HW2 Justin Phillips
spring 2019

Problem 1:(8points)
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) $T(n) = 3T(n-1)+1$
   muster method: a=3, b=1, f(n) = 1, d= 0
   since a >1 $T(n)=\Theta(n^0 3^{n/1}) = \Theta(3^n)$

b) $T(n)=T(n-2)+2$
   muster method: a=1,b=2, f(n)=2, d=0
   since a =1 $T(n)= \Theta(n^{d+1})= \Theta(n)$

c) $T(n)= 9T(n/2)+6n^2$
   master method: a=9, b=2->$n^{\log_2 9}=n^{3.169925001442312}$,$f(n)=6n^2$
   f(n) is dominate therefore we use case 3 $f(n) = \Omega(6n^{2+e})$ for e =1 and $9(cn/2)^2 <= cn^2$ for c=1/3

   $T(n) = \Theta(6n^2)$

d) $T(n)=2T(n/4)+2n^2$
   master method: a=2, b =4, $n^{\log_4 2}=n^{(1/2)}$, $f(n)=2n^2$
   f(n) is dominate therefore we use case 3 $f(n) = \Omega(2n^{2+e})$ for e =1 and $2(cn/2)^2 <= cn^2$ for c=1
   $T(n) = \Theta(2n^2)$

Problem 2:(7points)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.

Like the binary search we need to divide the sets but instead of divide and conquer stragedy of 2 we do 3. so we need to get 2 spots to be able to compare our target to. Depending on the comparison of the target we move to the correct recursion call. then rinse and repeat till the target is found and if it is not then we print no found and return

function ternary(int arr[],target number,low,high)

middle1 = set1 low – set1 high /3
middle2 = set2 low – set2 high /3

```
if(target == arr[middle1]
        return middle1;
if(target == arr[middle2]
        return middle2;

if(target < arr[middle1])
        return ternary(arr,low,middle1-1,target)
else if (target > arr[middle2])
        return ternary(arr,middle2+1,high, target)
else
        “entry not found”
        return -1
```

b)Give the recurrence for the ternary search algorithm

T(n) = T(n/3)+c

master method: a =1, b= 3,n $\log_3^1$= n^0=1, f(n)=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) = T(n/3)+c

master method: a =1, b= 3,n $\log_3^1$= n^0=1, f(n)=c

compare 1 to f(n) which are 2 constants so we have case 2: if(f(n)= $\Theta(n \log_b^a)$,

then T(n)= Θ (n $\log_b^a$ lgn) so f(n) = $\Theta(1)$ ->T(n)= Θ(lgn)

The running time of ternary and binary are both case 2 with a T(n)= Θ(lgn). It is basically doing the same thing except in ternary we have one more divide to do but we are still comparing our target to a value till we find out if our target is in the array.

Problem 3:(5points) 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(3n/2)+c

b)Solve the recurrence to determine the asymptotic running time

master method: a =3, b=3/2, n $\log_3^{(3/2)}$= $n^{2.71}$, f(n) = c

case 1 because $n^{2.71} > c$ so $f(n) = O(n^{2.71-e})$ for e=1 therfore $T(n) = \Theta(n^{2.71})$

5b)
```
#include <iostream>
#include <fstream>
#include <ctime>
#include <time.h>
#include <cstdlib>

void stoogesort(int arr[], int low, int high) {
	if (low >= high) {
		std::cout << "low>=high " << std::endl;
		return;
	}

	if (arr[low] > arr[high - 1]) {
		int temp = arr[low];
		arr[low] = arr[high - 1];
		arr[high - 1] = temp;
		//swap(arr[low], arr[high]);
	}

	if (high - low + 1 > 2) {
		int t = (high - low + 1) / 3;
		stoogesort(arr, low, high - t);
		stoogesort(arr, low + t, high);
		stoogesort(arr, low, high - t);
	}
}
void arrayFill(int myArray[], int size) {
	//int newArr[10];
	//int n = 5000;

	for (int i = 0; i <= size; i++) {
		myArray[i] = rand() % 10000 + 1;
		std::cout << myArray[i] << std::endl;
	}
}

void swap(int& i, int& j) {
	int t = i;
	i = j;
	j = t;
}
void printArr(std::ostream &stream, int arr[], int n) {

	stream << std::endl << std::endl;

	for (int i = 0; i < (n - 1); i++) {
		stream << arr[i] << " ";
	}

	stream << arr[n - 1];
	//stream << std::endl;
}

int main() {
```

```
	std::ofstream timeFile("stoogeTime.out");
	srand(time(NULL));
	clock_t t1, t2;
	//int *myArray = new int[num]; //dynamically allocate an array
	int i = 0;
	for (int i = 0; i < 10; i++) {
		int n = rand() % 5000 + 1; //size of array
		float timer = 0;
		float diff = 0;
		float seconds = 0;
		int myArray[n];
		arrayFill(myArray, n);
		t1 = clock();
		stoogesort(myArray,0, n-1);
		t2 = clock();
		diff = t2 - t1;
		std::cout << "Array Size " << n << std::endl;
		std::cout << "difference " << diff << std::endl;
		seconds = diff / CLOCKS_PER_SEC;
		std::cout << "Seconds " << seconds << std::endl;
		timeFile << n << "size " << seconds << "seconds. " << std::endl;
	}
	timeFile.close();
	return 0;

}
```

