Course: ENSF694 – Summer 2025

Lab #: Lab 2

Instructor: Mahmood Moussavi

Student Name: John Zhou

Submission Date: July 18, 2025

# Part I

## Exercise A

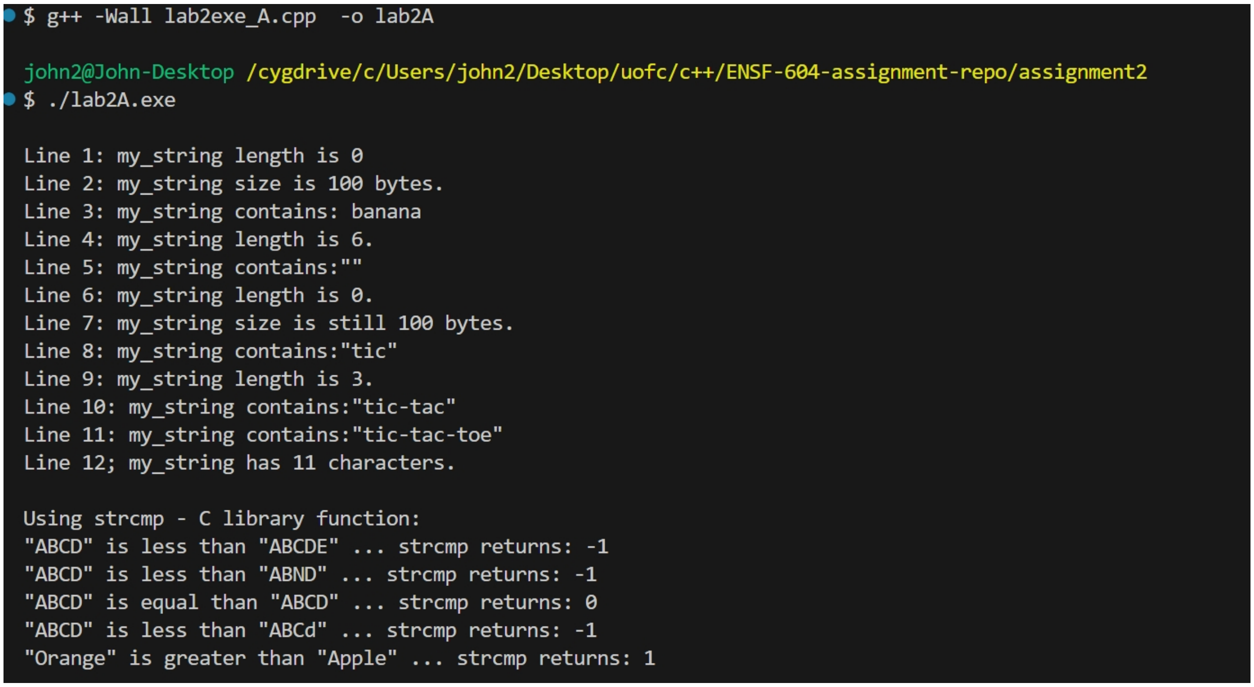
```
/*
 *  my_lab2exe_A.cpp
 *  ENSF 694 Lab 2 Exercise A
 * Created by Mahmood Moussavi
 *  Completed by: John Zhou
 */


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



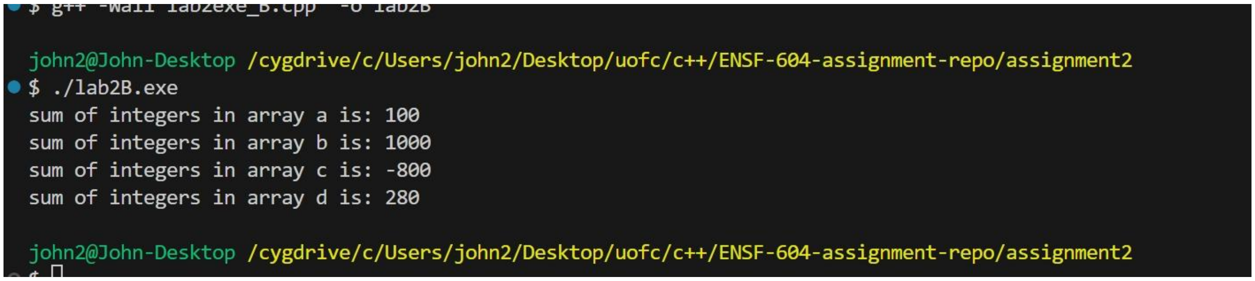
```
$ g++ -Wall lab2exe_A.cpp  -o lab2A

