(inputNumber);  }  **static** **void** reverseAndAdd(**int** number) {  **if** (*checkPalindrome*(number)) {  System.***out***.println("Given Number is already a palindrome");  } **else** {  **while** (!*checkPalindrome*(number)) {  **int** reverse = *reverseNumber*(number);  **int** sum = number + reverse;  System.***out***.println(number + " + " + reverse + " = " + sum);  number = sum;  }  }  }  **static** **boolean** checkPalindrome(**int** number) {  **int** reverse = *reverseNumber*(number);  **if** (reverse == number) {  **return** **true**;  } **else** {  **return** **false**;  }  }  **static** **int** reverseNumber(**int** number) {  **int** reverse = 0;  **int** rem = 0;  **while** (number != 0) {  rem = number % 10;  reverse = (reverse \* 10) + rem;  number = number / 10;  }  **return** reverse;  }  } |

# **Reverse Array without Additional Array**

|  |
| --- |
| **public** **class** ReverseArrayWithoutAdditionalArray {  **public** **static** **void** main(String[] args) {  *reverseArray*(**new** **int**[] { 4, 5, 8, 9, 10 });  }  **static** **void** reverseArray(**int** inputArray[]) {  System.***out***.println("Array Before Reverse : " + Arrays.*toString*(inputArray));  **int** temp;  **for** (**int** i = 0; i < inputArray.length / 2; i++) {  temp = inputArray[i];  inputArray[i] = inputArray[inputArray.length - 1 - i];  inputArray[inputArray.length - 1 - i] = temp;  }  System.***out***.println("Array After Reverse : " + Arrays.*toString*(inputArray));  System.***out***.println("=========================================");  }  } |

# **Reverse Each Word**

|  |
| --- |
| **public** **class** ReverseEachWord {  **public** **static** **void** main(String[] args) {  *reverseEachWordOfString*("Java Concept Of The Day");  }  **static** **void** reverseEachWordOfString(String inputString) {  String[] words = inputString.split(" ");  String reverseString = "";  **for** (**int** i = 0; i < words.length; i++) {  String word = words[i];  String reverseWord = "";  **for** (**int** j = word.length() - 1; j >= 0; j--) {  reverseWord = reverseWord + word.charAt(j);  }  reverseString = reverseString + reverseWord + " ";  }  System.***out***.println(inputString);  System.***out***.println(reverseString);  System.***out***.println("-------------------------");  }  } |

# **Reverse String with Preserving Position**

|  |
| --- |
| **public** **class** ReverseStringWithPreservingPosition {  **public** **static** **void** main(String[] args) {  *reverseString*("I Am Not String");  }  **static** **void** reverseString(String inputString) {  **char**[] inputStringArray = inputString.toCharArray();  **char**[] resultArray = **new** **char**[inputStringArray.length];  **for** (**int** i = 0; i < inputStringArray.length; i++) {  **if** (inputStringArray[i] == ' ') {  resultArray[i] = ' ';  }  }  **int** j = resultArray.length - 1;  **for** (**int** i = 0; i < inputStringArray.length; i++) {  **if** (inputStringArray[i] != ' ') {  **if** (resultArray[j] == ' ') {  j--;  }  resultArray[j] = inputStringArray[i];  j--;  }  }  System.***out***.println(inputString + " ---> " + String.*valueOf*(resultArray));  }  } |

# **Reverse the String**

|  |
| --- |
| **public** **class** ReverseTheString {  **public** **static** **void** main(String[] args) {  String str = "MyJava";  // 1. Using StringBuffer Class  StringBuffer sbf = **new** StringBuffer(str);  System.***out***.println(sbf.reverse()); // Output : avaJyM  // 2. Using iterative method  **char**[] strArray = str.toCharArray();  **for** (**int** i = strArray.length - 1; i >= 0; i--) {  System.***out***.print(strArray[i]); // Output : avaJyM  }  System.***out***.println();  // 3. Using Recursive Method  System.***out***.println(*recursiveMethod*(str)); // Output : avaJyM  }  // Recursive method to reverse string  **static** String recursiveMethod(String str) {  **if** ((**null** == str) || (str.length() <= 1)) {  **return** str;  }  **return** *recursiveMethod*(str.substring(1)) + str.charAt(0);  }  } |

# **Find Roman Equivalent of a Decimal Number**

|  |
| --- |
| **public** **class** RomanEquivalentDecimal {  **public** **static** **void** main(String[] args) {  String[] romanSymbols = { "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" };  **int**[] decimals = { 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter The Decimal Number Between 1 and 3999");  **int** inputNumber = sc.nextInt();  **int** copyOfInputNumber = inputNumber;  String roman = "";  **if** (inputNumber >= 1 && inputNumber <= 3999) {  **for** (**int** i = 0; i < 13; i++) {  **while** (inputNumber >= decimals[i]) {  inputNumber = inputNumber - decimals[i];  roman = roman + romanSymbols[i];  }  }  System.***out***.println("Roman Equivalent Of " + copyOfInputNumber + " is : " + roman);  } **else** {  System.***out***.println("Invalid Number");  }  }  } |

# **Saddle point of a matrix is an element in the matrix which is smallest in its row and largest in its column. A matrix can have many or no saddle points.**

|  |
| --- |
| **public** **class** SaddlePointProgram {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter the number of rows :");  **int** rows = sc.nextInt();  System.***out***.println("Enter the number of columns :");  **int** cols = sc.nextInt();  **int**[][] matrix = **new** **int**[rows][cols];  System.***out***.println("Enter the elements :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix[i][j] = sc.nextInt();  }  }  System.***out***.println("The input matrix is :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix[i][j] + "\t");  }  System.***out***.println();  }  *findSaddlePoint*(matrix);  sc.close();  }  **static** **void** findSaddlePoint(**int**[][] matrix) {  **for** (**int** i = 0; i < matrix.length; i++) {  **int** rowMin = matrix[i][0];  **int** colIndex = 0;  **boolean** saddlePoint = **true**;  **for** (**int** j = 1; j < matrix[i].length; j++) {  **if** (matrix[i][j] < rowMin) {  rowMin = matrix[i][j];  colIndex = j;  }  }  **for** (**int** j = 0; j < matrix.length; j++) {  **if** (matrix[j][colIndex] > rowMin) {  saddlePoint = **false**;  **break**;  }  }  **if** (saddlePoint) {  System.***out***.println("Saddle Point is : " + rowMin);  }  }  }  } |

# **Second Largest Number**

|  |
| --- |
| **public** **class** SecondLargestNumber {  **public** **static** **void** main(String[] args) {  System.***out***.println(*secondLargest*(**new** **int**[] { 45, 51, 28, 75, 49, 42 }));  }  **static** **int** secondLargest(**int**[] input) {  **int** firstLargest, secondLargest;  **if** (input[0] > input[1]) {  firstLargest = input[0];  secondLargest = input[1];  } **else** {  firstLargest = input[1];  secondLargest = input[0];  }  **for** (**int** i = 2; i < input.length; i++) {  **if** (input[i] > firstLargest) {  secondLargest = firstLargest;  firstLargest = input[i];  } **else** **if** (input[i] < firstLargest && input[i] > secondLargest) {  secondLargest = input[i];  }  }  **return** secondLargest;  }  } |

# **Performance of the Selection Sort Algorithm**

|  |
| --- |
| Worst Case : O(n^2) Average Case : O(n^2) Best Case : O(n^2)  **public** **class** SelectionSortInJava {  **public** **static** **void** main(String[] args) {  *selectionSort*(**new** **int**[] { 45, 84, 101, 62, 12, 45 });  }  **static** **void** selectionSort(**int**[] inputArray) {  **int** temp, pos;  **for** (**int** i = 0; i < inputArray.length - 1; i++) {  pos = i;  **for** (**int** j = i + 1; j < inputArray.length; j++) {  **if** (inputArray[j] < inputArray[pos]) {  pos = j;  }  }  temp = inputArray[i];  inputArray[i] = inputArray[pos];  inputArray[pos] = temp;  }  **for** (**int** i = 0; i < inputArray.length; i++) {  System.***out***.print(inputArray[i] + " ");  }  System.***out***.println();  }  } |

# **Separate Zeroes from NonZeros1**

|  |
| --- |
| **public** **class** SeparateZerosFromNonZeros1 {  **public** **static** **void** main(String[] args) {  *moveZerosToEnd*(**new** **int**[] { 12, 0, 7, 0, 8, 0, 3 });  }  **static** **void** moveZerosToEnd(**int** inputArray[]) {  **int** counter = 0;  **for** (**int** i = 0; i < inputArray.length; i++) {  **if** (inputArray[i] != 0) {  inputArray[counter] = inputArray[i];  counter++;  }  }  **while** (counter < inputArray.length) {  inputArray[counter] = 0;  counter++;  }  System.***out***.println(Arrays.*toString*(inputArray));  }  } |

# **Separate Zeroes from NonZeros2**

|  |
| --- |
| **public** **class** SeparateZerosFromNonZeros2 {  **public** **static** **void** main(String[] args) {  *moveZerosToFront*(**new** **int**[] { 12, 0, 7, 0, 8, 0, 3 });  }  **static** **void** moveZerosToFront(**int** inputArray[]) {  **int** counter = inputArray.length - 1;  **for** (**int** i = inputArray.length - 1; i >= 0; i--) {  **if** (inputArray[i] != 0) {  inputArray[counter] = inputArray[i];  counter--;  }  }  **while** (counter >= 0) {  inputArray[counter] = 0;  counter--;  }  System.***out***.println(Arrays.*toString*(inputArray));  }  } |

# **Set Execute Permission of a File in Java**

|  |
| --- |
| **public** **class** SetExecutePerm {  **public** **static** **void** main(String[] args) {  File file = **new** File("I:\\temp.txt");  **if** (file.exists()) {  **boolean** result = file.setExecutable(**true**);  System.***out***.println ("Is execute permission for owner set successfully? " + result);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  **if** (file.exists()) {  **boolean** result = file.setExecutable(**true**, **false**);  System.***out***.println ("Is execute permission for all set successfully? " + result);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  }  } |

# **Set File Permissions in Java 7**

|  |
| --- |
| **public** **class** SetFilePerm {  **public** **static** **void** main(String[] args) **throws** IOException {  File file = **new** File("I:\\temp.txt");  **if** (file.exists()) {  HashSet<PosixFilePermission> set = **new** HashSet<PosixFilePermission>();  // Adding owner's file permissions  set.add(PosixFilePermission.***OWNER\_EXECUTE***);  set.add(PosixFilePermission.***OWNER\_READ***);  set.add(PosixFilePermission.***OWNER\_WRITE***);  // Adding group's file permissions  set.add(PosixFilePermission.***GROUP\_EXECUTE***);  set.add(PosixFilePermission.***GROUP\_READ***);  set.add(PosixFilePermission.***GROUP\_WRITE***);  // Adding other's file permissions  set.add(PosixFilePermission.***OTHERS\_EXECUTE***);  set.add(PosixFilePermission.***OTHERS\_READ***);  set.add(PosixFilePermission.***OTHERS\_WRITE***);  Files.*setPosixFilePermissions*(Paths.*get*("I:\\temp.txt"), set);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  }  } |

# **Set Read Permission of a File in Java**

|  |
| --- |
| **public** **class** SetReadPerm {  **public** **static** **void** main(String[] args) {  File file = **new** File("I:\\temp.txt");  **if** (file.exists()) {  **boolean** result = file.setReadable(**true**);  System.***out***.println ("Is read permission for owner set successfully? " + result);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  **if** (file.exists()) {  **boolean** result = file.setReadable(**true**, **false**);  System.***out***.println ("Is read permission for all set successfully? " + result);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  }  } |

# **Set Write Permission of a File in Java**

|  |
| --- |
| **public** **class** SetWritePerm {  **public** **static** **void** main(String[] args) {  File file = **new** File("I:\\temp.txt");  **if** (file.exists()) {  **boolean** result = file.setWritable(**true**);  System.***out***.println ("Is write permission for owner set successfully? " + result);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  **if** (file.exists()) {  **boolean** result = file.setWritable(**true**, **false**);  System.***out***.println ("Is write permission for all set successfully? " + result);  } **else** {  System.***out***.println ("Sorry...File doesn't exist.");  }  }  } |

# **Sort Text File**

|  |
| --- |
| **public** **class** SortTextFile {  **public** **static** **void** main(String[] args) {  BufferedReader reader = **null**;  BufferedWriter writer = **null**;  ArrayList<String> lines = **new** ArrayList<String>();  **try** {  reader = **new** BufferedReader(**new** FileReader("C:\\input.txt"));  String currentLine = reader.readLine();  **while** (currentLine != **null**) {  lines.add(currentLine);  currentLine = reader.readLine();  }  Collections.*sort*(lines);  writer = **new** BufferedWriter(**new** FileWriter("C:\\output.txt"));  **for** (String line : lines) {  writer.write(line);  writer.newLine();  }  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (reader != **null**) {  reader.close();  }  **if** (writer != **null**) {  writer.close();  }  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# **Spiral Matrix In AntiClockwise**

|  |
| --- |
| /\*output  1 16 15 14 13  2 17 24 23 12  3 18 25 22 11  4 19 20 21 10  5 6 7 8 9  **public** **class** SpiralMatrixInAntiClockwise {  **public** **static** **void** main(String[] args) {  System.***out***.println("Enter The Value For N :");  Scanner sc = **new** Scanner(System.***in***);  **int** n = sc.nextInt();  **int**[][] spiral = **new** **int**[n][n];  **int** value = 1;  **int** minCol = 0;  **int** maxCol = n - 1;  **int** minRow = 0;  **int** maxRow = n - 1;  **while** (value <= n \* n) {  **for** (**int** i = minRow; i <= maxRow; i++) {  spiral[i][minCol] = value;  value++;  }  **for** (**int** i = minCol + 1; i <= maxCol; i++) {  spiral[maxRow][i] = value;  value++;  }  **for** (**int** i = maxRow - 1; i >= minRow; i--) {  spiral[i][maxCol] = value;  value++;  }  **for** (**int** i = maxCol - 1; i >= minCol + 1; i--) {  spiral[minRow][i] = value;  value++;  }  minCol++;  minRow++;  maxCol--;  maxRow--;  }  **for** (**int** i = 0; i < spiral.length; i++) {  **for** (**int** j = 0; j < spiral.length; j++) {  System.***out***.print(spiral[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Create spiral of numbers or spiral matrix or circular matrix of order n\*n in both clockwise and anti-clockwise directions.**

|  |
| --- |
| /\*Final Output :  1 2 3 4 5  16 17 18 19 6  15 24 25 20 7  14 23 22 21 8  13 12 11 10 9  **public** **class** SpiralMatrixInClockwise {  **public** **static** **void** main(String[] args) {  System.***out***.println("Enter The Value For N :");  Scanner sc = **new** Scanner(System.***in***);  **int** n = sc.nextInt();  **int**[][] spiral = **new** **int**[n][n];  **int** value = 1;  **int** minCol = 0;  **int** maxCol = n - 1;  **int** minRow = 0;  **int** maxRow = n - 1;  **while** (value <= n \* n) {  **for** (**int** i = minCol; i <= maxCol; i++) {  spiral[minRow][i] = value;  value++;  }  **for** (**int** i = minRow + 1; i <= maxRow; i++) {  spiral[i][maxCol] = value;  value++;  }  **for** (**int** i = maxCol - 1; i >= minCol; i--) {  spiral[maxRow][i] = value;  value++;  }  **for** (**int** i = maxRow - 1; i >= minRow + 1; i--) {  spiral[i][minCol] = value;  value++;  }  minCol++;  minRow++;  maxCol--;  maxRow--;  }  **for** (**int** i = 0; i < spiral.length; i++) {  **for** (**int** j = 0; j < spiral.length; j++) {  System.***out***.print(spiral[i][j] + "\t");  }  System.***out***.println();  }  }  } |

