1. Problem Statement

When a subclass inherits from a superclass, it also inherits its methods; however, it can also override the superclass methods (as well as declare and implement new ones). Consider the following Sports class:

```
class Sports{
   String getName(){
      return "Generic Sports";
   }
   void getNumberOfTeamMembers(){
      System.out.println( "Each team has n players in " + getName() );
   }
}
```

Next, we create a Soccer class that inherits from the Sports class. We can override the getName method and return a different, subclass-specific string:

```
class Soccer extends Sports{
   @Override
   String getName(){
      return "Soccer Class";
   }
}
```

Note: When overriding a method, you should precede it with the @Override annotation. The parameter(s) and return type of an overridden method must be exactly the same as those of the method inherited from the supertype.

Task

Complete the code in your editor by writing an overridden getNumberOfTeamMembers method that prints the same statement as the superclass' getNumberOfTeamMembers method, except that it replaces with (the number of players on a Soccer team).

Code Snippet

```
import java.util.*;
class Sports{
   String getName(){
      return "Generic Sports";
   }

   void getNumberOfTeamMembers(){
      System.out.println( "Each team has n players in " + getName() );
   }
}
class Soccer extends Sports{
   @Override
   String getName(){
      return "Soccer Class";
   }
  // add override here
}

public class Solution{

   public static void main(String []args){
      Sports c1 = new Sports();
      Soccer c2 = new Soccer();
      System.out.println(c1.getName());
      c1.getNumberOfTeamMembers();
      System.out.println(c2.getName());
      c2.getNumberOfTeamMembers();
   }
}
```

2. Problem Statement

Given a Book class and a Solution class, write a MyBook class that does the following:

- Inherits from Book
- Has a parameterized constructor taking these parameters:
    - string title
    - string author
    - int price

Implements the Book class abstract display() method so it prints these lines:

1. Title:, a space, and then the current instance's .
2. Author:, a space, and then the current instance's .
3. Price:, a space, and then the current instance's .

Note: Because these classes are being written in the same file, you must not use an access modifier (e.g.:) when declaring MyBook or your code will not execute.

Input Format

- You are not responsible for reading any input from stdin. The Solution class creates a Book object and calls the MyBook class.

- Constructor (passing it the necessary arguments). It then calls the display method on the Book object.

Output Format

The void display() method should print and label the respective title,author, and price of the MyBook object's instance (with each value on its own line) like so:

```
Title:     $title
Author: $author
Price:    $price
```

Note:- The $ is prepended to variable names to indicate they are placeholders for variables.

Sample Input

```
The Alchemist
Paulo Coelho
248
```

Sample Output

The following output is printed by your display() method:

```
Title: The Alchemist
Author: Paulo Coelho
```

Price: 248

Code Snippet

```
import java.util.*;
abstract class Book {
   String title;
   String author;

   Book(String title, String author) {
      this.title = title;
      this.author = author;
   }

   abstract void display();
}

// Declare your class here. Do not use the 'public' access modifier.
   // Declare the price instance variable

   /**
   *   Class Constructor
   *
   *   @param title The book's title.
   *   @param author The book's author.
   *   @param price The book's price.
   **/
   // Write your constructor here

   /**
   *   Method Name: display
   *
   *   Print the title, author, and price in the specified format.
   **/
   // Write your method here

// End class

public class Solution {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      String title = scanner.nextLine();
      String author = scanner.nextLine();
```

```
        int price = scanner.nextInt();
        scanner.close();

        Book book = new MyBook(title, author, price);
        book.display();
    }
}
```

3. Problem Statement

Write the following methods that return a lambda expression performing a specified action:

1. PerformOperation *isOdd()*: The lambda expression must return true if a number is odd or false if it is even.
2. PerformOperation *isPrime()*: The lambda expression must return true if a number is prime or false if it is composite.
3. PerformOperation *isPalindrome()*: The lambda expression must return true if a number is a palindrome or false if it is not.

