Stephen Townsend
CS 325-400
10/1/17

Assignment 1

1) Describe a Θ (n lgn) time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x. Explain why the running time is Θ (n lgn).

The running time is going to be Θ (n lgn) because the worst case of merge sort is O(n log n). Unless you use a slower searching algorithm the running time would be different. After the sort use a search such as a binary search (O(log n)). This search will go through the array to find two elements and for x – S[n]. The running time will be based on the worst case and that will be the merge sort O(n log n).

2) For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and explain.

F(n) = O(g(n)) → f(n) <= g(n); F(n) = Ω (g(n)) → f(n) >= g(n); F(n) = Θ (g(n)) → f(n) = g(n);

Using theorem $\lim_{n\to\infty} a^n = \begin{matrix} 0 \; if\ a < 1 \\ 1 \; if\ a = 1 \\ \infty \; if\ a > 1 \end{matrix}$

a. f(n) = $n^{0.25}$; g(n) = $n^{0.5}$

$$\lim_{n\to\infty} \frac{n^{0.25}}{n^{0.5}} = n^{0.25-0.5} = \lim_{n\to\infty} \frac{1}{n^{0.25}}$$

$Thus\ n^{0.25}\ is\ O(n^{0.5})$

The two functions when taking the limit is = 0 because the denominator is growing faster than the numerator and thus the result would be < 1. As a result, f(n) < g(n).

b. f(n) = log n^2; g(n) = ln n

$$\lim_{x\to\infty} \frac{log\ n^2}{log\ n} = \frac{2log\ n}{log\ n}$$

$Thus\ log\ n^2\ is\ \Theta(\ln n)$

The two functions when taking the limit is = 0 because with a log function the coefficient has no effect and that makes the situation log n / log n or log n = log n. As a result, f(n) and g(n) grow at the same rate.

c. f(n) = nlog n; g(n) =n$\sqrt{n}$

$$\lim_{n\to\infty} \frac{n\log n}{n\sqrt{n}} = \frac{\log n}{\sqrt{n}}$$

$Thus\ n\log n\ \ is\ \mathrm{O}(\mathrm{n}\sqrt{\mathrm{n}})$

The two functions when taking the limit is = 0 because the denominator is growing faster than the numerator and thus the result would be < 1. As a result, f(n) < g(n).

d. f(n) = 4^n; g(n) = 3^n

$$\lim_{n\to\infty} \frac{4^n}{3^n} = (\frac{4}{3})^n = \infty$$

$Thus\ 4^n\ is\ \Omega(3^n)$

The two functions when taking the limit is = ∞ because the numerator is growing faster than the denominator. This makes it > 1 and thus f(n) > g(n).

e. f(n) = 2^n; g(n) = 2^{n+1}

$$\lim_{n\to\infty} \frac{2^n}{2^{n+1}}$$

$Thus\ 2^n\ is\ \mathrm{O}(2^{n+1})$

The two functions when taking the limit is = 1 because the denominator is growing faster than the numerator. This makes it < 1 and thus f(n) < g(n).

f. f(n) = 2^n; g(n) = n!

$$\lim_{n\to\infty} \frac{2^n}{n!}$$

$Thus\ 2^n\ is\ \mathrm{O}(\mathrm{n}!)$

The two functions when taking the limit is = ∞ because the denominator is growing faster than the numerator. f(n) < g(n).

Stephen Townsend
CS 325-400
10/1/17

3) Let f_1 and f_2 be asymptotically positive non-decreasing functions. Prove or disprove each of the following conjectures. To disprove give a counter example.

a. If $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$ then $f_1(n)+ f_2(n) = O(g(n))$.

1. by definition $f_1(n) = O(g(n))$ implies there exist positive constants c_1 and n1 such that $0 \le f_1(n) \le c_1g(n)$ for all $n \ge n_1$

2. by definition $f_2(n) = O(g(n))$ implies there exist positive constants c_2 and n_2 such that $0 \le f_2(n) \le c_2g(n)$ for all $n \ge n_2$

3. Show $f_1(n)+ f_2(n) = O(g(n))$ that is there exist positive constants c_3, c_4 and n_3 such that $0 < c_3\, g(n) \le c_4\, g(n)$ for all $n \ge n_3$

4. Let $g = g(n)$

$f_1(n) \le g$ for $n \ge (n_1 + n_2)$

$f_2(n) \le g$

$0 < f_1 + f_2 \le g + g$

$0 < f_1 + f_2 \le 2g$

Therefore $f_1(n) + f_2(n) \le g(n)$ and $f_1(n) + f_2(n) = O(g(n))$

b. If $f(n) = O(g_1(n))$ and $f(n) = O(g_2(n))$ then $g_1(n) = \Theta(g_2(n))$

1. by definition $f(n) = O(g_1(n))$ implies there exist positive constants c_1 and n_1 such that $0 \le f(n) \le c_1g(n)$ for all $n \ge n_1$

