CS352 - Homework 1

Marc Tibbs (tibbsm@oregonstate.edu)

Due Date: January 14, 2018

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

– An example algorithm could sort the set S using a merge sort which executes in $\Theta(n\,log\,n)$. After sorting, the algorithm would then search the sorted set for pairs of integers which sum to x. This secondary search can be accomplished using a binary search algorithm which runs in $\Theta(log\,n)$ for each of the integers$n$ in the set S. The total time for searching for pairs that sum to x is then: $\Theta(n\,log\,n)$. The resulting execution time for the whole process can then be calculated as $\Theta(n\,log\,n) * (n\,log\,n)$ or $\Theta(2n\,log\,n)$. As we are only interested in the growth rate of the function we can drop the constant(2), which results in final execution time of $\Theta(n\,log\,n)$.

---

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.

a) $f(n) = n^{0.25} \qquad g(n) = n^{0.5}$

$\lim_{x\to\infty} \frac{n^{0.25}}{n^{0.5}} = \frac{n^{0.25}}{\sqrt{n}} = \frac{1}{n^{0.25}} \approx 0$

$\mathcal{O}$(g(n))

b) $f(n) = log\,n^2 \qquad g(n) = ln\,n$

$\lim_{x\to\infty} \frac{log\,n^2}{ln\,n} = \frac{2}{ln(10)} \approx 0.868589$

Θ(g(n))

c) $f(n) = n\,log\,n \qquad g(n) = n\sqrt{n}$

$\lim_{x\to\infty} \frac{n\,log\,n}{n\sqrt{n}} = 0$

$\mathcal{O}$(g(n))

d) $f(n) = 4^n \qquad g(n) = 3^n$

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

Ω(g(n))

e) $f(n) = 2^n \qquad g(n) = 2^{n+1}$

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

Θ(g(n))

f) $f(n) = 2^n \qquad g(n) = n!$

$\lim_{x\to\infty} \frac{2^n}{n!} = 0$

$\mathcal{O}$(g(n))

---

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 exists positive constants c_1 and n_0 such that $0 \leq f_1(n) \leq c_1 g(n)$ for all $n \geq n_0$.
2. By definition $f_2(n) = O(g(n))$ implies there exists positive constants c_2 and n_1 such that $0 \leq f_2(n) \leq c_2 g(n)$ for all $n \geq n_1$.
3. Show $f_1(n)+f_2(n) = O(g(n))$ and that there exists positive constants c_3 and n_2 such that $0 \leq f_1 + f_2 \leq c_3 g(n)$ for all $n \geq n_2$.

By combining 1 and 2: $0 \leq f_1(n) + f_2(n) \leq c_1 g(n) + c_2 g(n) \Rightarrow 0 \leq f_1(n) + f_2(n) \leq (c_1 + c_2) g(n)$

Let $(c_1 + c_2) = c_3$ so that $0 \leq f_1(n) + f_2(n) \leq c_3 g(n)$ for all $n \geq n_2$

And let $n_2 = max\{n_0, n_1\}$

Therefore, $f_1(n) + f_2(n) = O(g(n))$.

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

Counter Example to Disprove Conjecture:
Let $f(n) = n$, $g_1(n) = n^2$, and $g_2(n) = n^3$, which satisfies both $f(n) = O(g_1(n))$ and $f(n) = O(g_2(n))$ since $n = O(n^2))$ and $n = O(n^3)$ is true. However, it does not satisfy $g_1(n) = \Theta g_2(n) \Rightarrow n^2 \neq \Theta(n^3)$. By definition of $f(n) = \Theta g(n) \Leftrightarrow f(n) = \Omega g(n)$ and $f(n) = \mathcal{O} g(n)$, but $n^2 \neq \Omega(n^3)$. Therefore, $g_1(n) \neq \Theta g_2(n)$ and the conjecture is false.

---

4) Copy of code has been submitted to TEACH.

