Course: ENSF694 – Summer 2025

Lab #: Lab 2

Instructor: Mahmood Moussavi

Student Name: John Zhou

Submission Date: July 18, 2025

# Part I Exercise A

/\*

 \*  my\_lab2exe\_A.cpp

 \*  ENSF 694 Lab 2 Exercise A

 \* Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

int my\_strlen(const char \*s);

/\*  Duplicates my\_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 n);

/\*  Duplicates my\_strncat from <cstring>, except return type is void.

 \*  REQUIRES

 \*     dest points to the beginning of a string

 \*     source points to the beginning of a string

 \*     n integer that define the length of the string added to the destination

 \*  PROMISES

 \*     Appends at most n characters from source to the end of 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 my\_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.";

    /\* my\_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 << "\"";

    /\* my\_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 count=0;

    while (\*s){

        count++;

        s++;

    }

    return count;

}

void my\_strncat(char \*dest, const char \*source, int n) {

    while (\*dest != '\0') {

        dest++;

    }

    int i = 0;

    while (i < n && \*source != '\0') {

        \*dest = \*source;

        dest++;

        source++;

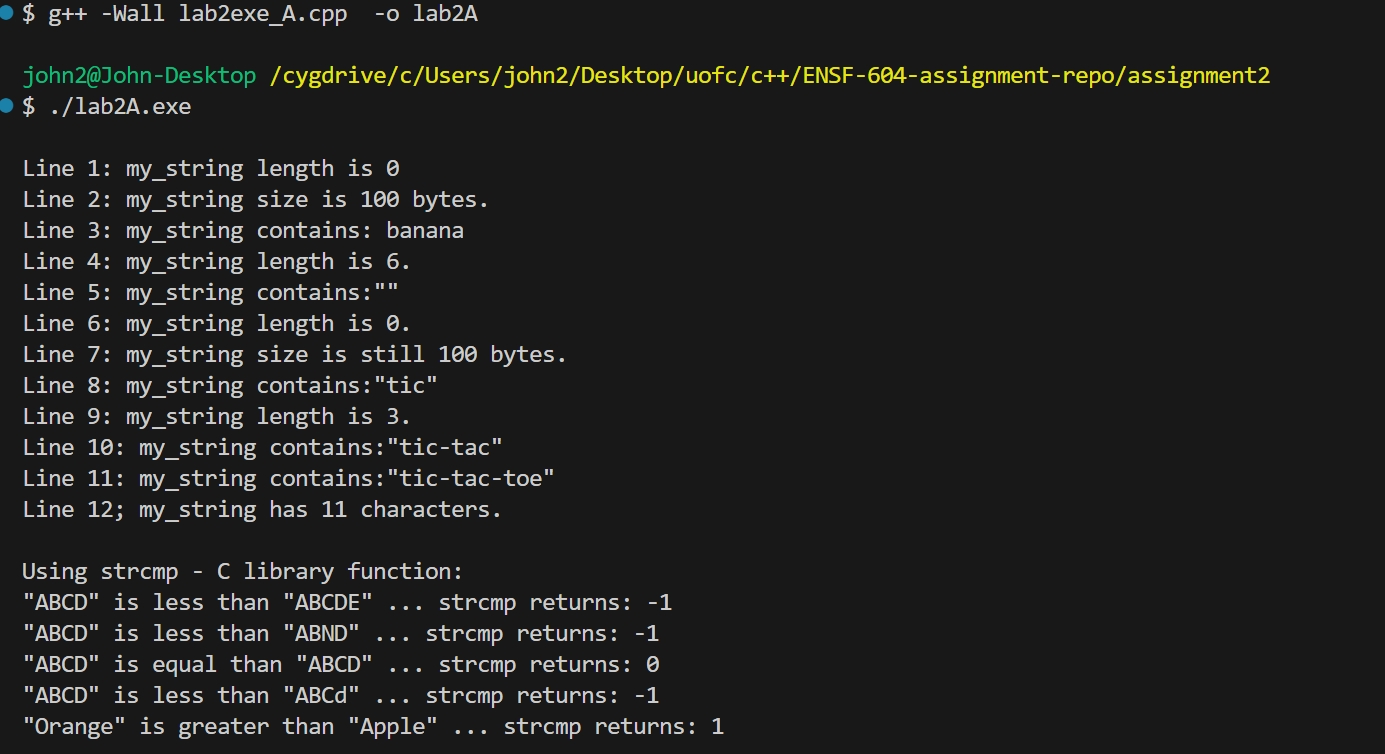
        i++;