2. by definition $f(n) = O(g_2(n))$ implies there exist positive constants c_2 and n_2 such that $0 \le f(n) \le c_2g(n)$ for all $n \ge n_2$

3. Show $g_1(n) = \Theta(g_2(n))$ that is there exist positive constants c_3, c_4 and n_2 such that $0 < c_3\, g(n) \le c_4\, g(n)$ for all $n \ge n_2$

Let $f(n) = n$. $g_1(n) = n^2$ and $g_2(n) = n^3$

$f(n) = O(g_1(n))$	$f(n) = O(g_2(n))$	$g_1(n) = \Theta(g_2(n))$
$n = O(n^2)$	$n = O(n^2)$	$n^2 \neq \Theta\ (n^3)$

Stephen Townsend
CS 325-400
10/1/17

5) Merge Sort vs Insertion Sort Running time analysis

a) Include a "text" copy of the modified code

Merge Sort

```
//Visual Studios - to remove compatibility warnings
#define _CRT_SECURE_NO_WARNINGS

//Header Files
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>

//Universal File Pointer
FILE *fp;

//Function prototypes
void mergeSort(int *arr, int, int, int);
void merge(int *arr, int, int, int, int);
void fillArrayGenerator(int *line, int);


/*******************************************
*                       main
* This will be used to direct the program.
*******************************************/
void main()
{
        srand(time(NULL));                              //Seed random number
        int runTime = true;
        char input[2] = { NULL };
        int size = 0;
        char str[2] = "n";
        double ranFor = 0;

        while (runTime == true)
        {
                printf("How large is array? ");
                scanf("%d", &size);

                printf("Random Number to Sort: %d\n", size);

                int *line = malloc(size * sizeof(int));

                memset(line, 0, sizeof line);           //Clear Buffer

                fillArrayGenerator(line, size);

                clock_t begin = clock();                //Start the cycle clock

                mergeSort(line, 0, (size - 1), size);   //Sort the row

                clock_t end = clock();                  //End the cycle clock

                double ranFor = (double)(end - begin) / CLOCKS_PER_SEC;

                printf("Running time: %f\n\n", ranFor);

                free(line);                             //Free memory array
        }
        printf("Ending Program\n");
}
```

Stephen Townsend
CS 325-400
10/1/17

```
/*******************************************
*                       mergeSort
* This function uses the merge sort.  It
* will recursively spit the array in half
* until it gets to the lowest point.
* From there the function will call merge
* to combine the split arrays in sorted
* order.
*******************************************/
void mergeSort(int *line, int low, int high, int size)
{
        /*Priority:  low (Lower limit) < split (middle limit) < high (higher limit)*/

        if (low < high)                                 //Check if Left less then right.  More than two elements available. Can be split
        {
                int split = (low + high) / 2;           //Find the middle location of the array

                mergeSort(line, low, split, size);      //Recursive left side of array
                mergeSort(line, (split + 1), high, size);       //Recursive right side of array
                merge(line, low, split, high, size);
        }
}
/*******************************************
*                       merge
* This function uses merges the split array
* from the mergesort function.  This will
* look at the left or right side and
* choose the smallest integer.  The integer
* will be placed in a temp array.  The temp
* array will then put the sorted elements
* back into the line array.
*******************************************/
void merge(int *line, int low, int split, int high, int size)
{
        int *temp = malloc(size * sizeof(int));         //array used for merging

        int left = low;                 //Left Side
        int right = split+1;            //Right Side
        int i = 0;

        while (left <= split && right <= high)          //while elements in both lists
        {
                if (line[left] < line[right])           //If left is less then right
                {
                        temp[i] = line[left];           //Place left into temp
                        //Shift to next location
                        i++;
                        left++;
                }
                else
                {
                        temp[i] = line[right];          //Place right into temp
                        //Shift to next location
                        i++;
                        right++;
                }
        }

        while (left <= split)                           //Check if there are more elements on the left
        {
                temp[i] = line[left];                   //Place into temp
                //Shift to next location
                i++;
                left++;
        }
while (right <= high)                                   //Check if there are more elements on the right
            {
                        temp[i] = line[right];          //Place into temp
                        //Shift to next location
                        i++;
                        right++;
            }

            i--;

            while (i >= 0)
            {
                        line[low + i] = temp[i];
                        i--;
            }

            free(temp);
}
/********************************************
*                       randomGenerator
* This function is used to generate random
* numbers to fill the dynamic array.  From
* 0 to 10000.
********************************************/
void fillArrayGenerator(int *line, int size)
{
            int i = 0;

            while (size != 0)
            {
                        line[i] = rand() % 10001;       //Select a random room between 0-10,000
                        i++;
                        size--;
            }
}
```

