Stephen Townsend
CS 325-400
10/8/17

Assignment 2

1) Give the asymptotic bounds for T(n) in each of the following recurrences. Make your bounds as tight as possible and justify your answers. Assume the base cases T(0)=1 and/or T(1) = 1.

a) $T(n) = 2T(n-2) + 1$

Muster Theorem

a = 2, b = 2, d = 0

a > 1 = $\Theta\left(\mathrm{n}^{d}\ \mathrm{a}^{\frac{n}{b}}\right) = \Theta\left(\mathrm{n}^{0}\ \mathrm{a}^{\frac{n}{2}}\right) = \ \Theta\ \left(1\ \mathrm{a}^{\frac{n}{2}}\right) = \ \ \Theta\left(\ \mathrm{a}^{\frac{n}{2}}\right)$

b) $T(n) = T(n-1) + n^3$

Muster Theorem

a = 1, b = 1, d = 3

a = 1 = $\Theta\left(\mathrm{n}^{d+1}\right) = \Theta(\mathrm{n}^{3+1}) = \ \Theta\ (\mathrm{n}^{4})$

c) $T(n) = 2T(\frac{n}{6}) + 2n^2$

Master Theorem

a = 2, b = 6, f(n) = $2n^2$

$\log_b a = \log_6 2$

$2n^2 > \ \log_6 2\ = \ \Theta(n^2)$

Stephen Townsend
CS 325-400
10/8/17

2) The quaternary search algorithm

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

The quaternary search algorithm should be used on a sorted list of integers. What is would do is take a list of arrays and determine a split value. This will divide the array into fourths to make every sub array approximately ¼ in size. It will then compare the element with the last value in the quadrant to determine if it is ≤. This will determine what quadrant it is in. If it is equal then you can return the value.

```
quaternarySearch (Arr[n .. n-1], low, high, search)
        firstFourth = ((high – low)/4)                          //1/4
        secondFourth = ((high – low)/2)                         //1/2
        thirdFourth = ((high – low) *(3/4))                     //3/4

if (search <= Arr[firstFourth])                                 //Check if in firstFourth
        if (search == Arr[firstFourth])
                return Arr[firstFourth]
        return quaternarySearch(Arr[], low, firstFourth-1, search)
else if (search <= Arr[secondFourth])                           //Check if in secondFourth
        if (search == Arr[secondFourth])
                return Arr[secondFourth]
        return quaternarySearch(Arr[], firstFourth+1, secondFourth-1, search)
else if (search <= Arr[thirdFourth])                            //Check if in thirdFourth
        if (search == Arr[thirdFourth])
                return Arr[thirdFourth]
        return quaternarySearch(Arr[], secondFourth+1, thirdFourth-1, search)
else
        return quaternarySearch(Arr[], thirdFourth+1, high, search)     //Check if in last
```

b) Give the recurrence for the quaternary search algorithm

$$T(n) = T\left(\frac{n}{4}\right) + 1$$

c) Solve the recurrence to determine the asymptotic running time of the algorithm. How does the running time of the quaternary search algorithm compare to that of the binary search algorithm.

a = 1, b = 4 f(n)= 0 (1^0)

$$\log_4 1 = 0$$

$$0 = 0 = \Theta(n^0 \log n) = \Theta(\log n)$$

Stephen Townsend
CS 325-400
10/8/17

3) 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.

For this type of algorithm, I would take a similar approach as merge sort. Split the array into half to split the work. Because this is an unsorted array we will have to hold to min a max of each subdivision. The first step is to split the array in half. Then split that subsection into half till you have two elements. The two elements are placed into a min and max variable. As the recursion goes up one level to check the next division we compare it against the elements in the min and max variable. If one of them is larger or lesser then the variable will equal that number. This will occur until the entire array has been searched through.

pseudocode

```
minMaxAlgo(arr[0..n], low, high, min, max)
if (n ==1)
        min = arr[0]
        max = arr[0]
else if (n==2)
        if arr[0] < arr[1]
                min = arr[0]
                max = arr[1]
        else
                min = arr[1]
                max = arr[0]
else if (low < high)
                split = (low + high) /2
                minMaxAlgo(arr[], low, split)
                minMaxAlgo(arr[], split+1, high)
  if (line[low] < line[high])
                if (arr[low] < min)
                        min = arr[low];
                if (arr[high] > max)
                        max = arr[high];
   else
                if (arr low] > max)
                        max = arr[low];
                if (arr[high] < min)
                        min = arr[high];
```

b) Give the recurrence.