    }

    \*dest = '\0';

}

Execution result



Exercise B

/\*

\* my\_lab1exe\_B.cpp

\* ENSF 694 Lab 2 Exercise B

\* Created by Mahmood Moussavi

\* Completed by: John Zhou

\*/

#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)

{

// int sum = 0;

// for(int i=0; i < n; i++)

// sum += a[i];

// return sum;

if (n == 0)

{

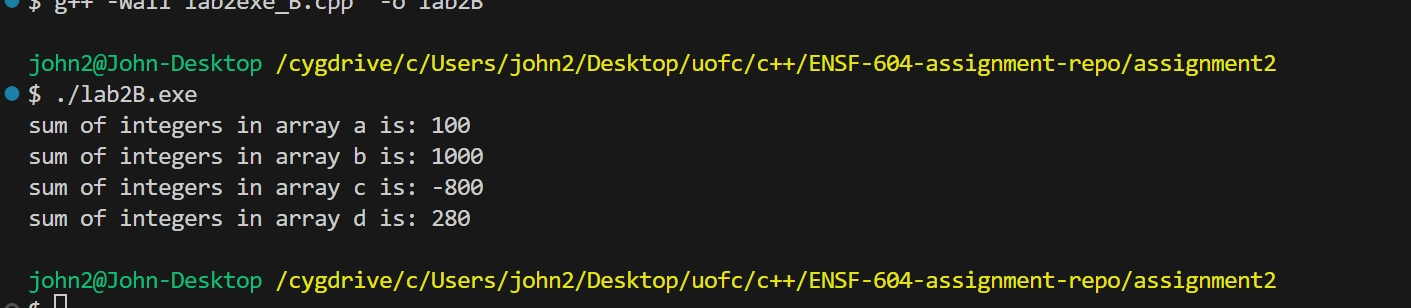
return 0;

}

return (a[0] + sum\_of\_array(a + 1, n - 1));

}

Execution result



Exercise C

/\*

 \*  lab1exe\_C.cpp

 \*  ENSF 694 Lab 1, exercise C

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#include <iostream>

using namespace std;

void time\_convert(int ms\_time, int \*minutes\_ptr, double \*seconds\_ptr);

/\*

 \* Converts time in milliseconds to time in minutes and seconds.

 \* For example, converts 123400 ms to 2 minutes and 3.4 seconds.

 \* REQUIRES:

 \*    ms\_time >= 0.

 \*    minutes\_ptr and seconds\_ptr point to variables.

 \* PROMISES:

 \*    0 <= \*seconds\_ptr & \*seconds\_ptr < 60.0

 \*    \*minutes\_ptr minutes + \*seconds\_ptr seconds is equivalent to

 \*    ms\_time ms.

 \*/

int main(void)

{

  int millisec;

  int minutes;

  double seconds;

  cout << "Enter a time interval as an integer number of milliseconds: ";

 // printf("Enter a time interval as an integer number of milliseconds: ");

  cin >> millisec;

  if (!cin) {

    cout << "Unable to convert your input to an int.\n";

    exit(1);

  }

  cout << "Doing conversion for input of " <<  millisec <<" milliseconds ... \n";

  /\* MAKE A CALL TO time\_convert HERE. \*/

  time\_convert(millisec,&minutes,&seconds);

  cout << "That is equivalent to " << minutes << " minute(s) and " << seconds << " second(s).\n";

  return 0;

