Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Tiring Resorts

PNPenguin has grown tired of delivering ice-cakes and has decided to start a new resort business that serves penguins who want to go on vacations. They have chosen two locations for their resort business – Premium Resort and Common Resort. Unfortunately due to understaffing, each resort can only serve one penguin at a time. There are n total penguins wanting to go on vacations and they don't mind which resort they end up going to. The i-th penguin stays $d[i]$ days and will pay $2 \cdot c[i]$ per day at Premium Resort or $c[i]$ per day at Common Resort. After each penguin leaves, PNPenguin needs ℓ days (for Common Resort) or 2ℓ days (for Premium Resort) to clean up the resort and get it ready to serve the next penguin.

PNPenguin wants to find a way to assign the n penguins to resorts to maximize their profit (assume no operation costs). Note that, although a bit scummy, PNPenguin does not have to assign all the vacationers to resorts.

(a) Design a dynamic programming algorithm that determines the maximum profit PNPenguin can make over T days by serving the penguin vacationers. Please provide the following:

(i) Subproblem definition.

(ii) Recurrence Relation (make sure to include base cases!).

(iii) Time Complexity Analysis.

(iv) Space Complexity Analysis.

(b) Now suppose each penguin has a specific start date and end date; that is, the ith penguin will arrive on $start[i]$ and leave on $end[i]$. Two penguins can start and end on the same date.

Assume for this subpart that all vacationers are super-duper tidy, meaning that clean up is no longer needed after each penguin leaves. Modify your algorithm from part (a) to find the new maximum profit PNPenguin can make.

2 Optimal Set Covering

The Set Cover problem is defined as follows: given a collection $S_1, S_2 \dots S_m$ where each S_i is a subsets of the universe $U = \{1, 2, \dots, n\}$, we want to pick the minimum number of sets from the collection such that their union contains all of U. In this problem we will use dynamic programming to come up with an exact solution.

(a) How long does a brute force solution of trying all the possibilities take? How large can m be compared to n in the worst case?

(b) As you will learn later on in the class, there does not exist a sub-exponential time algorithm for Set Cover. However, we can try to do better than exponential time in m by aiming for exponential time in only n instead. Describe a dynamic programming algorithm that accomplishes this in $O(2^n \cdot nm)$ time.

(a) Define your subproblem.

(b) Write your recurrence relation.

(c) Describe the order in which we should solve the subproblems.

(d) Analyze the runtime of this dynamic programming algorithm.

(e) Analyze the space complexity.

3 Balloon Popping Problem.

You are given a sequence of n balloons with each one of a different size. If a balloon is popped, then it produces noise equal to $g_{left} \cdot g_{popped} \cdot g_{right}$, where g_{popped} is the size of the popped balloon and g_{left} and g_{right} are the sizes of the balloons to its left and to its right. If there are no balloons to the left, then we set $g_{left} = 1$. Similarly, if there are no balloons to the right then we set $g_{right} = 1$, while calculating the noise produced.

After popping a balloon, the balloons to its left and right become neighbors. (Note that the total noise produced depends on the order in which the balloons are popped.)

In this problem we will design a polynomial-time dynamic programming algorithm to compute the the maximum noise that can be generated by popping the balloons.

Example:

Input (Sizes of the balloons in a sequence): ④ ⑤ ⑦
Output (Total noise produced by the optimal order of popping): 175

Walkthrough of the example:

- **Current State** ④ ⑤ ⑦
  Pop Balloon ⑤
  Noise Produced $= 4 \cdot 5 \cdot 7$

- **Current State** ④ ⑦
  Pop Balloon ④
  Noise Produced $= 1 \cdot 4 \cdot 7$

- **Current State** ⑦
  Pop Balloon ⑦
  Noise Produced $= 1 \cdot 7 \cdot 1$

- **Total Noise Produced** $= 4 \cdot 5 \cdot 7 + 1 \cdot 4 \cdot 7 + 1 \cdot 7 \cdot 1$.

(a) Define your subproblem as follows:

$$C(i, j) = \text{maximum amount of noise produced by popping balloons in the sublist } i, i+1, \ldots, j \text{ first before the other balloons.}$$

What are the base cases? Are there any special cases to consider?

(b) Write down the recurrence relation for your subproblems $C(i, j)$.

Hint: suppose the k-th balloon (where $k \in [i, j]$) is the last balloon popped, after popping all other balloons in $[i, j]$. What is the total noise produced so far?

(c) What is the runtime of a dynamic programming algorithm using this recurrence? What is its space complexity?

Note: you may assume that all arithmetic operations take constant time.

(d) Is it possible to further optimize the space complexity? In either case, explain your reasoning.

4 Finding More Counterexamples

In this problem, we will explore why we had to resort to Dynamic Programming for some problems from lecture instead of using a fast greedy approach. Give counter examples for the following greedy algorithms.

(a) For the travelling salesman problem, first sort the adjacency list of each vertex by the edge weights. Then, run DFS N times starting at a different vertex $v = 1, 2, \ldots N$ for each run. Every time we encounter a back edge, check if it forms a cycle of length N, and if it does then record the weight of the cycle. Return the minimum weight of any such cycle found so far.

(b) For the Longest Increasing Subsequence problem, consider this algorithm: Start building the LIS with the smallest element in the array and go iterate through the elements to its right in order. Every time we enounter an element $A[i]$ that is larger than the current last element in our LIS, we add $A[i]$ to the LIS.

(c) Can you construct a counter example for the previous algorithm where the smallest element in the array is $A[0]$?