1. Wap to print Fibonnaci series eg 1 1 2 3 5 8 …. up to a given number

```
class Fibonacci{
public static void main(String args[])
{
 int n1=0,n2=1,n3,i,count=10;
 System.out.print(n1+" "+n2);//printing 0 and 1
 for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 }
}
}
```

Output: 0 1 1 2 3 5 8 13 21 34

2. Wap to check if given number is prime number or not.

```
public class Prime{
 public static void main(String args[]){
  int i,m=0,flag=0;
  int n=3;//it is the number to be checked
  m=n/2;
  if(n==0||n==1){
   System.out.println(n+" is not prime number");
  }else{
```

```
  for(i=2;i<=m;i++){
   if(n%i==0){
    System.out.println(n+" is not prime number");
    flag=1;
    break;
   }
  }
  if(flag==0)  { System.out.println(n+" is prime number"); }
 }//end of else
}
}
```

Output:

3 is prime number

3. Wap to check if given string is palindrome or not.

```
public class Palindrome
{
   public static void main(String args[])
   {
      String a, b = "";
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the string you want to check:");
      a = s.nextLine();
      int n = a.length();
      for(int i = n - 1; i >= 0; i--)
      {
         b = b + a.charAt(i);
      }
      if(a.equalsIgnoreCase(b))
      {
         System.out.println("The string is palindrome.");
      }
```

```
        else
        {
            System.out.println("The string is not palindrome.");
        }
    }
}
```

Output:

```
Enter the string you want to check: abba
The string is palindrome.
```

4. Wap to check if given integer is palindrome or not.

```
class Palindrome{
 public static void main(String args[]){
  int r,sum=0,temp;
  int n=313;//It is the number variable to be checked for palindrome
  temp=n;
  while(n>0){
   r=n%10;  //getting remainder
   sum=(sum*10)+r;
   n=n/10;
  }
  if(temp==sum)
   System.out.println("Palindrome Number ");
  else
   System.out.println("not palindrome");
}
}
```

Output:

```
Palindrome Number
```

5. Wap to check if given number is armstrong or not.

```
class Armstrong{
  public static void main(String[] args)  {
```

```
   int c=0,a,temp;
   int n=371;//It is the number to check armstrong
   temp=n;
   while(n>0)
   {
   a=n%10;
   n=n/10;
   c=c+(a*a*a);
   }
   if(temp==c)
   System.out.println("Armstrong Number");
   else
      System.out.println("Not Armstrong Number");
  }
}
```

Output:

Armstrong Number

6. Wap for Factorial.

```
class Factorial{
 public static void main(String args[]){
  int i,fact=1;
  int number=6;//It is the number to calculate factorial
  for(i=1;i<=number;i++){
     fact=fact*i;
  }
  System.out.println("Factorial of "+number+" is: "+fact);
 }
}
```

Output:

Factorial of 6 is: 720

7. WAP to Reverse the String.

1) By StringBuilder / StringBuffer

```
public class StringFormatter {
public static String reverseString(String str){
   StringBuilder sb=new StringBuilder(str);
   sb.reverse();
   return sb.toString();
}
}
public class TestStringFormatter {
public static void main(String[] args) {
   System.out.println(StringFormatter.reverseString("India"));
   System.out.println(StringFormatter.reverseString("Hindustan"));
   }
}
```

Output:

```
aidnI
natsudniH
```

2) By Reverse Iteration

```
public class StringFormatter {
public static String reverseString(String str){
   char ch[]=str.toCharArray();
   String rev="";
   for(int i=ch.length-1;i>=0;i--){
      rev+=ch[i];
   }
   return rev;
}
}
public class TestStringFormatter {
public static void main(String[] args) {
   System.out.println(StringFormatter.reverseString("India"));
```

```
    System.out.println(StringFormatter.reverseString("Hindustan"));
    }
}
```

Output:

```
aidnI
natsudniH
```

8. Wap to remove duplicates from array.

```
public class RemoveDuplicate{
public static int removeDuplicate(int arr[], int n){
        if (n==0 || n==1){
            return n;
        }
        int[] temp = new int[n];
        int j = 0;
        for (int i=0; i<n-1; i++){
            if (arr[i] != arr[i+1]){
                temp[j++] = arr[i];
            }
         }
        temp[j++] = arr[n-1];
        // Changing original array
        for (int i=0; i<j; i++){
            arr[i] = temp[i];
        }
        return j;
    }
    public static void main (String[] args) {
        int arr[] = {5,10,20,20,30,5,30,40,50,50};
        int length = arr.length;
        length = removeDuplicate(arr, length);
        //printing array elements
```

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

Output:

5 10 20 30 40 50

9. Wap to calculate square root of number.

```
public class Squareroot {
        private static Scanner c;
        public static void main(String[] args)
        {
                double number, squareRoot;
                c = new Scanner(System.in);
                System.out.print(" Please Enter any Number : ");
                number = c.nextDouble();
                squareRoot = Math.sqrt(number);
                System.out.println("\n The Square of a Given Number  " + number + "  =  " +
squareRoot);
        }
}
```

Output:

Please enter any number : 144

The Square of a Given Number 144 is 12.0

10. Wap to reverse the word in a string.

```
import java.io.*;
import java.util.*;
class Reverse {
// reverses individual words of a string
static void reverseWords(String str)
{
   Stack<Character> st=new Stack<Character>();
```

```
    // Traverse given string
    for (int i = 0; i < str.length(); ++i) {
        if (str.charAt(i) != ' ')
            st.push(str.charAt(i));
        else {
            while (st.empty() == false) {
                System.out.print(st.pop());
            }
            System.out.print(" ");
        }
    }
    while (st.empty() == false) {
        System.out.print(st.pop());
    }
}
public static void main(String[] args)
{
   String str = "Hello World";
    reverseWords(str);
  }
}
```

Output:

dlroW olleH

11. Wap to check if given two string is Anagram or not.

```
public class Anagram {
    static void isAnagram(String str1, String str2) {
        String s1 = str1.replaceAll("\\s", "");
        String s2 = str2.replaceAll("\\s", "");
        boolean status = true;
        if (s1.length() != s2.length()) {
            status = false;
```

```
        } else {
            char[] ArrayS1 = s1.toLowerCase().toCharArray();
            char[] ArrayS2 = s2.toLowerCase().toCharArray();
            Arrays.sort(ArrayS1);
            Arrays.sort(ArrayS2);
            status = Arrays.equals(ArrayS1, ArrayS2);
        }
        if (status) {
            System.out.println(s1 + " and " + s2 + " are anagrams");
        } else {
            System.out.println(s1 + " and " + s2 + " are not anagrams");
        }
    }
    public static void main(String[] args) {
        isAnagram("Road", "Roar");
        isAnagram("Website", "Android");
    }
}
```

Output:

Road and Roar are anagrams

Website and Android are anagrams

12. Wap to determine if given year is leap year or not.

```
public class LeapYear {
public static void main(String[] args) {
//year we want to check int year = 2004;
//if year is divisible by 4, it is a leap year
if(year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0)) System.out.println("Year " + year + " is a leap year");
else
System.out.println("Year " + year + " is not a leap year");
}
```

```
}
```

Output :

Year 2004 is a leap year

13. Wap to print number 1 to 1000.

```
public class PrintNumbers
{
   public static void main(String[] args)
   {
      for(int i=1; i<=1000; i++)
      {
         System.out.println(i);
      }
   }
}
```

14. Wap to calculate sum of first 20 natural number.

```
public class SumNumbers
{
   public static void main(String[] args)
   {
      int sum = 0;
      for(int i=1; i<=20; i++)
      {
         sum += i;
      }
      System.out.println("Sum: " + sum);
   }
}
```

15. Wap to Decimal to Binary.

```
import java.util.Scanner;
public class DecimalToBinary
{
```

```
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Decimal Number : ");
        int inputNumber = scanner.nextInt();
        int copyOfInputNumber = inputNumber;
        String binary = "";
        int rem = 0;
        while (inputNumber > 0)
        {
            rem = inputNumber % 2;
            binary =  rem + binary;
            inputNumber = inputNumber/2;
        }
        System.out.println("Binary of "+copyOfInputNumber+" is "+binary);
    }
}
```

Output :

```
Enter Decimal Number :
30
Binary of 30 is 11110
```

16. Wap to convert Decimal to Octal.

```
import java.util.Scanner;
public class DecimalToOctal
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter Decimal Number : ");
        int inputNumber = scanner.nextInt();
        int copyOfInputNumber = inputNumber;
```

```
        String octal = "";
        int rem = 0;
        while (inputNumber > 0)
        {
           rem = inputNumber%8;
           octal =  rem + octal;
           inputNumber = inputNumber/8;
        }
        System.out.println("Octal of "+copyOfInputNumber+" is "+octal);
    }
}
```

Output :

```
Enter Decimal Number :
250
Octal of 250 is 372
```

17. Wap to convert Decimal to HexaDecimal.

```
import java.util.Scanner;
public class DecimalToHexaDecimal
{
    public static void main(String[] args)
    {
        Scanner scanner= new Scanner(System.in);
        System.out.println("Enter Decimal Number : ");
        int inputNumber = scanner.nextInt();
        int copyOfInputNumber = inputNumber;
        String hexa = "";
        char hexaDecimals[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
        int rem = 0;
        while (inputNumber > 0)
        {
           rem = inputNumber%16;
```

```
            hexa =  hexaDecimals[rem] + hexa;
            inputNumber = inputNumber/16;
        }
        System.out.println("HexaDecimal of "+copyOfInputNumber+" is "+hexa);
    }
}
```

Output :

```
Enter Decimal Number :
1500
HexaDecimal of 1500 is 5DC
```

18. Wap that reads a set of integers, then print the sum of even and odd integers.

```
import java.util.Scanner;
public class ReadIntegers
{
    public static void main(String[] args)
    {
        Scanner console = new Scanner(System.in);
        int number;
        char choice;
        int evenSum = 0;
        int oddSum = 0;
        do
        {
            System.out.print("Enter the number ");
            number = console.nextInt();
            if( number % 2 == 0)
            {
                evenSum += number;
            }
            else
            {
```

```
            oddSum += number;
          }
          System.out.print("Do you want to continue y/n? ");
          choice = console.next().charAt(0);
       }while(choice=='y' || choice == 'Y');
       System.out.println("Sum of even numbers: " + evenSum);
       System.out.println("Sum of odd numbers: " + oddSum);
   }
}
```

19. Wap to convert Arraylist to array.

```
public class MainClass
{
   public static void main(String[] args)
   {
      ArrayList<String> list = new ArrayList<String>();
      list.add("PHP");
      list.add("Python");
      list.add("C");
      list.add("SERVLETS");
      list.add("J2EE");
      System.out.println(list);
      Object[] array = list.toArray();
      for (Object object : array)
      {
         System.out.println(object);
      }
   }
}
```

Output:

PHP

Python

C

Servlet

J2EE

20. Wap to insert an element at particular position in an arraylist.

```
public class MainClass
{
   public static void main(String[] args)
   {
      ArrayList<String> al = new ArrayList<String>();
      al.add("First");
      al.add("Second");
      al.add("Third");
      al.add("Fourth");
      System.out.println(list);
      //Inserting "A" at index 1
      al.add(1, "A");
      //Inserting "B" at index 3
      al.add(3, "B");
      System.out.println(al);
   }
}
```

Output:

[First, A, Second, B, Third, Fourth]

21. Wap to remove an element at the particular positon of an arraylist.

```
public class MainClass
{
   public static void main(String[] args)
   {
      ArrayList<String> al = new ArrayList<String>();
      al.add("A");
      al.add("B");
```

```
        al.add("C");
        al.add("D");
        System.out.println(al);
        //Removing an element from position 2
        al.remove(2);
        System.out.println(al);
        //Removing an element from position 3
        al.remove(3);
        System.out.println(al);
    }
}
```

Output:

[A,B,C,D]

[A,B,D]

[A,B]

22. Wap to retrieve a portion of an arraylist.

```
public class MainClass
{
    public static void main(String[] args)
    {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(1);
        al.add(2);
        al.add(3);
        al.add(4);
        al.add(5);
        al.add(6);
        System.out.println(al);
        //Retrieving a SubList
        List<Integer> subList = al.subList(1, 4);
        System.out.println(subList);    //Output : [2, 3, 4]
```

```
        //Modifying the list
        al.set(2, 0);
        //Changes will be reflected in subList
        System.out.println(subList);    //Output : [2, 0, 4]
        //Modifying the subList
        subList.set(2, 0);
        //Changes will be reflected in list
        System.out.println(al);    //Output : [1, 2, 0, 0, 5, 6]
    }
}
```

Output:

```
[1 2 3 4 5 6]
[2 3 4]
[2 0 4]
[1 2 0 0 5 6]
```

23. Wap to insert more than one element at a particular position in arraylist.

```
public class MainClass
{
    public static void main(String[] args)
    {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(1);
        al.add(2);
        al.add(3);
        al.add(4);
        System.out.println(al);     //Output : [1, 2, 3, 4]
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(5);
        list.add(6);
        list.add(7);
        list.add(8);
```

```
        System.out.println(list);    //Output : [5, 6, 7, 8]
        //Inserting all elements of list at index 2 of al
        list1.addAll(2, list);
        System.out.println(al);    //Output : [1, 2, 5, 6, 7, 8, 3, 4]
    }
}
```

24. Wap to sort an array and search an element inside it.

```
import java.util.Arrays;
public class MainClass {
  public static void main(String args[]) throws Exception {
    int array[] = { 4, 2, -7 , 9, -25, 19, 14, -35 };
    Arrays.sort(array);
    printArray("Sorted array", array);
    int index = Arrays.binarySearch(array, 4);
    System.out.println("Found 4 @ " + index);
  }
  private static void printArray(String message, int array[]) {
    System.out.println(message + ": [length: " + array.length + "]");
    for (int i = 0; i < array.length; i++) {
      if(i != 0) {
        System.out.print(", ");
      }
      System.out.print(array[i]);
    }
    System.out.println();
  }
}
```

Output :-

```
Sorted array: [length: 8]
-35 -25 -7 2 4 9 14 19
Found 4 @ 4
```

25. Wap to sort an array and insert an element inside it.

```
import java.util.Arrays;
public class MainClass {
  public static void main(String args[]) throws Exception {
    int array[] = { 4, 2, -7 , 9, -25, 19, 14, -35 };
    Arrays.sort(array);
    printArray("Sorted array", array);
    int index = Arrays.binarySearch(array, 1);
    System.out.println("Didn't find 1 @ " + index);
    int newIndex = -index - 1;
    array = insertElement(array, 1, newIndex);
    printArray("With 1 added", array);
  }
  private static void printArray(String message, int array[]) {
    System.out.println(message + ": [length: " + array.length + "]");
    for (int i = 0; i < array.length; i++) {
      if (i != 0){
        System.out.print(", ");
      }
      System.out.print(array[i]);
    }
    System.out.println();
  }
  private static int[] insertElement(int original[], int element, int index) {
    int length = original.length;
    int destination[] = new int[length + 1];
    System.arraycopy(original, 0, destination, 0, index);
    destination[index] = element;
    System.arraycopy(original, index, destination, index + 1, length - index);
    return destination;
  }
```

```
}
```

Output:

```
Sorted array: [length: 8]
-35 -25 -7 2 4 9 14 19
Didn't find 1 @ 5
With 1 added: [length: 9]
-35 -25 -7 2 4 1 9 14 19
```

26. Wap to merge two array.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
public class Main {
  public static void main(String args[]) {
    String a[] = { "K", "O", "D" };
    String b[] = { "N", "E", "S", "T" };
    List list = new ArrayList(Arrays.asList(a));
    list.addAll(Arrays.asList(b));
    Object[] c = list.toArray();
    System.out.println(Arrays.toString(c));
  }
}
```

Output:

```
[K, O, D, N, E, S, T]
```

27. Wap to sort an array and search an element inside it.

```
import java.util.Arrays;
public class MainClass {
  public static void main(String args[]) throws Exception {
    int array[] = { 4, 2, -7 , 9, -25, 19, 14, -35 };
    Arrays.sort(array);
    printArray("Sorted array", array);
    int index = Arrays.binarySearch(array, 4);
```

```
    System.out.println("Found 4 @ " + index);
  }
  private static void printArray(String message, int array[]) {
    System.out.println(message + ": [length: " + array.length + "]");
    for (int i = 0; i < array.length; i++) {
      if(i != 0){
        System.out.print(", ");
      }
      System.out.print(array[i]);
    }
    System.out.println();
  }
}
```

Output:

```
Sorted array: [length: 10]
-35 -25 -7 2 4 9 14 19
Found 4 @ 4
```

28. Wap to check if two arrays are equal or not.

```
import java.util.Arrays;
public class Main {
  public static void main(String[] args) throws Exception {
    int[] arry = {2,3,5,7,9,10};
    int[] arry1 = {2,3,5,7,9,10};
    int[] arry2 = {1,2,3,4};
    System.out.println("Is array 1 equal to array 2? " +Arrays.equals(arry, arry1));
    System.out.println("Is array 1 equal to array 3? " +Arrays.equals(arry, arry2));
  }
}
```

Output:

```
Is array 1 equal to array 2? true
Is array 1 equal to array 3? false
```

29. Wap to iterate array-list using for-loop, while loop and advance for loop.

```
import java.util.*;
public class arrayList {
	public static void main(String[] args) {
		ArrayList al = new ArrayList();
		al.add("5");
		al.add("10");
		al.add("15");
		System.out.println(al.size());
		System.out.println("While Loop:");
		Iterator itr = al.iterator();
		while(itr.hasNext()) {
			System.out.println(itr.next());
		}
		System.out.println("Advanced For Loop:");
		for(Object obj : al) {
			System.out.println(obj);
	}
		System.out.println("For Loop:");
		for(int i=0; i<list.size(); i++) {
			System.out.println(al.get(i));
		}
}
}
```

Output:

```
3
While Loop:
5
10
15
Advanced For Loop:
```

5

10

15

For Loop:

5

10

15

30. Wap to find second largest number in an integer in array.

