LAB CONTENT

MODULE – ADS

Introduction:

Data Structures

Why do data structures matter?

We need to organize data so that it can be accessed quickly and usefully. Examples of data structures are queues, stacks, lists, heaps, search trees, hash tables, bloom filters, union-find, etc.

Why do we have so many data structures?

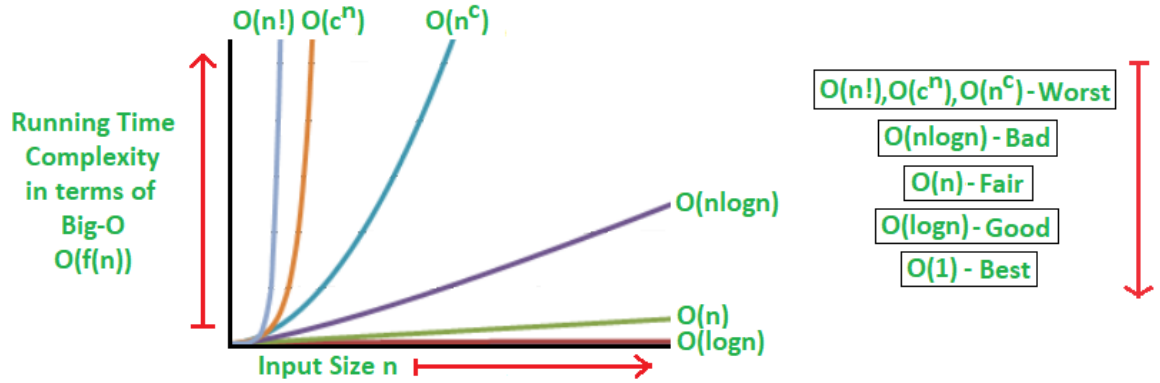
That's because different data structures support different sets of operations. In other words, each of them is suitable for different types of tasks.

So, a programmer should choose the minimal data structure that supports all the operations that is needed.

How can you decide if a program written by you is efficient or not? This is measured by complexities. Complexity is of two types:

1.Time Complexity: Time complexity is used to measure the amount of time required to execute the code.

2.Space Complexity: Space complexity means the amount of space required to execute successfully the functionalities of the code.



Prerequisite:

Before learning how to code in that language you should learn about the building pieces of the language: the basic syntax, the data types, variables, operators, conditional statements, loops (for, if, while, etc), functions, etc.

You may also learn the concept of OOP (Object Oriented Programming).

Lab Assignment: 1

Question:

Happy Numbers

Let the sum of the squares of the digits of a positive integer s0 be represented by s1. In a similar way, let the sum of the squares of the digits of s1 be represented by s2, and so on. If si=1 for some 1≤ i, then the original integer s0 is said to be happy. For example, starting with 7 gives the sequence 7, 49 (=7^2), 97 (=4^2+9^2), 130 (=9^2+7^2), 10 (=1^2+3^2), 1 (=1^2), so 7 is a happy number, which reaches 1 on 6 iterations.

The first few happy numbers are 1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100, …. The number of iterations i required for these to reach 1 are, respectively, 1, 6, 2, 3, 5, 4, 4, 3, 4, 5, 5, 3, ...

A number that is not happy is called unhappy. Once it is known whether a number is happy (unhappy), then any number in the sequence S1, S2, S3, ... will also be happy (unhappy). Unhappy numbers have eventually periodic sequences of Si which do not reach 1 (e.g., 4, 16, 37, 58, 89, 145, 42, 20, 4, ...).

You need to write a program to find all the happy numbers in a given closed interval, which reach 1 within 10 iterations.

Input Specification:
It is a single line input of two positive integers separated by a space.

Output Specification:
Print all happy numbers in the interval and the number of iterations required by it to reach 1, separated by a space and each in a new line.

