Arthur Liou
CS325

HW 1

1) (4 points) Give the asymptotic bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible and justify your answers.

a) $T(n) = 3T(n - 1) + 1$

Can't use Master method so substitution, like week2Practice.pdf - Question 2b.

Substitution unto itself

3(3T(n-2) + 1) + 1
= 9T(n-2) + 6
= 9(3T(n-3) + 1) + 6
= 27T(n-3) + 15)
1st = 9T(n-2) + 6
2nd = 27T(n-3) + 15)
1, 6, 15 pattern -> 4n + 1

- Thus, we now have 3^n + (4n + 1) as our bounds
- **T(n) = Θ(3^n + (4n + 1))**

b) $T(n) = 2T\ (\ n/4\ ) + nlgn$;

a = 2
b = 4
f(n) = nlgn
n ^ logb(a) = n^log4(2) = n ^ 0.5
n ^ 0.5 vs nlogn -> nlogn is faster.

By Case 3 of the Master Method, we have **T(n) = Θ(n lg n).**

2) (6 points) The ternary search algorithm is a modification of the binary search algorithm that splits the input not into two sets of almost-equal sizes, but into three sets of sizes approximately one third.

a) Verbally describe and write pseudo-code for the ternary search algorithm.

```
// Declare function
int ternary_search(int l,int r, int x)
{
//sdefine base case
   if(r>=l)
   {
//split into three parts via two midpoints
       int mid1 = l + (r-l)/3;
       int mid2 = r -  (r-l)/3;
//base case check during split
       if(ar[mid1] == x)
          return mid1;
       if(ar[mid2] == x)
          return mid2;
//3 different cases depending on whether x is lies before, between, or after the midpoints
       if(x<ar[mid1])
          return ternary_search(l,mid1-1,x);
       else if(x>ar[mid2])
          return ternary_search(mid2+1,r,x);
       else
          return ternary_search(mid1+1,mid2-1,x);
   }
}
```

b) Give the recurrence for the ternary search algorithm
Recurrence relation: **T(n)=T(2n/3)+c**

c) Solve the recurrence to determine the asymptotic running time of the algorithm. How does the running time of the ternary search algorithm compare to that of the binary search algorithm.
T(n)∈Θ(log n)

Ternary Search Run Time: O(log3 N), where n is the size of the array
Binary Search Run Time: O(log n)

Ternary search does more comparisons than binary search in worst case.

3) (6 points) Design and analyze a divide and conquer algorithm that determines the minimum and maximum value in an unsorted list (array).

a) Verbally describe and write pseudo-code for the min_and_max algorithm.
//Define algorithm

```
//define function
MaxMin(a/arr, i, j, max, min)
{
//if array is size of one
    if (i=j) then max := min := a[i];
//else array is size of 2
    else if (i=j-1) then
        {
            if (a[i] < a[j]) then max := a[j]; min := a[i];
            else max := a[i]; min := a[j];
        }
//else start divide and conquer (ie size > 2)
    else
    {
//find midpoint to split the array
        mid := ( i + j )/2;
//recursively call on each split array
        MaxMin( i, mid, max, min );
```

```
            MaxMin( mid+1, j, max1, min1 );
// Combine the solutions to find the min and max
            if (max < max1) then max := max1;
            if (min > min1) then min := min1;
      }
}
```

b) Give the recurrence.

T(n) = 2T(n/2) + 2

a = 2

b = 2

f(n) = 2

c) Solve the recurrence to determine the asymptotic running time of the algorithm. How does the theoretical running time of the recursive min_and_max algorithm compare to that of an iterative algorithm for finding the minimum and maximum values of an array.

O(n) = 3n/2 – 2

Recursive: O(n)

Iterative: O(n)

Using the recursive min_and_max algorithm, we can decrease the number of comparisons, but both are O(n) run time.

http://somnathkayal.blogspot.com/2012/08/finding-maximum-and-minimum-using.html

4) (4 points) Consider the following pseudocode for a sorting algorithm.

```
StoogeSort(A[0 ... n - 1])
if n = 2 and A[0] > A[1]
swap A[0] and A[1]
else if n > 2
m = ceiling(2n/3)
StoogeSort(A[0 ... m - 1])
StoogeSort(A[n - m ... n - 1])
Stoogesort(A[0 ... m - 1])
```

a) State a recurrence for the number of comparisons executed by STOOGESORT.

T(n) = 3T(2n/3) + O(1)

A = 3

B = 3/2

F(n) = O(1)

Master theorem – case 1.

T(n) = (n^log3/2 3)

b) Solve the recurrence to determine the asymptotic running time.