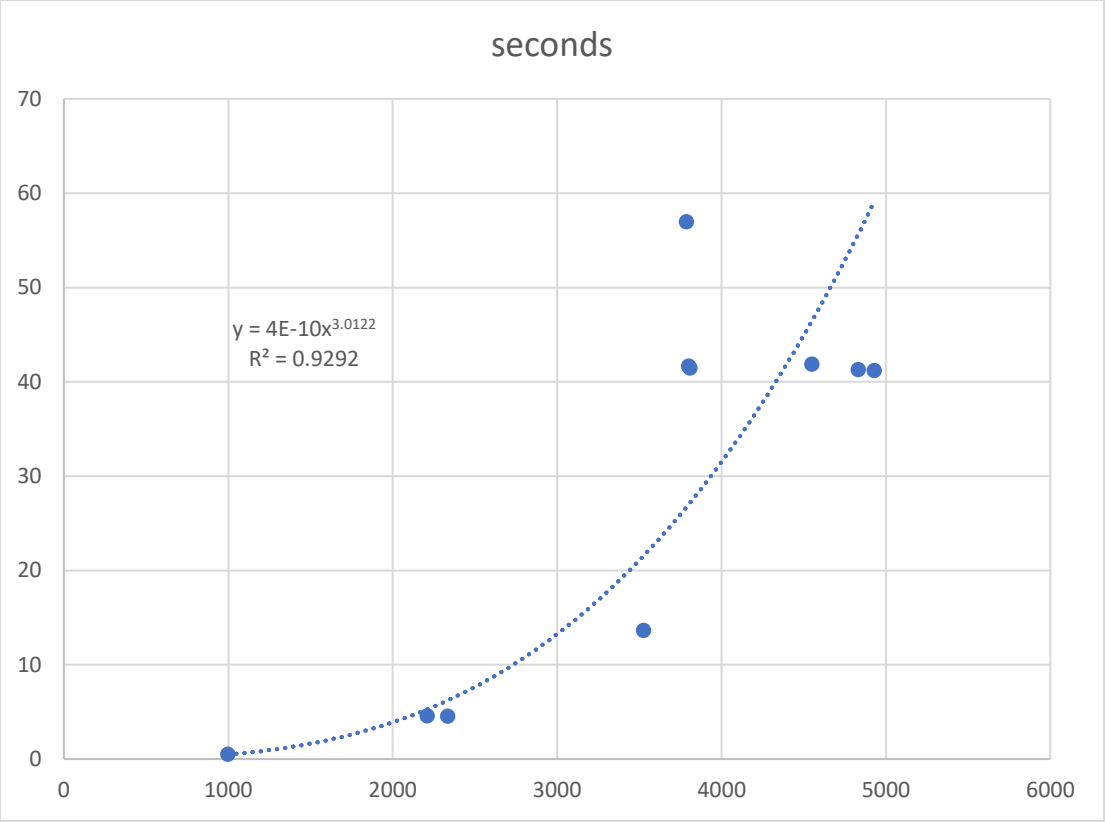
5c) Collect running times-Collect your timing data on the engineering server. You will need at least eight values of t (time) greater than 0. If there is variability in the times between runs of the same algorithm you may want to take the average time of several runs for each value of n. Create a table of running times for the algorithm.

stooge sort

size	seconds
4550	41.89
998	0.51
4832	41.29
2209	4.57
3786	56.98
3800	41.66
4928	41.2
3807	41.47
2334	4.55
3525	13.65

d)Plot data and fit a curve-Plot the running time data you collected on an individual graph with non the x-axis and time on the y-axis. You may use Excel, Matlab, R or any other software. What type of curve best fits the data set? Give the equation of the curve that best “fits” the data and draw that curve on the graph.

