Arrays and Lists with Numbers

Sorting

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 6};

        Arrays.sort(arr);

        System.out.println(Arrays.toString(arr));
        // Output: [1, 2, 5, 6, 9]
    }
}
```

Bubble Sort

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 6};

        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
```

```
            }
        }
    }
}
```

Selection Sort (O(n²))

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 6};

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

            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) minIndex = j;
            }

            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
}
```

Insertion Sort (O(n²)

```
public class Main {
    public static void main(String[] args) {
        int[] arr = {5, 2, 9, 1, 6};

        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
```

```
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }
}
```

MergeSort O(Nlogn)

```
public class MergeSort {

    public static void mergeSort(int[] arr) {
        if (arr.length < 2) return;

        int mid = arr.length / 2;
        int[] left = new int[mid];
        int[] right = new int[arr.length - mid];

        System.arraycopy(arr, 0, left, 0, mid);
        System.arraycopy(arr, mid, right, 0, arr.length - mid);

        mergeSort(left);
        mergeSort(right);

        merge(arr, left, right);
    }

    private static void merge(int[] arr, int[] left, int[] right) {
        int i = 0, j = 0, k = 0;
```

```
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j]) arr[k++] = left[i++];
            else arr[k++] = right[j++];
        }

        while (i < left.length) arr[k++] = left[i++];
        while (j < right.length) arr[k++] = right[j++];
    }
}
```

Implement binary search in a sorted array

```
class Solution {
    public int binarySearch(int[] arr, int target) {

        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;  // avoids overflow

            if (arr[mid] == target) {
                return mid;   // target found
            }
            else if (arr[mid] < target) {
                left = mid + 1;   // search right half
            }
            else {
                right = mid - 1;  // search left half
            }
        }

        return -1;  // target not found
    }
}
```

Find an element in an unsorted array.

```
class Solution {
    public int linearSearch(int[] arr, int target) {

        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;   // return index where target found
            }
        }

        return -1;  // not found
    }
}
```

Array Manipulation:

Find the maximum/minimum element in an

```
class Solution {
    public int[] findMinMax(int[] arr) {

        if (arr == null || arr.length == 0) {
            return new int[]{-1, -1};  // or throw exception
        }

        int min = arr[0];
        int max = arr[0];

        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
            if (arr[i] > max) {
                max = arr[i];
            }
        }

        return new int[]{min, max};
```

```
    }
}
```

Only max

```
public int findMax(int[] arr) {
    if (arr == null || arr.length == 0) return -1;

    int max = arr[0];

    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    return max;
}
```

Only min

```
public int findMin(int[] arr) {
    if (arr == null || arr.length == 0) return -1;

    int min = arr[0];

    for (int i = 1; i < arr.length; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
    }

    return min;
}
```

Find Sum of All Elements (O(n))

```
class Solution {
    public int findSum(int[] arr) {
```

```
        if (arr == null || arr.length == 0) return 0;

        int sum = 0;

        for (int num : arr) {
            sum += num;
        }

        return sum;
    }
}
```

AVG

```
class Solution {
    public double findAverage(int[] arr) {

        if (arr == null || arr.length == 0) return 0;

        int sum = 0;

        for (int num : arr) {
            sum += num;
        }

        return (double) sum / arr.length;
    }
}
```

Together

```
class Solution {
    public double[] sumAndAverage(int[] arr) {

        if (arr == null || arr.length == 0) return new double[]{0, 0};

        int sum = 0;
        for (int num : arr) {
            sum += num;
        }
```

```
        double avg = (double) sum / arr.length;

        return new double[]{sum, avg};
    }
}
```

```
Find pairs with a given sum
Pairs with sum 6:
(2, 4)
(5, 1)
(3, 3)
```

```
class Solution {

    public void findPairs(int[] arr, int target) {

        for (int i = 0; i < arr.length; i++) {
            for (int j = i + 1; j < arr.length; j++) {