---

5) Merge Sort vs Insertion Sort Running Time Analysis

a) Merge Sort & Insertion Sort Running Time Analysis Code in Appendix A

b) Sorting Algorithm Collected Data

integers(n)	Merge Sort time(secs)	Insertion Sort(secs)
100,000	0.02465	11.9353
200,000	0.04860	47.9447
300,000	0.07096	113.571
400,000	0.10124	202.278
500,000	0.12075	328.134
600,000	0.15641	479.399
700,000	0.17983	650.790
800,000	0.20299	769.885
900,000	0.22435	996.210
1,000,000	0.25619	1323.55

Merge & Insertion Sort Average Execution Times
Graphs 1, 2, & 4

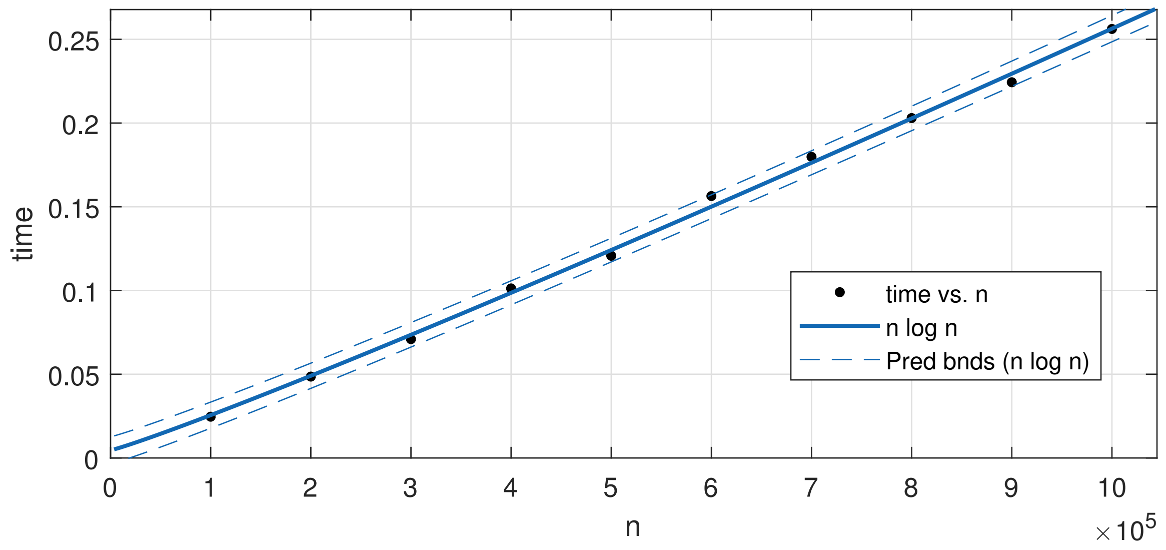
integers(n)	time (seconds)
10,000	0.12288
20,000	0.48330
30,000	1.08657
40,000	1.9673
50,000	3.10141
60,000	4.27698
70,000	5.77235
80,000	7.61551
90,000	9.46633
100,000	11.9692

