COP 3503 Homework #6: Dynamic Programming (2 Problems)

Problem A: Maximum k-Partition GCD Sum (50 points)

Filename: gcdsum.java, Time Limit: 2 seconds ***Standard Input, Standard Output***

A k-partition of a list is splitting the list into k separate consecutive contiguous segments, where each element in the list is included in exactly one segment and each segment contains at least one value. Below is a 4-partition of a list of size 10, where each segment is indicated by a different color:

3	9	6	8	4	12	13	11	22	33

For each segment of the partition, consider taking the greatest common divisor of all the numbers in the segment:

gcd(3, 9, 6) = 3
gcd(8, 4, 12) = 4
gcd(13) = 13
gcd(11, 22, 33) = 11

We define the gcd sum of a particular k-partition of a list to be the sum of the gcd's of each of the k partitions. For the 4-partition shown above, the corresponding gcd sum is 3 + 4 + 13 + 11 = 31.

The Problem

Given a list of ***n*** integers and an integer ***k***, determine the maximum k-partition GCD sum AND determine a partition of the input values that achieves this maximum sum.

The Input

The first line of input contains two integers, ***n*** ($1 \le n \le 5000$), representing the number of integers in the input list and ***k*** ($1 \le k \le \min(10, n)$), the number of segments to partition the input list into.

The next line will contain ***n*** space separated positive integers, indicating the input list, in order. Each of these integers will be in between 1 and 10^9, inclusive.

The Output

On a line by itself, output the maximum k-partition gcd sum. On the second line of output, indicate the 1-based starting index of each of the k segments, in increasing order. Output one space after each number and finish the output with a newline character.

Sample Input	**Sample Output**

`10 4`
`3 9 6 8 4 12 13 11 22 33`
	`67`
	`1 8 9 10`

`8 3`
`16 47 32 48 6 18 12 24`
	`65`
	`1 2 3`

`12 5`
`32 32 17 17 17 34 18 15 6 13 17 5`
	`116`
	`1 2 3 6 7`

Problem B: Maximum k-Smooth Subsequence Sum (50 points)

Filename: subseqsum.java, Time Limit: 2 seconds ***Standard Input, Standard Output***

A subsequence of a given input sequence is any sequence that can be obtained by deleting 0 or more elements from the input sequence. We define the sum of a subsequence to be the sum of the elements in a given subsequence. For this problem, not all subsequences are allowed. A subsequence is called “k-smooth” if for each pair of adjacent terms in the subsequence, the absolute value of their difference does not exceed k. For example, in the sequence given below, the terms highlighted in yellow for a valid 4-smooth subsequence:

3	9	6	8	4	12	13	11	22	33

The Problem

Given a list of ***n*** integers and an integer ***k***, determine the maximum k-smooth subsequence sum AND determine a subsequence of the input values that achieves that maximum sum.

The Input

The first line of input contains two integers, ***n*** ($1 \le n \le 5000$), representing the number of integers in the input list and ***k*** ($1 \le k \le 10^9$), the maximum difference allowed between consecutive terms in the chosen subsequence.

The next line will contain ***n*** space separated positive integers, indicating the input list, in order. Each of these integers will be in between 1 and 10^9, inclusive.

The Output

On a line by itself, output the maximum k-smooth subsequence sum. On the second line of output, indicate the 1-based index of each term in the subsequence.

Sample Input	**Sample Output**

`10 4`
`3 9 6 8 4 12 13 11 22 33`
	`59`
	`2 3 4 6 7 8`

`8 15`
`16 47 32 48 6 18 12 24`
	`133`
	`2 3 6 7 8`

`12 5`
`13 19 16 15 22 27 21 24 37 16 42 33`
	`92`
	`2 5 6 8`

Implementation Hints

Both problems require dynamic programming and have solutions that are somewhat similar in structure and somewhat short. It’s completely fine if your whole solution is in main (if written as an iterative DP), or if you just have one memorized function other than main (if using memorization). The process by which to generate the “build back” is fairly similar for both problems, so once you figure one of them out, the other one is likely to take much less time.

Rough Grading Breakdown

Each problem is out of 50 points and will have the points separated into three categories:

10 points – code/style points (for how the code looks and what is being attempted)
20 points – Execution Points based on the answer for the first line of output (max sum)
20 points – Execution Points based on the answer for the second line of output
(sequence/partition reconstruction)

What To Submit

For this assignment, please submit a two Java programs named **`gcdsum.java,`** for your solution to problem A, and **`subseqsum.java,`** for your solution to problem B. ***In your header comment, state whether you solved just task 1 or attempted to solve both tasks.***