```
public class MainClass
{
   static int secondLargest(int[] input)
   {
      int firstLargest, secondLargest;
      //Checking elements of input array
      if(input[0] > input[1])
      {
         //If first element is greater than second element
         firstLargest = input[0];
         secondLargest = input[1];
      }
      else
      {
         //If second element is greater than first element
         firstLargest = input[1];
         secondLargest = input[0];
      }
      //Checking remaining elements of input array
      for (int i = 2; i < input.length; i++)
      {
         if(input[i] > firstLargest)
         {
```

```
            secondLargest = firstLargest;
            firstLargest = input[i];
        }
        else if (input[i] < firstLargest && input[i] > secondLargest)
        {
            secondLargest = input[i];
        }
      }
      return secondLargest;
   }
   public static void main(String[] args)
   {
      System.out.println(secondLargest(new int[] {25, 12, 10, 33, 4, 90}));
      System.out.println(secondLargest(new int[] {122, 150, 108, 298, 399}));
      System.out.println(secondLargest(new int[] {1202, 1543, 1011, 2584, 5894}));
   }
}
```

Output :

10

122

1202

31. Wap to find missing number in an array.

```
public class MissingNumber
{
   static int sumOfNnumbers(int n)
   {
      int sum = (n * (n+1))/ 2;
      return sum;
   }
   static int sumOfElements(int[] array)
   {
```

```
        int sum = 0;
        for (int i = 0; i < array.length; i++)
        {
            sum = sum + array[i];
        }
        return sum;
    }
    public static void main(String[] args)
    {
        int n = 8;
        int[] a = {1, 5, 2, 3, 7, 6, 8};
        int sumOfNnumbers = sumOfNnumbers(n);
        int sumOfElements = sumOfElements(a);
        int missingNumber = sumOfNnumbers - sumOfElements;
        System.out.println("Missing Number is = "+missingNumber);
    }
}
```

32. Wap to get roman value of a decimal number.

KodNest 9047698

```
import java.util.Scanner;
public class MainClass
{
    public static void main(String[] args)
    {
        String[] romanSymbols = {"I", "IV", "V", "IX", "X", "XL", "L", "XC", "X", "CD", "D", "CM", "M"};
        int[] decimals = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000};
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter The Decimal Number Between 1 and 2999");
        int inputNumber = scanner.nextInt();
        int copyOfInputNumber = inputNumber;
        String roman = "";
```

```
        if (inputNumber >= 1 && inputNumber <= 2999)
        {
            for (int i = 0; i < 13; i++)
            {
                while(inputNumber >= decimals[i])
                {
                    inputNumber = inputNumber - decimals[i];
                    roman = roman + romanSymbols[i];
                }
            }
            System.out.println("Roman Value Of "+copyOfInputNumber+" is : "+roman);
        }
        else
        {
            System.out.println("Not a valid number");
        }
    }
}
```

Output :

```
Enter The Decimal Number Between 1 and 2999
1269
Roman Value Of 1269 is : MCCLXIX
```

33. Wap to move zero to an end of an array.

```
public class SeparateZeros
{
    static void moveZerosToEnd(int inputArray[])
    {
        int counter = 0;
        for (int i = 0; i < inputArray.length; i++)
        {
            if(inputArray[i] != 0)
```

```
        {
            inputArray[counter] = inputArray[i];
            //Incrementing the counter by 1
            counter++;
        }
      }
      //Assigning zero to remaining elements
      while (counter < inputArray.length)
      {
         inputArray[counter] = 0;
         counter++;
      }
      System.out.println(Arrays.toString(inputArray));
   }
   public static void main(String[] args)
   {
      moveZerosToEnd(new int[] {4, 10, 20, 0, 30, 0, 8});
      moveZerosToEnd(new int[] {-9, 0, 12, 24, 0, 15, 0, 22});
   }
}
```

Output :

[4, 10, 20, 30, 8, 0, 0]

[-9, 12, 24, 15, 22, 0, 0]

34. Wap to bring zero to front of an array.

```
public class SeparateZeros
{
   static void moveZerosToFront(int inputArray[])
   {
      int counter = inputArray.length-1;
      for (int i = inputArray.length-1; i >= 0; i--)
      {
```

```
            if(inputArray[i] != 0)
            {
                inputArray[counter] = inputArray[i];
                counter--;
            }
        }
        while (counter >= 0)
        {
            inputArray[counter] = 0;
            counter--;
        }
        System.out.println(Arrays.toString(inputArray));
    }
    public static void main(String[] args)
    {
        moveZerosToFront(new int[] {4, 10, 20, 0, 30, 0, 8});
        moveZerosToFront(new int[] {-9, 0, 12, 24, 0, 15, 0, 22});
    }
}
```

Output:

[0, 0, 4, 10, 20, 30, 8]

[0, 0, -9, 12, 24, 15, 22]

35. Wap to count occurances of character in string.

```
class CharCountInString
{
    static void characterCount(String inputString)
    {
        HashMap<Character, Integer> charCountMap = new HashMap<Character, Integer>();
        //Converting given string to char array
        char[] strArray = inputString.toCharArray();
        for (char c : strArray)
```

```
        {
            if(charCountMap.containsKey(c))
            {
                charCountMap.put(c, charCountMap.get(c)+1);
            }
            else
            {
                charCountMap.put(c, 1);
            }
        }
        System.out.println(charCountMap);
    }
    public static void main(String[] args)
    {
       characterCount("Internationalization");
    }
}
```

Output :

{I=1, n=4, t=2, e=1, r=1, a=3, i=3, z=1, o=2}

36. How to find all permutation of a given string

```
public class StringPermutations {
    public static void main(String args[]) {
        permutation("123");
    }
   public static void permutation(String input){
         permutation("", input);
   }
  private static void permutation(String perm, String word) {
        if (word.isEmpty()) {
            System.err.println(perm + word);
        } else {
```

```
        for (int i = 0; i < word.length(); i++) {
            permutation(perm + word.charAt(i), word.substring(0, i)
                        + word.substring(i + 1, word.length()));
        }
      }
   }
}
```

Output:

123

132

213

231

312

321

37. Wap to remove all white space from a string.

1) Using replaceAll() Method.

In the first method, we use replaceAll() method of String class to remove all white spaces (including tab also) from a string. This is the one of the easiest method to remove all white spaces from a string. This method takes two parameters. One is the string to be replaced and another one is the string to be replaced with. We pass the string “\s” to be replaced with an empty string “”.

2) Without Using replaceAll() Method.

In the second method, we remove all white spaces (including tab also) from a string without using replaceAll() method. First we convert the given string to char array and then we traverse this array to find white spaces. We append the characters which are not the white spaces to StringBuffer object.

```
class RemoveWhiteSpaces
{
   public static void main(String[] args)
   {
      String str = " Core Java jsp servlet  jdbc struts hibernate spring ";
```

```
        //1. Using replaceAll() Method
        String strWithoutSpace = str.replaceAll("\\s", "");
        System.out.println(strWithoutSpace);
Output : CoreJavajspservletsjdbcstrutshibernatespring
        //2. Without Using replaceAll() Method
        char[] strArray = str.toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < strArray.length; i++)
        {
           if( (strArray[i] != ' ') && (strArray[i] != '\t') )
           {
              sb.append(strArray[i]);
           }
        }
        System.out.println(sb);
    }
}
//Output : CoreJavajspservletsjdbcstrutshibernatespring
```

38. Wap to convert String To Integer in Java.

There are two methods available in java to convert string to integer. One is Integer.parseInt() method and another one is Integer.valueOf() method. Both these methods are static methods of java.lang.Integer class. Both these methods throw NumberFormatException if input string is not a valid integer. The main difference between Integer.parseInt() and Integer.valueOf() method is that parseInt() method returns primitive int where as valueOf() method returns java.lang.Integer object.

```
public class Demo1
{
   public static void main(String[] args)
   {
      String s = "2015";
      int i = Integer.parseInt(s);
      System.out.println(i);
```

```
    }
}
//Output : 2015
```

Example2:

```
public class Demo2
{
    public static void main(String[] args)
    {
        String s = "2015";
        int i = Integer.valueOf(s);
        System.out.println(i);
    }
}
//Output : 2015
```

39. Wap to convert Integer to String in Java.

You are also often need to do the reverse conversion i.e converting from integer to string. Java provides couple of methods to do that also. one is Integer.toString() method and another one is String.valueOf() method. Both these methods return string representation of the given integer.

```
public class Demo1
{
    public static void main(String[] args)
    {
        int i = 2015;
        String s = Integer.toString(i);
        System.out.println(s);
    }
}
//Output : 2015
```

Example2:

```
public class Demo2
{
```

```
    public static void main(String[] args)
    {
        int i = 2015;
        String s = String.valueOf(i);
        System.out.println(s);
    }
}
//Output : 2015
```

40. Wap to reverse a given string with preserving the position of spaces

KodNest reverse

```
public class MainClass1
{
    static void reverseString(String inputString)
    {
        //Converting inputString to char array 'inputStringArray'
        char[] inputStringArray = inputString.toCharArray();
        //Defining a new char array 'resultArray' with same size as inputStringArray
        char[] resultArray = new char[inputStringArray.length];
        //First for loop :
        //For every space in the 'inputStringArray',
        //we insert spaces in the 'resultArray' at the corresponding positions
        for (int i = 0; i < inputStringArray.length; i++)
        {
            if (inputStringArray[i] == ' ')
            {
                resultArray[i] = ' ';
            }
        }
        //Initializing 'j' with length of resultArray
        int j = resultArray.length-1;
        //Second for loop :
```

```
        //we copy every non-space character of inputStringArray
        //from first to last at 'j' position of resultArray
        for (int i = 0; i < inputStringArray.length; i++)
        {
           if (inputStringArray[i] != ' ')
           {
              //If resultArray already has space at index j then decrementing 'j'
              if(resultArray[j] == ' ')
              {
                 j--;
              }
              resultArray[j] = inputStringArray[i];
              j--;
           }
        }
        System.out.println(inputString+" ---> "+String.valueOf(resultArray));
     }
     public static void main(String[] args)
     {
        reverseString("I Am Not String");
        reverseString("JAVA JSP ANDROID");
      }
}
```

Output:

I Am Not String —> g ni rtS toNmAI

JAVA JSP ANDROID —> DIOR DNA PSJAVAJ

41. Wap to swap first and last character of words in a sentence.

Input : kodnest for freshers

Output :todnesk rof sresherf

Approach:As mentioned in the example we have to replace first and last character of word and keep rest of the alphabets as it is.

First we will create an Char array of given String by using toCharArray() method.

Now we iterate the char array by using for loop.

In for loop, we declare a variable whose value is dependent on i.

Whenever we found an alphabet we increase the value of i and whenever we reach at space, we are going to perform swapping between first and last character of the word which is previous of space.

```
class Demo
{
   static String count(String str)
   {
      // Create an equivalent char array
      // of given string
      char[] ch = str.toCharArray();
      for (int i = 0; i < ch.length; i++) {
         // k stores index of first character
         // and i is going to store index of last
         // character.
         int k = i;
         while (i < ch.length && ch[i] != ' ')
            i++;
         // Swapping
         char temp = ch[k];
         ch[k] = ch[i - 1];
         ch[i - 1] = temp;
         // We assume that there is only one space
         // between two words.
```

```
        }
        return new String(ch);
    }
    public static void main(String[] args)
    {
        String str = "kodnest for freshers";
        System.out.println(count(str));
    }
}
```

42. Wap to replace character in string.

```
public class StringReplace {
    public static void main(String args[]) {
     String word = "World";
     //replacing character in this String
     String replaced = word.replace("W", "K");
     System.out.println("Replacing character in String");
     System.out.println("Original String before replace : " + word);
     System.out.println("Replaced String : " + replaced);
     //replacing substring on String in Java
     String str = "PHP is good programming language";
     replaced = str.replaceAll("PHP", "Java");
     System.out.println("String before replace : " + str);
     System.out.println("String after replace : " + replaced);
     //replacing all space in String with # using regular expression
     replaced = str.replaceFirst("\\s", "#");
     System.out.println("Replacing first match of regex using replaceFirst()");
     System.out.println("Original String before replacement : " + str);
     System.out.println("Final String : " + replaced);
     System.out.println("Replacing all occurrence of substring which match regex");
     replaced = str.replaceAll("\\s", "#");
     System.out.println("ReplaceAll Example : " + replaced);
```

```
    }
}
```

Output:

```
Replacing character in String
Original String before replace : World
Replaced String : Korld
String before replace : PHP is good programming language
String after replace : Java is good programming language
Replacing first match of regex using replaceFirst()
Original String before replacement : PHP is good programming language
Final String : PHP#is good programming language
Replacing all occurrence of substring which match regex
ReplaceAll Example : PHP#is#good#programming#language
```

43. Wap to count number of words in string.

```
public class WordCount {
     static int wordcount(String string)
      {
        int count=0;
          char ch[]= new char[string.length()];
          for(int i=0;i<string.length();i++)
          {
             ch[i]= string.charAt(i);
             if( ((i>0)&&(ch[i]!=' ')&&(ch[i-1]==' ')) || ((ch[0]!=' ')&&(i==0)) )
                count++;
          }
          return count;
      }
     public static void main(String[] args) {
        String string ="Java is a best programming language";
       System.out.println(wordcount(string) + " words.");
   }
```

```
}
```

Output:

6 words.

44. Wap to swap two number without using third variable.

```
import java.util.Scanner;
class SwapTwoNumberWithoutThirdVariable
{
  public static void main(String args[])
  {
    int x, y;
    System.out.println("Enter the number for x and y ");
    Scanner in = new Scanner(System.in);
    x = in.nextInt();
    y = in.nextInt();
    System.out.println("Before Swapping\nx = "+x+"\ny = "+y);
    x = x + y
    y = x - y;
    x = x - y;
    System.out.println("After Swapping without third variable\nx = "+x+"\ny = "+y);
  }
}
```

Output:

Enter the number for x and y

25

49

Before Swapping

x = 25

y = 49

After Swapping without a third variable

x = 49

y = 25

45. Wap to implement Encapsulation in Java.

```
class Loan
{
private int duration; //private variables examples of encapsulation
private String loan;
private String borrower;
private String salary;
//public constructor can break encapsulation instead use factory method
private Loan(int duration, String loan, String borrower, String salary)
{
this.duration = duration;
this.loan = loan;
this.borrower = borrower;
this.salary = salary;
}
// create loan can encapsulate loan creation logic
public Loan createLoan(String loanType)
{
return loan;
}
}
```

46. Wap to achieve encapsulation.

To achieve encapsulation in Java:

• Declare the variables of a class as ‘private’.

• Provide public setter and getter methods to modify and view the variable’s values

```
public class EncapTest
{
private String name;
```

```
private String idNum;
private int age;
public int getAge()
{
return age;
}
public String getName()
{
return name;
}
public String getIdNum()
{
return idNum;
}
public void setAge( int newAge)
{
age = newAge;
}
public void setName(String newName)
{
name = newName;
}
public void setIdNum( String newId)
{
idNum = newId;
}
}
```

47. Wap to reverse a linked list.

```
class LinkList {
   static Node head;
   static class Node {
```

```
    int data;
    Node next;
    Node(int d) {
        data = d;
        next = null;
    }
}
/* Function to reverse the linked list */
Node reverse(Node node) {
    Node prev = null;
    Node current = node;
    Node next = null;
    while (current != null) {
        next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    node = prev;
    return node;
}
// prints content of double linked list
void printList(Node node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
}
public static void main(String[] args) {
    LinkedList list = new LinkedList();
    list.head = new Node(65);
```

```
        list.head.next = new Node(25);
        list.head.next.next = new Node(14);
        list.head.next.next.next = new Node(10);
        System.out.println("Given Linked list");
        list.printList(head);
        head = list.reverse(head);
        System.out.println("");
        System.out.println("Reversed linked list ");
        list.printList(head);
    }
}
```

Output:

Given linked list

65 25 14 10

Reversed Linked list

10 14 25 65

48. Wap to insert a new node at the middle of the singly linked list.

```
public class InsertNodeMiddle {
    class Node{
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public int size;
    public Node head = null;
    public Node tail = null;
    //addNode() this will add a new node to the list
    public void addNode(int data) {
```

```
    Node newNode = new Node(data);
    //Checking if the list is empty
    if(head == null) {
        head = newNode;
        tail = newNode;
    }
    else {
        tail.next = newNode;
        //newNode this will become new end of the list
        tail = newNode;
    }
    //This will count the number of nodes present in the list
    size++;
}
//This function will add the new node at the middle of the list.
public void addInMid(int data){
    Node newNode = new Node(data);
    //Checks if the list is empty
    if(head == null) {
        //If list is empty, both start and end would point to new node
        head = newNode;
        tail = newNode;
    }
    else {
        Node temp, current;
        //Store the mid position of the list
        int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);
        //Node temp will point to head
        temp = head;
        current = null;
        //Traverse through the list till the middle of the list is reached
```

```
        for(int i = 0; i < count; i++) {
            //Node current will point to temp
            current = temp;
            //Node temp will point to node next to it.
            temp = temp.next;
        }
        //current will point to new node
        current.next = newNode;
        //new node will point to temp
        newNode.next = temp;
    }
    size++;
}
//display() will display all the nodes present in the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
        //Prints each node by incrementing pointer
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    InsertNodeMiddle inm = new InsertNodeMiddle();
    //Adds data to the list
```

```
        inm.addNode(10);
        inm.addNode(20);
        System.out.println("Initial list: ");
        inm.display();
        //Inserting node '30' in the middle
        inm.addInMid(30);
        System.out.println( "Updated List: ");
        inm.display();
        //Inserting node '4' in the middle
        inm.addInMid(40);
        System.out.println("Updated List: ");
        inm.display();
    }
}
```

Output:

Initial list:

10 20

Updated List:

10 30 20

Updated List:

10 30 40 20

49. Wap to insert new node at the start of the singly linked list.