Code and Output

```
package AssigmentOne;
```

```java
import java.util.Scanner;
public class HappyNumbers {
        // Function to get the sum of squares of digits
    public static int getSum(int n) {
        int sum = 0;
        while (n != 0) {
            int digit = n % 10;
            sum += digit * digit;
            n /= 10;
        }
        return sum;
    }
    // Function to check if a number is happy
    public static boolean isHappy(int n) {
        int slow, fast;
        slow = fast = n;
        do {
            slow = getSum(slow);
            fast = getSum(getSum(fast));
        } while (slow != fast);
        return (slow == 1);
    }
    // Function to print all happy numbers in a range
    public static void printHappyNumbers(int l, int r) {
            System.out.println("HAPPY NUMBERS IN ABOVE RANGE ARE ");
        for (int i = l; i <= r; i++) {
            if (isHappy(i)) {
                System.out.println(i);
            }
        }
    }




    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the lower and upper limit of the range:");
        int l = sc.nextInt();
        int r = sc.nextInt();
        printHappyNumbers(l, r);
    }
}
```

```
Enter the lower and upper limit of the range:
44
68
HAPPY NUMBERS IN ABOVE RANGE
44
49
```

Lab Assignment: 2-1

Question:

An algebraic puzzle is called the Tower of Hanoi. It typically consists of three poles and several discs of various sizes that can slide onto any of the poles. The discs are arranged in a conical form at the beginning of the puzzle, the smallest disc at the top of an orderly stack in one pole.

The goal of the puzzle is to use a third pole (the "auxiliary pole") to move all the discs from one pole, referred to as the "source pole," to another pole, referred to as the "destination pole."

The following two rules apply to the puzzle:

1. A larger disc cannot be stacked on a smaller disc.

2. You can only move one disc at once.

Determine the movement of the discs along 4 towers using a recursive approach.

Solution:

- The program takes the number of discs from the user.
- It uses a recursive method ‘moveDiscs’ to solve the Tower of Hanoi problem.
- The recursive method follows the logic of moving n-1 discs to an auxiliary pole, moving the nth disc to the destination pole, and then moving the n-1 discs from the auxiliary pole to the destination pole.
- The base case of the recursion is when there's only one disc to move, which is done directly.

This is a classic example of a recursive solution to a problem that breaks it down into smaller subproblems.

Here's an explanation of the program:

The main method takes the number of discs as user input and initializes the four poles: source pole, destination pole, auxiliary pole 1, and auxiliary pole 2. It then calls the ‘moveDiscs’ method to start the puzzle.

The ‘moveDiscs’ method is a recursive function that takes five arguments: the number of discs to move, the source pole, the destination pole, auxiliary pole 1, and auxiliary pole 2.

The base case of the recursion is when there is only one disc to move, in which case we simply print the move and return.

Otherwise, we recursively move n-1 discs from the source pole to auxiliary pole 1, using the destination pole and auxiliary pole 2 as auxiliaries. This leaves the largest disc on the source pole.

We then move the largest disc from the source pole to the destination pole.

Finally, we recursively move the n-1 discs that we left on auxiliary pole 1 to the destination pole, using the source pole and auxiliary pole 2 as auxiliaries.

Code and output

```
import java.util.Scanner; // Import the Scanner class to take input from the user
public class TowerOfHanoi {
  public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in); // Create a Scanner object to
read input
      System.out.print("Enter the number of discs: ");

        int numDiscs = scanner.nextInt(); // Read the number of discs from the user
      // Define the labels for the four poles
      char sourcePole = 'A'; // Source pole
      char destinationPole = 'D'; // Destination pole
      char auxPole1 = 'B'; // Auxiliary pole 1
      char auxPole2 = 'C'; // Auxiliary pole 2
      // Call the moveDiscs method to solve the problem
      moveDiscs(numDiscs, sourcePole, destinationPole, auxPole1, auxPole2);
  }

  public static void moveDiscs(int numDiscs, char sourcePole, char destinationPole,
char auxPole1, char auxPole2) {

        // Base case: If there's only one disc, move it directly to the destination pole
      if (numDiscs == 1) {
        System.out.println("Move disc 1 from " + sourcePole + " to " +
destinationPole);
        return; // End the current instance of the method
      }
      // Recursive call to move n-1 discs from sourcePole to auxPole1
      moveDiscs(numDiscs - 1, sourcePole, auxPole1, destinationPole, auxPole2);
      // Move the nth disc from sourcePole to destinationPole
      System.out.println("Move disc " + numDiscs + " from " + sourcePole + " to " +
destinationPole);
// Recursive call to move n-1 discs from auxPole1 to destinationPole
      moveDiscs(numDiscs - 1, auxPole1, destinationPole, sourcePole, auxPole2);
 }}
Enter the number of discs:
3
Move disc 1 from A to D
Move disc 2 from A to B
Move disc 1 from D to B
Move disc 3 from A to D
Move disc 1 from B to A
```

```
Move disc 2 from B to D
Move disc 1 from A to D
```

Overall Complexity:

The time complexity of this algorithm is O(2^n), where n is the number of discs.

