Infosys SP/DSE Complete Problem Repository

Table of Contents

Volume A: All 13 Source Problems - Full Solutions & Variations

Target: Infosys Specialist Programmer (L3/SP/DSE)
Edition: 2025
Content: Complete analysis of all 13 problems from PrepInsta + variations

Table of Contents

PART I: EASY LEVEL PROBLEMS

PROBLEM 1: RPG Monster Defeat

Problem Statement

While playing an RPG game, you must defeat n monsters in a quest. Each monster i has:

- **power[i]**: Minimum experience points needed to defeat it
- **bonus[i]**: Experience points gained after defeating it

You start with **e** experience points. To defeat monster i:

- You need `experience >= power[i]`
- After defeating: `experience += bonus[i]`
- Can defeat monsters in **any order**

Task: Find maximum number of monsters you can defeat.

Input Format

```
Line 1: n (number of monsters)
Line 2: e (initial experience)
Lines 3 to n+2: power[i] for each monster
Lines n+3 to 2n+2: bonus[i] for each monster
```

Constraints:

- 1 ≤ n ≤ 10^5
- 1 ≤ e ≤ 10^9
- 1 ≤ power[i], bonus[i] ≤ 10^9

Theory & Approach

Greedy Observation

Key Insight: Always defeat monsters in **increasing order of power**.

Proof by Exchange Argument:

Suppose optimal solution defeats monster A before B where `power[A] > power[B]`.

Case 1: We can defeat both

- Current order: defeat A (need `exp >= power[A]`), then B
- Swapped order: defeat B (need `exp >= power[B]`), then A
- Since `power[B] < power[A]`, swapped order is always feasible
- Both orders defeat same monsters → Contradiction to optimality

Case 2: We can defeat only one

- If we choose A (harder), we miss easier B
- Choosing B first gives chance to then defeat A
- Greedy (easier first) is optimal

Conclusion: Sorting by power is optimal.

C++ Solution

```
#include &lt;bits/stdc++.h&gt;
using namespace std;

int maxMonsters(int n, long long exp, vector&lt;int&gt;&amp; power, vector&lt;int&gt;&amp; bonus) {
    // Create vector of pairs (power, bonus)
    vector&lt;pair&lt;int, int&gt;&gt; monsters;
    for (int i = 0; i &lt; n; i++) {
        monsters.push_back({power[i], bonus[i]});
    }

    // Sort by power (ascending)
    sort(monsters.begin(), monsters.end());

    int count = 0;
    for (auto&amp; [pow, bon] : monsters) {
        if (exp &gt;= pow) {
            exp += bon;
            count++;
        } else {
            break; // Cannot defeat this or any harder monster
        }
    }

    return count;
}

int main() {
    int n;
    long long e;
    cin &gt;&gt; n &gt;&gt; e;

    vector&lt;int&gt; power(n), bonus(n);
    for (int i = 0; i &lt; n; i++) cin &gt;&gt; power[i];
    for (int i = 0; i &lt; n; i++) cin &gt;&gt; bonus[i];

    cout &lt;&lt; maxMonsters(n, e, power, bonus) &lt;&lt; endl;
    return 0;
}
```

Time Complexity: O(n log n) — sorting dominates
Space Complexity: O(n) — storing monster pairs

Python Solution

```
def max_monsters(n, exp, power, bonus):
    # Create list of tuples (power, bonus) and sort
    monsters = sorted(zip(power, bonus))

    count = 0
    for pow, bon in monsters:
        if exp &gt;= pow:
            exp += bon
            count += 1
        else:
            break  # Cannot defeat any harder monster

    return count

# Input<a></a>
n = int(input())
e = int(input())
power = [int(input()) for _ in range(n)]
bonus = [int(input()) for _ in range(n)]
```

```
# Output<a></a>
print(max_monsters(n, e, power, bonus))
```

Variations & Related Problems

Variation 1: Maximize Experience (Not Count)

Problem: Instead of maximizing count, maximize final experience.

Approach: Same greedy (sort by power), but return final experience.

```
long long maxExperience(int n, long long exp, vector&lt;int&gt;&amp; power, vector&lt;int&gt;&amp; bonus) {
    vector&lt;pair&lt;int, int&gt;&gt; monsters;
    for (int i = 0; i &lt; n; i++) {
        monsters.push_back({power[i], bonus[i]});
    }

    sort(monsters.begin(), monsters.end());

    for (auto&amp; [pow, bon] : monsters) {
        if (exp &gt;= pow) {
            exp += bon;
        }
    }

    return exp;
}
```

Variation 2: With Negative Bonuses

Problem: Bonus can be negative (monster drains experience).