Sample Input

- The first line contains an integer,N(the number of test cases).
- The N subsequent lines each describe a test case in the form of 2 space-separated integers:
- The first integer specifies the condition to check for (1 for Odd/Even,2 for Prime, or 3 for Palindrome).

- The second integer denotes the number to be checked.

```
5
1 4
2 5
3 898
1 3
2 12
```

Sample Output

```
EVEN
PRIME
PALINDROME
ODD
```

Code Snippet

```
import java.io.*;
import java.util.*;
interface PerformOperation {
 boolean check(int a);
}
class MyMath {
 public static boolean checker(PerformOperation p, int num) {
  return p.check(num);
 }
  public static boolean isPrime(int a){
       for(int i = 2; i <= Math.sqrt(a); i++)
                if(a % i == 0)
                    return false;
        return true;

    }
    public static  PerformOperation isOdd(){
         //write your code
    }
    public static PerformOperation isPrime(){
         //write your code

    }
    public static PerformOperation isPalindrome(){
        //write your code

    }
    }
public class Solution {

 public static void main(String[] args) throws IOException {
  MyMath ob = new MyMath();
  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  int T = Integer.parseInt(br.readLine());
  PerformOperation op;
  boolean ret = false;
  String ans = null;
  while (T--> 0) {
   String s = br.readLine().trim();
   StringTokenizer st = new StringTokenizer(s);
   int ch = Integer.parseInt(st.nextToken());
   int num = Integer.parseInt(st.nextToken());
   if (ch == 1) {
    op = ob.isOdd();
    ret = ob.checker(op, num);
    ans = (ret) ? "ODD" : "EVEN";
   } else if (ch == 2) {
    op = ob.isPrime();
    ret = ob.checker(op, num);
    ans = (ret) ? "PRIME" : "COMPOSITE";
   } else if (ch == 3) {
    op = ob.isPalindrome();
    ret = ob.checker(op, num);
    ans = (ret) ? "PALINDROME" : "NOT PALINDROME";

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

4. Problem Statement

A left rotation operation on an array shifts each of the array's elements unit to the left. For example, if left rotations are performed on an array ,then the array would become . Note that the lowest index item moves to the highest index in a rotation. This is called a circular array.

Given an array ‘a’ of ‘n’ integers and a number ,d , perform d left rotations on the array. Return the updated array to be printed as a single line
of space-separated integers.

Function Description

Complete the function rotLeft in the editor below.

rotLeft has the following parameter(s):

int a[n]: the array to rotate
int d: the number of rotations

Returns

int a'[n]: the rotated array

Input Format

- The first line contains two space-separated integers n and d , the size of a and the number of left rotations.
- The second line contains n space-separated integers, each an a[i].

Constraints

1<=n<=10 power 5
1<=d<=n
1<=a[i]<=10 power 6

Sample Input

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

Sample Output

```
5 1 2 3 4
```

Explanation

When we perform d=4 left rotations, the array undergoes the following sequence of changes:

, [1,2,3,4,5] -> [2,3,4,5,1] -> [3,4,5,1,2] -> [4,5,1,2,3]

Code Snippet

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

class Result {

    /*
     * Complete the 'rotLeft' function below.
     *
     * The function is expected to return an INTEGER_ARRAY.
     * The function accepts following parameters:
     *  1. INTEGER_ARRAY a
     *  2. INTEGER d
     */

    public static List<Integer> rotLeft(List<Integer> a, int d) {
    // Write your code here
    }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new
FileWriter(System.getenv("OUTPUT_PATH")));

        String[] firstMultipleInput = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        int n = Integer.parseInt(firstMultipleInput[0]);

        int d = Integer.parseInt(firstMultipleInput[1]);

        String[] aTemp = bufferedReader.readLine().replaceAll("\\s+$", "").split(" ");

        List<Integer> a = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            int aItem = Integer.parseInt(aTemp[i]);
            a.add(aItem);
        }

        List<Integer> result = Result.rotLeft(a, d);

        for (int i = 0; i < result.size(); i++) {
            bufferedWriter.write(String.valueOf(result.get(i)));

            if (i != result.size() - 1) {
                bufferedWriter.write(" ");
            }
        }

        bufferedWriter.newLine();
        bufferedReader.close();
        bufferedWriter.close();
}
}
```