Each disc is moved twice (once during the recursive call and once during the print step).

Lab Assignment: 2-2

Question 1:

Recurse to find the suitable substring

Determine the number of sub-strings in a given string having the same starting and ending characters.

Input Specification:

A single line comprising a String (no spaces in-between) of length less than 255 characters. All characters of the string are uniformly either uppercase or lowercase.

Output Specification:

A single line printing a single integer representing the number of sub-strings such as its first and last characters are the same.

Sample Input	Sample Output
madam	7
madamiamadam	29
Abracadabra	23

Code and Output

```java
package assigmentTwoTwo;
import java.util.Scanner;
public class SubstringCount {
  public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     String input = sc.nextLine();
     int n = input.length();
     int count = 0;
     // Iterate through all substrings
     for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
           // Check if the first and last characters are the same
           if (input.charAt(i) == input.charAt(j)) {
              count++;
           }
        }
     }
     System.out.println(count);
  }
}
```

```
madam
7
```

Lab Assignment: 3

Question 1:

Given an array of N integers, write a java program to implement the Selection sort algorithm.

Solution:

The Selection sort algorithm works by maintaining two subarrays in any given array. The subarray is already sorted, and the remaining subarray is unsorted. In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Explanation:

The 'sort' method takes an array as input and sorts it using the selection sort algorithm.

The outer loop iterates n-1 times, where n is the length of the array.

In each iteration, the inner loop finds the minimum element in the unsorted subarray and swaps it with the first element of the unsorted subarray.

The 'printArray' method is used to print the sorted array.

In the 'main' method, we create an instance of the 'SelectionSort' class, create an array, sort it using the 'sort' method, and print the sorted array using the 'printArray' method.

Code and output

```
class SelectionSort {
  // This method sorts an array using selection sort algorithm
  void sort(int array[]) {
    // Get the length of the array
    int n = array.length;
```

```
        // Loop through each element of the array except the last one
        for (int i = 0; i < n - 1; i++) {

            // Assume the current position i is the smallest element
            int min_element = i;

            // Loop through the remaining elements to find the smallest one
            for (int j = i + 1; j < n; j++) {

            // If we find an element smaller than the current smallest, update min_element
                if (array[j] < array[min_element])
                    min_element = j;
            }

            // Swap the smallest element found with the element at position i
            int temp = array[min_element];  // Temporarily store the smallest element
            array[min_element] = array[i];  // Place the element at i into the position of the
smallest element
            array[i] = temp;              // Place the smallest element into the position i
        }
    }
    // This method prints the array elements









  void printArray(int array[]) {

            // Get the length of the array
        int n = array.length;

        // Loop through each element of the array and print it
        for (int i = 0; i < n; ++i)
            System.out.print(array[i] + " ");
        System.out.println();  // Print a new line at the end
    }
  // Main method to test the sorting
    public static void main(String args[]) {
        // Create an instance of the SelectionSort class
        SelectionSort ob = new SelectionSort();
```

```
    // Define an array to be sorted
    int array[] = {64, 25, 12, 22, 11};

    // Call the sort method to sort the array
    ob.sort(array);

    // Print a message indicating the array is sorted
    System.out.println("Sorted array");

    // Call the printArray method to print the sorted array
    ob.printArray(array);
}}
```

```
Sorted array
11 12 22 25 64
```

The time complexity of the selection sort algorithm is O(n^2), and the space complexity is O(1).

Lab Assignment: 3

Question 2:

Given an array of N integers, write a java program to implement the Quick sort algorithm.

Solution:

The Quick sort algorithm is a divide-and-conquer algorithm that works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then recursively sorted.

Explanation:

The ‘partition’ method takes an array, a low index, and a high index as input. It partitions the array around a pivot element, which is the last element of the array in this implementation.

The ‘partition’ method iterates through the array from the low index to the high index. If an element is less than the pivot, it is swapped with the element at the ‘i’ index, and ‘i’ is incremented.