Approach:

- Separate positive and negative bonus monsters
- Sort positive monsters by power (ascending)
- Sort negative monsters by (power - bonus) to minimize loss
- Defeat positive first, then negative if needed

```
int maxMonstersNegative(int n, long long exp, vector&lt;int&gt;&amp; power, vector&lt;int&gt;&amp; bonus) {
    vector&lt;pair&lt;int, int&gt;&gt; positive, negative;

    for (int i = 0; i &lt; n; i++) {
        if (bonus[i] &gt;= 0) {
            positive.push_back({power[i], bonus[i]});
        } else {
            negative.push_back({power[i], bonus[i]});
        }
    }

    // Sort positive by power
    sort(positive.begin(), positive.end());

    // Sort negative by (power - bonus) to minimize effective cost
    sort(negative.begin(), negative.end(), [](auto&amp; a, auto&amp; b) {
        return a.first - a.second &lt; b.first - b.second;
    });

    int count = 0;

    // Defeat positive monsters
    for (auto&amp; [pow, bon] : positive) {
        if (exp &gt;= pow) {
            exp += bon;
```

```
            count++;
        }
    }

    // Defeat negative monsters
    for (auto&amp; [pow, bon] : negative) {
        if (exp &gt;= pow) {
            exp += bon;
            count++;
        }
    }

    return count;
}
```

Variation 3: With Limited Monster Types

Problem: Each monster can only be defeated once, but multiple monsters of same type exist.

Approach: Group by type, sort groups, defeat greedily.

Edge Cases & Testing

```
// Test Case 1: All monsters too powerful
Input:
3
10
20 30 40
5 5 5
Expected Output: 0

// Test Case 2: Negative bonus causes failure
Input:
3
100
50 60 70
-10 -20 -80
Expected Output: 2 (defeat first two, avoid third)

// Test Case 3: All same power
Input:
3
100
50 50 50
10 20 30
Expected Output: 3 (order doesn't matter within same power)

// Test Case 4: Single monster
Input:
1
100
50
10
Expected Output: 1

// Test Case 5: Maximum constraints
Input:
100000
1000000000
// All power[i] = 1000000000, bonus[i] = 1
// Defeat first, then fail
Expected Output: 1
```

PROBLEM 2: Base Conversion (Minimum Base)

Problem Statement

Given a natural number M in decimal (base 10), find the **minimum base B** such that when M is represented in base B, **all digits are the same**.

Example:

- M = 63 (base 10) = 333 (base 4)
  - Check: $3 \times 4^2 + 3 \times 4 + 3 = 48 + 12 + 3 = 63$ ✓
- M = 41 (base 10) = 11 (base 40)
  - Check: $1 \times 40 + 1 = 41$ ✓

Input: M (1 ≤ M ≤ 10^12)
Output: Minimum base B

Theory & Approach

Mathematical Foundation

If M in base B is represented as d d d ... d (k times), then:

$$M = d \times (B^{k-1} + B^{k-2} + \ldots + B + 1) \tag{1}$$

Using geometric series:

$$M = d \times \frac{B^k - 1}{B - 1} \tag{2}$$

Rearranging:

$$M \times (B - 1) = d \times (B^k - 1) \tag{3}$$

Brute Force Approach

Algorithm:

1. Try each base B starting from 2
2. For each B, convert M to base B
3. Check if all digits are same
4. Return first valid B