```
public class InsertNodeStart {
    class Node{
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
```

```
//Represent the origin and destination of the singly linked list
public Node head = null;
public Node tail = null;
//addAtStart() will add a new node to the beginning of the list
public void addAtStart(int data) {
   //Create a new node
   Node newNode = new Node(data);
   //Checks if the list is empty
   if(head == null) {
      //If list is empty, both origin and destination will point to new node
      head = newNode;
      tail = newNode;
   }
   else {
      //Node temp will point to origin
      Node temp = head;
      //newNode will become new origin of the list
      head = newNode;
      //Node temp(previous origin) will be added after new origin
      head.next = temp;
   }
}
//display() will display all the nodes present in the list
public void display() {
   //Node current will point to origin
   Node current = head;
   if(head == null) {
      System.out.println("List is empty");
      return;
   }
   System.out.println("Adding nodes at the beginning of the list: ");
```

```
        while(current != null) {
            //Prints each node by incrementing pointer
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        InsertNodeStart ins = new InsertNodeStart();
        //Adding 10 to the list
        ins.addAtStart(10);
        ins.display();
        //Adding 20 to the list
        ins.addAtStart(20);
        ins.display();
        //Adding 30 to the list
        ins.addAtStart(30);
        ins.display();
        //Adding 40 to the list
        ins.addAtStart(40);
        ins.display();
    }
}
```

Output:

```
Adding nodes at the beginning of the list:
10
Adding nodes at the beginning of the list:
20 10
Adding nodes at the beginning of the list:
30 20 10
Adding nodes at the beginning of the list:
```

40 30 20 10

50. Wap to insert new node at the end of the singly linked list.

```
public class InsertNodeEnd {
   class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   //Represent the origin and destination of the singly linked list
   public Node head = null;
   public Node tail = null;
   //addAtEnd() will add a new node to the destination of the list
   public void addAtEnd(int data) {
      //Create a new node
      Node newNode = new Node(data);
      //Checks if the list is empty
      if(head == null) {
         //If list is empty, both origin and destionation will point to new node
         head = newNode;
         tail = newNode;
      }
      else {
         //newNode will be added after end such that end's next will point to newNode
         tail.next = newNode;
         //newNode will become new destination of the list
         tail = newNode;
      }
   }
```

```
//display() will display all the nodes present in the list
public void display() {
   //Node current will point to origin
   Node current = head;
   if(head == null) {
      System.out.println("List is empty");
      return;
   }
   System.out.println("Adding new node at the end of the list: ");
   while(current != null) {
      //Prints each node by incrementing pointer
      System.out.print(current.data + " ");
      current = current.next;
   }
   System.out.println();
}
public static void main(String[] args) {
   InsertNodeEnd ine = new InsertNodeEnd();
   //Adding 10 to the list
   ine.addAtEnd(10);
   ine.display();
   //Adding 20 to the list
   ine.addAtEnd(20);
   ine.display();
   //Adding 30 to the list
   ine.addAtEnd(30);
   ine.display();
   //Adding 40 to the list
   ine.addAtEnd(40);
   ine.display();
}
```

```
}
```

Output:

```
Adding new node at the end of the list:
10
Adding new node at the end of the list:
10 20
Adding new node at the end of the list:
10 20 30
Adding new node at the end of the list:
10 20 30 40
```

51. Wap to delete a node in the middle of the singly linked list.

```
public class deleteNodeMiddle{
class Node{
int data;
Node next;
public Node(int data)
{
this.data = data;
this.next = null;
}
}
//Represent the origin and destination of the singly linked list
public Node head = null;
public Node tail = null;
public int size;
// New node will be added
 public void addNode(int data) {
//Create a new node
 Node newNode = new Node(data);
//Checking if the list is empty
if(head == null) {
```

```
 //If list is empty, both start and end will point to new node
head = newNode;
tail = newNode;
}
else {
//newNode will be added after end such that end's next will point to newNode
tail.next = newNode;
//newNode will become new end of the list
tail = newNode;
}
size++;
}
//deleteFromMid()  this will delete a node from the middle of the list
void deleteFromMid() {
Node temp, current;
//Checking if the list is empty
if(head == null) {
System.out.println("List is empty");
return;
}
else {
//Store the mid position of the list
 int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);
//Checks whether the start is equal to the end or not, if yes then the list has only one node.
if( head != tail ) {
 //Initially, temp will point to start
temp = head;
current = null;
//Current will point to node previous to temp
 //If temp is pointing to node 2 then current will point to node 1.
for(int i = 0; i < count-1; i++){
```

```
current = temp;
temp = temp.next;
}
if(current != null) {
 //temp is the middle that needs to be removed.
 //So, current node will point to node next to temp by skipping temp.
 current.next = temp.next;
//Delete temp
 temp = null;
 }
 //If current points to NULL then, origin and destination will point to node next to temp.
 else {
head = tail = temp.next;
//Delete temp
temp = null;
}
}
//If the list contains only one element
 //then it will remove it and both start and end will point to NULL
else {
head = tail = null;
}
}
size--;
}
//display() will display all the nodes present in the list
public void display() {
//Node current will point to head
Node current = head;
if(head == null) {
System.out.println("List is empty");
```

```
 return;
        }
        while(current != null) {
           //Prints each node by incrementing pointer
           System.out.print(current.data + " ");
           current = current.next;
        }
        System.out.println();
    }
public static void main(String[] args) {
        deleteNodeMiddle dnm = new deleteNodeMiddle();
        //Adds data to the list
        dnm.addNode(10);
        dnm.addNode(20);
        dnm.addNode(30);
        dnm.addNode(40);
        //Printing original list
        System.out.println("Initial List: ");
        dnm.display();
        while(dnm.head != null) {
           dnm.deleteFromMid();
           //Printing updated list
           System.out.println("Updated List: ");
           dnm.display();
        }
    }
}
```

Output:

Initial List:

10 20 30 40

Updated List:

10 30 40

Updated List:

10 40

Updated List:

40

Updated List:

List is empty

52. Wap to delete a node at the start of the singly linked list.

```
public class DeleteNodeStart {
   class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   //Represent the start and end of the singly linked list
   public Node head = null;
   public Node tail = null;
   //addNode() this will add a new node to the list
   public void addNode(int data) {
      //Create a new node
      Node newNode = new Node(data);
      //Checking if the list is empty
      if(head == null) {
         //If list is empty, both start and end will point to new node
         head = newNode;
         tail = newNode;
      }
      else {
```

```
        //newNode will be added after end such that end's next will point to newNode
        tail.next = newNode;
        //newNode will become new end of the list
        tail = newNode;
    }
}
//deleteFromStart() this will delete a node from the start of the list
public void deleteFromStart() {
    //Checking if the list is empty
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    else {
        //Checking whether the list contains only one node
        //If not, the start will point to next node in the list and end will point to the new head.
        if(head != tail) {
            head = head.next;
        }
        //If the list contains only one node
        //then, it will remove it and both start and end will point to null
        else {
            head = tail = null;
        }
    }
}
//display() this will display all the nodes present in the list
public void display() {
    //Node current will point to start
    Node current = head;
    if(head == null) {
```

```
            System.out.println("List is empty");
            return;
        }
        while(current != null) {
            //Printing each node by incrementing pointer
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        DeleteNodeStart dns = new DeleteNodeStart();
        //Adding data to the list
        dns.addNode(10);
        dns.addNode(20);
        dns.addNode(30);
        dns.addNode(40);
        //Printing initial list
        System.out.println("Initial List: ");
        dns.display();
        while(dns.head != null) {
            dns.deleteFromStart();
            //Printing updated list
            System.out.println("Updated List: ");
            dns.display();
        }
    }
}
```

Output:

```
Initial List:
10 20 30 40
```

Updated List:

20 30 40

Updated List:

30 40

Updated List:

40

Updated List:

List is empty

53. Wap to remove a node from the end of the singly linked list.

```
public class DeleteNodeEnd {
   class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   //Represent the start and end of the singly linked list
   public Node head = null;
   public Node tail = null;
   //addNode() this will add a new node to the list
   public void addNode(int data) {
      //Create a new node
      Node newNode = new Node(data);
      //Checking if the list is empty
      if(head == null) {
         //If list is empty, both start and end will point to new node
         head = newNode;
         tail = newNode;
      }
```

```
        else {
            //newNode will be added after end such that end's next will point to newNode
            tail.next = newNode;
            //newNode will become new end of the list
            tail = newNode;
        }
    }
    //deleteFromEnd() thos will delete a node from end of the list
    public void deleteFromEnd() {
        //Checking if the list is empty
        if(head == null) {
            System.out.println("List is empty");
            return;
        }
        else {
            //Checking whether the list contains only one element
            if(head != tail ) {
                Node current = head;
                //Looping through the list till the second last element such that current.next is pointing to
end
                while(current.next != tail) {
                    current = current.next;
                }
                //Second last element will become new end of the list
                tail = current;
                tail.next = null;
            }
            //If the list contains only one element
            //Then it will remove it and both start and end will point to null
            else {
                head = tail = null;
```

```
        }
    }
}
//display() this will display all the nodes present in the list
public void display() {
    //Node current will point to start
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
        //Prints each node by incrementing pointer
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    DeleteNodeEnd dne = new DeleteNodeEnd();
    //Adding data to the list
    dne.addNode(10);
    dne.addNode(20);
    dne.addNode(30);
    dne.addNode(40);
    //Printing initial list
    System.out.println("Initial List: ");
    dne.display();
    while(dne.head != null) {
        dne.deleteFromEnd();
        //Printing updated list
```

```
            System.out.println("Updated List: ");
            dne.display();
        }
    }
}
```

Output:

```
Initial List:
10 20 30 40
Updated List:
10 20 30
Updated List:
10 20
Updated List:
10
Updated List:
List is empty
```

54. Wap to create and display circular Linked List.

```
public class CreateCircularLinkedList {
    public class Node{
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
        }
    }
    public Node head = null;
    public Node tail = null;
    //This function will add the new node at the end of the list.
    public void add(int data){
        //Create new node
        Node newNode = new Node(data);
```

```
    //Checks if the list is empty.
    if(head == null) {
        //If list is empty, both head and tail would point to new node.
        head = newNode;
        tail = newNode;
        newNode.next = head;
    }
    else {
        //tail will point to new node.
        tail.next = newNode;
        //New node will become new tail.
        tail = newNode;
        //Since, it is circular linked list tail will point to head.
        tail.next = head;
    }
}
//Displays all the nodes in the list
public void display() {
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
    }
    else {
        System.out.println("Circular linked list: ");
        do{
            //Prints each node by incrementing pointer.
            System.out.print(" "+ current.data);
            current = current.next;
        }while(current != head);
        System.out.println();
    }
```

```
    }
    public static void main(String[] args) {
        CreateCircularLinkedList ccll = new CreateCircularLinkedList();
        //Adds data to the list
        ccll.add(10);
        ccll.add(20);
        ccll.add(30);
        ccll.add(40);
        //Displays all the nodes present in the list
        ccll.display();
    }
}
```

Output:

Circular linked list:

10 20 30 40

55. Wap to generate Circular Linked List of N node and count nodes.

```
public class CountCircularNodes {
        public class Node{
            int data;
            Node next;
            public Node(int data) {
                this.data = data;
            }
        }
        public int count;
        //Declaring head and tail pointer as null.
        public Node head = null;
        public Node tail = null;
        //This function will add the new node at the end of the list.
        public void add(int data){
            //Create new node
```

```
    Node newNode = new Node(data);
    //Checks if the list is empty.
    if(head == null) {
        //If list is empty, both head and tail would point to new node.
        head = newNode;
        tail = newNode;
        newNode.next = head;
    }
    else {
        //tail will point to new node.
        tail.next = newNode;
        //New node will become new tail.
        tail = newNode;
        //Since, it is circular linked list tail will point to head.
        tail.next = head;
    }
}
//This function will count the nodes of circular linked list
public void countNodes() {
    Node current = head;
    do{
        //Increment the count variable by 1 for each node
        count++;
        current = current.next;
    }while(current != head);
    System.out.println("Count of nodes present in circular linked list: "+count);
}
public static void main(String[] args) {
    CountCircularNodes ccn = new CountCircularNodes();
    ccn.add(10);
    ccn.add(20);
```

```
        ccn.add(40);
        ccn.add(10);
        ccn.add(20);
        ccn.add(30);
        //Counts the number of nodes present in the list
        ccn.countNodes();
    }
}
```

Output:

```
Count of nodes present in circular linked list: 6
```

56. Wap to find minimum and maximum node value in Circular Linked list.

```
public class MinMaxNode {
   public class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
   public Node head = null;
   public Node tail = null;
   //This function will add the new node at the end of the list.
   public void add(int data){
      //Create new node
      Node newNode = new Node(data);
      //Checks if the list is empty.
      if(head == null) {
          //If list is empty, both head and tail would point to new node.
         head = newNode;
         tail = newNode;
         newNode.next = head;
```

```
    }
    else {
        //tail will point to new node.
        tail.next = newNode;
        //New node will become new tail.
        tail = newNode;
        //Since, it is circular linked list tail will points to head.
        tail.next = head;
    }
}
//Finds out the minimum value node in the list
public void minNode() {
    Node current = head;
    //Initializing min to initial node data
    int min = head.data;
    if(head == null) {
        System.out.println("List is empty");
    }
    else {
         do{
            //If current node's data is smaller than min
            //Then replace value of min with current node's data
            if(min > current.data) {
                min = current.data;
            }
            current= current.next;
        }while(current != head);
        System.out.println("Minimum value node in the list: "+ min);
    }
}
//Finds out the maximum value node in the list
```

```
public void maxNode() {
   Node current = head;
   //Initializing max to initial node data
   int max = head.data;
   if(head == null) {
      System.out.println("List is empty");
   }
   else {
       do{
          //If current node's data is greater than max
          //Then replace value of max with current node's data
          if(max < current.data) {
             max = current.data;
          }
          current= current.next;
         }while(current != head);
      System.out.println("Maximum value node in the list: "+ max);
   }
}
public static void main(String[] args) {
   MinMaxNode mmn = new MinMaxNode();
   //Adds data to the list
   mmn.add(12);
   mmn.add(24);
   mmn.add(8);
   mmn.add(16);
   //Prints the minimum value node in the list
   mmn.minNode();
   //Prints the maximum value node in the list
   mmn.maxNode();
}
```

```
}
```

Output:

Minimum value node in the list: 8

Maximum value node in the list: 24

57. Wap to insert node at the beginning of the Circular Linked list.

```
public class InsertNodeStart {
    public class Node{
       int data;
       Node next;
       public Node(int data) {
          this.data = data;
       }
    }
    //Declaring start and end pointer as null.
    public Node head = null;
    public Node tail = null;
    //This function will add the new node at the end of the list.
    public void addAtStart(int data){
       //Create new node
       Node newNode = new Node(data);
       //Checking if the list is empty.
       if(head == null) {
           //If list is empty, both start and end would point to new node.
          head = newNode;
          tail = newNode;
          newNode.next = head;
       }
       else {
          //Store data into temporary node
          Node temp = head;
          //New node will point to temp as next node
```

```
        newNode.next = temp;
        //New node will be the start node
        head = newNode;
        //Since, it is circular linked list tail will point to head.
        tail.next = head;
    }
}
//Displays all the nodes in the list
public void display() {
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
    }
    else {
        System.out.println("Adding nodes at the beginning of the list: ");
         do{
            //Printing each node by incrementing pointer.
            System.out.print(" "+ current.data);
            current = current.next;
        }while(current != head);
        System.out.println();
    }
}
public static void main(String[] args) {
    InsertNodeStart ins = new InsertNodeStart();
    //Adding 10 to the list
    ins.addAtStart(10);
    ins.display();
    //Adding 20 to the list
    ins.addAtStart(2);
    ins.display();
```

```
        //Adding 30 to the list
        ins.addAtStart(3);
        ins.display();
        //Adding 40 to the list
        ins.addAtStart(4);
        ins.display();
    }
}
```

Output:

```
Adding nodes at the beginning of the list:
10
Adding nodes at the beginning of the list:
20 10
Adding nodes at the beginning of the list:
30 20 10
Adding nodes at the beginning of the list:
40 30 20 10
```

58. Wap to insert a node at the end of the Circular linked list.

```
public class InsertNodeEnd {
    public class Node{
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
        }
    }
    //Declaring start and end pointer as null.
    public Node head = null;
    public Node tail = null;
    //This function will add the new node at the end of the list.
    public void addAtEnd(int data){
```

```
    //Create new node
    Node newNode = new Node(data);
    //Checking if the list is empty.
    if(head == null) {
        //If list is empty, both start and end would point to new node.
        head = newNode;
        tail = newNode;
        newNode.next = head;
    }
    else {
        //end will point to new node.
        tail.next = newNode;
        //New node will become new end.
        tail = newNode;
        //Since, it is circular linked list end will points to start.
        tail.next = head;
    }
}
//Displays all the nodes in the list
public void display() {
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
    }
    else {
        System.out.println("Adding nodes at the end of the list: ");
        do{
            //Prints each node by incrementing pointer.
            System.out.print(" "+ current.data);
            current = current.next;
        }while(current != head);
```

```
            System.out.println();
        }
    }
    public static void main(String[] args) {
        InsertNodeEnd ine = new InsertNodeEnd();
        //Adding 10 to the list
        ine.addAtEnd(10);
        ine.display();
        //Adding 20 to the list
        ine.addAtEnd(20);
        ine.display();
        //Adding 30 to the list
        ine.addAtEnd(30);
        ine.display();
        //Adding 40 to the list
        ine.addAtEnd(40);
        ine.display();
    }
}
```

Output:

```
Adding nodes at the end of the list:
10
Adding nodes at the end of the list:
10 20
Adding nodes at the end of the list:
10 20 30
Adding nodes at the end of the list:
10 20 30 40
```

59. Wap to insert a node at middle in Circular linked list.