# **Stop a Thread in Java Using a Boolean Variable**

|  |
| --- |
| **public** **class** StopThreadUsingBooleanVariable {  **public** **static** **void** main(String[] args) {  MyThread thread = **new** MyThread();  thread.start();  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  thread.stopRunning();  }  }  **class** MyThread **extends** Thread {  **private** **volatile** **boolean** flag = **true**;  **public** **void** stopRunning() {  flag = **false**;  }  @Override  **public** **void** run() {  **while** (flag) {  System.***out***.println ("I am running....");  }  System.***out***.println ("Stopped Running....");  }  } |

# **Stop Thread Using Interrupt**

|  |
| --- |
| **public** **class** StopThreadUsingInterrupt {  **public** **static** **void** main(String[] args) {  MyThread1 thread = **new** MyThread1();  thread.start();  **try** {  Thread.*sleep*(100);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  thread.interrupt();  }  }  **class** MyThread1 **extends** Thread {  @Override  **public** **void** run() {  **while** (!Thread.*interrupted*()) {  System.***out***.println ("I am running....");  }  System.***out***.println ("Stopped Running.....");  }  } |

# **String Rotation**

|  |
| --- |
| **public** **class** StringRotation {  **public** **static** **void** main(String[] args) {  String s1 = "JavaJ2eeStrutsHibernate";  String s2 = "StrutsHibernateJavaJ2ee";  **if** (s1.length() != s2.length()) {  System.***out***.println("s2 is not rotated version of s1");  } **else** {  String s3 = s1 + s1;  **if** (s3.contains(s2)) {  System.***out***.println("s2 is a rotated version of s1");  } **else** {  System.***out***.println("s2 is not rotated version of s1");  }  }  }  } |

# **Java Program to Find Continuous Sub Array in Array Whose Sum Is Equal to Number**

|  |
| --- |
| **public** **class** SubArrayWhoseSumIsNumber1 {  **public** **static** **void** main(String[] args) {  *findSubArray*(**new** **int**[] { 42, 15, 12, 8, 6, 32 }, 26);  }  **static** **void** findSubArray(**int**[] inputArray, **int** inputNumber) {  **int** sum = inputArray[0];  **int** start = 0;  **for** (**int** i = 1; i < inputArray.length; i++) {  sum = sum + inputArray[i];  **while** (sum > inputNumber && start <= i - 1) {  sum = sum - inputArray[start];  start++;  }  **if** (sum == inputNumber) {  System.***out***.println("Continuous sub array of " + Arrays.*toString*(inputArray) + " whose sum is "  + inputNumber + " is ");  **for** (**int** j = start; j <= i; j++) {  System.***out***.print(inputArray[j] + " ");  }  System.***out***.println();  }  }  }  } |

# **Sub Array Whose Sum Is Number2**

|  |
| --- |
| **public** **class** SubArrayWhoseSumIsNumber2 {  **public** **static** **void** main(String[] args) {  *findSubArray*(**new** **int**[] { 42, 15, 12, 8, 6, 32 }, 26);  }  **static** **void** findSubArray(**int**[] inputArray, **int** inputNumber) {  **int** sum = 0;  **for** (**int** i = 0; i < inputArray.length; i++) {  sum = inputArray[i];  **for** (**int** j = i + 1; j < inputArray.length; j++) {  sum = sum + inputArray[j];  **if** (sum == inputNumber) {  System.***out***.println("Continuous sub array of " + Arrays.*toString*(inputArray) + " whose sum is "  + inputNumber + " is ");  **for** (**int** k = i; k <= j; k++) {  System.***out***.print(inputArray[k] + " ");  }  System.***out***.println();  } **else** **if** (sum < inputNumber) {  **continue**;  } **else** **if** (sum > inputNumber) {  **break**;  }  }  }  }  } |

# **Sum of All Digits of Number**

|  |
| --- |
| **public** **class** SumOfAllDigitsOfNumber {  **public** **static** **void** main(String[] args) {  *sumOfAllDigits*(47862);  }  **static** **void** sumOfAllDigits(**int** inputNumber) {  **int** copyOfInputNumber = inputNumber;  **int** sum = 0;  **while** (copyOfInputNumber != 0) {  **int** lastDigit = copyOfInputNumber % 10;  sum = sum + lastDigit;  copyOfInputNumber = copyOfInputNumber / 10;  }  System.***out***.println("Sum Of All Digits In " + inputNumber + " = " + sum);  }  } |

# **Swap Two Strings**

|  |
| --- |
| **public** **class** SwapTwoStrings {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter First String :");  String s1 = sc.next();  System.***out***.println("Enter Second String :");  String s2 = sc.next();  System.***out***.println("Before Swapping :");  System.***out***.println("s1 : " + s1);  System.***out***.println("s2 : " + s2);  s1 = s1 + s2;  s2 = s1.substring(0, s1.length() - s2.length());  s1 = s1.substring(s2.length());  System.***out***.println("After Swapping :");  System.***out***.println("s1 : " + s1);  System.***out***.println("s2 : " + s2);  }  } |

# **Symmetric Matrix**

|  |
| --- |
| /\*Printing the input matrix :  7 5 1  5 8 3  1 3 4  The given matrix is symmetric…  **public** **class** SymmetricMatrix {  **public** **static** **void** main(String[] args) {  Scanner sc = **new** Scanner(System.***in***);  System.***out***.println("Enter the no. of rows : ");  **int** rows = sc.nextInt();  System.***out***.println("Enter the no. of columns : ");  **int** cols = sc.nextInt();  **int** matrix[][] = **new** **int**[rows][cols];  System.***out***.println("Enter the elements :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  matrix[i][j] = sc.nextInt();  }  }  System.***out***.println("Printing the input matrix :");  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  System.***out***.print(matrix[i][j] + "\t");  }  System.***out***.println();  }  **if** (rows != cols) {  System.***out***.println ("The given matrix is not a square matrix, so it can't be symmetric.");  } **else** {  **boolean** symmetric = **true**;  **for** (**int** i = 0; i < rows; i++) {  **for** (**int** j = 0; j < cols; j++) {  **if** (matrix[i][j] != matrix[j][i]) {  symmetric = **false**;  **break**;  }  }  }  **if** (symmetric) {  System.***out***.println ("The given matrix is symmetric...");  } **else** {  System.***out***.println ("The given matrix is not symmetric...");  }  }  sc.close();  }  } |

# **Write a java program or function to replace specific string in text file. Your program should take one text file as input and replace a specific string in that text file with new string.**

|  |
| --- |
| **public** **class** TextFileModification {  **public** **static** **void** main(String[] args) {  *modifyFile*("C:/StudentFile.txt", "85", "95");  System.***out***.println("done");  }  **static** **void** modifyFile(String filePath, String oldString, String newString) {  File fileToBeModified = **new** File(filePath);  String oldContent = "";  BufferedReader reader = **null**;  FileWriter writer = **null**;  **try** {  reader = **new** BufferedReader(**new** FileReader(fileToBeModified));  String line = reader.readLine();  **while** (line != **null**) {  oldContent = oldContent + line + System.*lineSeparator*();  line = reader.readLine();  }  String newContent = oldContent.replaceAll(oldString, newString);  writer = **new** FileWriter(fileToBeModified);  writer.write(newContent);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  reader.close();  writer.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# **Trigonometric Functions**

|  |
| --- |
| **public** **class** TrigonometricFunctions {  **public** **static** **void** main(String[] args) {  **double** angle = 90;.  **double** angleInRadians = Math.*toRadians*(angle);  **double** sineOfNum = Math.*sin*(angleInRadians);  **double** cosineOfNum = Math.*cos*(angleInRadians);  **double** tangentOfNum = Math.*tan*(angleInRadians);  **double** secOfNum = (1 / sineOfNum);  **double** cosecOfNum = (1 / cosineOfNum);  **double** cotOfNum = (1 / tangentOfNum);  System.***out***.println("sin(" + angle + ") = " + sineOfNum);  System.***out***.println("cos(" + angle + ") = " + cosineOfNum);  System.***out***.println("tan(" + angle + ") = " + tangentOfNum);  System.***out***.println("--------------------------------");  System.***out***.println("sec(" + angle + ") = " + secOfNum);  System.***out***.println("cosec(" + angle + ") = " + cosecOfNum);  System.***out***.println("cot(" + angle + ") = " + cotOfNum);  }  } |

# **Find Union of Multiple Arrays Using HashSet**

|  |
| --- |
| **public** **class** UnionOfMultipleArrays {  **public** **static** **void** main(String[] args) {  **int**[] inputArray1 = { 2, 3, 4, 7, 1 };  **int**[] inputArray2 = { 4, 1, 3, 5 };  **int**[] inputArray3 = { 8, 4, 6, 2, 1 };  **int**[] inputArray4 = { 7, 9, 4, 1 };  *union*(inputArray1, inputArray2, inputArray3, inputArray4);  }  **static** **void** union(**int**[]... inputArrays) {  HashSet<Integer> unionSet = **new** HashSet<Integer>();  System.***out***.println("Input Arrays :");  System.***out***.println("======================");  **for** (**int**[] inputArray : inputArrays) {  System.***out***.println(Arrays.*toString*(inputArray));  **for** (**int** i : inputArray) {  unionSet.add(i);  }  }  System.***out***.println("===========================");  System.***out***.println("Union Of All Input Arrays :");  System.***out***.println("===========================");  System.***out***.println(unionSet);  }  } |

# **Word Count in File**

|  |
| --- |
| **public** **class** WordCountInFile {  **public** **static** **void** main(String[] args) {  BufferedReader reader = **null**;  **int** charCount = 0;  **int** wordCount = 0;  **int** lineCount = 0;  **try** {  reader = **new** BufferedReader(**new** FileReader("C:\\sample.txt"));  String currentLine = reader.readLine();  **while** (currentLine != **null**) {  lineCount++;  String[] words = currentLine.split(" ");  wordCount = wordCount + words.length;  **for** (String word : words) {  charCount = charCount + word.length();  }  currentLine = reader.readLine();  }  System.***out***.println("Number Of Chars In A File : " + charCount);  System.***out***.println("Number Of Words In A File : " + wordCount);  System.***out***.println("Number Of Lines In A File : " + lineCount);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  reader.close(); // Closing the reader  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

# **Given a Boolean matrix mat [M] [N] of size M X N, modify it such that if a matrix cell mat[i] [j] is 1 (or true) then make all the cells of ith row and jth column as 1. Using two temporary arrays.**

|  |
| --- |
| Input Matrix  1 0 0 1  0 0 1 0  0 0 0 0  Matrix after modification  1 1 1 1  1 1 1 1  1 0 1 1  Time Complexity: O(M\*N) Auxiliary Space: O(M + N)  **public** **class** BooleanMatrix {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 0, 0, 1 }, { 0, 0, 1, 0 }, { 0, 0, 0, 0 }, };  System.***out***.println("Matrix Intially");  *printMatrix*(mat, 3, 4);  *modifyMatrix*(mat, 3, 4);  System.***out***.println("Matrix after modification n");  *printMatrix*(mat, 3, 4);  }  **public** **static** **void** modifyMatrix(**int** mat[][], **int** R, **int** C) {  **int** row[] = **new** **int**[R];  **int** col[] = **new** **int**[C];  **int** i, j;  **for** (i = 0; i < R; i++) {  row[i] = 0;  }  **for** (i = 0; i < C; i++) {  col[i] = 0;  }  **for** (i = 0; i < R; i++) {  **for** (j = 0; j < C; j++) {  **if** (mat[i][j] == 1) {  row[i] = 1;  col[j] = 1;  }  }  }  **for** (i = 0; i < R; i++) {  **for** (j = 0; j < C; j++) {  **if** (row[i] == 1 || col[j] == 1) {  mat[i][j] = 1;  }  }  }  }  **public** **static** **void** printMatrix(**int** mat[][], **int** R, **int** C) {  **int** i, j;  **for** (i = 0; i < R; i++) {  **for** (j = 0; j < C; j++) {  System.***out***.print(mat[i][j] + " ");  }  System.***out***.println();  }  }  } |

# **Boolean Matrix1 Method 2 (A Space Optimized Version of Method 1)**

|  |
| --- |
| Time Complexity: O(M\*N) Auxiliary Space: O(1)  **public** **class** BooleanMatrix1 {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 0, 0, 1 }, { 0, 0, 1, 0 }, { 0, 0, 0, 0 } };  System.***out***.println("Input Matrix :");  *printMatrix*(mat);  *modifyMatrix*(mat);  System.***out***.println("Matrix After Modification :");  *printMatrix*(mat);  }  **public** **static** **void** modifyMatrix(**int** mat[][]) {  **boolean** row\_flag = **false**;  **boolean** col\_flag = **false**;  **for** (**int** i = 0; i < mat.length; i++) {  **for** (**int** j = 0; j < mat[0].length; j++) {  **if** (i == 0 && mat[i][j] == 1)  row\_flag = **true**;  **if** (j == 0 && mat[i][j] == 1)  col\_flag = **true**;  **if** (mat[i][j] == 1) {  mat[0][j] = 1;  mat[i][0] = 1;  }  }  }  **for** (**int** i = 1; i < mat.length; i++) {  **for** (**int** j = 1; j < mat[0].length; j++) {  **if** (mat[0][j] == 1 || mat[i][0] == 1) {  mat[i][j] = 1;  }  }  }  **if** (row\_flag == **true**) {  **for** (**int** i = 0; i < mat[0].length; i++) {  mat[0][i] = 1;  }  }  **if** (col\_flag == **true**) {  **for** (**int** i = 0; i < mat.length; i++) {  mat[i][0] = 1;  }  }  }  **public** **static** **void** printMatrix(**int** mat[][]) {  **for** (**int** i = 0; i < mat.length; i++) {  **for** (**int** j = 0; j < mat[0].length; j++) {  System.***out***.print(mat[i][j]);  }  System.***out***.println("");  }  }  } |

# **Find a common element in all rows of a given row-wise sorted matrix: -** given a matrix where every row is sorted in increasing order. Write a function that finds and returns a common element in all rows. If there is no common element, then return -1.

|  |
| --- |
| Input: mat[4][5] = { {1, 2, 3, 4, 5},  {2, 4, 5, 8, 10},  {3, 5, 7, 9, 11},  {1, 3, 5, 7, 9},  };  Output: 5 Time complexity of the above hashing based solution is O(MN)  **public** **class** CommonElementinAllRows {  **static** **final** **int** ***M*** = 4;  **static** **final** **int** ***N*** = 5;  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 2, 3, 4, 5 }, { 2, 4, 5, 8, 10 }, { 3, 5, 7, 9, 11 }, { 1, 3, 5, 7, 9 } };  **int** result = *findCommon*(mat);  **if** (result == -1)  System.***out***.print("No common element");  **else**  System.***out***.print("Common element is " + result);  }  **static** **int** findCommon(**int** mat[][]) {  **int** column[] = **new** **int**[***M***];  **int** min\_row;  **int** i;  **for** (i = 0; i < ***M***; i++)  column[i] = ***N*** - 1;  min\_row = 0;  **while** (column[min\_row] >= 0) {  **for** (i = 0; i < ***M***; i++) {  **if** (mat[i][column[i]] < mat[min\_row][column[min\_row]])  min\_row = i;  }  **int** eq\_count = 0;  **for** (i = 0; i < ***M***; i++) {  **if** (mat[i][column[i]] > mat[min\_row][column[min\_row]]) {  **if** (column[i] == 0)  **return** -1;  column[i] -= 1;  } **else**  eq\_count++;  }  **if** (eq\_count == ***M***)  **return** mat[min\_row][column[min\_row]];  }  **return** -1;  }  } |