Time Complexity: O(B × log_B(M)) where B can be up to M-1

C++ Solution (Optimized)

```
#include &lt;bits/stdc++.h&gt;
using namespace std;

// Check if M in base B has all same digits
bool allSameDigits(long long M, long long base) {
    long long firstDigit = M % base;
    M /= base;

    while (M &gt; 0) {
        if (M % base != firstDigit) {
            return false;
        }
        M /= base;
    }
```

```
    return true;
}

long long minBase(long long M) {
    // Special case: M = 0 or 1
    if (M &lt;= 1) return M + 1;

    // Try each base starting from 2
    for (long long base = 2; base &lt;= M; base++) {
        if (allSameDigits(M, base)) {
            return base;
        }

        // Optimization: If base &gt; sqrt(M), only k=2 possible
        if (base * base &gt; M) {
            // For k=2: M = d×base + d = d×(base+1)
            // So d = M/(base+1), and M % (base+1) must be 0
            // Try all divisors of M
            break;
        }
    }

    // For large M, answer is M-1 (M = 11 in base M-1)
    return M - 1;
}

// Optimized version for large M
long long minBaseOptimized(long long M) {
    if (M &lt;= 1) return M + 1;

    // Check bases up to sqrt(M)
    long long sqrtM = sqrt(M);
    for (long long base = 2; base &lt;= sqrtM + 1; base++) {
        if (allSameDigits(M, base)) {
            return base;
        }
    }

    // Check if M = d×(base+1) for some base
    // This means M has exactly 2 digits in base
    for (long long d = 2; d * d &lt;= M; d++) {
        if (M % d == 0) {
            long long base = M / d - 1;
            if (base &gt; 1 &amp;&amp; allSameDigits(M, base)) {
                return base;
            }
        }
    }

    // Default: M = 11 in base M-1
    return M - 1;
}

int main() {
    long long M;
    cin &gt;&gt; M;
    cout &lt;&lt; minBaseOptimized(M) &lt;&lt; endl;
    return 0;
}
```

Time Complexity: O(√M) — optimized divisor checking
Space Complexity: O(1)

Python Solution

```python
def all_same_digits(M, base):
    """Check if M in given base has all same digits"""
    first_digit = M % base
    M //= base

    while M &gt; 0:
        if M % base != first_digit:
            return False
        M //= base

    return True

def min_base(M):
    """Find minimum base where M has all same digits"""
    if M &lt;= 1:
        return M + 1

    # Check bases from 2 to sqrt(M)
    sqrt_M = int(M ** 0.5) + 1
    for base in range(2, sqrt_M + 2):
        if all_same_digits(M, base):
            return base

    # Check divisors for k=2 case (M = d*base + d)
    for d in range(2, sqrt_M + 1):
        if M % d == 0:
            base = M // d - 1
            if base &gt; 1 and all_same_digits(M, base):
                return base

    # Default: M = 11 in base M-1
    return M - 1

# Input<a></a>
M = int(input())
print(min_base(M))
```

Variations

Variation 1: Find All Valid Bases

Problem: Find all bases B where M has all same digits.

```cpp
vector&lt;long long&gt; allValidBases(long long M) {
    vector&lt;long long&gt; result;

    for (long long base = 2; base &lt; M; base++) {
        if (allSameDigits(M, base)) {
            result.push_back(base);
        }
    }

    return result;
}
```

Variation 2: Maximum Base with Property

Problem: Find maximum base (instead of minimum).

Answer: Always M-1 (since M = 11 in base M-1).

Edge Cases

```
// Test Case 1: M = 1
Input: 1
Output: 2 (1 = 1 in any base &gt;= 2)

// Test Case 2: M = 2
Input: 2
Output: 3 (2 = 2 in base 3, but not in base 2)

// Test Case 3: Perfect power
Input: 64
Output: 2 (64 = 1000000 in base 2, all 1's? No)
// Actually: 64 = 100 in base 8 (not all same)
// 64 = 44 in base... no
// Answer: 63 (64 = 11 in base 63)

// Test Case 4: Prime number
Input: 41
Output: 40 (41 = 11 in base 40)

// Test Case 5: Large M
Input: 1000000000000
Output: ? (need to check divisors efficiently)
```

PROBLEM 3: Mountain Array Transformation

Problem Statement

Given array of size N, transform it into a **mountain array** with minimum changes.

Mountain Array Definition:

- Either ends have equal elements
- Moving towards middle from both ends, next element is exactly 1 more than previous
- Peak in middle

Examples:

- `[1, 2, 3, 2, 1]` ✓ Mountain
- `[6, 7, 8, 8, 7, 6]` ✓ Mountain
- `[1, 2, 4, 2, 1]` ✗ Not mountain (2 to 4 is +2, not +1)

Task: Find minimum changes needed.

Theory & Approach

Observation

Mountain Structure:

- For even N: Two middle elements must be equal
- For odd N: Single middle element is peak
- From each end, elements increase by 1 towards middle

Greedy Strategy

1. **Fix the middle** (or two middle elements if even N)
2. **Work outward** from middle
3. **Force arithmetic sequence** with difference 1

Why Greedy Works: Fixing middle and working outward ensures minimal changes (each element depends on its neighbor).

