Cypher 30 week 2 solutions

Mystery and repeating:

```
import java.util.Scanner;


public class Solution {


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


        // Read N
        int n = sc.nextInt();
        int[] arr = new int[n];


        // To track seen numbers
        boolean[] seen = new boolean[n + 1];
 // Indexing from 1 to N
        int duplicate = -1;
        long sum = 0;
        long expectedSum = (long) n * (n + 1) / 2;


        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
            sum += arr[i];
            if (seen[arr[i]]) {
                duplicate = arr[i];
            } else {
                seen[arr[i]] = true;
            }
        }
```

```
        int missing = (int) (expectedSum - (sum - duplicate));
        System.out.println(duplicate + "," + missing);

        sc.close();
    }
}
```

Hidden Assasin:

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

public class Solution {

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

        // Read input values
        int n = sc.nextInt(); // size of the array
        int k = sc.nextInt(); // required difference

        int[] arr = new int[n];
        HashSet<Integer> set = new HashSet<Integer>();

        // Read array elements
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
            set.add(arr[i]);
        }
```

```
        // Check if any pair has the absolute difference k
        boolean found = false;
        for (int i = 0; i < n; i++) {
            if (set.contains(arr[i] + k) || set.contains(arr[i] - k)) {
                found = true;
                break;
            }
        }

        if (found) {
            System.out.println("YES");
        } else {
            System.out.println("NO");
        }

        sc.close();
    }
}
```

Divide and paint :

```
import java.io.*;
import java.util.*;

public class Solution {
    private static boolean isPossible(int[] boards, int k, int maxWork) {


        int painters = 1;
        int currentSum = 0;

        for (int length : boards) {
```

```
            if (length > maxWork) return false; // A single board is too long
            if (currentSum + length <= maxWork) {
                currentSum += length;
            } else {
                painters++;
                currentSum = length;
                if (painters > k) return false;
            }
        }

        return true;
    }

    // Binary search to find the minimum time
    public static int minTimeToPaint(int[] boards, int k) {
        int low = Arrays.stream(boards).max().getAsInt();
        int high = Arrays.stream(boards).sum();
        int result = high;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (isPossible(boards, k, mid)) {
                result = mid;
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }

        return result;
    }
```

```
    // Main method to handle multiple test cases
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int T = sc.nextInt();  // Number of test cases

        while (T-- > 0) {
            int N = sc.nextInt(); // Number of boards
            int k = sc.nextInt(); // Number of painters
            int[] boards = new int[N];

            for (int i = 0; i < N; i++) {
                boards[i] = sc.nextInt();
            }

            System.out.println(minTimeToPaint(boards, k));
        }

        sc.close();


    }
}
```

Ravi and Snakes:

```
import java.io.*;
import java.util.*;

public class Solution {

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

```
 Scanner sc = new Scanner(System.in);
int T = sc.nextInt(); // Number of test cases

while (T-- > 0) {
    int N = sc.nextInt(); // Number of snakes
    int Q = sc.nextInt(); // Number of queries

    int[] snakes = new int[N];
    for (int i = 0; i < N; i++) {
        snakes[i] = sc.nextInt();
    }

    Arrays.sort(snakes); // Sort snake lengths

    // Precompute prefix sum for smaller snakes (for feeding cost)
    long[] prefixSum = new long[N];
    prefixSum[0] = snakes[0];
    for (int i = 1; i < N; i++) {
        prefixSum[i] = prefixSum[i - 1] + snakes[i];
    }

    for (int q = 0; q < Q; q++) {
        int K = sc.nextInt();

        // Binary search to find minimal index i where snakes[i] >= K
        int left = 0, right = N - 1, firstBigIndex = N;
        while (left <= right) {
            int mid = (left + right) / 2;
            if (snakes[mid] >= K) {
                firstBigIndex = mid;
                right = mid - 1;
```

```
            } else {
                left = mid + 1;
            }
        }

        int alreadyBig = N - firstBigIndex;
        int maxCount = alreadyBig;

        // Now try to upgrade some smaller snakes to reach K
        int low = 0, high = firstBigIndex - 1;
        int canUpgrade = 0;

        while (low <= high) {
            int mid = (low + high) / 2;
            int numToUpgrade = firstBigIndex - mid;
            long currentSum = prefixSum[firstBigIndex - 1];
            if (mid > 0) currentSum -= prefixSum[mid - 1];

            long requiredSum = (long) K * numToUpgrade;
            long needed = requiredSum - currentSum;
            if (mid >= needed) {
                canUpgrade = numToUpgrade;
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }

        System.out.println(alreadyBig + canUpgrade);
    }
}
```

```
        sc.close();

    }
}
```

Dino Problem :

```
import java.util.*;

public class Main {

    public static List<List<Integer>> findTripletsWithZeroSum(int[] A) {
        Arrays.sort(A);  // Sort the array
        int n = A.length;
        List<List<Integer>> triplets = new ArrayList<>();

        for (int i = 0; i < n - 2; i++) {
            // Skip duplicates
            if (i > 0 && A[i] == A[i - 1]) {
                continue;
            }

            int left = i + 1, right = n - 1;

            while (left < right) {
                int currentSum = A[i] + A[left] + A[right];

                if (currentSum == 0) {
                    triplets.add(Arrays.asList(A[i], A[left], A[right]));

                    // Skip duplicates for left and right
                    while (left < right && A[left] == A[left + 1]) {
```

```
                left++;
            }
            while (left < right && A[right] == A[right - 1]) {
                right--;
            }

            left++;
            right--;
        } else if (currentSum < 0) {
            left++;
        } else {
            right--;
        }
      }
    }

    return triplets;
}

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

    int N = sc.nextInt();  // Read number of elements
    int[] A = new int[N];

    for (int i = 0; i < N; i++) {
        A[i] = sc.nextInt();  // Read the elements of the array
    }

    List<List<Integer>> triplets = findTripletsWithZeroSum(A);
```

```
        for (List<Integer> triplet : triplets) {
            System.out.print("Triplet:[");
            for (int i = 0; i < triplet.size(); i++) {
                System.out.print(triplet.get(i));
                if (i < triplet.size() - 1) {
                    System.out.print(", ");
                }
            }
            System.out.println("]");
        }

        sc.close();
    }
}
```