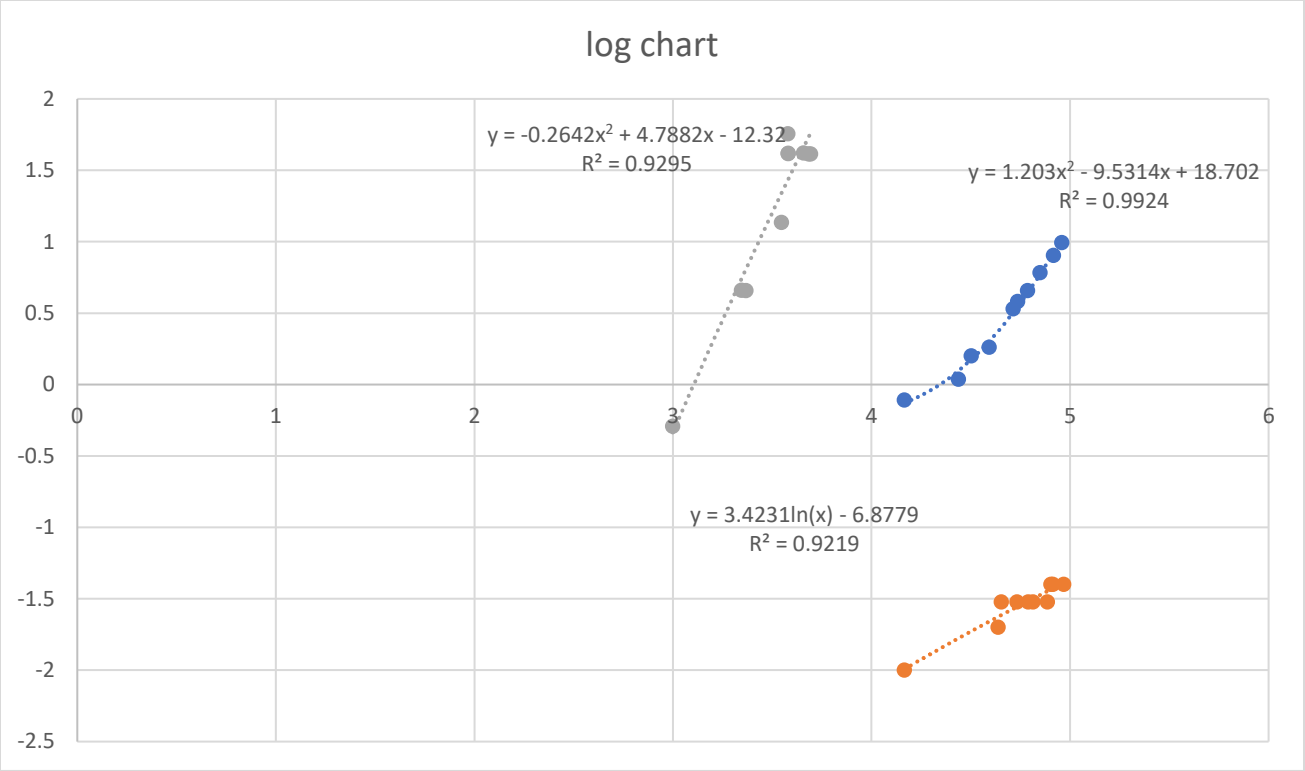
based on the graph I determined that the power curve best fits data.
the equation is $y=4E\text{-}10x^{3.0122}$



e)Comparison-Compare your experimental running times to the theoretical running time of the algorithm? Remember, the experimental running times were the “average case” since the input arrays contained random integers.

T(n) = $\Theta(n^{2.71})$ is the theoretical run time I determined in problem 3 and the from the data I collected my equation is $y=4E\text{-}10x^{3.0122}$. Which I can express as $n^{3.0122}$. It is not exactly the same but is close.

f) Combine -Plot the data from Stooge sort with the data from merge sort and insertion sort. If the scales are different you may want to use a log-log plot



stooge = silver

insert = blue

merge = orange