```
public class InsertNodeMiddle {
    public class Node{
```

```
    int data;
    Node next;
    public Node(int data) {
        this.data = data;
    }
}
public int size;
//Declaring start and end pointer as null.
public Node head = null;
public Node tail = null;
//This function will add the new node to the list.
public void add(int data){
    //Create new node
    Node newNode = new Node(data);
    //Checks if the list is empty.
    if(head == null) {
         //If list is empty, both start and end would point to new node.
        head = newNode;
        tail = newNode;
        newNode.next = head;
    }
    else {
        //tail will point to new node.
        tail.next = newNode;
        //New node will become new tail.
        tail = newNode;
        //Since, it is circular linked list tail will points to head.
        tail.next = head;
    }
    //Size will count the number of element in the list
    size++;
```

```
}
//This function will add the new node at the middle of the list.
public void addInMid(int data){
   Node newNode = new Node(data);
   //Checks if the list is empty.
   if(head == null){
      //If list is empty, both head and tail would point to new node.
      head = newNode;
      tail = newNode;
      newNode.next = head;
   }
   else{
      Node temp,current;
      //Store the mid-point of the list
      int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);
      //temp will point to head
      temp = head;
      current= null;
      for(int i = 0; i < count; i++){
         //Current will point to node previous to temp.
         current = temp;
         //Traverse through the list till the middle of the list is reached
         temp = temp.next;
      }
      //current will point to new node
      current.next = newNode;
      //new node will point to temp
      newNode.next = temp;
   }
   size++;
}
```

```
//Displays all the nodes in the list
public void display() {
   Node current = head;
   if(head == null) {
      System.out.println("List is empty");
   }
   else {
       do{
         //Prints each node by incrementing pointer.
         System.out.print(" "+ current.data);
         current = current.next;
      }while(current != head);
      System.out.println();
   }
}
public static void main(String[] args) {
   InsertNodeMiddle inm = new InsertNodeMiddle();
   //Adds data to the list
   inm.add(10);
   inm.add(20);
   inm.add(30);
   inm.add(40);
   System.out.println("Initial  list: ");
   inm.display();
   //Inserting node '50' in the middle
   inm.addInMid(50);
   System.out.println( "Updated List: ");
   inm.display();
   //Inserting node '6' in the middle
   inm.addInMid(60);
   System.out.println("Updated List: ");
```

```
            inm.display();
        }
}
```

Output:

Initial list:

10 20 30 40

Updated List:

10 20 50 30 40

Updated List:

10 20 50 60 30 40

60. Wap to remove node at the beginning of the Circular Linked list.

```
public class DeleteNodeStart {
    public class Node{
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
        }
    }
    //Declaring start and end pointer as null.
    public Node head = null;
    public Node tail = null;
    public void add(int data){
        Node newNode = new Node(data);
        if(head == null) {
            head = newNode;
            tail = newNode;
            newNode.next = head;
        }
        else {
            tail.next = newNode;
```

```
        tail = newNode;
        tail.next = head;
    }
}
//Deletes node from the start of the list
public void deleteStart() {
    //Checks whether list is empty
    if(head == null) {
        return;
    }
    else {
        if(head != tail ) {
            head = head.next;
            tail.next = head;
        }
        else {
            head = tail = null;
        }
    }
}
//Displays all the nodes in the list
public void display() {
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
    }
    else {
         do{
            //Prints each node by incrementing pointer.
            System.out.print(" "+ current.data);
            current = current.next;
```

```
            }while(current != head);
            System.out.println();
        }
    }
    public static void main(String[] args) {
        DeleteNodeStart dns = new DeleteStart();
        //Adds data to the list
        dns.add(10);
        dns.add(20);
        dns.add(30);
        dns.add(40);
        //Printing initial  list
        System.out.println("Initial List: ");
        dns.display();
        while(dns.head != null) {
            dns.deleteStart();
            //Printing updated list
            System.out.println("Updated List: ");
            dns.display();
        }
    }
}
```

Output:

```
Initial List:
 10 20 30 40
Updated List:
 20 30 40
Updated List:
 30 40
Updated List:
 40
```

Updated List:

List is empty

61. Wap to remove node at the end of the Circular Linked list.

```
public class DeleteNodeEnd {
   public class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
   //Declaring start and end pointer as null.
   public Node head = null;
   public Node tail = null;
   //This function will add the new node at the end of the list.
   public void add(int data){
      //Create new node
      Node newNode = new Node(data);
      //Checks if the list is empty.
      if(head == null) {
          //If list is empty, both head and tail would point to new node.
         head = newNode;
         tail = newNode;
         newNode.next = head;
      }
      else {
         //tail will point to new node.
         tail.next = newNode;
         //New node will become new tail.
         tail = newNode;
         //Since, it is circular linked list tail will point to head.
```

```
        tail.next = head;
    }
}
//Deletes node from end of the list
public void deleteEnd() {
    //Checks whether list is empty
    if(head == null) {
        return;
    }
    else {
        //Checks whether contain only one element
        if(head != tail ) {
            Node current = head;
            //Loop will iterate till the second last element as current.next is pointing to tail
            while(current.next != tail) {
                current = current.next;
            }
            //Second last element will be new tail
            tail = current;
            //Tail will point to head as it is a circular linked list
            tail.next = head;
        }
        //If the list contains only one element
        //Then it will remove it and both head and tail will point to null
        else {
            head = tail = null;
        }
    }
}
//Displays all the nodes in the list
public void display() {
```

```
        Node current = head;
        if(head == null) {
            System.out.println("List is empty");
        }
        else {
            do{
                //Prints each node by incrementing pointer.
                System.out.print(" "+ current.data);
                current = current.next;
            }while(current != head);
            System.out.println();
        }
    }
    public static void main(String[] args) {
        DeleteNodeEnd dne = new DeleteNodeEnd();
        //Adds data to the list
        dne.add(10);
        dne.add(20);
        dne.add(30);
        dne.add(40);
        //Printing Initial list
        System.out.println("Initial List: ");
        dne.display();
        while(dne.head != null) {
            dne.deleteEnd();
            //Printing updated list
            System.out.println("Updated List: ");
            dne.display();
        }
    }
}
```

Output:

Initial List:

10 20 30 40

Updated List:

10 20 30

Updated List:

10 20

Updated List:

10

Updated List:

List is empty

62. Wap to remove a node from middle of Circular Linked List.

```
public class DeleteNodeMiddle {
   public class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
public int size;
   //Declaring start and end pointer as null.
   public Node head = null;
   public Node tail = null;
   //This function will add the new node at the end of the list.
   public void add(int data){
      //Create new node
      Node newNode = new Node(data);
      //Checks if the list is empty.
      if(head == null) {
          //If list is empty, both head and tail would point to new node.
```

```
        head = newNode;
        tail = newNode;
        newNode.next = head;
    }
    else {
        //tail will point to new node.
        tail.next = newNode;
        //New node will become new tail.
        tail = newNode;
        //Since, it is circular linked list tail will point to head.
        tail.next = head;
    }
    //Counts the number of nodes in list
    size++;
}
//Deletes node from the middle of the list
public void deleteMid() {
    Node current, temp;
    //Checks whether list is empty
    if(head == null) {
        return;
    }
    else {
        //Store the mid position of the list
        int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);
        //Checks whether head is equal to tail or not, if yes then list has only one node.
        if( head != tail ) {
            //Initially temp will point to head;
            temp = head;
            current = null;
            //Current will point to node previous to temp
```

```
            //If temp is pointing to node 2 then current will points to node 1.
            for(int i = 0; i < count-1; i++){
               current = temp;
               temp = temp.next;
            }
            if(current != null) {
               //temp is the middle that needs to be removed.
               //So, current node will point to node next to temp by skipping temp.
               current.next = temp.next;
               //Delete temp;
               temp = null;
            }
            //Current points to null then head and tail will point to node next to temp.
            else {
               head = tail = temp.next;
               tail.next = head;
               //Delete temp;
               temp = null;
            }
         }
         //If the list contains only one element
         //then it will remove it and both head and tail will point to null
         else {
            head = tail = null;
         }
      }
      size--;
   }
   //Displays all the nodes in the list
   public void display() {
      Node current = head;
```

```
        if(head == null) {
            System.out.println("List is empty");
        }
        else {
            do{
                //Prints each node by incrementing pointer.
                System.out.print(" "+ current.data);
                current = current.next;
            }while(current != head);
            System.out.println();
        }
    }
    public static void main(String[] args) {
        DeleteNodeMiddle dnm = new DeleteNodeMiddle();
        //Adds data to the list
        dnm.add(10);
        dnm.add(20);
        dnm.add(30);
        dnm.add(40);
        //Printing Initial list
        System.out.println("Initial List: ");
        dnm.display();
        while(dnm.head != null) {
            dnm.deleteMid();
            //Printing updated list
            System.out.println("Updated List: ");
            dnm.display();
        }
    }
}
```

Output:

Initial List:

10 20 30 40

Updated List:

10 30 40

Updated List:

10 40

Updated List:

40

Updated List:

List is empty

63. Wap to implement Linked list as a queue (FIFO).

```
public class LinkedListExamples
{
   public static void main(String[] args)
   {
      LinkedList<Integer> que = new LinkedList<Integer>();
      //adding the elements into the queue
      que.offer(100);
      que.offer(200);
      que.offer(300);
      que.offer(400);
      //Printing the elements of queue
      System.out.println(que);      //Output : [100, 200, 300, 400]
      //Removing the elements from the queue
      System.out.println(que.poll());    //Output : 100
      System.out.println(que.poll());    //Output : 200
   }
}
```

64. Wap to replace an element at a specific position in linked list.

```
public class LinkedListReplace
{
```

```
    public static void main(String[] args)
    {
        LinkedList<String> list = new LinkedList<String>();
        //Adding elements at the end of the list
        list.add("First");
        list.add("Second");
        list.add("Third");
        list.add("Fouth");
        //Printing the elements of list
        System.out.println(list);      //Output : [First, Second, Third, Fourth]
        //Replacing an element at index 2 with "ZERO"
        list.set(2, "ABC");
        System.out.println(list);     //Output : [First, Second, ABC, Fourth]
    }
}
```

65. Wap to create clone of LinkedList.

```
public class LinkedListClone
{
    public static void main(String[] args)
    {
        LinkedList<Integer> linkedList1 = new LinkedList<Integer>();
        //adding the elements to linkedList1
        linkedList1.add(100);
        linkedList1.add(200);
        linkedList1.add(300);
        linkedList1.add(400);
        linkedList1.add(500);
        //Printing the elements of linkedList1
        System.out.println(linkedList1);       //Output : [100, 200, 300, 400, 500]
        //Creating another LinkedList
        LinkedList<Integer> linkedList2 = new LinkedList<Integer>();
```

```
        //Cloning the linkedList1 into linkedList2
        linkedList2 = (LinkedList<Integer>) linkedList1.clone();
        //Printing the elements of linkedList2
        System.out.println(linkedList2);     //Output : [100, 200, 300, 400, 500]
    }
}
```

66. Wap to find the middle Node of a Linked List in single pass.

```
public class LinkedListMiddleNode {
    public static void main(String args[]) {
        //creating LinkedList with 5 elements including head
      LinkList linkList = new LinkList();
      LinkList.Node head = linkList.head();
      linkList.add( new LinkList.Node("1"));
      linkList.add( new LinkList.Node("2"));
      linkList.add( new LinkList.Node("3"));
      linkList.add( new LinkList.Node("4"));
      //finding middle element of LinkedList in single pass
      LinkList.Node current = head;
      int length = 0;
      LinkList.Node middle = head;
      while(current.next() != null){
         length++;
         if(length%2 ==0){
            middle = middle.next();
         }
         current = current.next();
      }
      if(length%2 == 1){
         middle = middle.next();
      }
      System.out.println("length of LinkedList: " + length);
```

```
      System.out.println("middle element of LinkedList : "                    + middle);
    }
}
class LinkList{
    private Node head;
    private Node tail;
    public LinkList(){
        this.head = new Node("head");
        tail = head;
    }
    public Node head(){
        return head;
    }
    public void add(Node node){
        tail.next = node;
        tail = node;
    }
    public static class Node{
        private Node next;
        private String data;
        public Node(String data){
            this.data = data;
        }
        public String data() {
            return data;
        }
        public void setData(String data) {
            this.data = data;
        }
        public Node next() {
            return next;
```

```
    }
    public void setNext(Node next) {
        this.next = next;
    }
    public String toString(){
        return this.data;
    }
  }
}
```

Output:

```
length of LinkedList: 4
middle element of LinkedList: 2
```

67. Wap to convert binary tree to Doubly linked list.

```
public class BinaryTreeDoubly {
  public static class Node{
    int data;
    Node left;
    Node right;
    public Node(int data) {
      this.data = data;
      this.left = null;
      this.right = null;
    }
  }
  //Represent the root of the binary tree
  public Node root;
  Node head, tail = null;
  //convertbtToDLL() will convert the given binary tree to corresponding doubly linked list
  public void convertbtToDLL(Node node) {
    //Checks whether node is null
    if(node == null)
```

```
        return;
    //Convert left subtree to doubly linked list
    convertbtToDLL(node.left);
    //If list is empty, add node as head of the list
    if(head == null) {
        //Both head and tail will point to node
        head = tail = node;
    }
    //Otherwise, add node to the end of the list
    else {
        //node will be added after tail such that tail's right will point to node
        tail.right = node;
        //node's left will point to tail
        node.left = tail;
        //node will become new tail
        tail = node;
    }
    //Convert right subtree to doubly linked list
    convertbtToDLL(node.right);
}
//display() will print out the nodes of the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    System.out.println("Nodes of generated doubly linked list: ");
    while(current != null) {
        //Prints each node by incrementing the pointer.
```

```
            System.out.print(current.data + " ");
            current = current.right;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        BinaryTreeDoubly btd = new BinaryTreeToDLL();
        //Add nodes to the binary tree
        btd.root = new Node(10);
        btd.root.left = new Node(20);
        btd.root.right = new Node(30);
        btd.root.left.left = new Node(40);
        btd.root.left.right = new Node(50);
        btd.root.right.left = new Node(60);
        btd.root.right.right = new Node(70);
        //Converts the given binary tree to doubly linked list
        btd.convertbtToDLL(btd.root);
        //Displays the nodes present in the list
        btd.display();
    }
}
```

Output:

```
Nodes of generated doubly linked list:
40 20 50 10 60 30 70
```

68. Wap to generate a Doubly linked list of n nodes and count numbers of node.

```
public class CountNodes {
    class Node{
        int data;
        Node previous;
        Node next;
        public Node(int data) {
```

```
        this.data = data;
    }
}
Node head, tail = null;
//addNode() this will add a node to the list
public void addNode(int data) {
    //Create a new node
    Node newNode = new Node(data);
    //If list is empty
    if(head == null) {
        head = tail = newNode;
        head.previous = null;
        tail.next = null;
    }
    else {
        //newNode this will be added after tail such that tail's next will point to newNode
        tail.next = newNode;
        newNode.previous = tail;
        tail = newNode;
        tail.next = null;
    }
}
//countNodes() this will count the nodes present in the list
public int countNodes() {
    int counter = 0;
    //Node current will point to start
    Node current = head;
    while(current != null) {
        //Increment the counter by 1 for each node
        counter++;
        current = current.next;
```

```
    }
    return counter;
}
//display() this will print out the elements of the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    System.out.println("Elements of doubly linked list: ");
    while(current != null) {
        //Prints each node by incrementing the pointer.
        System.out.print(current.data + " ");
        current = current.next;
    }
}
public static void main(String[] args) {
    CountNodes cn = new CountList();
    //Add nodes to the list
    cn.addNode(10);
    cn.addNode(20);
    cn.addNode(30);
    cn.addNode(40);
    cn.addNode(50);
    //Displays the nodes present in the list
    cn.display();
    //Counts the nodes present in the given list
    System.out.println("\nCount of nodes present in the list: " + cn.countNodes());
}
```

```
}
```

Output:

Elements of doubly linked list:

10 20 30 40 50

Count of nodes present in the list: 5

70. Wap to get value of maximum and minimum node in Double Linked List.