}

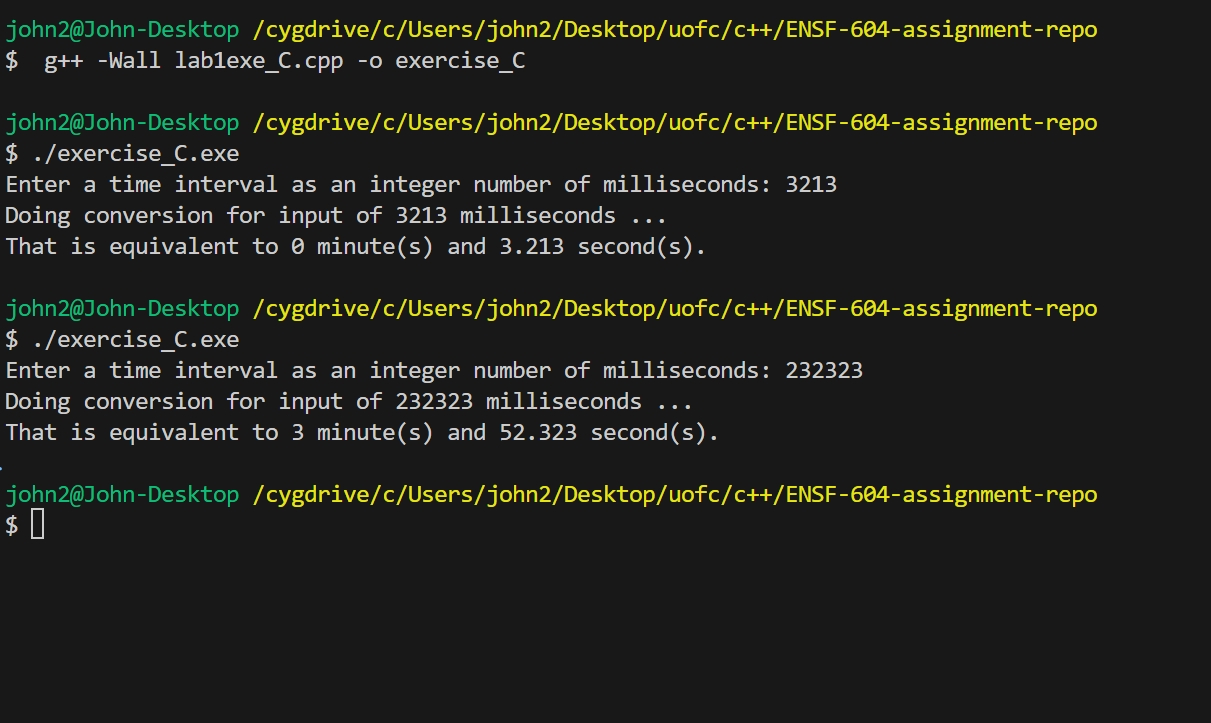
/\* PUT YOUR FUNCTION DEFINITION FOR time\_convert HERE. \*/

void time\_convert(int ms\_time, int \*minutes\_ptr, double \*seconds\_ptr){

  \*minutes\_ptr=ms\_time/60000;

  \*seconds\_ptr=ms\_time%60000/1000.0;

}

Execution result  


Exercise D

/\*

\* fibonacci.cpp

\* ENSF 694 Lab 2 Exercise D

\* Created by Mahmood Moussavi

\* Completed by: John Zhou

\*/

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <iostream>

#include <iomanip>

#include <chrono>

using namespace std;

#define N 2

void myPlot(int \*x, double \*y1, double \*y2, int size)

{

FILE \*gnuplotPipe = popen("gnuplot -persist", "w");

if (gnuplotPipe == NULL)

{

printf("Error: Could not open pipe to Gnuplot.\n");

return;

}

fprintf(gnuplotPipe, "set title 'Fibonacci Complexity Comparison'\n");

fprintf(gnuplotPipe, "set xlabel 'N (Input Size)'\n");

fprintf(gnuplotPipe, "set ylabel 'Execution Time (Seconds)'\n");

fprintf(gnuplotPipe, "set key outside\n");

fprintf(gnuplotPipe, "set grid\n");

fprintf(gnuplotPipe, "set terminal x11\n");

fprintf(gnuplotPipe, "set yrange [0:0.3]\n");

fprintf(gnuplotPipe, "set xtics rotate by -45\n");

fprintf(gnuplotPipe, "plot '-' using 1:2 with points pt 7 ps 1.5 lc rgb 'blue' title 'Iterative Method', '-' using 1:2 with points pt 7 ps 1.5 lc rgb 'red' title 'Matrix Exponentiation Method'\n");

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");

fclose(gnuplotPipe);

}

void myPlot\_for\_recursive\_method(int \*x, double \*y1, int size)

{

FILE \*gnuplotPipe = popen("gnuplot -persist", "w");

if (gnuplotPipe == NULL)

{

printf("Error: Could not open pipe to Gnuplot.\n");

return;

