Data Structures, Algorithm and Programs for Java Interview

- This is a combined list of questions from various data structure e.g. array, linked list, stack or queue. It includes some coding questions as well, which gel with data structures.

1. How to find middle element of linked list in one pass?

- One of the most popular question from data structures and algorithm, mostly asked on telephonic interview. Since many programmer know that, in order to find length of linked list we need to first traverse through linked list till we find last node, which is pointing to null, and then in second pass we can find middle element by traversing only half of length. They get confused when interviewer ask him to do same job in one pass. In order to find middle element of linked list in one pass, you need to maintain two-pointer, one increment at each node while other increments after two nodes at a time, by having this arrangement, when first pointer reaches end, second pointer will point to middle element of linked list. See this trick to find middle element of linked list in single pass for more details.

2. How to find if linked list has a loop ?

- This question has bit of similarity with earlier algorithm and data structure interview question. I mean we can use two pointer approach to solve this problem. If we maintain two pointers, and we increment one pointer after processing two nodes and other after processing every node, we are likely to find a situation where both the pointers will be pointing to same node. This will only happen if linked list has loop.

3. How to find 3rd element from end in a linked list in one pass?

- This is another frequently asked linked list interview question. This question is exactly similar to finding middle element of linked list in single pass. If we apply same trick of maintaining two pointers and increment other pointer, when first has moved up to 3rd element, than when first pointer reaches to the end of linked list, second pointer will be pointing to the 3rd element from last in a linked list.

4. In an integer array, there is 1 to 100 number, out of one is duplicate, how to find?

- This is a rather simple data structures question, especially for this kind of. In this case you can simply add all numbers stored in array, and total sum should be equal to `n(n+1)/2`. Now just subtract actual sum to expected sum, and that is your duplicate number. Of course there is a brute force way of checking each number against all other numbers, but that will result in performance of O(n^which is not good. By the way this trick will not work if array have multiple duplicates or its not numbers forming arithmetic progression. Here is example of one way to find duplicate number in array.

Find all duplicates in an array in linear time (v1)

- This is a common interview question where you need to write a program to find all duplicates in an array where the numbers in the array are in the range of `0` to `n-1` where `n` is the size of the array. For example: `[1, 2, 3, 3]` is okay but `[1, 2, 6, 3]` is not. In this version of the challenge there can be multiple duplicate numbers as well. The algorithm below is commented to explain what each piece of code does, but the general algorithm is:
- (1) Loop through the array (2) For each element, find array[absolute(array[i])] in the array and set its value to negative if positive (3) If in step 2 you encounter a negative number, then it means the element at index `i` in the array is a duplicate

An example

- arr = [1, 2, 2, 3, 1] At i = 0 arr[absolute(arr[0])] = 2 which is positive so make it negative arr = [1, -2, 2, 3, 1] At i = 1 arr[absolute(arr[1])] = 2 which is positive so make it negative arr = [1, -2, -2, 3, 1] At i = 2 arr[absolute(arr[2])] = -2 which is negative which means the element originally at index 2 is a duplicate duplicates = [2] At i = 3 arr[absolute(arr[3])] = 3 which is positive so make it negative arr = [1, -2, -2, -3, 1] At i = 4 arr[absolute(arr[4])] = -2 which is negative which means the element originally at index 4 is a duplicate duplicates = [2, 1]

Solution

```
function duplicates(arr) {

  // where we will store our duplicate values
  var dups = [];

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

    // get element in array
    var el = arr[Math.abs(arr[i])];

    // element in array is positive so make it negative
    if (el > 0) { arr[Math.abs(arr[i])] = -arr[Math.abs(arr[i])]; }

    // special case if element is zero
    // we set the value at this index to -arr.size as not to
    // mix this number up with the others because we know the
    // numbers are all in the range of 0 to n-1
    else if (el === 0) { arr[Math.abs(arr[i])] = -arr.length; }

    // element is negative so it is a duplicate
    else {
      if (Math.abs(arr[i]) === arr.length) { dups.push(0); }
      else { dups.push(Math.abs(arr[i])); }
    }

  }

  return dups;

}
```

- duplicates([0,2,0,1,3,3]);

if the array size is n and the array elements are in the range 1 to n-1(with one element repeated) then we will traverse the array once from index(0) to index(n-1). If we have k at index(0) make the array value negative at index k(if positive) else return that index if the value at that index is already negative.Will do the same for complete loop(while accessing the index ignore the sign of array element) until we get the repeated element.

Thus no extra space is required and it'll take O(n) space. But we are loosing the original array here so no need to worry, just make all the negative elements of the array as positive in another pass. ex. there is an array of size 5 with the values as- A = [1,4,3,2,1] for i = 0 A[0] = 1 so we will change A[1] which is +ve, now A = [1,-4,3,2,1] for i = 1 A[1] = -4 so we will change A[4] which is +ve, now A = [1,-4,3,2,-1] for i = 2 A[2] = 3 so we will change A[3] which is +ve, now A = [1,-4,3,-2,-1] for i = 3 A[3] = -2 so we will change A[2] which is +ve, now A = [1,-4,-3,2,-1] for i = 4 A[4] = -1 so we will change A[1] which is already -ve so report this index(1) as ans.

Solution 2* ***This method does not work if any of the elements in the array is smaller than 0 or greater than n-1.***

traverse the list for i= 0 to n-1 elements{check for sign of A[abs(A[i])] ;if positive thenmake it negative by A[abs(A[i])]=-A[abs(A[i])];else // i.e., A[abs(A[i])] is negativethis element (ith element of list) is a repetition}

Java code for the function is:

```
static 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]) + " ");
         }}
```

Time Complexity : O(n), as only one traversal of the array is required.

Testdata: If you are going for an SDET interview, seeing that your code passes all the possible testdata and handles every edge case.Positive Tests:{4,2,6,7,2}{1,2,3,4,5,5}{1,1,2,3,4,5}{0,0,0,0,0,0}Negative Tests{1,2,3,4,5,6}{-1,-4,1,2,3}

Another solution

Assuming that there is an upped bound for the values of the numbers in the array (which is the case with all built-in integer types in all programming languages I 've ever used -- for example, let's say they are 32-bit integers) there is a solution that uses constant space:

1. Create an array of N elements, where N is the upper bound for the integer values in the input array and initialize all elements to `0` or `false` or some equivalent. I 'll call this the *lookup* array.
2. Loop over the input array, and use each number to index into the lookup array. If the value you find is `1` or `true` (etc), the current number in the input array is a duplicate.
3. Otherwise, set the corresponding value in the lookup array to `1` or `true` to remember that we have seen this particular input number.

Technically, this is O(n) time and O(1) space, and it does not destroy the input array. Practically, you would need things to be going your way to have such a program actually run (e.g. it's out of the question if talking about 64-bit integers in the input).

Another solution

A **Simple Solution** is to use two nested loops. The outer loop picks an element one by one, the inner loop checks whether the element is repeated or not. Once we find an element that repeats, we break the loops and print the element. Time Complexity of this solution is $O(n^2)$

We can **Use Sorting** to solve the problem in O(nLogn) time. Following are detailed steps.1) Copy the given array to an auxiliary array temp[].2) Sort the temp array using a O(nLogn) time sorting algorithm.3) Scan the input array from left to right. For every element, count its occurrences in temp[] using binary search. As soon as we find an element that occurs more than once, we return the element. This step can be done in O(nLogn) time.

We can **Use Hashing** to solve this in O(n) time on average. The idea is to traverse the given array from right to left and update the minimum index whenever we find an element that has been visited on right side.

```
/* Java program to find first repeating element in arr[] */
import java.util.*;

class Main
{
    // This function prints the first repeating element in arr[]
    static void printFirstRepeating(int arr[])
    {
        // Initialize index of first repeating element
        int min = -1;

        // Creates an empty hashset
        HashSet<Integer> set = new HashSet<>();

        // Traverse the input array from right to left
        for (int i=arr.length-1; i>=0; i--)
        {
            // If element is already in hash set, update min
            if (set.contains(arr[i]))
                min = i;

            else  // Else add element to hash set
                set.add(arr[i]);
        }
```

```
        // Print the result
        if (min != -1)
          System.out.println("The first repeating element is " + arr[min]);
        else
          System.out.println("There are no repeating elements");
    }

    // Driver method to test above method
    public static void main (String[] args) throws java.lang.Exception
    {
        int arr[] = {10, 5, 3, 4, 3, 5, 6};
        printFirstRepeating(arr);
    }
}
```

5. How to reverse String in Java ?

- This is one of my favorite question. Since `String` is one of the most important type of programming, you expect lot of question related to String any data structure interview. There are many ways to reverse Sting in Java or any other programming language, and interviewer will force you to solve this problem by using without API i.e. without using `reverse()` method of `StringBuffer`. In follow-up he may ask to reverse String using recursion as well. See 3 ways to reverse String in Java to learn reversing String using both loops and recursion in Java.

6. Write a Java program to sort an array using Bubble Sort algorithm?

- I have always send couple of questions from searching and sorting in data structure interviews. Bubble sort is one of the simplest sorting algorithm but if you ask anyone to implement on the spot it gives you an opportunity to gauge programming skills of a candidate. See How to sort array using Bubble Sort in Java for complete solution of this datastrucutre interview question.

7. How do you find duplicates in an array if there is more than one duplicate?

- Sometime this is asked as follow-up question of earlier data structure interview question, related to finding duplicates in Array. One way of solving this problem is using a **HashTable** or HashMap data structure. You can traverse through array, and store each number as key and number of occurrence as value. At the end of traversal you can find all duplicate numbers, for which occurrence is more than one. In Java if a number already exists in HashMap then calling `get(index)` will return number otherwise it return null. this property can be used to insert or update numbers in `HashMap`.

8. What is binary search tree?

- Tree data structures
- Binary Search Tree has some special properties e.g. left nodes contains items whose value is less than root , right sub tree contains keys with higher node value than root, and there should not be any duplicates in the tree.
- Apart from definition, interview can ask you to implement binary search tree in Java and questions on tree traversal e.g. IN order, preorder, and post order traversals are quite popular data structure question.

9. How to reverse linked list using recursion and iteration?

- data structures

10. Write a Java program to implement Stack in Java?

- You can implement Stack by using array or linked list. This question expect you to implement standard method provided by stack data structure e.g. `push()` and `pop()`. Both `push()` and `pop()` should be happen at top of stack, which you need to keep track. It's also good if you can implement utility methods like `contains()`, `isEmpty()` etc. By the way JDK has java.util.Stack class and you can check it's code to get an idea. You can also check Effective Java book, where Josh Bloch has explains how an incorrect implementation of stack can cause memory leak in Java.
- That's all on this **list of data structure interview questions and answers**. This is one topic, which is always asked in any programming interview, doesn't matter if you are C, C++, or Java developer, basic knowledge of data structure like array, linked list, stack, queue, tree are must to clear any programming interview.

11. Find all anagrams in a given sorted dictionary - MetLife

- Find all anagrams of a word in a file. Input - only file name and word. Output - all set of word in file that are anagrams of word. Write production quality code.

Algorithms

12. *What is an algorithm?*

- Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.
- An algorithm is thus a sequence of computational steps that transform the input into the output.

13. *What is time complexity of Binary Search?*

- Time complexity of binary search is O(Logn).

14. *Can Binary Search be used for linked lists?*

- Since random access is not allowed in linked list, we cannot reach the middle element in O(1) time. Therefore Binary Search is not possible for linked lists

15. How to find if two given rectangles overlap?

- Two rectangles do not overlap if one of the following conditions is true.

1) One rectangle is above top edge of other rectangle.
2) One rectangle is on left side of left edge of other rectangle.

16. How to find angle between hour and minute hands at a given time?

- The idea is to take a reference point as 12. Find the angle moved by hour and minute hands, subtract the two angles to find the angle between them. See angle between hour hand and minute hand for more details

17. *When does the worst case of QuickSort occur?*

- In quickSort, we select a pivot element, then partition the given array around the pivot element by placing pivot element at its correct position in sorted array.
- The **worst case of quickSort occurs when one part after partition contains all elements and other part is empty.** For example, if the input array is sorted and if last or first element is chosen as a pivot, then the worst occurs.
- See http://quiz.geeksforgeeks.org/quick-sort/ for more details.

18. *A sorted array is rotated at some unknown point, how to efficiently search an element in it.*

- A simple approach is linear search, but we can search in O(Logn) time using Binary Search.
- Other variations of this problem like find the minimum element or maximum element in a sorted and rotated array.

19. *Given a big string of characters, how to efficiently find the first unique character in it?*

- The efficient solution is to use character as an index in a count array. Traverse the given string and store index of first occurrence of every character, also store count of occurrences. Then traverse the count array and find the smallest index with count as 1..

20. Given a big array, how to efficiently find k'th largest element in it?

- There can be many solutions for this. The best solution is to use min heap. We Build a Min Heap MH of the first k elements. For each element, after the kth element (arr[k] to arr[n-1]), compare it with root of MH, if the element is greater than the root then make it root and call heapify for MH, Else ignore it. Finally, MH has k largest elements and root of the MH is the kth largest element. See k largest(or smallest) elements for more details.

21. *Given an array of size n with range of numbers from 1 to n+1.* find the two repeating numbers,

- Use two loops. In the outer loop, pick elements one by one and count the number of occurrences of the picked element in the inner loop.
- This method doesn't use the other useful data provided in questions like range of numbers is between 1 to n and there are only two repeating elements.

```
class RepeatElement
{
```

```
    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] + " ");
            }
        }
    }

    public static void main(String[] args)
    {
        RepeatElement repeat = new RepeatElement();
        int arr[] = {4, 2, 4, 5, 2, 3, 1};
        int arr_size = arr.length;
        repeat.printRepeating(arr, arr_size);
    }
}
```

22. Given Graph use:

1. Breadth First Search (BFS)
2. Depth First Search (DFS)
3. Shortest Path from source to all vertices **Dijkstra**
4. Shortest Path from every vertex to every other vertex **Floyd Warshall**
5. To detect cycle in a Graph **Union Find**
6. Minimum Spanning tree **Prim**
7. Minimum Spanning tree **Kruskal**
8. Topological Sort
9. Boggle (Find all possible words in a board of characters)
10. Bridges in a Graph

23. Given Linked List use

1. Insertion of a node in Linked List (On the basis of some constraints)
2. Delete a given node in Linked List (under given constraints)
3. Compare two strings represented as linked lists
4. Add Two Numbers Represented By Linked Lists
5. Merge A Linked List Into Another Linked List At Alternate Positions
6. Reverse A List In Groups Of Given Size
7. Union And Intersection Of 2 Linked Lists
8. Detect And Remove Loop In A Linked List
9. Merge Sort For Linked Lists
10. Select A Random Node from A Singly Linked List

24. Given Dynamic Programming use

1. Longest Common Subsequence
2. Longest Increasing Subsequence
3. Edit Distance
4. Minimum Partition
5. Ways to Cover a Distance
6. Longest Path In Matrix
7. Subset Sum Problem
8. Optimal Strategy for a Game
9. 0-1 Knapsack Problem
10. Boolean Parenthesization Problem

25. Sorting And Searching use

1. Binary Search
2. Search an element in a sorted and rotated array
3. Bubble Sort
4. Insertion Sort
5. Merge Sort
6. Heap Sort (Binary Heap)
7. Quick Sort
8. Interpolation Search
9. Find Kth Smallest/Largest Element In Unsorted Array
10. Given a sorted array and a number x, find the pair in array whose sum is closest to x

26. Tree / Binary Search Tree use

1. Find Minimum Depth of a Binary Tree
2. Maximum Path Sum in a Binary Tree
3. Check if a given array can represent Preorder Traversal of Binary Search Tree
4. Check whether a binary tree is a full binary tree or not
5. Bottom View Binary Tree
6. Print Nodes in Top View of Binary Tree
7. Remove nodes on root to leaf paths of length < K
8. Lowest Common Ancestor in a Binary Search Tree
9. Check if a binary tree is subtree of another binary tree
10. Reverse alternate levels of a perfect binary tree

27. Number Theory use

1. Modular Exponentiation
2. Modular multiplicative inverse
3. Primality Test | Set 2 (Fermat Method)

4. Euler's Totient Function
5. Sieve of Eratosthenes
6. Convex Hull
7. Basic and Extended Euclidean algorithms
8. Segmented Sieve
9. Chinese remainder theorem
10. Lucas Theorem

28. BIT Manipulation use

1. Maximum Subarray XOR
2. Magic Number
3. Sum of bit differences among all pairs
4. Swap All Odds And Even Bits
5. Find the element that appears once
6. Binary representation of a given number
7. Count total set bits in all numbers from 1 to n
8. Rotate bits of a number
9. Count number of bits to be flipped to convert A to B
10. Find Next Sparse Number

29. String / Array use

1. Reverse an array without affecting special characters
2. All Possible Palindromic Partitions
3. Count triplets with sum smaller than a given value
4. Convert array into Zig-Zag fashion
5. Generate all possible sorted arrays from alternate elements of two given sorted arrays
6. Pythagorean Triplet in an array
7. Length of the largest subarray with contiguous elements
8. Find the smallest positive integer value that cannot be represented as sum of any subset of a given array
9. Smallest subarray with sum greater than a given value
10. Stock Buy Sell to Maximize Profit

30. Use a hashmap with string as key and list<string> as value where list of strings contain all anagrams of a key string.

1) For each word in the file, find beta string which is its sorted version ex abd is sorted version of bad, adb and dba. Put this word to that list whose key is this beta string. If it does not exist then create a new list with the beta string as key in map.
2) Finally print all strings from the list whose key is the input word(sorted/beta string).

Solution

```
import java.util.*;
import java.io.*;
public class Anagram {
```

```
    public static void main(String[] args) {
    String beta = args[1];
        try {
       Map<String, List<String>> m = new HashMap<String, List<String>>();
            Scanner s = new Scanner(new File(args[0]));
            while (s.hasNext()) {
                String word = s.next();
                String alpha = sorting(word);

                List<String> l = m.get(alpha);
                if (l == null)
                    m.put(alpha, l=new ArrayList<String>());
                l.add(word);
            }
       List<String> l = m.get(sorting(beta));
       Object[] arr = l.toArray();
           for (int i=0; i < arr.length; i++)
                System.out.println(arr[i]);
        }
    catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }
    }
    private static String sorting(String s) {
        char[] a = s.toCharArray();
        Arrays.sort(a);
        return new String(a);
    }
}
```

- Or, Given a sequence of words, print all anagrams together | Set Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

31. Find all anagrams in a given sorted dictionary - MetLife

Find all anagrams of a word in a file. Input - only file name and word. Output - all set of word in file that are anagrams of word. Write production quality code.

Algorithm

3) Use a **hashmap** with **string as key** and **list<string> as value** where list of strings contain all anagrams of a key string.
4) For each word in the file, find beta string which is its sorted version ex abd is sorted version of bad, adb and dba. Put this word to that list whose key is this beta string. If it does not exist then create a new list with the beta string as key in map.
5) Finally print all strings from the list whose key is the input word(sorted/beta string).

Solution

```
import java.util.*;
import java.io.*;
public class Anagram {
    public static void main(String[] args) {
    String beta = args[1];
        try {
       Map<String, List<String>> m = new HashMap<String, List<String>>();
            Scanner s = new Scanner(new File(args[0]));
            while (s.hasNext()) {
                String word = s.next();
                String alpha = sorting(word);

                List<String> l = m.get(alpha);
                if (l == null)
                    m.put(alpha, l=new ArrayList<String>());
                l.add(word);
            }
       List<String> l = m.get(sorting(beta));
```

```
        Object[] arr = l.toArray();
            for (int i=0; i < arr.length; i++)
                 System.out.println(arr[i]);
          }
      catch (Exception e) {
              System.out.println(e);
              System.exit(1);
          }
      }
      private static String sorting(String s) {
           char[] a = s.toCharArray();
          Arrays.sort(a);
          return new String(a);
      }
}
```

Or, Given a sequence of words, print all anagrams together | Set Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

32. Write a Java program to check if a number is Prime or not? (solution)

- A number is said prime if it is not divisible by any other number except itself. 1 is not considered prime, so your check must start with 2. Simplest solution of this to check every number until the number itself to see if its divisible or not. When Interviewer will ask you to improve, you can say that checking until square root of the number. If you can further improve the algorithm, you will more impress your interviewer. check out the solution for how to do this in Java

```
for(int i=2;i<=num/2;i++)
   {
      temp=num%i;
      if(temp==0)
      {
         isPrime=false;
         break;
      }
   }
```

33. How to Swap two numbers without using temp variable?

- Method 1 (Using Arithmetic Operators)The idea is to get sum in one of the two given numbers. The numbers can then be swapped using the sum and subtraction from sum.
- Method 2 (Using Bitwise XOR)The bitwise XOR operator can be used to swap two variables. The XOR of two numbers x and y returns a number which has all the bits as 1 wherever bits of x and y differ. For example XOR of 10 (In Binary 1and 5 (In Binary 0is 1111 and XOR of 7 (0and 5 (0is (0010).

34. Given pack of cards represented as A1,A2…A13 and B1,B2,..B13, and C1,c2,…c13 and d1,d2..d13.

35. You are given a file which has 51 card numbers. Only 1 card is missing. Find the missing card. - MetLife

It is based on following property of XOR operation:

```
1) If you XOR any number with itself, you will get ZERO.
i.e.. A ^ A = 2) If you XOR any number with ZERO, you get the same number
i.e. A ^ 0 = A
```

Say n= 5 and array is 1, 2, 4, 5 (i.e. 3 is missing here), so based on method 2:

```
1) X1 = 1 ^ 2 ^ 4 ^ 5 (XOR all the array elements)
2) X2 = 1 ^ 2 ^ 3 ^ 4 ^ 5 (XOR all numbers from 1 to n)
```

3) missing number = X1 ^ X2 = (1 ^ 2 ^ 4 ^ 5) ^ (1 ^ 2 ^ 3 ^ 4 ^ 5) =

Here each number in X1 gets XORed with same number present in X2 and it's becomes ZERO (1 ^ 1 = 0, 2 ^ 2 = 0, 4 ^ 4 = 0, 5 ^ 5 = 0).

36. The missing number 3 is only in X2 and not in X1, so it's becomes 0 ^ 3 = How to check if linked list contains loop in Java?

- Two pointers, fast and slow is used while iterating over linked list. Fast pointer moves two nodes in each iteration, while slow pointer moves to one node. If linked list contains loop or cycle than both fast and slow pointer will meet at some point during iteration. If they don't meet and fast or slow will point to null, then linked list is not cyclic and it doesn't contain any loop. Here is the exact algorithm

37. Write Java program to reverse String without using API?

- One more question to test problem solving skill of candidate. You wouldn't expect these kind of question in telephonic round of Java interview but these questions have now become norms. All interviewer is looking it for logic, you don't need to write the code but you should be able to think of solution.

- Iteratively:

```
static String reverseMe(String s) {
  StringBuilder sb = new StringBuilder();
  for(int i = s.length() - 1; i >= 0; --i)
    sb.append(s.charAt(i));
  return sb.toString();
}
```

- Recursively:

```
static String reverseMe(String s) {
   if(s.length() == 0)
     return "";
   return s.charAt(s.length() - + reverseMe(s.substring(0,s.length()-1));
}
```

38. How do you convert bytes to character in Java?

- Bytes are converted to character or text data using character encoding. When you read binary data from a file or network endpoint, you provide a character encoding to convert those bytes to equivalent character. Incorrect choice of character encoding may alter meaning of message by interpreting it differently.

39. Write a Program to find maximum and minimum number in array?

- This is another coding question test problem solving ability of candidate. Be ready for couple of follow up as well depending upon how you answer this question. Simplest way which comes in mind is to sort the array and then pick the top and bottom element. For a better answer see the solution.

40. Write a program to reverse Array in place?

- you need to reverse the array in place, which means you cannot use additional buffer, one or two variable will be fine. Note you cannot use any library code, you need to create your own logic.

41. How to find middle element of linked list in one pass?

- Another simple problem solving question for warm up. In a singly linked list you can only traverse in one direction and if you don't know the size then you cannot write a loop to exactly find out middle element, that is the crux of the problem. One solution is by using two pointers, fast and slow. Slower pointer moves 1 step when faster pointer move to 2 steps, causing slow to point to middle when fast is pointing to end of the list i.e. null. Check out solution for Java code sample.

42. Let's say that you have 25 horses, and you want to pick the fastest 3 horses out of those 25. In each race, only 5 horses can run at the same time because there are only 5 tracks. What is the minimum number of races required to find the 3 fastest horses without using a stopwatch? – MetLife

http://www.programmerinterview.com/index.php/puzzles/25-horses-3-fastest-5-races-puzzle/

2nd best solution, 7 races

- This question is an interesting one. Let's start by asking ourselves, what are our limitations? Well, we don't have a stopwatch so we can't time the horses. That means that we can't compare the race times of each horse – otherwise we

could simply have 5 races of 5 horse each and pick out the 3 fastest times to get the 3 fastest horses. Remember that we can only race 5 horses in each race (otherwise we could just have one race with 25 horses, pick the 3 fastest, and we would be done!).

Draw out a table

- In problems like this, it helps tremendously to create some sort of visual aid that you can refer to. With that in mind, we have created this table where each entry represents a different horse.

X1 X2 X3 X4 X5

X6 X7 X8 X9 X10

X11 X12 X13 X14 X15

X16 X17 X18 X19 X20

X21 X22 X23 X24 X25

- Let's say that we have 5 races of 5 horses each, so each row in the table above represents a race. So, "X1 X2 X3 X4 X5 " represents a race, and "X6 X7 X8 X9 X10 " represents another race, etc. In each row, the fastest horses are listed in descending order, from the fastest (extreme left) to the slowest (extreme right).

Only 5 horses each race

- So, now we ask ourselves: what do we know after these 5 races? Well, we do have the 5 five fastest horses from each race (X1, X6, X11, X16, and X21). But, does that mean we have the 5 fastest horses? Think about that for a second. Well, actually it does **not** mean that we have the 5 fastest horses. Because, what if the 5 fastest horses just happened to be in the first race – so X1 X2 X3 X4 X5 are the fastest horses. X1, X6, X11, X16, and X21 are all the fastest horses in their individual groups, but there could be one group that just happened to have all of the fastest horses. Remember we haven't compared all the horses to each other since we can only run 5 horses in a race, so that is still a possibility. This is very important to understand in this problem.
- We also know the 5 fastest horses from each group – but it's important to remember that the 5 group leaders are not necessarily the 5 fastest horses. So what can we do with that information?
- Well, we can race those 5 horses against each other (X1, X6, X11, X16, and X21) and that would be the 6th race. Let's say that the 3 fastest in that group are X1, X6, and X11 – automatically we can eliminate X16 and X21 since those 2 are definitely not in the top 3.
- What other horses can we eliminate after this 6th race? Well, we can automatically eliminate all the horses that X16 and X21 competed against in the preliminary races – since X16 and X21 are not in the top 3 then we also know that any horse that's slower than those 2 is definitely not in the top 3 either. This means we can eliminate X17 X18 X22 and X23 along with X16 and X21.
- Now, we also know that X1 is the fastest horse in the group since he was the fastest horse out of the 5 group leaders. So, we don't need to race X1 anymore. Are there any other horses that we can eliminate from further races? Well, actually there are. Think about it – if X6 and X11 are the 2nd and 3rd fastest in the group leaders, then we should be able to eliminate X8 since X6 raced against him and he was in 3rd place in that race. X7 could only possibly be the 3rd fastest, and since X8 is slower than X7, we can safely eliminate X8. We can also eliminate X12 and X13 since X11 was the 3rd fastest in the group leaders, and X12 and X13 were slower than X11.
- So, all together we can eliminate these horses after the 6th race: X17 X18 X22 X23 X16 X21, X12, X13, X8 and X1. This leaves us with the following horses to determine the 2nd and 3rd fastest horses:
- The question is asking for the best case scenario: minimum number of races.
- The best case would be if the data is sorted already, in other words, the fastest horse is horse #1 and the slowest horse is horse #25. So that is 6 races.

  Here are the races:

  Race 1:

  H1 – HFrom there you know the three fastest horses, but you still have to compare to the field. What you really need to compare is horse #3 against the field. The remaining races would be:

  Race 2:

  H3, H6-HRace 3:

  H3, H10-Race 4:

  H3, H14-Race 5:

H3, H18-Race 6:

H3, H22- http://www.mathsisfun.com/puzzles/weighing-9-balls-solution.html

43. Weighing 9 Balls Puzzle - MetLife

The Puzzle:

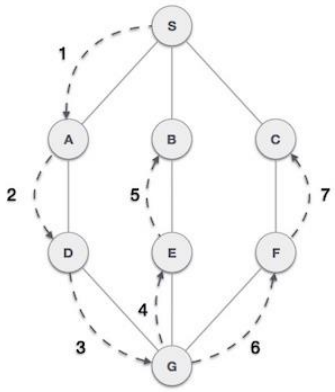
- You have 9 balls, equally big, equally heavy - except for one, which is a little heavier.
- How would you identify the heavier ball if you could use a pair of balance scales only twice?

Our Solution:

- Divide the 9 balls into 3 groups of 3. Compare the weight of two of those groups.
- The heavier group should then be obvious, it will either tip the scales, or, if the scales stay balanced, then it is the group you didn't include.
- Now, choose 2 balls from this group and compare their weights, and using the same logic as before, the heavier ball will be obvious.

44. Algorithm DFS Depth First Traversal- MetLife

- Data Structure used Stack
- a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



- As in the example given above, DFS algorithm traverses from A to B to C to D first then to E, then to F and lastly to G. It employs the following rules.

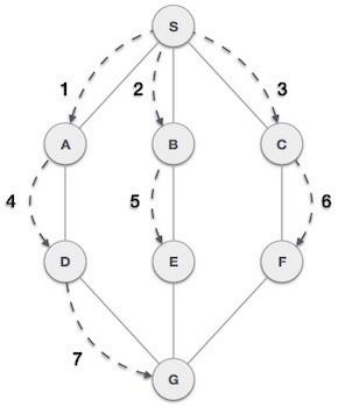
Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

45. Algorithm- Breadth First Traversal- MetLife

- Data Structure for BGS Breadth First Search Queue
- traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



- As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** − If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** − Repeat Rule 1 and Rule 2 until the queue is empty.
  - BFS explores/processes the closest vertices first and then moves outwards away from the source. Given this, you want to use a data structure that when queried gives you the oldest element, based on the order they were inserted. A queue is what you need in this case since it is first-in-first-out(FIFO).

  - DFS explores as far as possible along each branch first and then bracktracks. For this, a stack works better since it is LIFO(last-in-first-out)

http://stackoverflow.com/questions/3929079/how-can-i-remember-which-data-structures-are-used-by-dfs-and-bfs

5) http://www.geeksforgeeks.org/largest-sum-contiguous-subarray/

46. largest-sum-contiguous-subarray - MetLife

Objective: The maximum subarray problem is the task of finding the contiguous subarray within a one-dimensional array of numbers which has the largest sum.

Example:

```
int [] A = {−2, 1, −3, 4, −1, 2, 1, −5, 4};
```

Output: contiguous subarray with the largest sum is 4, −1, 2, 1, with sum Approach:

We will solve this problem in bottom-up manner.

"MS(i) *is maximum sum ending at index i"*

A[0]		A[i-1]	A[i]	...		A[n]

To calculate the solution for any element at index "*i*" has two options

- EITHER added to the solution found till "*i-1*"th index
- OR start a new sum from the index "*i*".

Recursive Solution:

MS(i) = Max[MS(i-1) + A[i] , A[i]]

Solution You have N stacks of books. Each stack of books has some non zero height Hi equal to the number of books on that stack (considering all the books are identical and each book has a height of 1 unit). In one move, you can select any number of consecutive stacks of books such that the height of each selected stack of books Hi <= K . Once such a sequence of stacks is chosen , You can collect any number of books from the chosen sequence of stacks .

What is the maximum number of books that you can collect this way ?

Input:

The first line of input contains an integer T denoting the no of test cases . Then T test cases follow. First line of each test case contains two space separated integers N and K where N is the number of stacks of books. Second line of each test case contains N space separated integers denoting the number of books Hi on each stack.

Output:

For each test case, print the maximum number of books you can collect.

Constraints:

1<=T<=11<=N<=11<=K<=11<=Hi<=1Example(To be used only for expected output):

Input

8 3 2 2 3 1 1 1 8 3 2 2 3 1 1 1

Output

Explanation :

For the first test case

N = 8 , K = 1 { 3 2 2 3 1 1 1 3 }

We can collect maximum books from consecutive stacks numbered 5, 6 and 7 having height less than equal to K.

For the second test case

N = 8 , K = 2 { 3 2 2 3 1 1 1 3 }

We can collect maximum books from consecutive stacks numbered 2 and 3 having height less than equal to K.

```
class GfG
{
    int max_Books(int a[], int n, int k)
    {
         // Your code here
    }
}
6) In an array all elements are repeated twice except one element. Find the
element.
eg: 2, 3, 4, 4, 3 – In this 2 is the element which is not repeated.
```

47. Design discussion on elevator. - MetLife

Hint: Ask questions related to elevator functionality; come up with a High Level design and Low level design. Be prepared for scheduling questions related to elevator.

48. "n" points are given , find the number of quadruplets which form square. - MetLife

The basic idea is to group the points by X or Y coordinate, being careful to avoid making groups that are too large.

Make a hash set of all the points. Something you can use to quickly check if a point is present.

HashSet extends AbstractSet and implements the Set **INTERFACE** . It creates a collection that uses a hash table for storage.

A hash table stores information by using a mechanism called **hashing**. In hashing, the informational content of a key is used to determine a unique value, called its hash code.

The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

Group the points by their X coordinate. Break any groups with more than sqrt(n) points apart, and re-group those now-free points by their Y coordinate. This guarantees the groups have at most sqrt(n) points and guarantees that for each square there's a group that has two of the square's corner points.

For every group g, for every pair of points p,q in g, check whether the other two points of the two possible squares containing p and q are present. Keep track of how many you find. Watch out for duplicates (are the two opposite points also in a group?).

Why does it work? Well, the only tricky thing is the regrouping. If either the left or right columns of a square are in groups that are not too large, the square will get found when that column group gets iterated. Otherwise both its top-left and top-right corners get regrouped, placed into the same row group, and the square will be found when that row group gets iterated.

49. What do you think the output of this code will be?

```
enum Day {
 MONDAY,TUESDAY,WEDNESDAY,THURSDAY,FRIDAY,SATURDAY,SUNDAY
}
public class BuggyBread1{
   public static void main (String args[]) {
    Set mySet = new TreeSet();
    mySet.add(Day.SATURDAY);
    mySet.add(Day.WEDNESDAY);
    mySet.add(Day.FRIDAY);
    mySet.add(Day.WEDNESDAY);
    for(Day d: mySet){
     System.out.println(d);
 }
   }
}
```

Your answer: WEDNESDAY

FRIDAY

SATURDAY

Only one FRIDAY will be printed since the Set doesn't allow duplicated.

Elements will be printed in the order that the constants are declared in in the Enum. TreeSet maintains the elements in the ascending order which is identified by the defined compareTo method. The compareTo method in the Enum has been defined such that the constant declared later are greater than the constants declared prior.

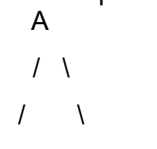
50.

51. Construct Tree from given Inorder and Preorder traversals- MetLife

Construct tree from Inorder and Preorder

Let us consider the below traversals:

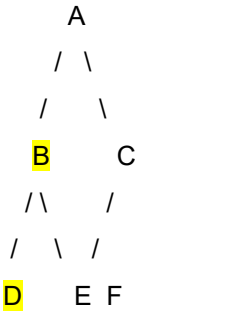
Inorder sequence: D B E A F C



Preorder sequence: A B D E C F

In a Preorder sequence, leftmost element is the root of the tree. So we know 'A' is root for given sequences. By searching 'A' in Inorder sequence, we can find out all elements on left side of 'A' are in left subtree and elements on right are in right subtree. So we know below structure now.

We recursively follow above steps and get the following tree.



Algorithm: buildTree()

1) Pick an element from Preorder. Increment a Preorder Index Variable (preIndex in below code) to pick next element in next recursive call.

2) Create a new tree node tNode with the data as picked element.

3) Find the picked element's index in Inorder. Let the index be inIndex.

4) Call buildTree for elements before inIndex and make the built tree as left subtree of tNode.

5) Call buildTree for elements after inIndex and make the built tree as right subtree of tNode.

6) return tNode.

52. Serialization - MetLife

Serializable is a marker **INTERFACE** with no methods defined it

Class

**How to make a Java class Serializable

Class needs to implement java.io.Serializable **INTERFACE** and JVM will take care of serializing the object.

Reference how serialization put constraints on your ability to change class is SerialVersionUID.

Object

 process of making the object's state persistent. That means the state of the object is converted into a stream of bytes and stored in a file.

 Mostly used in the networking programming. The objects that need to be transmitted through the network have to be dconverted into bytes.

```
class GalleryImage implements Serializable
```

53. Could you provide some implementation of a Dictionary having a large number of words?

Your answer: The simplest implementation that can be given is that of a List wherein one can place ordered words and perform a Binary search. The other implementation with a better search performance is HashMap where the key is used as the first character of the word and the value as a LinkedList.
Up another level, there are HashMaps like:

```
hashmap {
a ( key ) -> hashmap (key-aa , value (hashmap(key-aaa,value)
b ( key ) -> hashmap (key-ba , value (hashmap(key-baa,value)
z( key ) -> hashmap (key-za , value (hashmap(key-zaa,value)
}
```

Up to n levels where n is the average size of the word in the dictionary.

Java Puzzles

54. Dead code and unreachable code in java [puzzle]

November 28, 2012 by Lokesh Gupta

For this puzzle, I have given a piece of code below. Try to identify the problems in code, if it is compiled in eclipse IDE.

```
public class IdentifyProblemsInCode {
    public void howToDoInJava_method1() {
        System.out.println("how to do");
        return;
        System.out.println("in java");
    }
    public void howToDoInJava_method2() {
        System.out.println("how to do");
        if (true) {
            return;
        }
        System.out.println("in java");
    }
    public void howToDoInJava_method3() {
        System.out.println("how to do");
        while (true) {
            return;
        }
        System.out.println("in java");
    }
}
```

I am giving the answer of above puzzle in next section, but I would recommend you to try first yourself. Its for fun only.

Solution:

We all must have been faced compilation **errors** related to "**Unreachable code**" and some may have noticed "**dead code warning**". Above puzzle is related to them only.

In first method i.e. *howToDoInJava_method1()*, **second print statement is unreachable**, so compiler will complain for oblivious reasons.

In second method *howToDoInJava_method2(),* **second print statement is also unreachable, but strange compiler only warns you**. We will later try to get the logic here.

In third method also i.e. *howToDoInJava_method3()*, **second print statement is unreachable**, so compiler will complain again. Strange !!

Reasoning

The unreachable code in method 2 is called “Dead code”. This is purely Eclipse compiler reported error, and if you will compile above class with “***javac***“, java inbuilt compiler will only complain for other two methods. [First and Third].

Quote from java language specification :

“*The idea is that there must be some possible execution path from the beginning of the constructor, method, instance initializer or static initializer that contains the statement to the statement itself. The analysis takes into account the structure of statements. Except for the* ***special treatment of while, do, and for statements whose condition expression has the constant value true****, the values of expressions are not taken into account in the flow analysis*.”

What that means, is that the `if` block is not taken into account, since if you go through one of the paths of the `if` statement, you could reach second print statement. It all depends on compiler which determines this during compile time.

In other two statement, compiler has determined the un-reachability so it complains with error.

If we re-write second method again like this:

```
public void howToDoInJava_method2() {
        System.out.println("how to do");
        if (true) {
            return;
        }
        else
         {
             return;
          }
        System.out.println("in java");
    }
```

Now, compiler determine that in no way it can reach to last print statement, so it again reports unreachable code.

If you still have some doubts, please write down in comment section. I will try to resolve your queries.

55. How to create an instance of any class without using new keyword

November 28, 2012 by Lokesh Gupta

Using Class.forName()

ClassLoader loadClass()

Using clone()

Deserialization

Using reflection

Note: *I am writing pseudo code only. For building complete fully working sample code, please read about related feature.*

Using newInstance method of Class class

```
Class ref = Class.forName("DemoClass");
DemoClass obj = (DemoClass) ref.newInstance();
```

Class.forName() loads the class in memory. To create an instance of this class, we need to use newInstance().

Using class loader’s loadClass()

Just like above mechanism, class loader’s loadClass() method does the same thing.

```
instance.getClass().getClassLoader().loadClass("NewClass").newInstance();
```

Using clone() of java.lang.Object

This is also a way to have a new independent instance of a class.

```
NewClass obj = new NewClass();
NewClass obj2 = (NewClass) obj.clone();
```

Using Object serialization and deserialization

If you have gone through this article, you can understand that serialization and de-serialization is also a way to have another instance of a class in system.

```
ObjectInputStream objStream = new ObjectInputStream(inputStream);
 NewClass obj = (NewClass ) inStream.readObject();
```

Using Reflection

This is also a popular way to create new instances in most of available frameworks.

`constructor.newInstance(); or`

`class.newInstance();`

c) Using object deserialization

```
ObjectInputStream inStream = new ObjectInputStream(anInputStream );

MyClass object = (MyClass) inStream.readObject();
```

d) Creating string and array objects :

```
String s = "string object";

int[] a = {1, 2, 3, 4};
```

56. Java Check Palindrome Number Example

January 1, 2013 by Lokesh Gupta

A **palindrome** is a word, phrase, number, or other sequence of units that may be read the same way in either direction, generally if used comma, separators or other word dividers are ignored. [*Not mandatory*]

Similarly, palindrome numbers are those numbers which represent same number if all digits are reversed (Underscores can be ignored in large numbers such as 1_00_00_0. *Underscores are new addition in java 7 features*.

To verify, if a given number is palindrome is or not, we need to reverse the number and compare with original number if both are equal or not.

Lets write some code to solve this.

```
package com.howtodoinjava.puzzle;
public class PalindromeTest
{
    /**
     * Test the actual code if it works correctly
     * */
    public static void main(String[] args)
    {
        System.out.println(checkIntegerPalindrome( 100 )); //false
        System.out.println(checkIntegerPalindrome( 101 )); //true
        System.out.println(checkIntegerPalindrome( 500045 )); //false
        System.out.println(checkIntegerPalindrome( 50005 )); //true
    }
    /**
     * This function will test the equality if a number and its reverse.
     * @return true if number is palindrome else false
     * */
    public static boolean checkIntegerPalindrome(int number)
    {
        boolean isPalindrome = false;
        if(number == reverse(number))
        {
            isPalindrome = true;
        }
        return isPalindrome;
    }
    /**
     * This function will reverse a given number.
     * @return reverse number
     * */
    public static int reverse(int number)
    {
        int reverse = 0;
        int remainder = 0;
        do {
            remainder = number % 10;
            reverse = reverse * 10 + remainder;
```

```
            number = number / 10;
        } while (number > ;
        return reverse;
    }
}
Output:
```

```
false
true
false
true
```

October 20, 2015 by Lokesh Gupta

57. Puzzle : I have given a below map with the following below options,

```
Map map = new TreeMap();
map.put("test key 1", "test value 1");
map.put("test key 2", "test value 2");
map.put("test key 3", "test value 3");

System.out.println(map.put("test key 3", "test value 3"));
System.out.println(map.put("test key 4", "test value 4"));
```

Option A) System.out.println(map.put("test key 3", "test value 3"));**Answer)** This prints the output as = test value **Option B)** System.out.println(map.put("test key 4", "test value 4"));**Answer)** This prints the output as = null;

Solution : If you look at `Map.put()` operation, it **returns the value if key is already present in map**.

After adding key "test key 3", when again try to add it, it returns the value "test value 3".

When you add "test key 4" first time, it is no present in map, so map return's it's value is null.

Next time when you store "test key 4", this time entry is already present so value is returned as "test value 4"

58. Java puzzle – Find all the distinct duplicate elements

October 20, 2015 by Lokesh Gupta

Puzzle: Given an input array of n positive integers where the integers are in random order. Each number in that array can occur many times. You need to find all the distinct duplicate elements and put all those elements in an array i.e. output1. If no number is duplicate in input1 , then output1 should be {-1}.

Input Specifications: input1: number of elements in input2 (n) input 2: an array of n positive integers

Output Specifications: output1: array of distinct elements which are duplicate in input`Example 1: input1 : 6 input2 : {4,4,7,8,8,9} output1 : {4,8}`

`Example 2: input1 : 8 input2 : {2,3,6,8,90,58,58,60} output1 : {58}`

`Example 3: input1 : 7 input2 : {3,6,5,7,8,19,32} output1 : {-1}`

Solution

```
import java.util.HashSet;
import java.util.Set;

public class DuplicatesInArray
{
    public static void main(String[] args)
    {
        Integer[] array = {1,2,3,4,5,6,7,8};  //input          int size =
array.length;               //input
        Set<Integer> set = new HashSet<Integer>();
        Set<Integer> duplicates = new HashSet<Integer>();

        for(int i = 0; i < size ; i++)
        {
            if(set.add(array[i]) == false)
```

```
            {
                duplicates.add(array[i]);
            }
        }

        if(duplicates.size() ==
        {
            duplicates.add(-;
        }

        System.out.println(duplicates);
    }
}
```

59. Java puzzle : Good String – Bad String

October 20, 2015 by Lokesh Gupta

Puzzle: does not like strings which have same consecutive letters. Given a string S, you need to convert it into a Good String.

Solution

regex

```
public class GoodStringBadString
{
   public static void main(String[] args)
   {
      String input = "Good Oops, Bad Oops";
      String output = input.replaceAll("(?i)(\\p{L})\\1", "$1");
      System.out.println(output);
   }
}
```

```
Output: God Ops, Bad Ops
```

60. Puzzle – Check if string is complete (contains all alphabets)

October 20, 2015 by Lokesh Gupta

A string is said to be complete if it contains all the characters from a to z. Given a string, check if it complete or not. e.g.

Sample Input

```
wyyga
qwertyuioplkjhgfdsazxcvbnm
ejuxggfsts
```

Sample Output

```
NO
YES
NO
```

61. Solution using for loop and indexOf()

I wrote a quick function which is able to find this complete string.

```
package com.howtodoinjava.examples;

public class CheckAllAlphabetsAlgorithms
{
    public static void main(String[] args)
    {
        System.out.println( checkAllChars( "qwertyuioplkjhgfdsAzxcvbnm" ) );
```

```
        System.out.println( checkAllChars( "123" ) );
        System.out.println( checkAllChars( "ejuxggfsts" ) );
        System.out.println( checkAllChars( "wyyga" ) );
    }

    private static String checkAllChars ( String input )
    {
        //If input length is less than 26 then it can never be complete
        if(input.length() <
        {
            return "FALSE";
        }

        for (char ch = 'A'; ch <= 'Z'; ch++)
        {
            if (input.indexOf(ch) < 0 && input.indexOf((char) (ch + ) <
            {
                return "FALSE";
            }
        }
        return "TRUE";
    }
}
```

Output:

```
TRUE
FALSE
FALSE
FALSE
```

62. Puzzle – Return all the strings with the Nth longest length

October 20, 2015 by Lokesh Gupta

Algorithm: given a list of strings, return all the strings with the **nth longest length** in that list for example: `list - Yuri, Ron, Interview, Longest, List, Contain` and **nth = 1** will return just "**Interview**" whereas **nth = 2** will return both "**Longest**" and "**Contain**".

