1. Two Sum

Using HashMap: Time: 0(1) Space: O(N)

```
public int[] twoSum(int[] nums, int target) {
    HashMap<Integer,Integer> map = new HashMap<>();
    int[] output = new int[2];
    for(int i = 0; i < nums.length; i++){
        map.put(nums[i], i);
    }
    for(int i = 0; i < nums.length; i++){
        int rem = target - nums[i];
        if(map.containsKey(rem) && map.get(rem) > i){
            output[0] = i;
            output[1] = map.get(rem);
            break;
        }
    }
    return output;
}
```

2. Two Sum-II

```
public int[] twoSum(int[] numbers, int target) {
    int left = 0;
    int right = numbers.length - 1;
    int[] op = new int[2];
    while (left < right) {
        int sum = numbers[left] + numbers[right];
        if (sum > target) {
            right--;
        } else if (sum < target) {
            left++;
        } else {
            op[0] = left + 1;
            op[1] = right + 1;
            break;
        }
    }
    return op;
}
```

3. Merge Sorted Array

App1: Time: O(m lg m)

```
public void merge(int[] nums1, int m, int[] nums2, int n) {
    int start = 0;
    int end = nums1.length - 1;
    while (start < end) {
        int temp = nums1[start];
        nums1[start] = nums1[end];
        nums1[end] = temp;
        start++;
        end--;
    }
    if (n > 0 && m > 0) {
        for (int i = 0; i < n; i++) {
            nums1[i] = nums2[i];
        }
        Arrays.sort(nums1);
    } else {
        if (n > 0 && m < 1) {
            for (int i = 0; i < n; i++) {
                nums1[i] = nums2[i];
            }
        } else {
            Arrays.sort(nums1);
        }

    }
}
```

App2: O(n+m)

```
1  class Solution {
2      public void merge(int[] nums1, int m, int[] nums2, int n) {
3          int i = m - 1;
4          int j = n - 1;
5          int k = nums1.length - 1;
6
7          while (i >= 0 && j >= 0) {
8              if (nums1[i] > nums2[j]) {
9                  nums1[k] = nums1[i];
10                 k--;
11                 i--;
12             } else {
13                 nums1[k] = nums2[j];
14                 j--;
15                 k--;
16             }
17         }
18
19         while(j >= 0){
20             nums1[k] = nums2[j];
21             k--;
22             j--;
23         }
24     }
25 }
```

4. Pascal's Triangle

```
public List<List<Integer>> generate(int numRows) {
    List<List<Integer>> triangle = new ArrayList<>();
    if (numRows <= 0) {
        return triangle;
    }
    List<Integer> firstRow = new ArrayList<>();
    firstRow.add(1);
    triangle.add(firstRow);

    for (int i = 1; i < numRows; i++) {
        List<Integer> prevRow = triangle.get(i - 1);
        List<Integer> newRow = new ArrayList<>();
        newRow.add(1);
        for (int j = 1; j < i; j++) {
            newRow.add(prevRow.get(j - 1) + prevRow.get(j));
        }
        newRow.add(1);
        triangle.add(newRow);
    }

    return triangle;
}
```

5. Pascal's Triangle- II

```
public List<Integer> getRow(int rowIndex) {
    List<List<Integer>> pascal = generate(rowIndex + 1);

    return pascal.get(rowIndex);
}
public List<List<Integer>> generate(int numRows) {
    List<List<Integer>> triangle = new ArrayList<>();
    if (numRows <= 0) {
        return triangle;
    }
    List<Integer> firstRow = new ArrayList<>();
    firstRow.add(1);
    triangle.add(firstRow);

    for (int i = 1; i < numRows; i++) {
        List<Integer> prevRow = triangle.get(i - 1);
        List<Integer> newRow = new ArrayList<>();
        newRow.add(1);
        for (int j = 1; j < i; j++) {
            newRow.add(prevRow.get(j - 1) + prevRow.get(j));
        }
        newRow.add(1);
        triangle.add(newRow);
    }

    return triangle;
}
```

6. **Best Time to Buy and Sell Stock**

```
public int maxProfit(int[] prices) {
    int profit = 0;
    int minPrice = prices[0];
    for (int i = 1; i < prices.length; i++) {
        if (prices[i] < minPrice) {
            minPrice = prices[i];
            continue;
        } else {
            int newProfit = prices[i] - minPrice;
            if (newProfit > profit) {
                profit = newProfit;
            }
        }
    }
    return profit;
}
```