}

fprintf(gnuplotPipe, "set title 'Fibonacci Recursive Method'\n");

fprintf(gnuplotPipe, "set xlabel 'N (Input Size)'\n");

fprintf(gnuplotPipe, "set ylabel 'Execution Time (Seconds)'\n");

fprintf(gnuplotPipe, "set key outside\n");

fprintf(gnuplotPipe, "set grid\n");

fprintf(gnuplotPipe, "set terminal x11\n");

fprintf(gnuplotPipe, "set yrange [0:0.00001]\n");

fprintf(gnuplotPipe, "set xtics rotate by -45\n");

// Only one dataset plotted

fprintf(gnuplotPipe, "plot '-' using 1:2 with points pt 7 ps 1.5 lc rgb 'red' title 'Recursive Method'\n");

for (int i = 0; i < size; i++)

{

fprintf(gnuplotPipe, "%d %f\n", x[i], y1[i]);

}

fprintf(gnuplotPipe, "e\n");

fclose(gnuplotPipe);

}

// Function to multiply two matrices of size N x N

void multiplyMatrices(int A[N][N], int B[N][N], int result[N][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

result[i][j] = 0;

}

}

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

for (int k = 0; k < N; k++)

{

result[i][j] += A[i][k] \* B[k][j];

}

}

}

}

// Recursive funciont

void powerMatrix(int base[N][N], int exp, int result[N][N])

{

if (exp == 0)

{

result[0][0] = 1;

result[0][1] = 0;

result[1][0] = 0;

result[1][1] = 1;

return;

}

int temp[N][N];

powerMatrix(base, exp / 2, temp);

multiplyMatrices(temp, temp, result);

if (exp % 2 == 1)

{

multiplyMatrices(result, base, temp);

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

result[i][j] = temp[i][j];

}

}

}

}

// 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)

{

int prev = 0;

int cur = 1;

if (n == 0)

{

return 0;

}

if (n == 1)

{

return 1;

}

for (int i = 2; i < n; i++)

{

int temp = prev + cur;

prev = cur;

cur = temp;

}

return cur;

}

// 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)

{

using namespace std::chrono;

auto start = high\_resolution\_clock::now();

fibonacciFunc(n);

auto end = high\_resolution\_clock::now();

duration<double> time\_taken = end - start;

return time\_taken.count(); // returns time in seconds

}

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;

