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 an even number, 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:

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

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

2 Deliverables

Please submit all of the items requested below in a single PDF file on Canvas.

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 two 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.

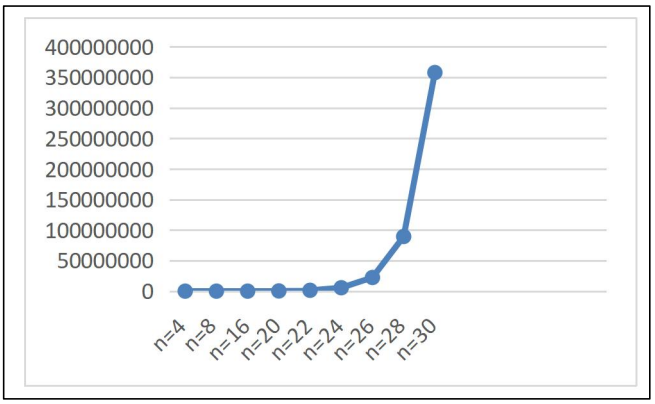
Timber: [33 28 35 25 29 34 28 32]
Maximum sum length is : 125
The number of Call the T(i,j) is : 85

Python Code:

```
import numpy as np

def timberRecursiveAlgo(i, j, timber_array):
    global callCount
    callCount= callCount+1
    if j == i:
        return timber_array[i-1]
    elif j == i+1:
        return max(timber_array[i-1], timber_array[j-1])
    else:
        return(max((timber_array[i-1]+min(timberRecursiveAlgo(i+2, j, timber_array), timberRecursiveAlgo(i+1, j-1,
timber_array))), (timber_array[j-1]+min(timberRecursiveAlgo(i+1, j-1, timber_array), timberRecursiveAlgo(i, j-2,
timber_array)))))

timber_array = [33, 28, 35, 25, 29, 34, 28, 32]
timber_array = np.asarray(timber_array)
callCount = 0
result = timberRecursiveAlgo(1, len(timber_array), timber_array)

print('Timber: ', timber_array)
print('Maximum sum length is :', result)
print('The number of Call the T(i,j) is : ', callCount)
```

2. [10] Run the code from the previous algorithm on a few different 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 discuss.

Number of Segments n	Number of Call the T(i,j) m	Running Time t（seconds）
n=4	5	0
n=8	85	0
n=16	21845	0.03
n=20	349525	0.29
n=22	1398101	1.15
n=24	5592405	6.56
n=26	22369621	19.57
n=28	89478485	78.28
n=30	357913941	329.96



Conclusion 1: The total complexity of the DP algorithm is : $O(2^n)$.

Because as the recursive Algorithm above:

The complexity of these two lines as below is O(n).

```
if j == i:
    return timber_array[i-1]
```

These lines are for initialize the base case:T(i, j) = l_i when j = i

The complexity of these two lines as below is O(n).

```
elif j == i+1:
    return max(timber_array[i-1], timber_array[j-1])
```

These lines are for initialize the base case:T(i, j) = max(l_i, l_j) when j = i + 1

The complexity of these two lines as below is O(2^n).

else:

return(max((timber_array[i-1]+min(timberRecursiveAlgo(i+2, j, timber_array), timberRecursiveAlgo(i+1, j-1, timber_array))), (timber_array[j-1]+min(timberRecursiveAlgo(i+1, j-1, timber_array), timberRecursiveAlgo(i, j-2, timber_array)))))

These lines are for computing the case:

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

Because The growth in runtime t is 4 times n(n is even) increases. So the complexity is **O($\frac{1}{4} * 4^{\frac{n}{2}}$) = O($2^n$), It is O($2^n$) after simplify**

The total complexity of the DP algorithm is : O(n) + O(n)+ O(2^n). **It is O(2^n) after simplify**

Discuss the table:The practice running time matches to the theory complexity O(2^n)

when n_1=20 ,the number of Call the T(i,j)is m_1=349525, the running time t_1 = 0.29

when n_2=30 ,the number of Call the T(i,j)is m_2=357913941, the running time t_2 = 329.96

According the theory **O(2^n),**

$$\frac{2^{n_2}}{2^{n_1}} = \frac{2^{30}}{2^{20}} = 2^{10} = 1024 ,$$

According the practice running time

$$\frac{t_2}{t_1} = \frac{329.96}{0.29} = 1137 \approx \mathbf{1024}$$

So,the practice matches to the theory.

