```cpp
#include <iostream>
#include<ostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include <vector>
#include <time.h>
#include <sstream>



typedef std::vector<int> SimilarityVector;
typedef std::vector<SimilarityVector> SimilarityMatrix;
typedef std::vector<std::string> SequenceVector;

timespec timespec_diff(timespec start, timespec end);
void printMatrix(const SimilarityMatrix &matrix);
void sequenceAlgo(SimilarityMatrix *similarityMatrix, SequenceVector *sequences);


/**
 * Main function.
 *
 */
int main(int argc, char *argv[])
{

    std::ifstream files[2];

    //files[0].open("C://Krishna/HPC-Files/Prog-Assign2/sequence.txt");
    files[0].open("C://Krishna/HPC-Files/Prog-Assign2/HIV-1_db.fasta");
    //files[1].open("C://Krishna/HPC-Files/Prog-Assign2/unknown.txt");
    files[1].open("C://Krishna/HPC-Files/Prog-Assign2/HIV-1_Polymerase.txt");
    if(!(files[0]&&files[1]))
    {
        std::cerr<<"Unable To load the file";
        exit(EXIT_FAILURE);
    }

    // read files
    SequenceVector sequences(2);

    std::string line;
    for(int i = 0; i < 2; ++i)
    {
        std::ifstream &file = files[i];
        std::string &sequence = sequences[i];
        while(getline(file, line))
        {
            if((line.size() > 0) && (line[line.size() - 1] == '\r'))
                line.resize(line.size() - 1);
            sequence += line;
        }
    }

    // check the sequence size is >= the sample size
    int rows = sequences[0].size() + 1;
    int cols = sequences[1].size() + 1;
    if(rows < cols)
    {
        exit(EXIT_FAILURE);
    }

    // create the matrix, setting all cells to -1
    SimilarityMatrix similarityMatrix(rows, SimilarityVector(cols, -1));


    // set the first row and and first column to 0
    for(int r = 0; r < rows; ++r)
        similarityMatrix[r][0] = 0;
    for(int c = 0; c < cols; ++c)
        similarityMatrix[0][c] = 0;

    // start timing
    struct timespec start, finish;
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);

    sequenceAlgo(&similarityMatrix, &sequences);




    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &finish);


        printMatrix(similarityMatrix);

std::cerr << "Total time (nanoseconds): " << timespec_diff(start, finish).tv_nsec << std::endl;

  return 0;
}



    void sequenceAlgo(SimilarityMatrix *similarityMatrix, SequenceVector *sequences)
    {
        int rows = similarityMatrix->size();
        int cols = (*similarityMatrix)[0].size();



    // Sequencing the Matrix
    for(int j=1 ; j< rows;j++)
        {


            for (int i = 1 ; i <=cols; i++)
            {

                int row = j;
                int col = i;




                int options[3];

                // Algorithm goes here
        options[0] = (*similarityMatrix)[row - 1][col - 1] + ((*sequences)[0][row - 1] == (*sequences)[1][col - 1] ? 1 : -1);
                options[1] = (*similarityMatrix)[row][col - 1] - 2;
                options[2] = (*similarityMatrix)[row - 1][col] - 2;

                int value = 0;
                for(int o = 0; o < 3; ++o)
                    if(options[o] > value)
                        value = options[o];


                (*similarityMatrix)[row][col] = value;
            }





        }
  }


void printMatrix(const SimilarityMatrix &matrix)
{
    for(SimilarityMatrix::const_iterator rowit = matrix.begin(); rowit != matrix.end(); ++rowit)
    {
        for(SimilarityVector::const_iterator colit = rowit->begin(); colit != rowit->end(); ++colit)
            std::cout << std::setw(3) << *colit << ' ';
        std::cout << '\n';
    }
    std::cout << std::flush;
}




timespec timespec_diff(timespec start, timespec end)
{
    timespec temp;
    if ((end.tv_nsec - start.tv_nsec) < 0)
    {
        temp.tv_sec = end.tv_sec - start.tv_sec-1;
    temp.tv_nsec = 1000000000 + end.tv_nsec - start.tv_nsec;
  }
    else
    {
    temp.tv_sec = end.tv_sec - start.tv_sec;
    temp.tv_nsec = end.tv_nsec - start.tv_nsec;
  }
  return temp;
}
```