C++ Solution

```
#include &lt;bits/stdc++.h&gt;
using namespace std;

int minChanges(vector&lt;int&gt;&amp; arr) {
    int n = arr.size();
    int changes = 0;

    if (n % 2 == 0) {
        // Even length
        int mid1 = n / 2 - 1;
        int mid2 = n / 2;

        // Make middle two elements equal
        if (arr[mid1] != arr[mid2]) {
            arr[mid2] = arr[mid1];
            changes++;
        }

        // Build left side (decreasing by 1)
        for (int i = mid1 - 1; i &gt;= 0; i--) {
            if (arr[i] != arr[i + 1] - 1) {
                arr[i] = arr[i + 1] - 1;
                changes++;
            }
        }

        // Build right side (mirror of left)
        for (int i = mid2 + 1; i &lt; n; i++) {
            int mirrorIdx = n - 1 - i;
            if (arr[i] != arr[mirrorIdx]) {
                arr[i] = arr[mirrorIdx];
                changes++;
            }
        }

    } else {
        // Odd length
        int mid = n / 2;

        // Build left side (decreasing by 1)
        for (int i = mid - 1; i &gt;= 0; i--) {
            if (arr[i] != arr[i + 1] - 1) {
                arr[i] = arr[i + 1] - 1;
                changes++;
            }
        }

        // Build right side (mirror of left)
        for (int i = mid + 1; i &lt; n; i++) {
            int mirrorIdx = n - 1 - i;
            if (arr[i] != arr[mirrorIdx]) {
                arr[i] = arr[mirrorIdx];
                changes++;
            }
        }
    }

    return changes;
}
```

```
int main() {
    int n;
    cin &gt;&gt; n;

    vector&lt;int&gt; arr(n);
    for (int i = 0; i &lt; n; i++) {
        cin &gt;&gt; arr[i];
    }

    cout &lt;&lt; minChanges(arr) &lt;&lt; endl;
    return 0;
}
```

Time Complexity: O(n)
Space Complexity: O(1) — in-place modification

Python Solution

```
def min_changes(arr):
    n = len(arr)
    changes = 0

    if n % 2 == 0:
        # Even length
        mid1 = n // 2 - 1
        mid2 = n // 2

        # Make middle two equal
        if arr[mid1] != arr[mid2]:
            arr[mid2] = arr[mid1]
            changes += 1

        # Build left side
        for i in range(mid1 - 1, -1, -1):
            if arr[i] != arr[i + 1] - 1:
                arr[i] = arr[i + 1] - 1
                changes += 1

        # Build right side (mirror)
        for i in range(mid2 + 1, n):
            mirror_idx = n - 1 - i
            if arr[i] != arr[mirror_idx]:
                arr[i] = arr[mirror_idx]
                changes += 1

    else:
        # Odd length
        mid = n // 2

        # Build left side
        for i in range(mid - 1, -1, -1):
            if arr[i] != arr[i + 1] - 1:
                arr[i] = arr[i + 1] - 1
                changes += 1

        # Build right side (mirror)
        for i in range(mid + 1, n):
            mirror_idx = n - 1 - i
            if arr[i] != arr[mirror_idx]:
                arr[i] = arr[mirror_idx]
                changes += 1

    return changes

# Input<a></a>
n = int(input())
arr = [int(input()) for _ in range(n)]
```

```
# Output<a></a>
print(min_changes(arr))
```

Variations

Variation 1: Return Transformed Array

```
vector&lt;int&gt; transformToMountain(vector&lt;int&gt; arr) {
    minChanges(arr); // Modifies arr in-place
    return arr;
}
```

Variation 2: Allow Decrements Only

Problem: Can only decrease elements (not increase).

Approach: Start from peak, decrease outward.

Edge Cases

```
// Test Case 1: Already mountain
Input: [1, 2, 3, 2, 1]
Output: 0

// Test Case 2: All same
Input: [5, 5, 5, 5, 5]
Output: 0 (already valid: peak = 5)

// Test Case 3: Single element
Input: [10]
Output: 0

// Test Case 4: Two elements
Input: [3, 5]
Output: 1 (change to [3, 3] or [5, 5])

// Test Case 5: Negative numbers allowed
Input: [1, 0, -1, 0, 1]
Output: 0 (already valid)
```

References

[1] PrepInsta. (2025). Infosys SP and DSE Coding Questions. https://prepinsta.com/infosys-sp-and-dse/

[2] Codeforces Educational Round 150 (Base Conversion Problem)

[3] LeetCode #941: Valid Mountain Array