Experiment-2

Student Name : Kumar Devashish		**UID**	**:**22BCS10248
Branch	**:** CSE	**Section/Group**	**:**FL_IOT-602-A
Semester	**:** 6th	**Date of Performance:**	04/02/2025
Subject Name	**:** AP Lab-II	**Subject Code**	**:**22CSP-351

Longest Nice Substring

```
class Solution {
    public String longestNiceSubstring(String s) {
        if (s.length() < 2) return "";

        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (s.indexOf(Character.toUpperCase(ch)) == -1 ||
s.indexOf(Character.toLowerCase(ch)) == -1) {
                String left = longestNiceSubstring(s.substring(0, i));
                String right = longestNiceSubstring(s.substring(i + 1));
                return left.length() >= right.length() ? left : right;
            }
        }
        return s;
    }
}
```

Reverse Bits

```
public class Solution {
    public int reverseBits(int n) {
        int result = 0;
        for (int i = 0; i < 32; i++) {
```

```
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
}
```

Hamming Weight

```
class Solution {
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += (n & 1);
            n >>>= 1;
        }
        return count;
    }
}
```

Maximum Subarray

```
class Solution {
    public int maxSubArray(int[] nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.length; i++) {
            currentSum = Math.max(nums[i], currentSum + nums[i]);
            maxSum = Math.max(maxSum, currentSum);
        }
        return maxSum;
    }
}
```

Super Pow

```
class Solution {
    private static final int MOD = 1337;

    public int superPow(int a, int[] b) {
        a %= MOD;
        int result = 1;
        for (int digit : b) {
            result = power(result, 10) * power(a, digit) % MOD;
        }
        return result;
    }

    private int power(int base, int exp) {
        int res = 1;
        while (exp > 0) {
            if ((exp & 1) == 1) res = res * base % MOD;
            base = base * base % MOD;
            exp >>= 1;
        }
        return res;
    }
}
```

Beautiful Array

```
import java.util.*;

class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = new ArrayList<>();
        result.add(1);
```

```
        while (result.size() < n) {
            List<Integer> temp = new ArrayList<>();
            for (int num : result) {
                if (num * 2 - 1 <= n) temp.add(num * 2 - 1);
            }
            for (int num : result) {
                if (num * 2 <= n) temp.add(num * 2);
            }
            result = temp;
        }

        return result.stream().mapToInt(i -> i).toArray();
    }
}
```

The Skyline Problem

```
import java.util.*;

class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> events = new ArrayList<>();
        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]});
            events.add(new int[]{b[1], b[2]});
        }

        events.sort((a, b) -> a[0] == b[0] ? Integer.compare(a[1], b[1]) :
Integer.compare(a[0], b[0]));

        List<List<Integer>> result = new ArrayList<>();
        PriorityQueue<Integer> pq = new
PriorityQueue<>(Collections.reverseOrder());
```

```
        pq.add(0);
        int prevMax = 0;

        for (int[] e : events) {
            if (e[1] < 0) pq.add(-e[1]);
            else pq.remove(e[1]);

            int currMax = pq.peek();
            if (currMax != prevMax) {
                result.add(Arrays.asList(e[0], currMax));
                prevMax = currMax;
            }
        }

        return result;
    }
}
```

Reverse Pairs

```
class Solution {
    public int reversePairs(int[] nums) {
        if (nums == null || nums.length == 0) return 0;
        return mergeSort(nums, 0, nums.length - 1);
    }

    private int mergeSort(int[] nums, int left, int right) {
        if (left >= right) return 0;

        int mid = left + (right - left) / 2;
        int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);
```

```
        int j = mid + 1;
        for (int i = left; i <= mid; i++) {
            while (j <= right && (long) nums[i] > 2L * nums[j]) j++;
            count += (j - (mid + 1));
        }

        merge(nums, left, mid, right);
        return count;
    }

    private void merge(int[] nums, int left, int mid, int right) {
        int[] temp = new int[right - left + 1];
        int i = left, j = mid + 1, k = 0;

        while (i <= mid && j <= right) {
            if (nums[i] <= nums[j]) temp[k++] = nums[i++];
            else temp[k++] = nums[j++];
        }

        while (i <= mid) temp[k++] = nums[i++];
        while (j <= right) temp[k++] = nums[j++];

        System.arraycopy(temp, 0, nums, left, temp.length);
    }
}
```

Merge Sorted Array

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1, j = n - 1, k = m + n - 1;
        while (i >= 0 && j >= 0) {
```

```
            if (nums1[i] > nums2[j]) nums1[k--] = nums1[i--];
            else nums1[k--] = nums2[j--];
        }
        while (j >= 0) nums1[k--] = nums2[j--];
    }
}
```

First Bad Version

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int left = 1, right = n;
        while (left < right) {
            int mid = left + (right - left) / 2;
            if (isBadVersion(mid)) right = mid;
            else left = mid + 1;
        }
        return left;
    }
}
```

Sort Colors

```
class Solution {
    public void sortColors(int[] nums) {
        int low = 0, mid = 0, high = nums.length - 1;

        while (mid <= high) {
            if (nums[mid] == 0) swap(nums, low++, mid++);
            else if (nums[mid] == 1) mid++;
            else swap(nums, mid, high--);
        }
    }
```

```
    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```