# **Count Number of islands where every island is row-wise and column-wise separated: -** given a rectangular matrix which has only two possible values ’X’ and ‘O’. The values ‘X’ always appear in form of rectangular islands and these islands are always row-wise and column-wise and column-wise separated by at least on line of ‘O’s. Note that islands can only be diagonally adjacent.

|  |
| --- |
| mat[M][N] = {{'O', 'O', 'O'},  {'X', 'X', 'O'},  {'X', 'X', 'O'},  {'O', 'O', 'X'},  {'O', 'O', 'X'},  {'X', 'X', 'O'}  };  Output: Number of islands is 3  Time complexity of this solution is O (MN).  **public** **class** CountNumberofIslands {  **public** **static** **void** main(String[] args) {  **int** m = 6;  **int** n = 3;  **int** mat[][] = { { 'O', 'O', 'O' }, { 'X', 'X', 'O' }, { 'X', 'X', 'O' }, { 'O', 'O', 'X' }, { 'O', 'O', 'X' },  { 'X', 'X', 'O' } };  System.***out***.println("Number of rectangular islands is: " + *countIslands*(mat, m, n));  }  **static** **int** countIslands(**int** mat[][], **int** m, **int** n) {  **int** count = 0;  **for** (**int** i = 0; i < m; i++) {  **for** (**int** j = 0; j < n; j++) {  **if** (mat[i][j] == 'X') {  **if** ((i == 0 || mat[i - 1][j] == 'O') && (j == 0 || mat[i][j - 1] == 'O'))  count++;  }  }  }  **return** count;  }  } |

# **Create a matrix with alternating rectangles of O and X**

|  |
| --- |
| Input: m = 3, n = 3  Output: Following matrix  X X X  X 0 X  X X X  Time Complexity: O(mn)  Auxiliary Space: O(mn)  **public** **class** CreateMatrixWithAlternatingRectangles {  **public** **static** **void** main(String[] args) {  System.***out***.println("Output for m = 5, n = 6");  *fill0X*(5, 6);  }  **static** **void** fill0X(**int** m, **int** n) {  **int** i, k = 0, l = 0;  **int** r = m, c = n;  **char** a[][] = **new** **char**[m][n];  **char** x = 'X';  **while** (k < m && l < n) {  **for** (i = l; i < n; ++i)  a[k][i] = x;  k++;  **for** (i = k; i < m; ++i)  a[i][n - 1] = x;  n--;  **if** (k < m) {  **for** (i = n - 1; i >= l; --i)  a[m - 1][i] = x;  m--;  }  **if** (l < n) {  **for** (i = m - 1; i >= k; --i)  a[i][l] = x;  l++;  }  x = (x == '0') ? 'X' : '0';  }  **for** (i = 0; i < r; i++) {  **for** (**int** j = 0; j < c; j++)  System.***out***.print(a[i][j] + " ");  System.***out***.println();  }  }  } |

# **Given an n x n square matrix, find sum of all sub-squares of size k x k**

|  |
| --- |
| Input: n = 5, k = 3  arr[][] = { {1, 1, 1, 1, 1},  {2, 2, 2, 2, 2},  {3, 3, 3, 3, 3},  {4, 4, 4, 4, 4},  {5, 5, 5, 5, 5},  };  Output:  18 18 18  27 27 27  36 36 36  Time complexity of above solution is O (k2n2).  **public** **class** FindSumofAllSubSquares {  **static** **final** **int** ***n*** = 5;  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 1, 1, 1, 1 }, { 2, 2, 2, 2, 2 }, { 3, 3, 3, 3, 3 }, { 4, 4, 4, 4, 4 }, { 5, 5, 5, 5, 5 } };  **int** k = 3;  *printSumSimple*(mat, k);  }  **static** **void** printSumSimple(**int** mat[][], **int** k) {  **if** (k > ***n***)  **return**;  **for** (**int** i = 0; i < ***n*** - k + 1; i++) {  **for** (**int** j = 0; j < ***n*** - k + 1; j++) {  **int** sum = 0;  **for** (**int** p = i; p < k + i; p++)  **for** (**int** q = j; q < k + j; q++)  sum += mat[p][q];  System.***out***.print(sum + " ");  }  System.***out***.println();  }  }  } |

# **Find Sum of All Sub Squares1**

|  |
| --- |
| **public** **class** FindSumofAllSubSquares1 {  **static** **int** *n* = 5;  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 1, 1, 1, 1, 1 }, { 2, 2, 2, 2, 2 }, { 3, 3, 3, 3, 3 }, { 4, 4, 4, 4, 4 },  { 5, 5, 5, 5, 5 }, };  **int** k = 3;  *printSumTricky*(mat, k);  }  **static** **void** printSumTricky(**int** mat[][], **int** k) {  **if** (k > *n*)  **return**;  **int** stripSum[][] = **new** **int**[*n*][*n*];  **for** (**int** j = 0; j < *n*; j++) {  **int** sum = 0;  **for** (**int** i = 0; i < k; i++)  sum += mat[i][j];  stripSum[0][j] = sum;  **for** (**int** i = 1; i < *n* - k + 1; i++) {  sum += (mat[i + k - 1][j] - mat[i - 1][j]);  stripSum[i][j] = sum;  }  }  **for** (**int** i = 0; i < *n* - k + 1; i++) {  **int** sum = 0;  **for** (**int** j = 0; j < k; j++)  sum += stripSum[i][j];  System.***out***.print(sum + " ");  **for** (**int** j = 1; j < *n* - k + 1; j++) {  sum += (stripSum[i][j + k - 1] - stripSum[i][j - 1]);  System.***out***.print(sum + " ");  }  System.***out***.println();  }  }  } |

# **Maximum size square sub-matrix with all 1s:-** given a binary matrix, find out the maximum size square sub-matrix with all 1s.

|  |
| --- |
| For example, consider the below binary matrix.  0 1 1 0 1  1 1 0 1 0  0 1 1 1 0  1 1 2 2 0  1 2 2 3 1  0 0 0 0 0  Time Complexity: O (m\*n) Auxiliary Space: O(m\*n)  **public** **class** MaximumSizeSquare {  **public** **static** **void** main(String[] args) {  **int** M[][] = { { 0, 1, 1, 0, 1 }, { 1, 1, 0, 1, 0 }, { 0, 1, 1, 1, 0 }, { 1, 1, 1, 1, 0 }, { 1, 1, 1, 1, 1 },  { 0, 0, 0, 0, 0 } };  *printMaxSubSquare*(M);  }  **static** **void** printMaxSubSquare(**int** M[][]) {  **int** i, j;  **int** R = M.length;  **int** C = M[0].length;  **int** S[][] = **new** **int**[R][C];  **int** max\_of\_s, max\_i, max\_j;  **for** (i = 0; i < R; i++)  S[i][0] = M[i][0];  **for** (j = 0; j < C; j++)  S[0][j] = M[0][j];  **for** (i = 1; i < R; i++) {  **for** (j = 1; j < C; j++) {  **if** (M[i][j] == 1)  S[i][j] = Math.*min*(S[i][j - 1], Math.*min*(S[i - 1][j], S[i - 1][j - 1])) + 1;  **else**  S[i][j] = 0;  }  }  max\_of\_s = S[0][0];  max\_i = 0;  max\_j = 0;  **for** (i = 0; i < R; i++) {  **for** (j = 0; j < C; j++) {  **if** (max\_of\_s < S[i][j]) {  max\_of\_s = S[i][j];  max\_i = i;  max\_j = j;  }  }  }  System.***out***.println("Maximum size sub-matrix is: ");  **for** (i = max\_i; i > max\_i - max\_of\_s; i--) {  **for** (j = max\_j; j > max\_j - max\_of\_s; j--) {  System.***out***.print(M[i][j] + " ");  }  System.***out***.println();  }  }  } |

# **Given a 2D array, print it in spiral form. See the following examples.**

|  |
| --- |
| Input:  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16  Output: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10  Time complexity of the above solution is O (mn).  **public** **class** PrintGivenMatrixSpiralForm {  **public** **static** **void** main(String[] args) {  **int** R = 3;  **int** C = 6;  **int** a[][] = { { 1, 2, 3, 4, 5, 6 }, { 7, 8, 9, 10, 11, 12 }, { 13, 14, 15, 16, 17, 18 } };  *spiralPrint*(R, C, a);  }  **static** **void** spiralPrint(**int** m, **int** n, **int** a[][]) {  **int** i, k = 0, l = 0;  **while** (k < m && l < n) {  **for** (i = l; i < n; ++i) {  System.***out***.print(a[k][i] + " ");  }  k++;  **for** (i = k; i < m; ++i) {  System.***out***.print(a[i][n - 1] + " ");  }  n--;  **if** (k < m) {  **for** (i = n - 1; i >= l; --i) {  System.***out***.print(a[m - 1][i] + " ");  }  m--;  }  **if** (l < n) {  **for** (i = m - 1; i >= k; --i) {  System.***out***.print(a[i][l] + " ");  }  l++;  }  }  }  } |

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

|  |
| --- |
| 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) Time Complexity: O(n)  **public** **class** SearchRowWiseColumnWiseSortedMatrix {  **public** **static** **void** main(String[] args) {  **int** mat[][] = { { 10, 20, 30, 40 }, { 15, 25, 35, 45 }, { 27, 29, 37, 48 }, { 32, 33, 39, 50 } };  *search*(mat, 4, 29);  }  **private** **static** **void** search(**int**[][] mat, **int** n, **int** x) {  **int** i = 0, j = n - 1;  **while** (i < n && j >= 0) {  **if** (mat[i][j] == x) {  System.***out***.print("n Found at " + i + " " + j);  **return**;  }  **if** (mat[i][j] > x)  j--;  **else**  i++;  }  System.***out***.print("n Element not found");  **return**;  }  } |

# **Implement Binary Search Tree (BST)**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl bst = **new** BinarySearchTreeImpl();  bst.insert(10);  bst.insert(20);  bst.insert(21);  bst.insert(8);  bst.insert(6);  bst.insert(16);  bst.insert(23);  }  }  **class** BstNode {  **private** BstNode left;  **private** BstNode right;  **private** Integer data;  **public** BstNode(Integer data) {  **this**.data = data;  }  **public** BstNode getLeft() {  **return** left;  }  **public** **void** setLeft(BstNode left) {  **this**.left = left;  }  **public** BstNode getRight() {  **return** right;  }  **public** **void** setRight(BstNode right) {  **this**.right = right;  }  **public** Integer getData() {  **return** data;  }  **public** **void** setData(Integer data) {  **this**.data = data;  }  } |

# **Find min and max value from Binary Search Tree (BST)**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl1 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** Integer findMinValue() {  **return** minValue(**this**.root);  }  **public** Integer findMaxValue() {  **return** maxValue(**this**.root);  }  **private** Integer minValue(BstNode node) {  **if** (node.getLeft() != **null**) {  **return** minValue(node.getLeft());  }  **return** node.getData();  }  **private** Integer maxValue(BstNode node) {  **if** (node.getRight() != **null**) {  **return** maxValue(node.getRight());  }  **return** node.getData();  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl1 bst = **new** BinarySearchTreeImpl1();  bst.insert(10);  bst.insert(20);  bst.insert(21);  bst.insert(8);  bst.insert(6);  bst.insert(16);  bst.insert(23);  bst.insert(2);  System.***out***.println("-------------------");  System.***out***.println("Min value: " + bst.findMinValue());  System.***out***.println("Max value: " + bst.findMaxValue());  }  } |

# **Find height of a Binary Search Tree (BST)**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl2 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** Integer findHeight() {  **return** getNodeHeight(**this**.root);  }  **private** Integer getNodeHeight(BstNode node) {  **if** (node == **null**) {  **return** -1;  }  **return** Math.*max*(getNodeHeight(node.getLeft()), getNodeHeight(node.getRight())) + 1;  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl2 bst = **new** BinarySearchTreeImpl2();  bst.insert(10);  bst.insert(20);  bst.insert(21);  bst.insert(8);  bst.insert(6);  bst.insert(16);  bst.insert(23);  bst.insert(2);  System.***out***.println("-------------------");  System.***out***.println("Height of the tree: " + bst.findHeight());  }  } |

# **Implement Binary Search Tree (BST) Level order traversal (breadth first).**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl4 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** **void** levelOrderTraversal() {  Queue<BstNode> discovedNodeQueue = **new** LinkedList<>();  **if** (**this**.root == **null**) {  System.***out***.println ("The tree is empty.");  **return**;  }  discovedNodeQueue.add(**this**.root);  **while** (!discovedNodeQueue.isEmpty()) {  BstNode tmpNode = discovedNodeQueue.remove();  **if** (tmpNode.getLeft() != **null**) {  discovedNodeQueue.add(tmpNode.getLeft());  }  **if** (tmpNode.getRight() != **null**) {  discovedNodeQueue.add(tmpNode.getRight());  }  System.***out***.print(tmpNode.getData() + " ");  }  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl4 bst = **new** BinarySearchTreeImpl4();  bst.insert(8);  bst.insert(10);  bst.insert(14);  bst.insert(3);  bst.insert(6);  bst.insert(7);  bst.insert(1);  bst.insert(4);  bst.insert(13);  System.***out***.println("-------------------");  System.***out***.println("Level order traversal");  bst.levelOrderTraversal();  }  } |

# **Implement Binary Search Tree (BST) pre-order traversal (depth first).**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl5 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** **void** preOrderTraversal() {  doPreOrder(**this**.root);  }  **private** **void** doPreOrder(BstNode root) {  **if** (root == **null**)  **return**;  System.***out***.print(root.getData() + " ");  doPreOrder(root.getLeft());  doPreOrder(root.getRight());  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl5 bst = **new** BinarySearchTreeImpl5();  bst.insert(8);  bst.insert(10);  bst.insert(14);  bst.insert(3);  bst.insert(6);  bst.insert(7);  bst.insert(1);  bst.insert(4);  bst.insert(13);  System.***out***.println("\n-------------------");  System.***out***.println("Pre Order Traversal");  bst.preOrderTraversal();  }  } |

# **Implement Binary Search Tree (BST) in-order traversal (depth first).**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl6 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** **void** inOrderTraversal() {  doInOrder(**this**.root);  }  **private** **void** doInOrder(BstNode root) {  **if** (root == **null**)  **return**;  doInOrder(root.getLeft());  System.***out***.print(root.getData() + " ");  doInOrder(root.getRight());  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl6 bst = **new** BinarySearchTreeImpl6();  bst.insert(8);  bst.insert(10);  bst.insert(14);  bst.insert(3);  bst.insert(6);  bst.insert(7);  bst.insert(1);  bst.insert(4);  bst.insert(13);  System.***out***.println("\n-------------------");  System.***out***.println("In Order Traversal");  bst.inOrderTraversal();  }  } |

