Algorithms
Dynamic Programming Project (100 pts)
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:

$$T(i,j) = \max\Big(l_i + \min\big[T(i+2,j), T(i+1,j-1)\big], l_j + \min\big[T(i+1,j-1), T(i,j-2)\big]\Big)$$

$$\text{Base Cases: } \begin{cases} T(i,j) = l_i & j = i \\ T(i,j) = \max(l_i, l_j) & j = i+1 \end{cases}$$

2 Deliverables

Please submit all of the items requested below in parts 1-5 to Gradescope in a report format. The verification in part 6 requires a code submission, which is on Gradescope as well.

Assume that n is even for parts 1-2, but n can be odd or even in parts 3-6.

1. [20] Write a recursive algorithm to solve the problem for the tree represented as [33, 28, 35, 25, 29, 34, 28, 32]. Implement the recurrence relation as-is, meaning your function should make <u>two</u> recursive calls to $T(i+1, j-1)$. *You may lose points if you do not make the redundant recursive call.* Including the initial call, how many calls are made to the function? Include your code in the report.

2. [10] Derive the asymptotic complexity and provide an exact bound (i.e., use Θ) for the recursive algorithm. Run the code from the previous algorithm on several different sizes of n to characterize the growth in runtime as n increases. Use a random number generator (RNG) to create the arrays. Provide a table or plot of your results, and compare/contrast the timing results with the derived asymptotic complexity.

   Assume n is even in your complexity analysis.

3. [20] Implement a dynamic programming algorithm (that uses a table to avoid recomputation) to compute the maximum sum of lengths that can be achieved. Include your code in the report.

   Assume n can be odd or even.

4. [10] Derive the asymptotic complexity and provide an exact bound (i.e., use Θ) for the dynamic programming algorithm. Run the code from the previous algorithm on several different sizes of n to characterize the growth in runtime as n increases. Use a random number generator (RNG) to create the arrays. Provide a table or plot of your results, and compare/contrast the timing results with the derived asymptotic complexity.

   Assume n can be odd or even.

5. [10] Implement a traceback step that identifies the order in which the segments are taken by both you and your neighbor (your choices are the 1st, 3rd, 5th, etc). Include your code in the report.

   Assume n can be odd or even.

   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 $l_i \geq l_j$.)

6. [30] Demonstrate that your code works correctly by submitting it to Gradescope.

   Input Format
   The input consists of two lines:

   - The first line contains $1 \leq n \leq 1000$, 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.

   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.

For example, an input of

```
4
5 6 9 7
```

would have the following output:

```
14
1 2 3 4
```