Though solution to "How to find the kth largest element in an unsorted array of length n in O(n)?" can be applied to string length, how to translate back to print all the strings with n length?

Solution

I have written a simple java program which is able to find "all Nth longest elements" from a list of strings. Program is as below:

```
package com.howtodoinjava.examples;

import java.util.ArrayList;
import java.util.List;
import java.util.TreeMap;

public class NthLogestStringAlgorithm
{
    public static void main(String[] args)
    {
        int n = 0;
        List<String> list = new ArrayList<String>();
        list.add("Yuri");
        list.add("Ron");
        list.add("Interview");
        list.add("Longest");
        list.add("List");
        list.add("Contain");

        System.out.println( findNthLongestElement(list, n) );
    }
```

```
    private static List<String> findNthLongestElement(List<String> list, int n)
    {
        if(n <  {
            return null; //Handle invalid case
        }

        TreeMap<Integer, List<String>> map = new TreeMap<>();

        for(String str : list)
        {
            Integer length = str.length();
            List<String> tempList = map.get(length) != null ? map.get(length) : new
ArrayList<String>();
            tempList.add(str);
            map.put(length, tempList);
        }
        return map.get( map.descendingKeySet().toArray()[n-1] );
    }
}
```

```
========Output of program=======
n = 0 => null
n = 1 => [Interview]
n = 2 => [Longest, Contain]
n = 3 => [Yuri, List]
n = 4 => [Ron]
```

How to Reverse String in Java

April 4, 2016 by Lokesh Gupta

It's common puzzle asked in java interviews at beginner level. Let's memorize their solutions for quick recall.

63. Reverse String by Characters

You can reverse a string by character easily, using a `StringBuilder.reverse()` method.

```
String sh = "HowToDoInJava.com";
System.out.println(sh + " -> " + new StringBuilder(sh).reverse( ));

Output:

HowToDoInJava.com -> moc.avaJnIoDoTwoH
```

64. Reverse String by Words

While reversing string content by words, most natural way is to use a `StringTokenizer` and a `Stack`. As you are aware that `Stack` is a class that implements an easy-to-use last-in, first-out (LIFO) stack of objects.

```
String s = "Java technology blog for smart java concepts and coding practices";

// Put words from String in Stack
Stack<String> myStack = new Stack<>();
StringTokenizer st = new StringTokenizer(s," ");

while (st.hasMoreTokens()) {
    myStack.push(st.nextToken());
}

// Build the reverse string
StringBuilder reverseString = new StringBuilder();
```

```
while (!myStack.empty()) {
    reverseString.append(myStack.pop() + " ");
}

System.out.println(reverseString.toString());

Output:

practices coding and concepts java smart for blog technology Java
```

65. Java Puzzle – Find Missing Number From Series

August 3, 2016 by Lokesh Gupta

This puzzle has been asked in Amazon.com. In this java puzzle, you have a series of numbers start (e.g. 1….N) and exactly one number in this series is missing. You have to write a java program to find missing number from series.

Solution

Surprisingly, solution of this puzzle is very simple only if you know it already.

Calculate `A = n (n+/2` where n is largest number in series 1…N.

Calculate B = Sum of all numbers in given series

Missing number = A – B

Let's a write the solution in code.

```
public class FindMissingNumber {
    public static void main(String[] args) {
        //10 is missing
        int[] numbers = {1,2,3,4,5,6,7,8,9, 11,12};

        int N = 12;
        int idealSum = (N * (N + ) / 2;
        int sum = calculateSum(numbers);

        int missingNumber = idealSum - sum;
        System.out.println(missingNumber);
    }

    private static int calculateSum(int[] numbers) {
        int sum = 0;
        for (int n : numbers) {
            sum += n;
        }
        return sum;
    }
}

Output:
```

Solution in Java

Above code, though simple, can be reduced by many lines using new language features such as lambda in java 8. Let's see how?

```
import java.util.Arrays;

public class FindMissingNumber {
    public static void main(String[] args) {
        //10 is missing
        int[] numbers = {1,2,3,4,5,6,7,8,9, 11,12};

        int N = 12;
        int idealSum = (N * (N + ) / 2;
        int sum = Arrays.stream(numbers).sum();

        int missingNumber = idealSum - sum;
        System.out.println(missingNumber);
    }
}

Output:
```

Puzzles like these are simple to solve, but it is always useful to know the solution before it is asked in any interview.

n this post, I have collected some java interview questions where you are asked to write a program or you have been told to guess the output of a given program. I have also tried to answer them. I hope it will be helpful for you.

66. FizzBuzz Problem.

Write a program in java which prints the numbers from 1 to 100. But, multiples of 3 should be replaced with “Fizz”, multiples of 5 should be replaced with “Buzz” and multiples of both 3 and 5 should be replaced with “FizzBuzz”?

```
public class FizzBuzzProblem
{
    public static void main(String args[])
    {
        for(int i = 1; i <= 100; i++)
        {
            if((i % (3*) ==
            {
                System.out.println("FizzBuzz");
            }
            else if ((i %  ==
            {
                System.out.println("Buzz");
              }
            else if ((i %  ==
            {
                System.out.println("Fizz");
            }
            else
            {
                System.out.println(i);
              }
        }
    }
}
```

67. Write a Java program to find out the nearest number to 100 of the given two numbers?

```
public class NearestNumber
{
    static int nearestTo100(int input1, int input
    {
        int diff1 = Math.abs(100 - input;

        int diff2 = Math.abs(100 - input;

        if(diff1 < diff
        {
            return input1;
        }
        else if (diff2 < diff
        {
            return input2;
        }
        else
        {
            return input1;
        }
    }

    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter First Number");
```

```
        int input1 = sc.nextInt();

        System.out.println("Enter Second Number");
        int input2 = sc.nextInt();

        System.out.println(nearestTo100(input1, input);
    }
}
```

68. There are two numbers ‘a’ and ‘b’. Write a java program which should print ‘a’ if ‘a’ is bigger than ‘b’ by 2 or more or should print ‘b’ if ‘b’ is bigger than ‘a’ by 2 or more. Otherwise, it should print “INCONCLUSIVE”?

```
public class BiggerNumber
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter First number");

        int a = sc.nextInt();

        System.out.println("Enter Second Number");

        int b = sc.nextInt();

        if((a > b) && (a - b) >=
        {
            System.out.println(a);
        }
        else if ((b > a) && (b - a) >=
        {
            System.out.println(b);
        }
        else
        {
            System.out.println("INCONCLUSIVE");
        }
    }
}
```

69. Given a string “JAVAJ2EE”, write java program to print this string in the below format?

JJ AJ A VJ A V AJ A V A JJ A V A J 2J A V A J 2 EJ A V A J 2 E E

```
public class PromrammingExample
{
    public static void main(String[] args)
    {
        String s = "JAVAJ2EE";

        char[] c = s.toCharArray();

        for(int i = 0; i < c.length; i++)
        {
            for(int j = 0; j <= i; j++)
            {
                System.out.print(c[j]+" ");
            }
            System.out.println();
        }
    }
```

```
}
```

70. How do you prove that String s1 = new String("ONE") and String s2 = "ONE" are two different objects in the memory?

Ans : Use "==" operator to compare s1 and s2. It will return 'false' if they are two different objects in the memory.

```
public class ProgrammingExamples
{
    public static void main(String[] args)
    {
        String s1 = "ONE";

        String s2 = new String("ONE");

        System.out.println(s1 == s;      //Output : false
    }
}
```

71. What will be the output of this program?

```
public class CodingExamples
{
    public static void main(String args[])
    {
        int i = (byte) + (char) - (int) + (long) - 1;

        System.out.println(i);
    }
}
```

72. Ans : Will this program compiles successfully?

```
public class CodingExamples
{
    public static void main(String args[])
    {
        System.out.println(;

         http://javaconceptoftheday.com/

        System.out.println(;
    }
}
```

Ans : Yes. Compiler will treat 'http' as label and remaining part (of Line will be commented.

73. What will be the output of the below program?

```
public class CodingExamples
{
    public static void main(String[] args)
    {
        String[] s1 = {"ONE", "TWO", "THREE", "FOUR"};

        String[] s2 = {"THREE", "TWO", new String("ONE")};
```

```
        System.out.println(s1[0] == s2[2]);

        System.out.println(s1[1] == s2[1]);

        System.out.println(s1[2] == s2[0]);
    }
}
```

Ans :falsetruetrue

74. Open a notepad in your system through a java program?

```
public class ClassesAndObjects
{
    public static void main(String[] args)
    {
        try
        {
            Runtime.getRuntime().exec("notepad.exe");
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

75. Write a Java program to print the current date in “dd MMM yyyy” format?

```
public class CurrentDate
{
    public static void main(String[] args)
    {
        Date date = new Date();

        SimpleDateFormat formatter = new SimpleDateFormat("dd MMM yyyy");

        System.out.println(formatter.format(date));
    }
}
```

---

76. Program: Find out duplicate number between 1 to N numbers.

Description:

You have got a range of numbers between 1 to N, where one of the number is repeated. You need to write a program to find out the duplicate number.

```
package com.java2novice.algos;

import java.util.ArrayList;
import java.util.List;

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));
    }
}
```

Output:

Duplicate Number:

77. How to reverse Singly Linked List?

Description:

Write a sample code to reverse Singly Linked List by iterating through it only once.

Recursive Method:

1) Divide the list in two parts - first node and rest of the linked list.
2) Call reverse for the rest of the linked list.
3) Link rest to first.
4) Fix head pointer.

```
package com.java2novice.ds.linkedlist;

public class SinglyLinkedListImpl<T> {

    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;
    }

    public static void main(String a[]){
        SinglyLinkedListImpl<Integer> sl = new SinglyLinkedListImpl<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();
    }
}

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;
        }
    }
}
```

Output:

```
Adding: Adding: Adding: Adding:
3       32      54      89

reversing the linked list

89      54      32      3
```

78. Program: Find out middle index where sum of both ends are equal.

Description:

```
You are given an array of numbers. Find out the array index or position
where sum of numbers preceeding the index is equals to sum of numbers
succeeding the index.
```

```
package com.java2novice.algos;

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());
        }
    }
}
```

Output:

```
Starting from index 0, adding numbers till index 2 and
adding rest of the numbers can be equal
```

79. Program: Write a singleton class.

Description:

Singleton class means you can create only one object for the given class. You can 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, wherein you can handle the object creation inside the class only. In this example we are creating object by using static block.

```
package com.java2novice.algos;

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();
    }
}
```

80. Program: Write a program to create deadlock between two threads.

Description:

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other. Deadlocks can occur in Java when the synchronized keyword causes the executing thread to block while waiting to get the lock, associated with the specified object. Since the thread might already hold locks associated with other objects, two threads could each be waiting for the other to release a lock. In such case, they will end up waiting forever.

```
package com.java2novice.algos;

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();
    }
}
```

81. Program: Write a program to reverse a string using recursive algorithm.

Description:

Write a program to reverse a string using recursive methods.

You should not use any string reverse methods to do this.

```
package com.java2novice.algos;

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"));
    }
}
```

Output:

```
Result: ecivon2avaJ
```

82. Program: Write a program to reverse a number.

Description:

Write a program to reverse a number using numeric operations. Below example shows how to reverse a number using numeric operations.

```
package com.java2novice.algos;

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));
    }
}
```

Output:

Result: 86Program:

83. Write a program to convert decimal number to binary format.

Description:

Write a program to convert decimal number to binary format using numeric operations.

Below example shows how to convert decimal number to binary format using numeric operations.

```
package com.java2novice.algos;

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);
    }
}
```

Output:

84. Program: Write a program to find perfect number or not.

Description:

```
A perfect number is a positive integer that is equal to the sum
of its proper positive divisors, that is, the sum of its positive
divisors excluding the number itself. Equivalently, a perfect number
is a number that is half the sum of all of its positive divisors.
The first perfect number is 6, because 1, 2 and 3 are its proper
positive divisors, and 1 + 2 + 3 = 6. Equivalently, the number is equal to half the sum
of all its positive divisors:
               ( 1 + 2 + 3 + 6 ) / 2 = 6.
```

```
package com.java2novice.algos;

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));
    }
}
```

Output:

```
It is a perfect number
Is perfect number: true
```

85. Program: Write a program to implement ArrayList.

Description:

```
Write a program to implement your own ArrayList class. It should
contain add(), get(), remove(), size() methods. Use dynamic array logic.
It should increase its size when it reaches threshold.
```

```
package com.java2novice.algos;

import java.util.Arrays;

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)+" ");
        }
    }
}
```

Output:

```
2 5 1 23 14
New length: Element at Index 5:List size: Removing element at index 2: 2 5 23 14 29
```

86. Program: Write a program to find maximum repeated words from a file.

Write a program to read words from a file. Count the repeated or duplicated words. Sort it by maximum repeated or duplicated word count.

```
Code:
package com.java2novice.algos;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;
import java.util.Map.Entry;

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());
        }
    }
}
```

```
Output:
```

one ==== the ==== that ==== of ==== in ==== some ==== to ==== summary ==== but ==== have ==== common ==== least ==== simplest ====

87. Program: Write a program to find out duplicate characters in a string.

Description:

Write a program to find out duplicate or repeated characters in a string, and calculate the count of repeatation.

```
package com.java2novice.algos;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

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");
    }
```

```
}
```

Output:

v--->a--->

88. Program: Write a program to find top two maximum numbers in a array.

Description:

Write a program to find top two maximum numbers in the given array. You should not use any sorting functions. You should iterate the array only once. You should not use any kind of collections in java.

```
package com.java2novice.algos;

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);
    }
}
```

Output:

First Max Number: Second Max Number:

89. Program: Write a program to sort a map by value.

Description:

Sort or order a HashMap or TreeSet or any map item by value. Write a comparator which compares by value, not by key. Entry class might hleps you here.

```
package com.java2novice.algos;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Map.Entry;

public class OrderByValue {

    public static void main(String a[]){
        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());
        }
    }
}
```

Output:

Why this kolavari ==== Unix ==== C++ ==== MAC ==== java ==== Java2Novice ====

90. Program: Write a program to find common elements between two arrays.

Description:

```
Write a program to identify common elements or numbers between
two given arrays. You should not use any inbuilt methods are list to
find common values.
```

```
package com.java2novice.algos;

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]);
                }
            }
        }
    }
}
```

Output:

91. Program: How to swap two numbers without using temporary variable?

Description:

Write a program to swap or exchange two numbers. You should not use any temporary or third variable to swap.

```
package com.java2novice.algos;

public class MySwapingTwoNumbers {

    public static void main(String a[]){
        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);
    }
}
```

Output:

```
Before swap:
x value: y value: After swap:
x value: y value:
```

92. Program: Write a program to print fibonacci series.

Description:

In mathematics, the Fibonacci numbers or Fibonacci series or Fibonacci sequence are the numbers in the following integer sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144... By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two. Below example shows how to create fibonacci series.

```
package com.java2novice.algos;
```

```
public class MyFibonacci {

    public static void main(String a[]){

         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] + " ");
         }
    }
}
```

Output:

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

93. Program: Write a program to find sum of each digit in the given number using recursion.

Description:

Below example shows how to find out sum of each digit in the given number using recursion logic. For example, if the number is 259, then the sum should be 2+5+9 = 16.

```
package com.java2novice.algos;

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));
    }
}
```

Output:

```
52
```

Sum is:

94. Program: Write a program to check the given number is a prime number or not?

Description:

A prime number (or a prime) is a natural number greater than 1 that has no positive divisors other than 1 and itself. A natural number greater than 1 that is not a prime number is called a composite number. For example, 5 is prime, as only 1 and 5 divide it, whereas 6 is composite, since it has the divisors 2 and 3 in addition to 1 and 6. The fundamental theorem of arithmetic establishes the central role of primes in number theory: any integer greater than 1 can be expressed as a product of primes that is unique up to ordering. This theorem requires excluding 1 as a prime.

```
package com.java2novice.algos;

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));
        System.out.println("Is 19 prime number? "+mpc.isPrimeNumber(19));
        System.out.println("Is 15 prime number? "+mpc.isPrimeNumber(15));
    }
}
```

Output:

```
Is 17 prime number? true
Is 19 prime number? true
Is 15 prime number? false
```

95. Program: Write a program to find the given number is Armstrong number or not?

Description:

```
Armstrong numbers are the sum of their own digits to the power of
the number of digits. It is also known as narcissistic numbers.
```

```
package com.java2novice.algos;

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));
        System.out.println("Is 523 Armstrong number? "+man.isArmstrongNumber(523));
        System.out.println("Is 153 Armstrong number? "+man.isArmstrongNumber(153));
    }
}
```

Output:

```
Is 371 Armstrong number? true
Is 523 Armstrong number? false
Is 153 Armstrong number? true
```

96. Program: Write a program to convert binary to decimal number.

Description:

Write a program to convert binary format to decimal number using numeric operations. Below example shows how to convert binary to decimal format using numeric operations.

```
package com.java2novice.algos;

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));
        System.out.println("110 ===> "+bd.getDecimalFromBinary(110));
        System.out.println("100110 ===> "+bd.getDecimalFromBinary(100110));
    }
}
```

Output:

11 ===> 110 ===> 100110 ===>

97. Program: Write a program to check the given number is binary number or not?

Description:

The binary numeral system, or base-2 number system, represents numeric values using two symbols: 0 and 1. More specifically, the usual base-2 system is a positional notation with a radix of 2. Because of its straightforward implementation in digital electronic circuitry using logic gates, the binary system is used internally by almost all modern computers.

```
package com.java2novice.algos;

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));
        System.out.println("Is 10300111 binary? :"+mbc.isBinaryNumber(10300111));
    }
}
```

Output:

```
Is 1000111 binary? :true
Is 10300111 binary? :false
```

98. Program: Write a program for Bubble Sort in java.

Description:

Bubble sort is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the 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. You can see the code implementation below:

```
package com.java2novice.algos;

public class MyBubbleSort {

    // logic to sort the elements
    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);

    }
}
```

Output:

```
2, 4, 6, 9, 12, 23, 0, 1, 34,

2, 4, 6, 9, 12, 0, 1, 23, 34,

2, 4, 6, 9, 0, 1, 12, 23, 34,

2, 4, 6, 0, 1, 9, 12, 23, 34,

2, 4, 0, 1, 6, 9, 12, 23, 34,

2, 0, 1, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,
```

99. Program: Write a program for Insertion Sort in java.

Description:

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 as 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, until no input elements remain. The choice of which element to remove from the input is arbitrary, and can be made using almost any choice algorithm. You can see the code implementation below:

```
package com.java2novice.algos;

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);
        }
    }
}
```

Output:

```
2, 4, 9, 6, 23, 12, 34, 0, 1,

2, 4, 9, 6, 23, 12, 34, 0, 1,

2, 4, 6, 9, 23, 12, 34, 0, 1,

2, 4, 6, 9, 23, 12, 34, 0, 1,

2, 4, 6, 9, 12, 23, 34, 0, 1,

2, 4, 6, 9, 12, 23, 34, 0, 1,

0, 2, 4, 6, 9, 12, 23, 34, 1,

0, 1, 2, 4, 6, 9, 12, 23, 34,
```

100. Program: Write a program to implement hashcode and equals.

Description:

The hashcode of a Java Object is simply a number, it is 32-bit signed int, that allows an object to be managed by a hash-based data structure. We know that hash code is an unique id number allocated to an object by JVM. But actually speaking, Hash code is not an unique number for an object. If two objects are equals then these two objects should return same hash code. So we have to implement hashcode() method of a class in such way that if two objects are equals, ie compared by equal() method of that class, then those two objects must return same hash code. If you are overriding hashCode you need to override equals method also. The below example shows how to override equals and hashcode methods. The class Price overrides equals and hashcode. If you notice the hashcode

implementation, it always generates unique hashcode for each object based on their state, ie if the object state is same, then you will get same hashcode. A HashMap is used in the example to store Price objects as keys. It shows though we generate different objects, but if state is same, still we can use this as key.

```
package com.java2novice.algos;

import java.util.HashMap;

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;
    }
}
```

Output:

```
In hashcode
In hashcode
In hashcode
In hashcode
Hashcode of the key: 19824796In hashcode
In equals
Value from map: Banana
```

101. Program: How to get distinct elements from an array by avoiding duplicate elements?

Description:

The below example shows how to avoid duplicate elements from an array and disply only distinct elements. Please use only arrays to process it.

```
package com.java2novice.algos;

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);
    }
}
```

Output:

```
5 2 7 4 8 3
```

102. Program: Write a program to get distinct word list from the given file.

Description:

Write a program to find all distinct words from the given file. Remove special chars like ".,;:" etc. Ignore case sensitivity.

```
package com.java2novice.algos;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

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);
        }
    }
```

```
}
```

Output:

```
the
while
statement
verifies
condition
before
entering
into
loop
to
see
whether
next
iteration
should
occur
or
not
do-while
executes
first
without
checking
it
after
finishing
each
will
always
execute
body
of
a
at
least
once
```

103. Program: Write a program to get a line with max word count from the given file.

Description:

Below example shows how to find out the line with maximum number of word count in the given file. In case if it has multiple lines with max number of words, then it has to list all those lines.

```
package com.java2novice.algos;

import java.io.BufferedReader;
import java.io.DataInputStream;
```

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

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);
        }
    }
}
```

MyTestFile.txt:

```
true, false, and null might seem like keywords, but they are actually literals.
You cannot use them as identifiers in your programs. The servlet context
is an interface which helps to communicate with other servlets. It contains
information about the Web application and container. It is kind of
application environment. Using the context, a servlet can obtain URL
references to resources, and store attributes that other servlets in the
context can use.
```

Output:

```
Max number of words in a line is: Line with max word count:
true, false, and null might seem like keywords, but they are actually literals.
```

104. Program: Write a program to convert string to number without using Integer.parseInt() method.

Description:

Below example shows how to convert string format of a number to number without calling Integer.parseInt() method. We can do this by converting each character into ascii format and form the number.

```
package com.java2novice.algos;

public class MyStringToNumber {

    public static int convert_String_To_Number(String numStr){

        char ch[] = numStr.toCharArray();
        int sum = 0;
        //get ascii value for zero
        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"));
        System.out.println("\"76289\" == "+convert_String_To_Number("76289"));
        System.out.println("\"90087\" == "+convert_String_To_Number("90087"));
    }
}
```

Output:

"3256" == 32"76289" == 762"90087" == 90Program:

105. Write a program to find two lines with max characters in descending order.

Description:

Write a program to read a multiple line text file and write the 'N' longest lines to the output console, where the file to be read is specified as command line aruguments. The program should read an input file. The first line should contain the value of the number 'N' followed by multiple lines. 'N' should be a valid positive integer.

```
package com.longest.lines;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Comparator;
import java.util.Set;
import java.util.TreeSet;

public class Main {

    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) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            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;
            }
        }

    }
}
```

Sample input file:
Java2novice
My Test line Java world
I know java language

This is a test program
java is simple

Output:
This is a test program
I know java language
My Test line

106. Program: Write a program to find the sum of the first 1000 prime numbers.

Description:
Write a program to find the sum of the first 1000 prime numbers.

```
package com.primesum;

public class Main {

    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;
    }
}
```

Output:

3682<<

107. Program: Find longest substring without repeating characters.

Description:

Given a string, find the longest substrings without repeating characters. Iterate through the given string, find the longest maximum substrings.

```
package com.java2novice.algos;

import java.util.HashSet;
import java.util.Set;

public class MyLongestSubstr {

    private Set<String> subStrList = new HashSet<String>();
    private int finalSubStrSize = 0;

    public Set<String> getLongestSubstr(String input){
        //reset instance variables
        subStrList.clear();
        finalSubStrSize = 0;
        // have a boolean flag on each character ascii value
        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"));
        System.out.println(mls.getLongestSubstr("java_language_is_sweet"));
        System.out.println(mls.getLongestSubstr("java_java_java_java"));
        System.out.println(mls.getLongestSubstr("abcabcbb"));
    }
}
```

Output:

[a2novice]

[uage_is]

[_jav, va_j]

[cab, abc, bca]

108. Program: Write a program to remove duplicates from sorted array.

Description:

Given array is already sorted, and it has duplicate elements. Write a program to remove duplicate elements and return new array without any duplicate elements. The array should contain only unique elements.

```
package com.java2novice.algos;

public class MyDuplicateElements {

    public static int[] removeDuplicates(int[] input){

        int j = 0;
```

```
        int i = 1;
        //return if the array length is less than         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+" ");
        }
    }
}
```

Output:

2 3 6 8 9 10 12

109. Program: How to sort a Stack using a temporary Stack?

Description:

You have a stack with full of integers. Sort it in the ascending order using another temporary array by using all stack functionality.

```
package com.java2novice.algo;

import java.util.Stack;

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));
    }
}
```

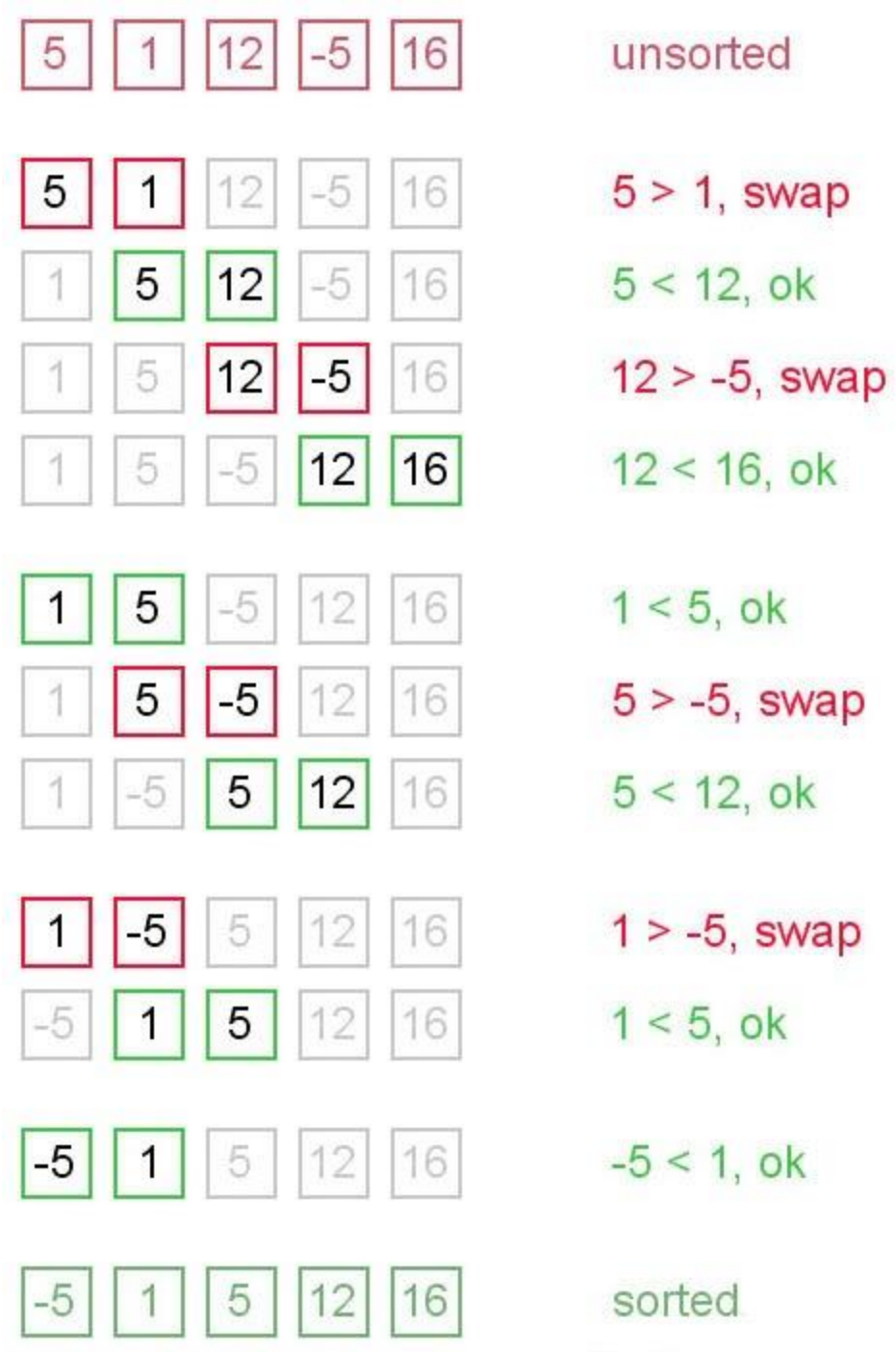
Output:

input: [34, 3, 31, 98, 92, 23]
============== debug logs ================
Element taken out: input: [34, 3, 31, 98, 92]
tmpStack: [23]
Element taken out: input: [34, 3, 31, 98]
tmpStack: [23, 92]
Element taken out: input: [34, 3, 31]
tmpStack: [23, 92, 98]
Element taken out: input: [34, 3, 98, 92]
tmpStack: [23, 31]
Element taken out: input: [34, 3, 98]
tmpStack: [23, 31, 92]
Element taken out: input: [34, 3]
tmpStack: [23, 31, 92, 98]
Element taken out: input: [34, 98, 92, 31, 23]
tmpStack: [3]
Element taken out: input: [34, 98, 92, 31]
tmpStack: [3, 23]
Element taken out: input: [34, 98, 92]
tmpStack: [3, 23, 31]
Element taken out: input: [34, 98]
tmpStack: [3, 23, 31, 92]
Element taken out: input: [34]
tmpStack: [3, 23, 31, 92, 98]
Element taken out: input: [98, 92]
tmpStack: [3, 23, 31, 34]
Element taken out: input: [98]
tmpStack: [3, 23, 31, 34, 92]
Element taken out: input: []
tmpStack: [3, 23, 31, 34, 92, 98]

========== debug logs ended ==========
final sorted list: [3, 23, 31, 34, 92, 98]

110. Program: Implement bubble sort in java.

Bubble sort, also referred to as sinking sort, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the 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. Although the algorithm is simple, most of the other sorting algorithms are more efficient for large lists.



Bubble sort has worst-case and average complexity both O(n2), where n is the number of items being sorted. There exist many sorting algorithms with substantially better worst-case or average complexity of O(n log n). Even other O(n2) sorting algorithms, such as insertion sort, tend to have better performance than bubble sort. Therefore, bubble sort is not a practical sorting algorithm when n is large.Performance of bubble sort over an already-sorted list (best-case) is O(n).

```
package com.java2novice.algos;

public class MyBubbleSort {
```

```
    // logic to sort the elements
    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);

    }
}
```

Output:

2, 4, 6, 9, 12, 23, 0, 1, 34,

2, 4, 6, 9, 12, 0, 1, 23, 34,

2, 4, 6, 9, 0, 1, 12, 23, 34,

2, 4, 6, 0, 1, 9, 12, 23, 34,

2, 4, 0, 1, 6, 9, 12, 23, 34,

2, 0, 1, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

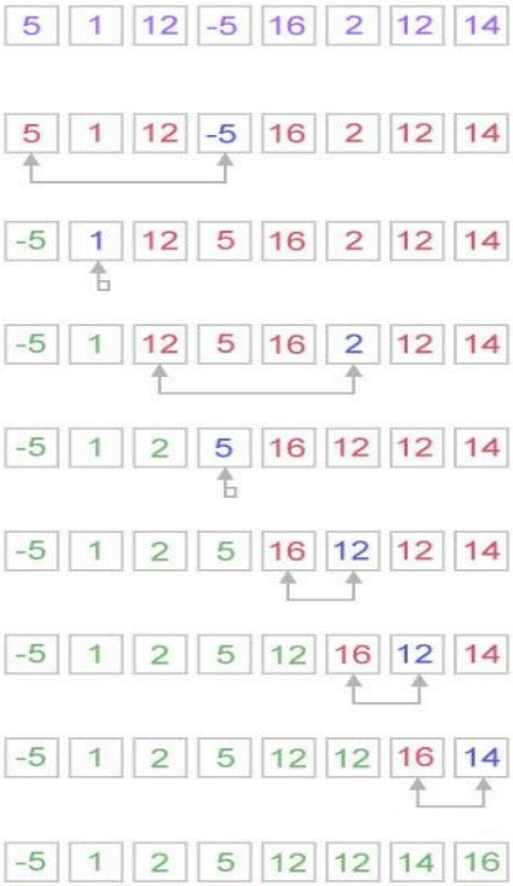
0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

0, 1, 2, 4, 6, 9, 12, 23, 34,

111. Program: Implement selection sort in java.

The selection sort is a combination of searching and sorting. During each pass, the unsorted element with the smallest (or largest) value is moved to its proper position in the array. The number of times the sort passes through the array is one less than the number of items in the array. In the selection sort, the inner loop finds the next smallest (or largest) value and the outer loop places that value into its proper location.



Selection sort is not difficult to analyze compared to other sorting algorithms since none of the loops depend on the data in the array. Selecting the lowest element requires scanning all n elements (this takesn − 1 comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining n − 1 elements and so on, for (n − 1) + (n − 2) + ... + 2 + 1 = n(n − 1) / 2 ∈ Θ(n2) comparisons. Each of these scans requires one swap for n − 1 elements.

```
package com.java2novice.algos;

public class MySelectionSort {

    public static int[] doSelectionSort(int[] arr){

        for (int i = 0; i < arr.length - 1; i++)
        {
```

```
            int index = i;
            for (int j = i + 1; j < arr.length; j++)
                if (arr[j] < arr[index])
                    index = j;

            int smallerNumber = arr[index];
            arr[index] = arr[i];
            arr[i] = smallerNumber;
        }
        return arr;
    }

    public static void main(String a[]){

        int[] arr1 = {10,34,2,56,7,67,88,42};
        int[] arr2 = doSelectionSort(arr1);
        for(int i:arr2){
            System.out.print(i);
            System.out.print(", ");
        }
    }
}
```

Output:

2, 7, 10, 34, 42, 56, 67, 88,

112. Program: Implement insertion sort in java.

Insertion sort is a simple sorting algorithm, it builds the final sorted array one item at a time. It is much less efficient on large lists than other sort algorithms.

Advantages of Insertion Sort: 1) It is very simple.2) It is very efficient for small data sets.3) It is stable; i.e., it does not change the relative order of elements with equal keys.4) In-place; i.e., only requires a constant amount O(1) of additional memory space.

Insertion sort iterates through the list by consuming one input element at each repetition, and growing a sorted output list. On a repetition, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

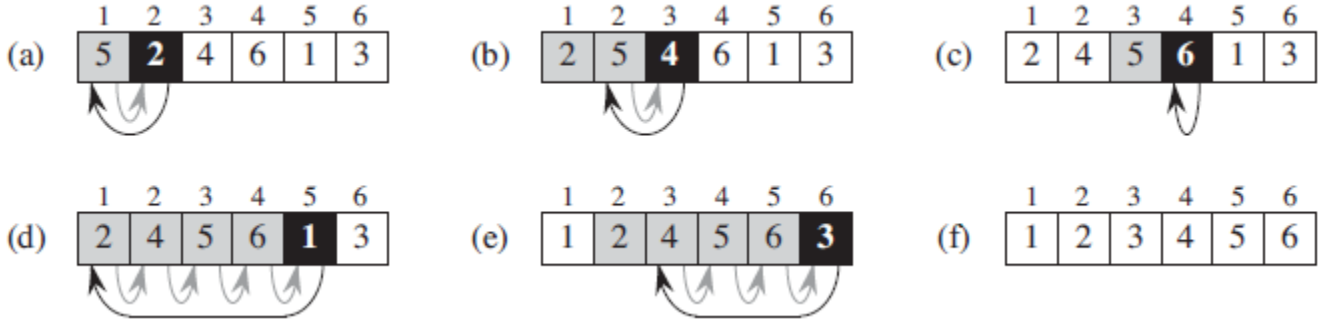


Image source: “Introduction to Algorithms”, The MIT Press

The best case input is an array that is already sorted. In this case insertion sort has a linear running time (i.e., Θ(n)). During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted

subsection of the array. The simplest worst case input is an array sorted in reverse order. The set of all worst case inputs consists of all arrays where each element is the smallest or second-smallest of the elements before it. In these cases every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. This gives insertion sort a quadratic running time (i.e., O(n2)). The average case is also quadratic, which makes insertion sort impractical for sorting large arrays. However, insertion sort is one of the fastest algorithms for sorting very small arrays, even faster than quicksort; indeed, good quicksort implementations use insertion sort for arrays smaller than a certain threshold, also when arising as subproblems; the exact threshold must be determined experimentally and depends on the machine, but is commonly around ten.

```
package com.java2novice.algos;

public class MyInsertionSort {

    public static void main(String a[]){
        int[] arr1 = {10,34,2,56,7,67,88,42};
        int[] arr2 = doInsertionSort(arr1);
        for(int i:arr2){
            System.out.print(i);
            System.out.print(", ");
        }
    }

    public static int[] doInsertionSort(int[] input){

        int temp;
        for (int i = 1; i < input.length; i++) {
            for(int j = i ; j > 0 ; j--){
                if(input[j] < input[j-1]){
                    temp = input[j];
                    input[j] = input[j-1];
                    input[j-1] = temp;
                }
            }
        }
        return input;
    }
}
```

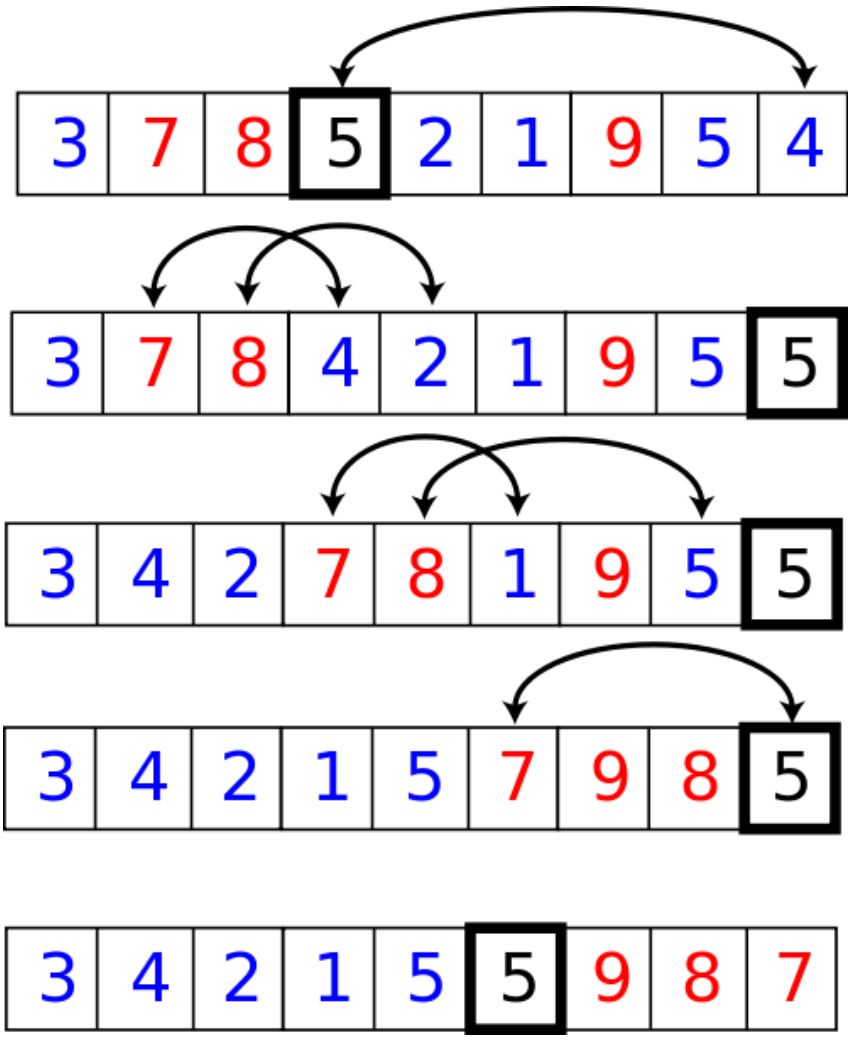
Output:

```
2, 7, 10, 34, 42, 56, 67, 88,
```

113. Program: Implement quick sort in java.

Quicksort or partition-exchange sort, is a fast sorting algorithm, which is using divide and conquer algorithm. Quicksort first divides a large list into two smaller sub-lists: the low elements and the high elements. Quicksort can then recursively sort the sub-lists.

Steps to implement Quick sort:1) Choose an element, called pivot, from the list. Generally pivot can be the middle index element.2) Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.3) Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values.



The complexity of quick sort in the average case is Θ(n log(n)) and in the worst case is Θ(n2).

```
package com.java2novice.sorting;

public class MyQuickSort {

    private int array[];
    private int length;
```

```
    public void sort(int[] inputArr) {

        if (inputArr == null || inputArr.length == 0) {
            return;
        }
        this.array = inputArr;
        length = inputArr.length;
        quickSort(0, length - 1);
    }

    private void quickSort(int lowerIndex, int higherIndex) {

        int i = lowerIndex;
        int j = higherIndex;
        // calculate pivot number, I am taking pivot as middle index number
        int pivot = array[lowerIndex+(higherIndex-lowerIndex)/2];
        // Divide into two arrays
        while (i <= j) {
            /**
             * In each iteration, we will identify a number from left side which
             * is greater then the pivot value, and also we will identify a number
             * from right side which is less then the pivot value. Once the search
             * is done, then we exchange both numbers.
             */
            while (array[i] < pivot) {
                i++;
            }
            while (array[j] > pivot) {
                j--;
            }
            if (i <= j) {
                exchangeNumbers(i, j);
                //move index to next position on both sides
                i++;
                j--;
            }
        }
        // call quickSort() method recursively
        if (lowerIndex < j)
            quickSort(lowerIndex, j);
        if (i < higherIndex)
            quickSort(i, higherIndex);
    }

    private void exchangeNumbers(int i, int j) {
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String a[]){

        MyQuickSort sorter = new MyQuickSort();
        int[] input = {24,2,45,20,56,75,2,56,99,53,12};
        sorter.sort(input);
        for(int i:input){
            System.out.print(i);
            System.out.print(" ");
        }
    }
}
```

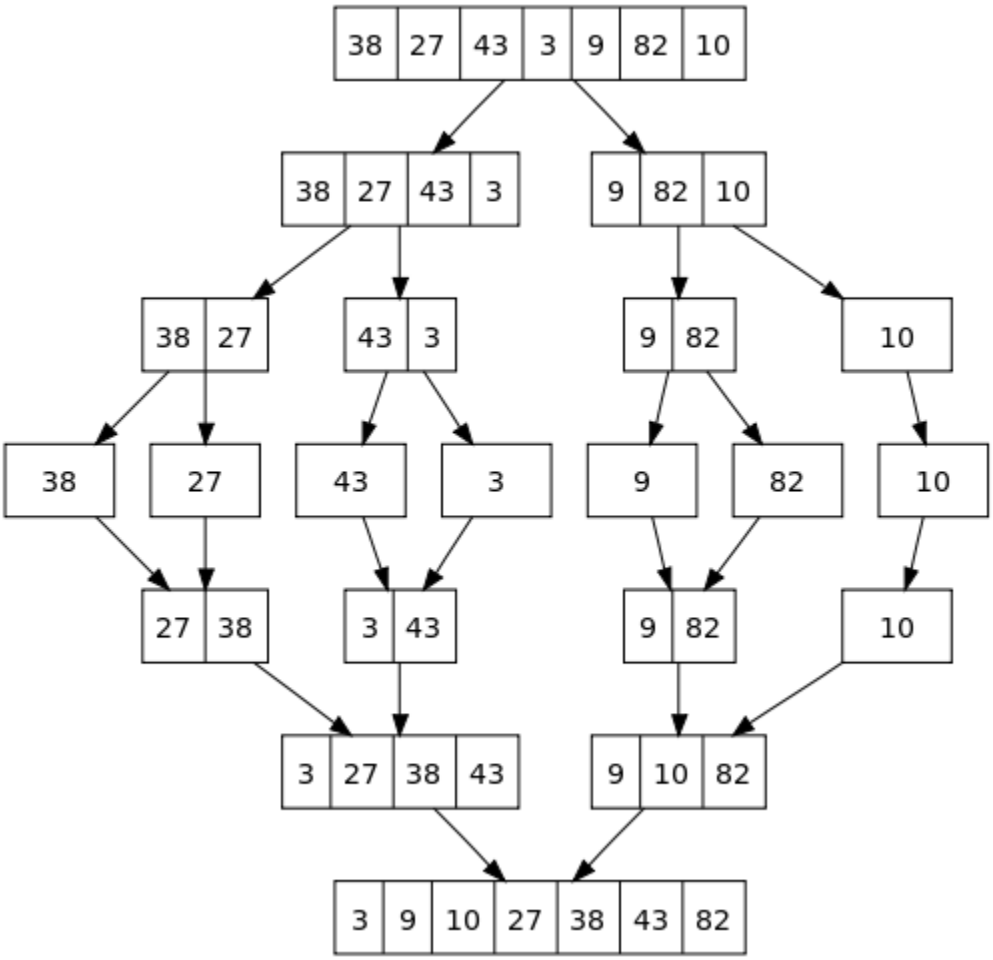
Output:

2 2 12 20 24 45 53 56 56 75 99

114. Program: Implement merge sort in java.

Merge sort is a divide and conquer algorithm.

Steps to implement Merge Sort:1) Divide the unsorted array into n partitions, each partition contains 1 element. Here the one element is considered as sorted.2) Repeatedly merge partitioned units to produce new sublists until there is only 1 sublist remaining. This will be the sorted list at the end.



Merge sort is a fast, stable sorting routine with guaranteed O(n*log(n)) efficiency. When sorting arrays, merge sort requires additional scratch space proportional to the size of the input array. Merge sort is relatively simple to code and offers performance typically only slightly below that of quicksort.

```
package com.java2novice.sorting;

public class MyMergeSort {

    private int[] array;
    private int[] tempMergArr;
    private int length;

    public static void main(String a[]){

        int[] inputArr = {45,23,11,89,77,98,4,28,65,43};
        MyMergeSort mms = new MyMergeSort();
```

```
        mms.sort(inputArr);
        for(int i:inputArr){
            System.out.print(i);
            System.out.print(" ");
        }
    }

    public void sort(int inputArr[]) {
        this.array = inputArr;
        this.length = inputArr.length;
        this.tempMergArr = new int[length];
        doMergeSort(0, length - 1);
    }

    private void doMergeSort(int lowerIndex, int higherIndex) {

        if (lowerIndex < higherIndex) {
            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            // Below step sorts the left side of the array
            doMergeSort(lowerIndex, middle);
            // Below step sorts the right side of the array
            doMergeSort(middle + 1, higherIndex);
            // Now merge both sides
            mergeParts(lowerIndex, middle, higherIndex);
        }
    }

    private void mergeParts(int lowerIndex, int middle, int higherIndex) {

        for (int i = lowerIndex; i <= higherIndex; i++) {
            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;
        while (i <= middle && j <= higherIndex) {
            if (tempMergArr[i] <= tempMergArr[j]) {
                array[k] = tempMergArr[i];
                i++;
            } else {
                array[k] = tempMergArr[j];
                j++;
            }
            k++;
        }
        while (i <= middle) {
            array[k] = tempMergArr[i];
            k++;
            i++;
        }

    }
}
```

Output:

```
4 11 23 28 43 45 65 77 89 98
```

115. Towers of Hanoi implementation using stack.

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:

1) Only one disk must be moved at a time.

2) Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.

3) No disk may be placed on top of a smaller disk.

In this example, we are solving it by using stacks.

```
package com.java2novice.ds.stack;

public class TowersOfHanoiImpl {

    private static MyDynamicStack[] tower;

    public static void towersOfHanoi(int n) {

        // create three stacks, tower[0] is scratch
        tower = new MyDynamicStack[4];
        for (int i = 0; i <= 3; i++){
            tower[i] = new MyDynamicStack(4);
        }
        for (int d = n; d > 0; d--){
            // initialize
            // add disk d to tower             tower[1].push(new Integer(d));
        }
        // move n disks from tower 1 to 2 using 3 as
        // intermediate tower
        showTowerStates(n, 1, 2, 3);
    }

    public static void showTowerStates(int n, int x, int y, int z) {

        if (n > 0) {
            try{
                showTowerStates(n - 1, x, z, y);
                // move d from top of tower x
                Integer d = (Integer) tower[x].pop();
                // to top of tower y
                tower[y].push(d);
                System.out.println("Move disk " + d
                        + " from tower "+ x + " to top of tower " + y);
                showTowerStates(n - 1, z, y, x);
            } catch(Exception ex){}
        }
    }

    public static void main(String[] args) {
        System.out.println("Running 3 disk problem:");
        towersOfHanoi(3);
    }
}

public class MyDynamicStack {

    private int stackSize;
    private int[] stackArr;
    private int top;
```

```
    /**
     * constructor to create stack with size
     * @param size
     */
    public MyDynamicStack(int size) {
        this.stackSize = size;
        this.stackArr = new int[stackSize];
        this.top = -1;
    }

    /**
     * This method adds new entry to the top
     * of the stack
     * @param entry
     * @throws Exception
     */
    public void push(int entry){
        if(this.isStackFull()){
            System.out.println(("Stack is full. Increasing the capacity."));
            this.increaseStackCapacity();
        }
        this.stackArr[++top] = entry;
    }

    /**
     * This method removes an entry from the
     * top of the stack.
     * @return
     * @throws Exception
     */
    public int pop() throws Exception {
        if(this.isStackEmpty()){
            throw new Exception("Stack is empty. Can not remove element.");
        }
        int entry = this.stackArr[top--];
        return entry;
    }

    /**
     * This method returns top of the stack
     * without removing it.
     * @return
     */
    public long peek() {
        return stackArr[top];
    }

    private void increaseStackCapacity(){

        int[] newStack = new int[this.stackSize*2];
        for(int i=0;i<stackSize;i++){
            newStack[i] = this.stackArr[i];
        }
        this.stackArr = newStack;
        this.stackSize = this.stackSize*2;
    }

    /**
     * This method returns true if the stack is
     * empty
     * @return
     */
    public boolean isStackEmpty() {
        return (top == -1);
    }

    /**
     * This method returns true if the stack is full
```

```
     * @return
     */
    public boolean isStackFull() {
        return (top == stackSize - 1);
    }

    public static void main(String[] args) {
        MyDynamicStack stack = new MyDynamicStack(2);
        for(int i=1;i<10;i++){
            stack.push(i);
        }
        for(int i=1;i<4;i++){
            try {
                stack.pop();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

Output:

```
Running 3 disk problem:
```

Move disk 1 from tower 1 to top of tower Move disk 2 from tower 1 to top of tower Move disk 1 from tower 2 to top of tower Move disk 3 from tower 1 to top of tower Move disk 1 from tower 3 to top of tower Move disk 2 from tower 3 to top of tower Move disk 1 from tower 1 to top of tower

116. Stack introduction & implementation

A Stack is an abstract data type or collection where in Push,the addition of data elements to the collection, and Pop, the removal of data elements from the collection, are the major operations performed on the collection. The Push and Pop operations are performed only at one end of the Stack which is referred to as the 'top of the stack'.

In other words,a Stack can be simply defined as Last In First Out (LIFO) data structure,i.e.,the last element added at the top of the stack(In) should be the first element to be removed(Out) from the stack.

Stack Operations:

Push: A new entity can be added to the top of the collection.Pop: An entity will be removed from the top of the collection.Peek or Top: Returns the top of the entity with out removing it.

Overflow State: A stack may be implemented to have a bounded capacity. If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state.

Underflow State: The pop operation removes an item from the top of the stack. A pop either reveals previously concealed items or results in an empty stack, but, if the stack is empty, it goes into underflow state, which means no items are present in stack to be removed.

A stack is a restricted data structure, because only a small number of operations are performed on it. The nature of the pop and push operations also means that stack elements have a natural order. Elements are removed from the stack in the reverse order to the order of their addition. Therefore, the lower elements are those that have been on the stack the longest.

Efficiency of Stacks

In the stack, the elements can be push or pop one at a time in constant O(1) time. That is, the time is not dependent on how many items are in the stack and is therefore very quick. No comparisons or moves are necessary.

```
package com.java2novice.ds.stack;

public class MyStackImpl {

    private int stackSize;
    private int[] stackArr;
    private int top;

    /**
     * constructor to create stack with size
     * @param size
     */
    public MyStackImpl(int size) {
        this.stackSize = size;
        this.stackArr = new int[stackSize];
        this.top = -1;
    }

    /**
     * This method adds new entry to the top
     * of the stack
     * @param entry
     * @throws Exception
     */
    public void push(int entry) throws Exception {
        if(this.isStackFull()){
            throw new Exception("Stack is already full. Can not add element.");
        }
        System.out.println("Adding: "+entry);
        this.stackArr[++top] = entry;
    }
```

```
    /**
     * This method removes an entry from the
     * top of the stack.
     * @return
     * @throws Exception
     */
    public int pop() throws Exception {
        if(this.isStackEmpty()){
            throw new Exception("Stack is empty. Can not remove element.");
        }
        int entry = this.stackArr[top--];
        System.out.println("Removed entry: "+entry);
        return entry;
    }

    /**
     * This method returns top of the stack
     * without removing it.
     * @return
     */
    public int peek() {
        return stackArr[top];
    }

    /**
     * This method returns true if the stack is
     * empty
     * @return
     */
    public boolean isStackEmpty() {
        return (top == -1);
    }

    /**
     * This method returns true if the stack is full
     * @return
     */
    public boolean isStackFull() {
        return (top == stackSize - 1);
    }

    public static void main(String[] args) {
        MyStackImpl stack = new MyStackImpl(5);
        try {
            stack.push(4);
            stack.push(8);
            stack.push(3);
            stack.push(89);
            stack.pop();
            stack.push(34);
            stack.push(45);
            stack.push(78);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        try {
            stack.pop();
            stack.pop();
            stack.pop();
            stack.pop();
            stack.pop();
            stack.pop();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output:

```
Adding: Adding: Adding: Adding: Removed entry: Adding: Adding: Stack is already full. Can
not add element.
Removed entry: Removed entry: Removed entry: Removed entry: Removed entry: Stack is
empty. Can not remove element.
```

117. Queue introduction & array based implementation

A queue is a kind of abstract data type or collection in which the entities in the collection are kept in order and the only operations on the collection are the addition of entities to the rear terminal position, called as enqueue, and removal of entities from the front terminal position, called as dequeue. The queue is called as First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. Often a peek or front operation is also entered, returning the value of the front element without dequeuing it. A queue is an example of a linear data structure, or more abstractly a sequential collection.

Queues provide services in computer science, transport, and operations research where various entities such as data, objects, persons, or events are stored and held to be processed later. In these contexts, the queue performs the function of a buffer.

Queue Operations:

enqueue: Adds an item onto the end of the queue.**front:** Returns the item at the front of the queue.**dequeue:** Removes the item from the front of the queue.

Overflow State: A queue may be implemented to have a bounded capacity. If the queue is full and does not contain enough space to accept an entity to be pushed, the queue is then considered to be in an overflow state.

Underflow State: The dequeue operation removes an item from the top of the queue. A dequeue operation either reveals previously concealed items or results in an empty queue, but, if the queue is empty, it goes into underflow state, which means no items are present in queue to be removed.

Efficiency of Queue

The time needed to add or delete an item is constant and independent of the number of items in the queue. So both addition and deletion can be O(1) operation.

```
package com.java2novice.ds.queue;

public class QueueImpl {

    private int capacity;
    int queueArr[];
    int front = 0;
    int rear = -1;
    int currentSize = 0;

    public QueueImpl(int queueSize){
        this.capacity = queueSize;
        queueArr = new int[this.capacity];
    }

    /**
     * this method adds element at the end of the queue.
     * @param item
     */
    public void enqueue(int item) {
        if (isQueueFull()) {
            System.out.println("Overflow ! Unable to add element: "+item);
        } else {
            rear++;
            if(rear == capacity-1){
                rear = 0;
```

```
            }
            queueArr[rear] = item;
            currentSize++;
            System.out.println("Element " + item+ " is pushed to Queue !");
        }
    }

    /**
     * this method removes an element from the top of the queue
     */
    public void dequeue() {
        if (isQueueEmpty()) {
            System.out.println("Underflow ! Unable to remove element from Queue");
        } else {
            front++;
            if(front == capacity-1){
                System.out.println("Pop operation done ! removed: "+queueArr[front-
1]);
                front = 0;
            } else {
                System.out.println("Pop operation done ! removed: "+queueArr[front-
1]);
            }
            currentSize--;
        }
    }

    /**
     * This method checks whether the queue is full or not
     * @return boolean
     */
    public boolean isQueueFull(){
        boolean status = false;
        if (currentSize == capacity){
            status = true;
        }
        return status;
    }

    /**
     * This method checks whether the queue is empty or not
     * @return
     */
    public boolean isQueueEmpty(){
        boolean status = false;
        if (currentSize == 0){
            status = true;
        }
        return status;
    }

    public static void main(String a[]){

        QueueImpl queue = new QueueImpl(4);
        queue.enqueue(4);
        queue.dequeue();
        queue.enqueue(56);
        queue.enqueue(2);
        queue.enqueue(67);
        queue.dequeue();
        queue.dequeue();
        queue.enqueue(24);
        queue.dequeue();
        queue.enqueue(98);
        queue.enqueue(45);
        queue.enqueue(23);
        queue.enqueue(435);
    }
```

```
}
```

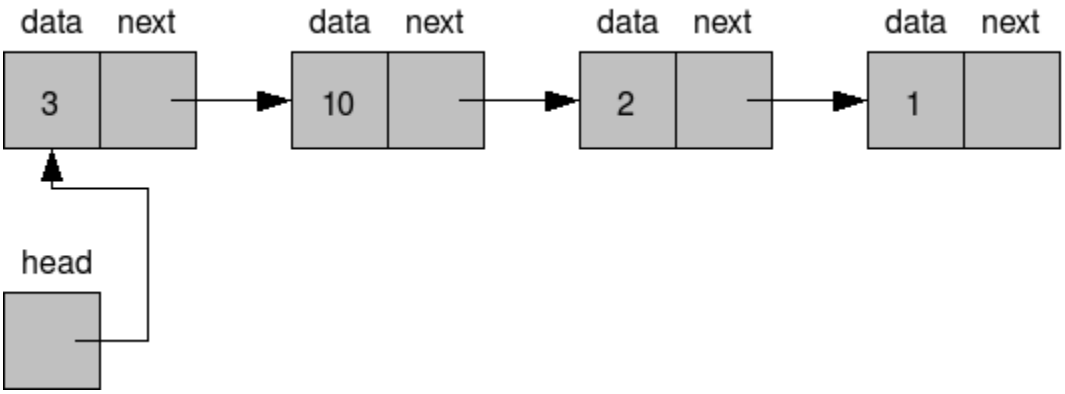
Output:

```
Element 4 is pushed to Queue !
Pop operation done ! removed: Element 56 is pushed to Queue !
Element 2 is pushed to Queue !
Element 67 is pushed to Queue !
Pop operation done ! removed: Pop operation done ! removed: Element 24 is pushed to Queue
!
Pop operation done ! removed: Element 98 is pushed to Queue !
Element 45 is pushed to Queue !
Element 23 is pushed to Queue !
```

Overflow ! Unable to add element:

118. Linked List Data Structure

A linked list is a data structure consisting of a group of nodes which together represent a sequence. Each node is composed of a data and a link or reference to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence.The last node is linked to a terminator used to signify the end of the list.



Linked lists are the simplest and most common data structures. They can be used to implement several other abstract data types, including lists, stacks, queues, associative arrays, and S-expressions, etc.

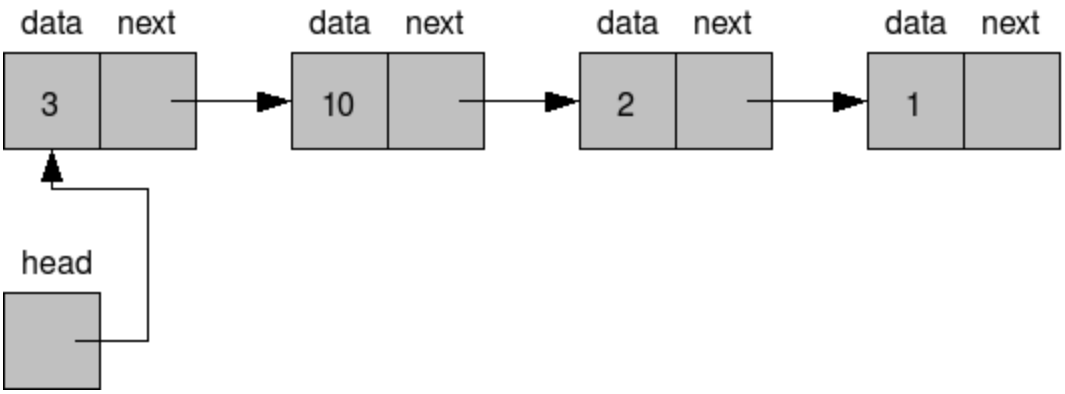
The benefits of a linked list over a conventional array is that the linked list elements can easily be inserted or removed without reallocation or reorganization of the entire structure because the data items need not be stored contiguously in memory or on disk. Linked lists allow insertion and removal of nodes at any point in the list.

On the other hand, simple linked lists do not allow random access to the data, or by using indexing. Thus, many basic operations like obtaining the last node of the list, or finding a node with required data, or locating the place where a new node should be inserted, may require scanning most of the list elements.

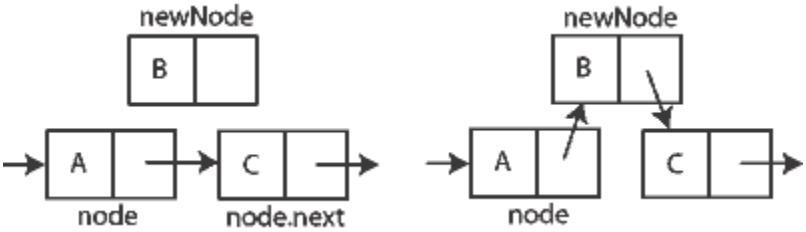
119. Singly linked list implementation

Singly Linked Lists are a type of data structure. It is a type of list. In a singly linked list each node in the list stores the contents of the node and a pointer or reference to the next node in the list. It does not store any pointer or reference to the previous node. It is called a singly linked list because each node only has a single link to another node. To store a single linked list, you only need to store a reference or pointer to the first node in that list. The last node has a pointer to nothingness to indicate that it is the last node.

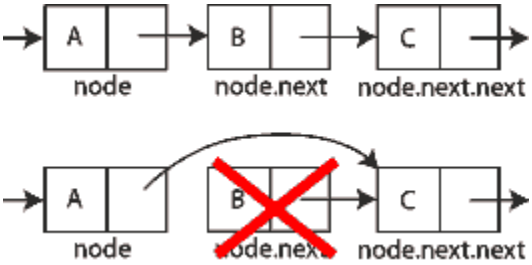
Here is the pictorial view of singly linked list:



Here is the pictorial view of inserting an element in the middle of a singly linked list:



Here is the pictorial view of deleting an element in the middle of a singly linked list:



Below shows the java implementation of singly linked list:

```
package com.java2novice.ds.linkedlist;

public class SinglyLinkedListImpl<T> {

    private Node<T> head;
    private Node<T> tail;

    public void add(T element){

        Node<T> nd = new Node<T>();
        nd.setValue(element);
        System.out.println("Adding: "+element);
        /**
         * check if the list is empty
         */
        if(head == null){
            //since there is only one element, both head and
            //tail points to the same object.
            head = nd;
```

```
            tail = nd;
        } else {
            //set current tail next link to new node
            tail.setNextRef(nd);
            //set tail as newly created node
            tail = nd;
        }
    }

    public void addAfter(T element, T after){

        Node<T> tmp = head;
        Node<T> refNode = null;
        System.out.println("Traversing to all nodes..");
        /**
         * Traverse till given element
         */
        while(true){
            if(tmp == null){
                break;
            }
            if(tmp.compareTo(after) == 0){
                //found the target node, add after this node
                refNode = tmp;
                break;
            }
            tmp = tmp.getNextRef();
        }
        if(refNode != null){
            //add element after the target node
            Node<T> nd = new Node<T>();
            nd.setValue(element);
            nd.setNextRef(tmp.getNextRef());
            if(tmp == tail){
                tail = nd;
            }
            tmp.setNextRef(nd);

        } else {
            System.out.println("Unable to find the given element...");
        }
    }

    public void deleteFront(){

        if(head == null){
            System.out.println("Underflow...");
        }
        Node<T> tmp = head;
        head = tmp.getNextRef();
        if(head == null){
            tail = null;
        }
        System.out.println("Deleted: "+tmp.getValue());
    }

    public void deleteAfter(T after){

        Node<T> tmp = head;
        Node<T> refNode = null;
        System.out.println("Traversing to all nodes..");
        /**
         * Traverse till given element
         */
        while(true){
            if(tmp == null){
                break;
            }
```

```
            if(tmp.compareTo(after) == 0){
                //found the target node, add after this node
                refNode = tmp;
                break;
            }
            tmp = tmp.getNextRef();
        }
        if(refNode != null){
            tmp = refNode.getNextRef();
            refNode.setNextRef(tmp.getNextRef());
            if(refNode.getNextRef() == null){
                tail = refNode;
            }
            System.out.println("Deleted: "+tmp.getValue());
        } else {
            System.out.println("Unable to find the given element...");
        }
    }

    public void traverse(){

        Node<T> tmp = head;
        while(true){
            if(tmp == null){
                break;
            }
            System.out.println(tmp.getValue());
            tmp = tmp.getNextRef();
        }
    }

    public static void main(String a[]){
        SinglyLinkedListImpl<Integer> sl = new SinglyLinkedListImpl<Integer>();
        sl.add(3);
        sl.add(32);
        sl.add(54);
        sl.add(89);
        sl.addAfter(76, 54);
        sl.deleteFront();
        sl.deleteAfter(76);
        sl.traverse();

    }
}

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;
```

```
        }
    }
}
```

Output:

```
Adding: Adding: Adding: Adding: Traversing to all nodes..
Deleted: Traversing to all nodes..
Deleted:
```

120. Program: How to iterate through collection objects?

Description:

You can iterate through any collection object by using Iterator object. It provides two methods to iterate. The hasNext() method returns true if the iteration has more elements. The next() method returns the next element in the iteration. Below example shows how to iterate through an ArrayList.

```
package com.java2novice.iterator;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class MyCollectionIterator {

    public static void main(String a[]){

        List<String> myList = new ArrayList<String>();
        myList.add("Java");
        myList.add("Unix");
        myList.add("Oracle");
        myList.add("C++");
        myList.add("Perl");
        Iterator<String> itr = myList.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }
    }
}
```

Output:

```
Java
Unix
Oracle
C++
Perl
```

121. Java ListIterator Sample Code

Description:

Using ListIterator, we can iterate all elements of a list in either direction. You can access next element by calling next() method, and also you can access previous element by calling previous() method on the list.

```
package com.myjava.listiterator;

import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class MyListIterator {
    public static void main(String a[]){
        List<Integer> li = new ArrayList<Integer>();
        ListIterator<Integer> litr = null;
        li.add(23);
        li.add(98);
        li.add(29);
        li.add(71);
        li.add(5);
        litr=li.listIterator();
        System.out.println("Elements in forward directiton");
        while(litr.hasNext()){
            System.out.println(litr.next());
        }
        System.out.println("Elements in backward directiton");
        while(litr.hasPrevious()){
            System.out.println(litr.previous());
        }
    }
}
```

Example Output

```
Elements in forward directiton
Elements in backward directiton
```

122. Java Enumeration Sample Code

Description:

A class that implements Enumeration interface provides access to series of elements one at a time. You need to call nextElement method to get next element in the series. Also hasMoreElements() method gives you status about the availability of next element in the series.

```
package com.myjava.Enumeration;

import java.util.Enumeration;
import java.util.Vector;

public class MyEnumeration {
```

```
    public static void main(String a[]){
        Vector<String> lang = new Vector<String>();
        Enumeration<String> en = null;
        lang.add("JAVA");
        lang.add("JSP");
        lang.add("SERVLET");
        lang.add("EJB");
        lang.add("PHP");
        lang.add("PERL");
        en = lang.elements();
        while(en.hasMoreElements()){
            System.out.println(en.nextElement());
        }
    }
}
```

Example Output

```
JAVA
JSP
SERVLET
EJB
PHP
PERL
```

123. Program: Basic Vector Operations.

Description:

Below example shows how to create vector object, adding elements to it, getting elements by specifying index, getting elements index, getting first element, getting last element, and is vector is empty or not.

```
package com.java2novice.vector;

import java.util.Vector;

public class BasicVectorOperations {

    public static void main(String a[]){
        Vector<String> vct = new Vector<String>();
        //adding elements to the end
        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        System.out.println(vct);
        //adding element at specified index
        vct.add(2,"Random");
        System.out.println(vct);
        //getting elements by index
        System.out.println("Element at index 3 is: "+vct.get(3));
        //getting first element
        System.out.println("The first element of this vector is:
"+vct.firstElement());
        //getting last element
        System.out.println("The last element of this vector is:
"+vct.lastElement());
        //how to check vector is empty or not
        System.out.println("Is this vector empty? "+vct.isEmpty());
```

```
    }
}
```

Output:

```
[First, Second, Third]
[First, Second, Random, Third]
Element at index 3 is: Third
The first element of this vector is: First
The last element of this vector is: Third
Is this vector empty? false
```

124. Program: Basic Hashtable Operations.

Description:

Below example shows basic operations on Hashtable like creating hashtable object, adding key-value pair, getting the value based on key, checking hashtable is empty or not, removing an element, and size of the hashtable.

```
package com.java2novice.hashtable;

import java.util.Hashtable;

public class MyHashtableOperations {

    public static void main(String a[]){
        //Create hashtable instance
        Hashtable<String,String> ht = new Hashtable<String,String>();
        //add key-value pair to hashtable
        ht.put("first", "FIRST INSERTED");
        ht.put("second", "SECOND INSERTED");
        ht.put("third","THIRD INSERTED");
        System.out.println(ht);
        //getting value for the given key from hashtable
        System.out.println("Value of key 'second': "+ht.get("second"));
        System.out.println("Is Hashtable empty? "+ht.isEmpty());
        ht.remove("third");
        System.out.println(ht);
        System.out.println("Size of the Hashtable: "+ht.size());
    }
}
```

Output:

```
{third=THIRD INSERTED, second=SECOND INSERTED, first=FIRST INSERTED}
Value of key 'second': SECOND INSERTED
Is Hashtable empty? false
```

```
{second=SECOND INSERTED, first=FIRST INSERTED}
```
Size of the Hashtable:

125. Program: How to iterate through Hashtable?

Description:

Below example shows how to read elements from Hashtable. You can iterate through each and every element by getting all keys as set object. Using each element as a key from set, you can values from Hashtable.

```
package com.java2novice.hashtable;

import java.util.Hashtable;
import java.util.Set;

public class MyHashtableRead {

    public static void main(String a[]){

        Hashtable<String, String> hm = new Hashtable<String, String>();
        //add key-value pair to Hashtable
        hm.put("first", "FIRST INSERTED");
        hm.put("second", "SECOND INSERTED");
        hm.put("third","THIRD INSERTED");
        System.out.println(hm);
        Set<String> keys = hm.keySet();
        for(String key: keys){
            System.out.println("Value of "+key+" is: "+hm.get(key));
        }
    }
}
```

Output:

```
{third=THIRD INSERTED, second=SECOND INSERTED, first=FIRST INSERTED}
Value of third is: THIRD INSERTED
Value of second is: SECOND INSERTED
Value of first is: FIRST INSERTED
```

126. Program: Basic HashSet Operations.

Description:

Below example shows basic operations on HashSet object like creating object, adding elements, verifying whether the hashset is empty or not, removing an element, size of the hashset, and to check whether an object exists or not.

```
package com.java2novice.hashset;

import java.util.HashSet;

public class MyBasicHashSet {

    public static void main(String a[]){
        HashSet<String> hs = new HashSet<String>();
        //add elements to HashSet
        hs.add("first");
        hs.add("second");
        hs.add("third");
        System.out.println(hs);
        System.out.println("Is HashSet empty? "+hs.isEmpty());
        hs.remove("third");
        System.out.println(hs);
        System.out.println("Size of the HashSet: "+hs.size());
        System.out.println("Does HashSet contains first element?
"+hs.contains("first"));
    }
}
```

Output:

```
[second, third, first]
Is HashSet empty? false
[second, first]
Size of the HashSet: Does HashSet contains first element? true
```

127. Program: Basic LinkedHashSet Operations.

Description:

Below example shows how to create LinkedHashSet object, adding elements to it, getting size of LinkedHashSet object, and is the set empty or not.

```
package com.java2novice.linkedhashset;

import java.util.LinkedHashSet;

public class MyLkdHashSetOperations {

    public static void main(String a[]){

        LinkedHashSet<String> lhs = new LinkedHashSet<String>();
        //add elements to HashSet
        lhs.add("first");
        lhs.add("second");
        lhs.add("third");
        System.out.println(lhs);
        System.out.println("LinkedHashSet size: "+lhs.size());
```

```
        System.out.println("Is LinkedHashSet emplty? : "+lhs.isEmpty());
    }
}
```

Output:

```
[first, second, third]
LinkedHashSet size: Is LinkedHashSet emplty? : false
```

128. Program: Basic TreeSet Operations.

Description:

Below example shows basic operations on TreeSet like creating object, adding elements to it, verifies elements existance, deleting all elements at once, size of the set and deleting a specific element.

```
package com.java2novice.treeset;

import java.util.TreeSet;

public class MyBasicTreeset {

    public static void main(String a[]){

        TreeSet<String> ts = new TreeSet<String>();
        ts.add("one");
        ts.add("two");
        ts.add("three");
        System.out.println("Elements: "+ts);
        //check is set empty?
        System.out.println("Is set empty: "+ts.isEmpty());
        //delete all elements from set
        ts.clear();
        System.out.println("Is set empty: "+ts.isEmpty());
        ts.add("one");
        ts.add("two");
        ts.add("three");
        System.out.println("Size of the set: "+ts.size());
        //remove one string
        ts.remove("two");
        System.out.println("Elements: "+ts);
    }
}
```

Output:

```
Elements: [one, three, two]
Is set empty: false
Is set empty: true
```

```
Size of the set: Elements: [one, three]
```

129. Program: Basic HashMap Operations.

Description:

Below example shows basic HashMap functionalities like creating object, adding entries, getting values by passing key, checking is hashmap is empty or not, deleting an element and size of the HashMap.

```
package com.java2novice.hashmap;

import java.util.HashMap;

public class MyBasicHashMap {

    public static void main(String a[]){
        HashMap<String, String> hm = new HashMap<String, String>();
        //add key-value pair to hashmap
        hm.put("first", "FIRST INSERTED");
        hm.put("second", "SECOND INSERTED");
        hm.put("third","THIRD INSERTED");
        System.out.println(hm);
        //getting value for the given key from hashmap
        System.out.println("Value of second: "+hm.get("second"));
        System.out.println("Is HashMap empty? "+hm.isEmpty());
        hm.remove("third");
        System.out.println(hm);
        System.out.println("Size of the HashMap: "+hm.size());
    }
}
```

Output:

```
{second=SECOND INSERTED, third=THIRD INSERTED, first=FIRST INSERTED}
Value of second: SECOND INSERTED
Is HashMap empty? false
{second=SECOND INSERTED, first=FIRST INSERTED}
```

Size of the HashMap:

130. Program: How to iterate through HashMap?

Description:

Below example shows how to read add elements from HashMap. The method keySet() returns all key entries as a set object. Iterating through each key, we can get corresponding value object.

```
package com.java2novice.hashmap;

import java.util.HashMap;
import java.util.Set;

public class MyHashMapRead {
    public static void main(String a[]){
        HashMap<String, String> hm = new HashMap<String, String>();
        //add key-value pair to hashmap
        hm.put("first", "FIRST INSERTED");
        hm.put("second", "SECOND INSERTED");
        hm.put("third","THIRD INSERTED");
        System.out.println(hm);
        Set<String> keys = hm.keySet();
        for(String key: keys){
            System.out.println("Value of "+key+" is: "+hm.get(key));
        }
    }
}
```

Output:

```
{second=SECOND INSERTED, third=THIRD INSERTED, first=FIRST INSERTED}
Value of second is: SECOND INSERTED
Value of third is: THIRD INSERTED
Value of first is: FIRST INSERTED
```

131. Program: Basic TreeMap Operations.

Description:

Below example shows basic operations on TreeMap like creating an object, adding key-value pair objects, gtting value by passing key object, checking whether the map has elements or not, deleting specific entry, and size of the TreeMap.

```
package com.java2novice.treemap;

import java.util.TreeMap;

public class MyBasicOperations {

    public static void main(String a[]){
        TreeMap<String, String> hm = new TreeMap<String, String>();
        //add key-value pair to TreeMap
        hm.put("first", "FIRST INSERTED");
        hm.put("second", "SECOND INSERTED");
        hm.put("third","THIRD INSERTED");
        System.out.println(hm);
        //getting value for the given key from TreeMap
        System.out.println("Value of second: "+hm.get("second"));
        System.out.println("Is TreeMap empty? "+hm.isEmpty());
        hm.remove("third");
        System.out.println(hm);
```

```
        System.out.println("Size of the TreeMap: "+hm.size());
    }
}
```

Output:

```
{first=FIRST INSERTED, second=SECOND INSERTED, third=THIRD INSERTED}
Value of second: SECOND INSERTED
Is TreeMap empty? false
{first=FIRST INSERTED, second=SECOND INSERTED}
```

Size of the TreeMap:

132. Program: How to create empty list using Collections class?

Description:

Collections.emptyList() method returns the empty list (immutable). This list is serializable.

```
package com.java2novice.collections;

import java.util.Collections;
import java.util.List;

public class MyEmptyList {

    public static void main(String a[]){

        List<String> myEmptyList = Collections.<String>emptyList();
        System.out.println("Empty list: "+myEmptyList);
    }
}
```

Output:

```
Empty list: []
```

133. Program: Write an example for Collections.checkedList() method.

Description:

Collections.checkedList() method returns a dynamically typesafe view of the specified list. Any attempt to insert an element of the wrong type will result in an immediate ClassCastException. Assuming a list contains no incorrectly typed elements prior to the time a dynamically typesafe view is generated, and that all subsequent access to the list takes place through the view, it is guaranteed that the list cannot contain an incorrectly typed element.

```
package com.java2novice.collections;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MyCheckedList {

    public static void main(String a[]){

        List myList = new ArrayList();
        myList.add("one");
        myList.add("two");
        myList.add("three");
        myList.add("four");
        List chkList = Collections.checkedList(myList, String.class);
        System.out.println("Checked list content: "+chkList);
        //you can add any type of elements to myList object
        myList.add(10);
        //you cannot add any type of elements to chkList object, doing so
        //throws ClassCastException
        chkList.add(10); //throws ClassCastException
    }
}
```

Output:

```
Checked list content: [one, two, three, four]
Exception in thread "main" java.lang.ClassCastException: Attempt to insert class
java.lang.Integer element into
                                    collection with element type class java.lang.String
        at java.util.Collections$CheckedCollection.typeCheck(Collections.java:2202)
        at java.util.Collections$CheckedCollection.add(Collections.java:2243)
        at com.java2novice.collections.MyCheckedList.main(MyCheckedList.java:22)
```

134. Program: How to Enumeration for ArrayList object?

Description:

Below example shows how to get enumeration object for ArrayList object. Collections.enumeration() method provides enumeration object.

```
package com.java2novice.collections;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.List;

public class MyListEnumeration {

    public static void main(String a[]){

        List<String> ls = new ArrayList<String>();
        ls.add("one");
        ls.add("two");
```

```
        ls.add("three");
        ls.add("four");
        Enumeration<String> enm = Collections.enumeration(ls);
        while(enm.hasMoreElements()){
            System.out.println(enm.nextElement());
        }
    }
}
```

Output:

```
one
two
three
four
```

135. How to create basic custom annotation?

Description:

- Annotations are created by using @ sign, folled by the keyword interface, and followed by annotation name as shown in the below example.
- Members can be decleared as shown in the example, it looks like methods. The example defines two members called name and desc. We should not provide implementation for these members.
- All annotations extends java.lang.annotation.Annotation interface. Annotations cannot include any extends caluse.
- Below example shows how to use this annotation to method.

```
package com.java2novice.annotations;

public @interface MySampleAnn {

    String name();
    String desc();
}

class MyAnnTest{

    @MySampleAnn(name = "test1", desc = "testing annotations")
    public void myTestMethod(){
        //method implementation
    }
}
```

Sorting, Creating different types of sorts.

136. What is recurrence for worst case of QuickSort and what is the time complexity in Worst case?

Recurrence is T(n) = T(n-1) + O(n) and time complexity is O(n^2)

The worst case of QuickSort occurs when the picked pivot is always one of the corner elements in sorted array. In

worst case, QuickSort recursively calls one subproblem with size 0 and other subproblem with size (n-1). So recurrence is T(n) = T(n-1) + T(0) + O(n) The above expression can be rewritten as T(n) = T(n-1) + O(n) 1 void exchange(int *a, int *b) { int temp; temp = *a; *a = *b; *b = temp; } int partition(int arr[], int si, int ei) { int x = arr[ei]; int i = (si - 1); int j; for (j = si; j <= ei - 1; j++) { if(arr[j] <= x) { i++; exchange(&arr[i], &arr[j]); } } exchange (&arr[i + 1], &arr[ei]); return (i + 1); } /* Implementation of Quick Sort arr[] --> Array to be sorted si --> Starting index ei --> Ending index */ void quickSort(int arr[], int si, int ei) { int pi; /* Partitioning index */ if(si < ei) { pi = partition(arr, si, ei); quickSort(arr, si, pi - 1); quickSort(arr, pi + 1, ei); } } [/sourcecode]

137. Suppose we have a O(n) time algorithm that finds median of an unsorted array. Now consider a QuickSort implementation where we first find median using the above algorithm, then use median as pivot. What will be the worst case time complexity of this modified QuickSort.

O(nLogn)

If we use median as a pivot element, then the recurrence for all cases becomes T(n) = 2T(n/2) + O(n) The above recurrence can be solved using Master Method. It falls in case 2 of master method.

138. Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Following example explains the above steps:

```
arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]
// and place it at beginning
11 25 12 22 64

// Find the minimum element in arr[1...4]
// and place it at beginning of arr[1...4]
11 12 25 22 64

// Find the minimum element in arr[2...4]
// and place it at beginning of arr[2...4]
11 12 22 25 64

// Find the minimum element in arr[3...4]
// and place it at beginning of arr[3...4]
11 12 22 25 64
```

139. Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:First Pass:(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:(**1 4** 2 5 8) –> (**1 4** 2 5 8)(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2(1 2 **4 5** 8) –> (1 2 **4 5** 8)(1

2 4 **5 8**) –> (1 2 4 **5 8**)Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:(**1 2** 4 5 8) –> (**1 2** 4 5 8)(1 **2 4** 5 8) –> (1 **2 4** 5 8)(1 2 **4 5** 8) –> (1 2 **4 5** 8)(1 2 4 **5 8**) –> (1 2 4 **5 8**)

```
/ Java program for implementation of Bubble Sort
class BubbleSort
{
    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])
                {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
    }

    /* Prints the array */
    void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method to test above
    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);
    }
}
/* This code is contributed by Rajat Mishra */
```

Output:

Sorted array:

11 12 22 25 34 64 90

140. Recursive Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:First Pass:(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:(**1 4** 2 5 8) –> (**1 4** 2 5 8)(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2(1 2 **4 5** 8) –> (1 2 **4 5** 8)(1 2 4 **5 8**) –> (1 2 4 **5 8**)Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:(**1 2** 4 5 8) –> (**1 2** 4 5 8)(1 **2 4** 5 8) –> (1 **2 4** 5 8)(1 2 **4 5** 8) –> (1 2 **4 5** 8)(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Following is iterative Bubble sort algorithm :

```
// Iterative Bubble Sort
bubbleSort(arr[], n)
{
  for (i = 0; i < n-1; i++)

     // Last i elements are already in place
     for (j = 0; j < n-i-1; j++)
       if (arr[j] > arr[j+1])
         swap(arr[j], arr[j+1]);
}
```

141. How to implement it recursively?

Recursive Bubble Sort has no performance/implementation advantages, but can be a good question to check one's understanding of Bubble Sort and recursion.

If we take a closer look at Bubble Sort algorithm, we can notice that in first pass, we move largest element to end (Assuming sorting in increasing order). In second pass, we move second largest element to second last position and so on.

Recursion Idea.

Base Case: If array size is 1, return.

Do One Pass of normal Bubble Sort. This pass fixes last element of current subarray.

Recur for all elements except last of current subarray.

Below is implementation of above idea.

```
// Java program for recursive implementation
// of Bubble sort

import java.util.Arrays;

public class GFG
{
    // A function to implement bubble sort
    static void bubbleSort(int arr[], int n)
    {
        // Base case
        if (n == 1)
            return;

        // One pass of bubble sort. After
        // this pass, the largest element
        // is moved (or bubbled) to end.
        for (int i=0; i<n-1; i++)
            if (arr[i] > arr[i+1])
            {
                // swap arr[i], arr[i+1]
                int temp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = temp;
            }

        // Largest element is fixed,
        // recur for remaining array
        bubbleSort(arr, n-1);
    }

    // Driver Method
    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));
    }
}
```

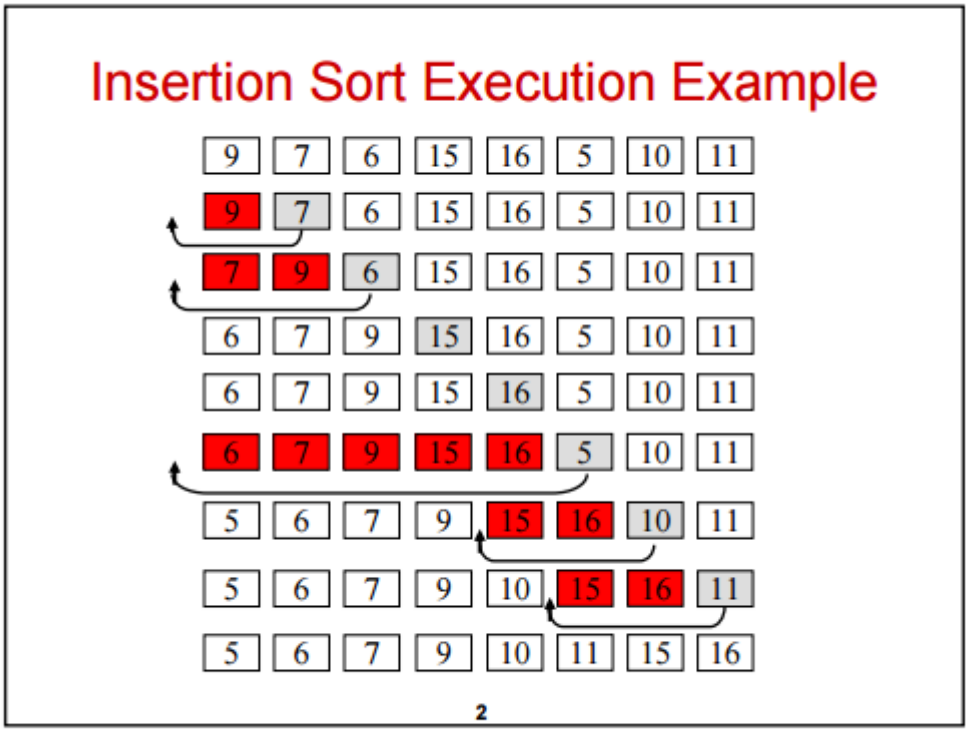
Output :

```
Sorted array :
11 12 22 25 34 64 90
```

142. Insertion Sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Algorithm// Sort an arr[] of size ninsertionSort(arr, n)Loop from i = 1 to n-1…….a) Pick element arr[i] and insert it into sorted sequence arr[0…i-1]



Example:

Another Example: 12, 11, 13, 5, 6

Let us loop for i = 1 (second element of the array) to 5 (Size of input array)

i = 1. Since 11 is smaller than 12, move 12 and insert 11 before 12**11, 12**, 13, 5, 6

i = 2. 13 will remain at its position as all elements in A[0..I-1] are smaller than 13**11, 12, 13**, 5, 6

i = 3. 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.**5, 11, 12, 13**, 6

i = 4. 6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position.**5, 6, 11, 12, 13**

```
// Java program for implementation of Insertion Sort
class InsertionSort
{
    /*Function to sort array using insertion sort*/
    void sort(int arr[])
    {
        int n = arr.length;
        for (int i=1; i<n; ++i)
        {
            int key = arr[i];
            int j = i-1;

            /* Move elements of arr[0..i-1], that are
               greater than key, to one position ahead
               of their current position */
            while (j>=0 && arr[j] > key)
            {
                arr[j+1] = arr[j];
                j = j-1;
            }
            arr[j+1] = key;
        }
    }
```

```
    /* A utility function to print array of size n*/
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");

        System.out.println();
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = {12, 11, 13, 5, 6};

        InsertionSort ob = new InsertionSort();
        ob.sort(arr);

        printArray(arr);
    }
} /* This code is contributed by Rajat Mishra. */
```

```
Output:
  5 6 11 12 13
```

Time Complexity: O(n*n)
Auxiliary Space: O(1)

Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Algorithmic Paradigm: Incremental Approach

Sorting In Place: Yes

Stable: Yes

Online: Yes

Uses: Insertion sort is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

143. What is Binary Insertion Sort?We can use binary search to reduce the number of comparisons in normal insertion sort. Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration. In normal insertion, sort it takes O(i) (at ith iteration) in worst case. we can reduce it to O(logi) by using binary search. The algorithm as a whole still has a running worst case running time of O(n2) because of the series of swaps required for each insertion. Refer this for implementation.

How to implement Insertion Sort for Linked List?Below is simple insertion sort algorithm for linked list.

1) Create an empty sorted (or result) list

2) Traverse the given list, do following for every node.

.....a) Insert current node in sorted way in sorted or result list.

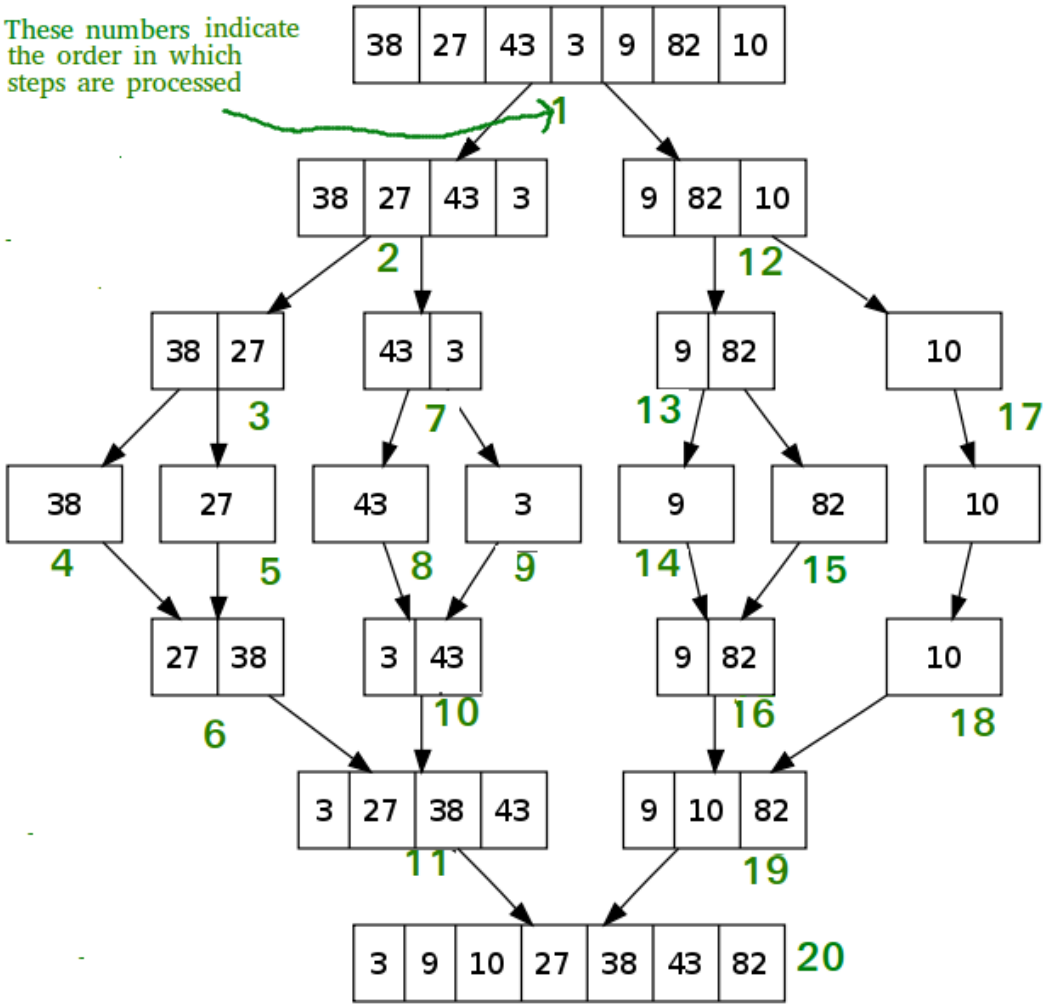
3) Change head of given linked list to head of sorted (or result) list.

Merge Sort

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

```
MergeSort(arr[], l,  r)
If r > l
1. Find the middle point to divide the array into two halves:
        middle m = (l+r)/2
2. Call mergeSort for first half:
        Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
        Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
        Call merge(arr, l, m, r)
```

The following diagram from wikipedia shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.




```
/* Java program for Merge Sort */
class MergeSort
{
    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    void merge(int arr[], int l, int m, int r)
    {
```

```
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;

        /* Create temp arrays */
        int L[] = new int [n1];
        int R[] = new int [n2];

        /*Copy data to temp arrays*/
        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];


        /* Merge the temp arrays */

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarry array
        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++;
        }

        /* Copy remaining elements of L[] if any */
        while (i < n1)
        {
            arr[k] = L[i];
            i++;
            k++;
        }

        /* Copy remaining elements of R[] if any */
        while (j < n2)
        {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    // Main function that sorts arr[l..r] using
    // merge()
    void sort(int arr[], int l, int r)
    {
        if (l < r)
        {
            // Find the middle point
            int m = (l+r)/2;

            // Sort first and second halves
            sort(arr, l, m);
            sort(arr , m+1, r);

            // Merge the sorted halves
```

```
            merge(arr, l, m, r);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method
    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);
    }
}
/* This code is contributed by Rajat Mishra */
```

Output:

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

Time Complexity: Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.T(n) = 2T(n/2) + The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is .Time complexity of Merge Sort is in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.