# **Implement Binary Search Tree (BST) post-order traversal (depth first).**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl7 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** **void** postOrderTraversal() {  doPostOrder(**this**.root);  }  **private** **void** doPostOrder(BstNode root) {  **if** (root == **null**)  **return**;  doPostOrder(root.getLeft());  doPostOrder(root.getRight());  System.***out***.print(root.getData() + " ");  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl7 bst = **new** BinarySearchTreeImpl7();  bst.insert(8);  bst.insert(10);  bst.insert(14);  bst.insert(3);  bst.insert(6);  bst.insert(7);  bst.insert(1);  bst.insert(4);  bst.insert(13);  System.***out***.println("\n-------------------");  System.***out***.println("Post Order Traversal");  bst.postOrderTraversal();  }  } |

# **How to delete a node from Binary Search Tree (BST)?**

|  |
| --- |
| **public** **class** BinarySearchTreeImpl8 {  **private** BstNode root;  **public** **boolean** isEmpty() {  **return** (**this**.root == **null**);  }  **public** BstNode getRoot() {  **return** **this**.root;  }  **public** **void** insert(Integer data) {  System.***out***.print("[input: " + data + "]");  **if** (root == **null**) {  **this**.root = **new** BstNode(data);  System.***out***.println(" -> inserted: " + data);  **return**;  }  insertNode(**this**.root, data);  System.***out***.print(" -> inserted: " + data);  System.***out***.println();  }  **private** BstNode insertNode(BstNode root, Integer data) {  BstNode tmpNode = **null**;  System.***out***.print(" ->" + root.getData());  **if** (root.getData() >= data) {  System.***out***.print(" [L]");  **if** (root.getLeft() == **null**) {  root.setLeft(**new** BstNode(data));  **return** root.getLeft();  } **else** {  tmpNode = root.getLeft();  }  } **else** {  System.***out***.print(" [R]");  **if** (root.getRight() == **null**) {  root.setRight(**new** BstNode(data));  **return** root.getRight();  } **else** {  tmpNode = root.getRight();  }  }  **return** insertNode(tmpNode, data);  }  **public** **void** delete(Integer data) {  deleteNode(**this**.root, data);  }  **private** BstNode deleteNode(BstNode root, Integer data) {  **if** (root == **null**)  **return** root;  **if** (data < root.getData()) {  root.setLeft(deleteNode(root.getLeft(), data));  } **else** **if** (data > root.getData()) {  root.setRight(deleteNode(root.getRight(), data));  } **else** {  // node with no leaf nodes  **if** (root.getLeft() == **null** && root.getRight() == **null**) {  System.***out***.println("deleting " + data);  **return** **null**;  } **else** **if** (root.getLeft() == **null**) {  // node with one node (no left node)  System.***out***.println("deleting " + data);  **return** root.getRight();  } **else** **if** (root.getRight() == **null**) {  // node with one node (no right node)  System.***out***.println("deleting " + data);  **return** root.getLeft();  } **else** {  // nodes with two nodes  // search for min number in right sub tree  Integer minValue = minValue(root.getRight());  root.setData(minValue);  root.setRight(deleteNode(root.getRight(), minValue));  System.***out***.println("deleting " + data);  }  }  **return** root;  }  **private** Integer minValue(BstNode node) {  **if** (node.getLeft() != **null**) {  **return** minValue(node.getLeft());  }  **return** node.getData();  }  **public** **void** inOrderTraversal() {  doInOrder(**this**.root);  }  **private** **void** doInOrder(BstNode root) {  **if** (root == **null**)  **return**;  doInOrder(root.getLeft());  System.***out***.print(root.getData() + " ");  doInOrder(root.getRight());  }  **public** **static** **void** main(String a[]) {  BinarySearchTreeImpl8 bst = **new** BinarySearchTreeImpl8();  bst.insert(8);  bst.insert(10);  bst.insert(14);  bst.insert(3);  bst.insert(6);  bst.insert(7);  bst.insert(1);  bst.insert(4);  bst.insert(13);  System.***out***.println("-------------------");  System.***out***.println("In Order Traversal");  bst.inOrderTraversal();  System.***out***.println();  bst.delete(13);  bst.inOrderTraversal();  System.***out***.println();  bst.delete(14);  bst.inOrderTraversal();  }  } |

# **Binary to Decimal**

|  |
| --- |
| **public** **class** BinaryToDecimal {  **public** **int** getDecimalFromBinary(**int** binary) {  **int** decimal = 0;  **int** power = 0;  **while** (**true**) {  **if** (binary == 0) {  **break**;  } **else** {  **int** tmp = binary % 10;  decimal += tmp \* Math.*pow*(2, power);  binary = binary / 10;  power++;  }  }  **return** decimal;  }  **public** **static** **void** main(String a[]) {  BinaryToDecimal bd = **new** BinaryToDecimal();  System.***out***.println("11 ===> " + bd.getDecimalFromBinary(11));  }  } |

# **Decimal to Binary**

|  |
| --- |
| **public** **class** DecimalToBinary {  **public** **void** printBinaryFormat(**int** number) {  **int** binary[] = **new** **int**[25];  **int** index = 0;  **while** (number > 0) {  binary[index++] = number % 2;  number = number / 2;  }  **for** (**int** i = index - 1; i >= 0; i--) {  System.***out***.print(binary[i]);  }  }  **public** **static** **void** main(String a[]) {  DecimalToBinary dtb = **new** DecimalToBinary();  dtb.printBinaryFormat(25);  }  } |

# **Write a program to find common elements between two arrays.**

|  |
| --- |
| **public** **class** CommonElementsInArray {  **public** **static** **void** main(String a[]) {  **int**[] arr1 = { 4, 7, 3, 9, 2 };  **int**[] arr2 = { 3, 2, 12, 9, 40, 32, 4 };  **for** (**int** i = 0; i < arr1.length; i++) {  **for** (**int** j = 0; j < arr2.length; j++) {  **if** (arr1[i] == arr2[j]) {  System.***out***.println(arr1[i]);  }  }  }  }  } |

# **Write a program to find out duplicate characters in a string.**

|  |
| --- |
| **public** **class** DuplicateCharsInString {  **public** **void** findDuplicateChars(String str) {  Map<Character, Integer> dupMap = **new** HashMap<Character, Integer>();  **char**[] chrs = str.toCharArray();  **for** (Character ch : chrs) {  **if** (dupMap.containsKey(ch)) {  dupMap.put(ch, dupMap.get(ch) + 1);  } **else** {  dupMap.put(ch, 1);  }  }  Set<Character> keys = dupMap.keySet();  **for** (Character ch : keys) {  **if** (dupMap.get(ch) > 1) {  System.***out***.println(ch + "--->" + dupMap.get(ch));  }  }  }  **public** **static** **void** main(String a[]) {  DuplicateCharsInString dcs = **new** DuplicateCharsInString();  dcs.findDuplicateChars("Java2Novice");  }  } |

# **Duplicate Number**

|  |
| --- |
| **public** **class** DuplicateNumber {  **public** **int** findDuplicateNumber(List<Integer> numbers) {  **int** highestNumber = numbers.size() - 1;  **int** total = getSum(numbers);  **int** duplicate = total - (highestNumber \* (highestNumber + 1) / 2);  **return** duplicate;  }  **public** **int** getSum(List<Integer> numbers) {  **int** sum = 0;  **for** (**int** num : numbers) {  sum += num;  }  **return** sum;  }  **public** **static** **void** main(String a[]) {  List<Integer> numbers = **new** ArrayList<Integer>();  **for** (**int** i = 1; i < 30; i++) {  numbers.add(i);  }  // add duplicate number into the list  numbers.add(22);  DuplicateNumber dn = **new** DuplicateNumber();  System.***out***.println("Duplicate Number: " + dn.findDuplicateNumber(numbers));  }  } |

# **Find out middle index where sum of both ends are equal.**

|  |
| --- |
| **public** **class** FindMiddleIndex {  **public** **static** **int** findMiddleIndex(**int**[] numbers) **throws** Exception {  **int** endIndex = numbers.length - 1;  **int** startIndex = 0;  **int** sumLeft = 0;  **int** sumRight = 0;  **while** (**true**) {  **if** (sumLeft > sumRight) {  sumRight += numbers[endIndex--];  } **else** {  sumLeft += numbers[startIndex++];  }  **if** (startIndex > endIndex) {  **if** (sumLeft == sumRight) {  **break**;  } **else** {  **throw** **new** Exception("Please pass proper array to match the requirement");  }  }  }  **return** endIndex;  }  **public** **static** **void** main(String a[]) {  **int**[] num = { 2, 4, 4, 5, 4, 1 };  **try** {  System.***out***.println("Starting from index 0, adding numbers till index " + *findMiddleIndex*(num) + " and");  System.***out***.println("adding rest of the numbers can be equal");  } **catch** (Exception ex) {  System.***out***.println(ex.getMessage());  }  }  } |

# **How to check the given Binary Tree is Binary Search Tree (BST) or not?**

|  |
| --- |
| **public** **class** IsBinarySearchTree {  **public** **boolean** isBinarySearchTree(BstNode root) {  **if** (root == **null**)  **return** Boolean.***TRUE***;  **return** isBstValid(root, Integer.***MIN\_VALUE***, Integer.***MAX\_VALUE***);  }  **private** **boolean** isBstValid(BstNode root, Integer minValue, Integer maxValue) {  **if** (root == **null**)  **return** Boolean.***TRUE***;  **if** (root.getData() >= minValue && root.getData() < maxValue  && isBstValid(root.getLeft(), minValue, root.getData())  && isBstValid(root.getRight(), root.getData(), maxValue)) {  **return** Boolean.***TRUE***;  } **else** {  **return** Boolean.***FALSE***;  }  }  **public** **static** **void** main(String a[]) {  BstNode root = **new** BstNode(3);  // left sub tree  BstNode node\_2 = **new** BstNode(2);  root.setLeft(node\_2);  BstNode node\_1 = **new** BstNode(1);  node\_2.setLeft(node\_1);  BstNode node\_4 = **new** BstNode(4);  node\_2.setRight(node\_4);  // right sub tree  BstNode node\_6 = **new** BstNode(6);  root.setRight(node\_6);  BstNode node\_5 = **new** BstNode(5);  node\_6.setLeft(node\_5);  BstNode node\_7 = **new** BstNode(7);  node\_6.setRight(node\_7);  IsBinarySearchTree ibsTree = **new** IsBinarySearchTree();  System.***out***.println(ibsTree.isBinarySearchTree(root));  }  } |

# **Is Perfect Number**

|  |
| --- |
| **public** **class** IsPerfectNumber {  **public** **boolean** isPerfectNumber(**int** number) {  **int** temp = 0;  **for** (**int** i = 1; i <= number / 2; i++) {  **if** (number % i == 0) {  temp += i;  }  }  **if** (temp == number) {  System.***out***.println("It is a perfect number");  **return** **true**;  } **else** {  System.***out***.println("It is not a perfect number");  **return** **false**;  }  }  **public** **static** **void** main(String a[]) {  IsPerfectNumber ipn = **new** IsPerfectNumber();  System.***out***.println("Is perfect number: " + ipn.isPerfectNumber(28));  }  } |

# **Write a program to find the sum of the first 1000 prime numbers.**

|  |
| --- |
| **public** **class** IsPrimeNumber {  **public** **static** **void** main(String args[]) {  **int** number = 2;  **int** count = 0;  **long** sum = 0;  **while** (count < 1000) {  **if** (*isPrimeNumber*(number)) {  sum += number;  count++;  }  number++;  }  System.***out***.println(sum);  }  **private** **static** **boolean** isPrimeNumber(**int** number) {  **for** (**int** i = 2; i <= number / 2; i++) {  **if** (number % i == 0) {  **return** **false**;  }  }  **return** **true**;  }  } |

# **Write a program to find two lines with max characters in descending order.**

|  |
| --- |
| **public** **class** MaxCharacters {  **public** **static** **void** main(String[] args) {  BufferedReader br = **null**;  String filePath = args[0];  **int** topList = 0;  Set<Entries> liSet = **new** TreeSet<Entries>(**new** MyComp());  **try** {  br = **new** BufferedReader(**new** FileReader(**new** File(filePath)));  String line = br.readLine();  topList = Integer.*parseInt*(line.trim());  **while** ((line = br.readLine()) != **null**) {  line = line.trim();  **if** (!"".equals(line)) {  liSet.add(**new** Entries(line.length(), line));  }  }  **int** count = 0;  **for** (Entries ent : liSet) {  System.***out***.println(ent.line);  **if** (++count == topList) {  **break**;  }  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **public** **static** **class** Entries {  Integer length;  String line;  **public** Entries(Integer l, String line) {  length = l;  **this**.line = line;  }  }  **public** **static** **class** MyComp **implements** Comparator<Entries> {  @Override  **public** **int** compare(Entries e1, Entries e2) {  **if** (e2.length > e1.length) {  **return** 1;  } **else** {  **return** -1;  }  }  }  } |

# **Write a program to find maximum repeated words from a file.**

|  |
| --- |
| **public** **class** MaxDuplicateWordCount {  **public** Map<String, Integer> getWordCount(String fileName) {  FileInputStream fis = **null**;  DataInputStream dis = **null**;  BufferedReader br = **null**;  Map<String, Integer> wordMap = **new** HashMap<String, Integer>();  **try** {  fis = **new** FileInputStream(fileName);  dis = **new** DataInputStream(fis);  br = **new** BufferedReader(**new** InputStreamReader(dis));  String line = **null**;  **while** ((line = br.readLine()) != **null**) {  StringTokenizer st = **new** StringTokenizer(line, " ");  **while** (st.hasMoreTokens()) {  String tmp = st.nextToken().toLowerCase();  **if** (wordMap.containsKey(tmp)) {  wordMap.put(tmp, wordMap.get(tmp) + 1);  } **else** {  wordMap.put(tmp, 1);  }  }  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (br != **null**)  br.close();  } **catch** (Exception ex) {  }  }  **return** wordMap;  }  **public** List<Entry<String, Integer>> sortByValue(Map<String, Integer> wordMap) {  Set<Entry<String, Integer>> set = wordMap.entrySet();  List<Entry<String, Integer>> list = **new** ArrayList<Entry<String, Integer>>(set);  Collections.*sort*(list, **new** Comparator<Map.Entry<String, Integer>>() {  **public** **int** compare(Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2) {  **return** (o2.getValue()).compareTo(o1.getValue());  }  });  **return** list;  }  **public** **static** **void** main(String a[]) {  MaxDuplicateWordCount mdc = **new** MaxDuplicateWordCount();  Map<String, Integer> wordMap = mdc.getWordCount("C:/MyTestFile.txt");  List<Entry<String, Integer>> list = mdc.sortByValue(wordMap);  **for** (Map.Entry<String, Integer> entry : list) {  System.***out***.println(entry.getKey() + " ==== " + entry.getValue());  }  }  } |