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment2
$ ./lab2A.exe

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
```

# 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

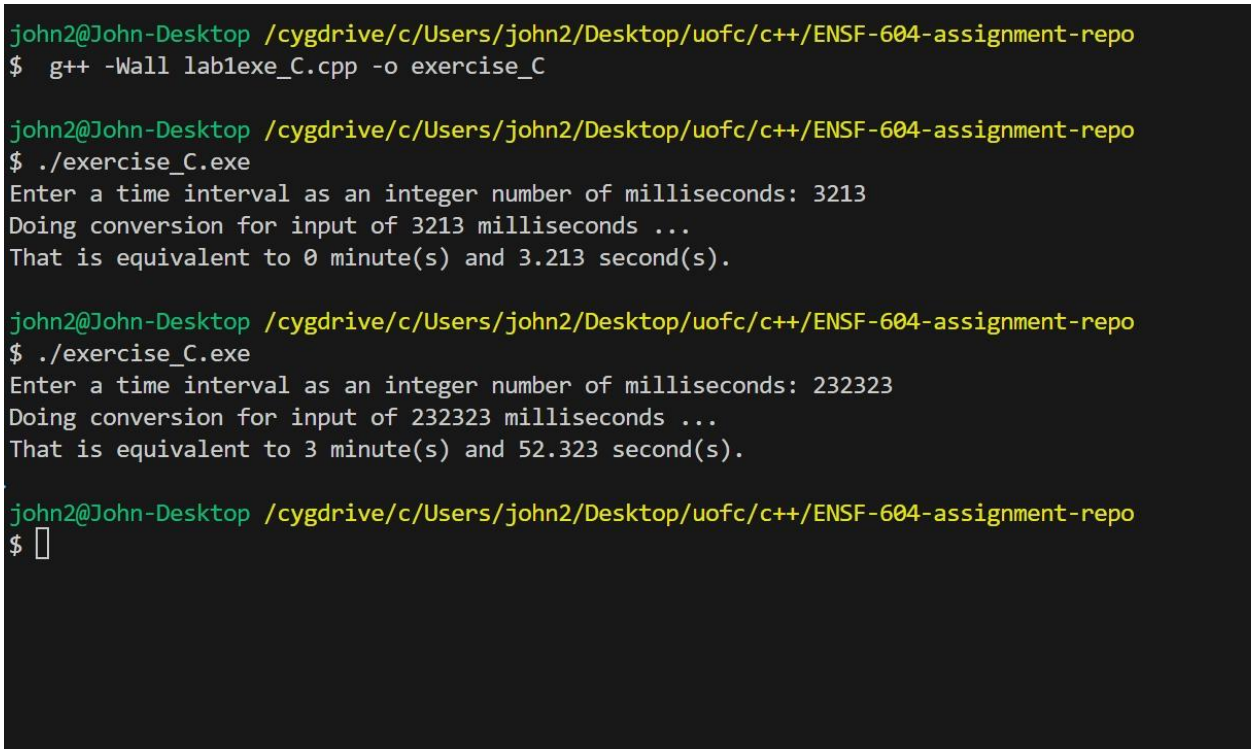


```
$ g++ -Wall lab2exe_B.cpp  -o lab2B

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment2
$ ./lab2B.exe
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

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment2
$ 
```

# 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



```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo
$  g++ -Wall lab1exe_C.cpp -o exercise_C

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo
$ ./exercise_C.exe
Enter a time interval as an integer number of milliseconds: 3213
Doing conversion for input of 3213 milliseconds ...
That is equivalent to 0 minute(s) and 3.213 second(s).

