Course: ENSF 694 – Summer 2024
Lab #: 2
Instructor: M. Moussavi
Student Name: Jaskirat Singh (Jazz)
Submission Date: 10 July 2024

Part I Exercise A Code:

```
/*
 *  my_lab2exe_A.cpp
 *  ENSF 694 Lab 2, part I, exercise A
 *  Completed by: Jaskirat Singh
 *  Submission date: July 10
 */

int my_strlen(const char *s);
/*  Duplicates strlen from <cstring>, except return type is int.
 *  REQUIRES
 *     s points to the beginning of a string.
 *  PROMISES
 *     Returns the number of chars in the string, not including the
 *     terminating null.
 */

void my_strncat(char *dest, const char *source, int numChars);
/*  Duplicates strncat from <cstring>, except return type is void.
 *  REQUIRES
 *     dest points to the beginning of the new destination string.
 *     source points to the beginning of the source string.
 *     numChars specifies how many characters need to be concatinated
 *  PROMISES
 *     Copies numChar number of characters from dest to source string.
 *     If the number of characters in source string are less than numChars,
 *     then all the characters from source are copied to dest.
 */

#include <iostream>
#include <cstring>
using namespace std;

int main(void)
{
    char str1[7] = "banana";
    const char str2[] = "-tacit";
    const char* str3 = "-toe";

    /* point 1 */
    char str5[] = "ticket";
    char my_string[100]="";
    int bytes;
    int length;

    /* using strlen libarary function */
    length = (int) my_strlen(my_string);
    cout << "\nLine 1: my_string length is " << length;

    /* using sizeof operator */
    bytes = sizeof (my_string);
    cout << "\nLine 2: my_string size is " << bytes << " bytes.";

    /* using strcpy libarary function */
    strcpy(my_string, str1);
    cout << "\nLine 3: my_string contains: " << my_string;

    length = (int) my_strlen(my_string);
    cout << "\nLine 4: my_string length is " << length << ".";

    my_string[0] = '\0';
    cout << "\nLine 5: my_string contains:\"" << my_string << "\"";

    length = (int) my_strlen(my_string);
    cout << "\nLine 6: my_string length is " <<  length << ".";

    bytes = sizeof (my_string);
    cout << "\nLine 7: my_string size is still " << bytes << " bytes.";

    /* strncat append the first 3 characters of str5 to the end of my_string */
```

```
    my_strncat(my_string, str5, 3);
    cout << "\nLine 8: my_string contains:\"" << my_string << "\"";

    length = (int) my_strlen(my_string);
    cout << "\nLine 9: my_string length is " << length << ".";

    my_strncat(my_string, str2,  4);
    cout << "\nLine 10: my_string contains:\"" << my_string << "\"";

    /* strncat append ONLY up ot '\0' character from str3 -- not 6 characters */
    my_strncat(my_string, str3, 6);
    cout << "\nLine 11: my_string contains:\"" << my_string << "\"";

    length = (int) my_strlen(my_string);
    cout << "\nLine 12; my_string has " << length << " characters.";

    cout << "\n\nUsing strcmp - C library function: ";

    cout << "\n\"ABCD\" is less than \"ABCDE\" ... strcmp returns: " <<
    strcmp("ABCD", "ABCDE");

    cout << "\n\"ABCD\" is less than \"ABND\" ... strcmp returns: " <<
    strcmp("ABCD", "ABND");

    cout << "\n\"ABCD\" is equal than \"ABCD\" ... strcmp returns: " <<
    strcmp("ABCD", "ABCD");

    cout << "\n\"ABCD\" is less than \"ABCd\" ... strcmp returns: " <<
    strcmp("ABCD", "ABCd");

    cout << "\n\"Orange\" is greater than \"Apple\" ... strcmp returns: " <<
    strcmp("Orange", "Apple") << endl;
    return 0;
}

int my_strlen(const char *s) {
    int length = 0;
    //Increment length and pointer value until '\0'
    while(*s != '\0') {
        length ++;
        s ++;
    }
    return length;
}

void my_strncat(char *dest, const char *source, int numChars) {
    //Increment dest until the end of the string is reached.
    while(*dest != '\0') { dest++; }

    //Loop until numChars characters are copied or the end of source is reached
    while(numChars && *source != '\0') {
        //Set source char to dest char
        *dest = *source;
        //cout << "Source is " << *source << " New dest is " << *dest << endl;
        source ++;
        dest ++;
        //Lower numChars until it reaches 0
        numChars --;
    }
    //Make dest a valid C-string by appending '\0' at the end
    *dest = '\0';
}
```