7. Best Time to Buy and Sell Stock II

```
public int maxProfit(int[] prices) {
    int profit = 0;
    for (int i = 1; i < prices.length; i++) {
        if (prices[i] > prices[i - 1])
            profit += prices[i] - prices[i - 1];
    }
    return profit;
}
```

8. Majority Element

```java
public int majorityElement(int[] nums) {
    int count = 1;
    int majority = nums[0];
    for (int i = 1; i < nums.length; i++) {
        if (majority == nums[i]) {
            count++;
        } else {
            count--;
            if (count == 0) {
                majority = nums[i];
                count++;
            }
        }
    }
    return majority;
```

9. Majority Element II

```java
public List<Integer> majorityElement(int[] nums) {
        List<Integer> res = new ArrayList<>();int majority = nums.length/3;
        if(nums.length == 0){
            return res;
        }
        int maj1 = 0;int maj2 = 1;int ct1 = 0;int ct2 = 0;
        for(int i = 0; i < nums.length; i++){
            if(nums[i] == maj1){
                ct1++;
            }else if(nums[i] == maj2){
                ct2++;
            }else if(ct1 == 0){
                maj1 = nums[i];
                ct1 = 1;
            }else if(ct2 == 0){
                maj2 = nums[i];
                ct2 = 1;
            }else{
                ct1--;
                ct2--;
            }
        }
        ct1 = 0;ct2 = 0;
        for(int num: nums){
            if(num == maj1){
                ct1++;
            }else if(num == maj2){
                ct2++;
            }
        }
        if(ct1 > majority){
            res.add(maj1);
        }if(ct2 > majority){
            res.add(maj2);
        }
        return res;
    }
```

10. Missing ranges(LintCode)

```
public class Solution {
    public List<String> findMissingRanges(int[] nums, int lower, int upper) {
      List<String> res = new ArrayList<>();
      if(nums.length == 0){
        res.add(formattedString(lower, upper));
        return res;
      }


      long prev = (long)lower - 1;

      for(int i = 0; i <= nums.length; i++){
        long curr = (i < nums.length) ? nums[i] : (long) (upper + 1);

        if(curr - prev > 1){
          res.add(formattedString(prev+1,curr-1));
        }

        prev = curr;
      }

      return res;
    }

    public String formattedString(long lower, long upper){
      if(lower == upper){
        return String.valueOf(lower);
      }else{
        return lower + "->" + upper;
      }
    }
}
```

11. 3Sum

BruteForce: O(n^3)

```java
import java.util.*;

public class Rough{
    public static void main(String[] args){
        int[] nums = {-1,0,1,2,-1,-4};
        System.out.println(threeSum(nums));
    }

    public static List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Set<List<Integer>> set = new HashSet<>();
        for(int i = 0 ; i < nums.length; i++){
            for(int j = i + 1; j < nums.length; j++){
                for(int k = j + 1; k < nums.length; k++){
                    if((nums[i] + nums[j] + nums[k]) == 0){
                        List<Integer> temp = new ArrayList<>(List.of(nums[i], nums[j], nums[k]));
                        Collections.sort(temp);
                        if(set.add(temp)){
                            res.add(temp);
                        }
                    }
                }
            }
        }
        return res;
    }
}
```

O(n^2)

```java
public static List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    Set<List<Integer>> set = new HashSet<>();
    for(int i = 0 ; i < nums.length; i++){
        HashSet<Integer> hs = new HashSet<>();
        for(int j = i + 1; j < nums.length; j++){
            int k = -(nums[i] + nums[j]);
            if(hs.contains(k)){
                List<Integer> temp = new ArrayList<>(List.of(nums[i], nums[j], k));
                Collections.sort(temp);
                if(set.add(temp)){
                    res.add(temp);
                }
            }
            hs.add(nums[j]);//adding nums[j] because this number is present in the nums array; not 'k' because 'k' may not be present in the
            //nums array
        }
    }
    return res;
}
```