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo
$ ./exercise_C.exe
Enter a time interval as an integer number of milliseconds: 232323
Doing conversion for input of 232323 milliseconds ...
That is equivalent to 3 minute(s) and 52.323 second(s).

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo
$ 
```

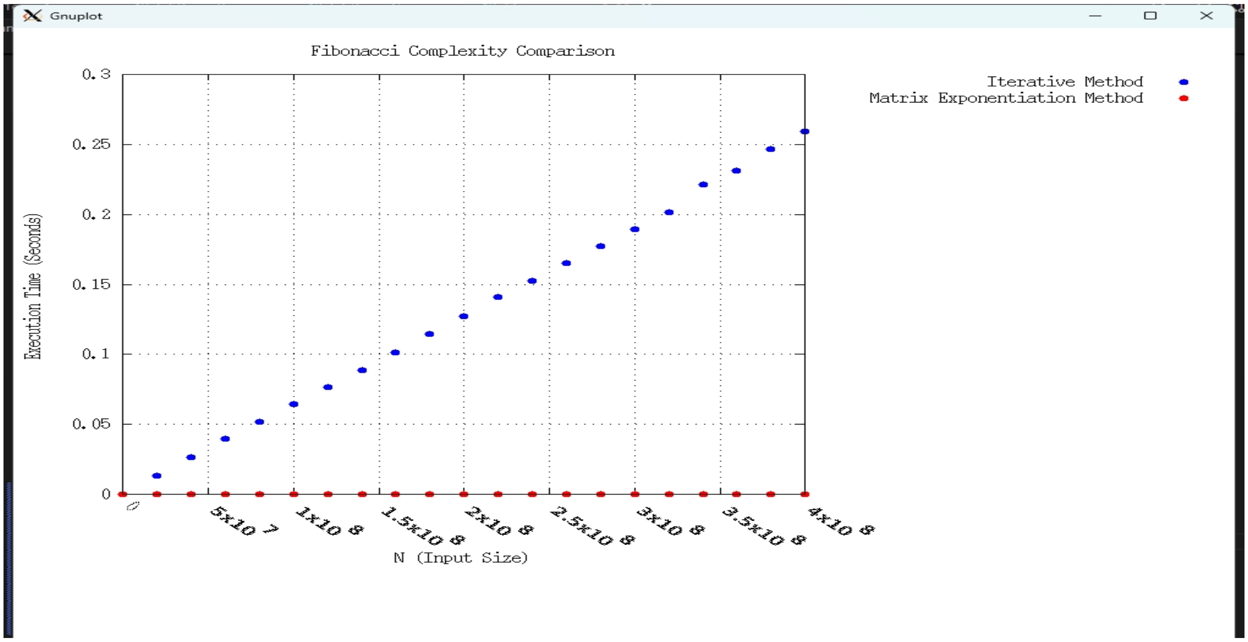
# Exercise D

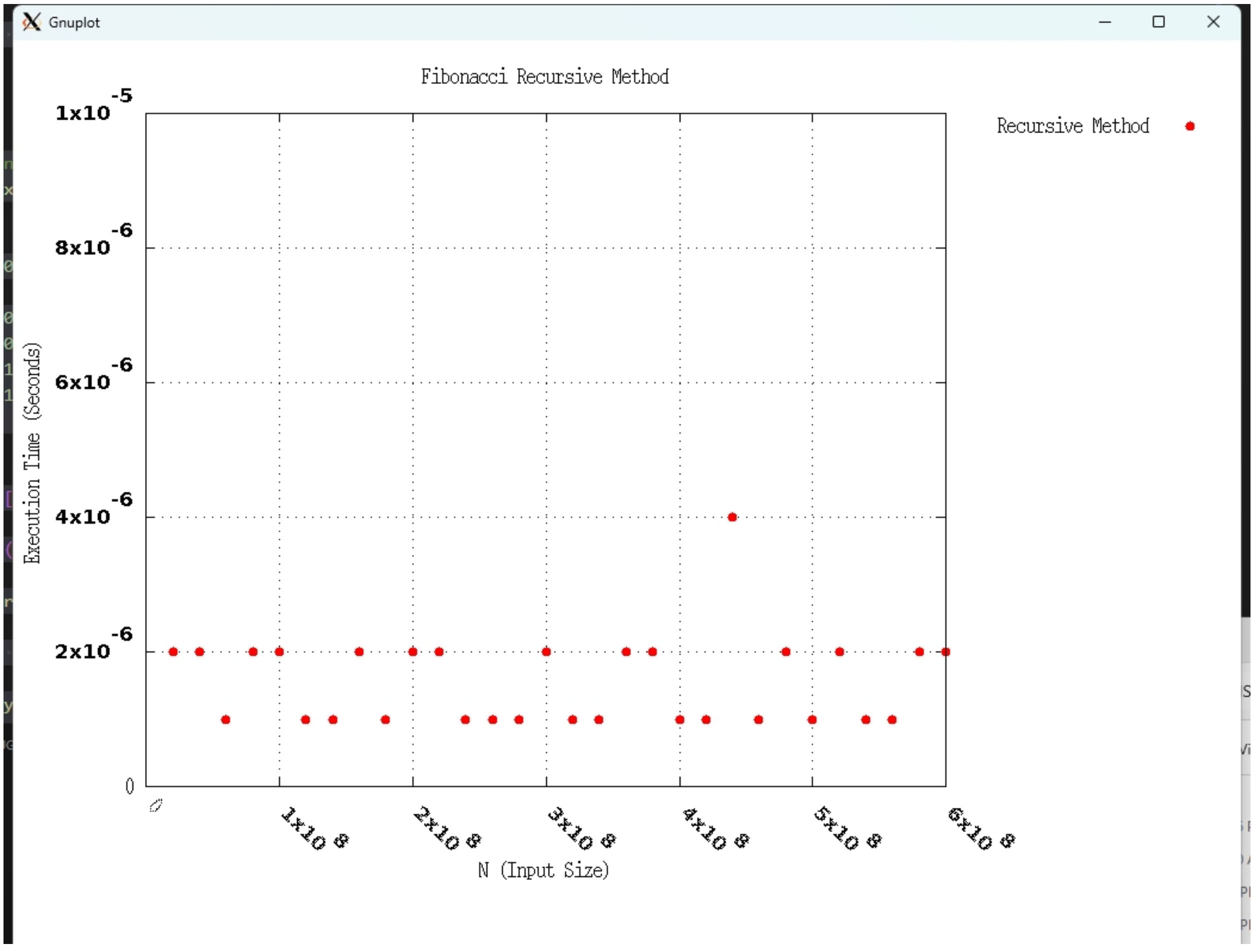
```
/*
 *  fibonacci.cpp
 *  ENSF 694 Lab 2 Exercise D
 * Created by Mahmood Moussavi
 *  Completed by: John Zhou
 */

