CS 374 **Spring 2015**

Homework 5

Nikhil Unni (nunni2)

2.

$$\text{Pal}(i, j, \text{str}) = \begin{cases} 1 & \text{if } i == j \\ \min_{i \le x \le j}(1 + \text{Pal}(x+1, j, str) \text{ if str[i:x] is a palindrome}) & \text{else} \end{cases}$$

Since every call to the function has to include the starting index, the algorithm just chooses any of the (n-i) available positions, and if the substring from i to that position is a palindrome, try that one. The algorithm tries all such possibilities. There's no fear of a bad value, since there's a way to come up with a palindrome no matter what choice you do (all single characters). So the function just gets the minimum of the choices, and memoizes along the way.

You can call the function on a string, str, of length $j + 1$, with $\text{Pal}(0, j, str)$.

The memoization structure would just be a 2D array of size nxn, where n is the length of the string. Because all of the possible problems are just substrings of the original string, we know that there are only $O(n^2)$ unique subproblems. A possible iterative solution could be to build up a 2D array of all (i,j) pairs that are palindromes, and then iterate through the array to find the shortest path to the last index of the string.

Because there are $O(n^2)$ unique subproblems, and it takes $O(n)$ time to verify a palindrome, the total time complexity of the algorithm is $O(n^3)$.