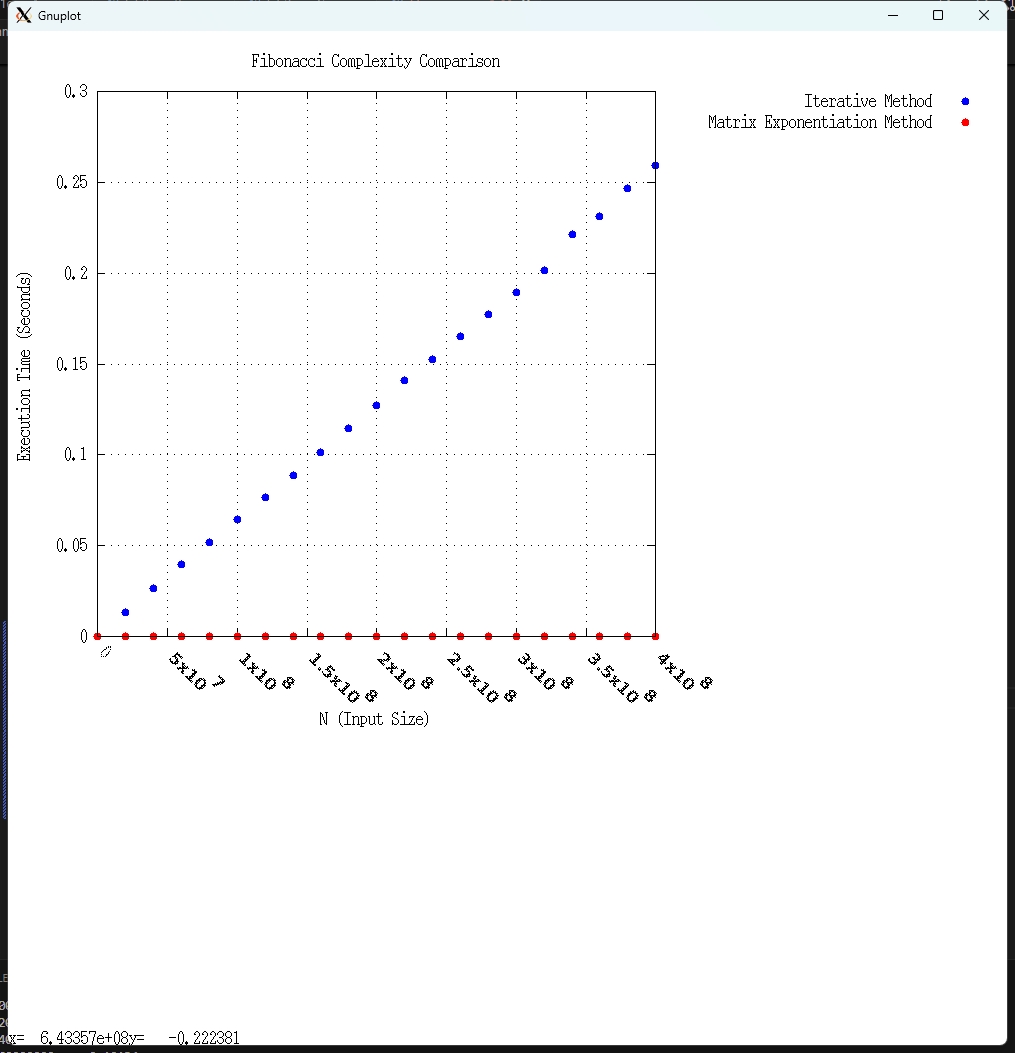
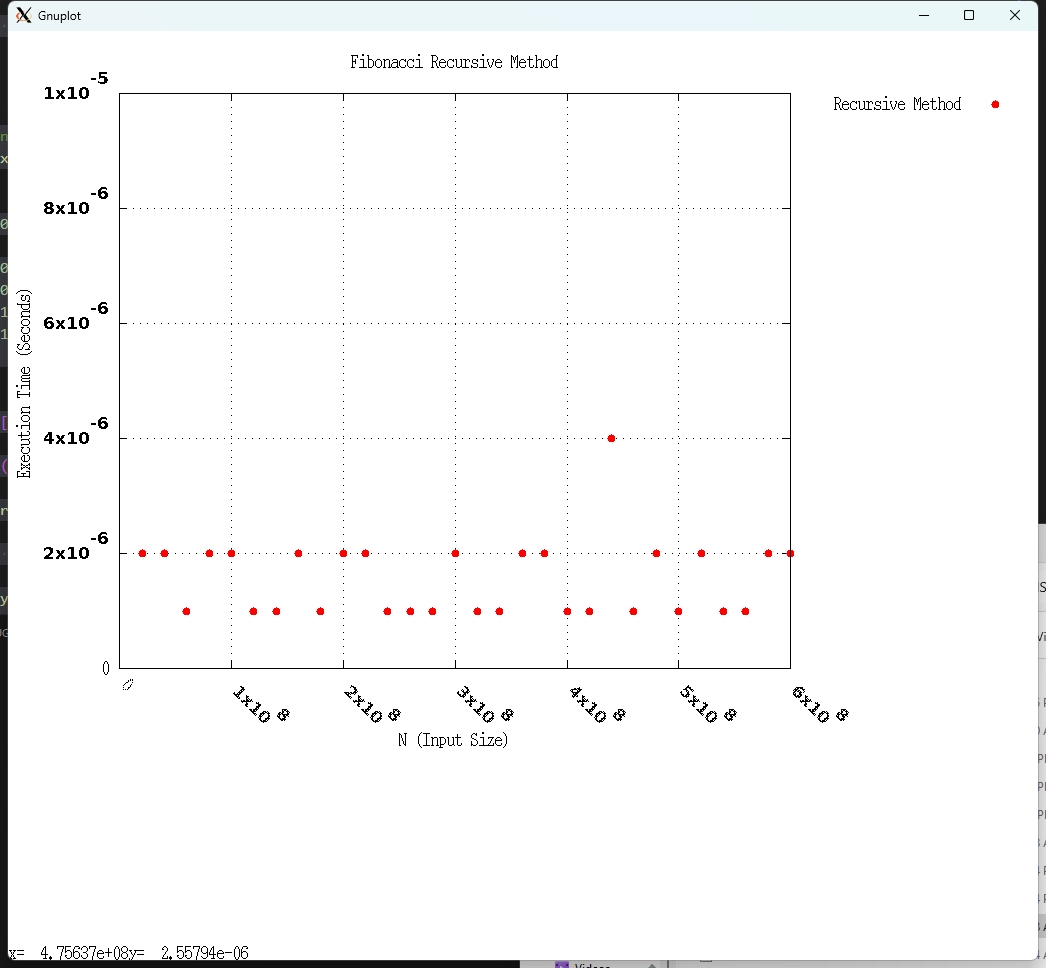