Part I Exercise A Output:

```
Line 1: my_string length is 0
Line 2: my_string size is 100 bytes.
Line 3: my_string contains: banana
Line 4: my_string length is 6.
Line 5: my_string contains:""
Line 6: my_string length is 0.
Line 7: my_string size is still 100 bytes.
Line 8: my_string contains:"tic"
Line 9: my_string length is 3.
Line 10: my_string contains:"tic-tac"
Line 11: my_string contains:"tic-tac-toe"
Line 12; my_string has 11 characters.

Using strcmp - C library function:
"ABCD" is less than "ABCDE" ... strcmp returns: -1
"ABCD" is less than "ABND" ... strcmp returns: -1
"ABCD" is equal than "ABCD" ... strcmp returns: 0
"ABCD" is less than "ABCd" ... strcmp returns: -1
"Orange" is greater than "Apple" ... strcmp returns: 1
Program ended with exit code: 0
```

Part I Exercise B Code:

```
/*
 *  lab2exe_B.cpp
 *  ENSF 694 Lab 2, exercise B
 *  Completed by: Jaskirat Singh
 *  Submission date: July 10
 */

#include <iostream>
#include <assert.h>
using namespace std;

int sum_of_array(const int *a, int n);
// REQUIRES
//   n > 0, and elements a[0] ... a[n-1] exist.
// PROMISES:
//  Return value is a[0] + a[1] + ... + a[n-1].

int main()
{
    int a[] = { 100 };
    int b[] = { 100, 200, 300, 400 };
    int c[] = { -100, -200, -200, -300 };
    int d[] = { 10, 20, 30, 40, 50, 60, 70 };


    int sum = sum_of_array(a, 1);
    cout << "sum of integers in array a is: " << sum << endl;

    sum = sum_of_array(b, 4);
    cout << "sum of integers in array b is: " << sum << endl;

    sum = sum_of_array(c, 4);
    cout << "sum of integers in array c is: " << sum << endl;

    sum = sum_of_array(d, 7);
    cout << "sum of integers in array d is: " << sum << endl;

    return 0;
}

int sum_of_array(const int *a, int n)
{
    //Base case
    if(n == 1) { return *a; }
    //Return a plus the next value of a
    return *a + sum_of_array(a + 1, n - 1);
}
```

Part I Exercise B Output:

```
sum of integers in array a is: 100
sum of integers in array b is: 1000
sum of integers in array c is: -800
sum of integers in array d is: 280
Program ended with exit code: 0
```