                if (arr[i] + arr[j] == target) {
                    System.out.println("(" + arr[i] + ", " + arr[j] + ")");
                }
            }
        }
    }

    public static void main(String[] args) {

        Solution sol = new Solution();

        int[] arr = {2, 7, 4, 5, 1, 3};
        int target = 6;

        sol.findPairs(arr, target);
    }
}
```

Rotation of array

```
public void rotateBrute(int[] arr, int k) {

    int n = arr.length;
    k = k % n;

    while (k-- > 0) {
        int last = arr[n - 1];

        for (int i = n - 1; i > 0; i--) {
            arr[i] = arr[i - 1];
        }

        arr[0] = last;
    }
}
```

AP AND GP

```
class Solution {

    // AP nth term
    public int apNthTerm(int a, int d, int n) {
        return a + (n - 1) * d;
    }

    // AP sum
    public int apSum(int a, int d, int n) {
        return n * (2 * a + (n - 1) * d) / 2;
    }

    // GP nth term
    public double gpNthTerm(int a, int r, int n) {
        return a * Math.pow(r, n - 1);
    }

    // GP sum
    public double gpSum(int a, int r, int n) {
        if (r == 1) return a * n;
```

```
        return a * (1 - Math.pow(r, n)) / (1 - r);
    }

    public static void main(String[] args) {

        Solution sol = new Solution();

        System.out.println(sol.apNthTerm(2, 3, 5)); // 14
        System.out.println(sol.apSum(2, 3, 5));     // 40

        System.out.println(sol.gpNthTerm(3, 2, 4)); // 24
        System.out.println(sol.gpSum(3, 2, 4));     // 45
    }
}
```

- Find the missing number in [redacted]

```
class Solution {

    public int findMissing(int[] arr) {

        int n = arr.length + 1; // because one number missing
        int expectedSum = n * (n + 1) / 2;

        int actualSum = 0;
        for (int num : arr) {
            actualSum += num;
        }

        return expectedSum - actualSum;
    }

    public static void main(String[] args) {

        Solution sol = new Solution();

        int[] arr = {1, 2, 4, 5};

        int missing = sol.findMissing(arr);
```

```
        System.out.println("Missing number: " + missing);
    }
}
```

Check if a number is a power of 2

n & (n - 1) == 0 → n is power of 2

Because subtracting 1 flips all bits after the rightmost set bit.

```
class Solution {

    public boolean isPowerOfTwo(int n) {
        if (n <= 0) return false;

        return (n & (n - 1)) == 0;
    }

    public static void main(String[] args) {
        Solution sol = new Solution();

        int n = 16;
        System.out.println(sol.isPowerOfTwo(n));  // true
    }
}

//second way

public boolean isPowerOfTwo(int n) {
    if (n <= 0) return false;

    while (n % 2 == 0) {
        n /= 2;
    }

    return n == 1;
}

12%2==0                16%2==0
```

6%2==0 8%2==0
3%2==0 ❌ 4%2==0
2%2==0
1

- Count set bits in a number

```
public int countSetBits(int n) {
    int count = 0;

    while (n > 0) {
        count += (n & 1);
        n >>= 1;
    }

    return count;
}

public int countSetBits(int n) {
    int count = 0;

    while (n > 0) {
        n = n & (n - 1);
        count++;
    }

    return count;
}
```

Swapping of two number

```
public class SwapNumbers {
    public static void main(String[] args) {
        int a = 5;
        int b = 7;

        System.out.println("Before swap: a = " + a + ", b = " + b);
```

```
        a = a + b;
        b = a - b;
        a = a - b;

        System.out.println("After swap: a = " + a + ", b = " + b);
    }
}

public class SwapNumbers {
    public static void main(String[] args) {
        int a = 5;
        int b = 7;

        System.out.println("Before swap: a = " + a + ", b = " + b);

        a = a ^ b;
        b = a ^ b;
        a = a ^ b;

        System.out.println("After swap: a = " + a + ", b = " + b);
    }
}
```