Assignment 2 - COSC 3320

Alex Bennett (ID: 1901408)

Theory Problem 1

(a)

To check if an element in L_1 is in L_2 we must iterate over each element of L_1 (n elements), and then compare each element of L_1 with every element of L_2. Thus the lower bound is $O(n^2)$.

(b)

Theory Problem 2

Theory Problem 3

To determine the average number of scalar multiplications for a sequence of n matrices we will use the following informal algorithm:

$S[i, i] = 0$
$S[i, i+1] = p_i + p_{i+1} + p_{i+2}$
$S[i, j] = \text{avg}(S[i, k] + S[k+1, j] + p_i + p_{k+1} + p_{k+1}),\ \text{for } i \leq k \leq j-1$

Where $S[i, j]$ is the matrix representing the average work for a parentheses configuration grouping every element from i to j and p_i represents the ith dimension of the original matrix sequence. The average work is equal to the sum of scalar multiplications for possible k-values divided by the total number of k-values.

Since this algorithm will be iterated over 3 nested for-loops the time complexity is $O(n^3)$. Since S is two dimensional our space complexity is $O(n^2)$

Programming Problem 1

Procedure

In this program we use three two-dimensional arrays X, Y, and Z, each of size n, with X and Y initialized to 1 for each entry. We then add corresponding entries of X and Y together via matrix addition to generate an entry in Z. We use two different algorithms to perform the addition, the first traversing the arrays in row-major order and the second traversing them in column-major order, and compare their timings.

Hypothesis

Given that both algorithms have the same number of operations, one might expect that the timings for the two algorithms will be similar. However, as we've studied in class, if a multidimensional array is stored in memory using row-major order, then row-major traversal is significantly faster than column-major traversal. Given that C++ must store these arrays in either row- or column-major order, my hypothesis is that there will be significant difference between the two algorithms.

C++ Implementation

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

        char *arg = argv[1];
        int n = atoi(arg);

        int** X = new int*[n];
        int** Y = new int*[n];
        int** Z = new int*[n];

        //Initialize X, Y, Z
        for (int i = 0; i < n; i++) {
                X[i] = new int[n];
                Y[i] = new int[n];
```

```
                Z[i] = new int[n];
        }

        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        X[i][j] = 1;
                        Y[i][j] = 1;
                }
        }

        //Version 1
        clock_t begin1 = clock();
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        Z[i][j] = X[i][j] + Y[i][j];
                }
        }
        clock_t end1 = clock();

        //Version 2
        clock_t begin2 = clock();
        for (int j = 0; j < n; j++) {
                for (int i = 0; i < n; i++) {
                        Z[i][j] = X[i][j] + Y[i][j];
                }
        }
        clock_t end2 = clock();

        int time1 = (double)(end1 - begin1)/(CLOCKS_PER_SEC/1000);
        int time2 = (double)(end2 - begin2)/(CLOCKS_PER_SEC/1000);

        printf("Version 1: %d milliseconds\n", time1);
        printf("Version 2: %d milliseconds\n", time2);

}
```

Results

Dimension of Array	Time of Vers. 1 (ms)	Time of Vers. 2 (ms)
128	0	0
256	0	0
512	0	3
1024	5	22
2048	20	108
4096	81	545
8192	325	2609
16384	1421	15618
32768	11147	109160
65536	46973	824990

Explanation