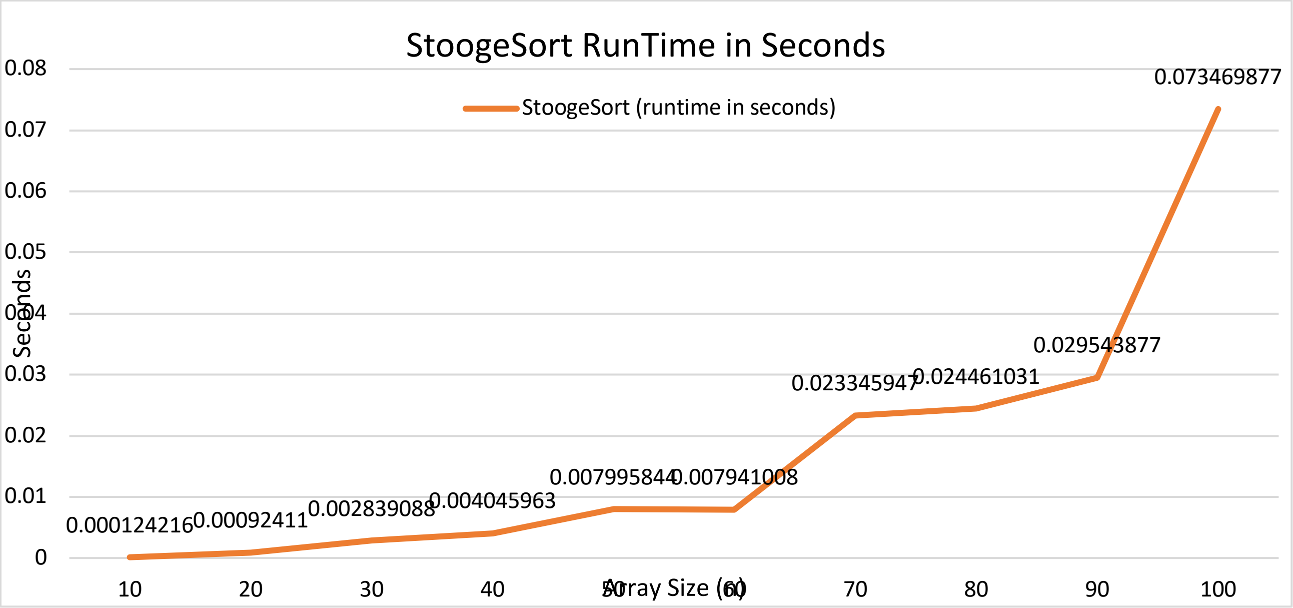
T(n) = (n^log3/2 3) **= O(n^2.71)**

Problem 5: (10 points)
a) Implement STOOGESORT from Problem 4 to sort an array/vector of integers. Implement the algorithm in the same language you used for the sorting algorithms in HW 1. Your program should be able to read inputs from a file called “data.txt” where the first value of each line is the number of integers that need to be sorted, followed by the integers (like in HW 1). The output will be written to a file called “stooge.out”.

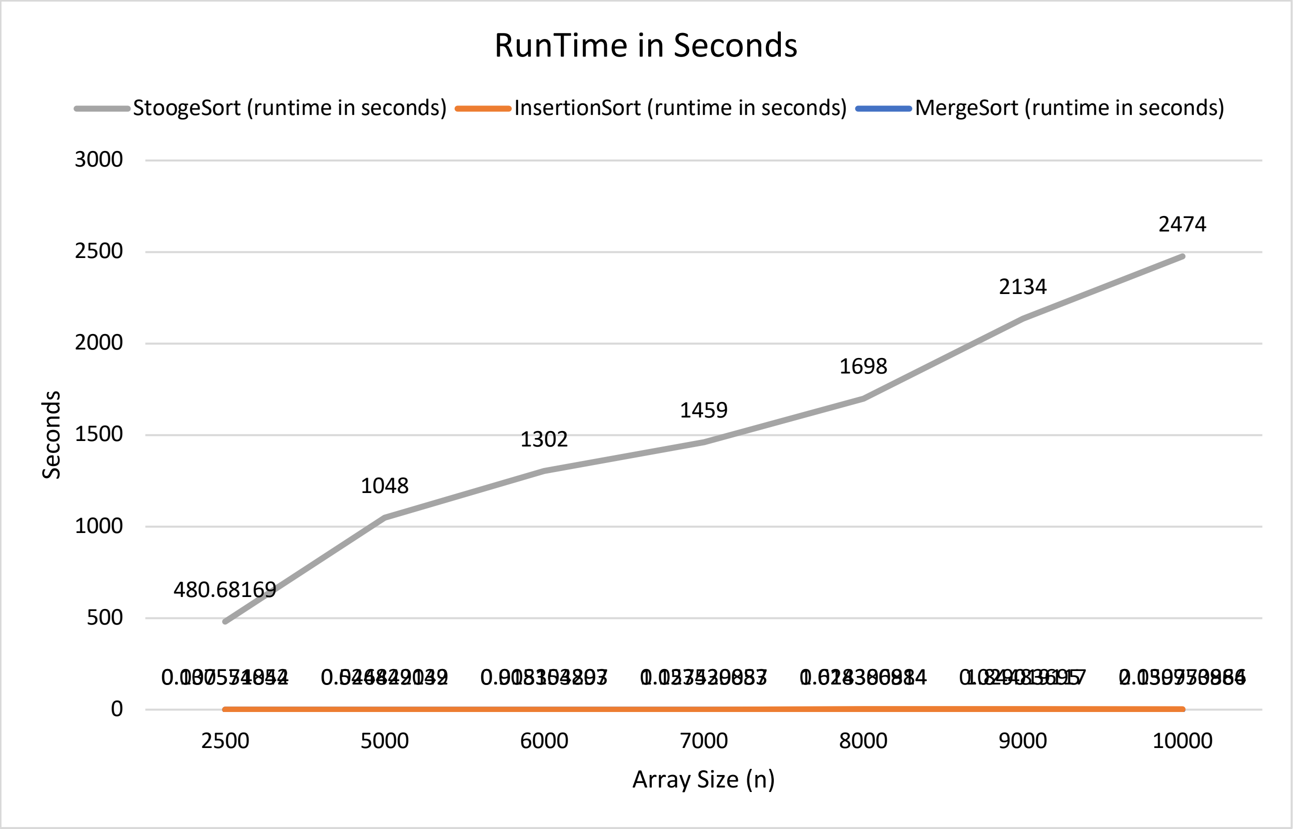
b) Now that you have proven that your code runs correctly using the data.txt input file, you can modify the code to collect running time data. Instead of reading arrays from a file to sort, you will now generate arrays of size n containing random integer values and then time how long it takes to sort the arrays. We will not be executing the code that generates the running time data so it does not have to be submitted to TEACH or even execute on flip. Include a “text” copy of the modified code in the written HW submitted in Canvas. You will need at least seven values of t (time) greater than 0. If there is variability in the times between runs of the algorithm you may want to take the average time of several runs for each value of n. Make a table containing the data you collected.