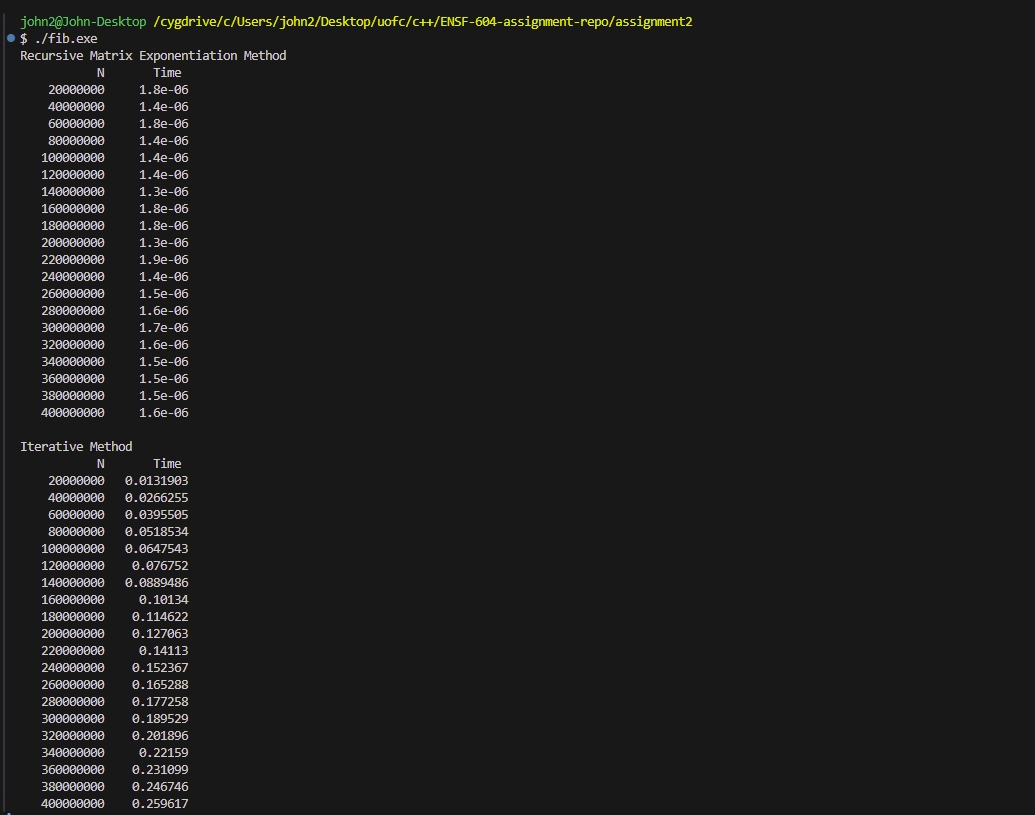
}