After the iteration, the pivot element is swapped with the element at the i+1 index. This ensures that all elements less than the pivot are on the left of the pivot, and all elements greater than the pivot are on the right.

The ‘quickSort’ method takes an array, a low index, and a high index as input. It recursively calls itself on the sub-arrays on the left and right of the pivot.

The ‘printArray’ method is used to print the sorted array.

In the main method, we create an instance of the ‘QuickSort’ class, create an array, sort it using the ‘quickSort’ method, and print the sorted array using the ‘printArray’ method.

Code and Output

```
class QuickSort {
  // Function to partition the array on the basis of pivot element
  int partition(int array[], int low, int high) {

      // Pivot (Element to be placed at the right position)
    int pivot = array[high];
    int i = (low - 1); // Index of smaller element
    // Traverse elements from low to high-1
    for (int j = low; j < high; j++) {

      // If current element is smaller than the pivot
      if (array[j] < pivot) {
        i++; // Increment index of smaller element
        // Swap array[i] and array[j]
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
      }
    }

      // Swap array[i+1] and array[high] (or pivot)
    int temp = array[i + 1];
    array[i + 1] = array[high];
```

```
        array[high] = temp;
        return i + 1; // Return the partitioning index
    }

        // Function to implement quicksort algorithm
    void quickSort(int array[], int low, int high) {
        if (low < high) {
        // pi is partitioning index, array[pi] is now at right place
        int pi = partition(array, low, high);
        // Recursively sort elements before partition and after partition
        quickSort(array, low, pi - 1);
        quickSort(array, pi + 1, high);
        }
    }

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

        // Main method to test the quicksort algorithm
    public static void main(String args[]) {
        QuickSort ob = new QuickSort();
        int array[] = {10, 7, 8, 9, 1, 5}; // Test array
        int n = array.length;
        ob.quickSort(array, 0, n - 1); // Sort the array
        System.out.println("Sorted array");
        ob.printArray(array); // Print the sorted array
}}
```

```
Sorted array
1 5 7 8 9 10
```

The time complexity of the Quick sort algorithm is O(n log n) on average, although it can be O(n^2) in the worst case if the pivot is chosen poorly. The space complexity is O(log n) due to the recursive calls.

Lab Assignment: 3

Question 3:

Write a program on modified Bubble Sort algorithm to sort a given array and then locate a user input number in the sorted array by using Binary Search algorithm.

Input Specification

• First line of the input is an integer N, representing the number of elements in the

array.

• Second line of the input will comprise of N integers separated by space,

representing the elements of the array.

• Third line of the input will an integer S, which is to be located in the sorted array using Binary Search.

Output Specification

• First line of the output should be the index value representing the location of

integer S in the sorted array.

• Second line of the output should be the number of integer comparisons done for

determining the location of the integer S.

Sample Input	Sample Input
5	6
40 20 30 50 10	5 1 7 4 9 3
40	1
Sample output	**Sample output**
3	0
2	2

Solution:

Bubble Sort:

The bubbleSort method sorts the array by repeatedly swapping adjacent elements if they are in the wrong order. This process is repeated until no swaps are needed, indicating the array is sorted.

Binary Search:

The binarySearch method searches for a specific element in a sorted array by repeatedly dividing the search interval in half. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half.

Printing the Array:

The printArray method prints each element of the array followed by a space.

Main Method:

The main method is the entry point of the program. It reads the array elements and the target element from the user, sorts the array using bubbleSort, prints the sorted array, and then searches for the target element using binarySearch, printing the result.

Exception Handling:

Added a try-catch block to handle InputMismatchException, which is thrown when the user enters input that is not an integer. This ensures the program can handle invalid input without crashing.

If an invalid input is detected, the program will print an error message and not proceed with sorting or searching.

Finally Block:

Added a finally block to close the Scanner object. This ensures the resource is properly released regardless of whether an exception occurs.

How the code works:

The program will prompt the user to enter the number of elements, the elements themselves, and the number to search for in the sorted array.

If the user enters anything other than an integer at any prompt, the program will catch the 'InputMismatchException' and print an error message.

If valid integers are provided, the program will sort the array, print it, and then perform a binary search to find the specified element, displaying the result accordingly.