Part I Exercise D Code:

```
/*
 *  lab2exe_D.cpp
 *  ENSF 694 Lab 2, exercise D
 *  Completed by: Jaskirat Singh
 *  Submission date: July 10
 */

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <iostream>
#include <iomanip>
using namespace std;
#define N 2


void myPlot(int* x, double *y1, double *y2, int size){
    //Function built with the help of "Gnuplot example.cpp" pprovided on D2L
    FILE * gnuplotPipe = popen ("/opt/homebrew/bin/gnuplot -persistent", "w");

    //Name for the plot
    const char* name="Fibonacci Iterative v.s. Recursive Approaches";

    //Setting title, axis names, grid and legend style
    fprintf(gnuplotPipe, "set title '%s'\n", name);
    fprintf(gnuplotPipe, "set xlabel 'N'\n");
    fprintf(gnuplotPipe, "set ylabel 'Time (seconds)'\n");
    fprintf(gnuplotPipe, "set grid\n");
    fprintf(gnuplotPipe, "set key inside left\n");
    fprintf(gnuplotPipe, "set style line 1 lt 1 lw 2 pt 7 ps 1.5 lc rgb 'blue'\n");
    fprintf(gnuplotPipe, "set style line 2 lt 1 lw 2 pt 7 ps 1.5 lc rgb 'red'\n");

    //Plotting the data
    fprintf(gnuplotPipe, "plot '-' pt 7 ps 1 lc 'blue' title 'Iterative', '-' pt 7 ps
1 lc 'red' title 'Recursive'");

    for (int i=0; i < size; i++)
        fprintf(gnuplotPipe, "%d %f\n", x[i], y1[i]);

    fprintf(gnuplotPipe, "e\n");

    for (int i=0; i < size; i++)
        fprintf(gnuplotPipe, "%d %f\n", x[i], y2[i]);

    fprintf(gnuplotPipe, "e\n");

    pclose(gnuplotPipe);
}


// Function to multiply two matrices of size N x N
void multiplyMatrix(int a[N][N], int b[N][N], int result[N][N]) {
    // Three loops for dot products
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            result[i][j] = 0;
            for(int k = 0; k < N; k++) {
                result[i][j] += a[i][k] * b[k][j];
```

```
            }
        }
    }
}

// Recursive function
void powerMatrix(int base[N][N], int exp, int result[N][N]) {
    // Base case exp = 1
    if(exp == 1){
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                result[i][j] = base[i][j];
            }
        }
        return;
    }

    int bufferMatrix[N][N]; // Temporary matrix for intermediate calculations

    // If exponent is even
    if(exp % 2 == 0){
        powerMatrix(base, exp / 2, bufferMatrix);
        multiplyMatrix(bufferMatrix, bufferMatrix, result);
    }
    // If exponent is odd
    else{
        powerMatrix(base, exp / 2, bufferMatrix);
        multiplyMatrix(bufferMatrix, bufferMatrix, result);
        multiplyMatrix(result, base, result);
    }
}

// Function to calculate the nth Fibonacci number using recursive matrix
exponentiation
int fibonacciRecursive(int n) {
    if (n == 0) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }

    int base[N][N] = {{1, 1}, {1, 0}};
    int result[N][N];
    powerMatrix(base, n - 1, result);
    return result[0][0];
}

// Function to calculate the nth Fibonacci number iteratively
int fibonacciIterative(int n) {
    // This function must be completed by the students and if necessary its return
value to be corrected.
    //Base case for n = 0
    if (n <= 0) {
        return 0;
    }
    //Base case for n = 1
    if (n == 1) {
        return 1;
    }

    int last = 0, current = 1;
```

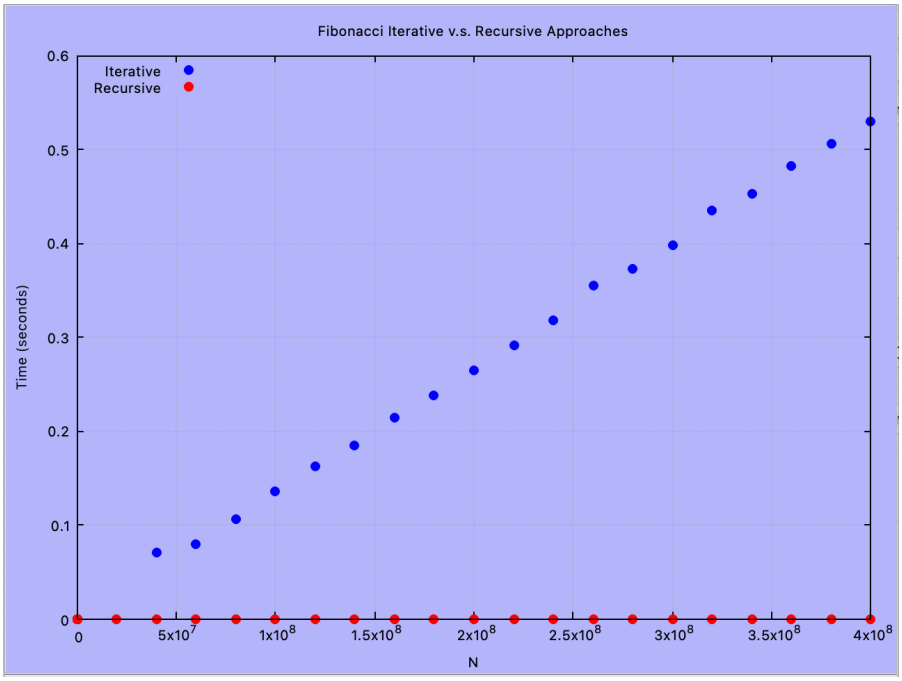
```
    for (int i = 2; i <= n; i++){
        int next = last + current;
        last = current;
        current = next;
    }

    return current;
}

// Function to measure the time taken by a function to calculate the nth Fibonacci
number
// This function is using a pointer to a funciton called fibonacciFunc
double measureTime(int (*fibonacciFunc)(int), int n) {
    // This funcitn must be completed by the students and if necessary its return
value to be corrected.
    clock_t start, end;

    start = clock();
    fibonacciFunc(n);
    end = clock();
    return (double)(end - start) / CLOCKS_PER_SEC;
}

int main(void) {
    const int maxN = 400000000; // Adjust maxN based on the range you want to test
    double recursive_result[50];
    double iterative_result[50];
    int N_value[50];

    cout << "Recursive Matrix Exponentiation Method\n";
    cout << setw(12) << "N" << setw(12) << "Time\n";
    for (int n = 20000000, i=0; n <= maxN; n+=20000000, i++) {
        double time = measureTime(fibonacciRecursive, n);
        recursive_result[i] = time;
        cout << setw(12) << n << setw(12) << recursive_result[i] << endl;
    }

    cout << "\nIterative Method\n";
    cout << setw(12) << "N" << setw(12) << "Time\n";
    for (int n = 20000000, i=0; n <= maxN; n+=20000000, i++) {
        double time = measureTime(fibonacciIterative, n);
        iterative_result[i] = time;
        cout << setw(12) << n << setw(12) << iterative_result[i] << endl;
        N_value[i] = n;
    }

    myPlot(N_value, iterative_result,  recursive_result, 30 );

    return 0;
}
```

