Trevor Stahl

CS 325

Homework Assignment 2

01/2019

Problem 1

Give the asymptotic bounds for T(n) in the following recurrence. Make your bounds as tight as possible and justify your answers.

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

Base Case:

T(0) = 1

Recursive:

T(1) = 3(T(0)) + 1 = 3(1) + 1 = 4

T(2) = 3(T(1)) + 1 = 3(4) +1 = 13

T(3) = 3(T(2)) + 1 = 3((3((3(1)+1))+1)) +1 = 40

3^n lower bound

3^n + 3^n-1 + 3^n-2 … + 3^0

3^n * (1 + 3^-1 + 3^-2 + …)

3^n * C2 upper bound

Thus T(n) = O(3^n) and Ω(3^n) which means Θ(3^n)

Problem 2

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.

This is a search algorithm which works on sorted arrays. It finds the desired value by recursively dividing the array into approximate thirds. Checking which of the thirds the target is in and breaking that third into sub thirds.

```
Function Ternary_search(A, n, T){
    Int L = 0
    Int R = n – 1
    While L <= R:
        Int upM = (((2)(R-L))/3) + L   /* 2/3 of n, integer rounding */
        Int loM = ((R-L)/3)  + L   /* 1/3 of n, division without remainder to get apx third*/
        If A[upM] == T:
            Return upM
        Else if A[loM] == T:
            Return loM
        Else if A[loM] > T:
            R = loM - 1
        Else if A[upM] < T:
            L = upM + 1

        Else:
            L = loM + 1
            R = upM - 1

    Return unsuccessful
}
```

b) Give the recurrence for the ternary search algorithm

$T(n) = T(n/3) + 1$

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.

$T(1) = 1$

$T(3) = 2$

$T(9) = T(9/3) + 1 = T(3) + 1 = 2 + 1 = 3$

$T(27) = T(27/3) + 1 = T(9) + 1 = 3 + 1 = 4$

$T(n) = T(n/3) + 1$

$T(n) = \text{Log}_3 n + 1 = O(\text{Log}_3 n)$

Only big O because of various early exits when checking if mid points are target.

Binary would be log base 2 instead of 3

All logs are in the same Big Oh class so ternary has same asymptotic performance as binary.
In practice, binary has smaller constants.

Problem 3

Design and analyze a divide and conquer algorithm that determines the minimum and maximum value in an unsorted list array.

a)

Algorithm breaks initial input into halves until they are of size 1. Then recombines these such that it maintains the min and max up through recursion.

```
function getMinMax (array)
{
    If array.size == 1
        Return (array[0], array[0]);
    Else:
        First_min, first_max = getMinMax(first half of array)
        second_min, second_max = getMinMax(second half of array)
        Return (min(first_min, second_min), max(first_max, second_max))
}
```

b)

$f(n) = 2(f(n/2)) + 1$

c)

$f(1) = 1$

$f(2) = 2(f(1)) + 1 = 2 + 1 = 3$

$f(4) = 2(f(2)) + 1 = 6 + 1 = 7$

$F(8) = 2(f(4)) + 1 = 15$

$F(n) = 2n - 1 = O(n)$

They are the same. Both O(n)

Problem 4

a)

```
function 4merge(first, second, third, fourth)
{
        Var result = empty list

        While any array has an element
                {
                        Find smallest value of all arrays
                        Add to end of result
                        Remove from sub array
                }
        Return result
}
function 4mergesort(one array)
{
        Splits array into 4 smaller arrays, or less if less than 4
        Calls itself on smaller arrays
        Return 4merge(4 smaller arrays)
}
```

b)

