1. Advanced String Permutation with Constraints

Problem: Write a program that generates all possible unique permutations of a given string, subject to the following constraints:

- No permutation should contain more than two consecutive identical characters.
- Certain substrings (provided as input) should not appear in any permutation.

Requirements:

- The function should accept a string `s` and a list of forbidden substrings.
- Generate only valid permutations, filtering out any that violate the constraints.
- Return all unique valid permutations.

Example Explanation:

- Consider the input string `"aabb"` and forbidden substring `["ab"]`.
- Permutations like `"aabb"` and `"abba"` should be excluded because they contain consecutive characters beyond the allowed limit or contain the substring `"ab"`.
- Valid permutations could include `["abab", "baba", “bbaa"]`.

Sample Input:

```
String s = “aabb”;
List<String> forbiddenSubstrings = Arrays.asList("ab");
```

Sample Output:

```
["abab", "baba", "bbaa"]
```

2. Array Partition with Minimum Difference

Problem: Given an array of integers, partition it into two subsets such that the absolute difference between the sums of the two subsets is minimized.

Requirements:

Implement a method that takes an array `arr` and returns the two subsets with the smallest difference in their sums.

Use dynamic programming to solve it efficiently, especially for larger arrays.

Example Explanation:

For the input `[1, 6, 11, 5]`, possible subset pairs could be:

- `[1, 11]` and `[6, 5]`, with sums `12` and `11`, yielding a difference of `1`.
- `[1, 6, 5]` and `[11]`, with sums `12` and `11`, also yielding a difference of `1`.

Either combination is a valid output, as the minimum difference achievable is `1`.

Sample Input:

```
int[] arr = {1, 6, 11, 5};
```

Sample Output:

```
Minimum Difference: 1

Subsets: [1, 11] and [6, 5]
```

3. Efficient Text Justification Algorithm

Problem: Write a function to format a list of words into a text paragraph with specified line width. Each line should have exactly `maxWidth` characters with evenly distributed spaces between words. Words should not be split across lines.

Requirements:

- ◦ Each line should have words separated by spaces such that the total line width is exactly `maxWidth`.
- ◦ Extra spaces should be added between words from left to right. The last line should be left-justified with no extra spaces between words.
- ◦ Ensure the algorithm can handle edge cases like single-word lines or lines with just enough words to reach the `maxWidth`.

Example Explanation:

- ◦ Given the words `["This", "is", "an", "example", "of", "text", "justification."]` with a `maxWidth` of `16`, the function should output:
  - ▪ `"This is an"`: This line contains exactly `16` characters, with spaces evenly distributed.
  - ▪ `"example of text"`: This line is also `16` characters, adjusted to fit the width.
  - ▪ `"justification.  "`: The last line is left-justified with remaining spaces at the end.

Sample Input:

```
String[] words = {"This", "is", "an", "example", "of",
"text", "justification."};

int maxWidth = 16;
```

Sample Output:

```
[  "This    is    an",  "example  of text",  "justification.
"]
```

4. High-Performance Log Analyzer with Custom Sorting

Problem: Design a log analyzer that processes a list of log entries, with each entry containing a timestamp, a log level (e.g., INFO, ERROR, WARN), and a message. Implement sorting and filtering functionalities.

Requirements:

- Sort logs based on different criteria (timestamp, log level, or message content).
- Filter logs by specific log levels or keywords.
- Group logs by a specified time interval (e.g., hourly or daily).
- Use efficient sorting algorithms and optimizations for large log files.

Example Explanation:

- For an input of logs like: `arduino`
- `"2024-11-08 10:00:00 INFO Starting system check",`
- `"2024-11-08 10:05:00 ERROR Disk failure detected",`
- `"2024-11-08 10:10:00 WARN Memory usage high",`
- `"2024-11-08 10:20:00 INFO System check complete"`
- Sorting by log level would result in ordering by `ERROR`, then `WARN`, then `INFO`.

Sample Input:

```
List<String> logs = Arrays.asList(

  "2024-11-08 10:00:00 INFO Starting system check",
  "2024-11-08 10:05:00 ERROR Disk failure detected",
  "2024-11-08 10:10:00 WARN Memory usage high",
  "2024-11-08 10:20:00 INFO System check complete"
);
String sortBy = "Log Level";
```

Sample Output:

```
[  "2024-11-08 10:05:00 ERROR Disk failure detected",
"2024-11-08 10:10:00 WARN Memory usage high",  "2024-11-08
10:00:00 INFO Starting system check",  "2024-11-08 10:20:00
INFO System check complete"]
```

5. Optimal Matrix Multiplication Order (Dynamic Programming)

Problem: Given a chain of matrices, find the optimal way to multiply them to minimize the number of scalar multiplications.

Requirements:

- You're given an array of dimensions representing matrices in a chain. Write a method to return the minimum cost and the order in which to perform matrix multiplications.

- Use dynamic programming to solve the problem efficiently.

Example Explanation:

- Given matrices with dimensions `[10, 30, 5, 60]`, where matrix dimensions are `10x30`, `30x5`, and `5x60`:
    - Possible orders are `((A1 x A2) x A3)` or `(A1 x (A2 x A3))`.
    - The minimum cost for `((A1 x A2) x A3)` is `4500` scalar multiplications.
    - The program should output both the minimum cost and the optimal multiplication order.

Sample Input:

```
int[] dimensions = {10, 30, 5, 60};
```

Sample Output:

```
Minimum Cost: 4500

Optimal Order: ((A1 x A2) x A3)
```