Part I Exercise D Program Output:

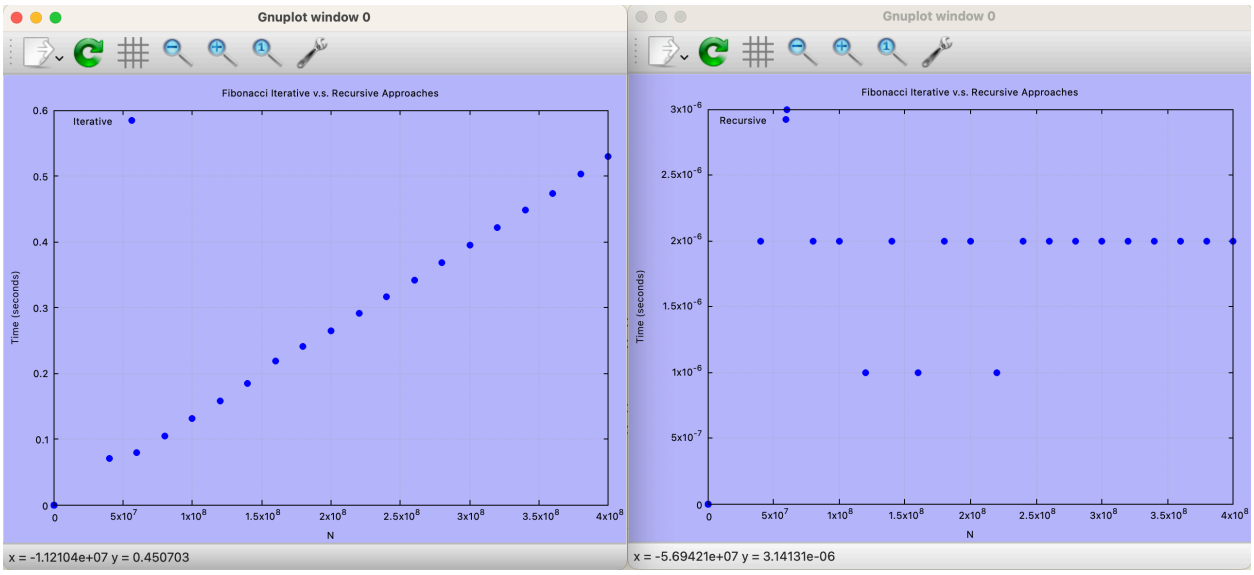
```
Recursive Matrix Exponentiation Method
           N       Time
    20000000       4e-06
    40000000       1e-06
    60000000       1e-06
    80000000       1e-06
   100000000       2e-06
   120000000       1e-06
   140000000       2e-06
   160000000       2e-06
   180000000       1e-06
   200000000       3e-06
   220000000       2e-06
   240000000       1e-06
   260000000       2e-06
   280000000       2e-06
   300000000       1e-06
   320000000       2e-06
   340000000       1e-06
   360000000       2e-06
   380000000       2e-06
   400000000       2e-06

Iterative Method
           N       Time
    20000000    0.028581
    40000000    0.070672
    60000000    0.079752
    80000000    0.106371
   100000000     0.13583
   120000000    0.163147
   140000000    0.185179
   160000000     0.21523
   180000000     0.23896
   200000000    0.265279
   220000000    0.291372
   240000000    0.318723
   260000000    0.355904
   280000000    0.373117
   300000000    0.398055
   320000000    0.435435
   340000000    0.453296
   360000000    0.482658
   380000000    0.506848
   400000000     0.52984
```