```
public class MinMaxNode {
   class Node{
      int data;
      Node previous;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
   Node head, tail = null;
   //addNode() this will add a node to the list
   public void addNode(int data) {
         Node newNode = new Node(data);
      //If list is empty
      if(head == null) {
         head = tail = newNode;
         head.previous = null;
         tail.next = null;
      }
      else {
         tail.next = newNode;
         //newNode's previous will point to end
         newNode.previous = tail;
         //newNode will become new end
         tail = newNode;
```

```
        tail.next = null;
    }
}
//MinimumNode() will find out minimum value node in the list
public int minimumNode() {
    //Node current will point to head
    Node current = head;
    int min;
    //Checks if list is empty
    if(head == null) {
        System.out.println("List is empty");
        return 0;
    }
    else {
        //Initially, min will store the value of head's data
        min = head.data;
        while(current != null) {
            //If the value of min is greater than the current's data
            //Then, replace the value of min with current node's data
            if(min > current.data)
                min = current.data;
            current = current.next;
        }
    }
    return min;
}
//MaximumNode() will find out maximum value node in the list
public int maximumNode() {
    //Node current will point to head
    Node current = head;
    int max;
```

```
        //Checks if list is empty
        if(head == null) {
            System.out.println("List is empty");
            return 0;
        }
        else {
            //Initially, max will store the value of head's data
            max = head.data;
            while(current != null) {
                if(current.data > max)
                    max = current.data;
                current = current.next;
            }
        }
        return max;
    }
    public static void main(String[] args) {
        MinMaxNode mmn = new MinMaxNode();
        //Add nodes to the list
        mmn.addNode(15);
        mmn.addNode(27);
        mmn.addNode(49);
        mmn.addNode(11);
        mmn.addNode(21);
        //Prints the minimum value node in the list
        System.out.println("Minimum value node in the list: "+ dList.minimumNode());
        //Prints the maximum value node in the list
        System.out.println("Maximum value node in the list: "+ dList.maximumNode());
    }
}
```

Output:

Minimum value node : 11

Maximum value node : 49

71. Wap to add new node at middle of Double Linked List.

```
public class InsertNodeMiddle {
   class Node{
      int data;
      Node previous;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
   public int size = 0;
   Node head, tail = null;
   //addNode() will add a node to the list
   public void addNode(int data) {
      //Create a new node
      Node newNode = new Node(data);
      //If list is empty
      if(head == null) {
         //Both head and tail will point to newNode
         head = tail = newNode;
         //head's previous will point to null
         head.previous = null;
         //tail's next will point to null, as it is the last node of the list
         tail.next = null;
      }
      else {
         //newNode will be added after tail such that tail's next will point to newNode
         tail.next = newNode;
         //newNode's previous will point to tail
```

```
        newNode.previous = tail;
        //newNode will become new tail
        tail = newNode;
        //As it is last node, tail's next will point to null
        tail.next = null;
    }
    //Size will count the number of nodes present in the list
    size++;
}
//addInMid() will add a node to the middle of the list
public void addInMid(int data) {
   //Create a new node
    Node newNode = new Node(data);
    //If list is empty
    if(head == null) {
        //Both head and tail will point to newNode
        head = tail = newNode;
        //head's previous will point to null
        head.previous = null;
        //tail's next point to null, as it is the last node of the list
        tail.next = null;
    }
    else {
        //current will point to head
        Node current = head, temp = null;
        //Store the mid position of the list
        int mid = (size % 2 == 0) ? (size/2) : ((size+1)/2);
        //Iterate through list till current points to mid position
        for(int i = 1; i < mid; i++){
            current = current.next;
        }
```

```
            //Node temp will point to node next to current
            temp = current.next;
            temp.previous = current;
            //newNode will be added between current and temp
            current.next = newNode;
            newNode.previous = current;
            newNode.next = temp;
            temp.previous = newNode;
        }
        size++;
    }
    //display() will print out the nodes of the list
    public void display() {
        //Node current will point to head
        Node current = head;
        if(head == null) {
            System.out.println("List is empty");
            return;
        }
        while(current != null) {
            //Prints each node by incrementing the pointer.
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        InsertNodeMiddle inm = new InsertNodeMiddle();
        //Add nodes to the list
        inm.addNode(10);
        inm.addNode(20);
```

```
        System.out.println("Initial list: ");
        inm.display();
        //Adding node '30' in the middle
        inm.addInMid(30);
        System.out.println( "Updated List: ");
        inm.display();
        //Adding node '40' in the middle
        inm.addInMid(4);
        System.out.println("Updated List: ");
        inm.display();
        //Adding node '50' in the middle
        inm.addInMid(50);
        System.out.println("Updated List: ");
        inm.display();
    }
}
```

Output:

Initial list:

10 20

Updated List:

10 30 20

Updated List:

10 30 40 20

Updated List:

10 30 50 40 20

72. Wap to add new node at the end of Double Linked List.

```
public class BinarySearchTreeImplementation {
    private BstNode root;
    public boolean isEmpty() {
        return (this.root == null);
    }
```

```
public void insert(Integer data) {
   System.out.print("[input: "+data+"]");
   if(root == null) {
      this.root = new BstNode(data);
      System.out.println(" -> inserted: "+data);
      return;
   }
   insertNode(this.root, data);
   System.out.print(" -> inserted: "+data);
   System.out.println();
}
private BstNode insertNode(BstNode root, Integer data) {
   BstNode tmpNode = null;
   System.out.print(" ->"+root.getData());
   if(root.getData() >= data) {
      System.out.print(" [L]");
      if(root.getLeft() == null) {
         root.setLeft(new BstNode(data));
         return root.getLeft();
      } else {
         tmpNode = root.getLeft();
      }
   } else {
      System.out.print(" [R]");
      if(root.getRight() == null) {
         root.setRight(new BstNode(data));
         return root.getRight();
      } else {
         tmpNode = root.getRight();
      }
   }
```

```
        return insertNode(tmpNode, data);
    }
    public static void main(String a[]) {
        BinarySearchTreeImpl bst = new BinarySearchTreeImpl();
        bst.insert(16);
        bst.insert(23);
        bst.insert(6);
        bst.insert(21);
        bst.insert(8);
        bst.insert(10);
        bst.insert(20);
    }
}
```

73. Wap to remove node from middle of Double Linked List.

```
Class DeleteNodeMiddle {
    class Node{
        int data;
        Node previous;
        Node next;
        public Node(int data) {
            this.data = data;
        }
    }
    public int size = 0;
    //Represent the head and tail of the doubly linked list
    Node head, tail = null;
    //addNode() will add a node to the list
    public void addNode(int data) {
        //Create a new node
        Node newNode = new Node(data);
        //If list is empty
```

```
    if(head == null) {
        //Both head and tail will point to newNode
        head = tail = newNode;
        //head's previous will point to null
        head.previous = null;
        //tail's next will point to null, as it is the last node of the list
        tail.next = null;
    }
    else {
        //newNode will be added after tail such that tail's next will point to newNode
        tail.next = newNode;
        //newNode's previous will point to tail
        newNode.previous = tail;
        //newNode will become new tail
        tail = newNode;
        //As it is last node, tail's next will point to null
        tail.next = null;
    }
    //Size will count the number of nodes present in the list
    size++;
}
//deleteFromMid() will delete a node from middle of the list
public void deleteFromMid() {
    //Checks whether list is empty
    if(head == null) {
        return;
    }
    else {
        //current will point to head
        Node current = head;
        //Store the mid position of the list
```

```
        int mid = (size % 2 == 0) ? (size/2) : ((size+1)/2);
        //Iterate through list till current points to mid position
        for(int i = 1; i < mid; i++){
            current = current.next;
        }
        //If middle node is head of the list
        if(current == head) {
            head = current.next;
        }
        //If middle node is tail of the list
        else if(current == tail) {
            tail = tail.previous;
        }
        else {
            current.previous.next = current.next;
            current.next.previous = current.previous;
        }
        //Delete the middle node
        current = null;
    }
    size--;
}
//display() will print out the nodes of the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
```

```
            //Prints each node by incrementing the pointer.
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        DeleteNodeMiddle dnm = new DeleteNodeMiddle();
        //Add nodes to the list
        dnm.addNode(10);
        dnm.addNode(20);
        dnm.addNode(30);
        dnm.addNode(40);
        dnm.addNode(50);
        //Printing original list
        System.out.println("Initial List: ");
        dnm.display();
        while(dnm.head != null) {
            dnm.deleteFromMid();
            //Printing updated list
            System.out.println("Updated List: ");
            dnm.display();
        }
    }
}
```

Output:

Initial List:

10 20 30 40 50

Updated List:

10 20 40 50

Updated List:

10 40 50

Updated List:

10 50

Updated List:

50

Updated List:

List is empty

74. Wap to remove node at the begining from Double Linked List.

```
public class DeleteNodeStart
{
   class Node{
      int data;
      Node previous;
      Node next;
      public Node(int data) {
         this.data = data;
      }
}
   //Represent the head and tail of the doubly linked list
   Node head, tail = null;
   //addNode() will add a node to the list
   public void addNode(int data) {
      //Create a new node
      Node newNode = new Node(data);
      //If list is empty
      if(head == null) {
         //Both head and tail will point to newNode
         head = tail = newNode;
         //head's previous will point to null
         head.previous = null;
         //tail's next will point to null, as it is the last node of the list
```

```
        tail.next = null;
    }
    else {
        //newNode will be added after tail such that tail's next will point to newNode
        tail.next = newNode;
        //newNode's previous will point to tail
        newNode.previous = tail;
        //newNode will become new tail
        tail = newNode;
        //As it is last node, tail's next will point to null
        tail.next = null;
    }
}
//deleteFromStart() will delete a node from the beginning of the list
public void deleteFromStart() {
    //Checks whether list is empty
    if(head == null) {
        return;
    }
    else {
        //Checks whether the list contains only one element
        if(head != tail) {
            //head will point to next node in the list
            head = head.next;
            //Previous node to current head will be made null
            head.previous = null;
        }
        //If the list contains only one element
        //then, it will remove node and now both head and tail will point to null
        else {
            head = tail = null;
```

```
        }
    }
}
//display() will print out the nodes of the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
        //Prints each node by incrementing the pointer.
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    DeleteNodeStart dns = new DeleteNodeStart();
    //Add nodes to the list
    dns.addNode(10);
    dns.addNode(20);
    dns.addNode(30);
    dns.addNode(40);
    dns.addNode(50);
    //Printing original list
    System.out.println("Initial  List: ");
    dns.display();
    while(dns.head != null) {
        dns.deleteFromStart();
```

```
        //Printing updated list
        System.out.println("Updated List: ");
        dns.display();
      }
   }
}
```

Output:

```
Initial List:
10 20 30 40 50
Updated List:
20 30 40 50
Updated List:
30 40 50
Updated List:
40 50
Updated List:
50
Updated List:
List is empty
```

75. Wap to remove node from the end of Doubly Linked List.

```
public class DeleteNodeEnd {
   class Node{
      int data;
      Node previous;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
   //Represent the head and tail of the doubly linked list
   Node head, tail = null;
```

```
//addNode() will add a node to the list
public void addNode(int data) {
   //Create a new node
   Node newNode = new Node(data);
   //If list is empty
   if(head == null) {
      //Both head and tail will point to newNode
      head = tail = newNode;
      //head's previous will point to null
      head.previous = null;
      //tail's next will point to null, as it is the last node of the list
      tail.next = null;
   }
   else {
      //newNode will be added after tail such that tail's next will point to newNode
      tail.next = newNode;
      //newNode's previous will point to tail
      newNode.previous = tail;
      //newNode will become new tail
      tail = newNode;
      //As it is last node, tail's next will point to null
      tail.next = null;
   }
}
//deleteFromEnd() will delete a node from the end of the list
public void deleteFromEnd() {
   //Checks whether list is empty
   if(head == null) {
      return;
   }
   else {
```

```
        //Checks whether the list contains only one node
        if(head != tail) {
            //Previous node to the tail will become new tail
            tail = tail.previous;
            //Node next to current tail will be made null
            tail.next = null;
        }
        //If the list contains only one element
        //Then it will remove node and now both head and tail will point to null
        else {
            head = tail = null;
        }
    }
}
//display() will print out the nodes of the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
        //Prints each node by incrementing the pointer.
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    DeleteNoteEnd dne = new DeleteEnd();
```

```
        //Add nodes to the list
        dne.addNode(10);
        dne.addNode(20);
        dne.addNode(30);
        dne.addNode(40);
        dne.addNode(50);
        //Printing Initial list
        System.out.println("Initial List: ");
        dne.display();
        while(dne.head != null) {
           dne.deleteFromEnd();
           //Printing updated list
           System.out.println("Updated List: ");
           dne.display();
        }
    }
}
```

Output:

```
Initial List:
10 20 30 40 50
Updated List:
10 20 30 40
Updated List:
10 20 30
Updated List:
10 20
Updated List:
10
Updated List:
List is empty
```

76. Wap to remove duplicate elements from Doubly Linked List.

```
public class RemoveDuplicateElemenstts {
   class Node{
      int data;
      Node previous;
      Node next;
      public Node(int data) {
         this.data = data;
      }
   }
   Node head, tail = null;
   //addNode() will add a node to the list
   public void addNode(int data) {
      Node newNode = new Node(data);
      //If list is empty
      if(head == null) {
         //Both head and tail will point to newNode
         head = tail = newNode;
         //head's previous will point to null
         head.previous = null;
         //tail's next will point to null, as it is the last node of the list
         tail.next = null;
      }
      else {
         //newNode will be added after tail such that tail's next will point to newNode
         tail.next = newNode;
         //newNode's previous will point to tail
         newNode.previous = tail;
         //newNode will become new tail
         tail = newNode;
         //As it is last node, tail's next will point to null
         tail.next = null;
```

```
    }
  }
  //removeDuplicateNode() will remove duplicate nodes from the list
  public void removeDuplicateNode() {
    //Node current will point to head
    Node current, index, temp;
    //Checks whether list is empty
    if(head == null) {
      return;
    }
    else {
      //Initially, current will point to head node
      for(current = head; current != null; current = current.next) {
        //index will point to node next to current
        for(index = current.next; index != null; index = index.next) {
          if(current.data == index.data) {
            //Store the duplicate node in temp
            temp = index;
            //index's previous node will point to node next to index thus, removes the duplicate
node
            index.previous.next = index.next;
            if(index.next != null)
              index.next.previous = index.previous;
            //Delete duplicate node by making temp to null
            temp = null;
          }
        }
      }
    }
  }
  //display() will print out the nodes of the list
```

```
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
        //Prints each node by incrementing the pointer.
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    RemoveDuplicateElements rde = new RemoveDuplicateElements();
    //Add nodes to the list
    rde.addNode(10);
    rde.addNode(20);
    rde.addNode(30);
    rde.addNode(20);
    rde.addNode(20);
    rde.addNode(40);
    rde.addNode(50);
    rde.addNode(30);
    System.out.println("Initial list: ");
    rde.display();
    //Removes duplicate nodes
    rde.removeDuplicateNode();
    System.out.println("List after removing duplicates: ");
    rde.display();
```

```
    }
}
```

Output:

```
Initial list:
10 20 30 20 20 40 50 30
List after removing duplicates:
10 20 30 40 50
```

77. Wap to rotate Doubly Linked List by N nodes.

```
public class RotatingList {
    class Node{
        int data;
        Node previous;
        Node next;
        public Node(int data) {
            this.data = data;
        }
    }
    int size = 0;
    //Represent the head and tail of the doubly linked list
    Node head, tail = null;
    //addNode() will add a node to the list
    public void addNode(int data) {
        //Create a new node
        Node newNode = new Node(data);
        //If list is empty
        if(head == null) {
            //Both head and tail will point to newNode
            head = tail = newNode;
            //head's previous will point to null
            head.previous = null;
            //tail's next will point to null, as it is the last node of the list
```

```
            tail.next = null;
        }
        else {
            //newNode will be added after tail such that tail's next will point to newNode
            tail.next = newNode;
            //newNode's previous will point to tail
            newNode.previous = tail;
            //newNode will become new tail
            tail = newNode;
            //As it is last node, tail's next will point to null
            tail.next = null;
        }
        //Size will count the number of nodes present in the list
        size++;
    }
    //rotateList() will rotate the list by given n nodes
    public void rotateList(int n) {
        //Initially, current will point to head
        Node current = head;
        //n should not be 0 or greater than or equal to number of nodes present in the list
        if(n == 0 || n >= size)
            return;
        else {
            //Traverse through the list till current point to nth node
            //after this loop, current will point to nth node
            for(int i = 1; i < n; i++)
                current = current.next;
            //Now to move entire list from head to nth node and add it after tail
            tail.next = head;
            //Node next to nth node will be new head
            head = current.next;
```

```
        //Previous node to head should be null
        head.previous = null;
        //nth node will become new tail of the list
        tail = current;
        //tail's next will point to null
        tail.next = null;
    }
}
//display() will print out the nodes of the list
public void display() {
    //Node current will point to head
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
        return;
    }
    while(current != null) {
        //Prints each node by incrementing the pointer.
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
public static void main(String[] args) {
    RotatingList rl = new RotatingList();
    //Add nodes to the list
    rl.addNode(10);
    rl.addNode(20);
    rl.addNode(30);
    rl.addNode(40);
    rl.addNode(50);
```

```
        System.out.println("Initial List: ");
        rl.display();
        //Rotates list by 3 nodes
        rl.rotateList(3);
        System.out.println("Updated List: ");
        rl.display();
    }
}
```

Output:

Initial List:

10 20 30 40 50

Updated List:

40 50 10 20 30

78. Wap to implement Binary Search Tree (BST).

```
ublic class BinarySearchTreeImplementation {
    private BstNode root;
    public boolean isEmpty() {
        return (this.root == null);
    }
    public void insert(Integer data) {
        System.out.print("[input: "+data+"]");
        if(root == null) {
            this.root = new BstNode(data);
            System.out.println(" -> inserted: "+data);
            return;
        }
        insertNode(this.root, data);
        System.out.print(" -> inserted: "+data);
        System.out.println();
    }
    private BstNode insertNode(BstNode root, Integer data) {
```

```
    BstNode tmpNode = null;
    System.out.print(" ->"+root.getData());
    if(root.getData() >= data) {
        System.out.print(" [L]");
        if(root.getLeft() == null) {
            root.setLeft(new BstNode(data));
            return root.getLeft();
        } else {
            tmpNode = root.getLeft();
        }
    } else {
        System.out.print(" [R]");
        if(root.getRight() == null) {
            root.setRight(new BstNode(data));
            return root.getRight();
        } else {
            tmpNode = root.getRight();
        }
    }
    return insertNode(tmpNode, data);
}
public static void main(String a[]) {
    BinarySearchTreeImpl bst = new BinarySearchTreeImpl();
    bst.insert(16);
    bst.insert(23);
    bst.insert(6);
    bst.insert(21);
    bst.insert(8);
    bst.insert(10);
    bst.insert(20);
}
```

```
}
```

79. Wap to find maximum width of a binary tree.