# **Write a program to get a line with max word count from the given file.**

|  |
| --- |
| **public** **class** MaxWordCountInLine {  **private** **int** currentMaxCount = 0;  **private** List<String> lines = **new** ArrayList<String>();  **public** **void** readMaxLineCount(String fileName) {  FileInputStream fis = **null**;  DataInputStream dis = **null**;  BufferedReader br = **null**;  **try** {  fis = **new** FileInputStream(fileName);  dis = **new** DataInputStream(fis);  br = **new** BufferedReader(**new** InputStreamReader(dis));  String line = **null**;  **while** ((line = br.readLine()) != **null**) {  **int** count = (line.split("\\s+")).length;  **if** (count > currentMaxCount) {  lines.clear();  lines.add(line);  currentMaxCount = count;  } **else** **if** (count == currentMaxCount) {  lines.add(line);  }  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (br != **null**)  br.close();  } **catch** (Exception ex) {  }  }  }  **public** **int** getCurrentMaxCount() {  **return** currentMaxCount;  }  **public** **void** setCurrentMaxCount(**int** currentMaxCount) {  **this**.currentMaxCount = currentMaxCount;  }  **public** List<String> getLines() {  **return** lines;  }  **public** **void** setLines(List<String> lines) {  **this**.lines = lines;  }  **public** **static** **void** main(String a[]) {  MaxWordCountInLine mdc = **new** MaxWordCountInLine();  mdc.readMaxLineCount("/Users/ngootooru/MyTestFile.txt");  System.***out***.println("Max number of words in a line is: " + mdc.getCurrentMaxCount());  System.***out***.println("Line with max word count:");  List<String> lines = mdc.getLines();  **for** (String l : lines) {  System.***out***.println(l);  }  }  } |

# **Armstrong Number**

|  |
| --- |
| **public** **class** MyArmstrongNumber {  **public** **boolean** isArmstrongNumber(**int** number) {  **int** tmp = number;  **int** noOfDigits = String.*valueOf*(number).length();  **int** sum = 0;  **int** div = 0;  **while** (tmp > 0) {  div = tmp % 10;  **int** temp = 1;  **for** (**int** i = 0; i < noOfDigits; i++) {  temp \*= div;  }  sum += temp;  tmp = tmp / 10;  }  **if** (number == sum) {  **return** **true**;  } **else** {  **return** **false**;  }  }  **public** **static** **void** main(String a[]) {  MyArmstrongNumber man = **new** MyArmstrongNumber();  System.***out***.println ("Is 371 Armstrong number? " + man.isArmstrongNumber(371));  }  } |

# **Write a program to implement ArrayList.**

|  |
| --- |
| **public** **class** MyArrayList {  **private** Object[] myStore;  **private** **int** actSize = 0;  **public** MyArrayList() {  myStore = **new** Object[10];  }  **public** Object get(**int** index) {  **if** (index < actSize) {  **return** myStore[index];  } **else** {  **throw** **new** ArrayIndexOutOfBoundsException();  }  }  **public** **void** add(Object obj) {  **if** (myStore.length - actSize <= 5) {  increaseListSize();  }  myStore[actSize++] = obj;  }  **public** Object remove(**int** index) {  **if** (index < actSize) {  Object obj = myStore[index];  myStore[index] = **null**;  **int** tmp = index;  **while** (tmp < actSize) {  myStore[tmp] = myStore[tmp + 1];  myStore[tmp + 1] = **null**;  tmp++;  }  actSize--;  **return** obj;  } **else** {  **throw** **new** ArrayIndexOutOfBoundsException();  }  }  **public** **int** size() {  **return** actSize;  }  **private** **void** increaseListSize() {  myStore = Arrays.*copyOf*(myStore, myStore.length \* 2);  System.***out***.println("\nNew length: " + myStore.length);  }  **public** **static** **void** main(String a[]) {  MyArrayList mal = **new** MyArrayList();  mal.add(**new** Integer(2));  mal.add(**new** Integer(5));  mal.add(**new** Integer(1));  mal.add(**new** Integer(23));  mal.add(**new** Integer(14));  **for** (**int** i = 0; i < mal.size(); i++) {  System.***out***.print(mal.get(i) + " ");  }  mal.add(**new** Integer(29));  System.***out***.println("Element at Index 5:" + mal.get(5));  System.***out***.println("List size: " + mal.size());  System.***out***.println("Removing element at index 2: " + mal.remove(2));  **for** (**int** i = 0; i < mal.size(); i++) {  System.***out***.print(mal.get(i) + " ");  }  }  } |

# **Binary Check**

|  |
| --- |
| **public** **class** MyBinaryCheck {  **public** **boolean** isBinaryNumber(**int** binary) {  **boolean** status = **true**;  **while** (**true**) {  **if** (binary == 0) {  **break**;  } **else** {  **int** tmp = binary % 10;  **if** (tmp > 1) {  status = **false**;  **break**;  }  binary = binary / 10;  }  }  **return** status;  }  **public** **static** **void** main(String a[]) {  MyBinaryCheck mbc = **new** MyBinaryCheck();  System.***out***.println("Is 1000111 binary? :" + mbc.isBinaryNumber(1000111));  }  } |

# **Bubble Sort: -** bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items ans swapping them is they are in wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements bubble to the top of the list, because it only uses comparisons to operate on elements, it is a comparison sort.

|  |
| --- |
| **public** **class** MyBubbleSort {  **public** **static** **void** bubble\_srt(**int** array[]) {  **int** n = array.length;  **int** k;  **for** (**int** m = n; m >= 0; m--) {  **for** (**int** i = 0; i < n - 1; i++) {  k = i + 1;  **if** (array[i] > array[k]) {  *swapNumbers*(i, k, array);  }  }  *printNumbers*(array);  }  }  **private** **static** **void** swapNumbers(**int** i, **int** j, **int**[] array) {  **int** temp;  temp = array[i];  array[i] = array[j];  array[j] = temp;  }  **private** **static** **void** printNumbers(**int**[] input) {  **for** (**int** i = 0; i < input.length; i++) {  System.***out***.print(input[i] + ", ");  }  System.***out***.println("\n");  }  **public** **static** **void** main(String[] args) {  **int**[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };  *bubble\_srt*(input);  }  } |

# **Insertion Sort:-** Insertion sort is a simple sorting algorithm that builds the final sorted array one item at a time.it is much less efficient on large lists than more advanced algorithms such a quicksort ,heapsort or merge sort . Every repetition of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list, unitl no input elements remains.the choice of which element to remove from the input is arbitrary, and can be made using almost any choice algorithm.

|  |
| --- |
| **public** **class** MyInsertionSort {  **public** **static** **void** main(String[] args) {  **int**[] input = { 4, 2, 9, 6, 23, 12, 34, 0, 1 };  *insertionSort*(input);  }  **private** **static** **void** printNumbers(**int**[] input) {  **for** (**int** i = 0; i < input.length; i++) {  System.***out***.print(input[i] + ", ");  }  System.***out***.println("\n");  }  **public** **static** **void** insertionSort(**int** array[]) {  **int** n = array.length;  **for** (**int** j = 1; j < n; j++) {  **int** key = array[j];  **int** i = j - 1;  **while** ((i > -1) && (array[i] > key)) {  array[i + 1] = array[i];  i--;  }  array[i + 1] = key;  *printNumbers*(array);  }  }  } |

# **Binary Insertion Sort**

|  |
| --- |
| **public** **class** BinaryInsertionSort {  **public** **static** **void** main(String[] args) {  **final** **int**[] arr = { 37, 23, 0, 17, 12, 72, 31, 46, 100, 88, 54 };  **new** BinaryInsertionSort().sort(arr);  **for** (**int** i = 0; i < arr.length; i++)  System.***out***.println(arr[i] + " ");  }  **public** **void** sort(**int** array[]) {  **for** (**int** i = 1; i < array.length; i++) {  **int** x = array[i];  **int** j = Math.*abs*(Arrays.*binarySearch*(array, 0, i, x) + 1);  System.*arraycopy*(array, j, array, j + 1, i - j);  array[j] = x;  }  }  } |

# **Bitonic Sort**

|  |
| --- |
| **public** **class** BitonicSort {  **void** compAndSwap(**int** a[], **int** i, **int** j, **int** dir) {  **if** ((a[i] > a[j] && dir == 1) || (a[i] < a[j] && dir == 0)) {  **int** temp = a[i];  a[i] = a[j];  a[j] = temp;  }  }  **void** bitonicMerge(**int** a[], **int** low, **int** cnt, **int** dir) {  **if** (cnt > 1) {  **int** k = cnt / 2;  **for** (**int** i = low; i < low + k; i++)  compAndSwap(a, i, i + k, dir);  bitonicMerge(a, low, k, dir);  bitonicMerge(a, low + k, k, dir);  }  }  **void** bitonicSort(**int** a[], **int** low, **int** cnt, **int** dir) {  **if** (cnt > 1) {  **int** k = cnt / 2;  bitonicSort(a, low, k, 1);  bitonicSort(a, low + k, k, 0);  bitonicMerge(a, low, cnt, dir);  }  }  **void** sort(**int** a[], **int** N, **int** up) {  bitonicSort(a, 0, N, up);  }  **static** **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  **int** a[] = { 3, 7, 4, 8, 6, 2, 1, 5 };  **int** up = 1;  BitonicSort ob = **new** BitonicSort();  ob.sort(a, a.length, up);  System.***out***.println("\nSorted array");  *printArray*(a);  }  } |

# **Bubble Sort**

|  |
| --- |
| **public** **class** BubbleSort {  **public** **static** **void** main(String[] args) {  BubbleSort ob = **new** BubbleSort();  **int** arr[] = { 64, 34, 25, 12, 22, 11, 90 };  ob.bubbleSort(arr);  System.***out***.println("Sorted array");  ob.printArray(arr);  }  **void** bubbleSort(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n - 1; i++)  **for** (**int** j = 0; j < n - i - 1; j++)  **if** (arr[j] > arr[j + 1]) {  **int** temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  } |

# **Bubble Sort1**

|  |
| --- |
| **public** **class** BubbleSort1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 64, 34, 25, 12, 22, 11, 90 };  **int** n = arr.length;  *bubbleSort*(arr, n);  System.***out***.println("Sorted array: ");  *printArray*(arr, n);  }  **static** **void** bubbleSort(**int** arr[], **int** n) {  **int** i, j, temp;  **boolean** swapped;  **for** (i = 0; i < n - 1; i++) {  swapped = **false**;  **for** (j = 0; j < n - i - 1; j++) {  **if** (arr[j] > arr[j + 1]) {  temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  swapped = **true**;  }  }  **if** (swapped == **false**)  **break**;  }  }  **static** **void** printArray(**int** arr[], **int** size) {  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  } |

# **Comb Sort**

|  |
| --- |
| **public** **class** CombSort {  **int** getNextGap(**int** gap) {  gap = (gap \* 10) / 13;  **if** (gap < 1)  **return** 1;  **return** gap;  }  **void** sort(**int** arr[]) {  **int** n = arr.length;  **int** gap = n;  **boolean** swapped = **true**;  **while** (gap != 1 || swapped == **true**) {  gap = getNextGap(gap);  swapped = **false**;  **for** (**int** i = 0; i < n - gap; i++) {  **if** (arr[i] > arr[i + gap]) {  **int** temp = arr[i];  arr[i] = arr[i + gap];  arr[i + gap] = temp;  swapped = **true**;  }  }  }  }  **public** **static** **void** main(String args[]) {  CombSort ob = **new** CombSort();  **int** arr[] = { 8, 4, 1, 56, 3, -44, 23, -6, 28, 0 };  ob.sort(arr);  System.***out***.println("sorted array");  **for** (**int** i = 0; i < arr.length; ++i)  System.***out***.print(arr[i] + " ");  }  } |

# **Counting Sort**

|  |
| --- |
| **public** **class** CountingSort {  **void** sort(**char** arr[]) {  **int** n = arr.length;  **char** output[] = **new** **char**[n];  **int** count[] = **new** **int**[256];  **for** (**int** i = 0; i < 256; ++i)  count[i] = 0;  **for** (**int** i = 0; i < n; ++i)  ++count[arr[i]];  **for** (**int** i = 1; i <= 255; ++i)  count[i] += count[i - 1];  **for** (**int** i = 0; i < n; ++i) {  output[count[arr[i]] - 1] = arr[i];  --count[arr[i]];  }  **for** (**int** i = 0; i < n; ++i)  arr[i] = output[i];  }  **public** **static** **void** main(String args[]) {  CountingSort ob = **new** CountingSort();  **char** arr[] = { 'g', 'e', 'e', 'k', 's', 'f', 'o', 'r', 'g', 'e', 'e', 'k', 's' };  ob.sort(arr);  System.***out***.print("Sorted character array is ");  **for** (**int** i = 0; i < arr.length; ++i)  System.***out***.print(arr[i]);  }  } |

# **Cycle Sort**

|  |
| --- |
| /\*Time Complexity : O(n2) Worst Case : O(n2) Average Case: O(n2) Best Case : O(n2)  **public** **class** CycleSort {  **public** **static** **void** cycleSort(**int** arr[], **int** n) {  **int** writes = 0;  **for** (**int** cycle\_start = 0; cycle\_start <= n - 2; cycle\_start++) {  **int** item = arr[cycle\_start];  **int** pos = cycle\_start;  **for** (**int** i = cycle\_start + 1; i < n; i++)  **if** (arr[i] < item)  pos++;  **if** (pos == cycle\_start)  **continue**;  **while** (item == arr[pos])  pos += 1;  **if** (pos != cycle\_start) {  **int** temp = item;  item = arr[pos];  arr[pos] = temp;  writes++;  }  **while** (pos != cycle\_start) {  pos = cycle\_start;  **for** (**int** i = cycle\_start + 1; i < n; i++)  **if** (arr[i] < item)  pos += 1;  **while** (item == arr[pos])  pos += 1;  **if** (item != arr[pos]) {  **int** temp = item;  item = arr[pos];  arr[pos] = temp;  writes++;  }  }  }  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 1, 8, 3, 9, 10, 10, 2, 4 };  **int** n = arr.length;  *cycleSort*(arr, n);  System.***out***.println("After sort : ");  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  } |

# **Gnome Sort**

|  |
| --- |
| **public** **class** GnomeSort {  **static** **void** gnomeSort(**int** arr[], **int** n) {  **int** index = 0;  **while** (index < n) {  **if** (index == 0)  index++;  **if** (arr[index] >= arr[index - 1])  index++;  **else** {  **int** temp = 0;  temp = arr[index];  arr[index] = arr[index - 1];  arr[index - 1] = temp;  index--;  }  }  **return**;  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 34, 2, 10, -9 };  *gnomeSort*(arr, arr.length);  System.***out***.print("Sorted sequence after applying Gnome sort: ");  System.***out***.println(Arrays.*toString*(arr));  }  } |

# **Heap Sort**

|  |
| --- |
| **public** **class** HeapSort {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 11, 13, 5, 6, 7 };  **int** n = arr.length;  HeapSort ob = **new** HeapSort();  ob.sort(arr);  System.***out***.println("Sorted array is");  *printArray*(arr);  }  **public** **void** sort(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = n / 2 - 1; i >= 0; i--)  heapify(arr, n, i);  **for** (**int** i = n - 1; i >= 0; i--) {  **int** temp = arr[0];  arr[0] = arr[i];  arr[i] = temp;  heapify(arr, i, 0);  }  }  **void** heapify(**int** arr[], **int** n, **int** i) {  **int** largest = i;  **int** l = 2 \* i + 1;  **int** r = 2 \* i + 2;  **if** (l < n && arr[l] > arr[largest])  largest = l;  **if** (r < n && arr[r] > arr[largest])  largest = r;  **if** (largest != i) {  **int** swap = arr[i];  arr[i] = arr[largest];  arr[largest] = swap;  heapify(arr, n, largest);  }  }  **static** **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  } |