Stephen Townsend
CS 325-400
10/1/17

Insertion Sort

```
//Visual Studios - to remove compatibility warnings
#define _CRT_SECURE_NO_WARNINGS

//Header Files
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdbool.h>

//Universal File Pointer
FILE *fp;

//Function prototypes
void insertionSort(int *line, int);
void fillArrayGenerator(int *line, int);

/*******************************************
*                       main
* This will be used to direct the program.
*******************************************/
void main()
{
        srand(time(NULL));                      //Seed random number
        int runTime = true;
        char input[2] = { NULL };
        int size = 0;
        char str[2] = "n";
        double ranFor = 0;

        while (runTime == true)
        {
                printf("How large is array? ");
                scanf("%d", &size);

                printf("Random Number to Sort: %d\n", size);

                int *line = malloc(size * sizeof(int));

                memset(line, 0, sizeof line);           //Clear Buffer

                fillArrayGenerator(line, size);

                clock_t begin = clock();                //Start the cycle clock

                insertionSort(line, size);              //Sort the row

                clock_t end = clock();                  //End the cycle clock

                double ranFor = (double)(end - begin) / CLOCKS_PER_SEC;

                printf("Running time: %f\n\n", ranFor);

                free(line);                                             //Free memory array
        }
        printf("Ending Program\n");
}
```

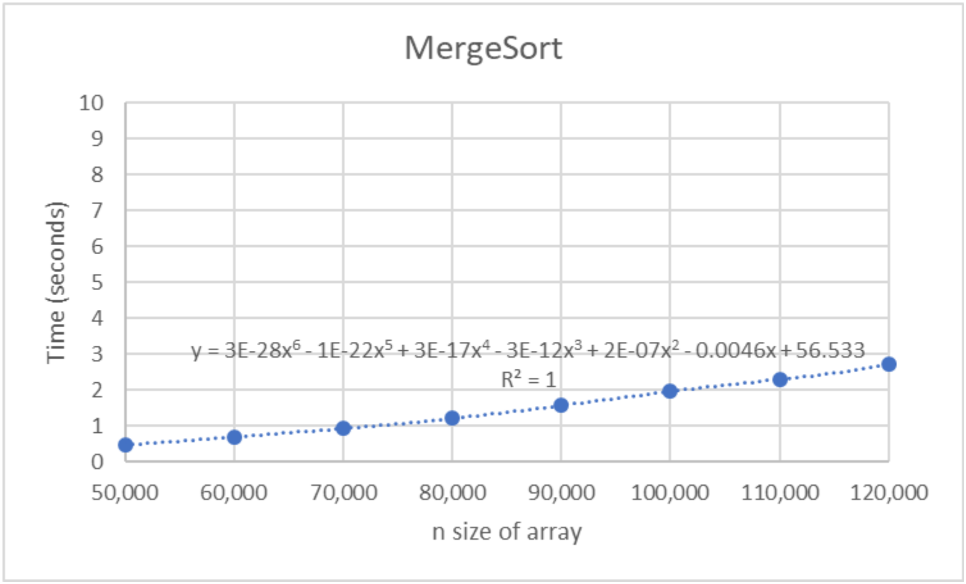
Stephen Townsend
CS 325-400
10/1/17

```
/*******************************************
*                       insertionSort
* This function uses the insertion sort.
* An element is compared to the previous
* and if it is less than that element
* the positions swap.  This will go down
* the array until all elements are sorted.
* The key is the element being compared.
* Anything greater moves ahead of the
* current position and the key moves down
* one position.
*******************************************/
void insertionSort(int *line, int size)
{
        int i;
        int prev;
        int key;

        for (i = 1; i < size; i++)
        {
                key = line[i];                          //Element to compare
                prev = i - 1;

                while (prev >= 0 && line[prev] > key)   //Check if at end location and location is greater than key
                {
                        line[prev + 1] = line[prev];    //Move prev element forward
                        prev = prev - 1;                //Go back one element
                }

                line[prev + 1] = key;                   //Move key element back
        }
}


/********************************************
*                       randomGenerator
* This function is used to generate a random
* number so that the program can use a random
* room during the connections process.
********************************************/
void fillArrayGenerator(int *line, int size)
{
        int i = 0;

        while (size != 0)
        {
                line[i] = rand() % 10001;               //Select a random room between 0-10,000
                i++;
                size--;
        }
}
```