Code and output

```
import java.util.InputMismatchException;
import java.util.Scanner;
class ModifiedBubbleSort {
  // Function to perform bubble sort on an array
  void bubbleSort(int array[]) {
    int n = array.length; // Get the length of the array
    boolean swapped; // A flag to keep track of swapping
    // Outer loop to traverse through all elements
    for (int i = 0; i < n - 1; i++) {
      swapped = false; // Initialize swapped as false
      // Inner loop to compare adjacent elements
```

```
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j] > array[j + 1]) { // If the element is greater than the next element
                    // Swap array[j] and array[j + 1]
                    int temp = array[j]; // Store the value of array[j] in temp
                    array[j] = array[j + 1]; // Assign the value of array[j + 1] to array[j]
                    array[j + 1] = temp; // Assign the value of temp to array[j + 1]
                    swapped = true; // Set swapped to true indicating a swap occurred
                }
            }
            // If no two elements were swapped in the inner loop, the array is sorted
            if (!swapped) {
                break; // Break out of the loop as the array is already sorted
            }
        }
    }
    // Function to perform binary search on a sorted array
    int binarySearch(int array[], int low, int high, int x) {
        if (high >= low) {
            int mid = (low + high) / 2; // Find the middle index
            // If the element is present at the middle itself
            if (array[mid] == x) {
                return mid; // Return the index
            }
            // If the element is smaller than mid, it can only be present in the left subarray
            if (array[mid] > x) {
                return binarySearch(array, low, mid - 1, x); // Recursively search in the left
subarray
            }
            // Otherwise, the element can only be present in the right subarray
            return binarySearch(array, mid + 1, high, x); // Recursively search in the right
subarray
        }
        // Element is not present in the array
        return -1;
    }
    // Function to print the elements of the array
    void printArray(int array[]) {
        int n = array.length; // Get the length of the array
        for (int i = 0; i < n; ++i) // Loop through the array
            System.out.print(array[i] + " "); // Print each element followed by a space
        System.out.println(); // Print a new line
    }
    // Main method to test the above methods
    public static void main(String args[]) {
```

```
        ModifiedBubbleSort ob = new ModifiedBubbleSort(); // Create an object of ModifiedBubbleSort
        Scanner scanner = new Scanner(System.in); // Create a Scanner object for user input
        try {
            System.out.println("Enter the number of elements in the array:");
            int n = scanner.nextInt(); // Read the number of elements in the array
            int array[] = new int[n]; // Create an array of the specified size
            System.out.println("Enter the elements of the array:");
            for (int i = 0; i < n; i++) { // Loop to read each element of the array
                array[i] = scanner.nextInt(); // Read and store the element in the array
            }
            System.out.println("Enter the number to locate in the sorted array:");
            int x = scanner.nextInt(); // Read the number to be searched in the array
            ob.bubbleSort(array); // Call bubbleSort to sort the array
            System.out.println("Sorted array");
            ob.printArray(array); // Print the sorted array
            int result = ob.binarySearch(array, 0, n - 1, x); // Call binarySearch to find the element
            if (result == -1) { // If the element is not found
                System.out.println("Element not present in array");
            } else { // If the element is found
                System.out.println("Element found at index " + result); // Print the index of the element
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter integers only.");
        } finally {
            scanner.close(); // Close the scanner
        }
    }
}
```

```
Enter the number of elements in the array:
4
Enter the elements of the array:
3
5
2
1
Enter the number to locate in the sorted array:
7
Sorted array
1 2 3 5
```

```
Element not present in array
```

The time complexity of the modified Bubble Sort algorithm is O(n^2) in the worst case, but it can be faster than the standard Bubble Sort algorithm if the array is already partially sorted. The space complexity is O(1) since no additional memory is required.

The time complexity of the Binary Search algorithm is O(log n) since the size of the subarray is halved in each recursive call. The space complexity is O(log n) due to the recursive calls.