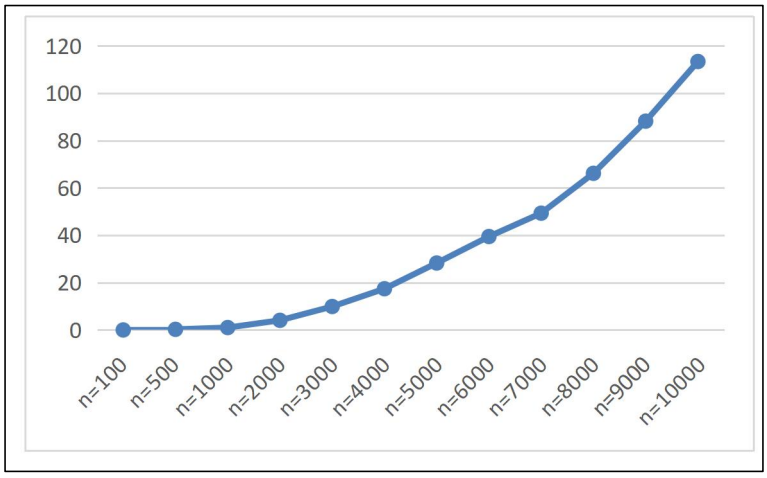
3. [20] Implement a dynamic programming algorithm (that uses a table to avoid recomputation) to compute the maximum sum of lengths that can be achieved. Include code in your report.

Python Code:

```
# from symbol import testlist_statest_tabler_expr
import numpy as np

def timberDPAlgo(timber_array):
    array_length = len(timber_array)
    DP_table = np.ones((array_length, array_length))

    for i in range(array_length):
        DP_table[i, i] = timber_array[i]

    for i in range(array_length-1):
        DP_table[i, i+1] = max(timber_array[i], timber_array[i+1])

    for k in range(2, array_length):
        for i in range(array_length-k):
            j = i + k
            DP_table[i, j] = max((timber_array[i]+min(DP_table[i+2, j], DP_table[i+1, j-1])),
(timber_array[j]+min(DP_table[i+1, j-1], DP_table[i, j-2])))

    return int(DP_table[0, array_length-1])

timber_array = [33, 28, 35, 25,29,34,28,32]
timber_array = np.asarray(timber_array)

result = timberDPAlgo(timber_array)

print('Timber: ', timber_array)
print('Maximum sum length is :', result)
```

4. [10] Run the code from the previous algorithm on a few different sizes of n to characterize the growth in runtime as n increases. Use a RNG to create the arrays. Provide a table or plot of your results and discuss.

Number of Segments n	Running Time t（seconds）
n=100	0.01
n=500	0.28
n=1000	1.03
n=2000	4.09
n=3000	9.92
n=4000	17.43
n=5000	28.29
n=6000	39.48
n=7000	49.34
n=8000	66.18
n=9000	88.21
n=10000	113.40



The Complexity of the DP Algorithm for T(i,j) : O(n^2)

Because as the DP Algorithm above:

The complexity of these two lines as below is O(n).

```
for i in range(array_length):
    DP_table[i, i] = timber_array[i]
```

These lines are for initialize the base case:T(i, j) = l_i when j = i

The complexity of these two lines as below is O(n).

```
for i in range(array_length-1):
    DP_table[i, i+1] = max(timber_array[i], timber_array[i+1])
```

These lines are for initialize the base case:T(i, j) = max(l_i, l_j) when j = i + 1

The complexity of these two lines as below is O(n^2).

```
for k in range(2, array_length):
    for i in range(array_length-k):
```

These lines are for computing the case:

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

There are one outer for loop and one inner loop in the DP Algorithm to compute DP_table[i, j].

