Algorithms
Dynamic Programming Project (100 pts total, 40 pts for this part)
Timber Problem

You are not allowed to use the internet or consult any external references. You may use lecture slides.

1 Problem Description

1.1 Introduction

The timber problem discussed in class and on the supplemental handout is briefly defined as follows: Given an array of n positive integers representing the length of segments that a tree will be split into, what is the maximum length of wood that you can leave the sawmill with if you can only take one segment at a time from the end of the log, alternating picks with a neighbor who is your intellectual equal.

1.2 Recurrence Relation

Recall from the handout that the recurrence relation for the timber problem is:

$$T(i,j) = (\sum_{k=i}^{j} l_k) - \min[T(i+1,j), T(i,j-1)]$$

$$\text{Base Case: } T(i,i) = l_i$$

2 Deliverables - Part 3 - Dynamic Programming + Traceback

Please submit all deliverables as requested below for each question.

1. [40] In addition to computing the maximum sum of lengths that can be achieved, we now want to incorporate a traceback step that outputs the order in which the segments are taken by both you and your neighbor (your choices are the 1st, 3rd, 5th, etc). **See the output format section below for important tie-breaking considerations.** Implementing traceback will require the following three steps:

   (a) Augment each cell of your DP table from Part 2 with a suitable field similar to the "parent" field used in edit distance. This additional field in each cell should take $O(1)$ space (so no data structures such as lists and vectors).

(b) Modify your DP Algorithm from Part 2 so that the "parent" field is suitably filled.

(c) Develop a post-processing step that uses the "parent" field to perform traceback and outputs the order in which segments are taken.

Either a top-down or bottom-up approach will be accepted, but note that a bottom-up approach is typically slightly more efficient in practice. Time limits on Gradescope will be set up so that significant inefficiencies will likely exceed the time limit. For example, if your approach results in a factor of n being multiplied to the time complexity of your Part 2 submission, that would probably exceed the time limit.

Extra Credit: Submit your code to Gradescope for verification. Passing 100% of test cases on the first submission will be awarded with 5 extra credit points. Passing 100% of test cases on the second submission will be awarded with 2.5 extra credit points. To be clear, this extra credit is only awarded if all tests are passed; failing (or exceeding the time limit) on even only one test = no extra credit. The third submission does not have any associated extra credit. Subsequent submissions will incur a penalty of 10% per additional attempt.

Input Format
The input consists of two lines:

- The first line contains $1 \leq n \leq 2000$, the number of segments in the tree. ***n* can be odd or even.**
- The second line contains n space-separated integers giving the value of each tree segment from left to right. The value of each tree segment t will be in the range $1 \leq t \leq 1000$.

Output Format
The output consists of two lines:

- On the first line, print the maximum sum of lengths that you can take.
- On the second line, print the segment numbers in the order in which they are taken, separated by spaces.

  Tie Breaking: For consistency with our solutions, if presented with two choices that result in the same optimal outcome, choose the tree segment that is on the left/bottom side of the tree (the i segment, not the j segment). (Note that this does not necessarily mean that $l_i \geq l_j$.)

For example, an input of

```
4
5 6 9 7
```

would have the following output:

```
14
1 2 3 4
```