#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);
}
```

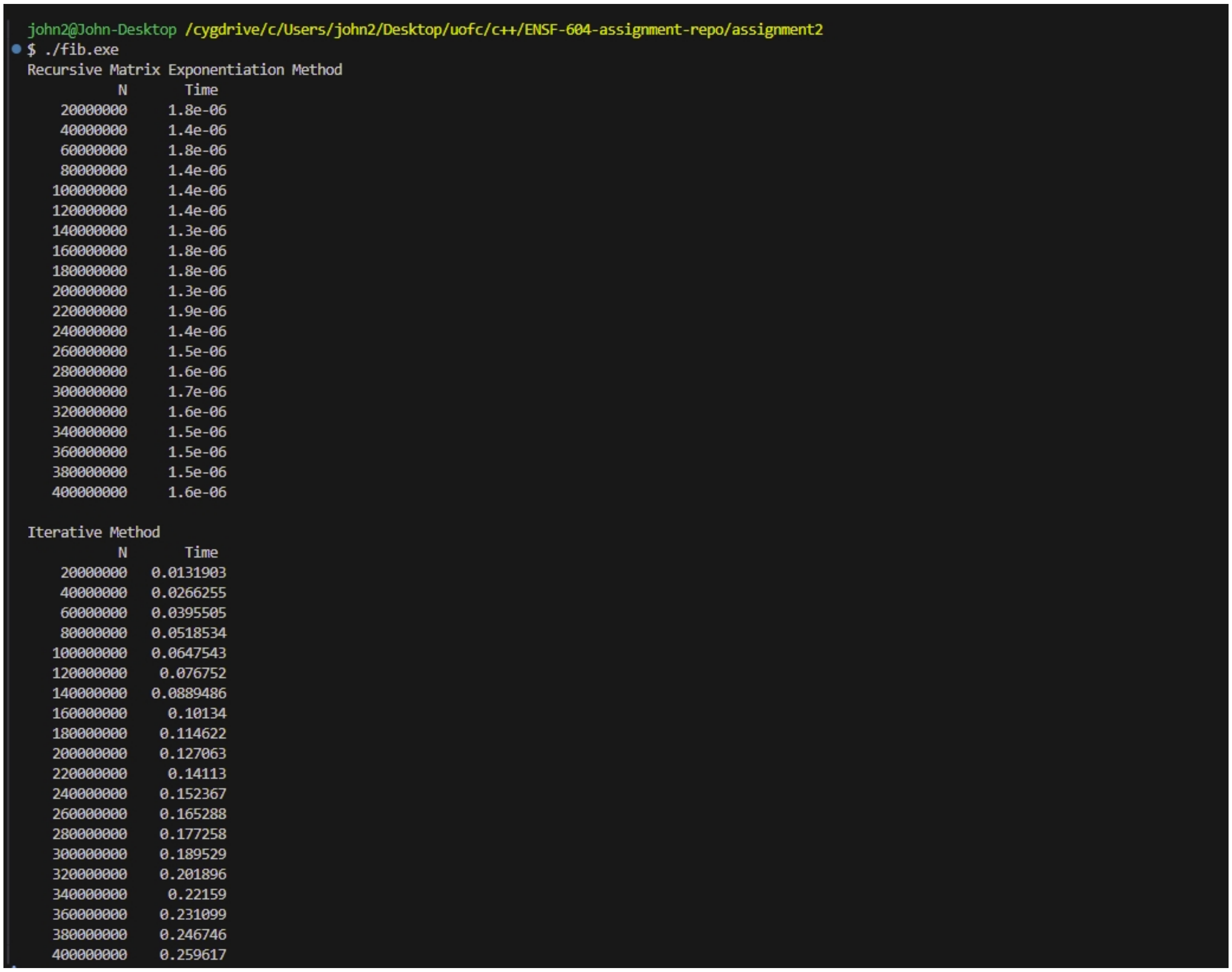
```
// Function to multiply two matrices of size N x N
void multiplyMatrices(int A[N][N], int B[N][N], int result[N][N])
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            result[i][j] = 0;
        }
    }