The total complexity of the DP algorithm is : O(n) + O(n)+ O(n^2). It is O(n^2) after simplify .

Discuss the table:The practice running time matches to the theory complexity O(n^2)

when n_3=1000, the running time t_3= 1.03

when n_4= 10000, the running time t_4= 113.4

$\frac{{n_4}^2}{{n_3}^2}$ = 100, and $\frac{t_4}{t_3}$ ≈ 100

So,the practice matches to the theory.

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 code in your report.

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 $l_i \geq l_j$.)

Python Code:

```
# from symbol import testlist_statest_tabler_expr
import numpy as np
```

```
def timberDPAlgoTraceback(timber_array):
    array_length = len(timber_array)
    DP_table = np.ones((array_length, array_length))
    Traceback_table = np.ones((array_length, array_length))
    left = -1 # left
    right = -2 # right

    for i in range(array_length):
        DP_table[i, i] = timber_array[i]
        Traceback_table[i, i] = timber_array[i]

    for i in range(array_length-1):
        DP_table[i, i+1] = max(timber_array[i], timber_array[i+1])
        if timber_array[i] >= timber_array[i+1]:
            Traceback_table[i, i+1] = -1    # left
        else:
            Traceback_table[i, i+1] = -2    # right

    for k in range(2, array_length):
        for i in range(array_length-k):
            j = i + k
            DP_table[i, j] = max((timber_array[i]+min(DP_table[i+2, j], DP_table[i+1, j-1])),
(timber_array[j]+min(DP_table[i+1, j-1], DP_table[i, j-2])))

            if (timber_array[i]+min(DP_table[i+2, j], DP_table[i+1, j-1])) >= (timber_array[j]+min(DP_table[i+1, j-1],
DP_table[i, j-2])):
                Traceback_table[i, j] = -1  # left
            else:
                Traceback_table[i, j] = -2  # right

    temp_i = 0
    temp_j = array_length - 1
    output_order = np.ones(array_length).astype(np.int)
    idx_output_order = 0
    while temp_i != temp_j:
        direction = Traceback_table[temp_i, temp_j]
        if direction == -1: # left
            output_order[idx_output_order] = temp_i+1
            temp_i = temp_i + 1
        else:    # right
            output_order[idx_output_order] = temp_j+1
            temp_j = temp_j - 1
        idx_output_order = idx_output_order + 1

        if temp_i == temp_j:
            output_order[idx_output_order] = temp_j+1

    return int(DP_table[0, array_length-1]), output_order

timber_array = [33, 28, 35, 23, 23, 25, 37, 40, 42, 24, 38, 29, 22, 40, 36, 42, 39, 37, 45, 32]
timber_array = np.asarray(timber_array)

result, output_order = timberDPAlgoTraceback(timber_array)

print(result)
print(' '.join(map(str, output_order)))    # without comma
```

6. [20] Demonstrate that your code works correctly by showing its results on the following instance (n = 20):

[33, 28, 35, 23, 23, 25, 37, 40, 42, 24, 38, 29, 22, 40, 36, 42, 39, 37, 45, 32]

Output Format

The output consists of two lines:

- The first line prints maximum sum of lengths that you can take.
- The second line prints the segment numbers in the order in which they are taken.

For the example [5, 6, 9, 7] the output would look like:

```
14
1 2 3 4
```

My Output

```
350
1 2 3 20 19 18 4 5 17 16 15 14 13 6 7 8 9 10 11 12
```

7. [10] Demonstrate that your code works for larger values of n by correctly solving the instance of the problem in the Timber Verification quiz on Canvas. For this, you only need to give the maximum sum of lengths, not the traceback step.

I have finised the Verification on the canvas.

The result is correct.