Part I Exercise D GNU Plot:

Combined Plot:



Separated Plots:



Part I Exercise E Code:

```
/*
 *  lab2exe_E.cpp
 *  ENSF 694 Lab 2, exercise E
 *  Completed by: Jaskirat Singh
 *  Submission date: July 10
 */

#include "compare_sorts.h"
using namespace std;

void to_lower(char *str) {
    while (*str) {
        *str = std::tolower(*str);
        ++str;
    }
}

void strip_punctuation(char *word) {
    int i = 0, j = 0;
    //Loop till end of C string
    while (word[i] != '\0') {
        if (isalnum(word[i]) || word[i] == '-') {
            word[j++] = word[i];
        }
        i++;
    }
    //End string to make into proper C-string
    word[j] = '\0';
}

bool is_unique(char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE], int num_words, const char
*word) {
    //Loop through all the words
    for(int i = 0; i < num_words; i++) {
        // Word not unique
        if(strcmp(words[i], word) == 0) {
            return false;
        }
    }
    // Word is unique
    return true;
}

void quicksort(int *indices, char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE], int left,
int right) {
    //Following algorithm shown in class
    if(left < right) {
        int index = indices[right];
        char *pivot = words[index];
        int i = left - 1;

        for(int j = left; j < right; j++) {
            if(strcmp(words[indices[j]], pivot) < 0) {
                i++;
                //Swap
                int temp = indices[i];
                indices[i] = indices[j];
                indices[j] = temp;
            }
        }
```

```
        //Swap
        int temp = indices[i + 1];
        indices[i + 1] = indices[right];
        indices[right] = temp;

        int part = i + 1;
        quicksort(indices, words, left, part - 1);
        quicksort(indices, words, part + 1, right);
    }
}

void shellsort(int *indices, char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE], int size) {
    //Reduce gap size after each iteration
    for(int gap = size/2; gap > 0; gap/=2) {
        //Iteratre throuugh
        for (int i = gap; i < size; i++) {
            int temp = indices[i];
            char *tempw = words[temp];

            int j;
            for(j = i; j >= gap && strcmp(words[indices[j - gap]], tempw) > 0; j -=
gap) {
                indices[j] = indices[j - gap];
            }

            indices[j] = temp;
        }
    }
}

void bubblesort(int *indices, char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE], int size) {
    //Following algorithm shown in class
    bool isSwapped;
    for(int i = 0; i < size - 1; i++) {
        isSwapped = false;
        for(int j = 0; j < size - i - 1; j++) {
            if(strcmp(words[indices[j]], words[indices[j + 1]]) > 0) {
                //swap
                int temp = indices[j];
                indices[j] = indices[j + 1];
                indices[j + 1] = temp;
                //swap performed
                isSwapped = true;
            }
        }
        //sorting complete when swapping is done
        if(!isSwapped) {
            break;
        }
    }
}

void read_words(const char *input_file, char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE],
int &num_words)
{
    std::ifstream infile(input_file);
    if (!infile) {
        std::cerr << "Error opening input file.\n";
        exit(1);
    }

    char word[MAX_WORD_SIZE + 1];
```

```
    num_words = 0;

    while (infile >> word) {
        strip_punctuation(word);
        to_lower(word);
        if (word[0] != '\0' && num_words < MAX_UNIQUE_WORDS && is_unique(words,
num_words, word)) {
            std::strncpy(words[num_words++], word, MAX_WORD_SIZE);
        }
    }
    infile.close();
}

void write_words(const char *output_file, char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE],
int *indices, int num_words)
{
    std::ofstream outfile(output_file);
    if (!outfile) {
        std::cerr << "Error opening output file.\n";
        exit(1);
    }

    for (int i = 0; i < num_words; ++i) {
        outfile << words[indices[i]] << '\n';
    }

    outfile.close();
}

void sort_and_measure_quicksort(char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE], int*
indices, int num_words, void (*sort_func)(int *, char [MAX_UNIQUE_WORDS]
[MAX_WORD_SIZE], int, int), const char *sort_name) {
    //Get start time
    auto start = chrono::high_resolution_clock::now();

    //Call sorting func
    sort_func(indices, words, 0, num_words - 1);

    //Get end time
    auto end = chrono::high_resolution_clock::now();

    //Calculate the duration
    chrono::duration<double> diff = end - start;

    //Print details
    cout << "Sorting with " << sort_name << " completed in " << fixed <<
setprecision(6) << diff.count() << " seconds." << endl;
}

void sort_and_measure_shell_bubble(char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE], int*
indices, int num_words, void (*sort_func)(int *, char [MAX_UNIQUE_WORDS]
[MAX_WORD_SIZE], int), const char *sort_name) {
    //Get start time
    auto start = chrono::high_resolution_clock::now();

    //Call sorting func
    sort_func(indices, words, num_words);

    //Get end time
    auto end = chrono::high_resolution_clock::now();

    //Calculate the duration
```