# **Iterative Merge Sort**

|  |
| --- |
| **public** **class** IterativeMergeSort {  **public** **static** **void** mergeSort(**int**[] array) {  **if** (array == **null**) {  **return**;  }  **if** (array.length > 1) {  **int** mid = array.length / 2;  **int**[] left = **new** **int**[mid];  **for** (**int** i = 0; i < mid; i++) {  left[i] = array[i];  }  **int**[] right = **new** **int**[array.length - mid];  **for** (**int** i = mid; i < array.length; i++) {  right[i - mid] = array[i];  }  *mergeSort*(left);  *mergeSort*(right);  **int** i = 0;  **int** j = 0;  **int** k = 0;  **while** (i < left.length && j < right.length) {  **if** (left[i] < right[j]) {  array[k] = left[i];  i++;  } **else** {  array[k] = right[j];  j++;  }  k++;  }  **while** (i < left.length) {  array[k] = left[i];  i++;  k++;  }  **while** (j < right.length) {  array[k] = right[j];  j++;  k++;  }  }  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 11, 13, 5, 6, 7 };  **int** i = 0;  System.***out***.println("Given array is");  **for** (i = 0; i < arr.length; i++)  System.***out***.print(arr[i] + " ");  *mergeSort*(arr);  System.***out***.println("\n");  System.***out***.println("Sorted array is");  **for** (i = 0; i < arr.length; i++)  System.***out***.print(arr[i] + " ");  }  } |

# **Iterative Merge Sort1**

|  |
| --- |
| **public** **class** IterativeMergeSort1 {  **static** **void** mergeSort(**int** arr[], **int** n) {  **int** curr\_size;  **int** left\_start;  **for** (curr\_size = 1; curr\_size <= n - 1; curr\_size = 2 \* curr\_size) {  **for** (left\_start = 0; left\_start < n - 1; left\_start += 2 \* curr\_size) {  **int** mid = left\_start + curr\_size - 1;  **int** right\_end = Math.*min*(left\_start + 2 \* curr\_size - 1, n - 1);  *merge*(arr, left\_start, mid, right\_end);  }  }  }  **static** **void** merge(**int** arr[], **int** l, **int** m, **int** r) {  **int** i, j, k;  **int** n1 = m - l + 1;  **int** n2 = r - m;  **int** L[] = **new** **int**[n1];  **int** R[] = **new** **int**[n2];  **for** (i = 0; i < n1; i++)  L[i] = arr[l + i];  **for** (j = 0; j < n2; j++)  R[j] = arr[m + 1 + j];  i = 0;  j = 0;  k = l;  **while** (i < n1 && j < n2) {  **if** (L[i] <= R[j]) {  arr[k] = L[i];  i++;  } **else** {  arr[k] = R[j];  j++;  }  k++;  }  **while** (i < n1) {  arr[k] = L[i];  i++;  k++;  }  **while** (j < n2) {  arr[k] = R[j];  j++;  k++;  }  }  **static** **void** printArray(**int** A[], **int** size) {  **int** i;  **for** (i = 0; i < size; i++)  System.***out***.printf("%d ", A[i]);  System.***out***.printf("\n");  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 11, 13, 5, 6, 7 };  **int** n = arr.length;  System.***out***.printf("Given array is \n");  *printArray*(arr, n);  *mergeSort*(arr, n);  System.***out***.printf("\nSorted array is \n");  *printArray*(arr, n);  }  } |

# **Merge Sort**

|  |
| --- |
| **public** **class** MergeSort {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 11, 13, 5, 6, 7 };  System.***out***.println("Given Array");  *printArray*(arr);  MergeSort ob = **new** MergeSort();  ob.sort(arr, 0, arr.length - 1);  System.***out***.println("\nSorted array");  *printArray*(arr);  }  **void** merge(**int** arr[], **int** l, **int** m, **int** r) {  **int** n1 = m - l + 1;  **int** n2 = r - m;  **int** L[] = **new** **int**[n1];  **int** R[] = **new** **int**[n2];  **for** (**int** i = 0; i < n1; ++i)  L[i] = arr[l + i];  **for** (**int** j = 0; j < n2; ++j)  R[j] = arr[m + 1 + j];  **int** i = 0, j = 0;  **int** k = l;  **while** (i < n1 && j < n2) {  **if** (L[i] <= R[j]) {  arr[k] = L[i];  i++;  } **else** {  arr[k] = R[j];  j++;  }  k++;  }  **while** (i < n1) {  arr[k] = L[i];  i++;  k++;  }  **while** (j < n2) {  arr[k] = R[j];  j++;  k++;  }  }  **void** sort(**int** arr[], **int** l, **int** r) {  **if** (l < r) {  **int** m = (l + r) / 2;  sort(arr, l, m);  sort(arr, m + 1, r);  merge(arr, l, m, r);  }  }  **static** **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  } |

# **Merge Sort 3 Way**

|  |
| --- |
| Input : 45, -2, -45, 78, 30, -42, 10, 19 , 73, 93 Output : -45 -42 -2 10 19 30 45 73 78 93  **public** **class** MergeSort3Way {  **public** **static** **void** mergeSort3Way(Integer[] gArray) {  **if** (gArray == **null**)  **return**;  Integer[] fArray = **new** Integer[gArray.length];  **for** (**int** i = 0; i < fArray.length; i++)  fArray[i] = gArray[i];  *mergeSort3WayRec*(fArray, 0, gArray.length, gArray);  **for** (**int** i = 0; i < fArray.length; i++)  gArray[i] = fArray[i];  }  **public** **static** **void** mergeSort3WayRec(Integer[] gArray, **int** low, **int** high, Integer[] destArray) {  **if** (high - low < 2)  **return**;  **int** mid1 = low + ((high - low) / 3);  **int** mid2 = low + 2 \* ((high - low) / 3) + 1;  *mergeSort3WayRec*(destArray, low, mid1, gArray);  *mergeSort3WayRec*(destArray, mid1, mid2, gArray);  *mergeSort3WayRec*(destArray, mid2, high, gArray);  *merge*(destArray, low, mid1, mid2, high, gArray);  }  **public** **static** **void** merge(Integer[] gArray, **int** low, **int** mid1, **int** mid2, **int** high, Integer[] destArray) {  **int** i = low, j = mid1, k = mid2, l = low;  **while** ((i < mid1) && (j < mid2) && (k < high)) {  **if** (gArray[i].compareTo(gArray[j]) < 0) {  **if** (gArray[i].compareTo(gArray[k]) < 0)  destArray[l++] = gArray[i++];  **else**  destArray[l++] = gArray[k++];  } **else** {  **if** (gArray[j].compareTo(gArray[k]) < 0)  destArray[l++] = gArray[j++];  **else**  destArray[l++] = gArray[k++];  }  }  **while** ((i < mid1) && (j < mid2)) {  **if** (gArray[i].compareTo(gArray[j]) < 0)  destArray[l++] = gArray[i++];  **else**  destArray[l++] = gArray[j++];  }  **while** ((j < mid2) && (k < high)) {  **if** (gArray[j].compareTo(gArray[k]) < 0)  destArray[l++] = gArray[j++];  **else**  destArray[l++] = gArray[k++];  }  **while** ((i < mid1) && (k < high)) {  **if** (gArray[i].compareTo(gArray[k]) < 0)  destArray[l++] = gArray[i++];  **else**  destArray[l++] = gArray[k++];  }  **while** (i < mid1)  destArray[l++] = gArray[i++];  **while** (j < mid2)  destArray[l++] = gArray[j++];  **while** (k < high)  destArray[l++] = gArray[k++];  }  **public** **static** **void** main(String args[]) {  Integer[] data = **new** Integer[] { 45, -2, -45, 78, 30, -42, 10, 19, 73, 93 };  *mergeSort3Way*(data);  System.***out***.println("After 3 way merge sort: ");  **for** (**int** i = 0; i < data.length; i++)  System.***out***.print(data[i] + " ");  }  } |

# **Iterative Quick Sort**

|  |
| --- |
| **public** **class** IterativeQuickSort {  **public** **static** **void** main(String[] args) {  **int** n = 5;  **int** arr[] = { 4, 2, 6, 9, 2 };  *qSort*(arr, 0, n - 1);  **for** (**int** i = 0; i < n; i++) {  System.***out***.print(arr[i] + " ");  }  }  **static** **int** partition(**int** arr[], **int** low, **int** high) {  **int** pivot = arr[high];  **int** i = (low - 1);  **for** (**int** j = low; j <= high - 1; j++) {  **if** (arr[j] <= pivot) {  i++;  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  **int** temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  **return** i + 1;  }  **static** **void** qSort(**int** arr[], **int** low, **int** high) {  **if** (low < high) {  **int** pi = *partition*(arr, low, high);  *qSort*(arr, low, pi - 1);  *qSort*(arr, pi + 1, high);  }  }  } |

# **Iterative Quick Sort1**

|  |
| --- |
| **public** **class** IterativeQuickSort1 {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 4, 3, 5, 2, 1, 3, 2, 3 };  **int** n = 8;  *quickSortIterative*(arr, 0, n - 1);  **for** (**int** i = 0; i < n; i++) {  System.***out***.print(arr[i] + " ");  }  }  **static** **int** partition(**int** arr[], **int** low, **int** high) {  **int** pivot = arr[high];  **int** i = (low - 1);  **for** (**int** j = low; j <= high - 1; j++) {  **if** (arr[j] <= pivot) {  i++;  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  **int** temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  **return** i + 1;  }  **static** **void** quickSortIterative(**int** arr[], **int** l, **int** h) {  **int**[] stack = **new** **int**[h - l + 1];  **int** top = -1;  stack[++top] = l;  stack[++top] = h;  **while** (top >= 0) {  h = stack[top--];  l = stack[top--];  **int** p = *partition*(arr, l, h);  **if** (p - 1 > l) {  stack[++top] = l;  stack[++top] = p - 1;  }  **if** (p + 1 < h) {  stack[++top] = p + 1;  stack[++top] = h;  }  }  }  } |

# **Quick Sort**

|  |
| --- |
| **public** **class** QuickSort {  **public** **static** **void** main(String[] args) {  **int** arr[] = { 10, 7, 8, 9, 1, 5 };  **int** n = arr.length;  QuickSort ob = **new** QuickSort();  ob.sort(arr, 0, n - 1);  System.***out***.println("sorted array");  *printArray*(arr);  }  **int** partition(**int** arr[], **int** low, **int** high) {  **int** pivot = arr[high];  **int** i = (low - 1);  **for** (**int** j = low; j < high; j++) {  **if** (arr[j] <= pivot) {  i++;  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  **int** temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  **return** i + 1;  }  **void** sort(**int** arr[], **int** low, **int** high) {  **if** (low < high) {  **int** pi = partition(arr, low, high);  sort(arr, low, pi - 1);  sort(arr, pi + 1, high);  }  }  **static** **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  } |

# **QuickSort\_using\_Doubly\_LinkedList**

|  |
| --- |
| **public** **class** QuickSort\_using\_Doubly\_LinkedList {  Node head;  **static** **class** Node {  **private** **int** data;  **private** Node next;  **private** Node prev;  Node(**int** d) {  data = d;  next = **null**;  prev = **null**;  }  }  Node lastNode(Node node) {  **while** (node.next != **null**)  node = node.next;  **return** node;  }  Node partition(Node l, Node h) {  **int** x = h.data;  Node i = l.prev;  **for** (Node j = l; j != h; j = j.next) {  **if** (j.data <= x) {  i = (i == **null**) ? l : i.next;  **int** temp = i.data;  i.data = j.data;  j.data = temp;  }  }  i = (i == **null**) ? l : i.next;  **int** temp = i.data;  i.data = h.data;  h.data = temp;  **return** i;  }  **void** \_quickSort(Node l, Node h) {  **if** (h != **null** && l != h && l != h.next) {  Node temp = partition(l, h);  \_quickSort(l, temp.prev);  \_quickSort(temp.next, h);  }  }  **public** **void** quickSort(Node node) {  Node head = lastNode(node);  \_quickSort(node, head);  }  **public** **void** printList(Node head) {  **while** (head != **null**) {  System.***out***.print(head.data + " ");  head = head.next;  }  }  **void** push(**int** new\_Data) {  Node new\_Node = **new** Node(new\_Data);  **if** (head == **null**) {  head = new\_Node;  **return**;  }  new\_Node.next = head;  head.prev = new\_Node;  new\_Node.prev = **null**;  head = new\_Node;  }  **public** **static** **void** main(String[] args) {  QuickSort\_using\_Doubly\_LinkedList list = **new** QuickSort\_using\_Doubly\_LinkedList();  list.push(5);  list.push(20);  list.push(4);  list.push(3);  list.push(30);  System.***out***.println("Linked List before sorting ");  list.printList(list.head);  System.***out***.println("\nLinked List after sorting");  list.quickSort(list.head);  list.printList(list.head);  }  } |

# **Pancake Sort**

|  |
| --- |
| **public** **class** PancakeSort {  **static** **void** flip(**int** arr[], **int** i) {  **int** temp, start = 0;  **while** (start < i) {  temp = arr[start];  arr[start] = arr[i];  arr[i] = temp;  start++;  i--;  }  }  **static** **int** findMax(**int** arr[], **int** n) {  **int** mi, i;  **for** (mi = 0, i = 0; i < n; ++i)  **if** (arr[i] > arr[mi])  mi = i;  **return** mi;  }  **static** **int** pancakeSort(**int** arr[], **int** n) {  **for** (**int** curr\_size = n; curr\_size > 1; --curr\_size) {  **int** mi = *findMax*(arr, curr\_size);  **if** (mi != curr\_size - 1) {  *flip*(arr, mi);  *flip*(arr, curr\_size - 1);  }  }  **return** 0;  }  **static** **void** printArray(**int** arr[], **int** arr\_size) {  **for** (**int** i = 0; i < arr\_size; i++)  System.***out***.print(arr[i] + " ");  System.***out***.println("");  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 23, 10, 20, 11, 12, 6, 7 };  **int** n = arr.length;  *pancakeSort*(arr, n);  System.***out***.println("Sorted Array: ");  *printArray*(arr, n);  }  } |

# **Pigeonhole Sort**

|  |
| --- |
| **public** **class** PigeonholeSort {  **public** **static** **void** pigeonhole\_sort(**int** arr[], **int** n) {  **int** min = arr[0];  **int** max = arr[0];  **int** range, i, j, index;  **for** (**int** a = 0; a < n; a++) {  **if** (arr[a] > max)  max = arr[a];  **if** (arr[a] < min)  min = arr[a];  }  range = max - min + 1;  **int**[] phole = **new** **int**[range];  Arrays.*fill*(phole, 0);  **for** (i = 0; i < n; i++)  phole[arr[i] - min]++;  index = 0;  **for** (j = 0; j < range; j++)  **while** (phole[j]-- > 0)  arr[index++] = j + min;  }  **public** **static** **void** main(String[] args) {  PigeonholeSort sort = **new** PigeonholeSort();  **int**[] arr = { 8, 3, 2, 7, 4, 6, 8 };  System.***out***.print("Sorted order is : ");  sort.*pigeonhole\_sort*(arr, arr.length);  **for** (**int** i = 0; i < arr.length; i++)  System.***out***.print(arr[i] + " ");  }  } |

