Please enter your name and uID below.

Name:

uID:

Collaborators, if any, and how you collaborated:

Submission notes

- Due at 11:59 pm on Thursday, September 29.
- Solutions must be typeset using one of the template files. For each problem, your answer must fit in the space provided (e.g. not spill onto the next page) *without* space-saving tricks like font/margin/line spacing changes.
- Upload a PDF version of your completed problem set to Gradescope.
- Teaching staff reserve the right to request original source/tex files during the grading process, so please retain these until an assignment has been returned.
- Please remember that for this problem set, you are allowed to collaborate in detail with your peers, as long as you cite them. However, you must write up your own solution, alone, from memory. If you do collaborate with other students in this way, you must identify the students and describe the nature of the collaboration. You are not allowed to create a group solution, and all work that you hand in must be written in your own words. Do not base your solution on any other written solution, regardless of the source.

1. (Narrow art gallery, 15pts) *Describe* an efficient dynamic programming algorithm for the narrow art gallery problem, discussed in written part 1 and the programming problem.

   To *describe* a dynamic programming algorithm in this class, it is sufficient to give a recursive formula solving the problem,[1] give an English description of the underlying recursive function being solved,[2] describe the data structure you would use to memoize your formula, describe in what order you could fill memo, and describe the running time of your algorithm.

   Pseudocode is not required, but you may provide pseudocode for your DP algorithm *instead* of a recursive formulation, as long as you still give an English description of the underlying recursive function being solved and the other requirements above are either clear in your code or answered directly.

[1]For this problem, this means writing your answer to part 1 as a formula.

[2]E.g. What do the parameters mean? What subproblem does does your formula return the answer to? An example of this would be "Let LIS(i, j) return the length of the longest increasing subsequence from index j onward, where everything must be larger than the element at index i."

2. (String splitting, 40pts)

   Solve the following two-part problem (problems 2a and 2b in the book). You should be able to fit concise but complete descriptions of algorithms for both parts on one page, but it is also OK to use two pages for this problem.

   > Describe efficient algorithms for the following variants of the text segmentation problem. Assume that you have a subroutine IsWord that takes an array of characters as input and returns True if and only if that string is a "word". Analyze your algorithms by bounding the number of calls to IsWord.
   >
   > (a) Given an array $A[1..n]$ of characters, compute the number of partitions of A into words. For example, given the string `ARTISTOIL`, your algorithm should return 2, for the partitions `ARTIST·OIL` and `ART·IS·TOIL`.
   >
   > (b) Given two arrays $A[1..n]$ and $B[1..n]$ of characters, decide whether A and B can be partitioned into words at the same indices. For example, the strings `BOTHEARTHANDSATURNSPIN` and `PINSTARTRAPSANDRAGSLAP` can be partitioned into words at the same indices as follows:
   >
   > `BOT·HEART·HAND·SAT·URNS·PIN`
   > `PIN·START·RAPS·AND·RAGS·LAP`