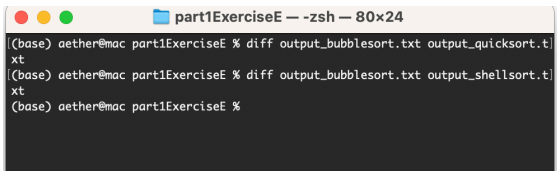
```
    chrono::duration<double> diff = end - start;

    //Print details
    cout << "Sorting with " << sort_name << " completed in " << fixed <<
setprecision(6) << diff.count() << " seconds." << endl;
}

int main() {
    const char *input_file = "/Users/aether/Documents/ENSF-694/ENSF-694Lab2/
part1ExerciseE/input.txt"; // Change this to your input file
    char words[MAX_UNIQUE_WORDS][MAX_WORD_SIZE];
    int num_words;

    read_words(input_file, words, num_words);

    int indices[num_words];
    for (int i = 0; i < num_words; ++i) {
        indices[i] = i;
    }

    sort_and_measure_quicksort(words,indices, num_words, quicksort, "Quick Sort");
    write_words("/Users/aether/Documents/ENSF-694/ENSF-694Lab2/part1ExerciseE/
output_quicksort.txt", words, indices, num_words);
    sort_and_measure_shell_bubble(words,indices, num_words, shellsort, "Shell Sort");
    write_words("//Users/aether/Documents/ENSF-694/ENSF-694Lab2/part1ExerciseE/
output_shellsort.txt", words, indices, num_words);
    sort_and_measure_shell_bubble(words, indices, num_words, bubblesort, "Bubble
Sort");
    write_words("/Users/aether/Documents/ENSF-694/ENSF-694Lab2/part1ExerciseE/
output_bubblesort.txt", words, indices, num_words);
    return 0;
}
```

Part I Exercise E Program Output:

```
Sorting with Quick Sort completed in 0.000021 seconds.
Sorting with Shell Sort completed in 0.000011 seconds.
Sorting with Bubble Sort completed in 0.000003 seconds.
Program ended with exit code: 0
```

Time variations due to IDE used. Xcode does certain optimizations so these times can be unreliable. I obtained different times for different IDEs used.

Part I Exercise E File Difference Check:



Part II Exercise A:

Ordered slowest to fastest growth rate:

37: Constant growth, unchanged with N.

$\frac{2}{N}$: Decreasing growth, becomes smaller with N. Since you can't have negative memory, this growth rate is still faster than constant growth.

$\sqrt{N}$: Slow growth. Less than linear.

