Homework 3

Student: Haoyuan Du, Aaron Marshall

Purpose:

- Practice writing recurrences (i.e., recursive solutions).
- Learn to design and implement algorithms using dynamic programming techniques.
- Continue to develop your algorithm analysis skills.

General Homework Policies:

- This homework assignment is due by the deadline given in Canvas. Late assignments will not be accepted and will receive a 0.
- Submit the `pdf` (it should generated by modifying the the LaTeX file for this assignment) and the completed `Interleaver.java` file through Canvas. If you choose to complete the extra credit you will also submit the file `MaxSum.java`. Before submitting please zip all the files together and submit one `.zip` file. See the *Additional Programming Instructions* section below for more detailed instructions.
- You will be assigned a partner for this assignment. Only one assignment should be submitted and you and your partner will both receive the same grade. Make sure to include your partner's name on the homework. Your assignment should be a true joint effort, equally created, and understood by both partners.
- You are not allowed to consult outside sources, other than the textbook, your notes, the Java API and the references linked from Canvas (i.e., no looking for answers on the internet).[1]
- Getting *ANY* solutions from the web, previous students etc. is *NOT* allowed.[1]
- You are not allowed to discuss this assignment with anyone except for your partner (if you have one) or the instructor.[1]
- Copying code from anywhere or anyone is not allowed (even previous code you have written). Allowing someone to copy your code is also considered cheating.[1]
- Your work will be graded on correctness and clarity. Write complete and precise answers and show all of your work.
- Questions marked (PRACTICE) will not be graded and do not need to be submitted. However it is highly recommended that you complete them.

[1] *See the section of the syllabus on academic dishonesty for more details.*

Homework Problems:

1. (5 points) Recall that a subarray is similar to a subsequence except that the elements must be consecutive. For instance, if A is $[5, 15, -30, 10, -5, 40, 10]$ then $[15, -30]$ is a subarray but $[5, 15, 40]$ is not. Use **dynamic programming** to give a linear-time algorithm for maximum sum subarray problem (seen before in class and on assignment 2):

Input: A list of numbers, $a_1, a_2, \ldots, a_n$.
Output: The **sum** of the subarray with maximum sum
Note that if the array contains only negative numbers then the answer should be negative. For example, if the input is `[-1, -2, -3]` then your method should return -1 (i.e. a subarray has length at least 1).

Your solution to must use **dynamic programming** (There should be a table!) and you should write your solution in the following form:

(a) (1 point) Define your subproblems (the entries of the table) in words. For example, in the domino tiling problem the definition is $D(i)$ = the number of ways to tile a $2 \times i$ checkerboard.

> **Solution:** L(i) = the maximum sum of the subarray at index i in array A.

(b) (2 points) State the recurrence for the entries of your table (NOT for the running time), and explain in words why it's true. For example, in the domino tiling problem the recurrence for the entries is $D(i) = D(i-1) + D(i-2)$.

> **Solution:** $L(i) = max\{L(i-1) + A(i), A(i)\}$
> Because the elements must be consecutive in subarray, at A[i], the maximum sum will either be "adding this A[i] to the previous subarray" or "only taking A[i] and throwing the previous subarray". It depends on which answer is larger. By doing this, we can make sure that as i is increasing, we always record the best answer so far in our L table. So it's true.

(c) (1 point) Give pseudocode for filling in the table AND returning the solution. Do NOT use memoization.

> **Solution:**
> function MaxSumSubarray($A[a_1, a_2..., a_n]$)
> L[1]=A[1] // Create table L
> for i=2, $i <= n$, i++ do
> L[i]=max(L[i-1]+A[i],A[i]) // Record the maximum sum at index i
> max=L[1]
> for i=2, $i <= n$, i++ do // Find the max in L
> if $L[i] > max$
> max=L[i]
> return max

(d) (1 point) Analyze the running time (give a Θ-bound AND justification).

> **Solution:** Our code has two for-loops which both are executed n times. In both loops, we do constant time of work. So it's n+n. Meanwhile, other parts take constant time. So it's 2n+c in total which is $\theta(n)$.

(e) (1 point) **(EXTRA CREDIT)** Implement your pseudo-code from part (c). Add code to the `maxSumSubarray` method in the `MaxSum.java` file provided with the assignment. Note

that you will receive NO credit if your code does not implement your pseudo-code and you must use dynamic programming (there must be a table!).

2. (5 points) Recall the interleaving problem from class. For strings $X = x_1 \dots x_n$, $Y = y_1 \dots y_m$ and $Z = z_1 \dots Z_{m+n}$, we say that Z is an *interleaving* of X and Y if it can be obtained by interleaving the bits in X and Y in a way that maintains the left-to-right order of the bits in X and Y. For example, if $X = abc$ and $Y = dcab$ then $x_1x_2y_1x_3y_2y_3y_4 = abdccab$ is an interleaving of X and Y whereas $acdcbab$ is not.

