UCSD 167113 Data Structure And Algorithms in C/C++
HW1
Cheng FEI

```
//
//  main.c
//  insert-algo-profiler
//
//  Created by Cheng FEI on 2022/9/30.
//

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

#define READINGS 60
#define INSERTS_PER_READING 1000
#define NUM_COMMANDS 2

int *insert(int *, int, int, int);
void plotBigO(int[], double[]);

int main(int argc, const char * argv[]) {
    // Initialize an empty array.
    int *array = NULL;
    int length = 0;
    // Clock settings.
    clock_t startTick;
    clock_t endTick;
    double elapsedSeconds;
    // Declare params for insert function.
    int index;
    int value;
    // Declare arrays of x-axis and y-axis values.
    int x[READINGS];
    double y[READINGS];
    // Set the random seed.
    srand((unsigned)time(0));
    // Print table column titles.
    printf("Array length\tSeconds per insert\n");

    // Loop READINGS readings.
    for (int i = 0; i < READINGS; i++) {
        // Start timing.
        startTick = clock();
        // Insert values INSERTS_PER_READINGS times.
        for (int j = 0; j < INSERTS_PER_READING; j++) {
            index = rand() % (length+1);
            value = rand();
            // If insert function fails, exit the program and alert the error.
            if (!(array = insert(array, length, index, value))) {
                fprintf(stderr, "Insert failed, exiting!");
                exit(EXIT_FAILURE);
            }
            length++;
        }
        // End timing.
        endTick = clock();
        // Record seconds taken in the reading.
        elapsedSeconds = (double) (endTick - startTick) / (INSERTS_PER_READING*CLOCKS_PER_SEC);
        // Print readings and elasped seconds in tabular format.
        printf("%*d\t%*f\n", 12, INSERTS_PER_READING*(i+1), 18, elapsedSeconds);
        // Store number of inserts.
        x[i] = INSERTS_PER_READING*(i+1);
        // Store elapsed seconds per insert.
        y[i] = elapsedSeconds;
    }
    // Plot elasped seconds per insert (x) against number of inserts (y) with gnuplot library.
    plotBigO(x, y);
    // Free the original dynamic array.
```

