Homework 5

Due Wed, August 15th at 11:59pm

Name: Zachary McNulty
Student ID: 1636402

Problem 2: Dynamic Programming

a)

Recursive Case:

$OPT(i, n) = n$ if $OPT(i + 1, n - 2) = n - 2$ and $S[i] == S[i + n - 1]$
Otherwise: $OPT(i, n) = max\,(OPT(i, n - 1), OPT(i + 1, n - 1))$

b)

Base Case:

$OPT(i, 0) = 0$ for all i
$OPT(i, 1) = 1$ for all i

c)

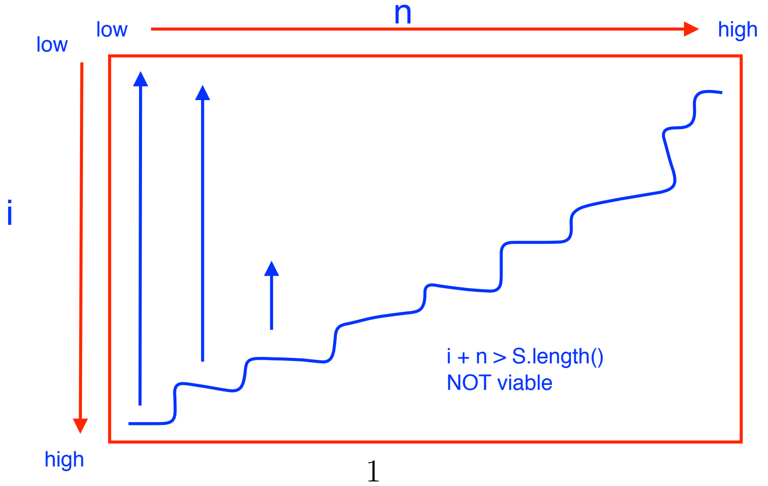
OPT(i, n) requires OPT(i + 1, n - 2) , OPT(i, n - 1), OPT(i+1, n - 1)
Each case of the recursive call either calls n values of $n - 2$ or $n - 1$, and thus this requires information from lower n values first.
Similiarly, each recursive call of OPT either calls i values of i or $i + 1$, and thus this requires information from higher i values first.

Thus $OPT(i, n)$ requires the subproblems with lower n values and higher/equal i values to be calculated first before it can be calculated.

d)

We should traverse these subproblems starting at high i values and working our way down, and low n values and working our way up. This setup is draw below, although the choice of traversing i first is arbitrary. Note that OPT(i,n) is limited in the i and n values it can take it: $i + n \leq S.length()$ at all times (otherwise our substring would run off the end of S.

Essentially, to solve OPT(i,n), we need to have already solved OPT for values to the left and down in the table