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(1)$$

Stephen Townsend
CS 325-400
10/8/17

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.

a = 2, b = 2 f(n)= 1

$1 = \log_2 2 = \Theta(n \log n)$

4) Stooge Sort

a) Verbally describe how the STOOGESORT algorithm sorts its input.

Stooge sort works its way down to two elements. Once there it will swap the position of those two elements if the first is greater than the second. If the current size is greater than 2 then the program will calculate and integer m by taking the ceiling of 2/3 of the length. Once m is calculated there are three different recursive calls.

The first recursion will pass and sort the first 2/3 of the array. The second recursive call will pass in the last 2/3 of the array and sort that. The third and final recursive call will pass int the first 2/3 of the array again. This is to double check that it is sorted. Once both arrays are sorted, the third recursive call will sort the first 2/3 array again.

b) Would STOOGESORT still sort correctly if we replaced k = ceiling(2n/3) with k = floor(2n/3)? If yes prove if no give a counterexample. (Hint: what happens when n = 4?)

No, using k = floor(2n/3) will not work. For a counter example I will use the array [15, 6, 5, 20].

- if n = 2. Currently n = 4. This is false. We will move on.
- else if n > 2. Currently n = 4. This is true.
- m = floor($\frac{2*4}{3}$). m = 2.
- The first recursion will pass in [15, 6]. We go back to the beginning where if n = 2 and if Arr[0] > Arr[1] we swap elements. Currently n = 2 and 15 > 6. This is true. Arr[0] and Arr[1] swap = [6, 15]. Our Array is now [6, 15, 5, 20].
- The second recursion will pass in the remaining array [5, 20]. We go back to the beginning where if n = 2 and if Arr[0] > Arr[1] we swap elements. Currently n = 2 and 2 > 20. This is false.
- The third recursion will pass in the begging of the array [6, 15]. We go back to the beginning where if n = 2 and if Arr[0] > Arr[1] we swap elements. Currently n = 2 and 6 > 15. This is false. We finished the sort.

If we look at the array it went from [15, 6, 5, 20] to [6, 15, 2, 20]. The new array is not sorted and this means that using the floor will not work.

Stephen Townsend
CS 325-400
10/8/17

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

$$T(n) = 3T\left(\frac{2n}{3}\right) + 1$$

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

a = 3, b = 3/2, f(n) = 1

$$\log_{3/2} 3 = \frac{\log 3}{\log(\frac{3}{2})} = 2.70$$

1 < 2.70 = $\Theta(n^{\log_{\frac{3}{2}} 3})$

= $\Theta(n^{2.70})$

5)

b) Include a “text” copy of the modified code

```
//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 stoogeSort(int *arr, 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;
```

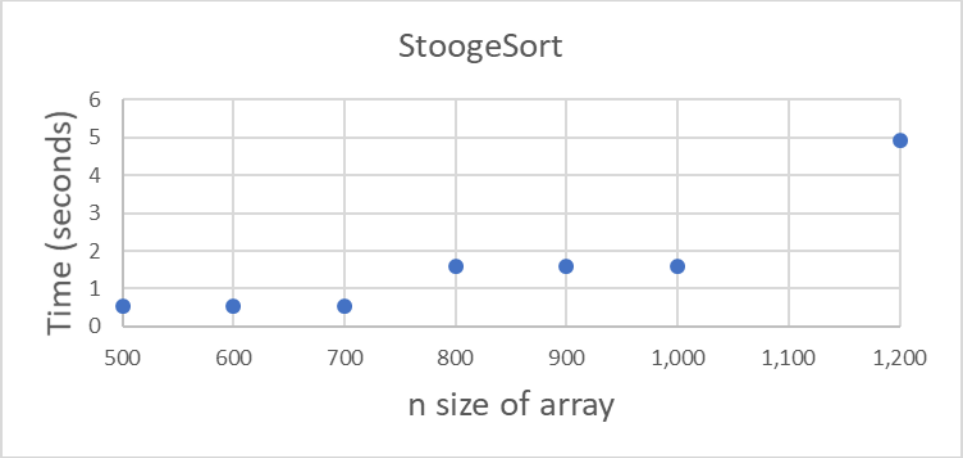
```
	while (runTime == true)
	{
		printf("How large is array? ");
		scanf("%d", &size);

		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

		stoogeSort(line, 0, (size - 1));		//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");
}
/****************************************
*			stoogeSort
* This function uses
****************************************/
void stoogeSort(int *line, int low, int high)
{
	int temp = 0;
	int m = 0;

	if (high == 2 || line[low] > line[high])
	{
		temp = line[low];
		line[low] = line[high];
		line[high] = temp;
	}
	if ((high - low + 1) > 2)
	{
		m = ((high - low + 1) / 3);
		stoogeSort(line, low, (high - m));
		stoogeSort(line, (low + m), high);
		stoogeSort(line, low, (high - m));
	}

	/*
	Source:
	Implementation based on:
	Homework pseudocode and https://en.wikipedia.org/wiki/Stooge_sort
	*/
}
```