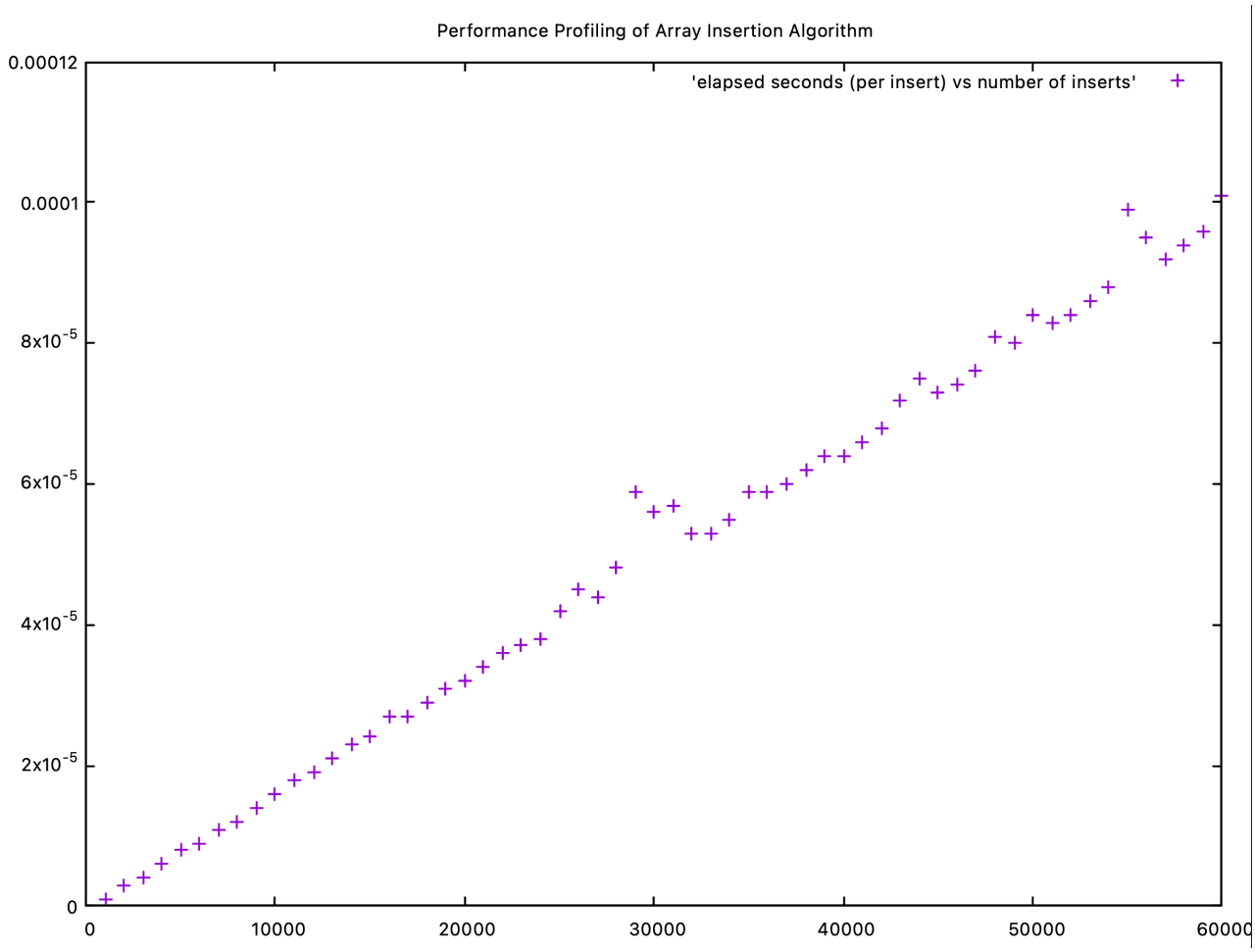
```
    free(array);
    return EXIT_SUCCESS;
}

/*
 Insert a new value by index into the old array and return new array.
 Params: array -- pointer to integer pointing at old array
                  (uninitialized if length is 0, released after calling this function)
 Params: length -- number of elements in old array.
 Params: index -- location to insert new value.
 Params: value -- integer to be inserted in old array.
 Returns: pointer to integer pointing at new array after insertion.
 */
int *insert(int *array, int length, int index, int value) {
    // Initialize new array.
    int *newArray;
    // Allocate dynamic memory to new array.
    if (!(newArray = (int *)malloc(sizeof(int)*(length+1)))) {                          // O(1)
        // If out of memory, print "Out of memory!" and return NULL.
        fprintf(stderr, "Out of memory!");                                              // O(1)
        return NULL;                                                                    // O(1)
    }
    // If old array is empty, simply add new value and return new array.
    if (length == 0) *newArray = value;                                                 // O(1)
    // If old array is not empty,
    else {
        // Copy values from 0 to the index in old array to new array.
        for (int i = 0; i < index; i++) newArray[i] = array[i];                         // O(n)
        // Insert new value into index.
        newArray[index] = value;                                                        // O(1)
        // Copy the rest values from index+1 to the end of old array to new array.
        for (int i = index; i < length; i++) newArray[i+1] = array[i];                  // O(n)
        // Free the dynamic memory occupied by old array.
        free(array);                                                                    // O(1)
    }
    return newArray;                                                                    // O(1)
    /*
     Overall Big-O Analysis:
     Time complexity = O(1) + O(1) + O(1) + O(1) + O(n)*O(1) + O(n)*O(1) + O(1)
                     = O(n)
     Among codes, two for-loops contribute most heavily to the time complexity.
     */
}

/*
 Plot elapsed seconds against number of inserts with gnuplot.
 Params: xvals -- int array with number of inserts.
 Params: yvals -- double array with elapsed seconds per insert.
 */
void plotBigO(int xvals[], double yvals[]) {
    // Commands setting, including chart's title and legend.
    char *commandsForGnuplot[] = {"set title \"Performance Profiling of Array Insertion
Algorithm\"", "plot 'elapsed seconds (per insert) vs number of inserts'"};
    FILE *temp = fopen("elapsed seconds (per insert) vs number of inserts", "w");
    /* Opens an interface that one can use to send commands as if they were typing into the
     * gnuplot command line.  "The -persistent" keeps the plot open even after your
     * C program terminates.
     */
    // Change location of gnuplot to its actual location in os.
    FILE *gnuplotPipe = popen ("/usr/local/bin/gnuplot -persistent", "w");
    int i;
    // Write the data to a temporary file
    for (i=0; i < READINGS; i++)
    {
    fprintf(temp, "%d %lf \n", xvals[i], yvals[i]);
    }
    // Send commands to gnuplot one by one.
    for (i=0; i < NUM_COMMANDS; i++)
    {
    fprintf(gnuplotPipe, "%s \n", commandsForGnuplot[i]);
    }
}
```

```
Console Output:
Array length   Seconds per insert
        1000             0.000001
        2000             0.000003
        3000             0.000005
        4000             0.000007
        5000             0.000009
        6000             0.000010
        7000             0.000012
        8000             0.000012
        9000             0.000014
       10000             0.000016
       11000             0.000018
       12000             0.000019
       13000             0.000021
       14000             0.000022
       15000             0.000024
       16000             0.000027
       17000             0.000029
       18000             0.000032
       19000             0.000031
       20000             0.000033
       21000             0.000034
       22000             0.000037
       23000             0.000040
       24000             0.000039
       25000             0.000040
       26000             0.000042
       27000             0.000044
       28000             0.000049
       29000             0.000048
       30000             0.000050
       31000             0.000050
       32000             0.000052
       33000             0.000056
       34000             0.000056
       35000             0.000058
       36000             0.000059
       37000             0.000062
       38000             0.000068
       39000             0.000064
       40000             0.000065
       41000             0.000065
       42000             0.000067
       43000             0.000070
       44000             0.000082
       45000             0.000082
       46000             0.000075
       47000             0.000076
       48000             0.000077
       49000             0.000079
       50000             0.000085
       51000             0.000085
       52000             0.000086
       53000             0.000088
       54000             0.000088
       55000             0.000090
       56000             0.000091
       57000             0.000092
       58000             0.000097
       59000             0.000097
       60000             0.000099
Program ended with exit code: 0
```

Plot:



Answers:

4. Line-by-line analysis has been included in the source code.
The overall big O performance is O(n), and two for-loops contribute most heavily to big O performance.

5. As the array grows, the performance degrades. So my big O analysis matches the program's result.