```
import java.util.LinkedList;
import java.util.Queue;
public class BinaryTreeWidth {
    public static class Node{
     int data;
     Node left;
     Node right;
     public Node(int data){
       //Assigning data to the new node, set left and right children to null
       this.data = data;
       this.left = null;
       this.right = null;
     }
    }
    //Representing the root of binary tree
    public Node root;
    public BinaryTree(){
     root = null;
    }
    //findMaximumWidth() this will find out the maximum width of the given binary tree
    public int findMaximumWidth() {
       int maxWidth = 0;
       //Variable nodesLevel keep tracks of number of nodes in each level
       int nodesLevel = 0;
       //queue will be used to keep track of nodes of tree level-wise
       Queue<Node> queue = new LinkedList<Node>();
       //Check if root is null, then width will be 0
       if(root == null) {
          System.out.println("Tree is empty");
```

```
        return 0;
    }
    else {
        //Add root node to queue as it represents the first level
        queue.add(root);
        while(queue.size() != 0) {
            //Variable nodesLevel will hold the size of queue i.e. number of elements in queue
            nodesLevel = queue.size();
            //maxWidth will hold maximum width.
            //If nodesLevel is greater than maxWidth then, maxWidth will hold the value of nodesLevel
            maxWidth = Math.max(maxWidth, nodesLevel);
            //If variable nodesLevel contains more than one node
            //then, for each node, we'll add left and right child of the node to the queue
            while(nodesLevel > 0) {
                Node current = queue.remove();
                if(current.left != null)
                    queue.add(current.left);
                if(current.right != null)
                    queue.add(current.right);
                nodesLevel--;
            }
        }
    }
    return maxWidth;
}
public static void main(String[] args) {
    BinaryTreeWidth bint = new BinaryTreeWidth();
    //Adding nodes to the binary tree
    bint.root = new Node(10);
    bint.root.left = new Node(20);
    bint.root.right = new Node(30);
```

```
        bint.root.left.left = new Node(40);
        bint.root.left.right = new Node(50);
        bint.root.right.left = new Node(60);
        bint.root.right.right = new Node(70);
        bint.root.left.left.left = new Node(80);
        //Display the maximum width of given tree
        System.out.println("Width of the binary tree: " + bint.findMaximumWidth());
    }
}
```

Output:

Width of the binary tree: 4

80. Wap to find largest Node in Binary tree.

```
public class LargestNodeBST {
   public static class Node{
      int data;
      Node left;
      Node right;
      public Node(int data){
         this.data = data;
         this.left = null;
         this.right = null;
      }
   }
    //Represent the root of binary tree
    public Node root;
    public LargestNode(){
     root = null;
    }
    //largestElement() this will find out the largest node in the binary tree
    public int largestElement(Node temp){
       //Checking whether tree is empty
```

```
    if(root == null) {
       System.out.println("Tree is empty");
       return 0;
     }
    else{
       int leftMax, rightMax;
       //Max will store temp's data
       int max = temp.data;
       //This will find largest element in left subtree
       if(temp.left != null){
          leftMax = largestElement(temp.left);
          //Comparing max with leftMax and store greater value into max
          max = Math.max(max, leftMax);
        }
        //This will find largest element in right subtree
        if(temp.right != null){
           rightMax = largestElement(temp.right);
           //Comparing max with rightMax and store greater value into max
           max = Math.max(max, rightMax);
        }
        return max;
    }
}
public static void main(String[] args) {
  LargestNodeBST bint = new LargestNodeBST();
  //Adding nodes to the binary tree
  bint.root = new Node(5);
  bint.root.left = new Node(19);
  bint.root.right = new Node(28);
  bint.root.left.left = new Node(42);
  bint.root.right.left = new Node(68);
```

```
        bint.root.right.right = new Node(10);
        //Display largest node in the binary tree
        System.out.println("Largest Node in the binary tree: " + bintt.largestElement(bint.root));
    }
}
```

Output:

Largest Node in the binary tree: 68

81. Wap to find smallest node in Binary tree.

```
public class SmallestNodeBST {
    public static class Node{
        int data;
        Node left;
        Node right;
        public Node(int data){
            //Assigning data to the new node, set left and right children to null
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
    //Represents the root of binary tree
    public Node root;
    public SmallestNode(){
        root = null;
    }
    //smallestElement() this will find out the smallest node in the binary tree
    public int smallestElement(Node temp){
        //Checking whether tree is empty
        if(root == null) {
            System.out.println("Tree is empty");
            return 0;
```

```
    }
    else {
        int leftMin, rightMin;
        //Min will store temp's data
        int min = temp.data;
        //This will find smallest element in left subtree
        if(temp.left != null){
          leftMin = smallestElement(temp.left);
          //If min is greater than leftMin then store the value of leftMin into min
          min = Math.min(min, leftMin);
        }
        //This will find smallest element in right subtree
        if(temp.right != null){
          rightMin = smallestElement(temp.right);
          //If min is greater than rightMin then store the value of rightMin into min
          min = Math.min(min, rightMin);
        }
        return min;
    }
}
public static void main(String[] args) {
  SmallestNodeBST bint = new SmallestNodeBST();
  //Adding nodes to the binary tree
  bint.root = new Node(8);
  bint.root.left = new Node(20);
  bint.root.right = new Node(15);
  bint.root.left.left = new Node(12);
  bint.root.right.left = new Node(25);
  bint.root.right.right = new Node(36);
  //Displays smallest node in the binary tree
  System.out.println("Smallest node in the binary tree: " + bint.smallestElement(bint.root));
```

```
    }
}
```

Output:

Smallest node in the binary tree: 8

82. Wap to find sum of all node in Binary Tree.

```
public class SumNodes {
    public static class Node{
     int data;
     Node left;
     Node right;
     public Node(int data){
        //Assigning data to the new node, set left and right children to null
        this.data = data;
        this.left = null;
        this.right = null;
     }
    }
    //Represent the root of binary tree
    public Node root;
    public SumOfNodes(){
     root = null;
    }
    //calculateSum() this will calculate the sum of all the nodes present in the binary tree
    public int calculateSum(Node temp){
     int sum, sumLeft, sumRight;
     sum = sumRight = sumLeft = 0;
     //Check whether tree is empty
     if(root == null) {
        System.out.println("Tree is empty");
        return 0;
     }
```

```
    else {
      //Calculate the sum of nodes present in left subtree
      if(temp.left != null)
        sumLeft = calculateSum(temp.left);
      //Calculate the sum of nodes present in right subtree
      if(temp.right != null)
        sumRight = calculateSum(temp.right);
      //Calculate the sum of all nodes by adding sumLeft, sumRight and root node's data
      sum = temp.data + sumLeft + sumRight;
      return sum;
    }
  }
  public static void main(String[] args) {
    SumNodes bint = new SumNodes();
    //Add nodes to the binary tree
    bint.root = new Node(8);
    bint.root.left = new Node(6);
    bint.root.right = new Node(4);
    bint.root.left.left = new Node(10);
    bint.root.right.left = new Node(15);
    //Displays the sum of all the nodes in the given binary tree
    System.out.println("Sum of all nodes of binary tree: " + bint.calculateSum(bint.root));
  }
 }
```

Output:

Sum of all nodes of binary tree: 43

83. Wap to find maximum depth or height of a tree.

```
public class BinaryTreeDepth {
  //Represent the node of binary tree
  public static class Node{
    int data;
```

```
    Node left;
    Node right;
    public Node(int data){
        //Assigning data to the new node, set left and right children to null
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
public Node root;
public BinaryTree(){
    root = null;
}
//findHeight() this will determine the maximum height of the binary tree
public int findHeight(Node temp){
    //Check whether tree is empty
    if(root == null) {
         System.out.println("Tree is empty");
        return 0;
    }
    else {
        int leftHeight = 0, rightHeight = 0;
        //Calculate the height of left subtree
        if(temp.left != null)
            leftHeight = findHeight(temp.left);
        //Calculate the height of right subtree
        if(temp.right != null)
            rightHeight = findHeight(temp.right);
        //Compare height of left subtree and right subtree
        //and store maximum of two in variable max
        int max = (leftHeight > rightHeight) ? leftHeight : rightHeight;
```

```
            //Calculate the total height of tree by adding height of root
            return (max + 1);
        }
    }
    public static void main(String[] args) {
        BinaryTreeDepth bint = new BinaryTreeDepth();
        //Adding nodes to the binary tree
        bint.root = new Node(10);
        bint.root.left = new Node(20);
        bint.root.right = new Node(30);
        bint.root.left.left = new Node(40);
        bint.root.right.left = new Node(50);
        bint.root.right.right = new Node(60);
        bint.root.right.right.right= new Node(70);
        bint.root.right.right.right.right = new Node(80);
        //Displays the maximum height of the given binary tree
        System.out.println("Height of given binary tree: " + bint.findHeight(bint.root));
  }
}
```

Output:

```
Height of given binary tree: 5
```

84. Wap to Connect Java Application with mysql database.

```
import java.sql.*;
class MysqlCon
{
    public static void main(String args[])
{
    try
    {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con=DriverManager.getConnection(
```

```
    "jdbc:mysql://localhost:3306/Employees","root","root");
     Statement stmt=con.createStatement();
     ResultSet rs=stmt.executeQuery("select * from emp");
     while(rs.next())
     System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
     con.close();
     }
    catch(Exception e)
     {
     System.out.println(e);
      }
}
}
```

85. Wap to get the object of ResultSetMetaData.

The getMetaData() method of ResultSet interface returns the object of ResultSetMetaData. Syntax:

```
      public ResultSetMetaData getMetaData()throws SQLException
```

```
import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();
ResultSetMetaData rsmd=rs.getMetaData();
System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1));
```

```
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

86. Wap to create table in database.

```
import java.sql.*;
public class JDBCExample
 {
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String URL = "jdbc:mysql://localhost/STUDENTS";
   static final String USER = "username";
   static final String PASS = "password";
   public static void main(String[] args)
 {
            Connection con = null;
             Statement stmt = null;
             Try
       {
     Class.forName("com.mysql.jdbc.Driver");
     con = DriverManager.getConnection(URL, USER, PASS);
     stmt = con.createStatement();
     String sql = "CREATE TABLE REGISTRATION " +
                  "(id INTEGER not NULL, " +
                  " first VARCHAR(255), " +
                  " last VARCHAR(255), " +
                  " age INTEGER, " +
                  " PRIMARY KEY ( id ))";
     stmt.executeUpdate(sql);
   }
catch(Exception e)
   {
```

```
    e.printStackTrace();
  }
finally
{
  try
   {
      con.close();
    stmt.close();
   }
   catch(Exception e)
  {
    e.printStackTrace();
  }
}
}
}
```

87. Wap to insert record in to table in the Database.

```
import java.sql.*;
public class JDBCExample {
  String URL = "jdbc:mysql://localhost:3306/STUDENTS";
  String USER = "username";
  String PASS = "password";
  public static void main(String[] args)
{
  Connection conn = null;
  Statement stmt = null;
  try
{
    Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection(URL, USER, PASS);
    stmt = conn.createStatement();
```

```
      String sql = "INSERT INTO Registration " +
                   "VALUES (100, 'Zara', 'Ali', 18)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration " +
                   "VALUES (101, 'Mahnaz', 'Fatma', 25)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration " +
                   "VALUES (102, 'Zaid', 'Khan', 30)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration " +
                   "VALUES(103, 'Sumit', 'Mittal', 28)";
      stmt.executeUpdate(sql);
    }
catch(Exception e)
{
      e.printStackTrace();
 }
finally
{
   try
    {
       con.close();
       stmt.close();
     }
     catch(Exception e)
   {
      e.printStackTrace();
   }
}
}
}
```

88. Wap to Delete records in to a table in the database.

```
public class JDBCExample
{
  String URL = "jdbc:mysql://localhost:3306/STUDENTS";
  String USER = "username";
  String PASS = "password";
  public static void main(String[] args)
  {
  Connection conn = null;
  Statement stmt = null;
  try{
        Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection(URL, USER, PASS);
    stmt = conn.createStatement();
    String sql = "DELETE FROM Registration " +
                 "WHERE id = 101";
    stmt.executeUpdate(sql);
    sql = "SELECT id, first, last, age FROM Registration";
    ResultSet rs = stmt.executeQuery(sql);
    while(rs.next())
    {
      int id  = rs.getInt("id");
      int age = rs.getInt("age");
      String first = rs.getString("first");
      String last = rs.getString("last");
      System.out.print(" ID:  " + id);
      System.out.print(" , Age: " + age);
      System.out.print(" , First: " + first);
      System.out.println(" , Last: " + last);
    }
  }
```

```
  catch(Exception e)
   {
        e.printStackTrace();
  }
 finally
{
  try
   {
      con.close();
     stmt.close();
      rs.close();
   }
    catch(Exception e)
  {
     e.printStackTrace();
  }
}
}
}
```

89. Wap to update records in to a table in the database.

```
import java.sql.*;
public class JDBCExample
{
  static final String DB_URL = "jdbc:mysql://localhost:3306/STUDENTS";
  static final String USER = "username";
  static final String PASS = "password";
  public static void main(String[] args)
 {
  Connection conn = null;
  Statement stmt = null;
  try
```

```
{
     Class.forName("com.mysql.jdbc.Driver");
     conn = DriverManager.getConnection(DB_URL, USER, PASS);
     stmt = conn.createStatement();
     String sql = "UPDATE Registration " +
                  "SET age = 30 WHERE id in (100, 101)";
     stmt.executeUpdate(sql);
     sql = "SELECT id, first, last, age FROM Registration";
     ResultSet rs = stmt.executeQuery(sql);
     while(rs.next())
    {
       int id  = rs.getInt("id");
       int age = rs.getInt("age");
       String first = rs.getString("first");
       String last = rs.getString("last");
       System.out.print("ID: " + id);
       System.out.print(", Age: " + age);
       System.out.print(", First: " + first);
       System.out.println(", Last: " + last);
     }
  }
  catch(Exception e)
   {
    e.printStackTrace();
   }
  finally
{
  try
   {
      con.close();
    stmt.close();
```

```
     rs.close();
    }
    catch(Exception e)
  {
     e.printStackTrace();
  }
}
}
}
```

90. Wap to print half pyramid using $.

```
$
$ $
$ $ $
$ $ $ $
$ $ $ $ $
```

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5;
      for(int i = 1; i <= rows; ++i) {
         for(int j = 1; j <= i; ++j) {
            System.out.print("$ ");
         }
         System.out.println();
      }
   }
}
```

91. Wap to print half triangle using numbers.

```
1
1 2
1 2 3
```

1 2 3 4

1 2 3 4 5

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5;
      for(int i = 1; i <= rows; ++i) {
         for(int j = 1; j <= i; ++j) {
            System.out.print(j + " ");
         }
         System.out.println();
      }
   }
}
```

92. Wap to print half triangle using alphabet.

A

B B

C C C

D D D D

E E E E E

```
public class Pattern {
   public static void main(String[] args) {
      char last = 'E', alphabet = 'A';
      for(int i = 1; i <= (last-'A'+1); ++i) {
         for(int j = 1; j <= i; ++j) {
            System.out.print(alphabet + " ");
         }
         ++alphabet;
         System.out.println();
      }
```

```
    }
}
```

93. Wap to print inverted half triangle using $

```
$ $ $ $ $
$ $ $ $
$ $ $
$ $
$
```

```
public class Pattern {
    public static void main(String[] args) {
        int rows = 5;
        for(int i = rows; i >= 1; --i) {
            for(int j = 1; j <= i; ++j) {
                System.out.print("$ ");
            }
            System.out.println();
        }
    }
}
```

94. Wap to print inverted half triangle using numbers.

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
public class Pattern {
    public static void main(String[] args) {
        int rows = 5;
        for(int i = rows; i >= 1; --i) {
```

```
        for(int j = 1; j <= i; ++j) {
            System.out.print(j + " ");
        }
        System.out.println();
    }
  }
}
```

95. Wap to print full triangle using $.

```
$
$ $ $
$ $ $ $ $
$ $ $ $ $ $ $
$ $ $ $ $ $ $ $ $
```

```
public class Pattern {
  public static void main(String[] args) {
    int rows = 5, k = 0;
    for(int i = 1; i <= rows; ++i, k = 0) {
      for(int space = 1; space <= rows - i; ++space) {
        System.out.print("  ");
      }
      while(k != 2 * i - 1) {
        System.out.print("$ ");
        ++k;
      }
      System.out.println();
    }
  }
}
```

96. Wap to print full pyramid using number

```
1
```

2 3 2

3 4 5 4 3

4 5 6 7 6 5 4

5 6 7 8 9 8 7 6 5

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5, k = 0, count = 0, count1 = 0;
      for(int i = 1; i <= rows; ++i) {
         for(int space = 1; space <= rows - i; ++space) {
            System.out.print("  ");
            ++count;
         }
         while(k != 2 * i - 1) {
            if (count <= rows - 1) {
               System.out.print((i + k) + " ");
               ++count;
            }
            else {
               ++count1;
               System.out.print((i + k - 2 * count1) + " ");
            }
            ++k;
         }
         count1 = count = k = 0;
         System.out.println();
      }
   }
}
```

97. Wap to print inverted full pyramid using $.

$ $ $ $ $ $ $ $ $

$ $ $ $ $ $ $

$ $ $ $ $

$ $ $

$

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 5;
      for(int i = rows; i >= 1; --i) {
         for(int space = 1; space <= rows - i; ++space) {
            System.out.print("  ");
         }
         for(int j=i; j <= 2 * i - 1; ++j) {
            System.out.print("$ ");
         }
         for(int j = 0; j < i - 1; ++j) {
            System.out.print("$ ");
         }
         System.out.println();
      }
   }
}
```

98. Wap to print Pascal's Triangle.

1

1 1

1 2 1

1 3 3 1

```
public class Pattern {
   public static void main(String[] args) {
      int rows = 4, cof = 1;
```

```
      for(int i = 0; i < rows; i++) {
         for(int space = 1; space < rows - i; ++space) {
            System.out.print("  ");
         }
         for(int j = 0; j <= i; j++) {
            if (j == 0 || i == 0)
               cof = 1;
            else
               cof = cof * (i - j + 1) / j;
            System.out.printf("%4d", cof);
         }
         System.out.println();
      }
   }
}
```

99. Wap to print Flyod triangle.