myPlot(N\_value, iterative\_result, recursive\_result, 30);

myPlot\_for\_recursive\_method(N\_value, recursive\_result, 30 );

return 0;

}

Gnuplot  
  
  
  
execution output 

# Exercise E

## /\*

## \* compare\_sorts.cpp

## \* ENSF 694 Lab 2 Exercise E

## \* Created by Mahmood Moussavi

## \* Completed by: John Zhou

## \*/

## #include "compare\_sorts.h"

## void to\_lower(char \*str)

## {

## while (\*str)

## {

## \*str = std::tolower(\*str);

## ++str;

## }

## }

## void strip\_punctuation(char \*word)

## {

## int i = 0, j = 0;

## while (word[i])

## {

## if ((word[i] >= 'a' && word[i] <= 'z') || (word[i] >= 'A' && word[i] <= 'Z') || word[i] == '-' || (word[i] >= '0' && word[i] <= '9'))

## {

## word[j] = word[i];

## j++;

## }

## i++;

## }

## word[j] = '\0';

## }

## bool is\_unique(char words[MAX\_UNIQUE\_WORDS][MAX\_WORD\_SIZE], int num\_words, const char \*word)

## {

## for (int i = 0; i < num\_words; i++)

## {

## if (std::strcmp(words[i], word) == 0)

## {

## return false;

## }

## }

## return true;

## }

## void quicksort(int \*indices, char words[MAX\_UNIQUE\_WORDS][MAX\_WORD\_SIZE], int left, int right)

## {

## if (left >= right)

## return;

## int pivot = indices[right];

## int i = left;

## for (int j = left; j <= right - 1; j++)

## {

## if (strcmp(words[indices[j]], words[pivot]) < 0)

## {

## std::swap(indices[i], indices[j]);

## i++;

## }

## }

## std::swap(indices[i], indices[right]);

## int pivotPoint = i;

## quicksort(indices, words, left, pivotPoint - 1);

## quicksort(indices, words, pivotPoint + 1, right);

## }

## void shellsort(int \*indices, char words[MAX\_UNIQUE\_WORDS][MAX\_WORD\_SIZE], int size)

## {

## for (int gap = size / 2; gap > 0; gap /= 2)

## {

## for (int i = gap; i < size; i++)

## {

## int temp = indices[i];

## int j = i;

## while (j >= gap && std::strcmp(words[indices[j - gap]], words[temp]) > 0)

## {

## indices[j] = indices[j - gap];

## j -= gap;

## }

## indices[j] = temp;

## }

## }

## }

## void bubblesort(int \*indices, char words[MAX\_UNIQUE\_WORDS][MAX\_WORD\_SIZE], int size)

## {

## for (int i = 0; i < size - 1; i++)

## {

## for (int j = 0; j < size - 1 - i; j++)

## {

## if (std::strcmp(words[indices[j]], words[indices[j + 1]]) > 0)

## {

## std::swap(indices[j], indices[j + 1]);

## }

## }

## }

## }

## 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)

## {

## auto start = std::chrono::high\_resolution\_clock::now();

## sort\_func(indices, words, 0, num\_words - 1);

## auto end = std::chrono::high\_resolution\_clock::now();

## std::chrono::duration<double> time\_taken = end - start;

## std::cout << sort\_name << " completed in " << time\_taken.count() << " seconds.\n";

## }

## 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)

## {

## auto start = std::chrono::high\_resolution\_clock::now();

## sort\_func(indices, words, num\_words);

## auto end = std::chrono::high\_resolution\_clock::now();

## std::chrono::duration<double> time\_taken = end - start;

## std::cout << sort\_name << " completed in " << time\_taken.count() << " seconds.\n";

## }

## int main()

## {

## const char \*input\_file = "C:\\Users\\john2\\OneDrive\\Desktop\\uofc\\c\\ENSF-604-assignment-repo\\assignment2\\input.txt";

## 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("C:\\Users\\john2\\OneDrive\\Desktop\\uofc\\c\\ENSF-604-assignment-repo\\assignment2\\output\_quicksort.txt", words, indices, num\_words);

## sort\_and\_measure\_shell\_bubble(words, indices, num\_words, shellsort, "Shell Sort");

## write\_words("C:\\Users\\john2\\OneDrive\\Desktop\\uofc\\c\\ENSF-604-assignment-repo\\assignment2\\output\_shellsort.txt", words, indices, num\_words);

## sort\_and\_measure\_shell\_bubble(words, indices, num\_words, bubblesort, "Bubble Sort");

## write\_words("C:\\Users\\john2\\OneDrive\\Desktop\\uofc\\c\\ENSF-604-assignment-repo\\assignment2\\output\_bubblesort.txt", words, indices, num\_words);

## return 0;

## }

## Program output All three output files have the same content

## 

## Execution screenshot



# Part II

# Exercise A

1. 2/N Decreasing as N gets bigger
2. 37 Constant. Not growing
3. Smaller than N
4. N Linear
5. N log( N) log(N) is not a constant and it is growing. This is greater than N
6. N^2 N grows faster than log(N)
7. 2^(N/2) exponential growth is the fastest

# Exercise B

(1)  
O(N) one loop n times

(2)  
O(N^2) two loop n times. n^2

(3)  
O(N^3) one loop n times. Another loop n^2 times. It sums to n^3 .

(4)  
O(N^2) two loops n times. n^2. This is slightly smaller than (2) because j and k may be less than n. The big O notation is still O(N^2)

(5)  
O(N^3) three loops n times. n^3. but slightly smaller than (3), because the j and k are less than n for a lot of the iteration. The big O notation is still O(N^2)

(6)

O(N^3) three loops n times. n^3.