5.Problem statement

You are working on a feature to distribute memory in the memory blocks. You have a memory block of size *N.* You can divide the memory block in a way that fulfills the following two conditions:

- After the division, each memory block should have lengths *A*, *B,* or *C*.
- After the division, the number of memory blocks should be maximum.

You are given integers *N, A, B,* and *C.*

Task

Print the maximum possible number of memory blocks. It is guaranteed that at least one correct memory division exists.

Example

Assumptions

- *N = 7*
- *A = 5*
- *B = 5*
- *C = 2*

Approach

- You can divide the memory block in such a way: The first block has a length 5, and the second block has a length 2. The answer is 2.

Function description

Complete the function Memory*()* which takes an integer *N,* integer *A*, integer *B,* and an integer *C.* This function takes the following parameters and returns the required answer:

- *N:* Represents the size of the block

- *A:* Represents the first valid division length
- *B:* Represents the second valid division length
- *C:* Represents the third valid division length

Input format

Note: This is the input format you must use to provide custom input (available above the Compile and Test button).

- The first line contains an integer *N* denoting the size of the memory block.
- The second line contains an integer *A* denoting the first valid division length.
- The third line contains an integer *B* denoting the second valid division length.
- The fourth line contains an integer C denoting the third valid division length.

Output format

Print the maximum possible number of memory blocks. It is guaranteed that at least one correct memory division exists.

Constraints

$1 \leq N, A, B, C \leq 4000$

IO example

Sample input

```
5
5
3
2
```

Sample output

```
2
```

Explanation

You can divide the memory block in such a way: The first block has length 2, and the second block has length 3. The answer is 2.

Code Snippet

```
import java.io.*;
import java.util.*;
public class TestClass {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        PrintWriter wr = new PrintWriter(System.out);
         int N = Integer.parseInt(br.readLine().trim());
         int A = Integer.parseInt(br.readLine().trim());
         int B = Integer.parseInt(br.readLine().trim());
         int C = Integer.parseInt(br.readLine().trim());

         int out_ = Memory(N, A, B, C);
         System.out.println(out_);

         wr.close();
         br.close();
    }
    static int Memory(int N, int A, int B, int C){
       // Write your code here
        int result = 0;

        return result;

    }
}
```

6 MCQ

1) What is bean in Spring?
   a) Component
   b) An Object
   c) A Class
   d) A Container

2) AOP Stands for?
   a) Asynchronous Oriented Programming
   b) Aspect Oriented Programming
   c) All OOps Program

3) What is Dependency Injection?
   a) It's an Inversion of Control for software applications.
   b) One of Spring Module
   c) A technique to get dependency of any project
   d) Used to promote tight coupling in code

4) What is the protocol used in REST?
   a) FT0P
   b) HTTP
   c) JMX
   d) SOAP

5) Lambdas introduced in Java 8 allow us to treat -

   a) Data as Code
   b) Code as data
   c) None
   d) all

6) In Java 8 Interfaces, methods can be:

   a) Default
   b) abstract
   c) all
   d) none

7) Which interface restricts duplicate elements?

   a) Set
   b) List
   c) Map
   d) All of these

8) The Comparable interface contains which called?

a) Compare
b) toCompare
c) compareTo
d) compareWith

9) The default capacity of a ArrayList is:

a) 12
b) 16
c) 1
d) All of the above

10) Which of these classes is a superclass of the String and StringBuffer class?

a) Java.util
b) java.lang
c) ArrayList
d) None of the Above