   We will use the following subproblem definition. Let $I(i, j) = TRUE$ if $z_1 \dots z_{i+j}$ is an interleaving of $x_1 \dots x_i$ and $y_1 \dots y_j$ and $FALSE$ otherwise. We came up with a recurrence for $I(i, j)$ for each of the following cases.

   1. $I(i, j) = I(i - 1, j)$ if $z_{i+j} = x_i \neq y_j$
   2. $I(i, j) = I(i, j - 1)$ if $z_{i+j} = y_j \neq x_i$
   3. $I(i, j) = I(i, j - 1) \vee I(i - 1, j)$ if $z_{i+j} = x_i = y_j$
   4. $I(i, j) =$ False if $z_{i+j} \neq x_i, z_{i+j} \neq y_i$

   You will implement a solution to the interleaving problem using dynamic programming and the above subproblem definition. For this problem you will add code to the `Interleaver.java` file provided with the assignment. Note that you are welcome to add additional private methods or data fields but you may not modify the method signature of the constructor or the `isInterleaved` method. Your solution should not print anything.

   (a) Implement the code for the above dynamic programming solution by adding code to the `isInterleaved` method. You must use memoization.

   (b) As a comment at the top of the `isInterleaved` method, analyze the running time of your algorithm in terms of $n = |X|$ and $m = |Y|$ (use O notation and explain your answer). Note that your implementation should have the best running time possible using the given subproblem and recurrence!

   (c) (1 point) **(EXTRA CREDIT)** Write a method that returns the actual interleaving by adding code to the `getSolution` method. Note that this method must call the method you wrote for part (a) to "fill the table". This method must not change the overall running time of the algorithm given above (i.e., it shouldn't take longer in terms of O notation than the time to "fill the table"). Read the documentation for the method carefully to determine what you should be returning!

   Hints: The solution above does not contain a base-case. Think about what the base case should be - you will need multiple base-cases. Also, your book (and the subproblem definition above) use strings/arrays that start at index 1. However in Java, strings/arrays start at index 0. When implementing your code you will need to account for this difference.

3. **(PRACTICE)** You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \dots < a_n$, where each a_i is measured from the

starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You'd ideally like to travel 200 miles a day, but this may be impossible to do exactly (depending on the spacing of the hotels). If you travel x miles during a day, the penalty for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty - that is, the sum, over all travel days, of the daily penalties. Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

Additional Programming Instructions:

Note that your code will be automatically run on a standard set of test cases. In order to ensure that you do not lose points, follow the instructions below.

- Your code must compile without any errors using the version of Java on the lab computers. If your code does not compile you will not receive any points for the assignment.
- Do not modify any of the methods signatures (i.e. name, return type or input type). Note that you are always welcome (and encouraged) to add additional methods but these will not be run directly by the test code.
- You are not allowed to use packages (e.g. no statement `package ...` at the top of your file).
- No extra folders or files in your submission. Zip up only the files you need to submit not the folder they are in.
- Your solution should not print anything unless explicitly instructed to.

Grading Criteria:

Your work will be graded on both correctness **and clarity**. Write complete and precise answers and show all of your work. Your pseudo-code and proofs should follow the guidelines posted on Canvas and discussed in class.

Grading Criteria for Problem 1:

Component	Requirement for Full Credit
Subproblem Definition (1pt)	The definition is clear. It includes full sentences and should be easily understood by any student in the class.
Recurrence (2pts)	The recurrence is clear, concise and correct. Make sure to include at least one base case! It is described in full sentences and should be easily understood by any student in the class. It includes a justification for WHY the recurrence is correct.
Pseudo-code (1pt)	The pseudo-code is clearly written following the guidelines discussed in class.
Running Time Analysis (1pt)	The solution gives both a bound and an explanation. Both are clearly written and correct (for the algorithm given!).
Implementation (EXTRA CREDIT) (1pt)	Your code will be run on a standard set of test cases. Make sure to test your code thoroughly! Note that you will lose points if you do not follow the general style guidelines given in the syllabus or if your implementation changes the running time of your pseudo-code. You will receive NO credit if your code does not implement your pseudo-code and you must use dynamic programming (there must be a table!).

Grading Criteria for Problem 2:

Component	Description
Style & Documentation (1 pts)	I'll be watching for style issues as well as correct output. See the syllabus for some general style guidelines (e.g. your code should be well-documented).
Big-O Analysis (1 pt)	Make sure to justify your analysis as well as give a bound (i.e., explain where your bound comes from and why it is correct). Your implementation must have the best running time possible using the given subproblem and recurrence.
Correct Output on Test Cases* (3 pt)	Your code will be run on a standard set of test cases. Make sure to test your code thoroughly!
EXTRA CREDIT (1pt)	Your `getSolution` method will be run on a standard set of test cases. Make sure to test your code thoroughly! Note that you will lose points if you do not follow the general style guidelines given in the syllabus or if your implementation changes the running time of the algorithm. Your method must use the table created by your `isInterleaved` method to find the solution.

*Note that if your code does not implement the given dynamic programming solution you will receive no credit regardless of whether the test cases execute correctly. Remember you are required to use memoization!