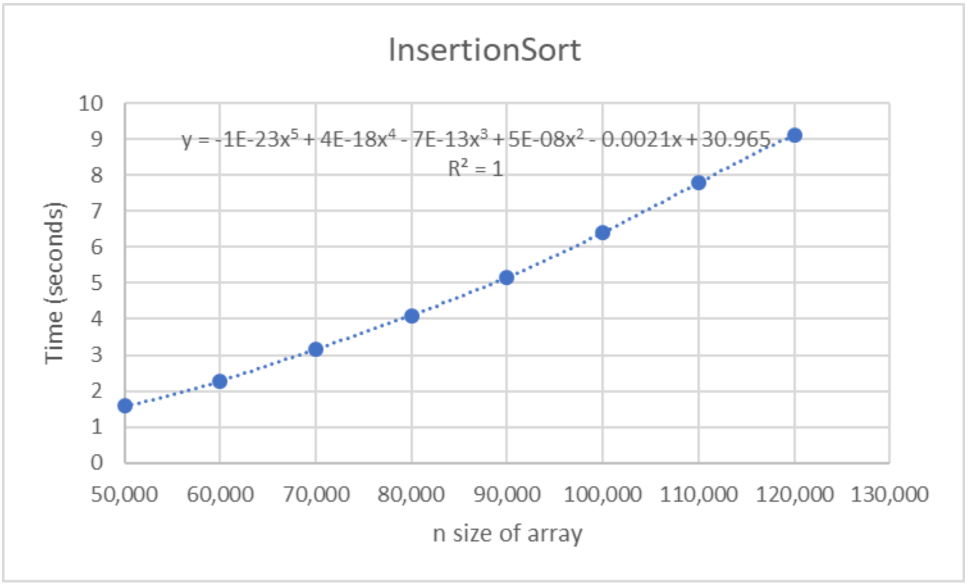
Stephen Townsend
CS 325-400
10/1/17

b, c, & d)

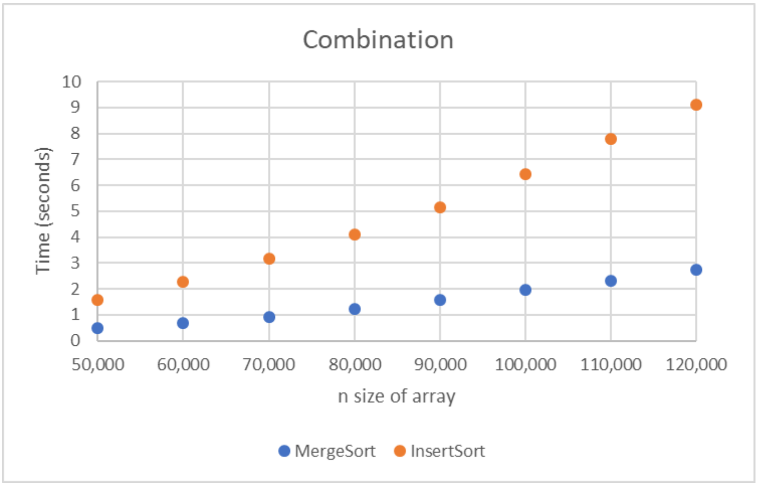
Merge Sort	
n	Time
50,000	0.483
60,000	0.696
70,000	0.93
80,000	1.21
90,000	1.569
100,000	1.965
110,000	2.301
120,000	2.724



Merge Sort	
n	Time
50,000	1.579
60,000	2.275
70,000	3.174
80,000	4.104
90,000	5.158
100,000	6.413
110,000	7.791
120,000	9.12



Merge Sort		
n	MergeSort	InsertSort
50,000	0.483	1.579
60,000	0.696	2.275
70,000	0.93	3.174
80,000	1.21	4.104
90,000	1.569	5.158
100,000	1.965	6.413
110,000	2.301	7.791
120,000	2.724	9.12



Stephen Townsend
CS 325-400
10/1/17

c) Which graphs represent the data best?

I believe that the combination graph represents the data the best because you can see an actual comparison between the two algorithims.

d) What type of curve best fits each data set?

For both the graphs, the best fit curve is a polynomial.

- For insertion sort, I used a 5th degree polynomial to get an R^2 of 1.
  - The equation for the insertion sort is:

    $y = 1E\text{-}23x^5 + 4E\text{-}18x^4 - 7E\text{-}13x^3 + 5E\text{-}08x^2 - 0.0021x + 30.965.$
- For Merge sort, I used a 6th degree polynomial to get an R^2 of 1.
  - The equation for the merge sort is:

    $y = 3E\text{-}28x^6 - 1E\text{-}22x^5 + 3E\text{-}17x^4 - 3E\text{-}12x^3 + 2E\text{-}07x^2 - 0.0046x + 56.533$
  - I tested a linear regression and that would have been sufficient with an R^2 value of 0.9882.

e) How do your experimental running times compare to the theoretical running times of the algorithms?

The running times seem to in line with how the two algorithms theoretically act. Merge sort runs faster and takes less time than the insertion sort.