Lab Assignment: 4

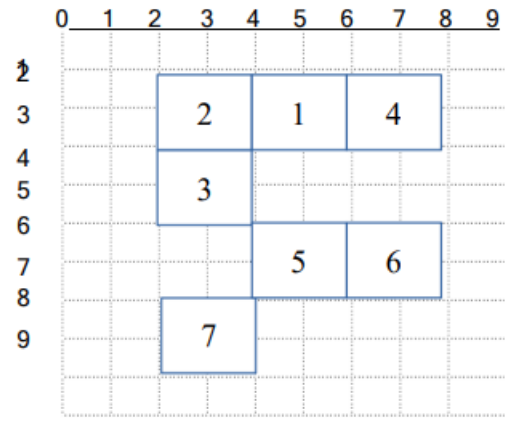
Question 3:

Padosan

A Square is a closed geometric figure with 4 equal sides, and its interior angles all right angles (90°). From this it follows that the opposite sides are parallel.

Consider a grid comprising several identical squares. Squares sharing one of its sides with another are adjacent squares (Padosan). Overlapping squares and squares sharing a single vertex point are not considered adjacent.

In the following figure, square 1 is adjacent to squares 2 & 4, square 2 is adjacent to squares 1 & 3, square 3 is adjacent to square 2, square 4 is adjacent to square 1, square 5 is adjacent to square 6, square 6 is adjacent to square 5 and square 7 is isolated (not adjacent to any of the other squares).



Write a program to determine the number of adjacent squares for a given square.

Input Specification:

1) The first line of the input will contain integer N where N is number of squares (1<=N<=50).

2) The next N lines will contain 8 positive integers in each line, each pair of the integers represents the (x, y) coordinates of one of the vertices of Nth square.

Output specification:

On each line print the square number and the number of the squares adjacent to it, separated by a space, for each square starting from square no. 1 to square no. N terminated by new line character.

Sample 1		Sample 2	
Sample Input - 1	**Sample Output - 1**	**Sample Input – 2**	**Sample Output - 2**
7 11313313 31515333 51717353 13333515 17373919 37575939 54747656	1 2 2 2 3 1 4 1 5 1 6 1 7 0	4 11313313 31515333 13333515 33535535	1 2 2 2 3 2 4 2

Code and Output

```
import java.util.Scanner;
//Define a class to represent a square
class Square {
int x1, y1, x2, y2, x3, y3, x4, y4;
// Constructor to initialize the square with its vertices
public Square(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
    this.x1 = x1;
    this.y1 = y1;
    this.x2 = x2;
    this.y2 = y2;
    this.x3 = x3;
    this.y3 = y3;
    this.x4 = x4;
    this.y4 = y4;
}
// Method to check if two squares are adjacent
public boolean isAdjacent(Square s) {
    // Two squares are adjacent if they share a side, i.e., their x-coordinates or y-
coordinates are the same and the other coordinates differ by 1
    return (this.x1 == s.x1 && this.x2 == s.x2 && Math.abs(this.y1 - s.y1) == 1) ||
           (this.y1 == s.y1 && this.y2 == s.y2 && Math.abs(this.x1 - s.x1) == 1);
}
}
public class Padosan {
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int N = sc.nextInt();  // Number of squares
    Square[] squares = new Square[N];  // Array to store the squares
    // Read the input squares
    for (int i = 0; i < N; i++) {
        squares[i] = new Square(sc.nextInt(), sc.nextInt(), sc.nextInt(), sc.nextInt(),
                                sc.nextInt(), sc.nextInt(), sc.nextInt(), sc.nextInt());
    }
    // For each square, count the number of adjacent squares
    for (int i = 0; i < N; i++) {
        int count = 0;
        for (int j = 0; j < N; j++) {
            // Check if the squares are different and adjacent
            if (i != j && squares[i].isAdjacent(squares[j])) {
                count++;
            }
        }
```

```
        // Print the square number and the number of adjacent squares
        System.out.println((i + 1) + " " + count);
    }
    sc.close();
}
```

```
2
0 0 1 0 1 1 0 1
2 2 3 2 3 3 2 3
1 0
2 0
```

Lab Assignment: 4

Question 2:

ShabdKosh

You have to write a program to chain some words. A word is properly chained if it starts with a trailing sub-string of its predecessor word with a minimum overlap of three (3) characters.

Given a number of words, you have to reorder them to appropriately chain them. The first word in the input is used as a starting word in the chain. It may happen that there is no chaining possible for a given set of words. If chaining is possible, assume that there will be a unique word chain.

Note: A word is a sequence of alphabetic characters.

Input Specification

The first line will be an integer N, indicating the number of words that will follow. Assume N will never be greater than twenty (20).

