Assignment No.1

Q1.Answer

```
import java.util.HashMap;

import java.util.Map;

public class TwoSum {

    public int[] twoSum(int[] nums, int target) {

        Map<Integer, Integer> numMap = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {

            int complement = target - nums[i];

            if (numMap.containsKey(complement)) {

                return new int[]{numMap.get(complement), i};

            }

            numMap.put(nums[i], i);

        }

        throw new IllegalArgumentException("No solution found.");

    }

}
```

Q.2Answer :

```
public class RemoveElement {
    public int removeElement(int[] nums, int val) {
        int k = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] ! =  val ) {
                nums[k] = nums[i];
                k++;
            }
        }
        return k;
    }}
```

Q.3 Answer

```
public class SearchInsert {
    public int searchInsert(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return left;
    }
}
```

Q.4.Answer:

```
public class PlusOne {
    public int[] plusOne(int[] digits) {
        int n = digits.length;
        for (int i = n - 1; i >= 0; i--) {
            if (digits[i] < 9) {
                digits[i]++;
                return digits;
            }
            digits[i] = 0;
        }
        int[] newDigits = new int[n + 1];
        newDigits[0] = 1;
        return newDigits;
    }
}
```

Q.5Answer:

```
public class MergeSortedArray {
   public void merge(int[] nums1, int m, int[] nums2, int n) {
      int index1 = m - 1;
      int index2 = n - 1;
      int mergedIndex = m + n – 1 ;

      // Merge nums1 and nums2 from the back
      while (index1 >= 0 && index2 >= 0) {

         if (nums1[index1] > nums2[index2]) {
            nums1 [mergedIndex] = nums1[index1];
            index1-- ;
         } else {
            nums1 [mergedIndex] = nums2[index2];
            index2-- ;
         }
         mergedIndex--;
      }
      while (index2 > = 0) {
         nums1 [mergedIndex] = nums2[index2];
         index2--;
         mergedIndex - - ;
      }
   }
}
```

Q.6 Answer

```
import java.util.HashSet;
import java.util.Set;

public class ContainsDuplicate {
    public boolean containsDuplicate(int[] nums) {
        Set<Integer> numSet = new HashSet<>();
        for (int num : nums) {
            if (numSet.contains(num)) {
                return true;
            }
            numSet.add(num);
        }
        return false;
    }
}
```

Q.7.Answer:

```
public class MoveZeroes {
    public void moveZeroes(int[] nums) {
        int index = 0; // Index for placing nonzero elements

        // Move nonzero elements to the front of the array
        for (int num : nums) {
            if (num != 0) {
                nums[index] = num;
                index++;
            }
        }

        // Fill the remaining positions with zeros
        while (index < nums.length) {
            nums[index] = 0;
            index++;
        }
    }
}
```

Q.8. Answer:

```
import java.util.HashSet;
import java.util.Set;

public class FindErrorNums {
   public int[] findErrorNums(int[] nums) {
      int n = nums.length;
      int duplicate = -1;
      int missing = -1;

      Set<Integer> numSet = new HashSet<>();
      for (int num : nums) {
         if (numSet.contains(num)) {
            duplicate = num;
         }
         numSet.add(num);
      }

      for (int i = 1; i <= n; i++) {
         if (! numSet .contains(i)) {
            missing = i;
            break;
         }
      }

      return new int[]{duplicate, missing};
   }
}
```