Array Size	StoogeSort (runtime in seconds)
10	0.00012421607971
20	0.000924110412598
30	0.00283908843994
40	0.0040459632873
50	0.00799584388733
60	0.00794100761414
70	0.023345947265
80	0.0244610309601
90	0.0295438766479
100	0.073469877243

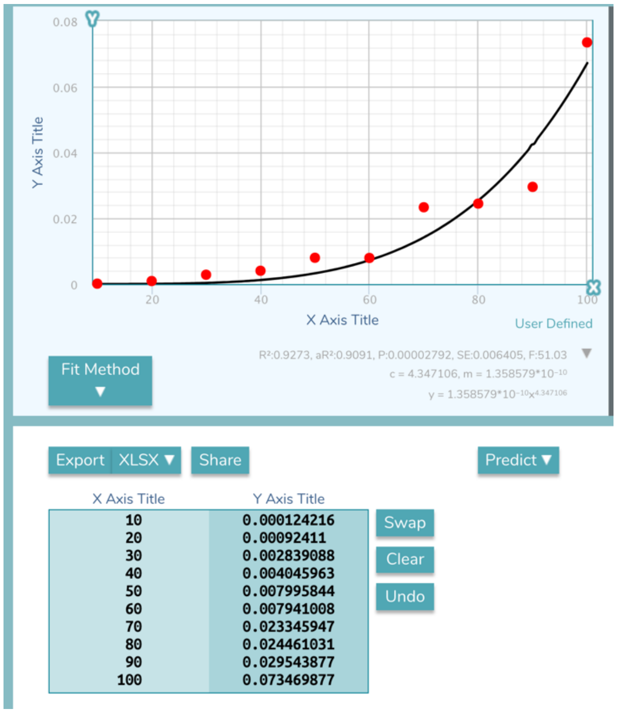
c) Plot the data you collected on an individual graph with n on the x-axis and time on the y-axis. You may use Excel, Matlab, R or any other software. Also plot the data from Stooge algorithm together on a combined graph with your results for merge and insertion sort from HW1.



Added to HW1 Graphs



d) What type of curve best fits the StoogeSort data set? Give the equation of the curve that best "fits" the data and draw that curve on the graphs of created in part c). Compare your experimental running time to the theoretical running time of the algorithm?



R²:0.9273, aR²:0.9091, P:0.00002792, SE:0.006405, F:51.03

c = 4.347106, m = 1.358579*10^{-10}

y = 1.358579*10^{-10}x$^{4.347106}$

X Axis Title	Y Axis Title
10	0.000124216
20	0.00092411
30	0.002839088
40	0.004045963
50	0.007995844
60	0.007941008
70	0.023345947
80	0.024461031
90	0.029543877
100	0.073469877

Equation for Curve of Best Fit -> y = (1.36*10^-10) * x^4.347
Theoretical : O(n^2.71)

My experimental running time was larger than the theoretical.