Stephen Townsend
CS 325-400
10/8/17

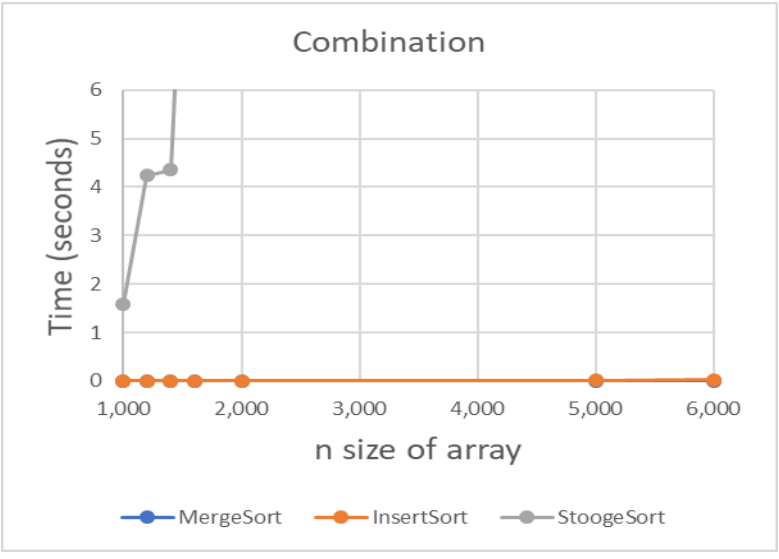
```
/*****************************************
*                    randomGenerator
* This function is used to generate a random
* number so that the program can use a random
* room durring the connections process.
*****************************************/
void fillArrayGenerator(int *line, int size)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

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

c) Plot the running time 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.

StoogeSort	
n	Time
500	0.524
600	0.528
700	0.536
800	1.584
900	1.586
1,000	1.586
1,200	4.24



Merge Sort			
n	MergeSort	InsertSort	StoogeSort
1,000	0.001	0	1.586
1,200	0.001	0.003	4.24
1,400	0.001	0.003	4.354
1,600	0.001	0.003	13.13
2,000	0.002	0.003	13.279
5,000	0.005	0.016	115.445
6,000	0.006	0.022	356.341



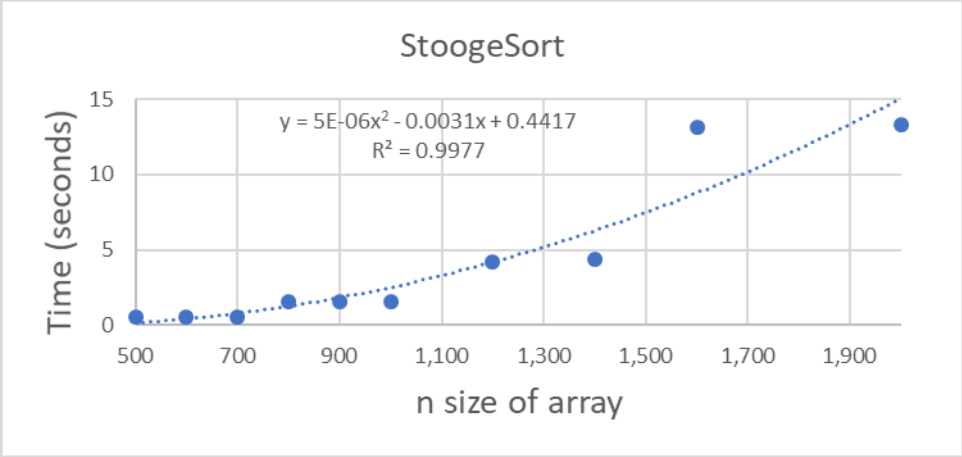
Stephen Townsend
CS 325-400
10/8/17

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).

For Stooge sort, I used a 2nd degree polynomial to get an R2 of 0.9977.

- The equation for the merge sort is:

  $y = 5E\text{-}06x^2 - 0.0031x + 0.4417$



My theoretical running time for stooge sort is $\Theta(n^{2.70})$. With this and the actual running time, from the function above, it seems like my results are in line with the theoretical running time.