    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


Gnuplot
Fibonacci Complexity Comparison
Iterative Method
Matrix Exponentiation Method
Execution Time (Seconds)
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5x10 7
1x10 8
1.5x10 8
2x10 8
2.5x10 8
3x10 8
3.5x10 8
4x10 8
N (Input Size)


Gnuplot
Fibonacci Recursive Method
Recursive Method
Execution Time (Seconds)
1x10 -5
8x10 -6
6x10 -6
4x10 -6
2x10 -6
0
0
1x10 8
2x10 8
3x10 8
4x10 8
5x10 8
6x10 8
N (Input Size)

## execution output



```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment2
$ ./fib.exe
Recursive Matrix Exponentiation Method
           N       Time
    20000000     1.8e-06
    40000000     1.4e-06
    60000000     1.8e-06
    80000000     1.4e-06
   100000000     1.4e-06
   120000000     1.4e-06
   140000000     1.3e-06
   160000000     1.8e-06
   180000000     1.8e-06
   200000000     1.3e-06
   220000000     1.9e-06
   240000000     1.4e-06
   260000000     1.5e-06
   280000000     1.6e-06
   300000000     1.7e-06
   320000000     1.6e-06
   340000000     1.5e-06
   360000000     1.5e-06
   380000000     1.5e-06
   400000000     1.6e-06

Iterative Method
           N       Time
    20000000   0.0131903
    40000000   0.0266255
    60000000   0.0395505
    80000000   0.0518534
   100000000   0.0647543
   120000000    0.076752
   140000000   0.0889486
   160000000     0.10134
   180000000    0.114622
   200000000    0.127063
   220000000     0.14113
   240000000    0.152367
   260000000    0.165288
   280000000    0.177258
   300000000    0.189529
   320000000    0.201896
   340000000     0.22159
   360000000    0.231099
   380000000    0.246746
   400000000    0.259617
```

## 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)
```

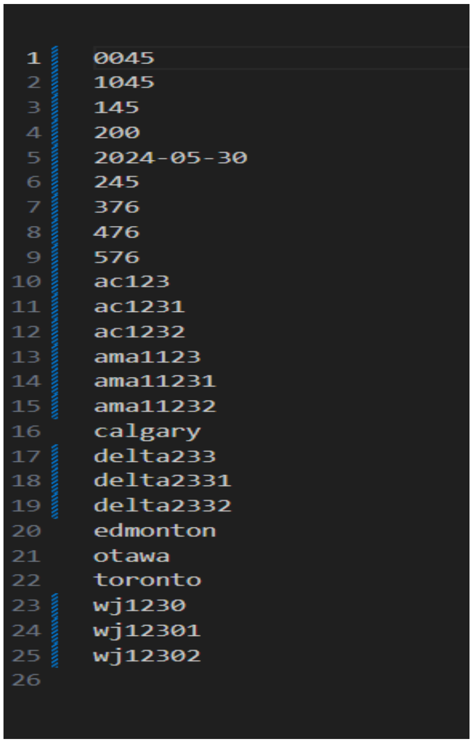
```
{
   for (int i = 0; i < num_words; i++)
   {
      if (std::strcmp(words[i], word) == 0)
      {
         return false;
      }
   }
   return true;
}

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;
}
```

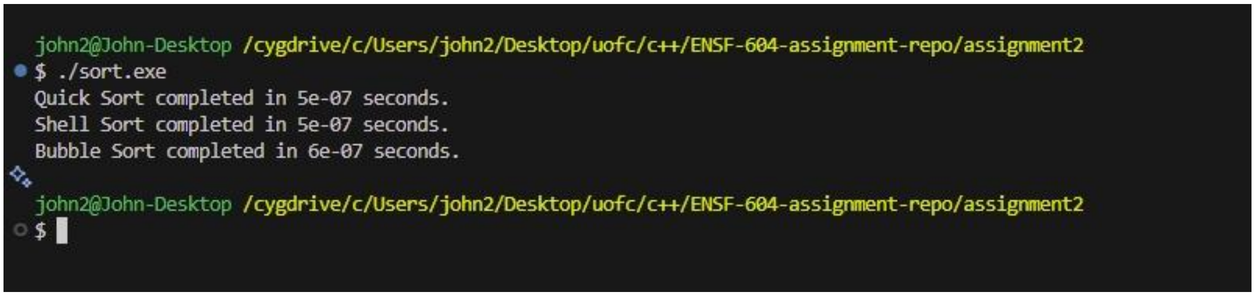
## Program output

All three output files have the same content



```
0045
1045
145
200
2024-05-30
245
376
476
576
ac123
ac1231
ac1232
ama1123
ama11231
ama11232
calgary
delta233
delta2331
delta2332
edmonton
otawa
toronto
wj1230
wj12301
wj12302

```

## Execution screenshot



```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment2
$ ./sort.exe
Quick Sort completed in 5e-07 seconds.
Shell Sort completed in 5e-07 seconds.
Bubble Sort completed in 6e-07 seconds.

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment2
$ 
```

# Part II

## Exercise A

1. 2/N Decreasing as N gets bigger
2. 37 Constant. Not growing
3. $\sqrt{N}$ 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.