Auxiliary Space: O(n)

Algorithmic Paradigm: Divide and Conquer

Sorting In Place: No in a typical implementation

Stable: Yes

Applications of Merge Sort

Merge Sort is useful for sorting linked lists in O(nLogn) time.

1. In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

   In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the

memory at (x + i*4). Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

2. Inversion Count Problem
3. Used in External Sorting

144. Iterative Merge Sort

Following is a typical recursive implementation of Merge Sort that uses last element as pivot.

```
// Recursive Java Program for merge sort

import java.util.Arrays;
public class GFG
{
    public static void mergeSort(int[] array)
    {
        if(array == null)
        {
            return;
        }

        if(array.length > 1)
        {
            int mid = array.length / 2;

            // Split left part
            int[] left = new int[mid];
            for(int i = 0; i < mid; i++)
            {
                left[i] = array[i];
            }

            // Split right part
            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;

            // Merge left and right arrays
            while(i < left.length && j < right.length)
            {
                if(left[i] < right[j])
                {
                    array[k] = left[i];
                    i++;
                }
                else
                {
                    array[k] = right[j];
                    j++;
                }
                k++;
            }
            // Collect remaining elements
            while(i < left.length)
```

```
            {
                array[k] = left[i];
                i++;
                k++;
            }
            while(j < right.length)
            {
                array[k] = right[j];
                j++;
                k++;
            }
        }
    }
    // Driver program to test above functions.
    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]+" ");
    }
}

// Code Contributed by Mohit Gupta_OMG
```

Output:

Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13

Iterative Merge Sort: The above function is recursive, so uses function call stack to store intermediate values of l and h. The function call stack stores other bookkeeping information together with parameters. Also, function calls involve overheads like storing activation record of the caller function and then resuming execution. Unlike Iterative QuickSort, the iterative MergeSort doesn't require explicit auxiliary stack.The above function can be easily converted to iterative version. Following is iterative Merge Sort.

```
/* Iterative C program for merge sort */
#include<stdlib.h>
#include<stdio.h>

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r);

// Utility function to find minimum of two integers
int min(int x, int y) { return (x<y)? x :y; }


/* Iterative mergesort function to sort arr[0...n-1] */
void mergeSort(int arr[], int n)
{
   int curr_size;  // For current size of subarrays to be merged
                   // curr_size varies from 1 to n/2
```

```
   int left_start; // For picking starting index of left subarray
                   // to be merged

   // Merge subarrays in bottom up manner.  First merge subarrays of
   // size 1 to create sorted subarrays of size 2, then merge subarrays
   // of size 2 to create sorted subarrays of size 4, and so on.
   for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size)
   {
       // Pick starting point of different subarrays of current size
       for (left_start=0; left_start<n-1; left_start += 2*curr_size)
       {
           // Find ending point of left subarray. mid+1 is starting
           // point of right
           int mid = left_start + curr_size - 1;

           int right_end = min(left_start + 2*curr_size - 1, n-1);

           // Merge Subarrays arr[left_start...mid] & arr[mid+1...right_end]
           merge(arr, left_start, mid, right_end);
       }
   }
}

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    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++;
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
```

```
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    mergeSort(arr, n);

    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Output:

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

Time complexity of above iterative function is same as recursive, i.e., Θ(nLogn).

145. QuickSort

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

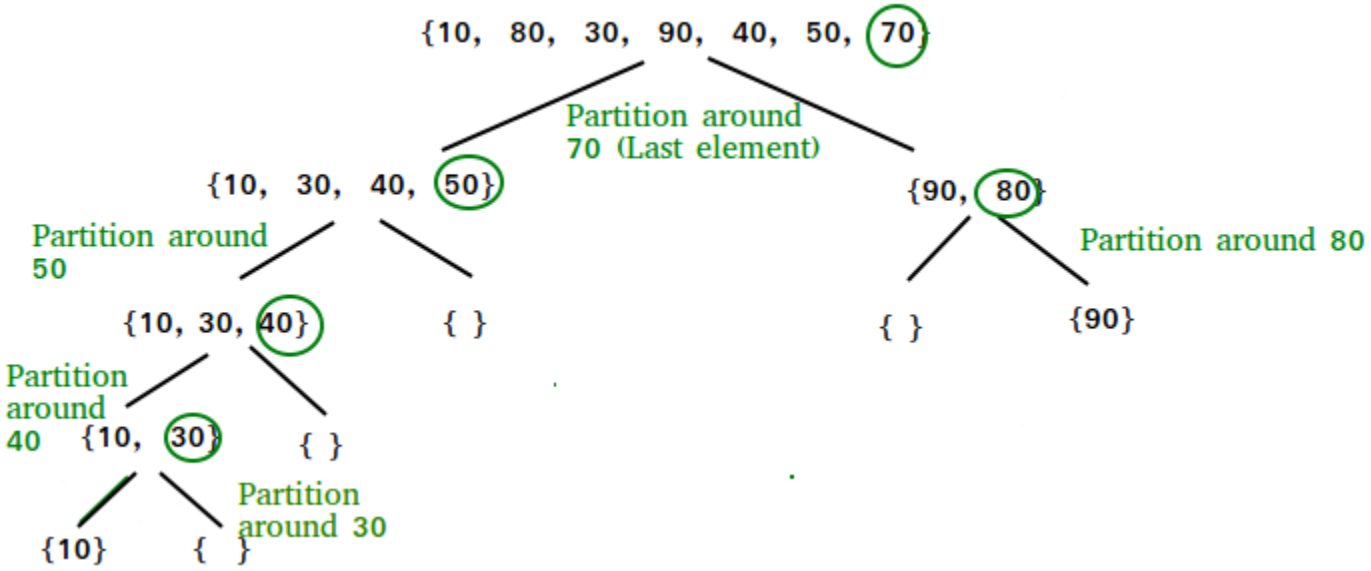
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
```

```
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```



Partition Algorithm There can be many ways to do partition, following pseudo code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Pseudo code for partition()

```
/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
    array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
```

```
    pivot = arr[high];

    i = (low - 1)  // Index of smaller element

    for (j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}
```

Illustration of partition() :

arr[] = {10, 80, 30, 90, 40, 50, 70}
Indexes: 0 1 2 3 4 5 6

low = 0, high = 6, pivot = arr[h] = 70
Initialize index of smaller element, **i = -1**

Traverse elements from j = low to high-1
j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 0
arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j
// are same

j = 1 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 1
arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing
// No change in i and arr[]

j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])
i = 2
arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped
j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]
i = 3
arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

We come out of loop because j is now equal to high-1.

Finally we place pivot at correct position by swapping arr[i+1] and arr[high] (or pivot)

arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

```
// Java program for implementation of QuickSort
class QuickSort
{
    /* This function takes last element as pivot,
       places the pivot element at its correct
       position in sorted array, and places all
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
    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 current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
      high  --> Ending index */
    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    /* A utility function to print array of size n */
```

```
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    // Driver program
    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);
    }
}
/*This code is contributed by Rajat Mishra */
```

Output:

Sorted array:

1 5 7 8 9 10

Analysis of QuickSort Time taken by QuickSort in general can be written as following.

T(n) = T(k) + T(n-k-1) + (n)

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

T(n) = T(0) + T(n-1) + (n)

which is equivalent to

T(n) = T(n-1) + (n)

The solution of above recurrence is (n^2).

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

T(n) = 2T(n/2) + (n)

The solution of above recurrence is (nLogn). It can be solved using case 2 of Master Theorem.

Average Case:To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

T(n) = T(n/9) + T(9n/10) + (n)

Solution of above recurrence is also O(nLogn)

Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like Merge Sort and Heap Sort, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

What is 3-Way QuickSort?In simple QuickSort algorithm, we select an element as pivot, partition the array around

pivot and recur for subarrays on left and right of pivot.Consider an array which has many redundant elements. For example, {1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 2, 2, 4, 1, 4, 4, 4}. If 4 is picked as pivot in Simple QuickSort, we fix only one 4 and recursively process remaining occurrences. In 3 Way QuickSort, an array arr[l..r] is divided in 3 parts:a) arr[l..i] elements less than pivot.b) arr[i+1..j-1] elements equal to pivot.c) arr[j..r] elements greater than pivot.See this for implementation.

How to implement QuickSort for Linked Lists?QuickSort on Singly Linked ListQuickSort on Doubly Linked List

Can we implement QuickSort Iteratively?Yes, please refer Iterative Quick Sort.

Why Quick Sort is preferred over MergeSort for sorting Arrays Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires O(N) extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have O(NlogN) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of O(nLogn). The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.

Quick Sort is also tail recursive, therefore tail call optimizations is done.

Why MergeSort is preferred over QuickSort for Linked Lists?In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the memory at (x + i*4). Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

How to optimize QuickSort so that it takes O(Log n) extra space in worst case?Please see QuickSort Tail Call Optimization (Reducing worst case space to Log n)

146. QuickSort

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

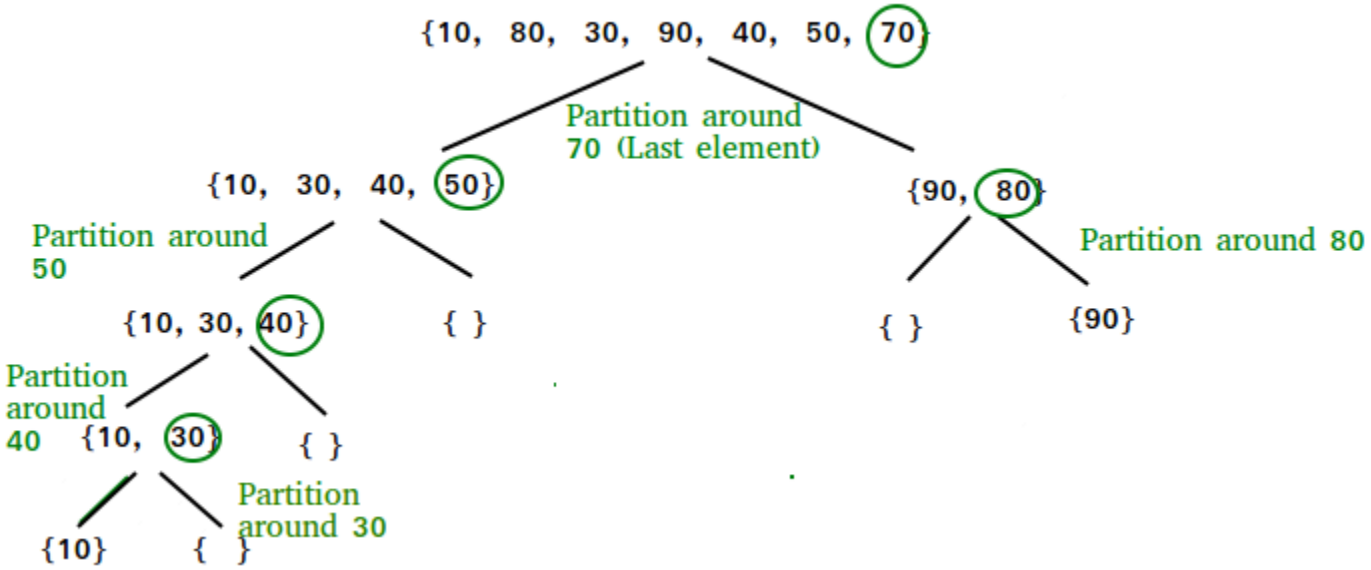
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
```

```
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```



Partition Algorithm There can be many ways to do partition, following pseudo code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

Pseudo code for partition()

```
/* This function takes last element as pivot, places
   the pivot element at its correct position in sorted
    array, and places all smaller (smaller than pivot)
   to left of pivot and all greater elements to right
   of pivot */
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
```

```
    pivot = arr[high];

    i = (low - 1)  // Index of smaller element

    for (j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}
```

Illustration of partition() :

arr[] = {10, 80, 30, 90, 40, 50, 70}

Indexes: 0 1 2 3 4 5 6

low = 0, high = 6, pivot = arr[h] = 70

Initialize index of smaller element, **i = -1**

Traverse elements from j = low to high-1

j = 0 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 0

arr[] = {10, 80, 30, 90, 40, 50, 70} // No change as i and j
// are same

j = 1 : Since arr[j] > pivot, do nothing

// No change in i and arr[]

j = 2 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 1

arr[] = {10, 30, 80, 90, 40, 50, 70} // We swap 80 and 30

j = 3 : Since arr[j] > pivot, do nothing

// No change in i and arr[]

j = 4 : Since arr[j] <= pivot, do i++ and swap(arr[i], arr[j])

i = 2

arr[] = {10, 30, 40, 90, 80, 50, 70} // 80 and 40 Swapped

j = 5 : Since arr[j] <= pivot, do i++ and swap arr[i] with arr[j]

i = 3

arr[] = {10, 30, 40, 50, 80, 90, 70} // 90 and 50 Swapped

We come out of loop because j is now equal to high-1.

Finally we place pivot at correct position by swapping arr[i+1] and arr[high] (or pivot)

arr[] = {10, 30, 40, 50, 70, 90, 80} // 80 and 70 Swapped

Now 70 is at its correct place. All elements smaller than 70 are before it and all elements greater than 70 are after it.

```
// Java program for implementation of QuickSort
class QuickSort
{
    /* This function takes last element as pivot,
       places the pivot element at its correct
       position in sorted array, and places all
       smaller (smaller than pivot) to left of
       pivot and all greater elements to right
       of pivot */
    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 current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
      arr[] --> Array to be sorted,
      low  --> Starting index,
      high  --> Ending index */
    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
```

```
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    // Driver program
    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);
    }
}
/*This code is contributed by Rajat Mishra */
```

Output:

Sorted array:

1 5 7 8 9 10

147. Analysis of QuickSortTime taken by QuickSort in general can be written as following.

```
T(n) = T(k) + T(n-k-1) +       (n)
```

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

```
T(n) = T(0) + T(n-1) +       (n)
which is equivalent to
T(n) = T(n-1) +       (n)
```

The solution of above recurrence is (n^2).

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

```
T(n) = 2T(n/2) +       (n)
```

The solution of above recurrence is (nLogn). It can be solved using case 2 of Master Theorem.

Average Case:To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

```
T(n) = T(n/9) + T(9n/10) +      (n)
```

Solution of above recurrence is also O(nLogn)

Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like Merge Sort and Heap Sort, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

What is 3-Way QuickSort?In simple QuickSort algorithm, we select an element as pivot, partition the array around pivot and recur for subarrays on left and right of pivot.Consider an array which has many redundant elements. For example, {1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 2, 2, 4, 1, 4, 4, 4}. If 4 is picked as pivot in Simple QuickSort, we fix only one 4 and recursively process remaining occurrences. In 3 Way QuickSort, an array arr[l..r] is divided in 3 parts:a) arr[l..i] elements less than pivot.b) arr[i+1..j-1] elements equal to pivot.c) arr[j..r] elements greater than pivot.See this for implementation.

148. How to implement QuickSort for Linked Lists?QuickSort on Singly Linked ListQuickSort on Doubly Linked List

Can we implement QuickSort Iteratively?Yes, please refer Iterative Quick Sort.

149. Why Quick Sort is preferred over MergeSort for sorting Arrays

Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires O(N) extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have O(NlogN) average complexity but the constants differ. For arrays, merge sort loses due to the use of extra O(N) storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of O(nLogn). The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.

Quick Sort is also tail recursive, therefore tail call optimizations is done.

150. Why MergeSort is preferred over QuickSort for Linked Lists?

In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the memory at (x + i*4). Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

151. Heap Sort

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

152. What is Binary Heap?

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible (Source Wikipedia)

A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

153. Why array based representation for Binary Heap?

Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by 2 * I + 1 and right child by 2 * I + 2 (assuming the indexing starts at 0).

154. Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.

2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.

3. Repeat above steps while size of heap is greater than 1.

155. How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

Lets understand with the help of an example:

```
Input data: 4, 10, 3, 5, 1
                    4(0)
          /   \
         10(1)   3(2)
            /   \
     5(3)    1(4)

The numbers in bracket represent the indices in the array
representation of data.

Applying heapify procedure to index 1:
          4(0)
         /   \
            10(1)    3(2)
           /   \
    5(3)    1(4)

Applying heapify procedure to index 0:
            10(0)
```

```
        /  \
         5(1)  3(2)
           /   \
         4(3)    1(4)
```

The heapify procedure calls itself recursively to build heap in top down manner.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

```
// C++ program for implementation of Heap Sort
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i;  // Initialize largest as root
    int l = 2*i + 1;  // left = 2*i + 1
    int r = 2*i + 2;  // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

// Driver program
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
```

```
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Output:

```
Sorted array is
5 6 7 11 12 13
```

Time Complexity: Time complexity of heapify is O(Logn). Time complexity of createAndBuildHeap() is O(n) and overall time complexity of Heap Sort is O(nLogn).

Applications of HeapSort1. Sort a nearly sorted (or K sorted) array**2.** k largest(or smallest) elements in an array

Heap sort algorithm has limited uses because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used. See Applications of Heap Data Structure

156. Counting Sort

Counting sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

Let us understand it with the help of an example.

```
For simplicity, consider the data in the range 0 to 9.
Input data: 1, 4, 1, 2, 7, 5, 2
  1) Take a count array to store the count of each unique object.
  Index:     0  1  2  3  4  5  6  7  8  9
  Count:     0  2  2  0   1  1  0  1  0  0

  2) Modify the count array such that each element at each index
  stores the sum of previous counts.
  Index:     0  1  2  3  4  5  6  7  8  9
  Count:     0  2  4  4  5  6  6  7  7  7

The modified count array indicates the position of each object in
the output sequence.

  3) Output each object from the input sequence followed by
  decreasing its count by 1.
  Process the input data: 1, 4, 1, 2, 7, 5, 2. Position of 1 is 2.
  Put data 1 at index 2 in output. Decrease count by 1 to place
  next data 1 at an index 1 smaller than this index.
```

```
// Java implementation of Counting Sort
class CountingSort
{
    void sort(char arr[])
    {
        int n = arr.length;

        // The output character array that will have sorted arr
        char output[] = new char[n];
```

```
        // Create a count array to store count of inidividul
        // characters and initialize count array as 0
        int count[] = new int[256];
        for (int i=0; i<256; ++i)
            count[i] = 0;

        // store count of each character
        for (int i=0; i<n; ++i)
            ++count[arr[i]];

        // Change count[i] so that count[i] now contains actual
        // position of this character in output array
        for (int i=1; i<=255; ++i)
            count[i] += count[i-1];

        // Build the output character array
        for (int i = 0; i<n; ++i)
        {
            output[count[arr[i]]-1] = arr[i];
            --count[arr[i]];
        }

        // Copy the output array to arr, so that arr now
        // contains sorted characters
        for (int i = 0; i<n; ++i)
            arr[i] = output[i];
    }

    // Driver method
    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]);
    }
}
```

Output:

```
Sorted character array is eeeefggkkorss
```

Time Complexity: O(n+k) where n is the number of elements in input array and k is the range of input.**Auxiliary Space:** O(n+k)

Points to be noted:1. Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted. Consider the situation where the input sequence is between range 1 to 10K and the data is 10, 5, 10K, 5K.**2.** It is not a comparison based sorting. It running time complexity is O(n) with space proportional to the range of data.**3.** It is often used as a sub-routine to another sorting algorithm like radix sort.**4.** Counting sort uses a partial hashing to count the occurrence of the data object in O(1).**5.** Counting sort can be extended to work for negative inputs also.

Exercise:1. Modify above code to sort the input data in the range from M to N.**2.** Modify above code to sort negative input data.**3.** Is counting sort stable and online?**4.** Thoughts on parallelizing the counting sort algorithm.

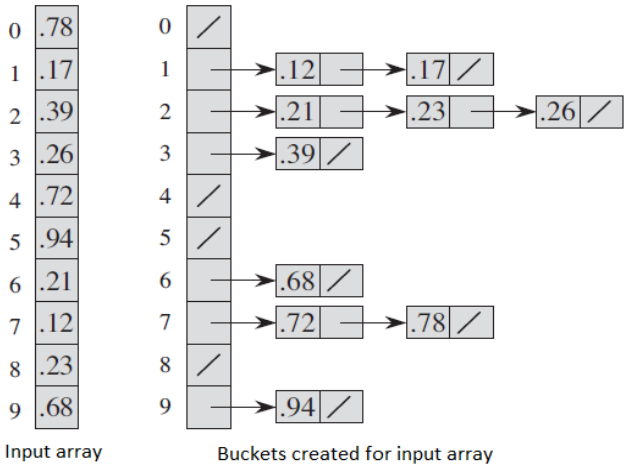
157. Bucket Sort

Bucket sort is mainly useful when input is uniformly distributed over a range. For example, consider the following problem. *Sort a large set of floating point numbers which are in range from 0.0 to 1.0 and are uniformly distributed across the range. How do we sort the numbers efficiently?*

A simple way is to apply a comparison based sorting algorithm. The lower bound for Comparison based sorting algorithm (Merge Sort, Heap Sort, Quick-Sort .. etc) is Ω(n Log n), i.e., they cannot do better than nLogn.Can we sort the array in linear time? Counting sort can not be applied here as we use keys as index in counting sort. Here keys are floating point numbers. The idea is to use bucket sort. Following is bucket algorithm.

```
bucketSort(arr[], n)
1) Create n empty buckets (Or lists).
2) Do following for every array element arr[i].
.......a) Insert arr[i] into bucket[n*array[i]]
3) Sort individual buckets using insertion sort.
4) Concatenate all sorted buckets.
```

Following diagram (taken from CLRS book) demonstrates working of bucket sort.



Time Complexity: If we assume that insertion in a bucket takes O(1) time then steps 1 and 2 of the above algorithm clearly take O(n) time. The O(1) is easily possible if we use a linked list to represent a bucket (In the following code, C++ vector is used for simplicity). Step 4 also takes O(n) time as there will be n items in all buckets.The main step to analyze is step 3. This step also takes O(n) time on average if all numbers are uniformly distributed (please refer CLRS book for more details)

```
Following is C++ implementation of the above algorithm.
// C++ program to sort an array using bucket sort
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

// Function to sort arr[] of size n using bucket sort
void bucketSort(float arr[], int n)
```

```
{
    // 1) Create n empty buckets
    vector<float> b[n];

    // 2) Put array elements in different buckets
    for (int i=0; i<n; i++)
    {
       int bi = n*arr[i]; // Index in bucket
       b[bi].push_back(arr[i]);
    }

    // 3) Sort individual buckets
    for (int i=0; i<n; i++)
       sort(b[i].begin(), b[i].end());

    // 4) Concatenate all buckets into arr[]
    int index = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
          arr[index++] = b[i][j];
}

/* Driver program to test above funtion */
int main()
{
    float arr[] = {0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434};
    int n = sizeof(arr)/sizeof(arr[0]);
    bucketSort(arr, n);

    cout << "Sorted array is \n";
    for (int i=0; i<n; i++)
       cout << arr[i] << " ";
    return 0;
}
```

Output:

```
Sorted array is
0.1234 0.3434 0.565 0.656 0.665 0.897
```

158. Binary Insertion Sort

We can use binary search to reduce the number of comparisons in normal insertion sort. Binary Insertion Sort find use binary search to find the proper location to insert the selected item at each iteration. In normal insertion, sort it takes O(i) (at ith iteration) in worst case. we can reduce it to O(logi) by using binary search.

```
// C program for implementation of binary insertion sort
#include <stdio.h>

// A binary search based function to find the position
// where item should be inserted in a[low..high]
int binarySearch(int a[], int item, int low, int high)
{
    if (high <= low)
        return (item > a[low])?  (low + 1): low;

    int mid = (low + high)/2;

    if(item == a[mid])
        return mid+1;

    if(item > a[mid])
        return binarySearch(a, item, mid+1, high);
    return binarySearch(a, item, low, mid-1);
}
```

```
// Function to sort an array a[] of size 'n'
void insertionSort(int a[], int n)
{
    int i, loc, j, k, selected;

    for (i = 1; i < n; ++i)
    {
        j = i - 1;
        selected = a[i];

        // find location where selected sould be inseretd
        loc = binarySearch(a, selected, 0, j);

        // Move all elements after location to create space
        while (j >= loc)
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = selected;
    }
}

// Driver program to test above function
int main()
{
    int a[] = {37, 23, 0, 17, 12, 72, 31,
               46, 100, 88, 54};
    int n = sizeof(a)/sizeof(a[0]), i;

    insertionSort(a, n);

    printf("Sorted array: \n");
    for (i = 0; i < n; i++)
        printf("%d ",a[i]);

    return 0;
}
```

Output:

```
Sorted array:
0 12 17 23 31 37 46 54 72 88 100
```

Time Complexity: The algorithm as a whole still has a running worst case running time of $O(n^2)$ because of the series of swaps required for each insertion.

This article is contributed by **Amit Auddy**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

159. Tree Sort

Tree sort is a sorting algorithm that is based on Binary Search Tree data structure. It first creates a binary search tree from the elements of the input list or array and then performs an in-order traversal on the created binary search tree to get the elements in sorted order.

Algorithm:

```
Step 1: Take the elements input in an array.
Step 2: Create a Binary search tree by inserting data items from the array into the
        binary searh tree.
Step 3: Perform in-order traversal on the tree to get the elements in sorted order.
```

```
// C++ program to implement Tree Sort
```

```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int key;
    struct Node *left, *right;
};

// A utility function to create a new BST Node
struct Node *newNode(int item)
{
    struct Node *temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Stores inoder traversal of the BST
// in arr[]
void storeSorted(Node *root, int arr[], int &i)
{
    if (root != NULL)
    {
        storeSorted(root->left, arr, i);
        arr[i++] = root->key;
        storeSorted(root->right, arr, i);
    }
}

/* A utility function to insert a new
   Node with given key in BST */
Node* insert(Node* node, int key)
{
    /* If the tree is empty, return a new Node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) Node pointer */
    return node;
}

// This function sorts arr[0..n-1] using Tree Sort
void treeSort(int arr[], int n)
{
    struct Node *root = NULL;

    // Construct the BST
    root = insert(root, arr[0]);
    for (int i=1; i<n; i++)
        insert(root, arr[i]);

    // Store inoder traversal of the BST
    // in arr[]
    int i = 0;
    storeSorted(root, arr, i);
}

// Driver Program to test above functions
int main()
{
    //create input array
    int arr[] = {5, 4, 7, 2, 11};
```

```
    int n = sizeof(arr)/sizeof(arr[0]);

    treeSort(arr, n);

    cout << "Sorted Array : ";
    for (int i=0; i<n; i++)
       cout << arr[i] << " ";

    return 0;
}
```

Output:

```
2 4 5 7 11
```

Average Case Time Complexity : O(n log n) Adding one item to a Binary Search tree on average takes O(log n) time. Therefore, adding n items will take O(n log n) time

Worst Case Time Complexity : $O(n^2)$. The worst case time complexity of Tree Sort can be improved by using a self-balancing binary search tree like Red Black Tree, AVL Tree. Using self-balancing binary tree Tree Sort will take O(n log n) time to sort the array in worst case.

Auxiliary Space : O(n)

160. Merge Sort for Linked Lists

Merge sort is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

Let head be the first node of the linked list to be sorted and headRef be the pointer to head. Note that we need a reference to head in MergeSort() as the below implementation changes next links to sort the linked lists (not data at the nodes), so head node has to be changed if the data at original head is not the smallest value in linked list.

```
MergeSort(headRef)
1) If head is NULL or there is only one element in the Linked List
    then return.
2) Else divide the linked list into two halves.
      FrontBackSplit(head, &a, &b); /* a and b are two halves */
3) Sort the two halves a and b.
      MergeSort(a);
      MergeSort(b);
4) Merge the sorted a and b (using SortedMerge() discussed here)
   and update the head pointer using headRef.
     *headRef = SortedMerge(a, b);
// Java program to illustrate merge sorted
// of linkedList

public class linkedList
{
    node head = null;
    // node a,b;
    static class node
    {
        int val;
        node next;
```

```
        public node(int val)
        {
            this.val = val;
        }
    }

    node sortedMerge(node a, node b)
    {
        node result = null;
        /* Base cases */
        if (a == null)
            return b;
        if (b == null)
            return a;

        /* Pick either a or b, and recur */
        if (a.val <= b.val)
        {
            result = a;
            result.next = sortedMerge(a.next, b);
        }
        else
        {
            result = b;
            result.next = sortedMerge(a, b.next);
        }
        return result;

    }

    node mergeSort(node h)
    {
        // Base case : if head is null
        if (h == null || h.next == null)
        {
            return h;
        }

        // get the middle of the list
        node middle = getMiddle(h);
        node nextofmiddle = middle.next;

        // set the next of middle node to null
        middle.next = null;

        // Apply mergeSort on left list
        node left = mergeSort(h);

        // Apply mergeSort on right list
        node right = mergeSort(nextofmiddle);

        // Merge the left and right lists
        node sortedlist = sortedMerge(left, right);
        return sortedlist;
    }

    // Utility function to get the middle of the linked list
    node getMiddle(node h)
    {
        //Base case
        if (h == null)
            return h;
        node fastptr = h.next;
        node slowptr = h;

        // Move fastptr by two and slow ptr by one
```

```
        // Finally slowptr will point to middle node
        while (fastptr != null)
        {
            fastptr = fastptr.next;
            if(fastptr!=null)
            {
                slowptr = slowptr.next;
                fastptr=fastptr.next;
            }
        }
        return slowptr;
    }

    void push(int new_data)
    {
        /* allocate node */
        node new_node = new node(new_data);

        /* link the old list off the new node */
        new_node.next = head;

        /* move the head to point to the new node */
        head = new_node;
    }

    // Utility function to print the linked list
    void printList(node headref)
    {
        while (headref != null)
        {
            System.out.print(headref.val + " ");
            headref = headref.next;
        }
    }

    public static void main(String[] args)
    {

        linkedList li = new linkedList();
        /*
         * Let us create a unsorted linked lists to test the functions Created
         * lists shall be a: 2->3->20->5->10->15
         */
        li.push(15);
        li.push(10);
        li.push(5);
        li.push(20);
        li.push(3);
        li.push(2);
        System.out.println("Linked List without sorting is :");
        li.printList(li.head);

        // Apply merge Sort
        li.head = li.mergeSort(li.head);
        System.out.print("\n Sorted Linked List is: \n");
        li.printList(li.head);
    }
}
```

// This code is contributed by Rishabh Mahrsee

Time Complexity: O(n Log n)

161. Which sorting algorithm makes minimum number of memory writes?

Minimizing the number of writes is useful when making writes to some huge data set is very expensive, such as with

EEPROMs or Flash memory, where each write reduces the lifespan of the memory.

Among the sorting algorithms that we generally study in our data structure and algorithm courses, Selection Sort makes least number of writes (it makes O(n) swaps). But, Cycle Sort almost always makes less number of writes compared to Selection Sort. In Cycle Sort, each value is either written zero times, if it's already in its correct position, or written one time to its correct position. This matches the minimal number of overwrites required for a completed in-place sort.

162. When does the worst case of Quicksort occur?

The answer depends on strategy for choosing pivot. In early versions of Quick Sort where leftmost (or rightmost) element is chosen as pivot, the worst occurs in following cases.

1) Array is already sorted in same order.2) Array is already sorted in reverse order.3) All elements are same (special case of case 1 and 2)

Since these cases are very common use cases, the problem was easily solved by choosing either a random index for the pivot, choosing the middle index of the partition or (especially for longer partitions) choosing the median of the first, middle and last element of the partition for the pivot. With these modifications, the worst case of Quick sort has less chances to occur, but worst case can still occur if the input array is such that the maximum (or minimum) element is always chosen as pivot.

163. Sorting Strings using Bubble Sort

Given an array of strings arr[]. Sort given strings using Bubble Sort and display the sorted array.

In Bubble Sort, the two successive strings arr[i] and arr[i+1] are exchanged whenever arr[i]> arr[i+1]. The larger values sink to the bottom and hence called sinking sort. At the end of each pass, smaller values gradually "bubble" their way upward to the top and hence called bubble sort.

After all the passes, we get all the strings in sorted order. Complexity of the above algorithm will be $O(N^2)$.

Let us look at the code snippet:

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 100

void sortStrings(char arr[][MAX], int n)
{
    char temp[MAX];

    // Sorting strings using bubble sort
    for (int j=0; j<n-1; j++)
    {
        for (int i=j+1; i<n; i++)
        {
            if (strcmp(arr[j], arr[i]) > 0)
            {
                strcpy(temp, arr[j]);
                strcpy(arr[j], arr[i]);
                strcpy(arr[i], temp);
            }
        }
    }
}

int main()
{
    char arr[][MAX] = {"GeeksforGeeks","Quiz","Practice","Gblogs","Coding"};
    int n = sizeof(arr)/sizeof(arr[0]);

    sortStrings(arr, n);

    printf("Strings in sorted order are : ");
    for (int i=0; i<n; i++)
        printf("\n String %d is %s", i+1, arr[i]);
```

```
    return 0;
}
```

Output:

```
Strings in sorted order are :
 String 1 is Coding
 String 2 is Gblogs
 String 3 is GeeksforGeeks
 String 4 is Practice
 String 5 is Quiz
```

This article is contributed by **Rahul Agrawal**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

164. Sorting Strings using Bubble Sort

Given an array of strings arr[]. Sort given strings using Bubble Sort and display the sorted array.

In Bubble Sort, the two successive strings arr[i] and arr[i+1] are exchanged whenever arr[i]> arr[i+1]. The larger values sink to the bottom and hence called sinking sort. At the end of each pass, smaller values gradually “bubble” their way upward to the top and hence called bubble sort.

After all the passes, we get all the strings in sorted order. Complexity of the above algorithm will be $O(N^2)$.

Let us look at the code snippet:

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 100

void sortStrings(char arr[][MAX], int n)
{
    char temp[MAX];

    // Sorting strings using bubble sort
    for (int j=0; j<n-1; j++)
    {
        for (int i=j+1; i<n; i++)
        {
            if (strcmp(arr[j], arr[i]) > 0)
            {
                strcpy(temp, arr[j]);
                strcpy(arr[j], arr[i]);
                strcpy(arr[i], temp);
            }
        }
    }
}

int main()
{
```

```
    char arr[][MAX] = {"GeeksforGeeks","Quiz","Practice","Gblogs","Coding"};
    int n = sizeof(arr)/sizeof(arr[0]);

    sortStrings(arr, n);

    printf("Strings in sorted order are : ");
    for (int i=0; i<n; i++)
        printf("\n String %d is %s", i+1, arr[i]);
    return 0;
}
```

Output:

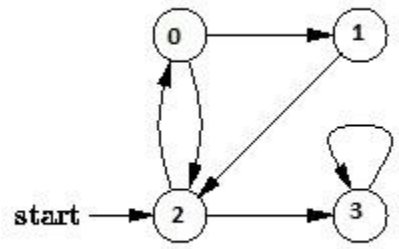
```
Strings in sorted order are :
 String 1 is Coding
 String 2 is Gblogs
 String 3 is GeeksforGeeks
 String 4 is Practice
 String 5 is Quiz
```

This article is contributed by **Rahul Agrawal**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

165. Breadth First Traversal or BFS for a Graph

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree (See method 2 of this post). The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.



Following are C++ and Java implementations of simple Breadth First Traversal from a given source.

The C++ implementation uses adjacency list representation of graphs. STL's list container is used to store lists of adjacent nodes and queue of nodes needed for BFS traversal.

```
// Java program to print BFS traversal from a given source vertex.
// BFS(int s) traverses vertices reachable from s.
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list
// representation
class Graph
{
    private int V;   // No. of vertices
    private LinkedList<Integer> adj[]; //Adjacency Lists

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v,int w)
    {
        adj[v].add(w);
    }

    // prints BFS traversal from a given source s
    void BFS(int s)
    {
        // Mark all the vertices as not visited(By default
        // set as false)
        boolean visited[] = new boolean[V];

        // Create a queue for BFS
        LinkedList<Integer> queue = new LinkedList<Integer>();

        // Mark the current node as visited and enqueue it
        visited[s]=true;
        queue.add(s);

        while (queue.size() != 0)
        {
            // Dequeue a vertex from queue and print it
            s = queue.poll();
            System.out.print(s+" ");

            // Get all adjacent vertices of the dequeued vertex s
            // If a adjacent has not been visited, then mark it
            // visited and enqueue it
            Iterator<Integer> i = adj[s].listIterator();
            while (i.hasNext())
            {
                int n = i.next();
                if (!visited[n])
                {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }
```

```
    // Driver method to
    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Breadth First Traversal "+
                           "(starting from vertex 2)");

        g.BFS(2);
    }
}
// This code is contributed by Aakash Hasija
```

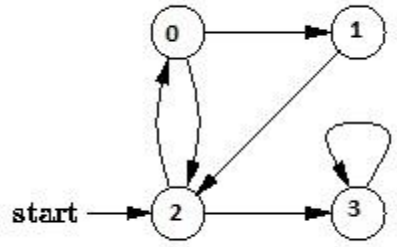
Output:

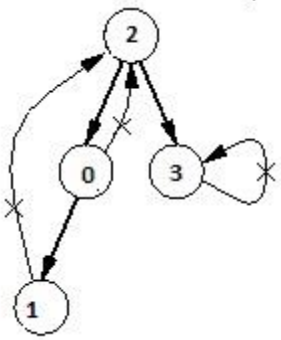
```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

166. Depth First Traversal or DFS for a Graph

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.





See this post for all applications of Depth First Traversal.Following are implementations of simple Depth First Traversal. The C++ implementation uses adjacency list representation of graphs. STL's list container is used to store lists of adjacent nodes.

```
// Java program to print DFS traversal from a given given graph
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list
```

```
// representation
class Graph
{
    private int V;   // No. of vertices

    // Array  of lists for Adjacency List Representation
    private LinkedList<Integer> adj[];

    // Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    //Function to add an edge into the graph
    void addEdge(int v, int w)
    {
        adj[v].add(w);  // Add w to v's list.
    }

    // A function used by DFS
    void DFSUtil(int v,boolean visited[])
    {
        // Mark the current node as visited and print it
        visited[v] = true;
        System.out.print(v+" ");

        // Recur for all the vertices adjacent to this vertex
        Iterator<Integer> i = adj[v].listIterator();
        while (i.hasNext())
        {
            int n = i.next();
            if (!visited[n])
                DFSUtil(n, visited);
        }
    }

    // The function to do DFS traversal. It uses recursive DFSUtil()
    void DFS(int v)
    {
        // Mark all the vertices as not visited(set as
        // false by default in java)
        boolean visited[] = new boolean[V];

        // Call the recursive helper function to print DFS traversal
        DFSUtil(v, visited);
    }

    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Depth First Traversal "+
                           "(starting from vertex 2)");
```

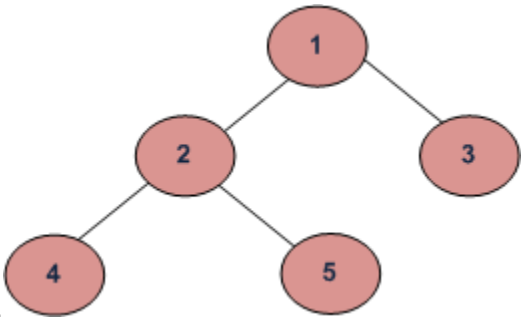
```
            g.DFS(2);
        }
}
```

// This code is contributed by Aakash Hasija

167. Find Minimum Depth of a Binary Tree

Given a binary tree, find its minimum depth. The minimum depth is the number of nodes along the shortest path from root node down to the nearest leaf node.

For example, minimum height of below Binary Tree is 2.



Note that the path must end on a leaf node. For example, minimum height of below Binary Tree is also 2.

```
        10
      /
5
```

Strings

168. Write a Java program to concatenate a given string to the end of another string.

```
//str1.concat(str2);
public class Exercise7 {
    public static void main(String[] args)
    {
        String str1 = "PHP Exercises and ";
        String str2 = "Python Exercises";

        System.out.println("String 1: " + str1);
        System.out.println("String 2: " + str2);

        // Concatenate the two strings together.
        String str3 = str1.concat(str2);
        // Display the new String.
        System.out.println("The concatenated string: " + str3);
    }
}
```

Sample Output:

```
String 1: PHP Exercises and
String 2: Python Exercises
The concatenated string: PHP Exercises and Python Exercises
```

169. Write a Java program to test if a given string contains the specified sequence of char values.

```
//str1.contains(str2)
public class Exercise8 {
    public static void main(String[] args){
        String str1 = "PHP Exercises and Python Exercises";
        String str2 = "and";
        System.out.println("Original String: " + str1);
        System.out.println("Specified sequence of char values: " + str2);
        System.out.println(str1.contains(str2));
```

```
    }
}
```

Sample Output:

```
Original String: PHP Exercises and Python Exercises
Specified sequence of char values: and
true
```

170. Write a Java program to compare a given string to the specified character sequence.

```
//str1.contentEquals(cs)
public class Exercise9 {
   public static void main(String[] args) {
      String str1 = "example.  com", str2 = "Example.  com";
      CharSequence cs = "example.  com";
      System.out.println("Comparing "+str1+" and "+cs+": " + str1.contentEquals(cs));
      System.out.println("Comparing "+str2+" and "+cs+": " + str2.contentEquals(cs));
   }
}
```

Sample Output:

```
Comparing example.  com and example.  com: true
Comparing Example.  com and example.  com: false
```

171. Write a Java program to compare a given string to the specified string buffer.

```
//str1.contentEquals(strbuf))
public class Exercise10 {
public static void main(String[] args) {
    String str1 = "example.  com", str2 = "Example.  com";
    StringBuffer strbuf = new StringBuffer(str1);

    System.out.println("Comparing "+str1+" and "+strbuf+": " + str1.contentEquals(strbuf));

    System.out.println("Comparing "+str2+" and "+strbuf+": " + str2.contentEquals(strbuf));

      }
}
```

Sample Output:

```
Comparing example.  com and example.  com: true
Comparing Example.  com and example.  com: false
```

172. Write a Java program to check whether a given string ends with the contents of another string.

```
//str1.endsWith(end_str);
public class Exercise12 {
    public static void main(String[] args)
    {
        String str1 = "Python Exercises";
        String str2 = "Python Exercise";
        // The String to check the above two Strings to see
        // if they end with this value (se).
        String end_str = "se";
        // Check first two Strings end with end_str
        boolean ends1 = str1.endsWith(end_str);
        boolean ends2 = str2.endsWith(end_str);
        // Display the results of the endsWith calls.
        System.out.println("\"" + str1 + "\" ends with " +
            "\"" + end_str + "\"? " + ends1);
        System.out.println("\"" + str2 + "\" ends with " +
            "\"" + end_str + "\"? " + ends2);
    }
}
```

Sample Output:

```
"Python Exercises" ends with "se"? false
```

```
"Python Exercise" ends with "se"? true
```

173. Write a Java program to check whether two String objects contain the same data.

```
// columnist1.equals(columnist2);
public class Exercise13 {
   public static void main(String[] args)
    {
        String columnist1 = "Stephen Edwin King";
        String columnist2 = "Walter Winchell";
        String columnist3 = "Mike Royko";
        // Are any of the above Strings equal to one another?
        boolean equals1 = columnist1.equals(columnist2);
        boolean equals2 = columnist1.equals(columnist3);
        // Display the results of the equality checks.
        System.out.println("\"" + columnist1 + "\" equals \"" +
            columnist2 + "\"? " + equals1);
        System.out.println("\"" + columnist1 + "\" equals \"" +
            columnist3 + "\"? " + equals2);
    }
}
```

Sample Output:

```
"Stephen Edwin King" equals "Walter Winchell"? false
"Stephen Edwin King" equals "Mike Royko"? false
```

174. Write a Java program to compare a given string to another string, ignoring case considerations.

```
public class Exercise14 {
  public static void main(String[] args)
    {
        String columnist1 = "Stephen Edwin King";
        String columnist2 = "Walter Winchell";
        String columnist3 = "stephen edwin king";
        // Test any of the above Strings equal to one another
        boolean equals1 = columnist1.equalsIgnoreCase(columnist2);
        boolean equals2 = columnist1.equalsIgnoreCase(columnist3);
        // Display the results of the equality checks.
        System.out.println("\"" + columnist1 + "\" equals \"" +
            columnist2 + "\"? " + equals1);
        System.out.println("\"" + columnist1 + "\" equals \"" +
            columnist3 + "\"? " + equals2);
    }
}
```

Sample Output:

```
"Stephen Edwin King" equals "Walter Winchell"? false
"Stephen Edwin King" equals "stephen edwin king"? true
```

175. Write a Java program to get the contents of a given string as a byte array.

```
import java.  util.Calendar;
public class Exercise16 {

  public static void main(String[] args)
    {
        String str = "This is a sample String.";
        // Copy the contents of the String to a byte array.
```

```
        byte[] byte_arr = str.getBytes();
        // Create a new String using the contents of the byte array.
        String new_str = new String(byte_arr);
        // Display the contents of the byte array.
        System.out.println("\nThe new String equals " +
            new_str + "\n");
    }
}
```

Sample Output:

```
The new String equals This is a sample String.
```

176. Write a Java program to get the contents of a given string as a character array.

```
public class Exercise17 {

  public static void main(String[] args)
    {
        String str = "This is a sample string.";
        // Copy the contents of the String to a byte array.
        // Only copy characters 4 through 10 from str.
        // Fill the source array starting at position 2.
        char[] arr = new char[] { ' ', ' ', ' ', ' ',
                                  ' ', ' ', ' ', ' ' };
        str.getChars(4, 10, arr, 2);
        // Display the contents of the byte array.
        System.out.println("The char array equals \"" +
            arr + "\"");
    }
}
```

Sample Output:

```
The char array equals "[C@2a139a55"
```

177. Write a Java program to create a unique identifier of a given string.

```
public class Exercise18 {

  public static void main(String[] args)
    {
        String str = "Python Exercises.";
        // Get the hash code for the above string.
        int hash_code = str.hashCode();
        // Display the hash code.
        System.out.println("The hash for " + str +
            " is " + hash_code);
    }
}
```

Sample Output:

```
The hash for Python Exercises. is 863132599
```

178. Write a Java program to get the index of all the characters of the alphabet.

Sample string of all alphabet: "The quick brown fox jumps over the lazy dog."