Insertion Sort Execution Times
Graph 3

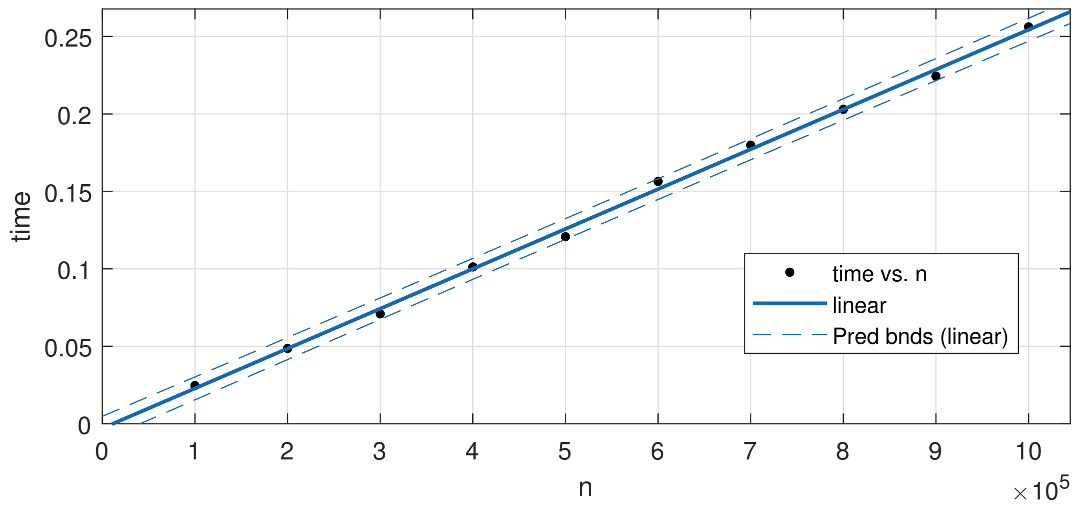
c-e) Sorting Algorithm Graphs

The individual graphs for the sorting algorithms are shown below. Additionally, the last graph is a combined graph showing data from both graphs. This last graph, however, does not show the data very well as the execution times for the two sorting algorithms differ by a large margin. As a result, the merge sort data is very hard to see. The combined graph does help to show the speed of the merge sort algorithm when compared to insertion sort algorithm. The individual graphs in turn clearly display the nature of the growth rates of each algorithm.

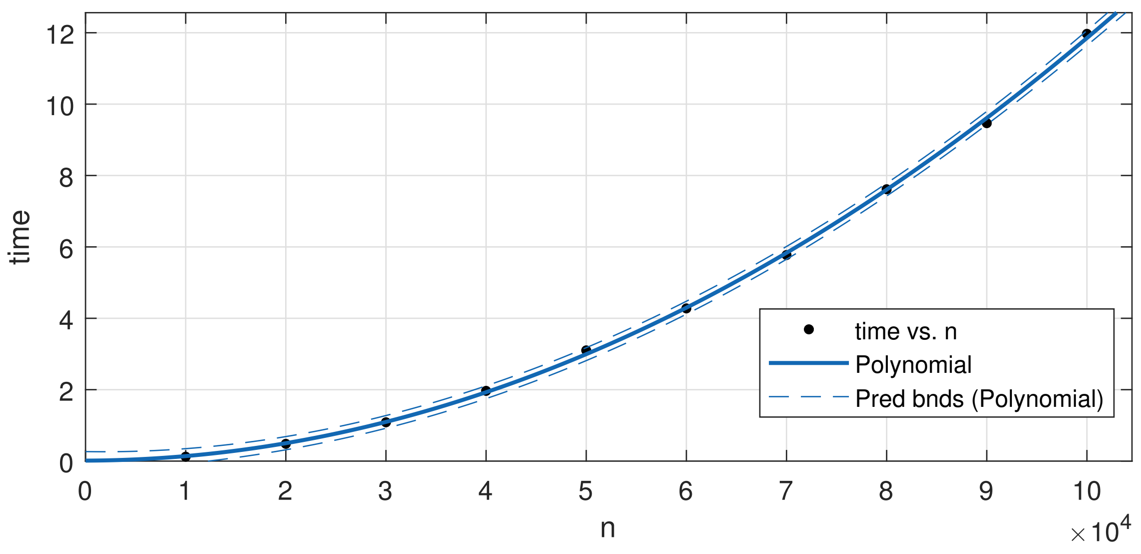
The experimental data that I collected for these algorithms matches their theoretical running times. The Merge Sort function time has a theoretical average running time of $\Theta(n(logn))$, which as shown in the graph below matches up snugly with the data I gathered. Likewise, The Insertion Sort function has a theoretical average running time of Θn^2 which matches up again with the data I gathered and is displayed in the graph below.