# **Radix Sort**

|  |
| --- |
| **public** **class** RadixSort {  **static** **int** getMax(**int** arr[], **int** n) {  **int** mx = arr[0];  **for** (**int** i = 1; i < n; i++)  **if** (arr[i] > mx)  mx = arr[i];  **return** mx;  }  **static** **void** countSort(**int** arr[], **int** n, **int** exp) {  **int** output[] = **new** **int**[n];  **int** i;  **int** count[] = **new** **int**[10];  Arrays.*fill*(count, 0);  **for** (i = 0; i < n; i++)  count[(arr[i] / exp) % 10]++;  **for** (i = 1; i < 10; i++)  count[i] += count[i - 1];  **for** (i = n - 1; i >= 0; i--) {  output[count[(arr[i] / exp) % 10] - 1] = arr[i];  count[(arr[i] / exp) % 10]--;  }  **for** (i = 0; i < n; i++)  arr[i] = output[i];  }  **static** **void** radixsort(**int** arr[], **int** n) {  **int** m = *getMax*(arr, n);  **for** (**int** exp = 1; m / exp > 0; exp \*= 10)  *countSort*(arr, n, exp);  }  **static** **void** print(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++)  System.***out***.print(arr[i] + " ");  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };  **int** n = arr.length;  *radixsort*(arr, n);  *print*(arr, n);  }  } |

# **Recursive Bubble Sort**

|  |
| --- |
| **public** **class** RecursiveBubbleSort {  **static** **void** bubbleSort(**int** arr[], **int** n) {  **if** (n == 1)  **return**;  **for** (**int** i = 0; i < n - 1; i++)  **if** (arr[i] > arr[i + 1]) {  **int** temp = arr[i];  arr[i] = arr[i + 1];  arr[i + 1] = temp;  }  *bubbleSort*(arr, n - 1);  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 64, 34, 25, 12, 22, 11, 90 };  *bubbleSort*(arr, arr.length);  System.***out***.println("Sorted array : ");  System.***out***.println(Arrays.*toString*(arr));  }  } |

# **Recursive Insertion Sort**

|  |
| --- |
| **public** **class** RecursiveInsertionSort {  **static** **void** insertionSortRecursive(**int** arr[], **int** n) {  **if** (n <= 1) **return**;  *insertionSortRecursive*(arr, n - 1);  **int** last = arr[n - 1];  **int** j = n - 2;  **while** (j >= 0 && arr[j] > last) {  arr[j + 1] = arr[j];  j--;  }  arr[j + 1] = last;  }  **public** **static** **void** main(String[] args) {  **int** arr[] = { 12, 11, 13, 5, 6 };  *insertionSortRecursive*(arr, arr.length);  System.***out***.println(Arrays.*toString*(arr));  }  } |

# **Selection Sort**

|  |
| --- |
| **public** **class** SelectionSort {  **void** sort(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n - 1; i++) {  **int** min\_idx = i;  **for** (**int** j = i + 1; j < n; j++)  **if** (arr[j] < arr[min\_idx])  min\_idx = j;  **int** temp = arr[min\_idx];  arr[min\_idx] = arr[i];  arr[i] = temp;  }  }  **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  SelectionSort ob = **new** SelectionSort();  **int** arr[] = { 64, 25, 12, 22, 11 };  ob.sort(arr);  System.***out***.println("Sorted array");  ob.printArray(arr);  }  } |

# **Shell Sort**

|  |
| --- |
| **public** **class** ShellSort {  **static** **void** printArray(**int** arr[]) {  **int** n = arr.length;  **for** (**int** i = 0; i < n; ++i)  System.***out***.print(arr[i] + " ");  System.***out***.println();  }  **int** sort(**int** arr[]) {  **int** n = arr.length;  **for** (**int** gap = n / 2; gap > 0; gap /= 2) {  **for** (**int** i = gap; i < n; i += 1) {  **int** temp = arr[i];  **int** j;  **for** (j = i; j >= gap && arr[j - gap] > temp; j -= gap)  arr[j] = arr[j - gap];  arr[j] = temp;  }  }  **return** 0;  }  **public** **static** **void** main(String args[]) {  **int** arr[] = { 12, 34, 54, 2, 3 };  System.***out***.println("Array before sorting");  *printArray*(arr);  ShellSort ob = **new** ShellSort();  ob.sort(arr);  System.***out***.println("Array after sorting");  *printArray*(arr);  }  } |

# **How to sort a Stack using a temporary Stack?** You have a stack with full of integers. Sorting it in the ascending order using another temporary rray by using all stack functionality.

|  |
| --- |
| **public** **class** StackSort {  **public** **static** Stack<Integer> sortStack(Stack<Integer> input) {  Stack<Integer> tmpStack = **new** Stack<Integer>();  System.***out***.println("=============== debug logs ================");  **while** (!input.isEmpty()) {  **int** tmp = input.pop();  System.***out***.println("Element taken out: " + tmp);  **while** (!tmpStack.isEmpty() && tmpStack.peek() > tmp) {  input.push(tmpStack.pop());  }  tmpStack.push(tmp);  System.***out***.println("input: " + input);  System.***out***.println("tmpStack: " + tmpStack);  }  System.***out***.println("=============== debug logs ended ================");  **return** tmpStack;  }  **public** **static** **void** main(String a[]) {  Stack<Integer> input = **new** Stack<Integer>();  input.add(34);  input.add(3);  input.add(31);  input.add(98);  input.add(92);  input.add(23);  System.***out***.println("input: " + input);  System.***out***.println("final sorted list: " + *sortStack*(input));  }  } |

# **Deadlock**

|  |
| --- |
| **public** **class** MyDeadlock {  String str1 = "Java";  String str2 = "UNIX";  Thread trd1 = **new** Thread("My Thread 1") {  **public** **void** run() {  **while** (**true**) {  **synchronized** (str1) {  **synchronized** (str2) {  System.***out***.println(str1 + str2);  }  }  }  }  };  Thread trd2 = **new** Thread("My Thread 2") {  **public** **void** run() {  **while** (**true**) {  **synchronized** (str2) {  **synchronized** (str1) {  System.***out***.println(str2 + str1);  }  }  }  }  };  **public** **static** **void** main(String a[]) {  MyDeadlock mdl = **new** MyDeadlock();  mdl.trd1.start();  mdl.trd2.start();  }  } |

# **How to get distinct elements from an array by avoiding duplicate elements?**

|  |
| --- |
| **public** **class** MyDisticntElements {  **public** **static** **void** printDistinctElements(**int**[] arr) {  **for** (**int** i = 0; i < arr.length; i++) {  **boolean** isDistinct = **false**;  **for** (**int** j = 0; j < i; j++) {  **if** (arr[i] == arr[j]) {  isDistinct = **true**;  **break**;  }  }  **if** (!isDistinct) {  System.***out***.print(arr[i] + " ");  }  }  }  **public** **static** **void** main(String a[]) {  **int**[] nums = { 5, 2, 7, 2, 4, 7, 8, 2, 3 };  MyDisticntElements.*printDistinctElements*(nums);  }  } |

# **Distinct File Words**

|  |
| --- |
| **public** **class** MyDistinctFileWords {  **public** List<String> getDistinctWordList(String fileName) {  FileInputStream fis = **null**;  DataInputStream dis = **null**;  BufferedReader br = **null**;  List<String> wordList = **new** ArrayList<String>();  **try** {  fis = **new** FileInputStream(fileName);  dis = **new** DataInputStream(fis);  br = **new** BufferedReader(**new** InputStreamReader(dis));  String line = **null**;  **while** ((line = br.readLine()) != **null**) {  StringTokenizer st = **new** StringTokenizer(line, " ,.;:\"");  **while** (st.hasMoreTokens()) {  String tmp = st.nextToken().toLowerCase();  **if** (!wordList.contains(tmp)) {  wordList.add(tmp);  }  }  }  } **catch** (FileNotFoundException e) {  e.printStackTrace();  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (br != **null**)  br.close();  } **catch** (Exception ex) {  }  }  **return** wordList;  }  **public** **static** **void** main(String a[]) {  MyDistinctFileWords distFw = **new** MyDistinctFileWords();  List<String> wordList = distFw.getDistinctWordList("C:/sample.txt");  **for** (String str : wordList) {  System.***out***.println(str);  }  }  } |

# **Write a program to remove duplicates from sorted array.**

|  |
| --- |
| **public** **class** MyDuplicateElements {  **public** **static** **int**[] removeDuplicates(**int**[] input) {  **int** j = 0;  **int** i = 1;  **if** (input.length < 2) {  **return** input;  }  **while** (i < input.length) {  **if** (input[i] == input[j]) {  i++;  } **else** {  input[++j] = input[i++];  }  }  **int**[] output = **new** **int**[j + 1];  **for** (**int** k = 0; k < output.length; k++) {  output[k] = input[k];  }  **return** output;  }  **public** **static** **void** main(String a[]) {  **int**[] input1 = { 2, 3, 6, 6, 8, 9, 10, 10, 10, 12, 12 };  **int**[] output = *removeDuplicates*(input1);  **for** (**int** i : output) {  System.***out***.print(i + " ");  }  }  } |

# **Fibonacci**

|  |
| --- |
| **public** **class** MyFibonacci {  **public** **static** **void** main(String[] args) {  **int** febCount = 15;  **int**[] feb = **new** **int**[febCount];  feb[0] = 0;  feb[1] = 1;  **for** (**int** i = 2; i < febCount; i++) {  feb[i] = feb[i - 1] + feb[i - 2];  }  **for** (**int** i = 0; i < febCount; i++) {  System.***out***.print(feb[i] + " ");  }  }  } |

# **Write a program to implement hashcode and equals.**

|  |
| --- |
| **public** **class** MyHashcodeImpl {  **public** **static** **void** main(String a[]) {  HashMap<Price, String> hm = **new** HashMap<Price, String>();  hm.put(**new** Price("Banana", 20), "Banana");  hm.put(**new** Price("Apple", 40), "Apple");  hm.put(**new** Price("Orange", 30), "Orange");  // creating new object to use as key to get value  Price key = **new** Price("Banana", 20);  System.***out***.println("Hashcode of the key: " + key.hashCode());  System.***out***.println("Value from map: " + hm.get(key));  }  }  **class** Price {  **private** String item;  **private** **int** price;  **public** Price(String itm, **int** pr) {  **this**.item = itm;  **this**.price = pr;  }  **public** **int** hashCode() {  System.***out***.println("In hashcode");  **int** hashcode = 0;  hashcode = price \* 20;  hashcode += item.hashCode();  **return** hashcode;  }  **public** **boolean** equals(Object obj) {  System.***out***.println("In equals");  **if** (obj **instanceof** Price) {  Price pp = (Price) obj;  **return** (pp.item.equals(**this**.item) && pp.price == **this**.price);  } **else** {  **return** **false**;  }  }  **public** String getItem() {  **return** item;  }  **public** **void** setItem(String item) {  **this**.item = item;  }  **public** **int** getPrice() {  **return** price;  }  **public** **void** setPrice(**int** price) {  **this**.price = price;  }  **public** String toString() {  **return** "item: " + item + " price: " + price;  }  } |

# **Find longest substring without repeating characters.**

|  |
| --- |
| **public** **class** MyLongestSubstr {  **private** Set<String> subStrList = **new** HashSet<String>();  **private** **int** finalSubStrSize = 0;  **public** Set<String> getLongestSubstr(String input) {  subStrList.clear();  finalSubStrSize = 0;  **boolean**[] flag = **new** **boolean**[256];  **int** j = 0;  **char**[] inputCharArr = input.toCharArray();  **for** (**int** i = 0; i < inputCharArr.length; i++) {  **char** c = inputCharArr[i];  **if** (flag[c]) {  extractSubString(inputCharArr, j, i);  **for** (**int** k = j; k < i; k++) {  **if** (inputCharArr[k] == c) {  j = k + 1;  **break**;  }  flag[inputCharArr[k]] = **false**;  }  } **else** {  flag[c] = **true**;  }  }  extractSubString(inputCharArr, j, inputCharArr.length);  **return** subStrList;  }  **private** String extractSubString(**char**[] inputArr, **int** start, **int** end) {  StringBuilder sb = **new** StringBuilder();  **for** (**int** i = start; i < end; i++) {  sb.append(inputArr[i]);  }  String subStr = sb.toString();  **if** (subStr.length() > finalSubStrSize) {  finalSubStrSize = subStr.length();  subStrList.clear();  subStrList.add(subStr);  } **else** **if** (subStr.length() == finalSubStrSize) {  subStrList.add(subStr);  }  **return** sb.toString();  }  **public** **static** **void** main(String a[]) {  MyLongestSubstr mls = **new** MyLongestSubstr();  System.***out***.println(mls.getLongestSubstr("java2novice"));  }  } |

# **Write a program to find sum of each digit in the given number using recursion.**

|  |
| --- |
| **public** **class** MyNumberSumRec {  **int** sum = 0;  **public** **int** getNumberSum(**int** number) {  **if** (number == 0) {  **return** sum;  } **else** {  sum += (number % 10);  getNumberSum(number / 10);  }  **return** sum;  }  **public** **static** **void** main(String a[]) {  MyNumberSumRec mns = **new** MyNumberSumRec();  System.***out***.println("Sum is: " + mns.getNumberSum(223));  }  } |

# **Prime Num Check**

|  |
| --- |
| **public** **class** MyPrimeNumCheck {  **public** **boolean** isPrimeNumber(**int** number) {  **for** (**int** i = 2; i <= number / 2; i++) {  **if** (number % i == 0) {  **return** **false**;  }  }  **return** **true**;  }  **public** **static** **void** main(String a[]) {  MyPrimeNumCheck mpc = **new** MyPrimeNumCheck();  System.***out***.println ("Is 17 prime number? " + mpc.isPrimeNumber(17));  }  } |

# **Write a singleton class:-**Singleton class means you can create only one object for the given class. You cab create a singleton class by making its constructor as private, so that you can restrict the creation of the object. Provide a static method to get instance of the object, where in you can handle the object creation inside the class only.

|  |
| --- |
| In this example we are creating object by using static block.  **public** **class** MySingleton {  **private** **static** MySingleton *myObj*;  **static** {  *myObj* = **new** MySingleton();  }  **private** MySingleton() {  }  **public** **static** MySingleton getInstance() {  **return** *myObj*;  }  **public** **void** testMe() {  System.***out***.println ("Hey.... it is working!!!");  }  **public** **static** **void** main(String a[]) {  MySingleton ms = *getInstance*();  ms.testMe();  }  } |

# **Write a program to convert string to number without using Integer.parseInt () method.**

|  |
| --- |
| **public** **class** MyStringToNumber {  **public** **static** **int** convert\_String\_To\_Number(String numStr) {  **char** ch[] = numStr.toCharArray();  **int** sum = 0;  **int** zeroAscii = (**int**) '0';  **for** (**char** c : ch) {  **int** tmpAscii = (**int**) c;  sum = (sum \* 10) + (tmpAscii - zeroAscii);  }  **return** sum;  }  **public** **static** **void** main(String a[]) {  System.***out***.println("\"3256\" == " + *convert\_String\_To\_Number*("3256"));  }  } |