```
1
2 3
4 5 6


public class Pattern {
   public static void main(String[] args) {
      int rows = 3, number = 1;
      for(int i = 1; i <= rows; i++) {
         for(int j = 1; j <= i; j++) {
            System.out.print(number + " ");
            ++number;
         }
         System.out.println();
      }
   }
```

}

100. Wap to print the following pattern.

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows ");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j <= i; j++)
         {
            System.out.print(i + " ");
         }
         System.out.println();
      }
   }
}
```

101. Wap to print following pattern.

1

1 2

```
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows  ");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j <= i; j++)
         {
            System.out.print(j + " ");
         }
         System.out.println();
      }
      for (int i = rows; i >= 1; i--)
      {
         for (int j = 1; j < i; j++)
         {
            System.out.print(j + " ");
         }
```

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

102. Wap to print the following pattern.

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
import java.util.Scanner;
public class Pattern
{
    public static void main(String[] args)
    {
        // Create a new Scanner object
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows ");
        int rows = sc.nextInt();
        for (int i = rows; i >= 1; i--)
        {
            for (int j = 1; j <= i; j++)
            {
                System.out.print(j + " ");
```

```
        }
        System.out.println();
      }
      for (int i = 1; i <= rows; i++)
      {
        for (int j = 1; j <= i; j++)
        {
          System.out.print(j + " ");
        }
        System.out.println();
      }
   }
}
```

103. Wap to print following pattern.

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
```

```
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows ");
        int rows = sc.nextInt();
        for (int i = rows; i >= 1; i--)
        {
           for (int j = i; j >= 1; j--)
           {
              System.out.print(j + " ");
           }
           System.out.println();
        }
        for (int i = 1; i <= rows; i++)
        {
           for (int j = i; j >= 1; j--)
           {
              System.out.print(j + " ");
           }
           System.out.println();
        }
     }
}
```

104. Wap to print following pattern.

5 4 3 2 1

5 4 3 2

5 4 3

5 4

5

```
import java.util.Scanner;
public class Pattern
```

```
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows  ");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = rows; j >= i; j--)
         {
            System.out.print(j + " ");
         }
         System.out.println();
      }
   }
}
```

105. Wap to print following pattern.

```
5
5 4
5 4 3
5 4 3 2
5 4 3 2 1
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
```

```
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows");
        int rows = sc.nextInt();
        for (int i = rows; i >= 1; i--)
        {
            for (int j = rows; j >= i; j--)
            {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

106. Wap to print following pattern.

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

```
import java.util.Scanner;
public class Pattern
{
    public static void main(String[] args)
    {
        // Create a new Scanner object
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows");
        int rows = sc.nextInt();
```

```
        for (int i = 1; i <= rows; i++)
        {
            for (int j = i; j >= 1; j--)
            {
                System.out.print(j + " ");
            }
            System.out.println();
        }
    }
}
```

107. Wap to print following pattern.

```
1
2 7
3 8 13
4 9 14 19
5 10 15 20 25
```

```
import java.util.Scanner;
public class Pattern
{
    public static void main(String[] args)
    {
        // Create a new Scanner object
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows");
        int rows = sc.nextInt();
        for (int i = 1; i <= rows; i++)
        {
            int temp = i;
            for (int j = i; j >= 1; j--)
```

```
        {
            System.out.print(temp + " ");
            temp = temp + rows;
        }
        System.out.println();
      }
   }
}
```

108. Wap to print following pattern.

```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j <= i; j++)
         {
            System.out.print(j + " ");
         }
```

```
        for (int k = i - 1; k >= 1; k--)
        {
            System.out.print(k + " ");
        }
        System.out.println();
      }
   }
}
```

109. Wap to print following pattern.

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j < i; j++)
         {
            System.out.print(" ");
         }
```

```
        for (int k = 1; k <= rows - i + 1; k++)
        {
           System.out.print(k + " ");
        }
        System.out.println();
      }
   }
}
```

110. Wap to print following pattern.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
```

```
            for (int j = rows; j > i; j--)
            {
                System.out.print(" ");
            }
            for (int k = 1; k <= i; k++)
            {
                System.out.print(k + " ");
            }
            System.out.println();
        }
        for (int i = 1; i <= rows; i++)
        {
            for (int j = 1; j <= i; j++)
            {
                System.out.print(" ");
            }
            for (int k = 1; k <= rows - i; k++)
            {
                System.out.print(k + " ");
            }
            System.out.println();
        }
    }
}
```

111. Wap to print following pattern.

```
12345
2345
345
45
5
5
```

45

345

2345

12345

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j < i; j++)
         {
            System.out.print(" ");
         }
         for (int k = i; k <= rows; k++)
         {
            System.out.print(k);
         }
         System.out.println();
      }
      for (int i = rows; i >= 1; i--)
      {
         for (int j = 1; j < i; j++)
         {
```

```
            System.out.print(" ");
          }
          for (int k = i; k <= rows; k++)
          {
            System.out.print(k);
          }
          System.out.println();
        }
      }
}
```

112. Wap to print following pattern.

```
1 2 3 4 5
2 3 4 5
3 4 5
4 5
5
5
4 5
3 4 5
2 3 4 5
1 2 3 4 5
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
```

```
        int rows = sc.nextInt();
        for (int i = 1; i <= rows; i++)
        {
            for (int j = 1; j < i; j++)
            {
                System.out.print(" ");
            }
            for (int k = i; k <= rows; k++)
            {
                System.out.print(k + " ");
            }
            System.out.println();
        }
        for (int i = rows; i >= 1; i--)
        {
            for (int j = 1; j < i; j++)
            {
                System.out.print(" ");
            }
            for (int k = i; k <= rows; k++)
            {
                System.out.print(k + " ");
            }
            System.out.println();
        }
    }
}
```

113. Wap to print following pattern.

5

4 5

3 4 5

2 3 4 5

1 2 3 4 5

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = rows; i >= 1; i--)
      {
         for (int j = 1; j < i; j++)
         {
            System.out.print(" ");
         }
         for (int k = i; k <= rows; k++)
         {
            System.out.print(k + " ");
         }
         System.out.println();
      }
   }
}
```

114. Wap to print following pattern.

1

1 0

1 0 1

1 0 1 0

1 0 1 0 1

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j <= i; j++)
         {
            System.out.print(j % 2 + " ");
         }
         System.out.println();
      }
   }
}
```

115. Wap to print following pattern.

1 0 0 0 0

0 2 0 0 0

0 0 3 0 0

0 0 0 4 0

0 0 0 0 5

```
import java.util.Scanner;
```

```
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j < i; j++)
         {
            System.out.print("0 ");
         }
         System.out.print(i + " ");
         for (int k = i; k < rows; k++)
         {
            System.out.print("0 ");
         }
         System.out.println();
      }
   }
}
```

116. Wap to print following pattern.

1 1 1 1 1

1 1 1 2 2

1 1 3 3 3

1 4 4 4 4

5 5 5 5 5

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = rows; j > i; j--)
         {
            System.out.print(1 + " ");
         }
         for (int k = 1; k <= i; k++)
         {
            System.out.print(i + " ");
         }
         System.out.println();
      }
   }
}
```

117. Wap to print following pattern.

```
1 2 3 4 5 4 3 2 1
2 3 4 5 4 3 2
3 4 5 4 3
4 5 4
5
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = i; j <= rows; j++)
         {
            System.out.print(j + " ");
         }
         for (int k = rows - 1; k >= i; k--)
         {
            System.out.print(k + " ");
         }
         System.out.println();
      }
   }
}
```

118. Wap to print following pattern.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = rows; j > i; j--)
         {
            System.out.print(" ");
         }
         for (int k = 1; k <= i; k++)
         {
            System.out.print(i + " ");
         }
         System.out.println();
      }
   }
}
```

119. Wap to print following pattern.

```
5 5 5 5 5
4 5 5 5 5
3 4 5 5 5
2 3 4 5 5
1 2 3 4 5
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = rows; i >= 1; i--)
      {
         for (int j = i; j < rows; j++)
         {
            System.out.print(j + " ");
         }
         for (int k = rows - i; k < rows; k++)
         {
            System.out.print(5 + " ");
         }
         System.out.println();
      }
   }
}
```

120. Wap to print following pattern.

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

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      int k = 1;
      for (int i = 1; i <= rows; i++)
      {
         k=i;
         for (int j = 1; j <= i; j++)
         {
            System.out.print(k + " ");
            k = k + rows - j;
         }
         System.out.println();
      }
   }
}
```

121. Wap to print following pattern.

```
1
2 4
3 6 9
4 8 12 16
5 10 15
6 12
7
```

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      int temp = 1;
      for(int i=1; i<=rows/2+1; i++)
      {
         for(int j=1;j<=i;j++)
         {
            System.out.print(temp*j+" ");
         }
         System.out.println();
         temp++;
      }
      for(int i=1; i<=rows/2; i++)
      {
         for(int j=1;j<=rows/2+1-i;j++)
         {
            System.out.print(temp*j+" ");
         }
         System.out.println();
         temp++;
      }
   }
```

}

122. Wap to print following pattern.

1

2 9

3 8 10

4 7 11 14

5 6 12 13 15

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 0; i < rows; i++)
      {
         for (int j = 0; j <= i; j++)
         {
            if (j % 2 == 0)
            {
               System.out.print(1 + j * rows - (j - 1) * j / 2 + i - j + " ");
            } else
            {
               System.out.print(1 + j * rows - (j - 1) * j / 2 + rows - 1 - i + " ");
            }
         }
       System.out.println();
```

```
    }
  }
}
```

123. Wap to print following pattern.

1 10 11 20 21

2 9 12 19 22

3 8 13 18 23

4 7 14 17 24

5 6 15 16 25

```
import java.util.Scanner;
public class Pattern
{
  public static void main(String[] args)
  {
    // Create a new Scanner object
    Scanner sc = new Scanner(System.in);
    // Get the number of rows from the user
    System.out.println("Enter the number of rows");
    int rows = sc.nextInt();
    for (int i = 0; i < rows; i++)
    {
      for (int j = 0; j < rows; j++)
      {
        if (j % 2 == 0)
          System.out.print((rows * (j)) + i + 1 + " ");
        else
          System.out.print((rows * (j + 1)) - i + " ");
      }
      System.out.print("\n");
    }
```

```
    }
}
```

124. Wap to print following pattern.

5 5 5 5 5

5 4 4 4 4

5 4 3 3 3

5 4 3 2 2

5 4 3 2 1

```
import java.util.Scanner;
public class Pattern
{
    public static void main(String[] args)
    {
        // Create a new Scanner object
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows");
        int rows = sc.nextInt();
        int temp = 0;
        for (int i = rows; i >= 1; i--)
        {
            for (int j = rows ; j >= i; j--)
            {
                System.out.print(j + " ");
                temp =j;
            }
            for (int k = rows - i+1; k < rows; k++)
            {
                System.out.print(temp + " ");
            }
```

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

125. Wap to print following pattern.

```
5
4 5 4
3 4 5 4 3
2 3 4 5 4 3 2
1 2 3 4 5 4 3 2 1
2 3 4 5 4 3 2
3 4 5 4 3
4 5 4
5
```

```
import java.util.Scanner;
public class Pattern
{
    public static void main(String[] args)
    {
        // Create a new Scanner object
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows");
        int rows = sc.nextInt();
        for (int i = rows; i >= 1; i--)
        {
            for (int j = i; j <= rows; j++)
            {
                System.out.print(j + " ");
            }
```

```
            for (int k = rows-1; k >= i; k--)
            {
               System.out.print(k + " ");
            }
            System.out.println();
         }
         for (int i = 2; i <= rows; i++)
         {
            for (int j = i; j <= rows; j++)
            {
               System.out.print(j + " ");
            }
            for (int k = rows-1; k >= i; k--)
            {
               System.out.print(k + " ");
            }
            System.out.println();
         }
      }
}
```

126. Wap to print following pattern.

1 2 3 4 5

2 3 4 5 1

3 4 5 1 2

4 5 1 2 3

5 1 2 3 4

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
```

```
  {
    // Create a new Scanner object
    Scanner sc = new Scanner(System.in);
    // Get the number of rows from the user
    System.out.println("Enter the number of rows");
    int rows = sc.nextInt();
    for (int i = 1; i <= rows; i++)
    {
      int j = i;
      for (int k = 1; k <= rows; k++)
      {
        System.out.print(j + " ");
        j++;
        if (j > rows)
          j = 1;
      }
      System.out.println();
    }
    sc.close();
  }
}
```

127. Wap to print following pattern.

1 3 5 7 9

3 5 7 9 1

5 7 9 1 3

7 9 1 3 5

9 1 3 5 7

```
import java.util.Scanner;
public class Pattern
{
```

```
    public static void main(String[] args)
    {
        // Create a new Scanner object
        Scanner sc = new Scanner(System.in);
        // Get the number of rows from the user
        System.out.println("Enter the number of rows");
        int rows = sc.nextInt();
        for (int i = 1; i <= rows; i++)
        {
            int j = (i * 2) - 1;
            for (int k = 1; k <= rows; k++)
            {
                System.out.print(j + " ");
                j += 2;
                if (j > (rows * 2) - 1)
                    j = 1;
            }
            System.out.println();
        }
        sc.close();
    }
}
```

128. Wap to print following pattern.

```
1 1
12 21
123 321
1234 4321
1234554321
```

```
import java.util.Scanner;
public class Pattern
```

```
{
   public static void main(String[] args)
   {
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i = 1; i <= rows; i++)
      {
         for (int j = 1; j <= i; j++)
         {
            System.out.print(j);
         }
         for (int j= i*2 ; j < rows*2; j++)
         {
            System.out.print(" ");
         }
         for (int l = i; l >= 1; l--)
         {
            System.out.print(l);
         }
         System.out.println();
      }
      sc.close();
   }
}
```

129. Wap to print following pattern.

```
1
2 3
6 5 4
```

7 8 9 10

15 14 13 12 11

```
import java.util.Scanner;
public class Pattern
{
   public static void main(String[] args)
   {
      int curRow = 1;
      int counter = 1;
      // Create a new Scanner object
      Scanner sc = new Scanner(System.in);
      // Get the number of rows from the user
      System.out.println("Enter the number of rows");
      int rows = sc.nextInt();
      for (int i=1; i<= rows; i++)
      {
         if (i % 2 == 0)
         {
            for (int j = 1; j<=i; j++)
            {
               System.out.print(counter  +" ");
               counter++;
            }
         }
         else
         {
            int reverse = curRow + counter - 1;
            for (int j = 0; j<i; j++)
            {
               System.out.print(reverse--  +" ");
```

```
            counter++;
          }
        }
        System.out.println();
        curRow++;
      }
    }
}
```

130. Bubble Sort

Bubble sorting is one of the simplest sorts. The general idea is to swap small numbers to the front by comparing and exchanging with adjacent elements. This process is similar to the fact that the blisters rise, hence the name. Take a chestnut and sort the unordered sequences of 5, 3, 8, 6, and 4. First, bubbling from the back to the front, 4 and 6 comparison, swapping 4 to the front, and the sequence becomes 5, 3, 8, 4, 6. In the same way, 4 and 8 are exchanged, and 5, 3, 4, 8, 6, 3, and 4 are exchanged. 5 and 3 exchange, become 3,5,4,8,6,3. This time the bubbling is finished, the smallest number 3 is discharged to the front. Bubbling the remaining sequences in turn yields an ordered sequence. The time complexity of bubble sorting is O(n^2).

```
Public  class  BubbleSort {
   Public  static  void  bubbleSort ( int [] arr ) {
      if (arr ==  null  || arr . length ==  0 )
         return ;
      for ( int i = 0 ; i < arr . length - 1 ; i ++ ) {
         for ( Int j = arr . length - 1 ; j > i; j-- ) {
            if (arr[j] < arr[j - 1 ]) {
               Swap(arr, j - 1 , j);
            }
         }
      }
   }
   Public  static  void  swap ( int [] arr , int  i , int  j ) {
      int temp = arr[i];
      Arr[i] = arr[j];
```

```
        Arr[j] = temp;
    }
}
```

131. Select Sort

The idea of choosing sorting is actually similar to bubble sorting, which puts the smallest element to the front after a sort. But the process is different, bubble sorting is through adjacent comparisons and exchanges. The choice of sorting is through the choice of the whole. For a chestnut, simple sorting of the unordered sequence of 5, 3, 8, 6, 4, first choose the smallest number other than 5 to exchange with 5, that is, select 3 and 5 exchanges, after a sorting becomes 3,5,8,6,4. Select and exchange the remaining sequences at once, and finally get an ordered sequence. In fact, the choice of sorting can be seen as the optimization of the bubble sorting, because the purpose is the same, but the sorting is only exchanged under the premise of determining the minimum number, which greatly reduces the number of exchanges. The time complexity of selecting sorting is O(n^2)

```
public class SelectSort {
    public static void selectSort(int[] arr) {
        if(arr == null || arr.length == 0)
            return ;
        int minIndex = 0;
        for(int i=0; i<arr.length-1; i++) {
            minIndex = i;
            for(int j=i+1; j<arr.length; j++) {
                if(arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            if(minIndex != i) {
                swap(arr, i, minIndex);
            }
        }
    }
    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
```

```
        arr[j] = temp;
    }
}
```

132. Insert Sort

Insert sorting does not achieve sorting by swapping locations but by finding the appropriate location insert elements. I believe that everyone has had the experience of playing cards, especially the number of cards. You may have to sort out your cards when you split the cards. How do you sort them out when you have more cards? Just get a card and find a suitable place to insert. This principle is actually the same as insert sorting. For a chestnut, simply insert the 5, 3, 8, 6, 4 unordered sequence, first assume that the position of the first number is correct. Think about the first card when you get it. . Then 3 is to be inserted in front of 5, and 5 is moved back to 3, 5, 8, 6, and 4. It should be the same when thinking about finishing cards. Then 8 does not move, 6 is inserted in front of 8, 8 is shifted back by one, 4 is inserted in front of 5, and one is moved backward from 5 to one. Note that when inserting a number, make sure that the number before the number is already ordered. The time complexity of simple insert sorting is also O(n^2).

```
public class InsertSort {
    public static void insertSort(int[] arr) {
        if(arr == null || arr.length == 0)
            return ;
        for(int i=1; i<arr.length; i++) {
            int j = i;
            int target = arr[i];
            while(j > 0 && target < arr[j-1]) {
                arr[j] = arr[j-1];
                j --;
            }
            arr[j] = target;
        }
    }
}
```

133. Quick Sort

Quick sorting is very high-end when listening to a name. Quick sorting is actually the best performing sorting algorithm in practical applications. Quick sorting is high-end, but in fact its idea is from bubble sorting. Bubble sorting is to minimize the bubbling to the top by comparing and swapping

adjacent elements, while fast sorting is to compare and exchange decimals and large numbers. Not only will the decimals bubble up to the top but also the big numbers below.