Merge Sort (n log n)
f(x) = a*x*log(x)+b
R-square: 0.9981



Merge Sort (linear)
f(x) = p1*x + p2
R-square: 0.9983



Insertion Sort (polynomial)

f(x) = p1*x$^2 + p2 * x + p3$

R-square: 0.9996

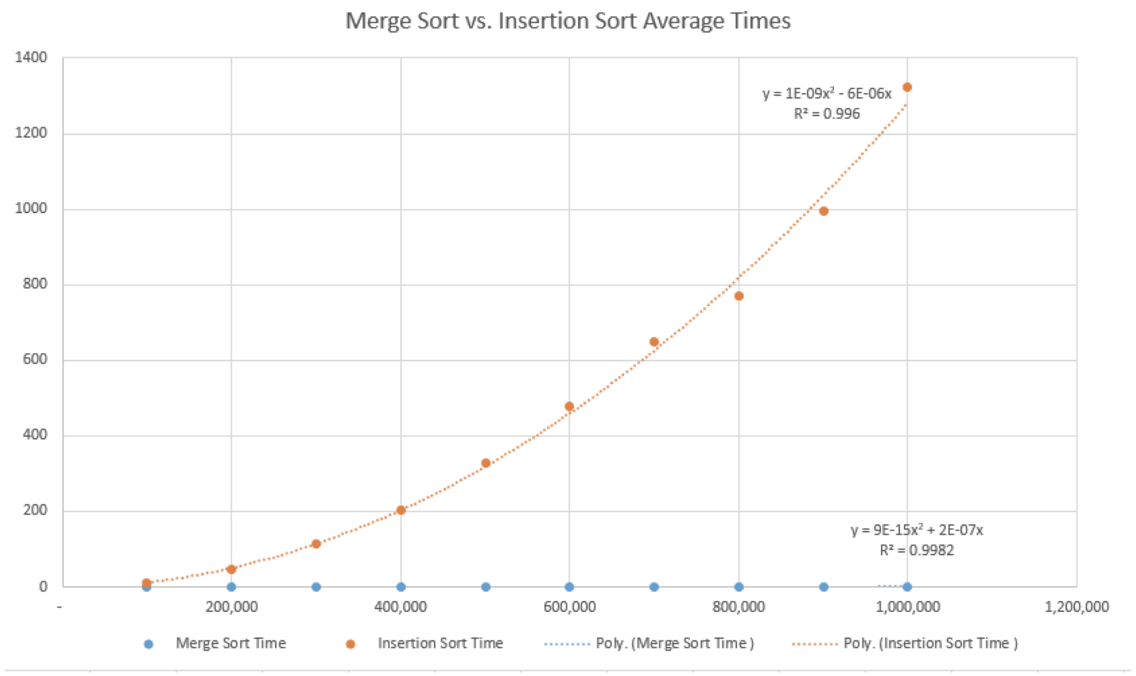


Figure 1: Insertion Sort vs. Merge Sort

Appendix A: Modified Code for Testing Sorting Algorithm Times :

```
//Sort algorithms from submitted code were used without any modifications.
void insertSort(int array[], int size);
void merge(int a[], int lowIndex1, int highIndex1, int lowIndex2, int highIndex2);
void mergeSort(int array[], int lowIndex, int highIndex);

int main(int argc, const char * argv[]) {

    std::ofstream outfile("mergeTimes.txt");
    //std::ofstream outfile("sortTimes.txt");
    const int arrSize = 1000002;
    int array[arrSize];

    for (int k = 0; k <= 1000000; k=k+100000){

        float total = 0, average = 0;

        outfile << k << ", " ;
        std::cout << k << ", ";


        for (int i = 1; i <= 3; i++){

            //Create array with random ints
            srand(time_t(NULL));

            for (int j = 0; j < k; j++){
                array[j] = rand() % 1000;
            }

            clock_t t1, t2;
            t1 = clock();

            mergeSort(array, 0, k-1);
            //insertSort(array, k);

            t2 = clock();

            float diff ((float)t2 - (float)t1);     //Total time
            float seconds = diff/CLOCKS_PER_SEC;
            total = total + seconds;
            outfile << seconds << ", ";
            std::cout << seconds << ", ";
        }
        average = total / 3;
        outfile << average << std::endl;
        std::cout << average << std::endl;
    }

    outfile.close();
}
```