O(nlogn) + O(n^2)

```java
public static List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> res = new ArrayList<>();
    Set<List<Integer>> set = new HashSet<>();
    Arrays.sort(nums);
    int n = nums.length;
    for(int i = 0; i < n; i++){
        if(i > 0 && nums[i] == nums[i-1])  continue;
        int j = i + 1;
        int k = n - 1;
        while(j < k){
            int sum = nums[i] + nums[j] + nums[k];
            if(sum < 0){
                j++;
            }
            else if(sum > 0){
                k--;
            }
            else{
                List<Integer> temp = new ArrayList<>(List.of(nums[i], nums[j], nums[k]));
                Collections.sort(temp);
                if(set.add(temp)){
                    res.add(temp);
                }
                j++;
                k--;
                if(nums[j] == nums[j-1]) j++;
                if(nums[k] == nums[k+1]) k--;
            }
        }
    }
    return res;
}
```

12. 3 Sum Smaller

O(n^2)

```java
public static int threeSumSmaller(int[] nums, int target){
    int count = 0;
    Arrays.sort(nums);
    int n = nums.length;

    //Edge Cases: if length of nums < 3
    if(n < 3){
        return 0;
    }

    for(int i = 0; i < n; i++){
        if(i > 0 && nums[i] == nums[i-1]) continue;
        int left = i + 1;
        int right = n - 1;
        while(left < right){
            int sum = nums[i] + nums[left] + nums[right];
            if(sum ≥ target){
                right--;
            }
            else{
                count += right - left;
                left++;
            }
        }
    }
    return count;
```

13. 3 Sum Closest

O(N^2)

```java
public static int threeSumClosest(int[] nums, int target) {
    int n = nums.length;
    // Edge Case:
    if (n < 3) {
        return 0;
    }
    int closestSum = nums[0] + nums[1] + nums[2];
    Arrays.sort(nums);
    for (int i = 0; i < n; i++) {
        int left = i + 1;
        int right = n - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            // System.out.println("Sum" +sum);
            if(Math.abs(sum - target) < Math.abs(closestSum - target)){
                closestSum = sum;
            }
            // System.out.println("Closest Sum" +closestSum);
            if (sum < target) {
                left++;
            } else if (sum > target) {
                right--;
            } else {
                break;
            }
        }
    }
    return closestSum;
}
```

14. 4Sum

O(N^3)

```java
public static List<List<Integer>> fourSum(int[] nums, int target) {
    int n = nums.length;
    List<List<Integer>> result = new ArrayList<>();
    if (n < 4) {
        return result;
    }
    Set<List<Integer>> set = new HashSet<>();
    Arrays.sort(nums);
    for (int i = 0; i < n; i++) {
        if (i > 0 && nums[i] == nums[i - 1])
            continue;
        for (int j = i + 1; j < n; j++) {
            if (j > i + 1 && nums[j] == nums[j - 1])
                continue;
            int k = j + 1;// after j
            int l = n - 1;// last element of the nums array
            while (k < l) {
                long sum = (long) nums[i] + nums[j] + nums[k] + nums[l];
                if (sum < target) {
                    k++;
                } else if (sum > target) {
                    l--;
                } else {
                    List<Integer> temp = new ArrayList<>(List.of(nums[i], nums[j], nums[k], nums[l]));
                    if (set.add(temp)) {
                        result.add(temp);
                    }
                    k++;
                    l--;
                }
            }
        }
    }
    return result;
}
```

15 Rotate Image

Brute Force

```java
public static void rotate(int[][] matrix) {
    int n = matrix.length;
    int[][] resultMatrix = new int[matrix.length][matrix[0].length];
    for(int i = 0; i < matrix.length; i++){
        for(int j = 0; j < matrix[0].length; j++){
            int k = (n-1)-i;
            resultMatrix[j][k] = matrix[i][j];
        }
    }
    // System.out.println(Arrays.deepToString(resultMatrix));
}
```

Optimal

```java
public static void rotate(int[][] matrix) {
    int n = matrix.length;
    // Transpose
    for (int i = 0; i <= n - 2; i++) {
        for (int j = i + 1; j <= n - 1; j++) {
            int temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
        }
    }

    // System.out.println(Arrays.deepToString(matrix));

    // Reverse
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < matrix[i].length/2; j++) {
            int temp = matrix[i][j];
            matrix[i][j] = matrix[i][(matrix[i].length - 1)-j];
            matrix[i][(matrix[i].length - 1)-j] = temp;

        }
    }
    System.out.println(Arrays.deepToString(matrix));
}
```