Assignment 11

Name: DONGWOOK LEE

Problem 11.1 *Longest ordered subarray* (4 points)

```python
def set_cp_array(A, B):
    cp = 1
    B.append(cp)
    for i in range(0, len(A)-1):
        if A[i] < A[i+1]:
            cp += 1
        B.append(cp)


def set_longest_ordered_array(A, B, C):
    set_cp_array(A, B)
    to_append = A[0]
    for i in range(1, len(B)):
        if B[i-1] == B[i]:
            if A[i-1] < A[i]:
                to_append = A[i-1]
            else:
                to_append = A[i]
        else:  # where B[temp] < B[i]
            C.append(to_append)
            to_append = A[i]
    if to_append > C[len(C)-1]:
        C.append(to_append)


if __name__ == '__main__':
    A = [8, 3, 6, 50, 10, 8, 100, 30, 60, 40, 80]
    B = list()
    C = list()

    set_longest_ordered_array(A, B, C)

    # Check by print
    for i in range(0, len(C)):
        print(C[i], end=' ')
```

Result:

```
C:\Users\danie\PycharmProjects\Long
3 6 8 30 40 80
Process finished with exit code 0
```

With our input array A with elements {8, 3, 6, 50, 10, 8, 100, 30, 60, 40, 80},

we make a new array B with elements {1, 1, 2, 3, 3, 3, 4, 4, 5, 5, 6}

(where i^{th} component of B expresses the size of longest ordered subarray made up by elements A[0]~A[i])

➔ last element of B expresses the size of longest ordered subarray made up by all elements in array A (which is the size of our final result)

Then, regroup the elements in array A according to the values in array B.

We group the elements in array A which have same value in array B at same position.

(Result: {{8, 3}, {6}, {50, 10, 8}, {100, 30}, {60, 40}, {80}})

In each group, we select the element with min value and then append it to array C

{{8, 3}, {6}, {50, 10, 8}, {100, 30}, {60, 40}, {80}} ➔ C : {3, 6, 8, 30, 40, 80}

Problem 11.2 *Sum in triangles*

(a) (5 points)

```python
if __name__ == '__main__':
    A = list()  # Input  Array
    A_sum = list()
    B = list()  # Result Array
    temp = list()
    row_num = int(input("How many rows do you have ? >> "))

    # Get Triangular integer inputs
    # and put them into array
    for i in range(0, row_num):
        temp = list(map(int, input().split()))
        A.extend(temp)
        A_sum.extend(temp)

    # Sum up from below to top
    current_row = row_num - 1
    while current_row > 0:
        for i in range(int(((current_row)*(current_row-1))/2), int(((current_row+1)*(current_row))/2)):
            selected_int = max(A_sum[i+current_row], A_sum[i+current_row+1])
            A_sum[i] += selected_int
        current_row -= 1

    # Now we have our maximum sum value
    print(A_sum[0])

    # Find the corresponding elements current_row = 1
    idx = 0
    B.append(A[idx])
    for i in range(1, row_num):
        left_idx = idx + i
        right_idx = idx + i + 1
        if A_sum[left_idx] > A_sum[right_idx]:
            B.append(A[left_idx])
            idx = left_idx
        else:
            B.append(A[right_idx])
            idx = right_idx

    for i in range(0, len(B)):
        print(B[i], end=" ")
    print()
```

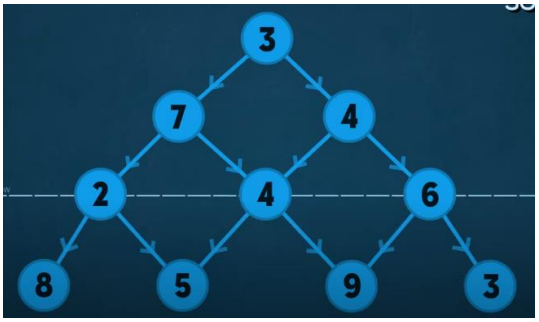
```
How many rows do you have ? >> 5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
30
7 3 8 7 5

Process finished with exit code 0
```

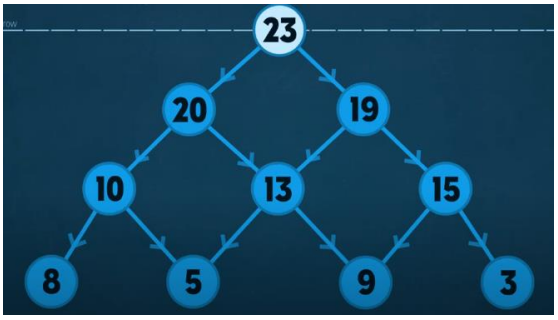
First, we get triangular inputs of integers and put them into an array A. (Also make array A_sum for further procedure)

Then in array A-sum, starting from the 2nd row from below to the top,

we sum up the value of each node and the value of one of its children (larger one between left and right children)

Ex)





Now reversely in array A-sum, starting from top row to below,

follow the path to its child where it has larger value than its sibling node.

Finally, we just have to append the elements in array A

which are in the same positions with elements in A-sum's path into our result array C.

(b) (2 points)

If we are using Brute-Force algorithm, in which we find out all the possible solution paths and then compare them, we will be getting 2^{m-1} number of paths where m equals the total row number of input triangle.

And then we compare all the possibilities, so time complexity of Brute-Force algorithm will be $O(2^n)$ (where n = m-1)

Our algorithm, however, is indeed faster than Brute-Force algorithm as we can conclude from below.

In the python code, except the part printing out all the result elements, we have 3 major loops and they have corresponding time complexity.

```
1)  for i in range(0, row_num):                                                                  # O(m)
        ...
2)  while current_row > 0:                                                                       # O(m-1)
        for i in range(int(((current_row)*(current_row-1))/2), int(((current_row+1)*(current_row))/2)):   # O(n)
            ...                                                                                  # where n=# of elements in 2nd last row = m-1
3)  for i in range(1, row_num):                                                                  # O(m-1)
        ...
```

For all the other parts of code has constant time complexity, we don't need to consider them.
Conclusively, the code is wholly bounded to
$O((m-1)*(n))$ (The most time complexed loop) $= O((m-1)*(m-1)) = O(m^2)$,
which is much faster than $O(2^n)$ complexity of Brute-Force algorithm.

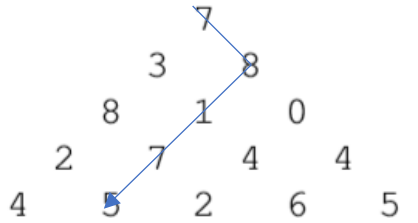
(c) (1 point)

I can give a counter example which shows that greedy algorithm does not give the maximum sum value of a certain path.

Assume we have our triangular data set as below

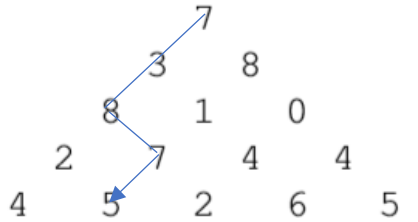
If we are using greedy algorithm to find out the path which yields our final result as largest sum, we can just go from top node to below following the bigger child (whether left or right).

In this problem, this greedy approach will make our result path as below,



resulting our sum as 7 + 8 + 1 + 7 + 5 = 28

In real, however, the maximum path sum that we can get from this triangular input is 30,



following this path.

At last, we can conclude that greedy algorithm is not feasible for this problem.