Give a chestnut: Quickly sort the unordered sequence of 5, 3, 8, 6, 4. The idea is that the right pointer is smaller than the reference number, and the left pointer is larger than the reference number and exchanged.

5,3,8,6,4 Use 5 as the benchmark for comparison. Finally, move 5 small to the left of 5, and move to the right of 5 by 5.

5,3,8,6,4 First set i, j two pointers point to the two ends, j pointer scan first (think why?) 4 to 5 small stop. Then i scan, 8 to 5 stop. Exchange i, j position.

5,3,4,6,8 Then the j pointer is scanned again. At this time, when j scans 4, the two pointers meet. stop. Then exchange 4 and the reference number.

4,3,5,6,8 After one division, the goal of the left side is smaller than 5 and the right side is larger than 5. The left and right subsequences are then recursively sorted to finally obtain an ordered sequence.

There is a question left on it. Why do you have to move the pointer first? First of all, this is not absolute, depending on the position of the reference number, because when the last two pointers meet, the reference number is exchanged to the position of the encounter. Generally, the first number is selected as the reference number, then it is on the left side, so the number of the last encounters must be exchanged with the reference number, so the number of encounters must be smaller than the reference number. So the j pointer moves first to find a number smaller than the base number.

Quick sorting is unstable, and its time average time complexity is O(nlgn).

```
public class QuickSort {
  //一次划分
  public static int partition(int[] arr, int left, int right) {
    int pivotKey = arr[left];
    int pivotPointer = left;
    while(left < right) {
      while(left < right && arr[right] >= pivotKey)
```

```
            right --;
        while(left < right && arr[left] <= pivotKey)
            left ++;
        swap(arr, left, right); //把大的交换到右边，把小的交换到左边。
    }
    swap(arr, pivotPointer, left); //最后把pivot交换到中间
    return left;
  }
  public static void quickSort(int[] arr, int left, int right) {
    if(left >= right)
        return ;
    int pivotPos = partition(arr, left, right);
    quickSort(arr, left, pivotPos-1);
    quickSort(arr, pivotPos+1, right);
  }
  public static void sort(int[] arr) {
    if(arr == null || arr.length == 0)
        return ;
    quickSort(arr, 0, arr.length-1);
  }
  public static void swap(int[] arr, int left, int right) {
    int temp = arr[left];
    arr[left] = arr[right];
    arr[right] = temp;
  }
}
```

134. Heap Sorting

Heap sorting is a sorting of choices implemented by means of heaps, and the idea is sorted with simple choices. The following is an example of a large top heap. Note: If you want to sort in ascending order, use the big top heap, and vice versa. The reason is that the top element of the heap needs to be swapped to the end of the sequence.

First, implementing heap sorting requires solving two problems:

How to make a heap from an unordered sequence key?

How do you adjust the remaining elements to a new heap after outputting the top element?

The first problem is that you can directly use a linear array to represent a heap. Building a heap from the initial unordered sequence requires a bottom-up adjustment from the first non-leaf element to a heap.

The second question, how to adjust to piles? The first is to exchange the top and bottom elements. Then compare the left and right child nodes of the current top element, because in addition to the current heap top element, the left and right child heaps satisfy the condition, then the current top top element needs to be exchanged with the larger (large top heap) of the left and right child nodes until Leaf node. We call this adjustment from the top of the pile to the screen.

The process of building a heap from an unordered sequence is a process of repeated screening. If this sequence is considered to be a complete binary tree, then the last non-terminal node is n/2 to take the bottom element, and thus can be filtered. Give a chestnut:

The stacking of 49, 38, 65, 97, 76, 13, 27, 49 sequences The initial heap and adjustment process is as follows:

```
public class HeapSort {
   public static void heapAdjust(int[] arr, int start, int end) {
      int temp = arr[start];
      for(int i=2*start+1; i<=end; i*=2) {
         if(i < end && arr[i] < arr[i+1]) {
            i ++;
         }
         if(temp >= arr[i]) {
            break;
         }
```

```
            arr[start] = arr[i];
            start = i;
        }
        arr[start] = temp;
    }
    public static void heapSort(int[] arr) {
        if(arr == null || arr.length == 0)
            return ;
        for(int i=arr.length/2; i>=0; i--) {
            heapAdjust(arr, i, arr.length-1);
        }
        for(int i=arr.length-1; i>=0; i--) {
            swap(arr, 0, i);
            heapAdjust(arr, 0, i-1);
        }
    }
    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

135. Hill Sorting

Hill sorting is an efficient implementation of insert sorting, also called downsizing incremental sorting. In simple insert sorting, if the sequence to be sorted is positive, the time complexity is O(n). If the sequence is basically ordered, the efficiency of using direct insert sorting is very high. Hill sorting takes advantage of this feature. The basic idea is: first divide the entire sequence to be recorded into several sub-sequences for direct insertion sorting, and then perform a direct insertion sorting on all records when the records in the entire sequence are basically ordered.

Give a chestnut:

As can be seen from the above sorting process, the feature of the Hill sorting is that the subsequence is not simply segmented by segment, but a record separated by an increment is composed into a subsequence. As in the example above, the increment of the first sort is 5, and the increment of the second sort is 3. Since the keywords recorded in the first two insertions are compared with the keywords of the previous record in the same subsequence, the smaller records are not moved step by step, but jumped to Moving forward, so that when the last order is sorted, the entire sequence has been basically ordered, as long as a small amount of comparison and movement of the record is made. Therefore, Hill sorting is more efficient than direct insert sorting.

The analysis of Hill sorting is complex, and time complexity is a function of the increment taken, which involves some mathematical problems. However, based on a large number of experiments, when n is in a certain range, the time complexity can reach O(n^1.3).

```
public class ShellSort {
   public static void shellInsert(int[] arr, int d) {
      for(int i=d; i<arr.length; i++) {
         int j = i - d;
         int temp = arr[i];
         while (j>=0 && arr[j]>temp) {
            arr[j+d] = arr[j];
            j -= d;
         }
         if (j != i - d)
            arr[j+d] = temp;
      }
   }
   public static void shellSort(int[] arr) {
      if(arr == null || arr.length == 0)
         return ;
      int d = arr.length / 2;
      while(d >= 1) {
         shellInsert(arr, d);
         d /= 2;
      }
```

```
    }
}
```

136. Merge Sort

Merging and sorting is another different sorting method, because the merge sorting uses the idea of recursive partitioning, so it is easier to understand. The basic idea is to recursively sub-problems and then merge the results. The sequence to be sorted is treated as two ordered subsequences, then the two subsequences are merged, and the subsequences are then treated as two ordered sequences. . . . . Looking backwards, in fact, it is the first two or two mergers, and then the four or four mergers. . . Eventually an ordered sequence is formed. The space complexity is O(n) and the time complexity is O(nlogn).

```
public class MergeSort {
    public static void mergeSort(int[] arr) {
        mSort(arr, 0, arr.length-1);
    }
    public static void mSort(int[] arr, int left, int right) {
        if(left >= right)
            return ;
        int mid = (left + right) / 2;
        mSort(arr, left, mid);
        mSort(arr, mid+1, right);
        merge(arr, left, mid, right);
    }
    public static void merge(int[] arr, int left, int mid, int right) {
        //[left, mid] [mid+1, right]
        int[] temp = new int[right - left + 1];
        int i = left;
        int j = mid + 1;
        int k = 0;
        while(i <= mid && j <= right) {
            if(arr[i] <= arr[j]) {
                temp[k++] = arr[i++];
            }
```

```
            else {
                temp[k++] = arr[j++];
            }
        }
        while(i <= mid) {
            temp[k++] = arr[i++];
        }
        while(j <= right) {
            temp[k++] = arr[j++];
        }
        for(int p=0; p<temp.length; p++) {
            arr[left + p] = temp[p];
        }
    }
}
```

137. Count Sort

If an interviewer asks you to write an O(n) time complexity sorting algorithm during the interview, you should never say: This is impossible! Although the lower limit of the previous sort based on comparison is O(nlogn). However, there is indeed a linear time complexity ordering, except that there is a premise that the number to be sorted must satisfy a certain range of integers, and counting sorting requires more auxiliary space. The basic idea is to count the number of each number by using the number to be sorted as the subscript of the count array. Then output sequentially to get an ordered sequence.

```
public class CountSort {
    public static void countSort(int[] arr) {
        if(arr == null || arr.length == 0)
            return ;
        int max = max(arr);
        int[] count = new int[max+1];
        Arrays.fill(count, 0);
        for(int i=0; i<arr.length; i++) {
            count[arr[i]] ++;
```

```
        }
        int k = 0;
        for(int i=0; i<=max; i++) {
            for(int j=0; j<count[i]; j++) {
                arr[k++] = i;
            }
        }
    }
    public static int max(int[] arr) {
        int max = Integer.MIN_VALUE;
        for(int ele : arr) {
            if(ele > max)
                max = ele;
        }
        return max;
    }
}
```

138. Bucket Sort

Bucket sorting is an improvement and promotion of counting sorting, but there are many materials on the Internet that confuse counting sorting and bucket sorting. In fact, bucket sorting is much more complicated than counting sorting.

The basic idea of bucket sorting:

Suppose there is a set of key sequence K[1….n] of length N. First divide this sequence into M subintervals (barrels). Then, based on a mapping function, the keyword k of the sequence to be sorted is mapped into the i-th bucket (ie, the subscript i of the bucket array B), then the keyword k is taken as an element in B[i] (each Bucket B[i] is a set of sequences of size N/M). Then compare and sort all the elements in each bucket B[i] (you can use fast queue). Then enumerate all the contents of the output B[0]….B[M] in sequence, which is an ordered sequence. Bindex=f(key) where bindex is the subscript of the bucket array B (ie, the bindex bucket), and k is the key of the sequence to be sorted. The key to bucket sorting efficiency is the mapping function, which must be done: if the keyword k1 < k2, then f(k1) <= f(k2).

In other words, the minimum data in B(i) is greater than the largest data in B(i-1). Obviously, the determination of the mapping function has a lot to do with the characteristics of the data itself.

Give a chestnut:

Suppose the sequence K = {49, 38, 35, 97, 76, 73, 27, 49}. These data are all between 1 and 100. So we customize 10 buckets and then determine the mapping function f(k) = k/10. Then the first keyword 49 will be located in the 4th bucket (49/10=4). All the keywords are piled into the bucket in turn, and quickly sorted in each non-empty bucket to get the figure as shown. An ordered sequence can be obtained by sequentially outputting the data in each B[i].

Bucket sorting analysis:

Bucket sorting utilizes the mapping of functions, reducing almost all comparison work. In fact, the calculation of the f(k) value of bucket sorting is equivalent to the division in the fast row, the subsequence in the hill sorting, the subproblem in the merge sorting, and the large amount of data has been segmented into basically ordered. Data block (bucket). Then you only need to do advanced comparison sorting on a small amount of data in the bucket.

The time complexity of bucket sorting for N keywords is divided into two parts:

(1) Cycle calculation of the bucket mapping function for each keyword. This time complexity is O(N).

(2) Using the advanced comparison sorting algorithm to sort all the data in each bucket, the time complexity is ∑ O(Ni*logNi). Where Ni is the amount of data for the i-th bucket.

Obviously, part (2) is the decisive factor in the performance of the barrel sorting. Minimizing the amount of data in the bucket is the only way to increase efficiency (because the best average time complexity based on comparison sorting can only reach O(N*logN)). Therefore, we need to do the following two things:

(1) The mapping function f(k) can evenly distribute N data into M buckets, so that each bucket has [N/M] data amount.

(2) Try to increase the number of buckets. In the limit case, only one data can be obtained per bucket, which completely avoids the “comparison” sorting operation of the data in the bucket. Of course, it is not easy to do this. In the case of a large amount of data, the f(k) function will make the

number of bucket collections huge and the space was wasted. This is a trade-off between time and space costs.

For N rows of data to be queued, M buckets, the average bucket sorting average time complexity of each bucket [N/M] data is:

O(N)+O(M*(N/M) log(N/M))=O(N+N (logN-logM))=O(N+N logN-N logM) When N=M, That is, in the limit case, there is only one data per bucket. The best efficiency of bucket sorting can reach O(N).

Summary: The average time complexity of bucket sorting is linear O(N+C), where C=N*(logN-logM). If the number of buckets M is larger with respect to the same N, the efficiency is higher, and the best time complexity reaches O(N). Of course, the space complexity of bucket sorting is O(N+M). If the input data is very large and the number of buckets is very large, the space cost is undoubtedly expensive. In addition, bucket sorting is stable.

```
public class BucketSort {
   public static void bucketSort(int[] arr) {
      if(arr == null && arr.length == 0)
         return ;
      int bucketNums = 10;
      List<List<Integer>> buckets = new ArrayList<List<Integer>>();
      for(int i=0; i<10; i++) {
         buckets.add(new LinkedList<Integer>());
      }
      for(int i=0; i<arr.length; i++) {
         buckets.get(f(arr[i])).add(arr[i]);
      }
      for(int i=0; i<buckets.size(); i++) {
         if(!buckets.get(i).isEmpty()) {
            Collections.sort(buckets.get(i);
         }
      }
         int k = 0;
```

```
        for(List<Integer> bucket : buckets) {
            for(int ele : bucket) {
                arr[k++] = ele;
            }
        }
    }
    public static int f(int x) {
        return x / 10;
    }
}
```

139. Cardinality Sort

The cardinality sorting is a sorting method different from the previous sorting method, and the cardinality sorting does not require comparison between the recording keywords. Cardinality sorting is a method of sorting single logical keywords by means of multi-keyword sorting. The so-called multi-keyword sorting is to have multiple keywords with different priorities. For example, if the scores of the two people are the same, the higher the language is in the front and the higher in the Chinese language, the higher the mathematics is in the front. . . If the numbers are sorted, the ones, tens, and hundreds are the keywords of different priorities. If the sort is to be sorted, the ones, tens, and hundreds are incremented at a time. The cardinality sorting is achieved by multiple allocations and collections, and the keywords with low priority are assigned and collected first.

```
Public  class  RadixSort {
    Public  static  void  radixSort ( int [] arr ) {
        if (arr ==  null  && arr . length ==  0 )
            return ;
        Int maxBit = getMaxBit(arr);
        For ( int i = 1 ; i <= maxBit; i ++ ) {
            List< List< Integer > > buf = distribute(arr, i); // Assign
            collecte(arr, buf); // Collection
        }
    }
   public static List< List< Integer > > distribute ( int [] arr , int iBit ) {
        List< List< Integer > > buf = new ArrayList< List< Integer > > ();
        for ( int j = 0 ; j
```

```
        < 10 ; j ++ ) {
          Buf . add( new  LinkedList< Integer > ());
        }
        For ( int i = 0 ; i < arr . length; i ++ ) {
          Buf . get(getNBit(arr[i], iBit)) . add(arr[i]);
        }
        Return buf;
    }
    public static void collecte ( int [] arr , List< List< Integer > > buf ) {
        int k = 0 ;
        for ( List< Integer > bucket : buf) {
          for ( int ele : bucket) {
            Arr[k ++ ] = ele;
          }
        }
    }
  public static int getMaxBit ( int [] arr ) {
        int max = Integer . MIN_VALUE ;
        for ( int ele : arr) {
          int len = (ele + " " ) . length();
          if (len > max)
            Max = len;
        }
    public static int getNBit ( int x , int n ) {
        String sx = x +  " " ;
        if (sx . length() < n)
          return  0 ;
        else
          return sx . charAt(sx . length() - n) -  ' 0 ' ;
    }
}
```