```
public class Exercise19 {

  public static void main(String[] args)
    {
        String str = "The quick brown fox jumps over the lazy dog.";
        // Get the index of all the characters of the alphabet
        // starting from the beginning of the String.
        int a = str.indexOf("a", 0);
        int b = str.indexOf("b", 0);
        int c = str.indexOf("c", 0);
        int d = str.indexOf("d", 0);
        int e = str.indexOf("e", 0);
        int f = str.indexOf("f", 0);
        int g = str.indexOf("g", 0);
        int h = str.indexOf("h", 0);
        int i = str.indexOf("i", 0);
        int j = str.indexOf("j", 0);
        int k = str.indexOf("k", 0);
        int l = str.indexOf("l", 0);
        int m = str.indexOf("m", 0);
        int n = str.indexOf("n", 0);
        int o = str.indexOf("o", 0);
        int p = str.indexOf("p", 0);
        int q = str.indexOf("q", 0);
        int r = str.indexOf("r", 0);
        int s = str.indexOf("s", 0);
        int t = str.indexOf("t", 0);
        int u = str.indexOf("u", 0);
        int v = str.indexOf("v", 0);
        int w = str.indexOf("w", 0);
        int x = str.indexOf("x", 0);
        int y = str.indexOf("y", 0);
        int z = str.indexOf("z", 0);
        // Display the results of all the indexOf method calls.
        System.out.println(" a  b c  d e  f  g h i  j");
        System.out.println("========================");
        System.out.println(a + " " + b + " " + c + " " + d + " " +
                           e + " " + f + " " + g + " " + h + " " +
                           i + " " + j + "\n");
        System.out.println("k  l  m  n  o  p q  r  s  t");
        System.out.println("==========================");
        System.out.println(k + " " + l + " " + m + " " + n + " " +
                           o + " " + p + " " + q + " " + r + " " +
                           s + " " + t + "\n");
        System.out.println("u  v  w  x  y  z");
        System.out.println("================");
        System.out.println(u + " " + v + " " + w + " " + x + " " +
                           y + " " + z);
    }
}
```

Sample Output:

```
a  b c  d e  f  g h i  j
=========================
36 10 7 40 2 16 42 1 6 20

k  l  m  n  o  p q  r  s  t
============================
8 35 22 14 12 23 4 11 24 31
u  v  w  x  y  z
================
5 27 13 18 38 3
```

---

179. Write a Java program to get the canonical representation of the string object.

```
public class Exercise20 {
public static void main(String[] args)
    {
        // Create three strings in three different ways.
        String str1 = "Java Exercises";
        String str2 = new StringBuffer("Java").append(" Exercises").toString();
        String str3 = str2.intern();
        // Determine which strings are equivalent using the ==
        // operator (as compared to calling equals(), which is
        // a more expensive operation.
        System.out.println("str1 == str2? " + (str1 == str2));
        System.out.println("str1 == str3? " + (str1 == str3));
    }
}
```

Sample Output:

```
str1 == str2? false
str1 == str3? true
```

180. Write a Java program to get the last index of a string within a string.

Sample string of all alphabet: "The quick brown fox jumps over the lazy dog."

```
public class Exercise21 {
public static void main(String[] args)
    {
        String str = "The quick brown fox jumps over the lazy dog.";
        // Get the index of all the characters of the alphabet
        // starting from the beginning of the String.
        int a = str.lastIndexOf("a", str.length() - 1);
        int b = str.lastIndexOf("b", str.length() - 1);
        int c = str.lastIndexOf("c", str.length() - 1);
        int d = str.lastIndexOf("d", str.length() - 1);
        int e = str.lastIndexOf("e", str.length() - 1);
        int f = str.lastIndexOf("f", str.length() - 1);
        int g = str.lastIndexOf("g", str.length() - 1);
        int h = str.lastIndexOf("h", str.length() - 1);
        int i = str.lastIndexOf("i", str.length() - 1);
        int j = str.lastIndexOf("j", str.length() - 1);
        int k = str.lastIndexOf("k", str.length() - 1);
        int l = str.lastIndexOf("l", str.length() - 1);
```

```
        int m = str.lastIndexOf("m", str.length() - 1);
        int n = str.lastIndexOf("n", str.length() - 1);
        int o = str.lastIndexOf("o", str.length() - 1);
        int p = str.lastIndexOf("p", str.length() - 1);
        int q = str.lastIndexOf("q", str.length() - 1);
        int r = str.lastIndexOf("r", str.length() - 1);
        int s = str.lastIndexOf("s", str.length() - 1);
        int t = str.lastIndexOf("t", str.length() - 1);
        int u = str.lastIndexOf("u", str.length() - 1);
        int v = str.lastIndexOf("v", str.length() - 1);
        int w = str.lastIndexOf("w", str.length() - 1);
        int x = str.lastIndexOf("x", str.length() - 1);
        int y = str.lastIndexOf("y", str.length() - 1);
        int z = str.lastIndexOf("z", str.length() - 1);
        // Display the results of all the lastIndexOf method calls.
        System.out.println(" a  b c  d  e  f  g  h i  j");
        System.out.println("==========================");
        System.out.println(a + " " + b + " " + c + " " + d + " " +
                           e + " " + f + " " + g + " " + h + " " +
                           i + " " + j + "\n");
        System.out.println("k  l  m  n  o  p q  r  s  t");
        System.out.println("==========================");
        System.out.println(k + " " + l + " " + m + " " + n + " " +
                           o + " " + p + " " + q + " " + r + " " +
                           s + " " + t + "\n");
        System.out.println(" u  v  w  x  y  z");
        System.out.println("================");
        System.out.println(u + " " + v + " " + w + " " + x + " " +
                           y + " " + z);
    }
}
```

Sample Output:

```
 a  b c  d  e  f  g  h i  j
==========================
36 10 7 40 33 16 42 32 6 20

k  l  m  n  o  p q  r  s  t
==========================
8 35 22 14 41 23 4 29 24 31

 u  v  w  x  y  z
================
21 27 13 18 38 3
```

181. Write a java program to get the length of a given string.

```
public class Exercise22 {
public static void main(String[] args)
  {
    String str = "example.  com";
    // Get the length of str.
    int len = str.length();
    System.out.println("The string length of '"+str+"' is: "+len);
   }
```

```
}
```

Sample Output:

```
The string length of 'example.  com' is: 1
```

182. Write a Java program to find whether a region in the current string matches a region in another string.

```
public class Exercise23 {

  public static void main(String[] args)
    {
       //String str1 = "Red Green Orange Yellow";
        //String str2 = "Yellow Orange Green Red";

        String str1 = "Shanghai Houston Colorado Alexandria";
        String str2 = "Alexandria Colorado Houston Shanghai";
        // Determine whether characters 0 through 7 in str1
        // match characters 28 through 35 in str2.
        boolean match1 = str1.regionMatches(0, str2, 28, 8);
        // Determine whether characters 9 through 15 in str1
        // match characters 9 through 15 in str2.
        boolean match2 = str1.regionMatches(9, str2, 9, 8);
        // Display the results of the regionMatches method calls.
        System.out.println("str1[0 - 7] == str2[28 - 35]? " + match1);
        System.out.println("str1[9 - 15] == str2[9 - 15]? " + match2);
    }
}
```

Sample Output:

```
str1[0 - 7] == str2[28 - 35]? true
str1[9 - 15] == str2[9 - 15]? false
```

183. Write a Java program to replace all the 'd' characters with 'f' characters.

```
public class Exercise24 {

   public static void main(String[] args)
    {
        String str = "The quick brown fox jumps over the lazy dog.";
        // Replace all the 'd' characters with 'f' characters.
        String new_str = str.replace('d', 'f');
        // Display the strings for comparison.
        System.out.println("Original string: " + str);
        System.out.println("New String: " + new_str);
    }
}
```

Sample Output:

```
Original string: The quick brown fox jumps over the lazy dog.
New String: The quick brown fox jumps over the lazy fog.
```

184. Write a Java program to replace each substring of a given string that matches the given regular expression with the given replacement.

Sample string: "The quick brown fox jumps over the lazy dog."
In the above string replace all the fox with cat.

```
public class Exercise25 {

   public static void main(String[] args)
    {
        String str = "The quick brown fox jumps over the lazy dog.";
         // Replace all the 'dog' with 'cat'.
        String new_str = str.replaceAll("fox", "cat");

         // Display the strings for comparison.
        System.out.println("Original string: " + str);
        System.out.println("New String: " + new_str);
    }
}
```

Sample Output:

```
Original string: The quick brown fox jumps over the lazy dog.
New String: The quick brown cat jumps over the lazy dog.
```

185. Write a Java program to check whether a given string starts with the contents of another string.

```
public class Exercise26 {

   public static void main(String[] args)
    {
        String str1 = "Red is favorite color.";
        String str2 = "Orange is also my favorite color.";
        // The String to check the above two Strings to see
        // if they start with this value (Red).
        String startStr = "Red";
        // Do either of the first two Strings start with startStr?
        boolean starts1 = str1.startsWith(startStr);
        boolean starts2 = str2.startsWith(startStr);
        // Display the results of the startsWith calls.
        System.out.println( str1 + " starts with " +
             startStr + "? " + starts1);
        System.out.println(str2 + " starts with " +
             startStr + "? " + starts2);
    }
}
```

Sample Output:

```
Red is favorite color. starts with Red? true
Orange is also my favorite color. starts with Red? false
```

186. Write a Java method to find the smallest number among three numbers.

Test Data:

Input the first number: 25

Input the Second number: 37

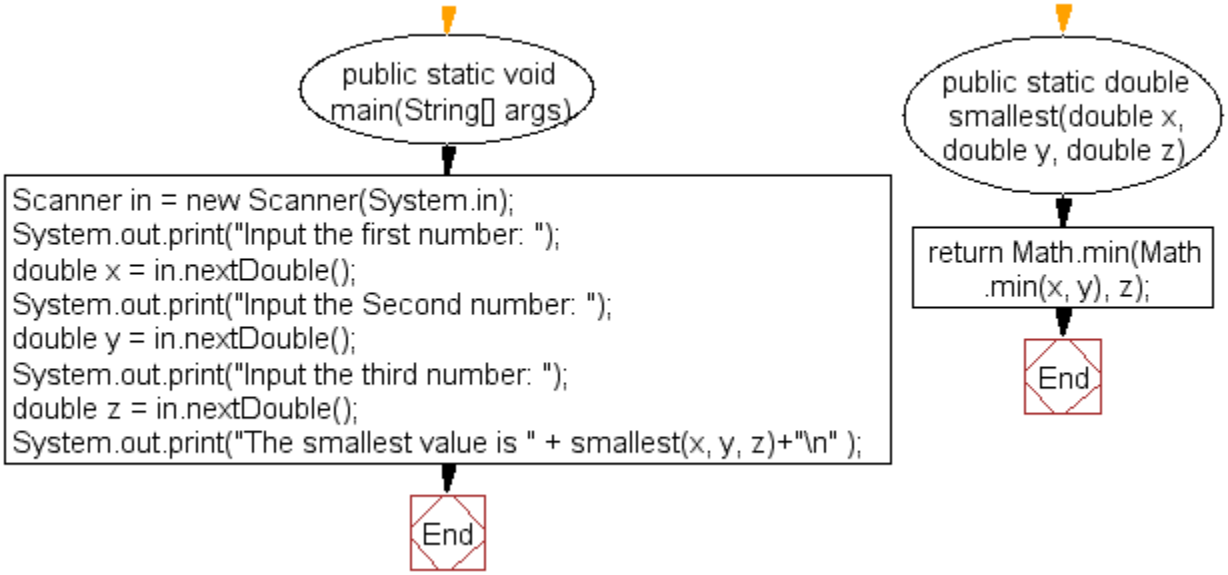
Input the third number: 2

```
import java.  util.Scanner;
public class Exercise1 {
 public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input the first number: ");
        double x = in.nextDouble();
        System.out.print("Input the Second number: ");
        double y = in.nextDouble();
        System.out.print("Input the third number: ");
        double z = in.nextDouble();
        System.out.print("The smallest value is " + smallest(x, y, z)+"\n" );
    }
   public static double smallest(double x, double y, double z)
    {
        return Math.min(Math.min(x, y), z);
    }
}
```

Sample Output:

```
Input the first number: 25
Input the Second number: 37
Input the third number: 29

The smallest value is 25.
```

Flowchart:



187. Write a Java method to compute the average of three numbers.

Test Data:

Input the first number: 25

Input the second number: 45

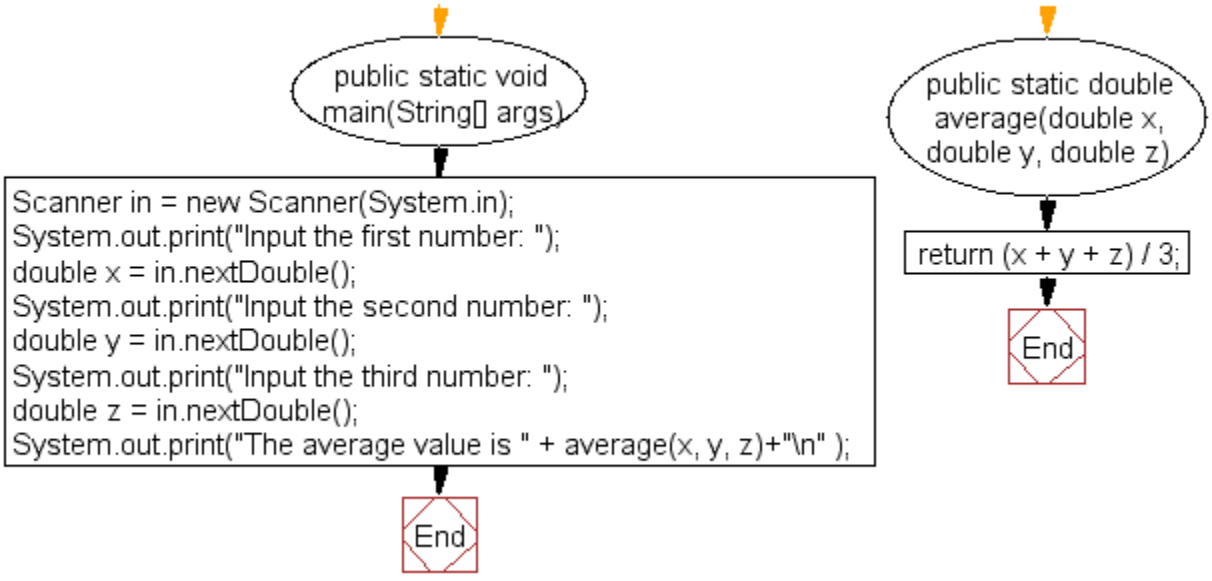
Input the third number: 6

```
import java.  util.Scanner;
public class Exercise2 {
 public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input the first number: ");
        double x = in.nextDouble();
        System.out.print("Input the second number: ");
        double y = in.nextDouble();
        System.out.print("Input the third number: ");
        double z = in.nextDouble();
        System.out.print("The average value is " + average(x, y, z)+"\n" );
    }
  public static double average(double x, double y, double z)
    {
        return (x + y + z) / 3;
    }
}
```

Sample Output:

```
Input the first number: 25
Input the second number: 45
Input the third number: 65

The average value is 45.
```

Flowchart:



188. Write a Java method to display the middle character of a string.

Note: a) If the length of the string is odd there will be two middle characters.

b) If the length of the string is even there will be one middle character.

Test Data:

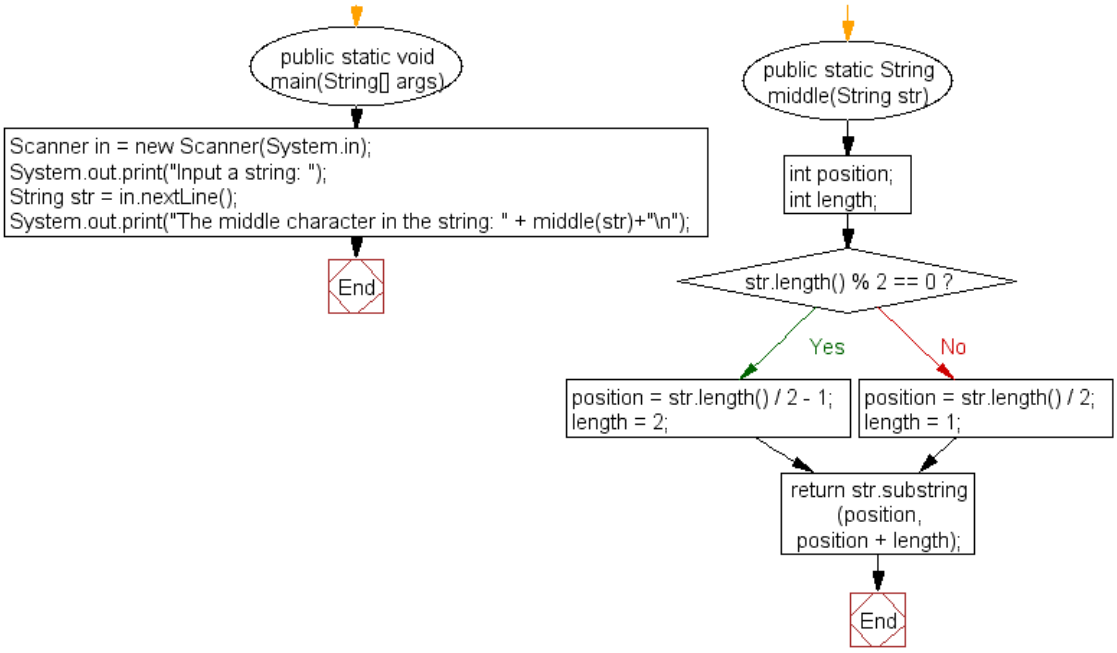
Input a string: 35

```
import java.  util.Scanner;
public class Exercise3 {
  public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input a string: ");
        String str = in.nextLine();
        System.out.print("The middle character in the string: " + middle(str)+"\n");
    }
 public static String middle(String str)
    {
        int position;
        int length;
        if (str.length() % 2 == 0)
        {
            position = str.length() / 2 - 1;
            length = 2;
        }
        else
        {
            position = str.length() / 2;
            length = 1;
        }
        return str.substring(position, position + length);
    }
}
```

Sample Output:

```
Input a string: 350
The middle character in the string:
```

Flowchart:



189. Write a Java method to count all vowels in a string.

Test Data:

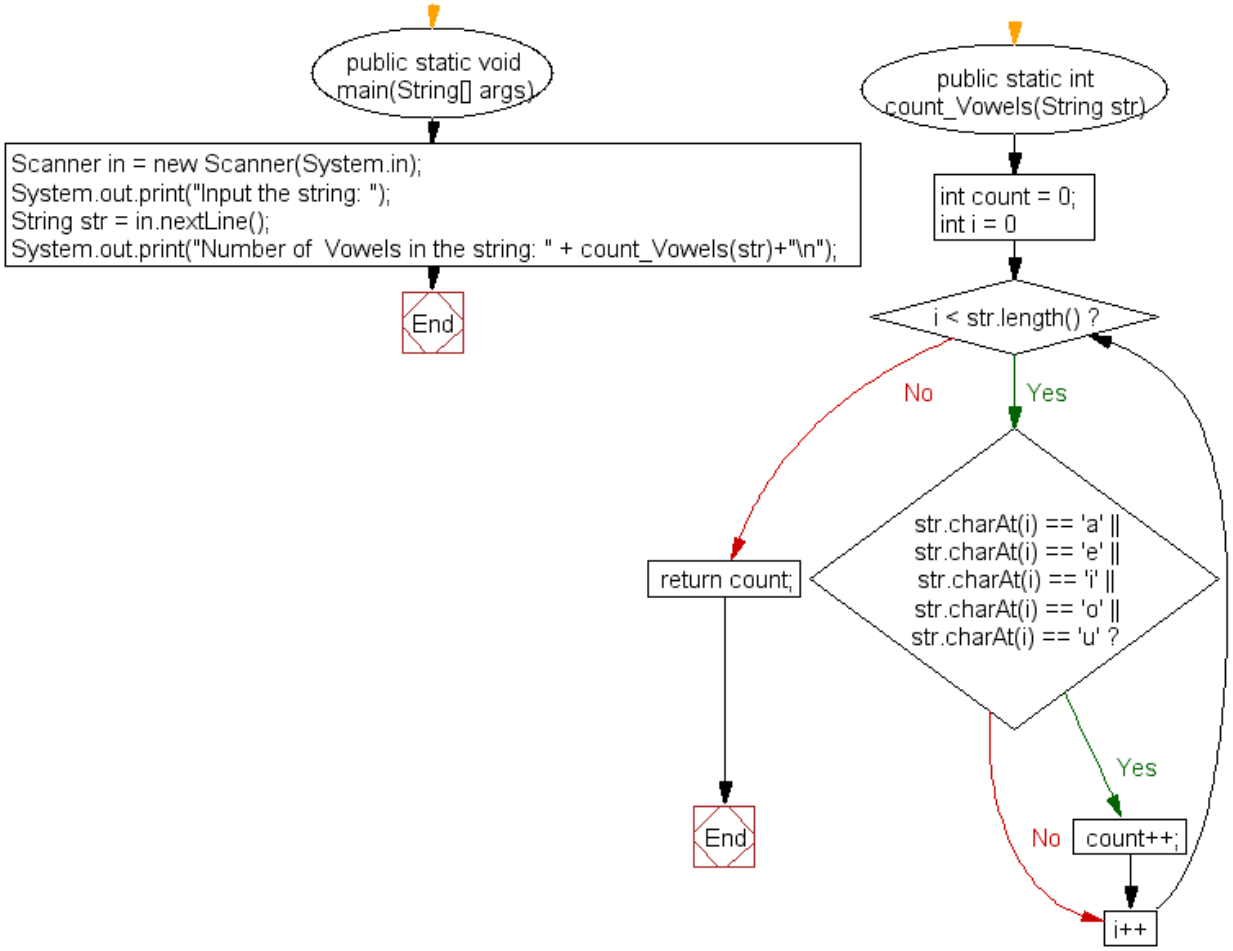
Input the string: w3resource

```
import java.  util.Scanner;
public class Exercise4 {
  public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input the string: ");
        String str = in.nextLine();
        System.out.print("Number of  Vowels in the string: " + count_Vowels(str)+"\n");
    }
 public static int count_Vowels(String str)
    {
        int count = 0;
        for (int i = 0; i < str.length(); i++)
        {
            if (str.charAt(i) == 'a' || str.charAt(i) == 'e' || str.charAt(i) == 'i'
                    || str.charAt(i) == 'o' || str.charAt(i) == 'u')
            {
                count++;
            }
        }
        return count;
    }
}
```

Sample Output:

```
Input the string: w3resource
Number of  Vowels in the string:
```

Flowchart:



190. Write a Java method to count all words in a string.

Test Data:

Input the string: The quick brown fox jumps over the lazy dog.

```
import java.  util.Scanner;
public class Exercise5 {
  public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input the string: ");
        String str = in.nextLine();
        System.out.print("Number of words in the string: " + count_Words(str)+"\n");
    }
 public static int count_Words(String str)
    {
       int count = 0;
```

```
        if (!(" ".equals(str.substring(0, 1))) || !(" ".equals(str.substring(str.length() -
1))))
        {
            for (int i = 0; i < str.length(); i++)
            {
                if (str.charAt(i) == ' ')
                {
                    count++;
                }
            }
            count = count + 1;
        }
        return count; // returns 0 if string starts or ends with space " ".
    }
}
```

Sample Output:

```
Input the string: The quick brown fox jumps over the lazy dog
Number of words in the string: 9
```

Flowchart:

191. Write a Java method to compute the sum of the digits in an integer.

Test Data:

Input an integer: 2

```
import java.  util.Scanner;
public class Exercise6 {
  public static void main(String[] args)
    {
      Scanner in = new Scanner(System.in);
      System.out.print("Input an integer: ");
      int digits = in.nextInt();
     System.out.println("The sum is " + sumDigits(digits));
    }
 public static int sumDigits(long n) {
     int result = 0;

     while(n > 0) {
       result += n % 10;
       n /= 10;
     }

     return result;
   }

}
```

Sample Output:

```
Input an integer: 25

The sum is
```

192. Write a Java method to display the first 50 pentagonal numbers.

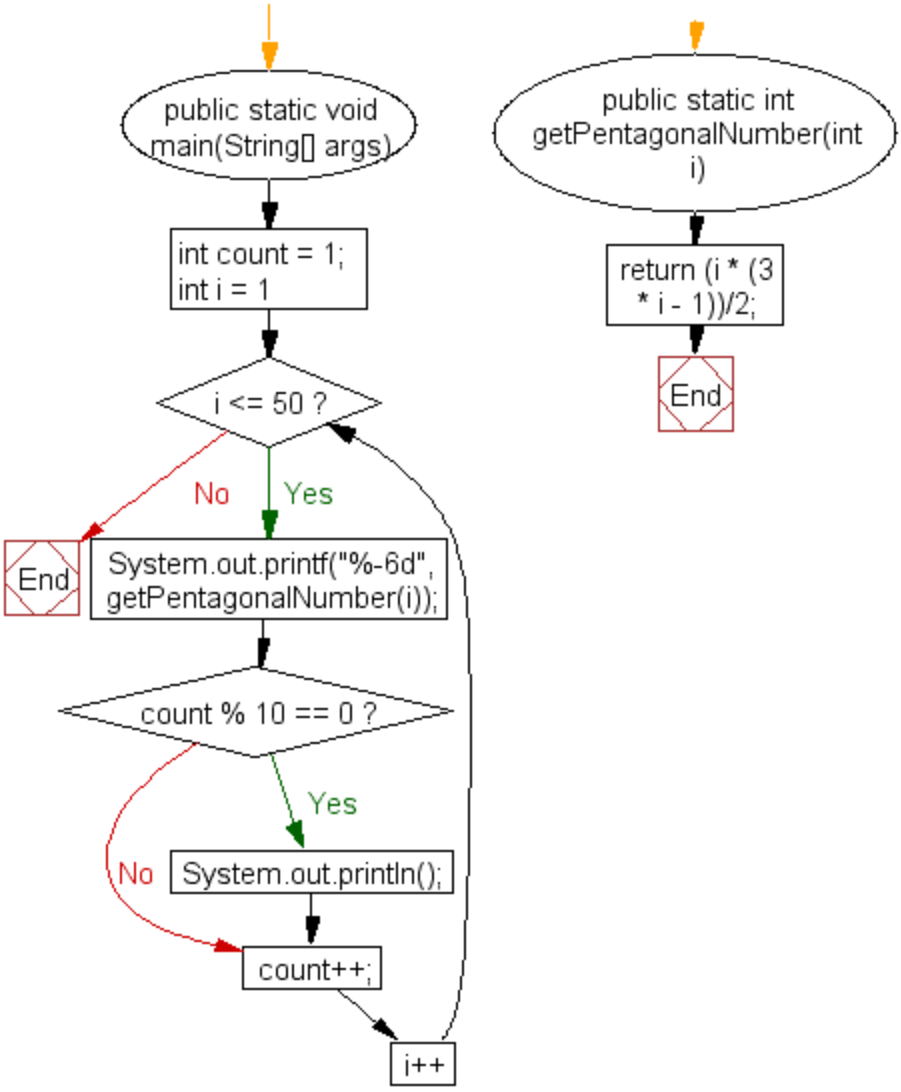
A pentagonal number is a figurate number that extends the concept of triangular and square numbers to the pentagon, but, unlike the first two, the patterns involved in the construction of pentagonal numbers are not rotationally symmetrical.

```
import java.  util.Scanner;
public class Exercise7 {
public static void main(String[] args) {
     int count = 1;
     for(int i = 1; i <= 50; i++){
       System.out.printf("%-6d",getPentagonalNumber(i));
       if(count % 10 == 0) System.out.println();
       count++;
     }
    }
  public static int getPentagonalNumber(int i) {
     return (i * (3 * i - 1))/2;
   }
}
```

Sample Output:

```
1     5     12    22    35    51    70    92    117   145
176   210   247   287   330   376   425   477   532   590
651   715   782   852   925   1001  1080  1162  1247  1335
1426  1520  1617  1717  1820  1926  2035  2147  2262  2380

2501  2625  2752  2882  3015  3151  3290  3432  3577  372
```

Flowchart:



193. Write a Java method to compute the future investment value at a given interest rate for a specified number of years.

Sample data (Monthly compounded):

Input the investment amount: 1000

Input the rate of interest: 10

Input number of years:

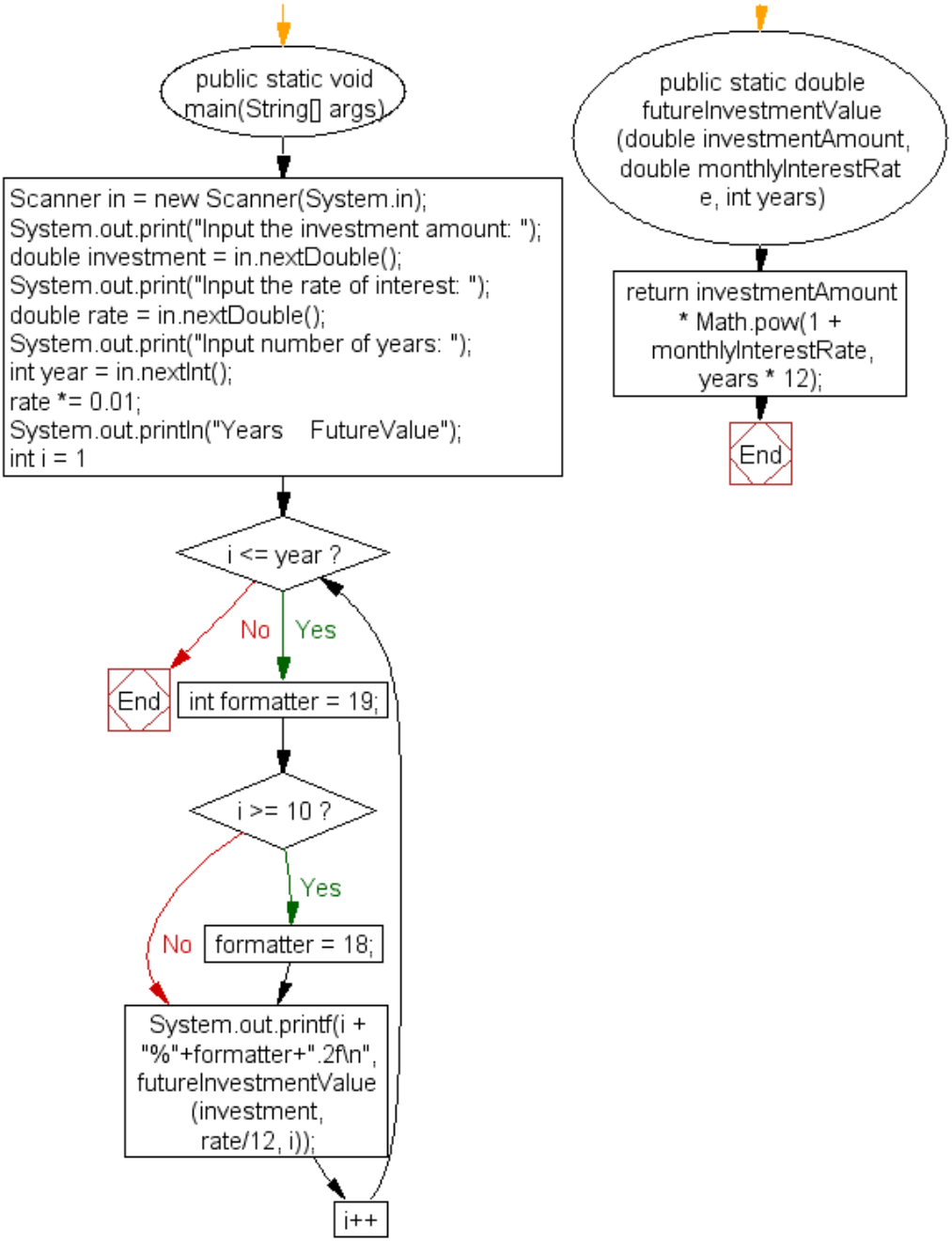
```
import java.  util.Scanner;
public class Exercise8 {

 public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Input the investment amount: ");
   double investment = in.nextDouble();
   System.out.print("Input the rate of interest: ");
```

```
    double rate = in.nextDouble();
    System.out.print("Input number of years: ");
    int year = in.nextInt();

    rate *= 0.01;

    System.out.println("Years    FutureValue");
    for(int i = 1; i <= year; i++) {
      int formatter = 19;
        if (i >= 10) formatter = 18;
      System.out.printf(i + "%"+formatter+".2f\n", futureInvestmentValue(investment, rate/12,
i));
        }
     }

 public static double futureInvestmentValue(double investmentAmount, double
monthlyInterestRate, int years) {
     return investmentAmount * Math.pow(1 + monthlyInterestRate, years * 12);
   }
}
```

Sample Output:

```
Input the investment amount: 1000
Input the rate of interest: 10
Input number of years: 5
Years    FutureValue
1            1104.71
2            1220.39
3            1348.18
4            1489.35

5            1645.3
```

Flowchart:



194. Write a Java method to print characters between two characters (i.e. A to P).

Note: Prints 20 characters per line

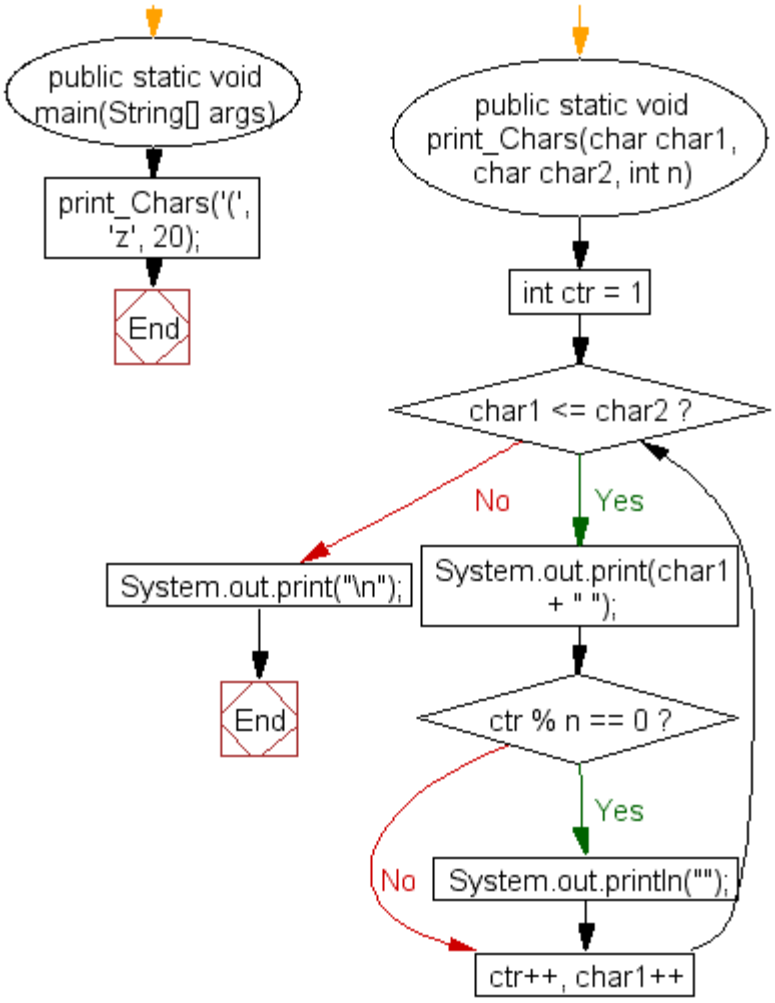
```
public class Exercise9 {
   public static void main(String[] args) {
        print_Chars('(', 'z', 20);
    }
 public static void print_Chars(char char1, char char2, int n) {
        for (int ctr = 1; char1 <= char2; ctr++, char1++) {
            System.out.print(char1 + " ");
```

```
            if (ctr % n == 0) System.out.println("");
        }
      System.out.print("\n");
    }
}
```

Sample Output:

```
( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ;
< = > ? @ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c
d e f g h i j k l m n o p q r s t u v w
x y z
```

Flowchart:



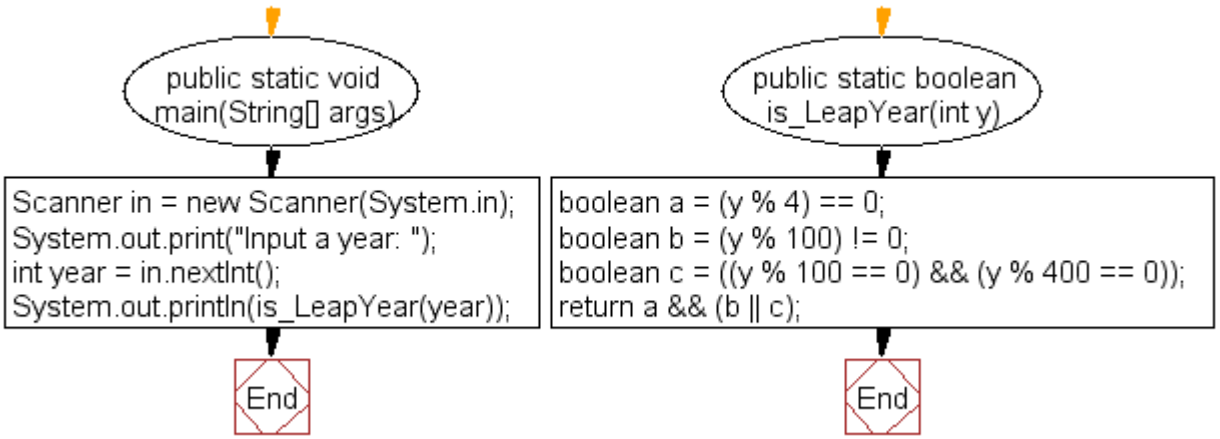
195. Write a Java method to check whether an year (integer) entered by the user is a leap year or not.

```java
import java.  util.Scanner;
public class Exercise10 {

 public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input a year: ");
        int year = in.nextInt();
        System.out.println(is_LeapYear(year));
    }

 public static boolean is_LeapYear(int y)
    {
        boolean a = (y % 4) == 0;
        boolean b = (y % 100) != 0;
        boolean c = ((y % 100 == 0) && (y % 400 == 0));
        return a && (b || c);
    }
}
```

Sample Output:

```
Input a year: 2017
false
```

Flowchart:



196. Write a Java method to check whether a string is a valid password.

Password rules:

A password must have at least ten characters.

A password consists of only letters and digits.

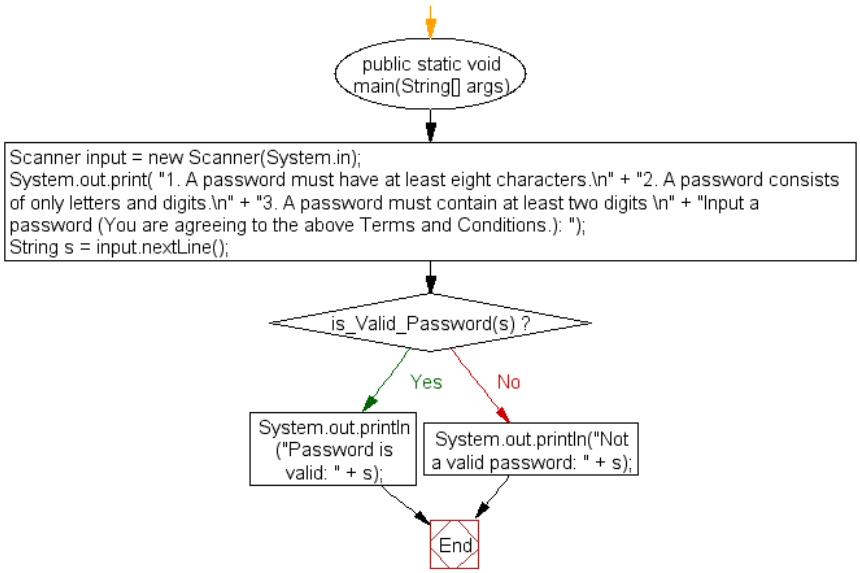
A password must contain at least two digits.

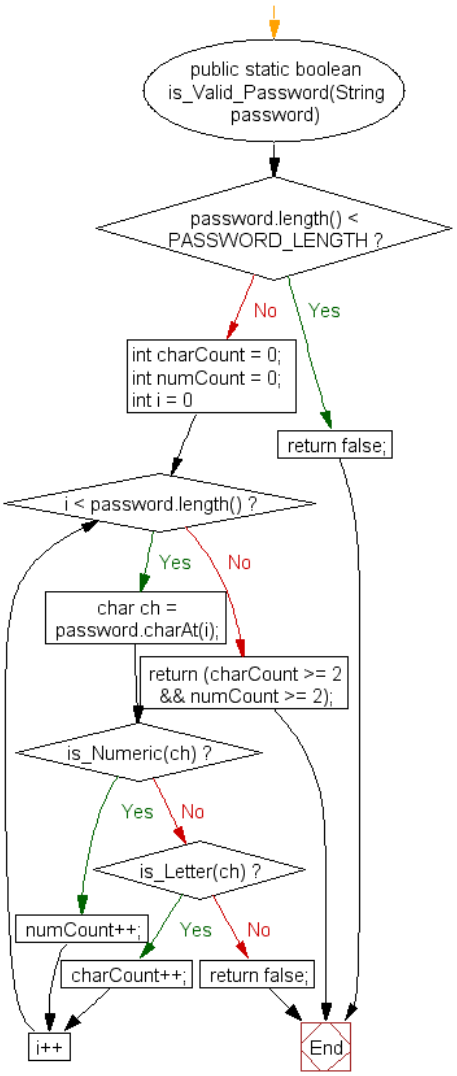
```
import java.  util.Scanner;
public class Exercise11 {

public static final int PASSWORD_LENGTH = 8;
public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print(
                "1. A password must have at least eight characters.\n" +
                "2. A password consists of only letters and digits.\n" +
                "3. A password must contain at least two digits \n" +
                "Input a password (You are agreeing to the above Terms and Conditions.): ");
        String s = input.nextLine();
        if (is_Valid_Password(s)) {
            System.out.println("Password is valid: " + s);
        } else {
            System.out.println("Not a valid password: " + s);
        }
    }
    public static boolean is_Valid_Password(String password) {
        if (password.  length() < PASSWORD_LENGTH) return false;
        int charCount = 0;
        int numCount = 0;
        for (int i = 0; i < password.  length(); i++) {
            char ch = password.  charAt(i);
            if (is_Numeric(ch)) numCount++;
            else if (is_Letter(ch)) charCount++;
            else return false;
        }

        return (charCount >= 2 && numCount >= 2);
    }
    public static boolean is_Letter(char ch) {
        ch = Character.toUpperCase(ch);
        return (ch >= 'A' && ch <= 'Z');
    }

    public static boolean is_Numeric(char ch) {
        return (ch >= '0' && ch <= '9');
    }
}
```

Sample Output:

```
1. A password must have at least eight characters.
2. A password consists of only letters and digits.
3. A password must contain at least two digits
Input a password (You are agreeing to the above Terms and Conditions.): abcd1234
Password is valid: abcd1234
```

Flowchart:





197. Write a Java method (takes a number n as input) to displays an n-by-n matrix.

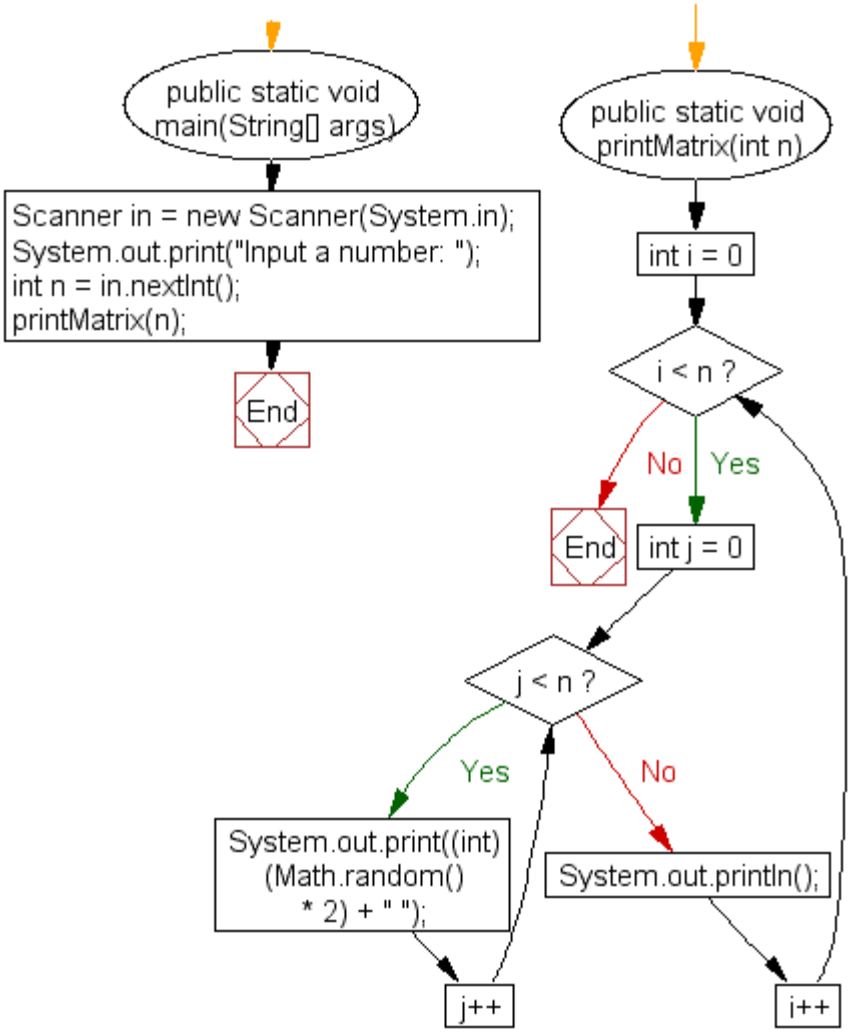
```
import java.  util.Scanner;
public class Exercise12 {

public static void main(String[] args)
    {

        Scanner in = new Scanner(System.in);
        System.out.print("Input a number: ");
        int n = in.nextInt();
        printMatrix(n);
    }

 public static void printMatrix(int n) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                System.out.print((int)(Math.random() * 2) + " ");
            }
```

```
            System.out.println();
        }
    }
}
```

Sample Output:

```
Input a number: 10
1 0 0 1 1 0 0 0 1 1
0 0 1 0 1 0 1 0 0 0
0 1 0 1 0 0 0 0 0 1
1 1 1 0 0 0 0 1 1 1
1 1 0 1 1 1 0 1 0 0
1 0 0 0 1 1 0 0 0 0
0 0 1 0 0 0 0 1 1 1
1 1 0 1 0 1 0 0 1 0
0 0 1 0 0 0 0 1 1 0

1 1 1 0 0 1 1 1 1
```

Flowchart:



198. Write Java methods to calculate the area of a triangle.

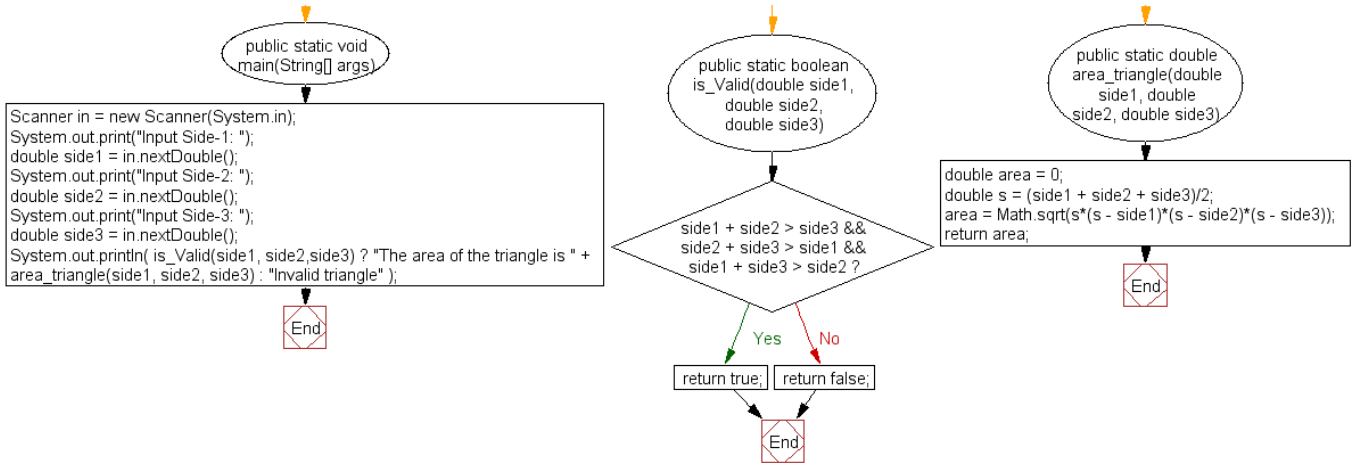
```
import java.  util.Scanner;
public class Exercise13 {
  public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
         System.out.print("Input Side-1: ");
         double side1 = in.nextDouble();
         System.out.print("Input Side-2: ");
         double side2 = in.nextDouble();
         System.out.print("Input Side-3: ");
         double side3 = in.nextDouble();
         System.out.println( is_Valid(side1, side2,side3) ?
                "The area of the triangle is " + area_triangle(side1, side2, side3) : "Invalid
triangle" );
    }
  public static boolean is_Valid(double side1, double side2, double side3) {
        if( side1 + side2 > side3 &&
                side2 + side3 > side1 &&
```

