Dynamic Programming – Presentation 2

Problem:

Given an ordered set S = { a1, a2, a3, ... , an }, you have to find the size of the largest subset of S, P = { ax1, ax2, ax3, ... , axm } such that the elements in P are in the same order, and the value of each element in P are strictly increasing.

For example, given the ordered set {1, 3, 3, 2, 4} the largest subsets would be {1, 2,4} or {1, 3, 4}, but not {1, 3, 3, 4}, so you would return 3.

Solution:

$$f(X_i, M) = \begin{cases} f(X_{i+1}, M) & \text{if } X_i <= M \\ \max(1 + f(X_{i+1}, X_i), f(X_{i+1}, M)) & \text{if } X_i > M \end{cases}$$

Where M is the largest element currently in the subset.

In Java, the recursive solution is:

```
private int[][] memo;
private int largestSubsetPublic(int[] set){
	int largestNumInSet = -1;
	for (int i = 0; i < set.length;i++){
		if (largestNumInSet< set[i])
			largestNumInSet= set[i];
	}
	memo = new int[set.length][largestNumInSet+2];
	for (int i = 0;i<memo.length;i++)
	{
		for (int j = 0;j<memo[0].length;j++)
			memo[i][j] = -1;
	}
	return maxSubset(set, 0, -1);
}
private int maxSubset(int[] set, int index, int largestNum){
	if (index >= set.length)
		return 0;
	if(memo[index][largestNum+1]>-1)
		return memo[index][largestNum+1];
	int returnValue = -1;
	if (largestNum>=set[index])
		returnValue = maxSubset(set, index+1,largestNum);
	else
		returnValue = Math.max(1 + maxSubset(set, index + 1, set[index]),
				maxSubset(set, index+1,largestNum));
	memo[index][largestNum+1] = returnValue;
	return returnValue;
}
```

And in PLT-Scheme (using the built in, map-based memoization library):

```
{define/memo (largestSubset set largestNum)
 (cond((null? set) 0)
	((>= largestNum (car set)) (largestSubset (cdr set) largestNum))
	(else (max (+ 1 (largestSubset (cdr set) (car set)))
		(largestSubset (cdr set) largestNum)))
  )
}
```

Assignment 1 – String Compression:

You're working for a company creating a new text compression program. You've been assigned the task of creating an algorithm that will give an estimate, before the compression program itself runs, how good a job that compression program will be able to do.

The compression algorithm works by taking a string, and finding any repeating sections, and replacing those repeating sections with just one copy (and noting how many copies of that section to produce). For example, the string "AAAGH!!!" can be expressed as "$(A)^3GH(!)^3$" while the string "WOOFWOOFMEOWMEOW" can be written as "$(W(O)^2F)^2(MEOW)^2$"
Your job is to take any input string and determine the smallest number of letters needed to rewrite it in this way (disregarding the notation describing the number of repetitions, and the added brackets).

Some example inputs and outputs:

Given "AAAGH!!!" you would output 4

Given "WOOFWOOFMEOWMEOW" you would return 7.

Given "3.14159Help! Help! I'm trapped in a universe factory!71214" you would return 51.