Solutions: Sequence Alignment

created by Lisa Wang

a)

```
#include <iostream>
#include "console.h"
#include "gwindow.h"
#include "simpio.h"

using namespace std;

// Scorings:
int matchPts = 1;
int mismatchPnlt = 1;
int gapPnlt = 3;

/* maxOfThree
 * ----------
 * Helper function that returns the largest of three
 * integers
 */
int maxOfThree(int a, int b, int c) {
    int best = max(a, b);
    return max(best, c);
}

/* computeBestAlignScore
 * ----------------------
 * Recursive function, returns the score of the best
 * alignment of
 * the two strings seq1 and seq2.
 */
int computeBestAlignScore(string seq1, string seq2) {
    // base cases:
    if (seq1 == "") return seq2.length() * (-gapPnlt);
    if (seq2 == "") return seq1.length() * (-gapPnlt);

    int noGapScore = computeBestAlignScore(seq1.substr(1),
          seq2.substr(1));
    if (seq1[0] == seq2[0]) noGapScore+=matchPts;
    else noGapScore -= mismatchPnlt;
    int gap1Score = computeBestAlignScore(seq1,
          seq2.substr(1)) - gapPnlt;
    int gap2Score = computeBestAlignScore(seq1.substr(1),
seq2) - gapPnlt;
    return maxOfThree(noGapScore, gap1Score, gap2Score);
}

int main() {
    while (true) {
        string seq1 = getLine("First sequence: ");
        string seq2 = getLine("Second sequence: ");
        int score = computeBestAlignScore(seq1, seq2);
        cout << score << endl;
    }
    return 0;
}
```

c) There are three recursive calls on each level, so if we draw a tree of function calls, each node has three children. Since we are only taking off one character from one or both strings at a time, the depth of the tree is at least min(m,n) where m is the length of the first string and n the length of the second string. Hence, the lower bound of the complexity is exponential in the min length of the input strings.

d) An exponential runtime is really bad! You probably noticed that when you tried out longer strings. But there are faster ways of solving the sequence alignment problem and other recursive problems. One method is memoization, which means memorizing intermediate results in a table to avoid doing work we have already done before.
We could even go one step further and use dynamic programming. If you are interested in cool algorithms and the reasons why some are performing better than others, take CS161 in the future!