```
                side1 + side3 > side2) return true;
        else return false;
    }
   public static double area_triangle(double side1, double side2, double side3) {
        double area = 0;
        double s = (side1 + side2 + side3)/2;
        area = Math.sqrt(s*(s - side1)*(s - side2)*(s - side3));
        return area;
    }
}
```

Sample Output:

```
Input Side-1: 10
Input Side-2: 15
Input Side-3: 20
The area of the triangle is 72.61843774138907
```

Flowchart:



199. Write a Java method to create the area of a pentagon.

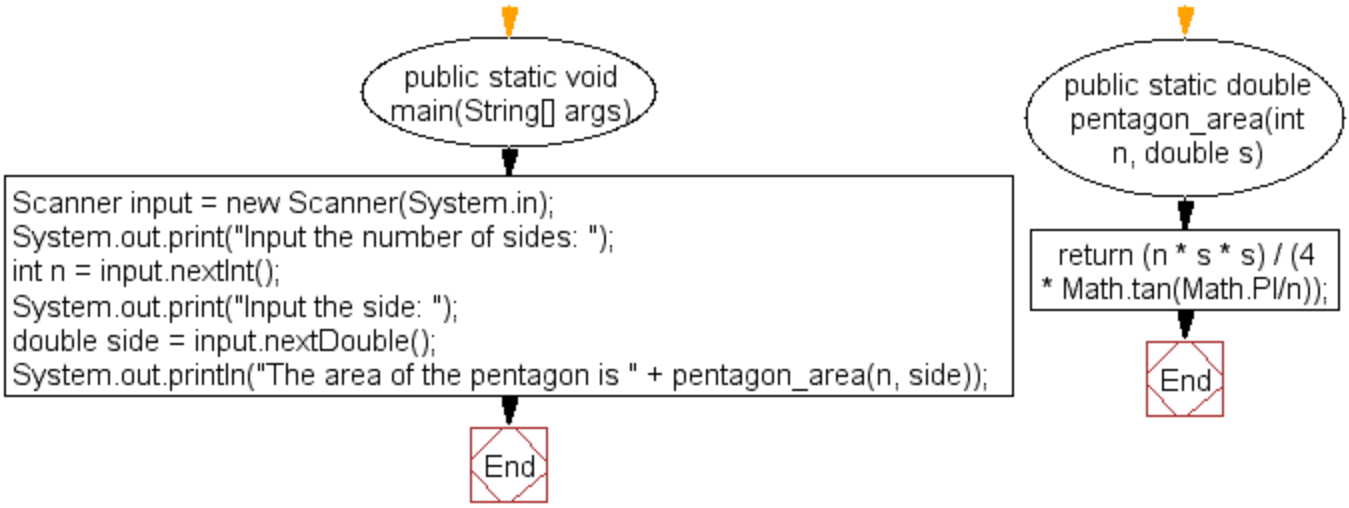
```
import java.  util.Scanner;
public class Exercise14 {

  public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Input the number of sides: ");
        int n = input.nextInt();
        System.out.print("Input the side: ");
        double side = input.nextDouble();
        System.out.println("The area of the pentagon is " + pentagon_area(n, side));
    }
    public static double pentagon_area(int n, double s) {
```

```
        return  (n * s * s) / (4 * Math.tan(Math.PI/n));
    }
}
```

Sample Output:

```
Input the number of sides: 5
Input the side: 6

The area of the pentagon is 61.9371864212028
```

Flowchart:



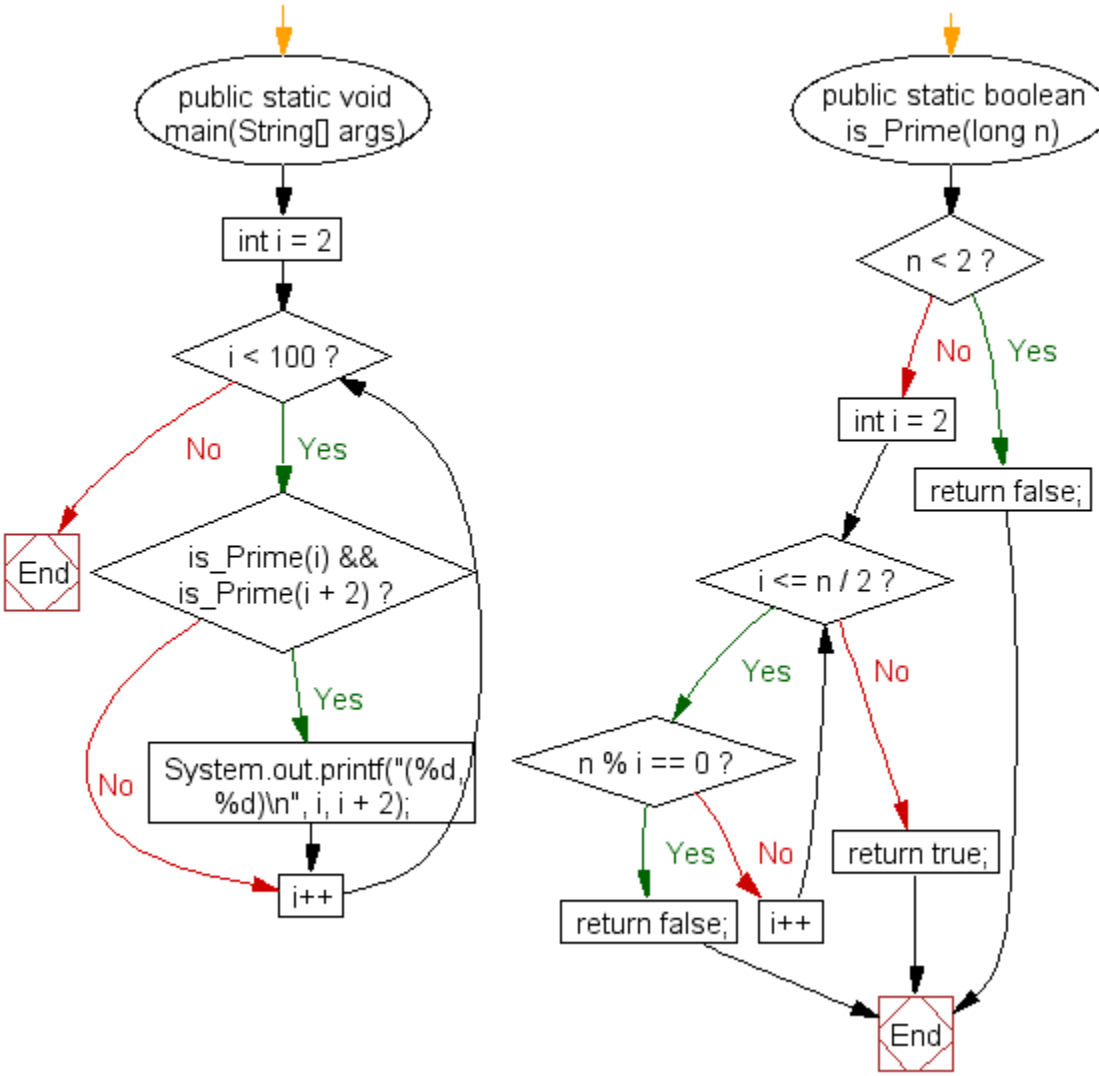
200. Write a Java method to find all twin prime numbers less than 100.

```
import java.  util.Scanner;
public class Exercise16 {

 public static void main(String[] args) {
        for (int i = 2; i < 100; i++) {
            if (is_Prime(i) && is_Prime(i + 2)) {
                System.out.printf("(%d, %d)\n", i, i + 2);
            }
        }
    }
    public static boolean is_Prime(long n) {
        if (n < 2) return false;
        for (int i = 2; i <= n / 2; i++) {
            if (n % i == 0) return false;
        }
        return true;
    }
}
```

Sample Output:

```
(3, 5)
(5, 7)
(11, 13)
(17, 19)
(29, 31)
(41, 43)
(59, 61)
(71, 73)
```

Flowchart :



Java Collection, ArrayList Exercises:

201. Write a Java program to create a new array list, add some elements (string) and print out the collection.

```
import java.  util.*;
public class Exercise1 {
  public static void main(String[] args) {
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  System.out.println(list_Strings);
 }
}
```

Sample Output:

```
Note: Exercise1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
[Red, Green, Orange, White, Black]
```

202. Write a Java program to iterate through all elements in a array list.

```
import java.  util.*;
  public class Exercise2 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  // Print the list
  for (String element : list_Strings) {
    System.out.println(element);
    }
 }
}
```

Sample Output:

```
Red
Green
Orange
White
Black
```

203. Write a Java program to insert an element into the array list at the first position.

```
import java.  util.*;
  public class Exercise3 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  // Print the list
  System.out.println(list_Strings);
  // Now insert a color at the first and last position of the list
  list_Strings.add(0, "Pink");
  list_Strings.add(5, "Yellow");
  // Print the list
  System.out.println(list_Strings);
 }
}
```

Sample Output:

```
Note: Exercise3.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
[Red, Green, Orange, White, Black]
[Pink, Red, Green, Orange, White, Yellow, Black]
```

204. Write a Java program to retrieve an element (at a specified index) from a given array list.

```
import java.  util.*;
  public class Exercise4 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  // Print the list
  System.out.println(list_Strings);
  // Retrive the first and third element
  String element = list_Strings.get(0);
  System.out.println("First element: "+element);
  element = list_Strings.get(2);
  System.out.println("Third element: "+element);
 }
}
```

Sample Output:

```
[Red, Green, Orange, White, Black]
First element: Red
Third element: Orange
```

205. Write a Java program to update specific array element by given element.

```
import java.  util.*;
  public class Exercise5 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  // Print the list
  System.out.println(list_Strings);
  // Update the third element with "Yellow"
  list_Strings.set(2, "Yellow");
   // Print the list again
  System.out.println(list_Strings);
 }
}
```

Sample Output:

```
[Red, Green, Orange, White, Black]
[Red, Green, Yellow, White, Black]
```

206. Write a Java program to remove the third element from a array list.

```
import java.  util.*;
  public class Exercise6 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  // Print the list
  System.out.println(list_Strings);
  // Remove the third element from the list.
  list_Strings.remove(2);
  // Print the list again
  System.out.println("After removing third element from the list:\n"+list_Strings);
 }
}
```

Sample Output:

```
[Red, Green, Orange, White, Black]
After removing third element from the list:
[Red, Green, White, Black]
```

207. Write a Java program to search an element in a array list.

```
import java.  util.*;
  public class Exercise7 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  // Search the value Red
    if (list_Strings.contains("Red")) {
    System.out.println("Found the element");
    } else {
    System.out.println("There is no such element");
    }
 }
}
```

Sample Output:

```
Found the element
```

208. Write a Java program to sort a given array list.

```
import java.  util.*;
  public class Exercise8 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  System.out.println("List before sort: "+list_Strings);
  Collections.sort(list_Strings);
  System.out.println("List after sort: "+list_Strings);
 }
}
```

Sample Output:

```
List before sort: [Red, Green, Orange, White, Black]
List after sort: [Black, Green, Orange, Red, White]
```

209. Write a Java program to copy one array list into another.

```
import java.  util.*;
  public class Exercise9 {
  public static void main(String[] args) {
  List<String> List1 = new ArrayList<String>();
  List1.add("1");
```

```
  List1.add("2");
  List1.add("3");
  List1.add("4");
  System.out.println("List1: " + List1);
  List<String> List2 = new ArrayList<String>();
  List2.add("A");
  List2.add("B");
  List2.add("C");
  List2.add("D");
  System.out.println("List2: " + List2);
  // Copy List1 to List  Collections.copy(List1, List2);
  System.out.println("Copy List to List2,\nAfter copy:");
  System.out.println("List1: " + List1);
  System.out.println("List2: " + List2);
 }
}
```

Sample Output:

```
List1: [1, 2, 3, 4]
List2: [A, B, C, D]
Copy List to List2,
After copy:
List1: [A, B, C, D]
List2: [A, B, C, D]
```

210. Write a Java program to shuffle elements in a array list.

```
import java.  util.*;
  public class Exercise10 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  System.out.println("List before shuffling:\n" + list_Strings);
  Collections.shuffle(list_Strings);
  System.out.println("List after shuffling:\n" + list_Strings);
 }
}
```

Sample Output:

```
List before shuffling:
[Red, Green, Orange, White, Black]
List after shuffling:
[White, Orange, Black, Red, Green]
```

211. Write a Java program to reverse elements in a array list.

```
import java.  util.*;
  public class Exercise11 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
```

```
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  System.out.println("List before reversing :\n" + list_Strings);
  Collections.reverse(list_Strings);
  System.out.println("List after reversing :\n" + list_Strings);
 }
}
```

Sample Output:

```
List before reversing :
[Red, Green, Orange, White, Black]
List after reversing :
[Black, White, Orange, Green, Red]
```

212. Write a Java program to extract a portion of a array list.

```
import java.  util.*;
  public class Exercise12 {
  public static void main(String[] args) {
  // Creae a list and add some colors to the list
  List<String> list_Strings = new ArrayList<String>();
  list_Strings.add("Red");
  list_Strings.add("Green");
  list_Strings.add("Orange");
  list_Strings.add("White");
  list_Strings.add("Black");
  System.out.println("Original list: " + list_Strings);
  List<String> sub_List = list_Strings.subList(0, 3);
  System.out.println("List of first three elements: " + sub_List);
 }
}
```

Sample Output:

```
Original list: [Red, Green, Orange, White, Black]
List of first three elements: [Red, Green, Orange]
```

213. Write a Java program to compare two array lists.

```
import java.  util.*;
  public class Exercise13 {
  public static void main(String[] args) {
   ArrayList<String> c1= new ArrayList<String>();
          c1.add("Red");
          c1.add("Green");
          c1.add("Black");
          c1.add("White");
          c1.add("Pink");
          ArrayList<String> c2= new ArrayList<String>();
          c2.add("Red");
          c2.add("Green");
          c2.add("Black");
          c2.add("Pink");
          //Storing the comparison output in ArrayList<String>
          ArrayList<String> c3 = new ArrayList<String>();
          for (String e : c1)
             c3.add(c2.contains(e) ? "Yes" : "No");
          System.out.println(c3);
```

```
    }
}
```

Sample Output:

```
[Yes, Yes, Yes, No, Yes]
```

214. Write a Java program of swap two elements in an array list.

```
import java.  util.ArrayList;
import java.  util.Collections;
  public class Exercise14 {
  public static void main(String[] args) {
   ArrayList<String> c1= new ArrayList<String>();
          c1.add("Red");
          c1.add("Green");
          c1.add("Black");
          c1.add("White");
          c1.add("Pink");
          System.out.println("Array list before Swap:");
          for(String a: c1){
          System.out.println(a);
        }
          //Swapping 1st(index 0) element with the 3rd(index 2) element
         Collections.swap(c1, 0, 2);
          System.out.println("Array list after swap:");
          for(String b: c1){
          System.out.println(b);
         }
    }
}
```

Sample Output:

```
Array list before Swap:
Red
Green
Black
White
Pink
Array list after swap:
Black
Green
Red
White
Pink
```

215. Write a Java program to join two array lists.

```
import java.  util.ArrayList;
import java.  util.Collections;
  public class Exercise15 {
  public static void main(String[] args) {
   ArrayList<String> c1= new ArrayList<String>();
          c1.add("Red");
          c1.add("Green");
          c1.add("Black");
          c1.add("White");
          c1.add("Pink");
          System.out.println("List of first array: " + c1);
```

```
          ArrayList<String> c2= new ArrayList<String>();
          c2.add("Red");
          c2.add("Green");
          c2.add("Black");
          c2.add("Pink");
          System.out.println("List of second array: " + c2);

      // Let join above two list
        ArrayList<String> a = new ArrayList<String>();
        a.  addAll(c1);
        a.  addAll(c2);
        System.out.println("New array: " + a);

    }
}
```

Sample Output:

```
List of first array: [Red, Green, Black, White, Pink]
List of second array: [Red, Green, Black, Pink]
New array: [Red, Green, Black, White, Pink, Red, Green, Black, Pink]
```

216. Write a Java program to append the specified element to the end of a linked list.

```
import java.  util.LinkedList;
  public class Exercise1 {
  public static void main(String[] args) {
     // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
          l_list.add("White");
          l_list.add("Pink");
          l_list.add("Yellow");

   // print the list
   System.out.println("The linked list: " + l_list);
   }
}
```

Sample Output:

```
The linked list: [Red, Green, Black, White, Pink, Yellow]
```

217. Write a Java program to iterate through all elements in a linked list.

```
import java.  util.LinkedList;
  public class Exercise2 {
  public static void main(String[] args) {
    // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
```

```
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
          l_list.add("White");
          l_list.add("Pink");
         // Print the linked list
  for (String element : l_list) {
    System.out.println(element);
    }
 }
}
```

Sample Output:

```
Red
Green
Black
White
Pink
```

218. Write a Java program to iterate through all elements in a linked list starting at the specified position.

```
import java.  util.LinkedList;
import java.  util.Iterator;
  public class Exercise3 {
  public static void main(String[] args) {
    // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
          l_list.add("White");
          l_list.add("Pink");
// set Iterator at specified index
   Iterator p = l_list.listIterator(1);
   // print list from second position
   while (p.hasNext()) {
   System.out.println(p.next());
   }
   }
}
```

Sample Output:

```
Green
Black
White
Pink
```

219. Write a Java program to iterate a linked list in reverse order.

```
import java.  util.LinkedList;
import java.  util.Iterator;
  public class Exercise4 {
  public static void main(String[] args) {
    // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
          l_list.add("Pink");
          l_list.add("orange");

        // print original list
   System.out.println("Original linked list:" + l_list);

    Iterator it = l_list.descendingIterator();
     // Print list elements in reverse order
     System.out.println("Elements in Reverse Order:");
     while (it.hasNext()) {
        System.out.println(it.next());
     }
  }
}
```

Sample Output:

```
Original linked list:[Red, Green, Black, Pink, orange]
Elements in Reverse Order:
orange
Pink
Black
Green
Red
```

220. Write a Java program to insert the specified element at the specified position in the linked list.

```
import java.  util.LinkedList;
public class Exercise5 {
   public static void main(String[] args) {
     // create an empty linked list
     LinkedList <String> l_list = new LinkedList <String> ();
     // use add() method to add values in the linked list
     l_list.add("Red");
     l_list.add("Green");
     l_list.add("Black");
     l_list.add("White");
     l_list.add("Pink");
     System.out.println("Original linked list: ");
     System.out.println("Let add the Yellow color after the Red Color: " + l_list);
     l_list.add(1, "Yellow");
     // print the list
     System.out.println("The linked list:" + l_list);
   }
}
```

Sample Output:

```
Original linked list:
Let add the Yellow color after the Red Color: [Red, Green, Black, White
, Pink]
The linked list:[Red, Yellow, Green, Black, White, Pink]
```

221. Write a Java program to insert elements into the linked list at the first and last position.

```
import java.  util.LinkedList;
  public class Exercise6 {
  public static void main(String[] args) {
   // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
     System.out.println("Original linked list:" + l_list);
   // Add an element at the beginning
    l_list.addFirst("White");

   // Add an element at the end of list
    l_list.addLast("Pink");
     System.out.println("Final linked list:" + l_list);
 }
}
```

Sample Output:

```
Original linked list:[Red, Green, Black]
Final linked list:[White, Red, Green, Black, Pink]
```

222. Write a Java program to insert the specified element at the front of a linked list.

```
import java.  util.LinkedList;
  public class Exercise7 {
  public static void main(String[] args) {
   // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
     System.out.println("Original linked list:" + l_list);
  // Add an element to front of LinkedList
     l_list.offerFirst("Pink");
     System.out.println("Final linked list:" + l_list);
 }
}
```

Sample Output:

```
Original linked list:[Red, Green, Black]
Final linked list:[Pink, Red, Green, Black]
```

223. Write a Java program to insert the specified element at the end of a linked list.

```
import java.  util.LinkedList;
  public class Exercise8 {
  public static void main(String[] args) {
   // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
     System.out.println("Original linked list:" + l_list);
  // Add an element at the end of a linked list
     l_list.offerLast("Pink");
     System.out.println("Final linked list:" + l_list);
 }
}
```

Sample Output:

```
Original linked list:[Red, Green, Black]
Final linked list:[Red, Green, Black, Pink]
```

224. Write a Java program to insert some elements at the specified position into a linked list.

```
import java.  util.LinkedList;
public class Exercise9 {
 public static void main(String[] args) {
  // create an empty linked list
  LinkedList <String> l_list = new LinkedList <String> ();
  // use add() method to add values in the linked list
  l_list.add("Red");
  l_list.add("Green");
  l_list.add("Black");
  // print original list
  System.out.println("Original linked list:" + l_list);
  // create a new collection and add some elements
  LinkedList <String> new_l_list = new LinkedList <String> ();
  new_l_list.add("White");
  new_l_list.add("Pink");
  // Add the collection in the second position of the existing linked list
  l_list.addAll(1, new_l_list);
  // print the new list
  System.out.println("LinkedList:" + l_list);
 }
}
```

Sample Output:

```
Original linked list:[Red, Green, Black]
LinkedList:[Red, White, Pink, Green, Black]
```

225. Write a Java program to remove a specified element from a linked list.

```
import java.  util.*;
public class Exercise12 {
 public static void main(String[] args) {
  // create an empty linked list
  LinkedList <String> l_list = new LinkedList <String> ();
  // use add() method to add values in the linked list
  l_list.add("Red");
  l_list.add("Green");
  l_list.add("Black");
  l_list.add("Pink");
  l_list.add("orange");
  // print the list
  System.out.println("The Original linked list: " + l_list);
  // Remove the element in third position from the said linked list
  l_list.remove(2);
  System.out.println("The New linked list: " + l_list);
 }
}
```

Sample Output:

```
The Original linked list: [Red, Green, Black, Pink, orange]
The New linked list: [Red, Green, Pink, orange]
```

226. Write a Java program to remove all the elements from a linked list.

```
import java.  util.*;
  public class Exercise14 {
  public static void main(String[] args) {
    // create an empty linked list
     LinkedList<String> l_list = new LinkedList<String>();
   // use add() method to add values in the linked list
          l_list.add("Red");
          l_list.add("Green");
          l_list.add("Black");
          l_list.add("Pink");
          l_list.add("orange");

     // print the list
   System.out.println("The Original linked list: " + l_list);
  // Removing all the elements from the linked list
    l_list.clear();

    System.out.println("The New linked list: " + l_list);
  }
}
```

Sample Output:

```
The Original linked list: [Red, Green, Black, Pink, orange]
The New linked list: []
```

227. Write a Java program of swap two elements in an linked list.

```
import java.  util.*;
public class Exercise15 {
 public static void main(String[] args) {
  // create an empty linked list
  LinkedList <String> l_list = new LinkedList <String> ();
  // use add() method to add values in the linked list
  l_list.add("Red");
  l_list.add("Green");
  l_list.add("Black");
  l_list.add("Pink");
  l_list.add("orange");
  // print the list
  System.out.println("The Original linked list: " + l_list);

   //Swapping 1st(index 0) element(Red) with the 3rd(index 2) element (Black)
    Collections.swap(l_list, 0, 2);
    System.out.println("The New linked list after swap: " + l_list);
 }
}
```

Sample Output:

```
The Original linked list: [Red, Green, Black, Pink, orange]
The New linked list after swap: [Black, Green, Red, Pink, orange]
```

228. Write a Java program to clone an linked list to another linked list.

```
import java.  util.*;
public class Exercise18 {
 public static void main(String[] args) {
  // create an empty linked list
  LinkedList <String> c1 = new LinkedList <String> ();
            c1.add("Red");
          c1.add("Green");
          c1.add("Black");
          c1.add("White");
          c1.add("Pink");
          System.out.println("Original linked  list: " + c1);
           LinkedList <String> newc1 = new LinkedList <String> ();
                newc1 = (LinkedList)c1.clone();
          System.out.println("Cloned linked list: " + newc1);
}
}
```

Sample Output:

```
Note: Exercise18.java uses unchecked or unsafe operations.
```

```
Note: Recompile with -Xlint:unchecked for details.
Original linked  list: [Red, Green, Black, White, Pink]
Cloned linked list: [Red, Green, Black, White, Pink]
```

229. Write a Java program to retrieve but does not remove, the first element of a linked list.

```
import java.  util.*;
public class Exercise20 {
 public static void main(String[] args) {
  // create an empty linked list
  LinkedList <String> c1 = new LinkedList <String> ();
            c1.add("Red");
          c1.add("Green");
          c1.add("Black");
          c1.add("White");
          c1.add("Pink");
          System.out.println("Original linked list: " + c1);
     // Retrieve but does not remove, the first element of a linked list
    String x = c1.peekFirst();
    System.out.println("First element in the list: " + x);
    System.out.println("Original linked list: " + c1);
 }
}
```

Sample Output:

```
Original linked list: [Red, Green, Black, White, Pink]
First element in the list: Red
Original linked list: [Red, Green, Black, White, Pink]
```

230. Write a Java program to convert a linked list to array list.

```
import java.  util.*;
public class Exercise23 {
 public static void main(String[] args) {
  // create an empty linked list
  LinkedList <String> linked_list = new LinkedList <String> ();
            linked_list.add("Red");
          linked_list.add("Green");
          linked_list.add("Black");
          linked_list.add("White");
          linked_list.add("Pink");
          System.out.println("Original linked list: " + linked_list);

      //Convert a linked list to array list
     List<String> list = new ArrayList<String>(linked_list);
    for (String str : list){
      System.out.println(str);
    }
 }
}
```

Sample Output:

```
Original linked list: [Red, Green, Black, White, Pink]
```

```
Red
Green
Black
White
Pink
```

231. Write a Java program to compare two linked lists.

```
import java.  util.*;
  public class Exercise24 {
  public static void main(String[] args) {
   LinkedList<String> c1= new LinkedList<String>();
          c1.add("Red");
          c1.add("Green");
          c1.add("Black");
          c1.add("White");
          c1.add("Pink");
          LinkedList<String> c2= new LinkedList<String>();
          c2.add("Red");
          c2.add("Green");
          c2.add("Black");
          c2.add("Orange");
          //comparison output in linked list
          LinkedList<String> c3 = new LinkedList<String>();
          for (String e : c1)
             c3.add(c2.contains(e) ? "Yes" : "No");
          System.out.println(c3);
     }
}
```

Sample Output:

```
[Yes, Yes, Yes, No, No]
```

HashSet

232. Write a Java program to append the specified element to the end of a hash set.

```
import java.  util.HashSet;
  public class Exercise1 {
  public static void main(String[] args) {
     // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          h_set.add("Pink");
          h_set.add("Yellow");
   // print the hash set
   System.out.println("The Hash Set: " + h_set);
   }
}
```

Sample Output:

```
The Hash Set: [Red, White, Pink, Yellow, Black, Green]
```

233. Write a Java program to iterate through all elements in a hash list.

```
import java.  util.*;
import java.  util.Iterator;
  public class Exercise2 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          h_set.add("Pink");
          h_set.add("Yellow");
  // set Iterator
    Iterator<String> p = h_set.iterator();
  // Iterate the hash set
   while (p.hasNext()) {
   System.out.println(p.next());
   }
   }
}
```

Sample Output:

```
Red
White
Pink
Yellow
Black
Green
```

234. Write a Java program to get the number of elements in a hash set.

```
import java.  util.*;
import java.  util.Iterator;
  public class Exercise3 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          h_set.add("Pink");
          h_set.add("Yellow");
    System.out.println("Original Hash Set: " + h_set);
    System.out.println("Size of the Hash Set: " + h_set.size());
   }
}
```

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
Size of the Hash Set: 6
```

235. Write a Java program to empty an hash set.

```
import java.  util.*;
  public class Exercise4 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          h_set.add("Pink");
          h_set.add("Yellow");
    System.out.println("Original Hash Set: " + h_set);
    // Remove all elements
    h_set.removeAll(h_set);
    System.out.println("Hash Set after removing all the elements "+h_set);
   }
}
```

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
Hash Set after removing all the elements []
```

236. Write a Java program to test a hash set is empty or not.

```
import java.  util.*;
  public class Exercise5 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet();
   // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          h_set.add("Pink");
          h_set.add("Yellow");
    System.out.println("Original Hash Set: " + h_set);
    System.out.println("Checking the above array list is empty or not! "+h_set.isEmpty());
    System.out.println("Remove all the elements from a Hash Set: ");
    h_set.removeAll(h_set);
    System.out.println("Hash Set after removing all the elements "+h_set);
   }
}
```

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
Checking the above array list is empty or not! false
Remove all the elements from a Hash Set:
Hash Set after removing all the elements []
```

237. Write a Java program to clone a hash set to another hash set.

```
import java.  util.*;
  public class Exercise6 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
         h_set.add("Red");
         h_set.add("Green");
         h_set.add("Black");
         h_set.add("White");
         h_set.add("Pink");
         h_set.add("Yellow");
    System.out.println("Original Hash Set: " + h_set);
    HashSet <String> new_h_set = new HashSet <String> ();
         new_h_set = (HashSet)h_set.clone();
         System.out.println("Cloned Hash Set: " + new_h_set);
   }
}
```

Sample Output:

```
Note: Exercise6.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
Cloned Hash Set: [Red, White, Pink, Yellow, Black, Green]
```

238. Write a Java program to convert a hash set to an array.

```
import java.  util.*;
  public class Exercise7 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
         h_set.add("Red");
         h_set.add("Green");
         h_set.add("Black");
         h_set.add("White");
         h_set.add("Pink");
         h_set.add("Yellow");
      System.out.println("Original Hash Set: " + h_set);
      // Creating an Array
      String[] new_array = new String[h_set.size()];
      h_set.toArray(new_array);

     // Displaying Array elements
     System.out.println("Array elements: ");
      for(String element : new_array){
        System.out.println(element);
     }
   }
}
```

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
Array elements:
Red
White
Pink
Yellow
Black
Green
```

239. Write a Java program to convert a hash set to a tree set.

```
import java.  util.*;
  public class Exercise8 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
   // use add() method to add values in the hash set
         h_set.add("Red");
         h_set.add("Green");
         h_set.add("Black");
         h_set.add("White");
         h_set.add("Pink");
         h_set.add("Yellow");
      System.out.println("Original Hash Set: " + h_set);

     // Creat a TreeSet of HashSet elements
     Set<String> tree_set = new TreeSet<String>(h_set);

     // Display TreeSet elements
     System.out.println("TreeSet elements: ");
     for(String element : tree_set){
        System.out.println(element);
     }
  }
}
```

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
TreeSet elements:
Black
Green
Pink
Red
White
Yellow
```

240. Write a Java program to convert a hash set to a List/ArrayList.

```
import java.  util.*;
  public class Exercise9 {
  public static void main(String[] args) {
         // Create a empty hash set
     HashSet<String> h_set = new HashSet<String>();
```

```
   // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          h_set.add("Pink");
          h_set.add("Yellow");
       System.out.println("Original Hash Set: " + h_set);

     // Create a List from HashSet elements
      List<String> list = new ArrayList<String>(h_set);

      // Display ArrayList elements
      System.out.println("ArrayList contains: "+ list);
   }
}
```

Sample Output:

```
Original Hash Set: [Red, White, Pink, Yellow, Black, Green]
ArrayList contains: [Red, White, Pink, Yellow, Black, Green]
```

241. Write a Java program to compare two hash sets.

```
import java.  util.*;
  public class Exercise10 {
  public static void main(String[] args) {
     // Create a empty hash set
        HashSet<String> h_set = new HashSet<String>();
     // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
          HashSet<String>h_set2 = new HashSet<String>();
          h_set2.add("Red");
          h_set2.add("Pink");
          h_set2.add("Black");
          h_set2.add("Orange");
          //comparison output in hash set
          HashSet<String>result_set = new HashSet<String>();
         for (String element : h_set){
             System.out.println(h_set2.contains(element) ? "Yes" : "No");
          }
     }
}
```

Sample Output:

```
Yes
No
Yes
No
```

TreeSet

242. Write a Java program to compare two sets and retain elements which are same on both sets.

```
import java.  util.*;
  public class Exercise11 {
  public static void main(String[] args) {
     // Create a empty hash set
        HashSet<String> h_set1 = new HashSet<String>();
     // use add() method to add values in the hash set
          h_set1.add("Red");
          h_set1.add("Green");
          h_set1.add("Black");
          h_set1.add("White");
          System.out.println("Frist HashSet content: "+h_set1);
          HashSet<String>h_set2 = new HashSet<String>();
          h_set2.add("Red");
          h_set2.add("Pink");
          h_set2.add("Black");
          h_set2.add("Orange");
          System.out.println("Second HashSet content: "+h_set2);
          h_set1.retainAll(h_set2);
          System.out.println("HashSet content:");
          System.out.println(h_set1);
     }
}
```

Sample Output:

```
Frist HashSet content: [Red, White, Black, Green]
Second HashSet content: [Red, Pink, Black, Orange]
HashSet content:
[Red, Black]
```

243. Write a Java program to remove all of the elements from a hash set.

```
import java.  util.*;
  public class Exercise12 {
  public static void main(String[] args) {
     // Create a empty hash set
        HashSet<String> h_set = new HashSet<String>();
     // use add() method to add values in the hash set
          h_set.add("Red");
          h_set.add("Green");
          h_set.add("Black");
          h_set.add("White");
           System.out.println("Original hash set contains: "+ h_set);
      //  clear() method removes all the elements from a hash set
      // and the set becomes empty.
          h_set.clear();

     // Display original hash set content again
          System.out.println("HashSet content: "+h_set);
```

```
    }
}
```

Sample Output:

```
Original hash set contains: [Red, White, Black, Green]
HashSet content: []
```

244. Write a Java program to create a new tree set, add some colors (string) and print out the tree set.

```
import java.  util.TreeSet;
public class Exercise1 {
  public static void main(String[] args) {
  TreeSet<String> tree_set = new TreeSet<String>();
  tree_set.add("Red");
  tree_set.add("Green");
  tree_set.add("Orange");
  tree_set.add("White");
  tree_set.add("Black");
  System.out.println("Tree set: ");
  System.out.println(tree_set);
 }
}
```

Sample Output:

```
Tree set:
[Black, Green, Orange, Red, White]
```

245. Write a Java program to iterate through all elements in a tree set.

```
import java.  util.TreeSet;
public class Exercise2 {
  public static void main(String[] args) {
  TreeSet<String> tree_set = new TreeSet<String>();
  tree_set.add("Red");
  tree_set.add("Green");
  tree_set.add("Orange");
  tree_set.add("White");
  tree_set.add("Black");
  // Print the tree list
  for (String element : tree_set) {
    System.out.println(element);
    }
 }
}
```

Sample Output:

```
Black
Green
Orange
Red
White
```

246. Write a Java program to add all the elements of a specified tree set to another tree set.

```
import java.  util.TreeSet;
public class Exercise3 {
  public static void main(String[] args) {
  TreeSet<String> tree_set1 = new TreeSet<String>();
  tree_set1.add("Red");
  tree_set1.add("Green");
  tree_set1.add("Orange");
  System.out.println("Tree set1: "+tree_set1);
  TreeSet<String> tree_set2 = new TreeSet<String>();
  tree_set2.add("Pink");
  tree_set2.add("White");
  tree_set2.add("Black");
  System.out.println("Tree set2: "+tree_set2);
   // adding treetwo to treeone
   tree_set1.addAll(tree_set2);
   System.out.println("Tree set1: "+tree_set1);
 }
}
```

Sample Output:

```
Tree set1: [Green, Orange, Red]
Tree set2: [Black, Pink, White]
```

247. Write a Java program to create a reverse order view of the elements contained in a given tree set.

```
import java.  util.TreeSet;
import java.  util.Iterator;
  public class Exercise4 {
  public static void main(String[] args) {
    // create an empty tree set
     TreeSet<String> t_set = new TreeSet<String>();
   // use add() method to add values in the tree set
          t_set.add("Red");
          t_set.add("Green");
          t_set.add("Black");
          t_set.add("Pink");
          t_set.add("orange");
    // print original list
   System.out.println("Original tree set:" + t_set);
     Iterator it = t_set.descendingIterator();
     // Print list elements in reverse order
     System.out.println("Elements in Reverse Order:");
     while (it.hasNext()) {
        System.out.println(it.next());
     }
  }
}
```

Sample Output:

```
Original tree set:[Black, Green, Pink, Red, orange]
Elements in Reverse Order:
orange
```

```
Red
Pink
Green
Black
```

248. Write a Java program to get the first and last elements in a tree set.

```
import java.  util.TreeSet;
  public class Exercise5 {
  public static void main(String[] args) {
  TreeSet<String> tree_set = new TreeSet<String>();
  tree_set.add("Red");
  tree_set.add("Green");
  tree_set.add("Orange");
  tree_set.add("White");
  tree_set.add("Black");
  System.out.println("Tree set: ");
  System.out.println(tree_set);
   // Find first element of the tree set
    Object first_element = tree_set.first();
    System.out.println("First Element is: "+first_element);

    // Find last element of the tree set
    Object last_element = tree_set.last();
    System.out.println("Last Element is: "+last_element);
 }
}
```

Sample Output:

```
Tree set:
[Black, Green, Orange, Red, White]
First Element is: Black
Last Element is: White
```

249. Write a Java program to clone a tree set list to another tree set.

```
import java.  util.TreeSet;
import java.  util.Iterator;
  public class Exercise6 {
  public static void main(String[] args) {
    // create an empty tree set
     TreeSet<String> t_set = new TreeSet<String>();
   // use add() method to add values in the tree set
          t_set.add("Red");
          t_set.add("Green");
          t_set.add("Black");
          t_set.add("Pink");
          t_set.add("orange");

   System.out.println("Original tree set:" + t_set);
    TreeSet<String> new_t_set = (TreeSet<String>)t_set.clone();
          System.out.println("Cloned tree list: " + t_set);
     }
```

```
}
```

Sample Output:

```
Note: Exercise6.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Original tree set:[Black, Green, Pink, Red, orange]
Cloned tree list: [Black, Green, Pink, Red, orange]
```

250. Write a Java program to get the number of elements in a tree set.

```
import java.  util.TreeSet;
import java.  util.Iterator;
  public class Exercise7 {
  public static void main(String[] args) {
    // create an empty tree set
     TreeSet<String> t_set = new TreeSet<String>();
   // use add() method to add values in the tree set
          t_set.add("Red");
          t_set.add("Green");
          t_set.add("Black");
          t_set.add("Pink");
          t_set.add("orange");
     System.out.println("Original tree set: " + t_set);
    System.out.println("Size of the tree set: " + t_set.size());
   }
}
```

Sample Output:

```
Original tree set: [Black, Green, Pink, Red, orange]
Size of the tree set: 5
```

251. Write a Java program to retrieve and remove the first element of a tree set.

```
import java.  util.TreeSet;
import java.  util.Iterator;
  public class Exercise14 {
  public static void main(String[] args) {
 // creating TreeSet
   TreeSet <Integer>tree_num = new TreeSet<Integer>();
   TreeSet <Integer>treeheadset = new TreeSet<Integer>();

   // Add numbers in the tree
   tree_num.add(10);
   tree_num.add(22);
   tree_num.add(36);
   tree_num.add(25);
   tree_num.add(16);
   tree_num.add(70);
   tree_num.add(82);
   tree_num.add(89);
   tree_num.add(14);
```

```
    System.out.println("Original tree set: "+tree_num);
    System.out.println("Removes the first(lowest) element: "+tree_num.pollFirst());
    System.out.println("Tree set after removing first element: "+tree_num);
    }
}
```

Sample Output:

```
Original tree set: [10, 14, 16, 22, 25, 36, 70, 82, 89]
Removes the first(lowest) element: 10
Tree set after removing first element: [14, 16, 22, 25, 36, 70, 82, 89]
```

252. Write a Java program to retrieve and remove the last element of a tree set.

```
import java.  util.TreeSet;
import java.  util.Iterator;
  public class Exercise15 {
  public static void main(String[] args) {
 // creating TreeSet
    TreeSet <Integer>tree_num = new TreeSet<Integer>();
    TreeSet <Integer>treeheadset = new TreeSet<Integer>();

    // Add numbers in the tree
    tree_num.add(10);
    tree_num.add(22);
    tree_num.add(36);
    tree_num.add(25);
    tree_num.add(16);
    tree_num.add(70);
    tree_num.add(82);
    tree_num.add(89);
    tree_num.add(14);
    System.out.println("Original tree set: "+tree_num);
    System.out.println("Removes the last element: "+tree_num.pollLast());
    System.out.println("Tree set after removing last element: "+tree_num);
    }
}
```

Sample Output:

```
Original tree set: [10, 14, 16, 22, 25, 36, 70, 82, 89]
Removes the last element: 89
Tree set after removing last element: [10, 14, 16, 22, 25, 36, 70, 82]
```

253. Write a Java program to remove a given element from a tree set.

```
import java.  util.TreeSet;
import java.  util.Iterator;
  public class Exercise16 {
  public static void main(String[] args) {
 // creating TreeSet
    TreeSet <Integer>tree_num = new TreeSet<Integer>();
    TreeSet <Integer>treeheadset = new TreeSet<Integer>();
```

```
    // Add numbers in the tree
    tree_num.add(10);
    tree_num.add(22);
    tree_num.add(36);
    tree_num.add(25);
    tree_num.add(16);
    tree_num.add(70);
    tree_num.add(82);
    tree_num.add(89);
    tree_num.add(14);
    System.out.println("Original tree set: "+tree_num);
    System.out.println("Removes 70 from the list: "+tree_num.remove(70));
    System.out.println("Tree set after removing last element: "+tree_num);
    }
}
```

Sample Output:

```
Original tree set: [10, 14, 16, 22, 25, 36, 70, 82, 89]
Removes 70 from the list: true
Tree set after removing last element: [10, 14, 16, 22, 25, 36, 82, 89]
```

254. Write a Java program to create a new priority queue, add some colors (string) and print out the elements of the priority queue.

```
import java.  util.PriorityQueue;
public class Exercise1 {
  public static void main(String[] args) {
  PriorityQueue<String> queue=new PriorityQueue<String>();
  queue.  add("Red");
  queue.  add("Green");
  queue.  add("Orange");
  queue.  add("White");
  queue.  add("Black");
  System.out.println("Elements of the Priority Queue: ");
  System.out.println(queue);
 }
}
```

Sample Output:

```
Elements of the Priority Queue:
[Black, Green, Orange, White, Red]
```

255. Write a Java program to iterate through all elements in priority queue.

```
import java.  util.PriorityQueue;
public class Exercise2 {
  public static void main(String[] args) {
    PriorityQueue<String> pq = new PriorityQueue<String>();
  pq.add("Red");
  pq.add("Green");
```

```
  pq.add("Orange");
  pq.add("White");
  pq.add("Black");
  System.out.println("Elements of the Priority Queue: ");
  // iterate the Priority Queue
  for (String element : pq) {
    System.out.println(element);
    }
 }
}
```

Sample Output:

```
Elements of the Priority Queue:
Black
Green
Orange
White
Red
```

Priority Queue

256. Write a Java program to add all the elements of a priority queue to another priority queue.

```
import java.  util.PriorityQueue;
public class Exercise3 {
public static void main(String[] args) {
  PriorityQueue<String> queue1 = new PriorityQueue<String>();
  queue1.add("Red");
  queue1.add("Green");
  queue1.add("Orange");
  System.out.println("Priority Queue1: "+queue1);
  PriorityQueue<String> queue2 = new PriorityQueue<String>();
  queue2.add("Pink");
  queue2.add("White");
  queue2.add("Black");
  System.out.println("Priority Queue2: "+queue2);
    // adding queue2 to queue   queue1.addAll(queue2);
   System.out.println("New Priority Queue1: "+queue1);
 }
}
```

Sample Output:

```
Priority Queue1: [Green, Red, Orange]
Priority Queue2: [Black, White, Pink]
New Priority Queue1: [Black, Green, Orange, Red, White, Pink]
```

257. Write a Java program to remove all the elements from a priority queue.

```
import java.  util.*;
  public class Example5 {
  public static void main(String[] args) {
   // Create Priority Queue
           PriorityQueue<String> pq1 = new PriorityQueue<String>();
```

```
   // use add() method to add values in the Priority Queue
          pq1.add("Red");
          pq1.add("Green");
          pq1.add("Black");
          pq1.add("White");
    System.out.println("Original Priority Queue: "+pq1);

   // Removing all the elements from the Priority Queue
    pq1.clear();

    System.out.println("The New Priority Queue: " + pq1);
  }
}
```

Sample Output:

```
Original Priority Queue: [Black, Red, Green, White]
The New Priority Queue: []
```

258. Write a Java program to compare two priority queues.

```
import java.  util.PriorityQueue;
  public class Exercise7 {
  public static void main(String[] args) {
  // Create a empty Priority Queue
        PriorityQueue<String> pq1 = new PriorityQueue<String>();
     // use add() method to add values in the Priority Queue
          pq1.add("Red");
          pq1.add("Green");
          pq1.add("Black");
          pq1.add("White");
          System.out.println("First Priority Queue: "+pq1);
          PriorityQueue<String> pq2 = new PriorityQueue<String>();
          pq2.add("Red");
          pq2.add("Pink");
          pq2.add("Black");
          pq2.add("Orange");
          System.out.println("Second Priority Queue: "+pq2);
          //comparison output in Priority Queue
         for (String element : pq1){
             System.out.println(pq2.contains(element) ? "Yes" : "No");
          }
     }
}
```

Sample Output:

```
First Priority Queue: [Black, Red, Green, White]
Second Priority Queue: [Black, Orange, Pink, Red]
Yes
Yes
No
No
```

259. Write a Java program to retrieve the first element of the priority queue.

```
import java.  util.PriorityQueue;
  public class Example8 {
  public static void main(String[] args) {
   // Create Priority Queue
      PriorityQueue<Integer> pq1 = new PriorityQueue<Integer>();
      PriorityQueue<Integer> pq2 = new PriorityQueue<Integer>();

    // Add numbers in the Queue
    pq1.add(10);
    pq1.add(22);
    pq1.add(36);
    pq1.add(25);
    pq1.add(16);
    pq1.add(70);
    pq1.add(82);
    pq1.add(89);
    pq1.add(14);
    System.out.println("Original Priority Queue: "+pq1);
    System.out.println("The first element of the Queue: "+pq1.peek());
    }
}
```