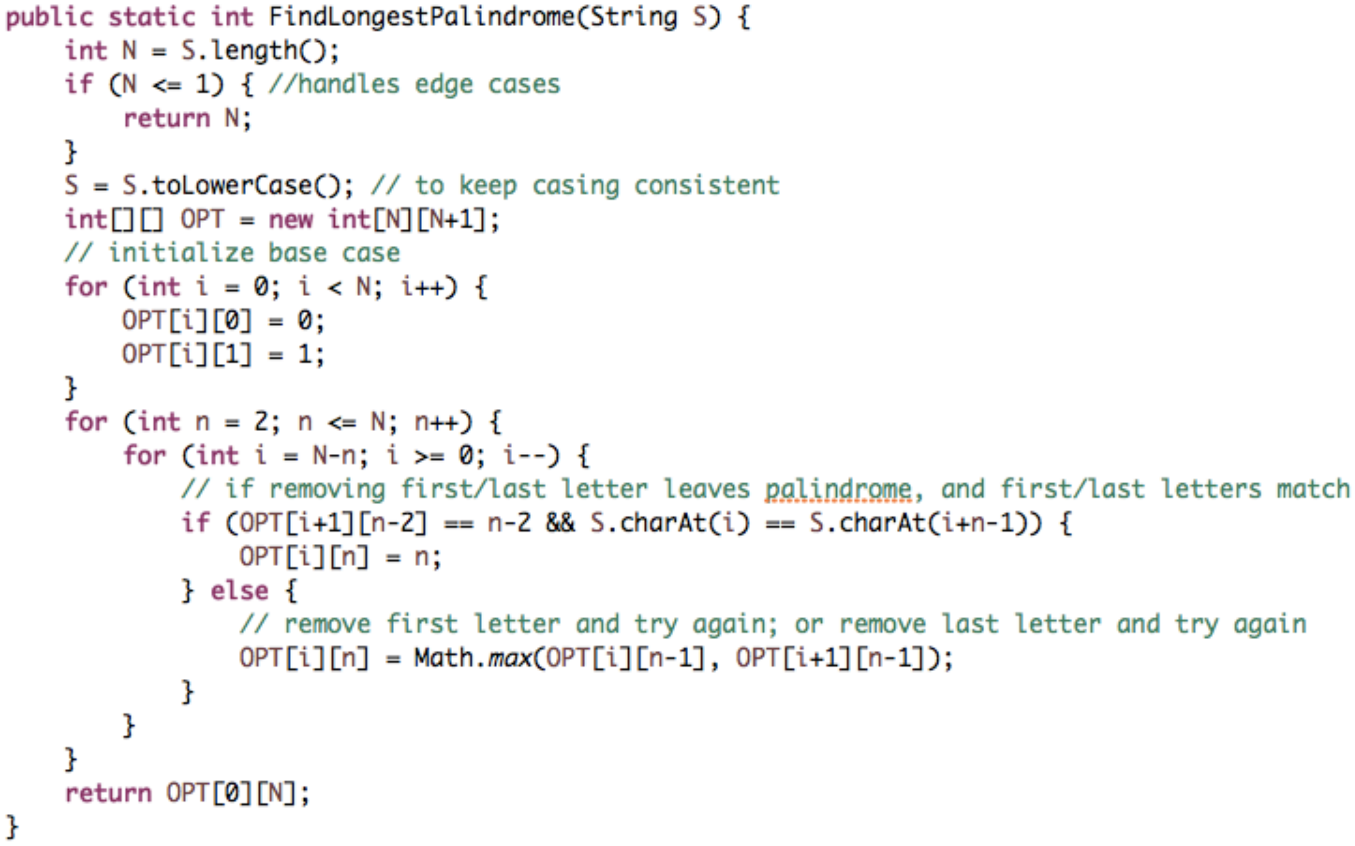
e)

```
public int findLongestPalindrome(String S)
N = S.length();
if N <= 1 if (N ¡= 1) // handles edge cases
    return N;

S = S.toLowerCase(); // to make casing consistent
int[][] OPT = new int[N][N+1];
OPT[$i$][0] = 0 for all $i$
OPT[$i$][1] = 1 for all $i$
for $n = 2 \rightarrow N$
    for $i = N - n \rightarrow 0$ // $N - n$ forces $i + n \leq N = S.length()$
        if OPT[i+1][ n-2] = n-2 and first/last characters of substring match
            OPT[i][n] = n;
        else
            OPT[i][n] = max(OPT[i][n-1], OPT[i+1][n-1]) // remove first letter or remove last letter
return OPT[0][N];
```

** For convenience, I have provided the Java code on the following page**

Note that this code relies on the fact that if OPT(i+1, n-2) = n-2, then the full substring is a palindrome. Thus L + substring + L is also a palindrome implying OPT(i, n) = n.



```java
public static int FindLongestPalindrome(String S) {
    int N = S.length();
    if (N <= 1) { //handles edge cases
        return N;
    }
    S = S.toLowerCase(); // to keep casing consistent
    int[][] OPT = new int[N][N+1];
    // initialize base case
    for (int i = 0; i < N; i++) {
        OPT[i][0] = 0;
        OPT[i][1] = 1;
    }
    for (int n = 2; n <= N; n++) {
        for (int i = N-n; i >= 0; i--) {
            // if removing first/last letter leaves palindrome, and first/last letters match
            if (OPT[i+1][n-2] == n-2 && S.charAt(i) == S.charAt(i+n-1)) {
                OPT[i][n] = n;
            } else {
                // remove first letter and try again; or remove last letter and try again
                OPT[i][n] = Math.max(OPT[i][n-1], OPT[i+1][n-1]);
            }
        }
    }
    return OPT[0][N];
}
```