Programming Assignment #3

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. **You may not submit code other than that which you write yourself or is provided with the assignment.** This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment. **Do not make your code publicly available (eg github repo) as this enables others to cheat and you will be held responsible.**

Problem Description: Unyielding crows

You have recently downloaded an old farm simulator game. When you start the game, you find yourself in charge of a farm with a bit of a crow problem. Each morning thousands of crows descend on the farmland and decimate the crops. Eventually the birds will eat everything and all the farm plots will be in ruin.
Your objective is simple: with what time remains you are to maximize the total amount of food you harvest.

The farm is divided up into N plots. Initially, all N plots are in good condition and can be harvested from each day. The farm is long and thus the plots can be represented as N squares on a line as depicted in Figure 1. In total, you harvest p_i food from each plot i, each day.

The crows will eventually devour the entire farm; thus, your objective is to maximize the amount of food you harvest before all the plots are devastated.

At the beginning of the turn or day a wave of crows fly in from either the East or West. Before the turn/day starts, you can set out a scarecrow between two neighboring plots that have not been destroyed. Subsequently, the crows will fly the remaining farm either from east or west – devouring all the plots before the scarecrow. At the end of the turn/day, you count the total amount of food you harvested from the side of the scarecrow **that the crows did not ravage** and pick up the scarecrow to move to a new location. The same process is repeated every turn/day until the entire farm in ruins.

You should be particularly careful when choosing the location of the scarecrow. It's known that crows are quite intelligent, and they will find the more bountiful half, and fly it from the direction that minimizes the amount of food your people harvest in the **long run**. That is, the crows will fly in from a direction that *minimizes the amount of food you harvest (after you've chosen a location for the wall) in the CURRENT AND ALL FUTURE DAYS.*

Example: Consider $N = 6$ plots and $p_1 = 3, p_2 = 2, p_3 = 4, p_4 = 2, p_5 = 6, p_6 = 8$. The best option for you is to place a scarecrow between plots 4 and 5. The crows fly from east and devour plots 5 and 6 and that allows you to harvest 11 food (from plots $1, 2, 3$ and 4) by the end of the day. At the beginning of the next day you place a scarecrow between the plots 2 and 3. The crows

Figure 1: If the crows fly from the west, plots 5 and 6 will remain healthy and the total food for the current day will be $p_5 + p_6$. If crows fly from the east, plots 1,2,3 and 4 will remain healthy and the total food for the current day will be $p_1 + p_2 + p_3 + p_4$.

fly from east and devour plots 3 and 4 allowing you to harvest 5 food (from plot 1 and 2) by the end of the day. At the beginning of the third turn/day, you place a scarecrow between 1 and 2. The crows fly from west and devour plot 1 allowing you to harvest 2 food from plot 2. There are no more neighbor plots for you to place a scarecrow thus your farm is in ruin, but you harvest a total amount of food harvested equal to 18.

Instructions

Implement a dynamic programming algorithm to find the maximum amount of food to be obtained given an array of plots. The first part of this assignment will be to generate your report before you begin code implementation. This includes finding the recurrence formula and proving your algorithm's complexity. The second part of this assignment will be your actual code implementation. You will be given very little starter code in the form of `Driver3.java` and `Assignment3.java`, as well as some small test cases. Your solution should be built in the `maxFoodCount` function given to you in `Assignment3.java`. You may use `Driver3.java` as a means to test your code. Simply pass the name of a text file (just plot food count separated by spaces) as an argument to test.

Part 1: Report [20 points]

Write a short report that includes the information below. There are reasonable $O(n^3)$, $O(n^2 \log n)$, and $O(n^2)$ solutions. Faster run-time will give more points!
Note: You might find the $O(n^2)$ solution challenging to come up with.
Note: You can get full credit on the Code part of the assignment with a $O(n^2 \log n)$ solution.

(a) Explain what the dynamic programming subproblems are, and provide a recursive formula for OPT.

(b) Provide a succinct argument of correctness, and explain and justify the complexity of your algorithm.

Part 2: Code [80 points]

This part of your assignment will be graded both on correctness, and time-complexity.

- Given an `int[] plots`, it is your job to return the maximum food count in function `maxFoodCount`. Each index i in the array has an integer value corresponding to p_i. Your value range is as follows, $1 \leq p_i \leq 30000$.

- For the first 70% percent of the test case $2 \leq N \leq 500$. (A properly implemented $O(n^3)$ solution should suffice for these test cases.)
- For the next 30% percent of the test case $2000 \leq N \leq 2500$. (Both $O(n^2)$ or $O(n^2 \log n)$ should pass these test cases.)
- Be sure to practice good programming style, as poor style may result in a deduction of points (e.g. do not name your variables foo1, foo2, int1, and int2).

Memory limit: 128MB

Input File Format

The input file is made up of a single line of numbers. Each number represents a plot on the farm and how much food it grows per turn.

For example, if the input file is as follows:

1 2 7 4 5 6

The plot furthest to the West(left) grows 1 food per day, and the plot furthest to the East (right) grows 6 food per day.

We will parse the input files for you, so you don't need to worry too much about the input file format except when trying to make your own test cases.
We will use different inputs for testing, but test cases will have valid inputs so you do not need to handle invalid inputs throughout your whole program.

Getting Started:

(a) Download the starter material from canvas.

(b) Do an initial compile of the starter code in your favorite Java IDE or on the ECE LRC Linux machines(reccomended), see note below on how to compile.

(c) Test your code using the inputs given and the Driver class.

(d) Submit your code to Gradescope to validate your code. You have unlimited submissions before the due date. Your latest submission will be the only submission we look at.

Running Code

Compile Code: *javac *.java*

Run Driver: *java Driver input/input01.txt*
You will need to test and validate your algorithm on your own to check for correctness.

What To Submit

You should submit to Gradescope `Program3.java`. Please do not submit `Driver.java`. Failure to follow these instructions will result in a penalty of up to 10 points.

Your PDF report should be legibly scanned and submitted to Gradescope. Both your code and PDF report must be submitted by 11:59 PM on Monday, December 5th, 2022. If you are unable to complete the assignment by this time, you may submit the assignment late until Wednesday, December 7th, 2022 at 11:59 PM for a 20 point penalty.