The next N lines of input will contain words, which are to be chained. Assume that the maximum length of a word will never exceed thirty (30) characters.

Output Specification

Your program should output the chain of words, one word on a separate line. If there is no chain possible from the given words, the program should print IMPOSSIBLE.

Sample 1		**Sample 2**	
Input 1	***Output 1***	***Input 2***	***Output 2***
2 start finish	IMPOSSIBLE	8 whisper format perform sonnet person shopper workshop network	whisper person sonnet network workshop shopper perform format

Code and Output

```
package assigmentFour;
import java.util.*;
public class WordChain {
   private static String[] words;
   private static boolean[] visited;
   private static String[] result;
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      int N = scanner.nextInt();
      words = new String[N];
      visited = new boolean[N];
      result = new String[N];
      for (int i = 0; i < N; i++) {
         words[i] = scanner.next();
      }
      result[0] = words[0];
      visited[0] = true;
      if (findChain(1)) {
         for (String word : result) {
            System.out.println(word);
         }
      } else {
         System.out.println("IMPOSSIBLE");
      }
   }
   private static boolean findChain(int index) {
      if (index == words.length) {
         return true;
      }
      for (int i = 0; i < words.length; i++) {
         if (!visited[i] && overlap(result[index - 1], words[i])) {
            result[index] = words[i];
            visited[i] = true;
            if (findChain(index + 1)) {
               return true;
            }
            visited[i] = false;
         }
      }
      return false;
```

```
    }
    private static boolean overlap(String a, String b) {
        int len = Math.min(a.length(), b.length());
        for (int i = 3; i <= len; i++) {
            if (a.substring(a.length() - i).equals(b.substring(0, i))) {
                return true;
            }
        }
        return false;
    }
}
```

```
2
Start
Finnish
IMPOSSIBLE
```

Practice Questions

Question 1:

Write a program in Java to Reverse a Singly Linked List

Code and Output

```
public class RevLinkedList {
	static SinglyLinkedNode head;
	public void altNode(int data) {
		SinglyLinkedNode newNode=new SinglyLinkedNode(data);
		if(head==null) {
			head=newNode;
			return;
		}
		SinglyLinkedNode temp=head;
		newNode.next=temp;
		head=newNode;
		return;

	}
	public void reverseLinkedList() {
		if(head==null || head.next==null) {
			return;
		}
		SinglyLinkedNode prevNode=head;
		SinglyLinkedNode currNode=head.next;
		while(currNode != null) {
			SinglyLinkedNode nextNode=currNode.next;
			currNode.next=prevNode;

			prevNode=currNode;
			currNode=nextNode;
		}
		head.next=null;
		head=prevNode;

	}
	public void disp() {
		SinglyLinkedNode temp=head;
		while (temp!=null) {
			System.out.print(temp.data+" ");
			temp=temp.next;
		}
		System.out.println();
	}
	public static void main(String[] args) {
		RevLinkedList s=new RevLinkedList();
		head=new SinglyLinkedNode(10);
		s.altNode(20);
		s.altNode(30);
```

```
                s.altNode(56);
                s.disp();
                s.reverseLinkedList();
                s.disp();


        }

}
class SinglyLinkedNode{
        int data;
        SinglyLinkedNode next;
        SinglyLinkedNode (int data){
                this.data=data;
                next=null;
        }
}
```

Output

```
56 30 20 10
10 20 30 56
```

Question 2:

Write a program in Java to print the alternate node in the given Singly Linked List.

Code and Output

```
class Node {
   int data;
   Node next;
   Node(int data) {
      this.data = data;
      this.next = null;
   }
}
public class AlternateNodesLinkedList {
   public static void printAlternateNodes(Node head) {
      while (head != null) {
         System.out.print(head.data + " ");
         if (head.next != null) {
            head = head.next.next;
         }
         else {
            head = head.next;
         }
```

```
        }
    }
    public static void main(String[] args) {
        // Example usage:
        Node head = new Node(8);
        head.next = new Node(23);
        head.next.next = new Node(11);
        head.next.next.next = new Node(29);
        head.next.next.next.next = new Node(12);
        System.out.print("Alternate nodes: ");
        printAlternateNodes(head);
    }
}
```

Output:

```
Alternate nodes: 8 11 12
```