Sample Output:

```
Original Priority Queue: [10, 14, 36, 16, 22, 70, 82, 89, 25]
The first element of the Queue: 1
```

260. Write a Java program to retrieve and remove the first element.

```
import java.  util.PriorityQueue;
  public class Exercise9 {
  public static void main(String[] args) {
   // Create Priority Queue
      PriorityQueue<Integer> pq1 = new PriorityQueue<Integer>();
      PriorityQueue<Integer> pq2 = new PriorityQueue<Integer>();
    // Add numbers in the Priority Queue
    pq1.add(10);
    pq1.add(22);
    pq1.add(36);
    pq1.add(25);
    pq1.add(16);
    pq1.add(70);
    pq1.add(82);
    pq1.add(89);
    pq1.add(14);
    System.out.println("Original Priority Queue: "+pq1);
    System.out.println("Removes the first element: "+pq1.poll());
    System.out.println("Priority Queue after removing first element: "+pq1);
    }
}
```

Sample Output:

```
Original Priority Queue: [10, 14, 36, 16, 22, 70, 82, 89, 25]
Removes the first element: 10
Priority Queue after removing first element: [14, 16, 36, 25, 22, 70, 2, 89]
```

Map

261. Write a Java program to associate the specified value with the specified key in a HashMap.

```
import java.  util.*;
public class Example1 {
  public static void main(String args[]) {
  HashMap<Integer,String> hash_map= new HashMap<Integer,String>();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
  for(Map.Entry x:hash_map.entrySet()){
   System.out.println(x.getKey()+" "+x.getValue());
  }
 }
}
```

Sample Output:

```
1 Red
2 Green
3 Black
4 White
5 Blue
```

262. Write a Java program to count the number of key-value (size) mappings in a map.

```
import java.  util.*;
public class Example2 {
   public static void main(String args[]){
  HashMap<Integer,String> hash_map= new HashMap<Integer,String>();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
  System.out.println("Size of the hash map: "+hash_map.size());
 }
}
```

Sample Output:

```
Size of the hash map: 5
```

263. Write a Java program to copy all of the mappings from the specified map to another map.

```
import java.  util.*;
public class Example3 {
```

```
   public static void main(String args[]) {
  // create two hash maps
  HashMap <Integer,String> hash_map1 = new HashMap <Integer,String> ();
  HashMap <Integer,String> hash_map2 = new HashMap <Integer,String> ();
  // populate hash maps
  hash_map1.put(1, "Red");
  hash_map1.put(2, "Green");
  hash_map1.put(3, "Black");
  System.out.println("\nValues in first map: " + hash_map1);
  hash_map2.put(4, "White");
  hash_map2.put(5, "Blue");
  hash_map2.put(6, "Orange");
  System.out.println("\nValues in second map: " + hash_map2);
  // put all values in secondmap
  hash_map2.putAll(hash_map1);
  System.out.println("\nNow values in second map: " + hash_map2);
 }
}
```

Sample Output:

```
Values in first map: {1=Red, 2=Green, 3=Black}

Values in second map: {4=White, 5=Blue, 6=Orange}

Now values in second map: {1=Red, 2=Green, 3=Black, 4=White, 5=Blue, 6=
```

264. Write a Java program to remove all the mappings from a map.

```
import java.  util.*;
public class Example4 {
   public static void main(String args[]) {
  HashMap <Integer,String> hash_map = new HashMap <Integer,String> ();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
  // print the map
  System.out.println("The Original linked map: " + hash_map);
  // Removing all the elements from the linked map
  hash_map.clear();
  System.out.println("The New map: " + hash_map);
 }
}
```

Sample Output:

```
The Original linked map: {1=Red, 2=Green, 3=Black, 4=White, 5=Blue}
The New map: {}
```

265. Write a Java program to check whether a map contains key-value mappings (empty) or not.

```
import java.  util.*;
public class Example5 {
```

```
   public static void main(String args[]) {
  HashMap <Integer,String> hash_map = new HashMap <Integer,String> ();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
  // check if map is empty
  boolean result = hash_map.isEmpty();
  // check the result
  System.out.println("Is hash map empty: " + result);
  // Removing all the elements from the linked map
  hash_map.clear();
  // check if map is empty
  result = hash_map.isEmpty();
  // check the result
  System.out.println("Is hash map empty: " + result);
 }
}
```

Sample Output:

```
Is hash map empty: false
Is hash map empty: true
```

266. Write a Java program to get a shallow copy of a HashMap instance.

```
import java.  util.*;
public class Example6 {
   public static void main(String args[]){
  HashMap<Integer,String> hash_map= new HashMap<Integer,String>();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
    // print the map
   System.out.println("The Original map: " + hash_map);
   HashMap<Integer,String> new_hash_map= new HashMap<Integer,String>();
   new_hash_map = (HashMap)hash_map.clone();
   System.out.println("Cloned map: " + new_hash_map);
     }
}
```

Sample Output:

```
Note: Example6.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
The Original map: {1=Red, 2=Green, 3=Black, 4=White, 5=Blue}
Cloned map: {1=Red, 2=Green, 3=Black, 4=White, 5=Blue}
```

267. Write a Java program to test if a map contains a mapping for the specified key.

```
import java.  util.*;
public class Example7 {
```

```
   public static void main(String args[]) {
  HashMap < String, Integer > hash_map = new HashMap < String, Integer > ();
  hash_map.put("Red", 1);
  hash_map.put("Green", 2);
  hash_map.put("Black", 3);
  hash_map.put("White", 4);
  hash_map.put("Blue", 5);
  // print the map
  System.out.println("The Original map: " + hash_map);
  System.out.println("1. Is key 'Green' exists?");
  if (hash_map.containsKey("Green")) {
   //key exists
   System.out.println("yes! - " + hash_map.get("Green"));
  } else {
   //key does not exists
   System.out.println("no!");
  }
  System.out.println("\n2. Is key 'orange' exists?");
  if (hash_map.containsKey("orange")) {
   System.out.println("yes! - " + hash_map.get("orange"));
  } else {
   System.out.println("no!");
  }
 }
}
```

Sample Output:

```
The Original map: {Red=1, White=4, Blue=5, Black=3, Green=2}
1. Is key 'Green' exists?
yes! - 2

2. Is key 'orange' exists?
no!
```

268. Write a Java program to test if a map contains a mapping for the specified value.

```
import java.  util.*;
public class Example8 {
   public static void main(String args[]) {
  HashMap < Integer, String > hash_map = new HashMap < Integer, String > ();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
  // print the map
  System.out.println("The Original map: " + hash_map);
  System.out.println("1. Is value \'Green\' exists?");
  if (hash_map.containsValue("Green")) {
   //value exists
   System.out.println("Yes! ");
  } else {
   //value does not exists
   System.out.println("no!");
```

```
  }
  System.out.println("\n2. Is value \'orange\' exists?");
  if (hash_map.containsValue("orange")) {
   System.out.println("yes! - " );
  } else {
   System.out.println("No!");
  }
 }
}
```

Sample Output:

```
The Original map: {1=Red, 2=Green, 3=Black, 4=White, 5=Blue}
1. Is value 'Green' exists?
Yes!

2. Is value 'orange' exists?
No!
```

269. Write a Java program to create a set view of the mappings contained in a map.

```
import java.  util.*;
public class Example9 {
    public static void main(String args[]) {
  HashMap < Integer, String > hash_map = new HashMap < Integer, String > ();
  hash_map.put(1, "Red");
  hash_map.put(2, "Green");
  hash_map.put(3, "Black");
  hash_map.put(4, "White");
  hash_map.put(5, "Blue");
  // create set view for the map
  Set set = hash_map.entrySet();
  // check set values
  System.out.println("Set values: " + set);
 }
}
```

Sample Output:

```
Set values: [1=Red, 2=Green, 3=Black, 4=White, 5=Blue]
```

270. Write a Java program to get the value of a specified key in a map.

```
import java.  util.*;
public class Example10 {
    public static void main(String args[]){
   HashMap<Integer,String> hash_map= new HashMap<Integer,String>();
   hash_map.put(1,"Red");
  hash_map.put(2,"Green");
  hash_map.put(3,"Black");
  hash_map.put(4,"White");
  hash_map.put(5,"Blue");
     // get value of key     String val=(String)hash_map.get(3);
   // check the value
   System.out.println("Value for key 3 is: " + val);
```

```
 }
}
```

Sample Output:

```
Value for key 3 is: Black
```

271. Write a Java program to get a set view of the keys contained in this map.

```
import java.  util.*;
public class Example11 {
     public static void main(String args[]){

  HashMap<Integer,String> hash_map= new HashMap<Integer,String>();

  hash_map.put(1,"Red");
  hash_map.put(2,"Green");
  hash_map.put(3,"Black");
  hash_map.put(4,"White");
  hash_map.put(5,"Blue");

  // get keyset value from map
   Set keyset = hash_map.keySet();

  // check key set values
   System.out.println("Key set values are: " + keyset);
 }
}
```

Sample Output:

```
Key set values are: [1, 2, 3, 4, 5]
```

272. Write a Java program to get a collection view of the values contained in this map.

```
import java.  util.*;
public class Example12 {
     public static void main(String args[]){

  HashMap<Integer,String> hash_map= new HashMap<Integer,String>();
    hash_map.put(1,"Red");
  hash_map.put(2,"Green");
  hash_map.put(3,"Black");
  hash_map.put(4,"White");
  hash_map.put(5,"Blue");

   // checking collection view of the map
   System.out.println("Collection view is: "+ hash_map.values());
 }
}
```

Sample Output:

```
Collection view is: [Red, Green, Black, White, Blue]
```

273. Write a Java program to get the first (lowest) key and the last (highest) key currently in a map.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example9 {
     public static void main(String args[]) {
  // Create a tree map
  TreeMap <String,String> tree_map1 = new TreeMap <String,String> ();
  // Put elements to the map
  tree_map1.put("C2", "Red");
  tree_map1.put("C1", "Green");
  tree_map1.put("C4", "Black");
  tree_map1.put("C3", "White");
  System.out.println("Orginal TreeMap content: " + tree_map1);
  System.out.println("Greatest key: " + tree_map1.firstKey());
  System.out.println("Least key: " + tree_map1.lastKey());
 }
}
```

Sample Output:

```
Orginal TreeMap content: {C1=Green, C2=Red, C3=White, C4=Black}
Greatest key: C1
Least key: C
```

274. Write a Java program to get a reverse order view of the keys contained in a given map.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example10 {
     public static void main(String args[]) {
  // Create a tree map
  TreeMap <String,String> tree_map1 = new TreeMap <String,String> ();
  // Put elements to the map
  tree_map1.put("C2", "Red");
  tree_map1.put("C1", "Green");
  tree_map1.put("C4", "Black");
  tree_map1.put("C3", "White");
  System.out.println("Orginal TreeMap content: " + tree_map1);
  System.out.println("Reverse order view of the keys: " + tree_map1.descendingKeySet());
 }
}
```

Sample Output:

```
Orginal TreeMap content: {C1=Green, C2=Red, C3=White, C4=Black}
Reverse order view of the keys: [C4, C3, C2, C1]
```

275. Write a Java program to get the portion of a map whose keys are strictly less than a given key.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example13 {
       public static void main(String args[]) {
  // Create a tree map
  TreeMap < Integer, String > tree_map1 = new TreeMap < Integer, String > ();
```

```
  // Put elements to the map
  tree_map1.put(10, "Red");
  tree_map1.put(20, "Green");
  tree_map1.put(40, "Black");
  tree_map1.put(50, "White");
  tree_map1.put(60, "Pink");
  System.out.println("Orginal TreeMap content: " + tree_map1);
  System.out.println("Checking the entry for 10: ");
  System.out.println("Key(s): " + tree_map1.headMap(10));
  System.out.println("Checking the entry for 30: ");
  System.out.println("Key(s): " + tree_map1.headMap(30));
  System.out.println("Checking the entry for 70: ");
  System.out.println("Key(s): " + tree_map1.headMap(70));
 }
}
```

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 40=Black, 50=White, 60=Pink
}
Checking the entry for 10:
Key(s): {}
Checking the entry for 30:
Key(s): {10=Red, 20=Green}
Checking the entry for 70:
Key(s): {10=Red, 20=Green, 40=Black, 50=White, 60=Pink}
```

276. Write a Java program to get the portion of this map whose keys are less than (or equal to, if inclusive is true) a given key.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example14 {
       public static void main(String args[]) {
  // Create a tree map
  TreeMap < Integer, String > tree_map1 = new TreeMap < Integer, String > ();
  // Put elements to the map
  tree_map1.put(10, "Red");
  tree_map1.put(20, "Green");
  tree_map1.put(40, "Black");
  tree_map1.put(50, "White");
  tree_map1.put(60, "Pink");
  System.out.println("Orginal TreeMap content: " + tree_map1);
  System.out.println("Checking the entry for 10: ");
  System.out.println("Key(s): " + tree_map1.headMap(10, true));
  System.out.println("Checking the entry for 20: ");
  System.out.println("Key(s): " + tree_map1.headMap(20, true));
  System.out.println("Checking the entry for 70: ");
  System.out.println("Key(s): " + tree_map1.headMap(70, true));
 }
}
```

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 40=Black, 50=White, 60=Pink
}
Checking the entry for 10:
Key(s): {10=Red}
Checking the entry for 20:
```

```
Key(s): {10=Red, 20=Green}
Checking the entry for 70:
Key(s): {10=Red, 20=Green, 40=Black, 50=White, 60=Pink}
```

277. Write a Java program to get NavigableSet view of the keys contained in a map.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example18 {
         public static void main(String args[]) {
  // Create a tree map
  TreeMap < Integer, String > tree_map1 = new TreeMap < Integer, String > ();
  // Put elements to the map
  tree_map1.put(10, "Red");
  tree_map1.put(20, "Green");
  tree_map1.put(40, "Black");
  tree_map1.put(50, "White");
  tree_map1.put(60, "Pink");
  System.out.println("Orginal TreeMap content: " + tree_map1);
  System.out.println("Orginal TreeMap content: " + tree_map1.navigableKeySet());
 }
}
```

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 40=Black, 50=White, 60=Pink
}
Orginal TreeMap content: [10, 20, 40, 50, 60]
```

278. Write a Java program to remove and get a key-value mapping associated with the least key in a map.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example19 {
         public static void main(String args[]) {
  // Create a tree map
  TreeMap < Integer, String > tree_map = new TreeMap < Integer, String > ();
  // Put elements to the map
  tree_map.put(10, "Red");
  tree_map.put(20, "Green");
  tree_map.put(40, "Black");
  tree_map.put(50, "White");
  tree_map.put(60, "Pink");
  // polling first entry
  System.out.println("Value before poll: " + tree_map);
  System.out.println("Value returned: " + tree_map.pollFirstEntry());
  System.out.println("Value after poll: " + tree_map);
 }
}
```

Sample Output:

```
Value before poll: {10=Red, 20=Green, 40=Black, 50=White, 60=Pink}
Value returned: 10=Red
Value after poll: {20=Green, 40=Black, 50=White, 60=Pink}
```

279. Write a Java program to get the portion of a map whose keys range from a given key to another key.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example22 {
          public static void main(String args[]) {
  // Declare tree maps
  TreeMap < Integer, String > tree_map = new TreeMap < Integer, String > ();
  SortedMap < Integer, String > sub_tree_map = new TreeMap < Integer, String > ();
  // Put elements to the map
  tree_map.put(10, "Red");
  tree_map.put(20, "Green");
  tree_map.put(30, "Black");
  tree_map.put(40, "White");
  tree_map.put(50, "Pink");
  System.out.println("Orginal TreeMap content: " + tree_map);
  sub_tree_map = tree_map.subMap(20, true, 40, true);
  System.out.println("Sub map key-values: " + sub_tree_map);
 }
}
```

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 30=Black, 40=White, 50=Pink
}
Sub map key-values: {20=Green, 30=Black, 40=White}
```

280. Write a Java program to get a key-value mapping associated with the least key greater than or equal to the given key. Return null if there is no such key.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example25 {
          public static void main(String args[]) {
  // Declare tree maps
  TreeMap < Integer, String > tree_map = new TreeMap < Integer, String > ();
  // Put elements to the map
  tree_map.put(10, "Red");
  tree_map.put(20, "Green");
  tree_map.put(30, "Black");
  tree_map.put(40, "White");
  tree_map.put(50, "Pink");
  System.out.println("Orginal TreeMap content: " + tree_map);
  System.out.println("Keys greater than or equal to 20: " + tree_map.ceilingEntry(20));
  System.out.println("Keys greater than or equal to 40: " + tree_map.ceilingEntry(40));
  System.out.println("Keys greater than or equal to 50: " + tree_map.ceilingEntry(50));
 }
}
```

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 30=Black, 40=White, 50=Pink
}
Keys greater than or equal to 20: 20=Green
Keys greater than or equal to 40: 40=White
```

```
Keys greater than or equal to 50: 50=Pink
```

281. Write a Java program to get the least key greater than or equal to the given key. Returns null if there is no such key.

```
import java.  util.*;
import java.  util.Map.Entry;
public class Example26 {
          public static void main(String args[]) {
  // Declare tree maps
  TreeMap < Integer, String > tree_map = new TreeMap < Integer, String > ();
  // Put elements to the map
  tree_map.put(10, "Red");
  tree_map.put(20, "Green");
  tree_map.put(40, "Black");
  tree_map.put(50, "White");
  tree_map.put(60, "Pink");
  System.out.println("Orginal TreeMap content: " + tree_map);
  System.out.println("Keys greater than or equal to 20: " + tree_map.ceilingKey(20));
  System.out.println("Keys greater than or equal to 30: " + tree_map.ceilingKey(30));
  System.out.println("Keys greater than or equal to 50: " + tree_map.ceilingKey(50));
 }
}
```

Sample Output:

```
Orginal TreeMap content: {10=Red, 20=Green, 40=Black, 50=White, 60=Pink
}
Keys greater than or equal to 20: 20
Keys greater than or equal to 30: 40
Keys greater than or equal to 50: 50
```

282. Write a Java program to reverse an integer number.

```
public class Example6 {
   public static void main(String[] args) {
    int num =1287;
    int is_positive = 1;
         if (num < 0) {
             is_positive = -1;
             num = -1 * num;
         }
         int sum  = 0;
         while (num > 0) {
             int r = num % 10;

             int maxDiff = Integer.MAX_VALUE - sum * 10;
             if (sum > Integer.MAX_VALUE / 10 || r > maxDiff)
              System.out.println("Wrong number");;

             sum = sum * 10 + r;
             num /= 10;
         }
         System.out.println(is_positive * sum);
```

```
    }
}
```

Sample Output:

283. Write a Java program to convert Roman number to an integer number.

```
public class Example7 {
   public static void main(String[] args) {
    String str = "DCCVII";
    int len = str.length();
         str = str + " ";
         int result = 0;
         for (int i = 0; i < len; i++) {
             char ch   = str.charAt(i);
             char next_char = str.charAt(i+1);

             if (ch == 'M') {
                 result += 1000;
             } else if (ch == 'C') {
                 if (next_char == 'M') {
                     result += 900;
                     i++;
                 } else if (next_char == 'D') {
                     result += 400;
                     i++;
                 } else {
                     result += 100;
                 }
             } else if (ch == 'D') {
                 result += 500;
             } else if (ch == 'X') {
                 if (next_char == 'C') {
                     result += 90;
                     i++;
                 } else if (next_char == 'L') {
                     result += 40;
                     i++;
                 } else {
                     result += 10;
                 }
             } else if (ch == 'L') {
                 result += 50;
             } else if (ch == 'I') {
                 if (next_char == 'X') {
                     result += 9;
                     i++;
                 } else if (next_char == 'V') {
                     result += 4;
                     i++;
                 } else {
                     result++;
                 }
             } else { // if (ch == 'V')
                 result += 5;
             }
         }
        System.out.println("\nRoman Number: "+str);
       System.out.println("Equivalent Integer: "+result+"\n");
    }
}
```

Sample Output:

```
Roman Number: DCCVII
Equivalent Integer: 70
```

284. Write a Java program to convert a float value to absolute value.

```
import java.  util.*;
public class Example9 {
 public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input a float number: ");
        float  x = in.nextFloat();
        System.out.printf("The absolute value of %.2f is: %.2f",x, convert(x));
       System.out.printf("\n");
    }
    public static float convert(float n)
   {
      float absvalue = (n >= 0) ? n : -n;
      return absvalue;
   }
}
```

Sample Output:

```
Input a float number: 12.53
The absolute value of 12.53 is: 12.53
```

285. Write a Java program to find the length of the longest sequence of zeros in binary representation of an integer.

Sample example:

Number 7 has binary representation 111 and has no binary gaps.

Number 8 has binary representation 1000 and contains a binary gap of length 0.

Number 457 has binary representation 111001001 and contains a binary gap of length 2.

Number 40 has binary representation 101000 and contains one binary gap of length 1.

Number 12546 has binary representation 11000100000010 and contains highest binary gap of length 6.

```
import java.  util.*;
 public class Example13 {
     public static void main(String[] args){

       int dec_num, rem, quot, i=1, j;
        int bin_num[] = new int[100];
        Scanner scan = new Scanner(System.in);

        System.out.print("Input a Decimal Number : ");
        dec_num = scan.nextInt();
```

```
        quot = dec_num;

        while(quot != 0)
        {
            bin_num[i++] = quot%2;
            quot = quot/2;
        }
       String binary_str="";
       System.out.print("Binary number is: ");
        for(j=i-1; j>0; j--)
        {
          binary_str = binary_str + bin_num[j];
        }
       System.out.print(binary_str);
        i = binary_str.length()-1;
        while(binary_str.charAt(i)=='0') {
            i--;
        }
        int length = 0;
        int ctr = 0;
        for(; i>=0; i--) {
            if(binary_str.charAt(i)=='1') {
                length = Math.max(length, ctr);
                ctr = 0;
            } else {
                ctr++;
            }
        }
        length = Math.max(length, ctr);
        System.out.println("\nLength of the longest sequence: "+length);
    }
}
```

Sample Output:

```
Input a Decimal Number : 7
Binary number is: 111
Length of the longest sequence: 0
```

286. Write a Java program to convert temperature from Fahrenheit to Celsius degree.

Test Data

Input a degree in Fahrenheit: 21

```
import java.  util.Scanner;
public class Exercise1 {
    public static void main(String[] Strings) {
        Scanner input = new Scanner(System.in);
        System.out.print("Input a degree in Fahrenheit: ");
        double fahrenheit = input.nextDouble();
        double  celsius =(( 5 *(fahrenheit - 32.0)) / 9.0);
```

```
        System.out.println(fahrenheit + " degree Fahrenheit is equal to " + celsius + " in
Celsius");
    }
}
```

Sample Output:

```
Input a degree in Fahrenheit: 212
212.0 degree Fahrenheit is equal to 100.0 in Celsius
```

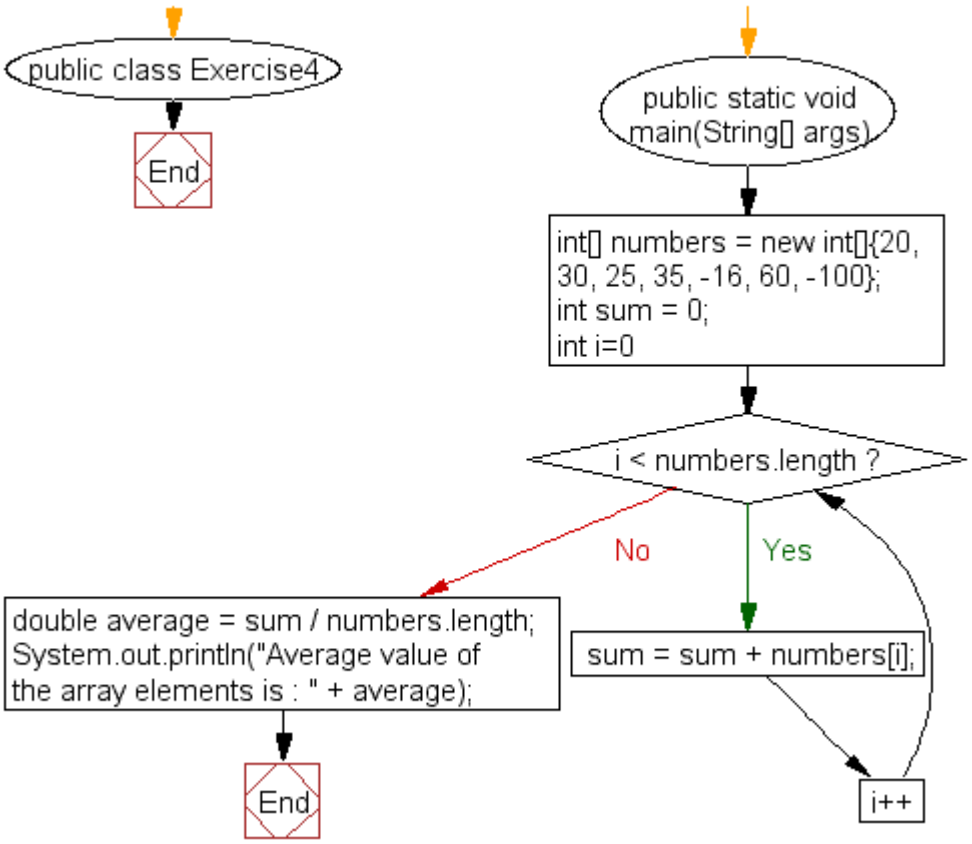
287. Write a Java program to calculate the average value of array elements.

```
public class Exercise4 {
public static void main(String[] args) {

       int[] numbers = new int[]{20, 30, 25, 35, -16, 60, -100};
       //calculate sum of all array elements
       int sum = 0;
       for(int i=0; i < numbers.length ; i++)
        sum = sum + numbers[i];
       //calculate average value
        double average = sum / numbers.length;
        System.out.println("Average value of the array elements is : " + average);
   }
}
```

Sample Output:

```
Average value of the array elements is : 7.0
```

Flowchart:



288. Write a Java program to remove a specific element from an array.

```
import java.  util.Arrays;
public class Exercise7 {

public static void main(String[] args) {
   int[] my_array = {25, 14, 56, 15, 36, 56, 77, 18, 29, 49};

   System.out.println("Original Array : "+Arrays.toString(my_array));

  // Remove the second element (index->1, value->14) of the array
   int removeIndex = 1;
   for(int i = removeIndex; i < my_array.length -1; i++){
        my_array[i] = my_array[i + 1];
      }
// We cannot alter the size of an array , after the removal, the last and second last element
in the array will exist twice
    System.out.println("After removing the second element: "+Arrays.toString(my_array));
 }
 }
```

Sample Output:

```
Original Array : [25, 14, 56, 15, 36, 56, 77, 18, 29, 49]
```

```
After removing the second element: [25, 56, 15, 36, 56, 77, 18, 29, 49, 49]
```

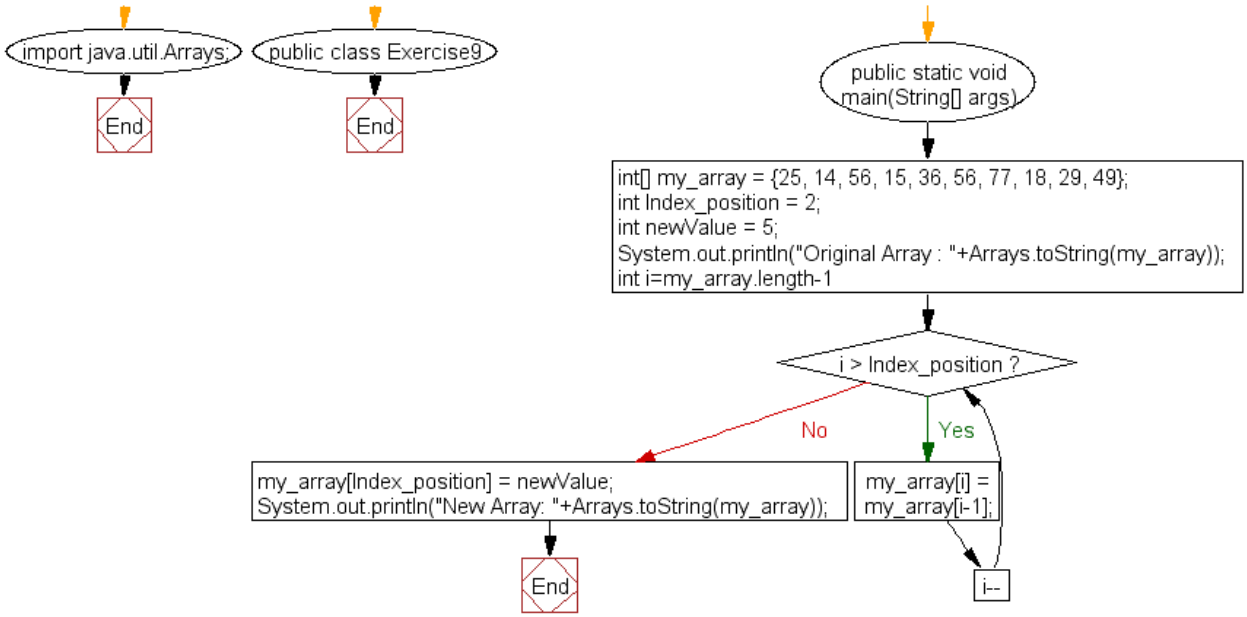
289. Write a Java program to insert an element (specific position) into an array.

```
import java.  util.Arrays;
public class Exercise9 {

public static void main(String[] args) {
   int[] my_array = {25, 14, 56, 15, 36, 56, 77, 18, 29, 49};
    // Insert an element in 3rd position of the array (index->2, value->5)

   int Index_position = 2;
   int newValue    = 5;
  System.out.println("Original Array : "+Arrays.toString(my_array));

  for(int i=my_array.length-1; i > Index_position; i--){
    my_array[i] = my_array[i-1];
   }
   my_array[Index_position] = newValue;
   System.out.println("New Array: "+Arrays.toString(my_array));
 }
 }
```

Sample Output:

```
Original Array : [25, 14, 56, 15, 36, 56, 77, 18, 29, 49]
New Array: [25, 14, 5, 56, 15, 36, 56, 77, 18, 29]
```

Flowchart:



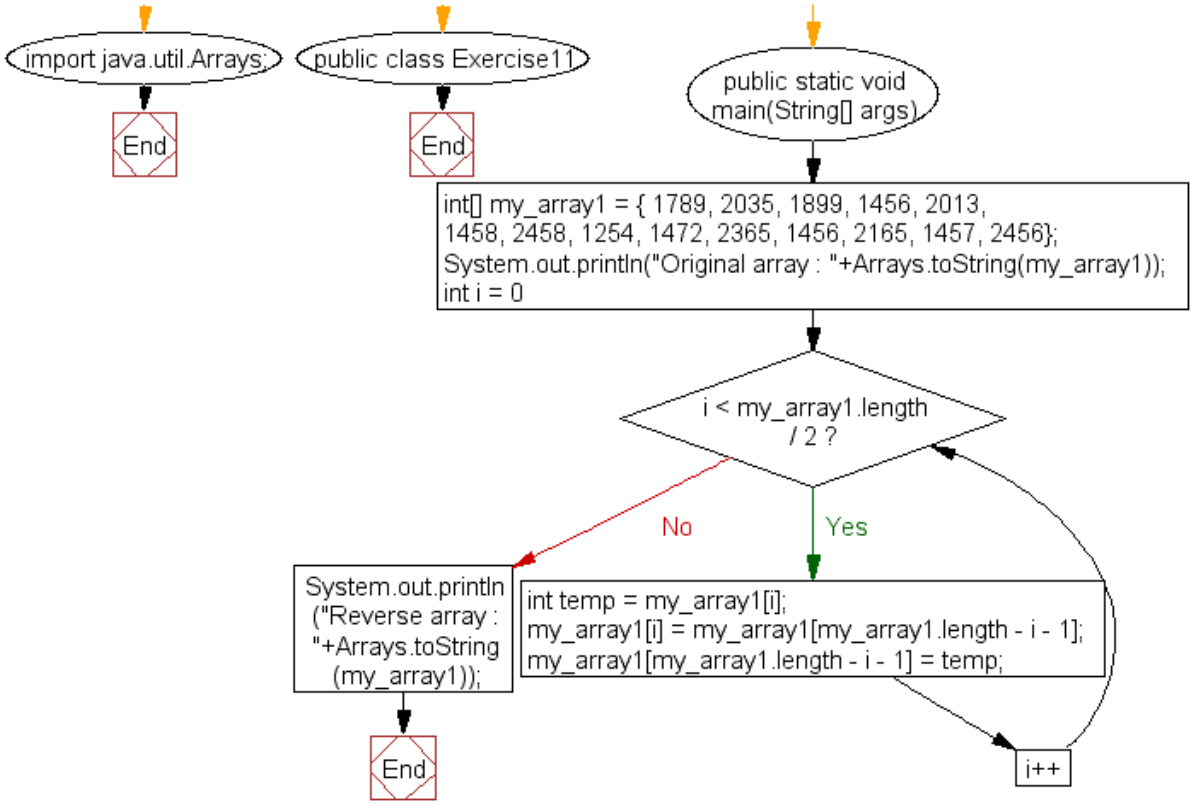
290. Write a Java program to reverse an array of integer values.

```
import java.  util.Arrays;
public class Exercise11 {
public static void main(String[] args){
    int[] my_array1 = {
            1789, 2035, 1899, 1456, 2013,
            1458, 2458, 1254, 1472, 2365,
            1456, 2165, 1457, 2456};
  System.out.println("Original array : "+Arrays.toString(my_array1));
   for(int i = 0; i < my_array1.length / 2; i++)
  {
    int temp = my_array1[i];
    my_array1[i] = my_array1[my_array1.length - i - 1];
    my_array1[my_array1.length - i - 1] = temp;
  }
    System.out.println("Reverse array : "+Arrays.toString(my_array1));
 }
}
```

Sample Output:

```
Original array : [1789, 2035, 1899, 1456, 2013, 1458, 2458, 1254, 1472, 2365, 1456,
2165, 1457, 2456]
Reverse array : [2456, 1457, 2165, 1456, 2365, 1472, 1254, 2458, 1458, 2013, 1456,
1899, 2035, 1789]
```

Flowchart:



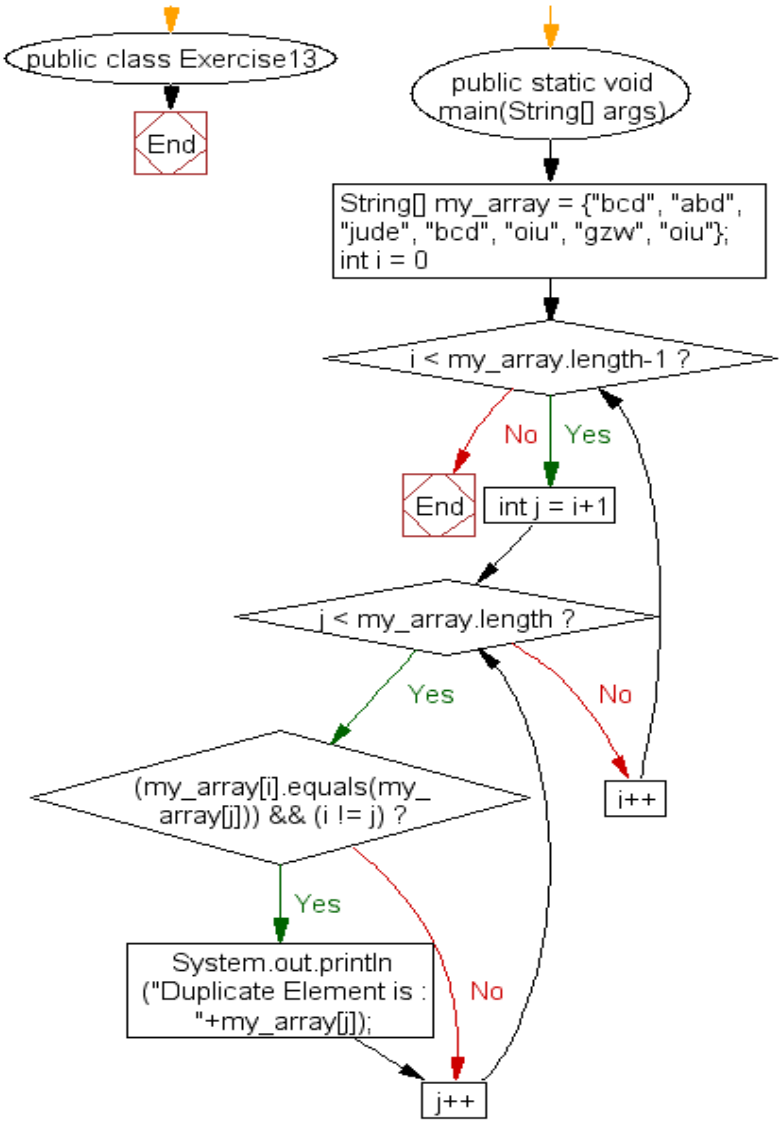
291. Write a Java program to find the duplicate values of an array of string values.

```
public class Exercise13 {
public static void main(String[] args)
    {
        String[] my_array = {"bcd", "abd", "jude", "bcd", "oiu", "gzw", "oiu"};

        for (int i = 0; i < my_array.length-1; i++)
        {
            for (int j = i+1; j < my_array.length; j++)
            {
                if( (my_array[i].equals(my_array[j])) && (i != j) )
                {
                    System.out.println("Duplicate Element is : "+my_array[j]);
                }
            }
        }
    }
}
```

Sample Output:

```
Duplicate Element is : bcd
Duplicate Element is : oiu
```

Flowchart:



292. Write a Java program to find the duplicate values of an array of integer values.

```
import java.  util.Arrays;
public class Exercise12 {
  public static void main(String[] args)
    {
        int[] my_array = {1, 2, 5, 5, 6, 6, 7, 2};

        for (int i = 0; i < my_array.length-1; i++)
        {
            for (int j = i+1; j < my_array.length; j++)
            {
                if ((my_array[i] == my_array[j]) && (i != j))
                {
                    System.out.println("Duplicate Element : "+my_array[j]);
```

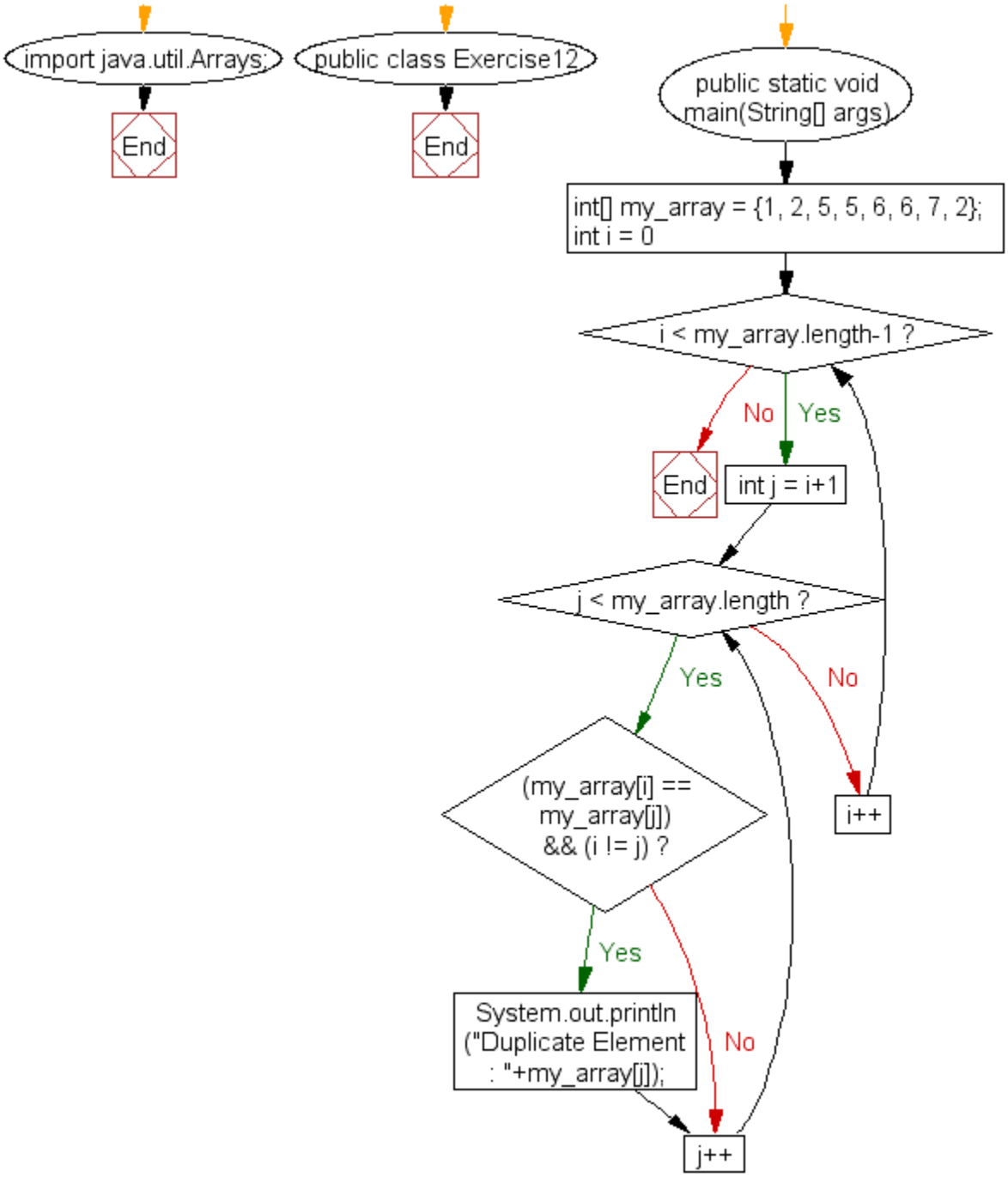
```
                }
            }
        }
    }
}
```

Sample Output:

```
Duplicate Element : 2
Duplicate Element : 5
```

```
Duplicate Element :
```

Flowchart:



293. Write a Java program to find the common elements between two arrays (string values).

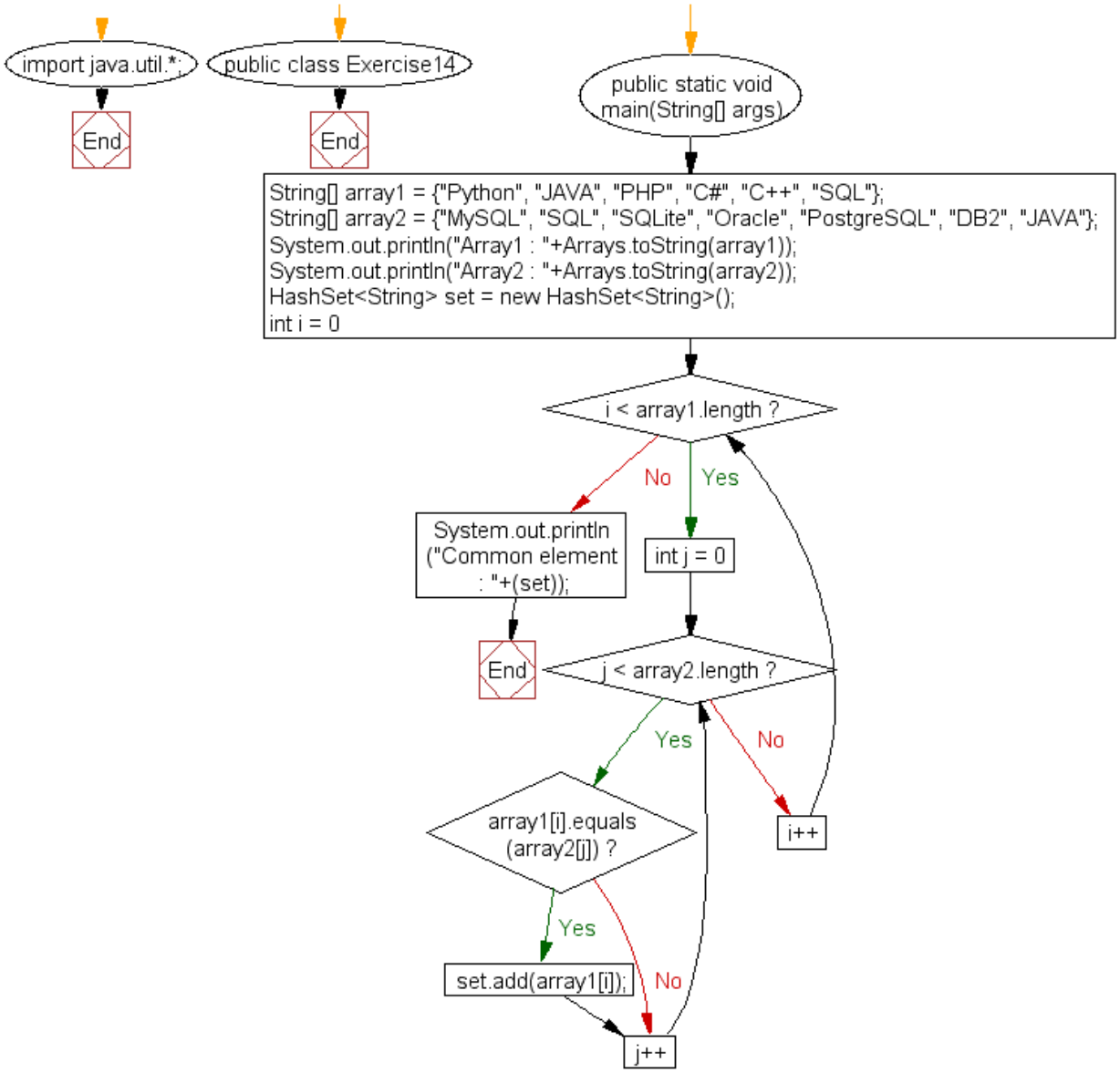
```
import java.  util.*;
public class Exercise14 {
public static void main(String[] args)
    {
       String[] array1 = {"Python", "JAVA", "PHP", "C#", "C++", "SQL"};
```

```
        String[] array2 = {"MySQL", "SQL", "SQLite", "Oracle", "PostgreSQL", "DB2", "JAVA"};

        System.out.println("Array1 : "+Arrays.toString(array1));
        System.out.println("Array2 : "+Arrays.toString(array2));

        HashSet<String> set = new HashSet<String>();

         for (int i = 0; i < array1.length; i++)
         {
             for (int j = 0; j < array2.length; j++)
             {
                 if(array1[i].equals(array2[j]))
                 {
                     set.add(array1[i]);
                 }
             }
         }

         System.out.println("Common element : "+(set));      //OUTPUT : [THREE, FOUR, FIVE]
    }
}
```

Sample Output:

```
Array1 : [Python, JAVA, PHP, C#, C++, SQL]
Array2 : [MySQL, SQL, SQLite, Oracle, PostgreSQL, DB2, JAVA]
Common element is : [JAVA, SQL]
```

Flowchart:



294. Write a Java program to remove duplicate elements from an array.