# **How to swap two numbers without using temporary variable?**

|  |
| --- |
| **public** **class** MySwapingTwoNumbers {  **public** **static** **void** main(String[] args) {  **int** x = 10;  **int** y = 20;  System.***out***.println("Before swap:");  System.***out***.println("x value: " + x);  System.***out***.println("y value: " + y);  x = x + y;  y = x - y;  x = x - y;  System.***out***.println("After swap:");  System.***out***.println("x value: " + x);  System.***out***.println("y value: " + y);  }  } |

# **Number Reverse**

|  |
| --- |
| **public** **class** NumberReverse {  **public** **int** reverseNumber(**int** number) {  **int** reverse = 0;  **while** (number != 0) {  reverse = (reverse \* 10) + (number % 10);  number = number / 10;  }  **return** reverse;  }  **public** **static** **void** main(String a[]) {  NumberReverse nr = **new** NumberReverse();  System.***out***.println("Result: " + nr.reverseNumber(17868));  }  } |

# **Write a program to sort a map by value.**

|  |
| --- |
| **public** **class** OrderByValue {  **public** **static** **void** main(String[] args) {  Map<String, Integer> map = **new** HashMap<String, Integer>();  map.put("java", 20);  map.put("C++", 45);  map.put("Java2Novice", 2);  map.put("Unix", 67);  map.put("MAC", 26);  map.put("Why this kolavari", 93);  Set<Entry<String, Integer>> set = map.entrySet();  List<Entry<String, Integer>> list = **new** ArrayList<Entry<String, Integer>>(set);  Collections.*sort*(list, **new** Comparator<Map.Entry<String, Integer>>() {  **public** **int** compare(Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2) {  **return** (o2.getValue()).compareTo(o1.getValue());  }  });  **for** (Map.Entry<String, Integer> entry : list) {  System.***out***.println(entry.getKey() + " ==== " + entry.getValue());  }  }  } |

# **Reverse Singly Linked List**

|  |
| --- |
| **class** Node<T> **implements** Comparable<T> {  **private** T value;  **private** Node<T> nextRef;  **public** T getValue() {  **return** value;  }  **public** **void** setValue(T value) {  **this**.value = value;  }  **public** Node<T> getNextRef() {  **return** nextRef;  }  **public** **void** setNextRef(Node<T> ref) {  **this**.nextRef = ref;  }  @Override  **public** **int** compareTo(T arg) {  **if** (arg == **this**.value) {  **return** 0;  } **else** {  **return** 1;  }  }  }  **public** **class** ReverseSinglyLinkedList<T> {  **public** **static** **void** main(String[] args) {  ReverseSinglyLinkedList<Integer> sl = **new** ReverseSinglyLinkedList<Integer>();  sl.add(3);  sl.add(32);  sl.add(54);  sl.add(89);  System.***out***.println();  sl.traverse();  System.***out***.println();  sl.reverse();  sl.traverse();  }  **private** Node<T> head;  **public** **void** add(T element) {  Node<T> nd = **new** Node<T>();  nd.setValue(element);  System.***out***.println("Adding: " + element);  Node<T> tmp = head;  **while** (**true**) {  **if** (tmp == **null**) {  // since there is only one element, both head and tail points to the same object.  head = nd;  **break**;  } **else** **if** (tmp.getNextRef() == **null**) {  tmp.setNextRef(nd);  **break**;  } **else** {  tmp = tmp.getNextRef();  }  }  }  **public** **void** traverse() {  Node<T> tmp = head;  **while** (**true**) {  **if** (tmp == **null**) {  **break**;  }  System.***out***.print(tmp.getValue() + "\t");  tmp = tmp.getNextRef();  }  }  **public** **void** reverse() {  System.***out***.println("\nreversing the linked list\n");  Node<T> prev = **null**;  Node<T> current = head;  Node<T> next = **null**;  **while** (current != **null**) {  next = current.getNextRef();  current.setNextRef(prev);  prev = current;  current = next;  }  head = prev;  }  } |

# **Write a program to print all permutations of a given string.**

|  |
| --- |
| **public** **class** StringPermutationsEx {  **public** **static** **void** main(String a[]) {  List<String> output = StringPermutationsEx.*generatePermutations*("xyz");  System.***out***.println("Result size: " + output.size());  output.stream().forEach(System.***out***::println);  System.***out***.println("------------------");  output = StringPermutationsEx.*generatePermutations*("ABCD");  System.***out***.println("Result size: " + output.size());  output.stream().forEach(System.***out***::println);  }  **public** **static** List<String> generatePermutations(String input) {  List<String> strList = **new** ArrayList<String>();  StringPermutationsEx.*permutations*("", input, strList);  **return** strList;  }  **private** **static** **void** permutations(String consChars, String input, List<String> opContainer) {  **if** (input.isEmpty()) {  opContainer.add(consChars + input);  **return**;  }  **for** (**int** i = 0; i < input.length(); i++) {  *permutations*(consChars + input.charAt(i), input.substring(0, i) + input.substring(i + 1), opContainer);  }  }  } |

# **String Recursive Reversal**

|  |
| --- |
| **public** **class** StringRecursiveReversal {  String reverse = "";  **public** String reverseString(String str) {  **if** (str.length() == 1) {  **return** str;  } **else** {  reverse += str.charAt(str.length() - 1) + reverseString(str.substring(0, str.length() - 1));  **return** reverse;  }  }  **public** **static** **void** main(String a[]) {  StringRecursiveReversal srr = **new** StringRecursiveReversal();  System.***out***.println("Result: " + srr.reverseString("Java2novice"));  }  } |

# **Write a program to find top two maximum numbers in an array.**

|  |
| --- |
| **public** **class** TwoMaxNumbers {  **public** **void** printTwoMaxNumbers(**int** [] nums) {  **int** maxOne = 0;  **int** maxTwo = 0;  **for** (**int** n: nums) {  **if** (maxOne < n) {  maxTwo = maxOne;  maxOne = n;  } **else** **if** (maxTwo < n) {  maxTwo = n;  }  }  System.***out***.println("First Max Number: " + maxOne);  System.***out***.println("Second Max Number: " + maxTwo);  }  **public** **static** **void** main (String a[]) {  **int** num [] = { 5, 34, 78, 2, 45, 1, 99, 23 };  TwoMaxNumbers tmn = **new** TwoMaxNumbers();  tmn.printTwoMaxNumbers(num);  }  } |

# **Composition: -** Composition is a restricted form of Aggregation in which two entities are highly dependent on each other. It represents part of relationship. In Composition, both the entities are dependent on each other, where there is composition between two entities, the composed object cannot exist without the other entity.

|  |
| --- |
| Class Book {  Public String title;  Public String author;  Book (String title, String author){  This.title = title;  This.author = author;  }  }  Class Library {  Private final List<Book> books;  Library (List<Book> books) {  This.books = books;  }  Public List<Book> getTotalBooksInLibrary(){  Return books;  }  }  Class Composition {  Public static void main (String args []) {  Book b1 = new Book (“a”, “b”);  Book b2 = new Book (“c”, “d”);  Book b3 = new Book (“e”, “f”);  List<Book> b = new ArrayList<Book> ();  b.add(b1);  b.add(b2);  b.add(b3);  Library l = new Library(b);  List<Book> bks = l. getTotalBooksInLibrary();  For(Book bs : bks){  System.out.println(bs.title + “ “ bs.author);  } |

# **Aggregation: -** It represents HAS-A relationship.It is unidirectional association i.e. a one way relationship. In Aggregation, both the entities can serive individually which means ending one entity will not effect the other entity. Code reuse is best achieved by aggregartion.

|  |
| --- |
| Class Student {  Sting name;  Int id;  Sting dept;  Student (Sting name, int id, Sting dept) {  This.name = name;  This.id = id;  This.dept = dept;  }  }  Class Depratment{  String name;  Private List<Student> std;  Department (Sting name , List<Student>std){  This.name = name;  This.std = std;  }  Public List<Student> getStudent(){  Return std;  }  }  Class Institute{  String intName;  Private List<Department> dept;  Institute(String intName , List<Department> dept){  This.intName = intName;  This.dept = dept;  }  Public int getTotalStudentInInstitute(){  Int no = 0;  List<Student> std;  For(Department de : dept){  Std= de.getStudent();  For(Student s:std){  No++;  }  }  Return no++;  }  }  Class Aggregation{  Student s1 = new Student (“a”, 1,“cse”);  Student s2 = new Student (“b”, 2, “cse”);  Student s3 = new Student (“c”, 3, “cse”);  Student s4 = new Student (“d”, 4, “ee”);  Student s5 = new Student (“e”,5, “ee”);  List<Student> cse\_std = new ArrayList<Student> ();  Cse\_std.add(s1);  Cse\_std.add(s2);  Cse\_std.add(s3);  List<Student> ee\_std = new ArrayList<Student> ();  Ee\_std.add(s4);  Ee\_std.add(s5);  Department cse = new Department (“cse” , cse\_std);  Department ee = new Department (“ee” , ee\_std);  List<Department> depts = new ArrayList<Department> ();  Depts.add(cse);  Depts.add(ee);  Institute it = new Institute (“BITS”, depts);  System.out.println(it. getTotalStudentInInstitute);  }  } |

# **Association: -** Association is relationship between two separate classes which establishes through their objects.Association can be one-to-one , one-to-many, many-to-one, many-to-many.

|  |
| --- |
| Class Bank {  Private String name;  Bank (String name){  This.name = name;  }  Public Stirng getBankName(){  Return this.name;  }  }  Class Employee {  Private String name;  Employee (String name){  This.name = name;  }  Public Stirng getEmployeeName(){  Return this.name;  }  }  Class Association {  Public static void main (Stirng args []){  Bank b = new Bank(“axis”);  Employee e = new Employee(“pretty”);  System.out.println(e.getEmployeeName() + b.getBankName();  }  } |

# **Train Ticket Booking: - There is only 1 ticket available in train, and two passengers are trying to book that ticket at same time.**

|  |
| --- |
| **class** TicketBooking **implements** Runnable {  **int** ticketsAvailable=1;  **public** **void** run () {            System.*out*.println("Waiting to book ticket for: "+Thread.*currentThread*(). getName ());  **synchronized (this) {**  **if**(ticketsAvailable>0) {                         System.*out*.println("Booking ticket for: "+Thread.*currentThread*(). getName ());  **try** {                                Thread.*sleep*(1000);                         } **catch** (Exception e) {}                           ticketsAvailable--;                         System.*out*.println("Ticket BOOKED for: "+ Thread.*currentThread*(). getName ());                         System.*out*.println("currently ticketsAvailable = "+ticketsAvailable);                   } **else** {                         System.*out*.println("Ticket NOT BOOKED for: "+                                   Thread.*currentThread*(). getName ());                   }  **}**     }  }  **public** **class** MyClass {  **public** **static** **void** main (String args[])   {            TicketBooking obj=**new** TicketBooking();            Thread thread1=**new** Thread (obj,"Passenger1 Thread");            Thread thread2=**new** Thread (obj,"Passenger2 Thread");            thread1.start();            thread2.start();     }  }  /\*OUTPUT  Waiting to book ticket for: Passenger2 Thread  Waiting to book ticket for: Passenger1 Thread  Booking ticket for: Passenger2 Thread  **Ticket BOOKED for: Passenger2 Thread**  currently ticketsAvailable = 0  **Ticket NOT BOOKED for: Passenger1 Thread**  \*/ |

# **Check string contains special characters in java using regex.**

|  |
| --- |
| **public** **class** StringContainsSpecialCharactersExample {  **public** **static** **void** main(String[] args) {  Pattern pattern = Pattern.*compile*("[a-zA-Z0-9]\*");  String str = "abc@";  Matcher matcher = pattern.matcher(str);  **if** (!matcher.matches()) {  System.***out***.println("string '"+str + "' contains special character");  } **else** {  System.***out***.println("string '"+str + "' doesn't contains special character");  }  }  }  string 'abc@' contains special character |

# **Generate account number dynamically:-** Java program to design a class account using the inheritance and static that show all function of bank and generate account number dynamically.

|  |
| --- |
| **public** **class** GenerateBankAccountNumber {  **static** **int** *acc\_no* = 10001;  **float** amount;  **public** GenerateBankAccountNumber() {  amount = 1000;  System.***out***.println(" your account number is = " + *acc\_no*);  *acc\_no*++;  }  **public** **void** getAmount() {  System.***out***.println("current balance = " + amount);  }  **public** **void** display() {  System.***out***.println("account number = " + *acc\_no*);  System.***out***.println("current balance = " + amount);  }  **public** **void** widthdraw(**float** x) {  **if** (amount == 1000 || amount <= x) {  System.***out***.println(" Sorry you can't widthdraw amount = " + amount);  } **else** {  amount = amount - x;  System.***out***.println(" widthdraw amount = " + amount);  System.***out***.println(" after widthdraw ");  getAmount();  }  }  **public** **void** deposit(**float** x) {  **if** (x == 0.0) {  System.***out***.println(" oops you can't be deposit");  } **else** {  amount = amount + x;  System.***out***.println(" after deposit");  getAmount();  }  }  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub  GenerateBankAccountNumber b = **new** GenerateBankAccountNumber();  b.deposit(0);  b.widthdraw(120.3f);  b.display();  System.***out***.println("\n");  GenerateBankAccountNumber b1 = **new** GenerateBankAccountNumber();  b1.deposit(1000.0f);  b1.widthdraw(150.5f);  b1.display();  }  } |

# **Highest score Student: -** WAP that accepts 10 students’ records (roll no and score) and prints them in decreasing order of scores. In case there are multiple records printing to the same student, the program should choose a single record containing the highest score. Then program should capable of accepting a multiline score. Each subsequent line of input will contain a student record, that is roll no and score. The output should consist of the conbimation of roll no and corresponding score in decreasing order of same.

|  |
| --- |
| I/P :- 1001-40 , 1002-50 , 1003 -60 , 1002 – 80 , 1005 -35 , 1005 -55 , 1007-68 , 1009-99, 1009-10, 1004-89  O/P:- 1009 -99 , 1004 – 89 , 1002 -80 , 1007 -68 , 1003-60,1005-55,1001-40  Public class main{  Public ststic void main(Sting args[]){  String data = new String[10];  Int splitData[] [] = new int [10][2];  Scanner sc = new Scanner(System.in);  Sop(“enter 10 records”);  For(int i=0; i<10; i++){  Data [i] = sc.next();  String str [] = Data[i].split(“-“);  splitData[i][0] = Integer.parseInt(str[1]);  splitData[i][1] = Integer.parseInt(str[0]);  }  Arrays.sort(spiltData , new Comparator<int[]>(){  Public int compare(int[]o1, int[]o2){  Return o2[0]-o1[0];  }  });  Byte flag = 0;  Sop(“output”);  For(int I =0; i<10; i++){  Flag = 0;  For(int j = i-1 ; j>= 0; j--){  If(splitData[i][1] == splitData[j][1]){  Flag = 1;  Break;  }  }  If(flag ==0){  Sop(splitData[i][1]+ “-“+ splitData[i][0]);  }  }  }  } |

# **WAP to sort the numeric sub array using Arrays parallelSortUserInput.**

|  |
| --- |
| I/P :- {6,7,4,5,8,3,9,1,0,2}  O/P:- {6,7,3,4,5,8,9,1,0,2}    Integer [] a = {6,7,4,5,8,3,9,1,0,2};  Int from = 2;  Int to = 6;  Public static void getSortedSubArray(Integer [] a, int from , int to){  Arrays.parallelSort(a,from,to);  System.out.print(Arrays.toString(a));  } |