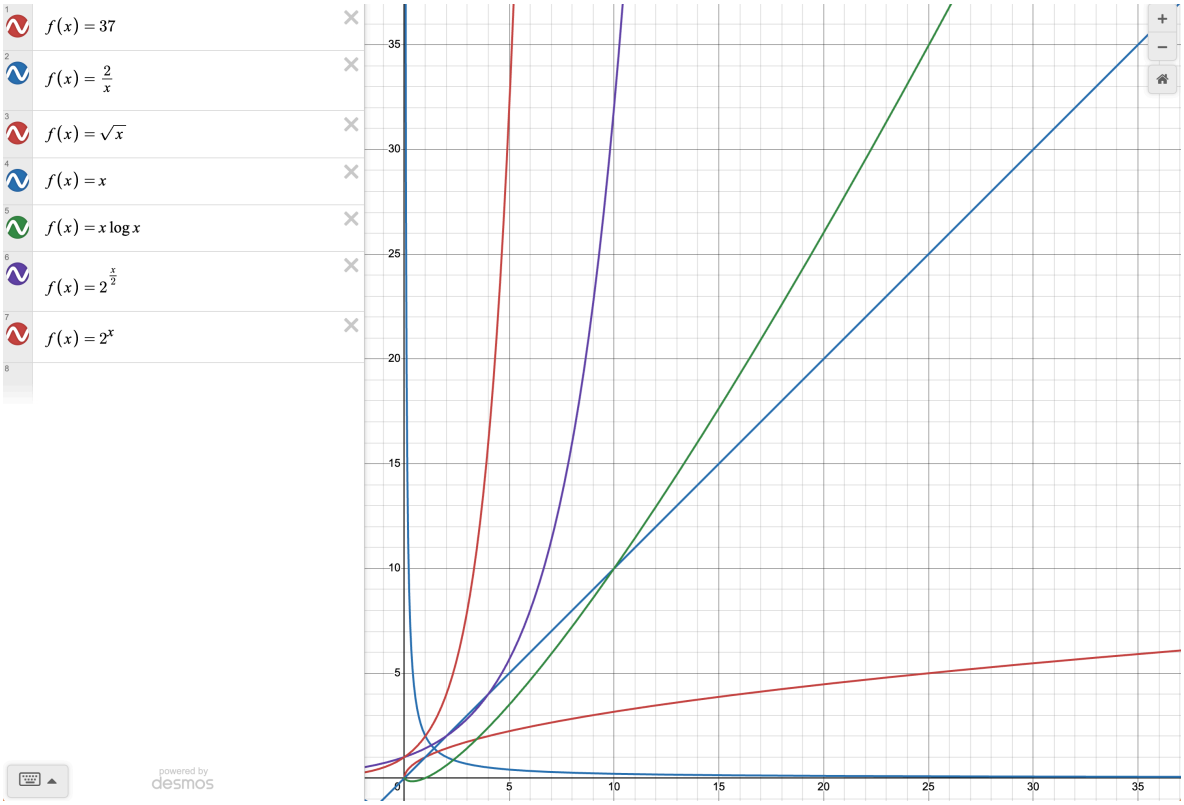
N: Linear growth rate, constant rate.

$NlogN$: logN increases growth rate of N.

$2^{\frac{N}{2}}$: Exponential growth. The division by 2 slows the growth slightly compared to no division.

2^N: Fastest exponential growth.

The functions can be plotted in a graphic calculator to visualize and confirm this assessment.



Part II Exercise B:

(1)
```
sum = 0;
for( i = 0; i < n; ++i )
     ++sum;
```

O(n). N iterations of the loop. Constant C for the code in the loop, which can be ignored.

(2):
```
sum = 0;
for( i = 0; i < n; ++i )
     for( j = 0; j < n; ++j )
          ++sum;
```

O(n²). N iterations of N iterations of the loop. Nested loops cause exponential growth. Constant C for the code in the loop can be ignored.

(3):
```
sum = 0;
for( i = 0; i < n; ++i )
     for( j = 0; j < n * n; ++j )
          ++sum;
```

O(n³). N iterations of N iterations of the loop that runs N * N times. Ignoring the constant terms, we get N * N * N.

(4):
```
sum = 0;
for( i = 0; i < n; ++i )
     for( j = 0; j < i; ++j )
          ++sum;
```

O(n²). n iterations of i iterations of the loop. Ignoring constants, nested loops causes arithmetic growth.

$\frac{n(n-1)}{2} = \frac{n^2-n}{2}$ As n approaches infinity, the n^2 term is the leading term.

(5):
```
sum = 0;
for( i = 0; i < n; ++i )
     for( j = 0; j < i; ++j )
          for( k = 0; k < j; ++k )
               ++sum;
```

O(n³). n iterations of i iterations k iterations of the loop. Using arithmetic series formulae we get:

$\frac{n*(n-1)*(n-2)}{6} = \frac{n^3-3n^2+2n}{6}$ As n approaches infinity, n^3 is the leading term.

(6):

```
sum = 0;
for( i = 0; i < n; ++i )
     for( j = 0; j < n; ++j )
          for( k = 0; k < n; ++k )
               ++sum;
```

$O(n^3)$. N iterations of N iterations of N iterations. Constants can be ignored.