```
import java.  util.Arrays;

public class Exercise16 {
   static void unique_array(int[] my_array)
    {
        System.out.println("Original Array : ");

        for (int i = 0; i < my_array.length; i++)
        {
            System.out.print(my_array[i]+"\t");
        }
```

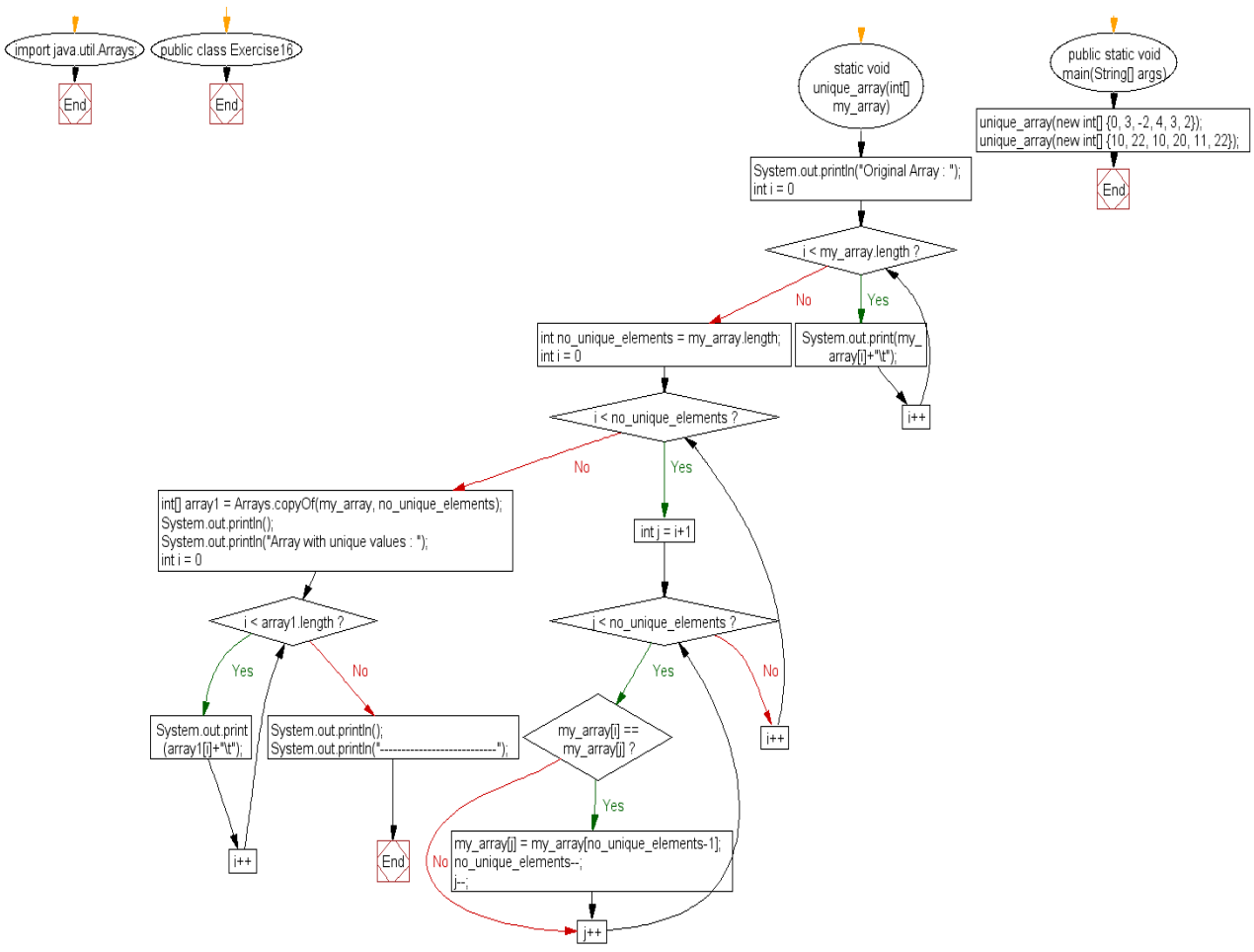
```
        //Assuming all elements in input array are unique

        int no_unique_elements = my_array.length;

        //Comparing each element with all other elements

        for (int i = 0; i < no_unique_elements; i++)
        {
            for (int j = i+1; j < no_unique_elements; j++)
            {
                //If any two elements are found equal

                if(my_array[i] == my_array[j])
                {
                    //Replace duplicate element with last unique element

                    my_array[j] = my_array[no_unique_elements-1];

                    no_unique_elements--;

                    j--;
                }
            }
        }

        //Copying only unique elements of my_array into array
        int[] array1 = Arrays.copyOf(my_array, no_unique_elements);

        //Printing arrayWithoutDuplicates

        System.out.println();

        System.out.println("Array with unique values : ");

        for (int i = 0; i < array1.length; i++)
        {
            System.out.print(array1[i]+"\t");
        }

        System.out.println();

        System.out.println("----------------------------");
    }

    public static void main(String[] args)
    {
        unique_array(new int[] {0, 3, -2, 4, 3, 2});

        unique_array(new int[] {10, 22, 10, 20, 11, 22});

      }
}
```

Sample Output:

```
Original Array :
```

```
0       3       -2      4       3       2
Array with unique values :
0       3       -2      4       2
---------------------------
Original Array :
10      22      10      20      11      22
Array with unique values :
10      22      11      20
---------------------------
```

Flowchart:



295. Write a Java program to add two matrices of the same size.

Java Code :

```
import java.  util.Scanner;
public class Exercise19 {
 public static void main(String args[])
```

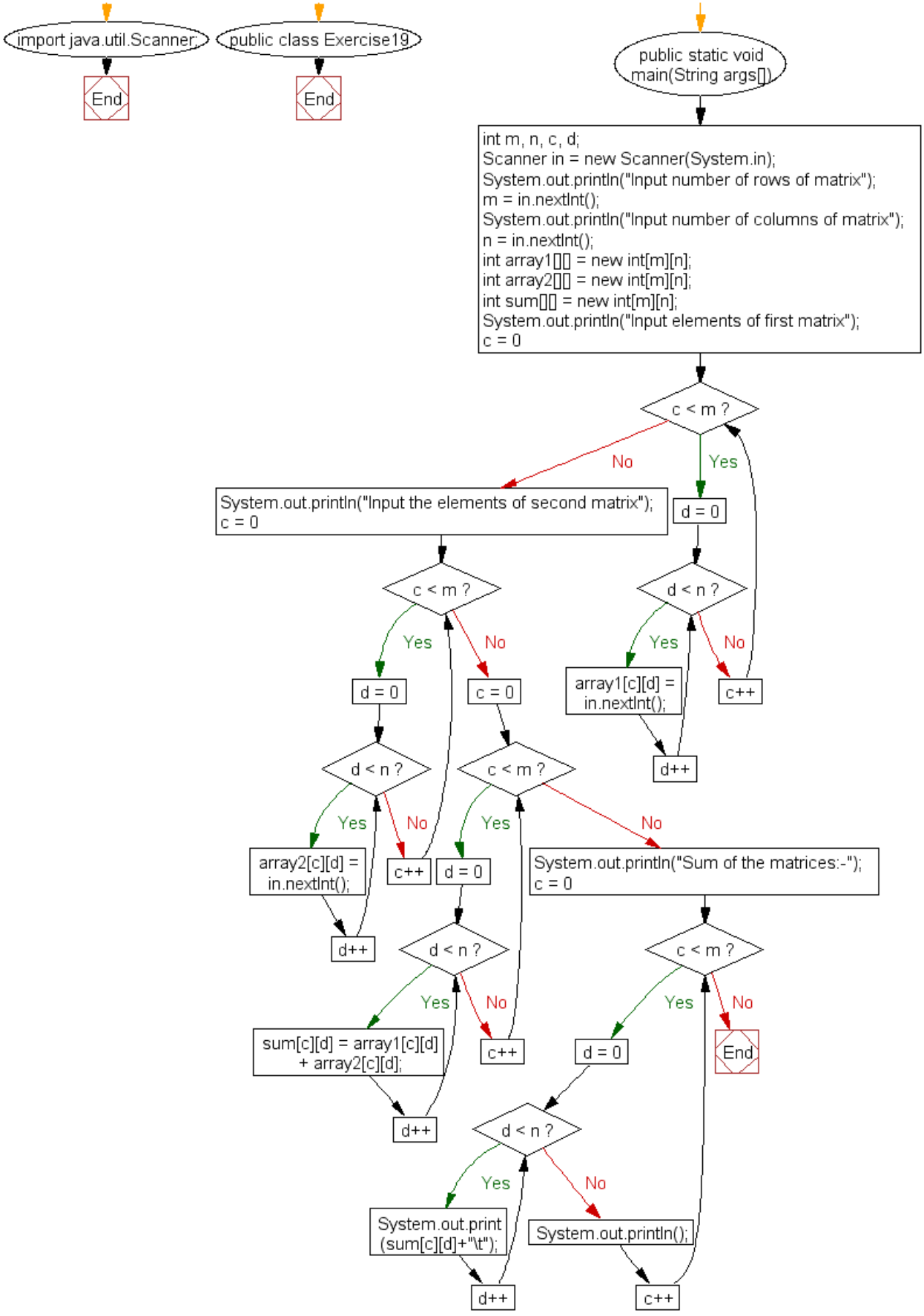
```
   {
      int m, n, c, d;
      Scanner in = new Scanner(System.in);

      System.out.println("Input number of rows of matrix");
      m = in.nextInt();
      System.out.println("Input number of columns of matrix");
      n  = in.nextInt();

      int array1[][] = new int[m][n];
      int array2[][] = new int[m][n];
      int sum[][] = new int[m][n];

      System.out.println("Input elements of first matrix");

      for (  c = 0 ; c < m ; c++ )
         for ( d = 0 ; d < n ; d++ )
            array1[c][d] = in.nextInt();

      System.out.println("Input the elements of second matrix");

      for ( c = 0 ; c < m ; c++ )
         for ( d = 0 ; d < n ; d++ )
            array2[c][d] = in.nextInt();

      for ( c = 0 ; c < m ; c++ )
         for ( d = 0 ; d < n ; d++ )
             sum[c][d] = array1[c][d] + array2[c][d];

      System.out.println("Sum of the matrices:-");

      for ( c = 0 ; c < m ; c++ )
      {
         for ( d = 0 ; d < n ; d++ )
            System.out.print(sum[c][d]+"\t");

         System.out.println();
      }
   }
}
```

Sample Output:

```
Input number of rows of matrix
2
Input number of columns of matrix
2
Input elements of first matrix
1
2
3
4
Input the elements of second matrix
5
6
7
```

```
8
Sum of the matrices:-
6       8
10      12
```

Flowchart:


import java.util.Scanner;
End
public class Exercise19
End
public static void main(String args[])
int m, n, c, d;
Scanner in = new Scanner(System.in);
System.out.println("Input number of rows of matrix");
m = in.nextInt();
System.out.println("Input number of columns of matrix");
n = in.nextInt();
int array1[][] = new int[m][n];
int array2[][] = new int[m][n];
int sum[][] = new int[m][n];
System.out.println("Input elements of first matrix");
c = 0
c < m ?
No
Yes
System.out.println("Input the elements of second matrix");
c = 0
d = 0
c < m ?
Yes
No
d = 0
c = 0
d < n ?
Yes
No
array1[c][d] = in.nextInt();
c++
d++
d < n ?
Yes
No
array2[c][d] = in.nextInt();
c++
d++
c < m ?
Yes
No
d = 0
System.out.println("Sum of the matrices:-");
c = 0
d < n ?
Yes
No
sum[c][d] = array1[c][d] + array2[c][d];
c++
d++
c < m ?
Yes
No
d = 0
End
d < n ?
Yes
No
System.out.print (sum[c][d]+"\t");
System.out.println();
d++
c++

296. Write a Java program to convert an array to ArrayList.

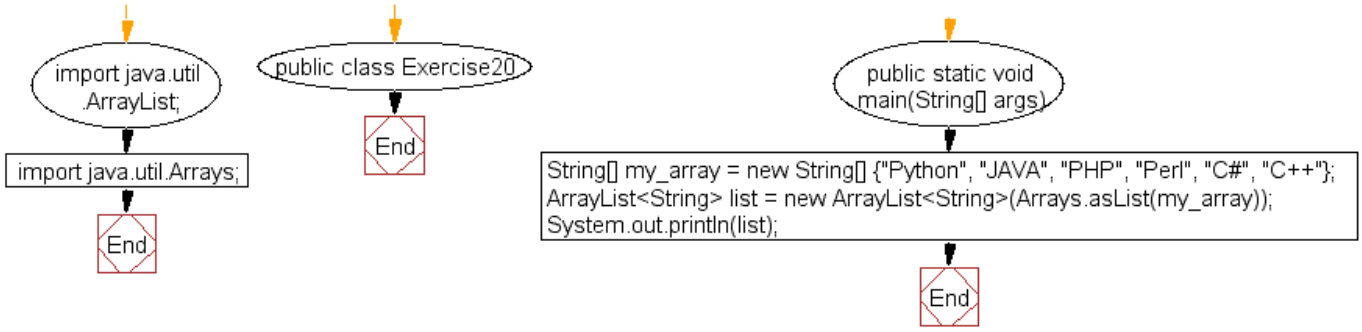
Java Code :

```
import java.  util.ArrayList;
  import java.  util.Arrays;
public class Exercise20 {
  public static void  main(String[] args)
  {
  String[]  my_array = new String[] {"Python", "JAVA", "PHP",  "Perl", "C#", "C++"};
  ArrayList<String>  list = new ArrayList<String>(Arrays.asList(my_array));

  System.out.println(list);
  }
}
```

Sample Output:

```
[Python, JAVA, PHP, Perl, C#, C++]
```

Flowchart:



297. Write a Java program to convert an ArrayList to an array.

Java Code :

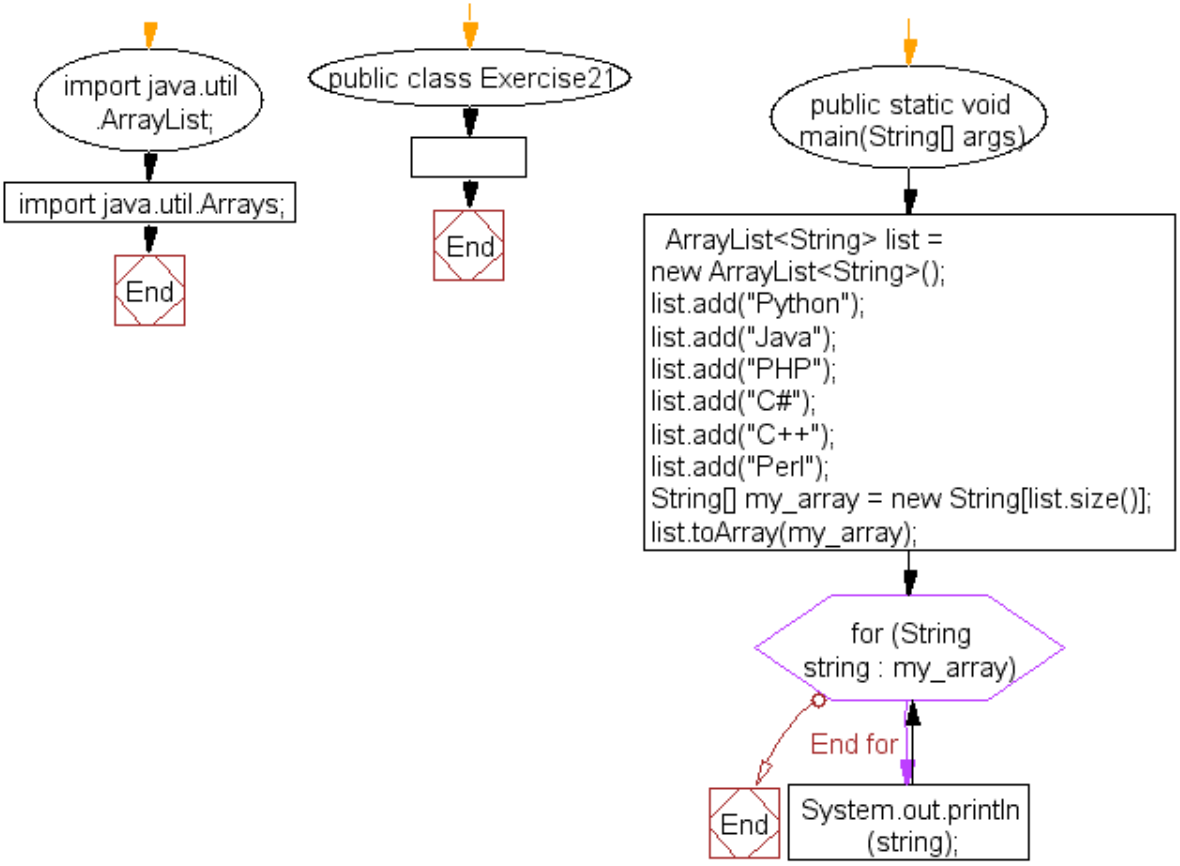
```
import java.  util.ArrayList;
  import java.  util.Arrays;
  public class Exercise21 {
  public static void  main(String[] args)
  {
  ArrayList<String> list = new ArrayList<String>();
  list.add("Python");

  list.add("Java");
```

```
list.add("PHP");

list.add("C#");

list.add("C++");

list.add("Perl");

String[]  my_array = new String[list.size()];

list.toArray(my_array);

for (String  string : my_array)
{
System.out.println(string);
}
}
}
```

Sample Output:

```
Python
Java
PHP
C#
C++
Perl
```

Flowchart:

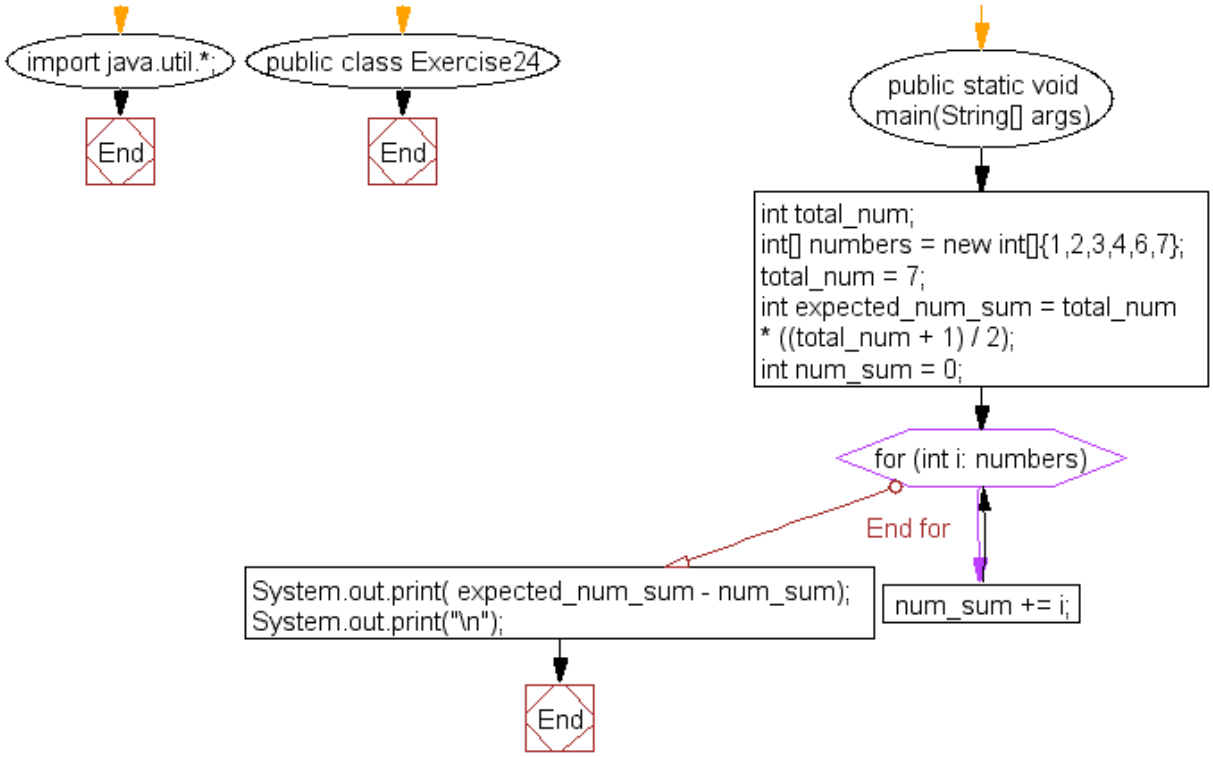


298. Write a Java program to find a missing number in an array.

```
import java.  util.*;
public class Exercise24 {
 public static void main(String[] args) {
   int total_num;
   int[] numbers = new int[]{1,2,3,4,6,7};
   total_num = 7;
   int expected_num_sum = total_num * ((total_num + 1) / 2);
   int num_sum = 0;
   for (int i: numbers) {
    num_sum += i;
   }
       System.out.print( expected_num_sum - num_sum);
      System.out.print("\n");
  }
 }
```

Sample Data: 1,2,3,4,6,Sample Output:

Flowchart:



299. Write a Java program to move all 0's to the end of an array. Maintain the relative order of the other (non-zero) array elements.

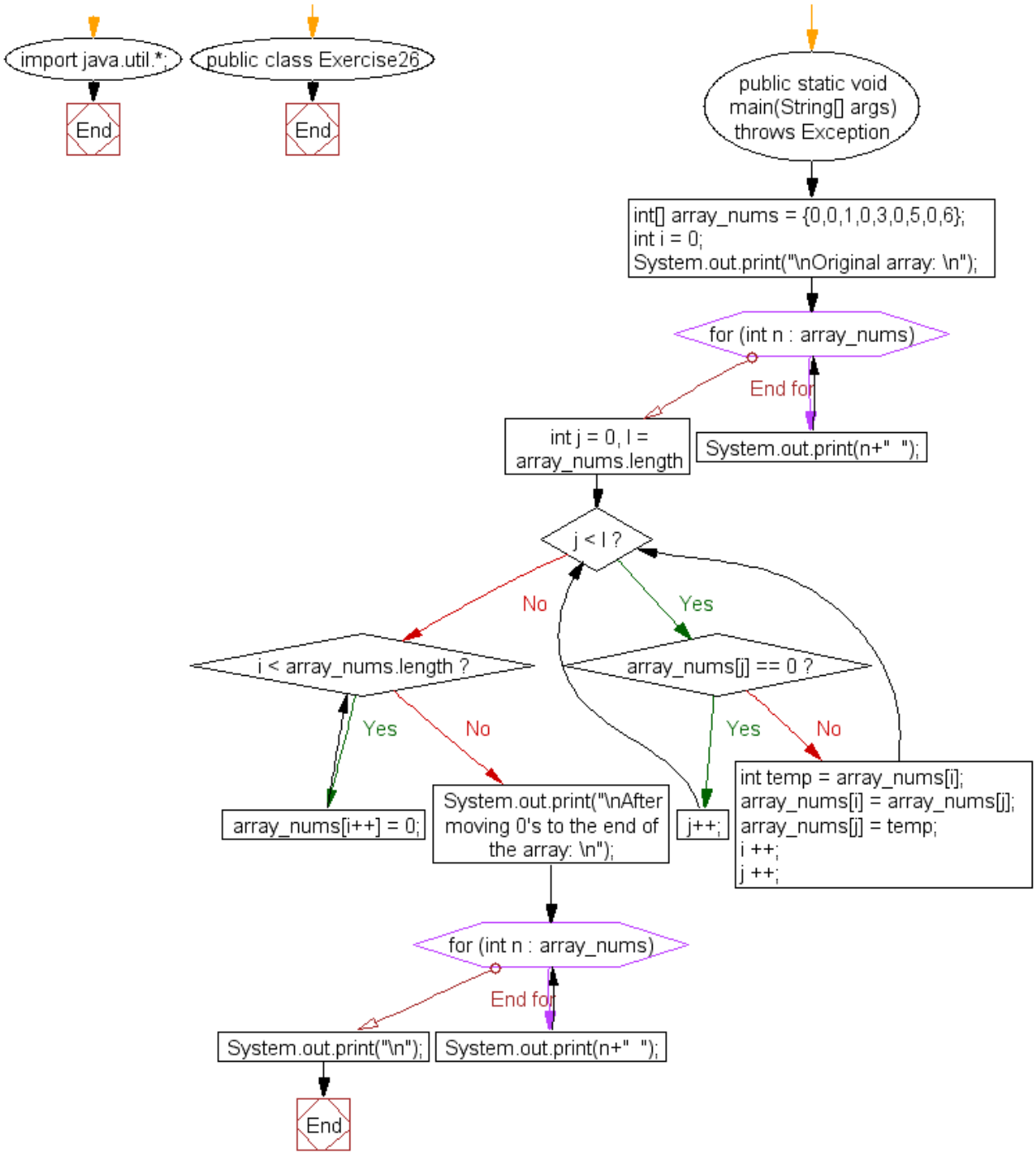
```
import java.  util.*;
 public class Exercise26 {
     public static void main(String[] args) throws Exception {
        int[] array_nums = {0,0,1,0,3,0,5,0,6};
         int i = 0;
     System.out.print("\nOriginal array: \n");
     for (int n : array_nums)
            System.out.print(n+"  ");

        for(int j = 0, l = array_nums.length; j < l;) {
            if(array_nums[j] == 0)
                j++;
            else {
                int temp = array_nums[i];
                array_nums[i] = array_nums[j];
                array_nums[j] = temp;
                i ++;
                j ++;
            }
        }
        while (i < array_nums.length)
```

```
            array_nums[i++] = 0;
     System.out.print("\nAfter moving 0's to the end of the array: \n");
        for (int n : array_nums)
            System.out.print(n+"  ");
       System.out.print("\n");
    }
}
```

Sample Output:

```
Original array:
0  0  1  0  3  0  5  0  6
After moving 0's to the end of the array:
1  3  5  6  0  0  0  0  0
```

Flowchart:



300. Write a Java program to compute the average value of an array of integers except the largest and smallest values.

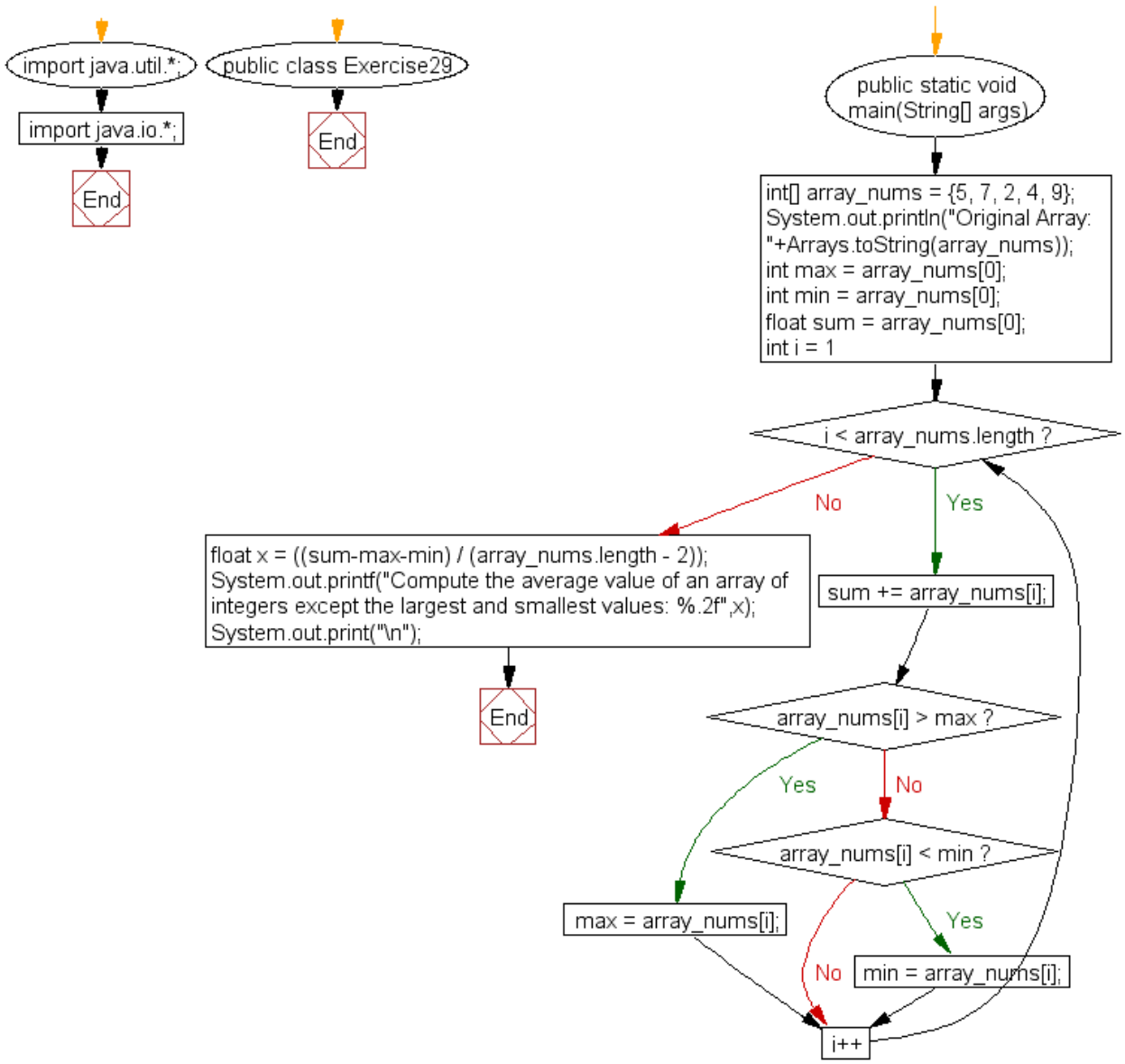
```
import java.  util.*;
import java.  io.*;
 public class Exercise29 {
 public static void main(String[] args)
```

```
{
    int[] array_nums = {5, 7, 2, 4, 9};
   System.out.println("Original Array: "+Arrays.toString(array_nums));
   int max = array_nums[0];
   int min = array_nums[0];
   float sum = array_nums[0];
   for(int i = 1; i < array_nums.length; i++)
   {
     sum  += array_nums[i];
     if(array_nums[i] > max)
       max = array_nums[i];
     else if(array_nums[i] < min)
       min = array_nums[i];
   }
   float x = ((sum-max-min) / (array_nums.length - 2));
   System.out.printf("Compute the average value of an array of integers except the largest and
smallest values: %.2f",x);
   System.out.print("\n");
  }
}
```

Sample Output:

```
Original Array: [5, 7, 2, 4, 9]
Compute the average value of an array of integers except the largest an
d smallest values: 5.3
```

Flowchart:



301. Write a Java program to check if an array of integers contains two specified elements 65 and 77.

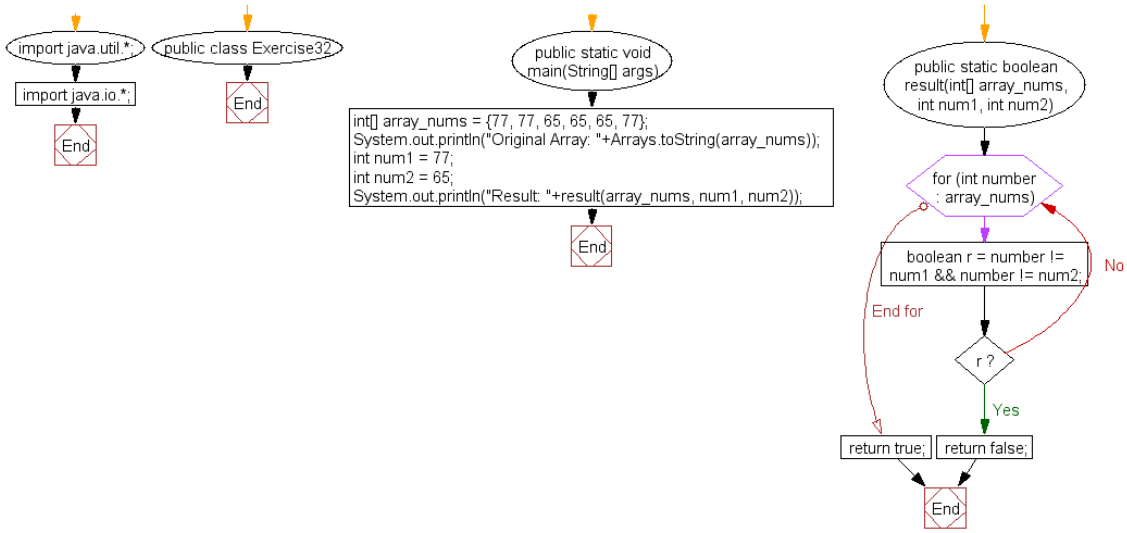
```
import java.  util.*;
import java.  io.*;
 public class Exercise32 {
 public static void main(String[] args)
 {
    int[] array_nums = {77, 77, 65, 65, 65, 77};
   System.out.println("Original Array: "+Arrays.toString(array_nums));
   int num1 = 77;
    int num2 = 65;

   System.out.println("Result: "+result(array_nums, num1, num2));
    }
```

```
  public static boolean result(int[] array_nums, int num1, int num2) {
    for (int number : array_nums) {
      boolean r = number != num1 && number != num2;
      if (r) {
        return false;
        }
     }
        return true;
   }
}
```

Sample Output:

```
Original Array: [77, 77, 65, 65, 65, 77]
Result: true
```

Flowchart:



302. Write a Java program to find the length of the longest consecutive elements sequence from a given unsorted array of integers.

Sample array: [49, 1, 3, 200, 2, 4, 70, 5]

The longest consecutive elements sequence is [1, 2, 3, 4, 5], therefore the program will return its length 5.

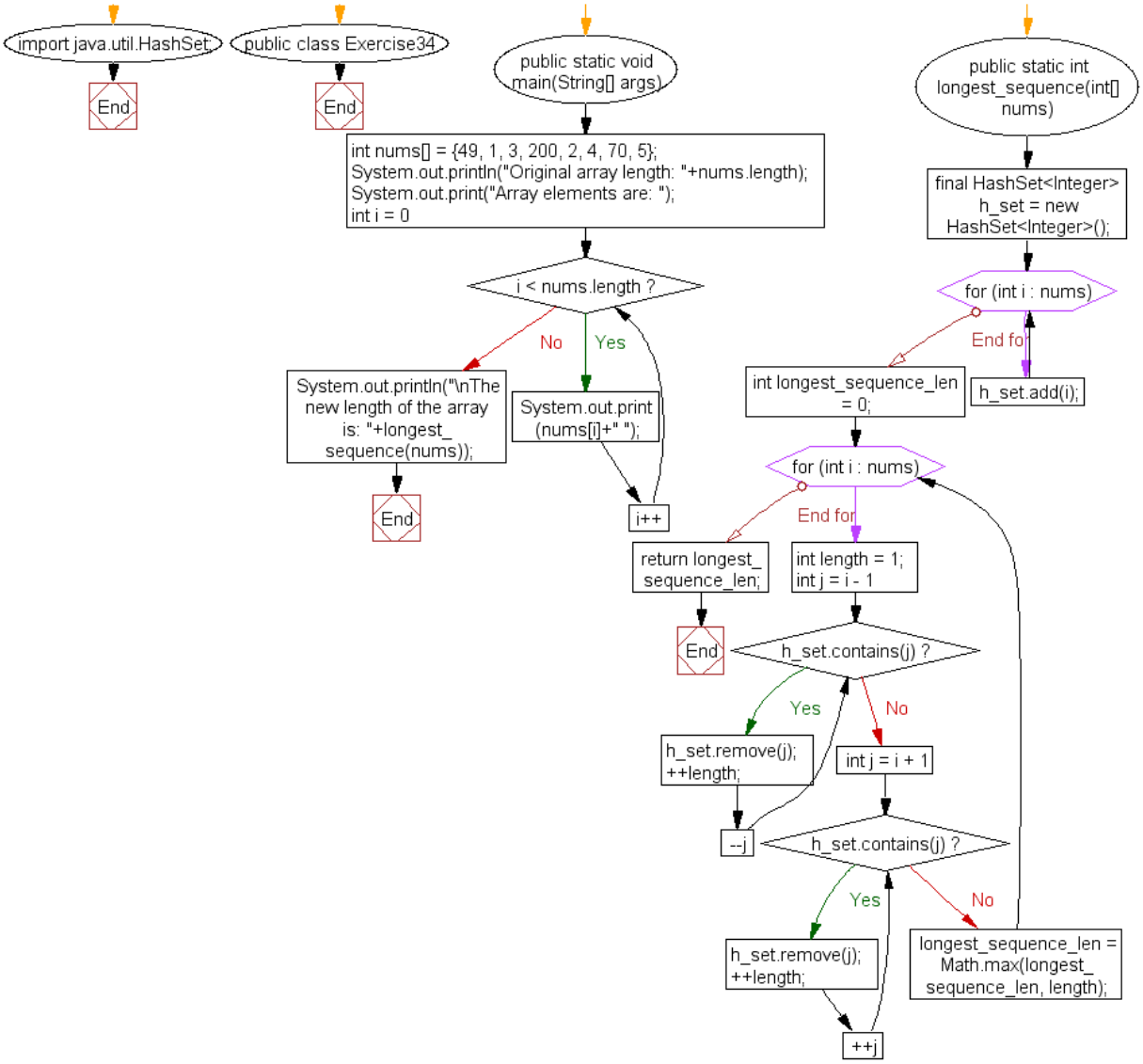
```
import java.  util.HashSet;
public class Exercise34 {
   public static void main(String[] args) {
        int nums[] = {49, 1, 3, 200, 2, 4, 70, 5};
     System.out.println("Original array length: "+nums.length);
     System.out.print("Array elements are: ");
       for (int i = 0; i < nums.length; i++)
        {
            System.out.print(nums[i]+" ");
        }
```

```
     System.out.println("\nThe new length of the array is: "+longest_sequence(nums));

    }

    public static int longest_sequence(int[] nums) {
      final HashSet<Integer> h_set = new HashSet<Integer>();
        for (int i : nums) h_set.add(i);
        int longest_sequence_len = 0;
        for (int i : nums) {
            int length = 1;
            for (int j = i - 1; h_set.contains(j); --j) {
                h_set.remove(j);
                ++length;
            }
            for (int j = i + 1; h_set.contains(j); ++j) {
                h_set.remove(j);
                ++length;
            }
            longest_sequence_len = Math.max(longest_sequence_len, length);
        }
        return longest_sequence_len;
    }
}
```

Sample Output:

```
Original array length: 8
Array elements are: 49 1 3 200 2 4 70 5

The new length of the array is:
```

Flowchart:



303. Write a Java program to find all the unique triplets such that sum of all the three elements [x, y, z (x ≤ y ≤ z)] equal to a specified number.

Sample array: [1, -2, 0, 5, -1, -4]

Target value: 2.

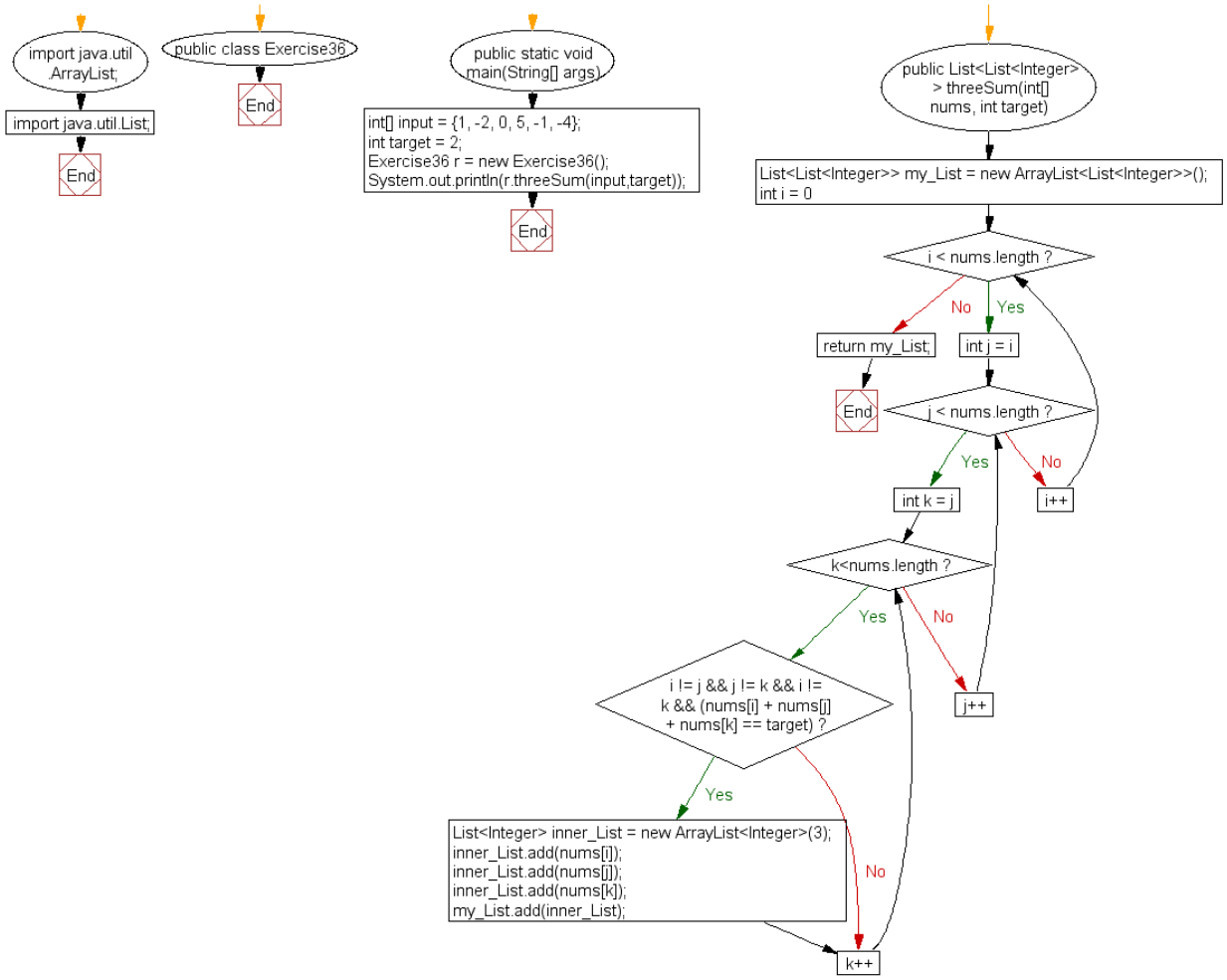
```
import java.  util.ArrayList;
import java.  util.List;
public class Exercise36 {
    public static void main(String[] args) {
        int[] input = {1, -2, 0, 5, -1, -4};
     int target = 2;
     Exercise36 r = new Exercise36();
        System.out.println(r.threeSum(input,target));
    }
    public List<List<Integer>> threeSum(int[] nums, int target) {
```

```
        List<List<Integer>> my_List = new ArrayList<List<Integer>>();

        for(int i = 0; i < nums.length; i++){
            for(int j = i; j < nums.length ;j++){
                for(int k = j; k<nums.length;k++){
                    if ( i != j && j != k && i != k && (nums[i] + nums[j] + nums[k] ==
target)){
                        List<Integer> inner_List = new ArrayList<Integer>(3);
                        inner_List.add(nums[i]);
                        inner_List.add(nums[j]);
                        inner_List.add(nums[k]);
                        my_List.add(inner_List);
                    }
                }
            }
        }
       return my_List;
    }
  }
```

Sample Output:

```
[[1, 5, -4], [-2, 5, -1]]
```

Flowchart:



Write a Java program to create an array of its anti-diagonals from a given square matrix.

Example:

Input :

1 2

3 4

Output:

[

[1],

[2, 3],

[4]

]

Input:

[10, 20, 30]

[50, 60, 70]

[90, 100, 110]

Output:
[10]
[20, 50]
[30, 60, 90]
[70, 100]
[110]

```
//https://github.  com/nagajyothi/Arrays/blob/master/Diagonal.java
import java.  util.*;
public class Exercise3{
    public static ArrayList<ArrayList<Integer>> Exercise37(ArrayList<ArrayList<Integer>> A) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        int m = a.  size();
        int n = a.  get(0).size();
        ArrayList<Integer> temp = new ArrayList<Integer>();
        temp.add(a.  get(0).get(0));
        result.add(new ArrayList<Integer>(temp));
        int i = 0;
        while(i < m){
            System.out.printf("For i : %d \n", i );
            int j = i+1;
            while(j < n){
                int k = i;
                int l = j;
                temp.clear();
                System.out.printf("\t For j : %d \n", j );
                while(l >= 0 && k < m){
                    System.out.printf("\t \t For k : %d and l : %d  add \n", k, l, a.
get(k).get(l) );
                    temp.add(a.  get(k).get(l));
                    k++;
                    l--;
                }
                System.out.println("\t \t Temp : " + temp);

                result.add(new ArrayList<Integer>(temp));
                j++;
            }


            i++;
        }
        temp.clear();
        temp.add(a.  get(m-1).get(n-1));
        result.add(new ArrayList<Integer>(temp));
        return result;
    }

    public static ArrayList<ArrayList<Integer>>
diagonalEfficient(ArrayList<ArrayList<Integer>> A) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        int m = a.  size();
        int n = a.  get(0).size();
        ArrayList<Integer> temp = new ArrayList<Integer>();
        temp.add(a.  get(0).get(0));
```

```
        result.add(new ArrayList<Integer>(temp));


            int j;
            int i = 0;
            j = i+1;
            while(j < n){
                int k = i;
                int l = j;
                temp.clear();
                while(k < m && l >= 0){
                    temp.add(a.  get(k).get(l));
                    k++;
                    l--;
                }
                result.add(new ArrayList<Integer>(temp));
                j++;
            }

            i = 1;
            j = n-1;
            while(i < m){
                int k = i;
                int l = j;
                temp.clear();
                while(k < m && l >= 0){
                    temp.add(a.  get(k).get(l));
                    k++;
                    l--;
                }
                result.add(new ArrayList<Integer>(temp));
                i++;
            }



        temp.clear();
        return result;
    }
    public static void main(String[] args){
        ArrayList<ArrayList<Integer>> A = new ArrayList<ArrayList<Integer>>();
        ArrayList<Integer> temp = new ArrayList<Integer>();
        temp.add(10);
        temp.add(20);
        temp.add(30);
        a.  add(new ArrayList<Integer>(temp));
        temp.clear();

        temp.add(50);
        temp.add(60);
        temp.add(70);
        a.  add(new ArrayList<Integer>(temp));
        temp.clear();

        temp.add(90);
```

```
        temp.add(100);
        temp.add(110);
        a.  add(new ArrayList<Integer>(temp));
        temp.clear();
        for(ArrayList<Integer> t : A)
            System.out.println(t);

        ArrayList<ArrayList<Integer>> result  = diagonalEfficient(A);
        for(ArrayList<Integer> t : result)
            System.out.println(t);
    }
}
```

Sample Output:

```
[
 [1],
 [2, 3],
 [4]
 ]
```

304. Write a Java program that reads a number and display the square, cube, and fourth power.

Test Data

Input the side length value: 1

```
import java.  util.Scanner;
public class Exercise8 {
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Input the side length value: ");
        double val = in.nextDouble();
        System.out.printf("Square: %12.2f\n", val * val);
        System.out.printf("Cube: %14.2f\n", val * val * val);
        System.out.printf("Fourth power: %6.2f\n", Math.pow(val, 4));
    }
}
```

Sample Output:

```
Input the side length value: 15
Square: .2f
Cube: .2f
Fourth power: 50625.00
```