Q.1.

```java
package arraysAndStrings;
import java.util.Scanner;
public class Q1ReverseArray {

	public static void reverseArray(int[] arr) {
		int n = arr.length;
		for(int i = 0, j = n-1; i < j; ++i, --j) {
			int temp = arr[i];
			arr[i] = arr[j];
			arr[j] = temp;
		}
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner s = new Scanner(System.in);
		int n = s.nextInt();
		int[] arr = new int[n];
		for(int i =0; i < n; ++i)
			arr[i] = s.nextInt();
		reverseArray(arr);
		System.out.println("Reversed array is: ");
		for(int i = 0; i < n; ++i)
			System.out.print(arr[i] + " ");
	}

}
```

Main logic here

Q.2.

main body:

```
package arraysAndStrings;

import java.util.Scanner;
import java.util.Arrays;

public class Q2FindUniqueElement {

	//functions here

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner s = new Scanner(System.in);
		int n = s.nextInt();
		int[] arr = new int[n];
		for(int i =0; i < n; ++i)
			arr[i] = s.nextInt();
		System.out.println(uniqueElement(arr));
	}

}
```

Solution1: Brute force approach – $O(n^2)$

```
public static int uniqueElement(int[] arr) {
		int n = arr.length;
		for(int i = 0; i < n; ++i) {
			boolean found = false;
			for(int j = 0; j < n; ++j) {
				if(i != j && arr[i] == arr[j]) {
					found = true;
					break;
				}
			}
			if(!found)
				return arr[i];
		}
		return -1;
	}
```

Solution2: Sorting the array first – O(nlogn)

```
public static int uniqueElement(int[] arr) {
            Arrays.sort(arr);
            int n = arr.length;
            for(int i = 0; i < n-1; ) {
                  if(arr[i] == arr[i+1])
                        i += 2;
                  else
                        return arr[i];
            }
            return arr[n-1];
      }
```

Solution3: Using Bit Manipulation (XOR) – O(n)

```
public static int uniqueElement(int[] arr) {
            int ans = 0;
            for(int i =0; i < arr.length; ++i)
                  ans = ans ^ arr[i];
            return ans;
      }
```

Q.3.

Solution1: Using the usual sorting

Time Complexity:

O(n2) if bubble/insertion/selection sort is used
O(nlogn) if inbuilt function or Merge/Quick/Heap sort is used

```
public static void sort01(int[] arr) {
            Arrays.sort(arr);
      }
```

Solution2: By counting the number of zeros (Two array traversals required) – Time Complexity – O(n)

```
public static void sort01(int[] arr) {
            int n = arr.length;
            int count0 = 0;
            for(int i = 0; i < n; ++i)
                  if(arr[i] == 0)
                        count0++;
            int i = 0;
            for(; i < count0; ++i)
                  arr[i] = 0;
            for(; i < n; ++i)
                  arr[i] = 1;
      }
```

Solution3: Two pointer technique (Only one array traversal required) – Time Complexity – O(n)

```
public static void sort01(int[] arr) {
        int i = 0, j = arr.length - 1;
        while(i < j) {
            if(arr[i] == 1 && arr[j] == 0) {
                arr[i] = 0;
                arr[j] = 1;
                ++i;
                --j;
            } else if (arr[i] == 0)
                ++i;
            else if (arr[j] == 1)
                --j;
            else {
                ++i;
                --j;
            }
        }
    }
```