Problem Set 6

This problem set is divided into two parts: Part A problems are programming tasks, and Part B problems are theory questions.

Part A questions are due Tuesday, December 1st at 11:59PM.
Part B questions are due Thursday, December 3rd at 11:59PM.

Solutions should be turned in through the course website in PDF form using LaTeX or scanned handwritten solutions.

A template for writing up solutions in LaTeX is available on the course website.

Remember, your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A: Due Tuesday, December 1st

1. (25 points) Placing Parentheses

You are given an arithmetic expression containing $n$ real numbers and $n - 1$ operators, each either $+$ or $\times$. Your goal is to perform the operations in an order that maximizes the value of the expression. That is, insert $n - 1$ pairs of parentheses into the expression so that its value is maximized.

For example:

- For the expression $6 \times 3 + 2 \times 5$, the optimal ordering is to add the middle numbers first, then perform the multiplications: $((6 \times (3 + 2)) \times 5) = 150$.
- For the expression $0.1 \times 0.1 + 0.1$, the optimal ordering is to perform the multiplication first, then the addition: $((0.1 \times 0.1) + 0.1) = 0.11$.
- For the expression $(-3) \times 3 + 3$, the optimal ordering is $((-3) \times 3) + 3 = -6$.

(a) (10 points) Clearly state the set of subproblems that you will use to solve this problem.
(b) (10 points) Write a recurrence relating the solution of a general subproblem to solutions of smaller subproblems.
(c) (5 points) Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem.
2. **(25 points) Building Blocks**

You have \( N \) rectangular building blocks. You want to build as tall a tower as possible by stacking some subset of the blocks in a vertical sequence. (Note that you may only use each block once.) For the tower to be stable, the base of each block must fit within the top of the block below it. Block \( i \) has dimensions \( l_i \times w_i \times h_i \) where \( l_i \), \( w_i \), and \( h_i \) are the length, width, and height of the block, respectively. Each block must be aligned with the coordinate axes and you may not rotate the blocks. Given these constraints, find the height of the tallest possible tower.

For example:

- Given three blocks with dimensions \( 4 \times 3 \times 3 \) and \( 5 \times 2 \times 2 \) and \( 5 \times 3 \times 5 \), the tallest tower is made by stacking the first block on top of the third block and has height 8. We could also stack the second block on top of the third block, but that would only have height 7. We cannot stack the first and second blocks together because the second block is longer while the first block is wider.

(a) **(10 points)** Clearly state the set of subproblems that you will use to solve this problem.

(b) **(10 points)** Write a recurrence relating the solution of a general subproblem to solutions of smaller subproblems.

(c) **(5 points)** Analyze the running time of your algorithm, including the number of subproblems and the time spent per subproblem. Hint: your overall algorithm should be \( O(N^2) \).

(d) **(optional)** Reduce the complexity of your algorithm to \( O(N \log N) \).

---

**Part B: Due Thursday, December 3rd**

**(50 points) Website Rankings**

In class, you saw that the length of the longest common subsequence can be computed in \( O(n^2) \) time for two strings \( x \) and \( y \) of length \( n \), using dynamic programming. For general \( x \) and \( y \), it is not known if this value can be computed in \( O(n^{1.999}) \) time. In this problem, you will learn how to compute it in \( O(n \log n) \) time for non-repetitive strings.

**Theory part (do not submit a solution to it).**

Definitions:

- Let \( z = (z_1, \ldots, z_n) \) be a sequence of integers. We say that \( t = (t_1, \ldots, t_k) \) is a subsequence of \( z \) if there is an sequence \( (i_1, \ldots, i_k) \) of \( k \) indices such that \( i_1 < i_2 < \ldots < i_k \), and for all \( j, t_j = z_{i_j} \).

- Let \( z = (z_1, \ldots, z_n) \) be a sequence of integers. We write \( \text{LIS}(z) \) to denote the length of the longest increasing subsequence of \( z \).

**Example:** \( \text{LIS}( (6, 2, 7, 5, 3, 4) ) = 3 \), and this corresponds to the subsequence \( (2, 3, 4) \).
Let \( x = (x_1, \ldots, x_n) \) and \( y = (y_1, \ldots, y_m) \) be sequences of integers. We write \( \text{LCS}(x, y) \) to denote the length of the longest common subsequence of \( x \) and \( y \).

**Example:** \( \text{LCS}( (6, 2, 7, 5, 3, 4), (5, 6, 1, 7, 3, 4) ) = 4 \), and this corresponds to the subsequence \( (6, 7, 3, 4) \).

Let \( z = (z_1, \ldots, z_n) \) be a sequence of integers. We say that \( z \) is non-repetitive if no integer appears twice in it.

**Example:** \( (5, 6, 1, 7, 3, 4) \) is non-repetitive, and \( (4, 6, 1, 7, 1, 3) \) is not.

Design algorithms for the following two problems:

1. Show how to compute \( \text{LIS}(z) \) for a sequence \( z \) of \( n \) integers in \( O(n \log n) \) time.
   
   **Hint 1:** Use the following sequence of arrays \( A_i \). Let \( A_i[j] \), where \( i, j \in \{1, 2, \ldots, n\} \), be the lowest integer that ends an increasing length-\( j \) subsequence of \( (z_1, \ldots, z_i) \). If \( (z_1, \ldots, z_i) \) has no increasing subsequence of length \( j \), then \( A_i[j] = \infty \).
   
   **Hint 2:** How can \( A_i \) be turned into \( A_{i+1} \) in \( O(\log n) \) time? How can \( \text{LIS}(z) \) be extracted from \( A_n \)?

2. Show how to compute \( \text{LCS}(x, y) \) for two non-repetitive integer sequences \( x \) and \( y \) of length \( n \) in \( O(n \log n) \) time.
   
   **Hint 1:** Reduce to the previous problem.
   
   **Hint 2:** Create a sequence \( z \) from \( y \) in the following way. Remove all integers that do not appear in \( x \), and replace the other ones by their index in \( x \). How is \( \text{LIS}(z) \) related to \( \text{LCS}(x, y) \)?

**Coding part (50 points).** Consider two web search engines A and B. We send the same query, say “6.006”, to both search engines, and in reply we get a ranking of the first \( k \) pages according to each of them. How can we measure the similarity of these two rankings? Various methods for this have been designed, but here, we’ll use the simplest of them: LCS, the length of the longest common subsequence of the rankings.

Your task is to write a program that uses the above \( O(n \log n) \) algorithm to compute LCS for two rankings. The program has to read the rankings from the standard input, and write their LCS to the standard output. We assume that pages are identical only if their URLs are identical. No URL appears twice in a ranking. You can use the Python dictionary to convert the input into an instance of the LIS problem.

**Input Format**

- **Line 1:** The number \( k \) of URLs we receive from each of the search engines.
- **Lines 2\ldots k + 1:** The list of \( k \) URLs received from A.
- **Lines \( k + 2 \ldots 2k + 3 \):** The list of \( k \) URLs received from B.
Sample Input

5
http://courses.csail.mit.edu/6.006/spring08/
http://courses.csail.mit.edu/6.006/fall07/
http://alg.csail.mit.edu/
http://courses.csail.mit.edu/6.006/fall09/
http://courses.csail.mit.edu/6.006/fall09/
http://courses.csail.mit.edu/6.006/spring08/
http://mit.worldcat.org/profiles/MITLibraries/lists/899062
http://courses.csail.mit.edu/6.006/fall07/

Output Format

**Line 1:** The value of LCS for the two sequences of URLs.

Sample Output

3

**Note:** More sample inputs and outputs are posted on the website.