$f(n) = 4(f(n/4)) + n$

$f(1) = 1$

$f(4) = 4(f(1)) + 4 = 4 + 4 = 8$

$f(16) = 4(f(4)) + 16 = 8 + 16 = 24$

$f(64) = 4(f(16)) + 64 = 96 + 64 = 160$

$f(n) = O(n \log n)$

Problem 5

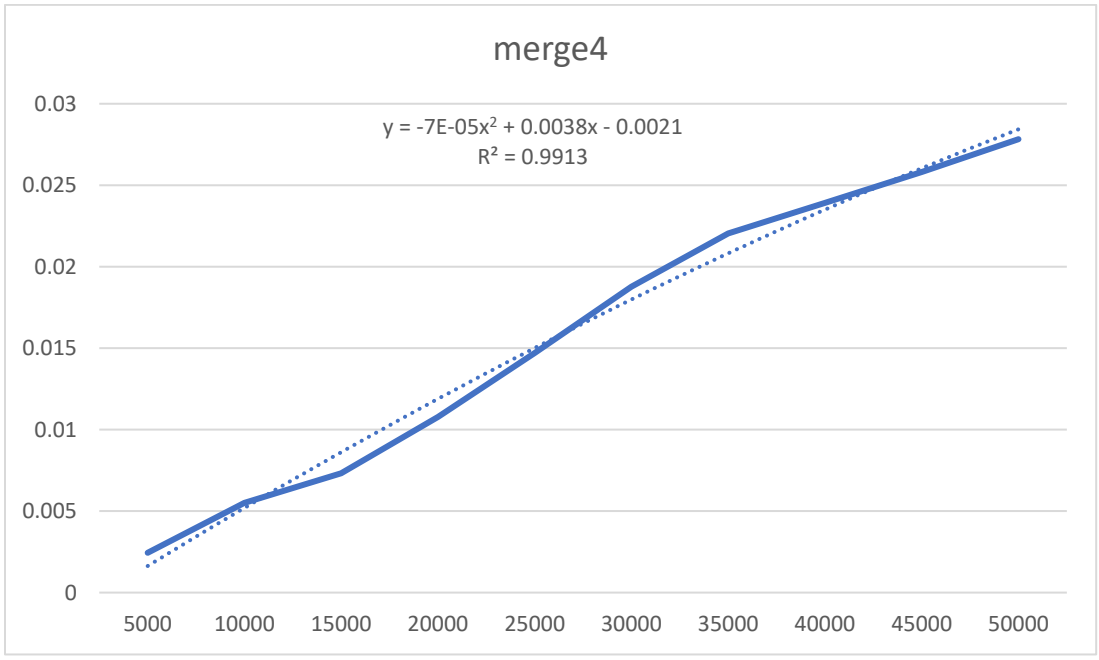
a) File sumbited to teach in zip file

b) File submitted to teach with readme file in a zip with part a)

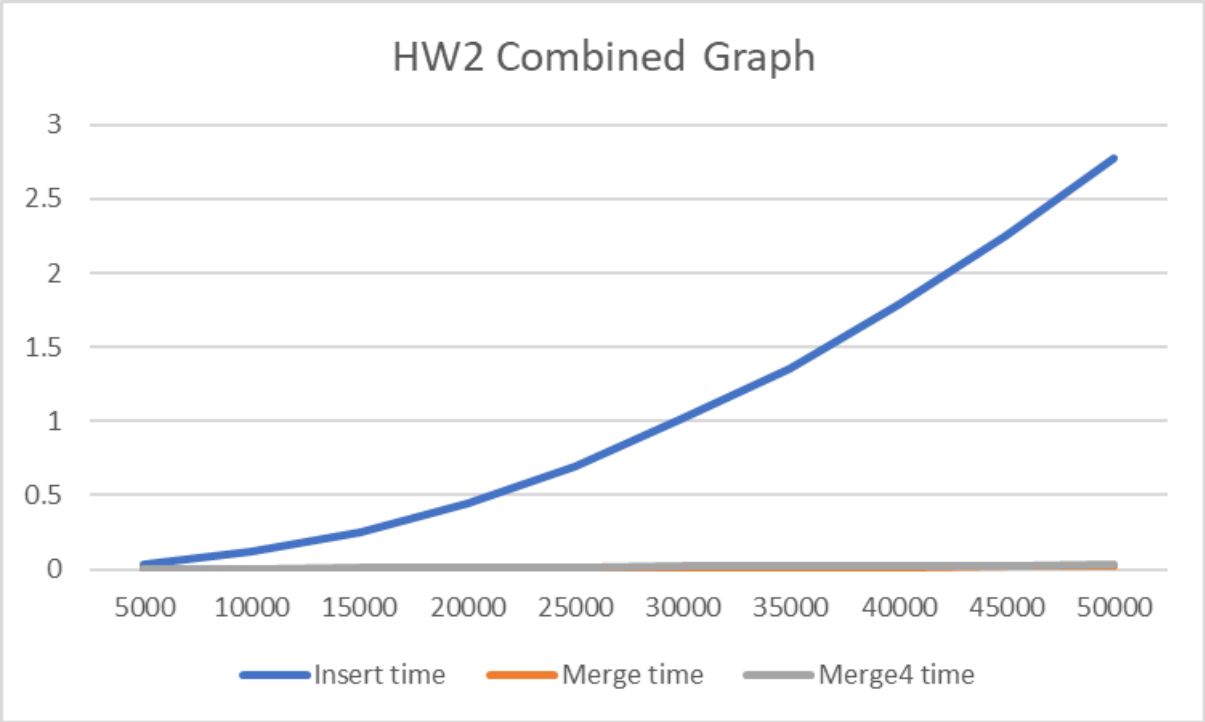
c)

5000	0.00244252
10000	0.0055051
15000	0.00732677
20000	0.0107831
25000	0.0147037
30000	0.0187687
35000	0.0220423
40000	0.0239127
45000	0.0258088
50000	0.027825

d)



e)



Merge and merge4 are overlapping.

Txt file of modified running time file:

```
#include <fstream>
#include <iostream>
#include <chrono>

// C program for insertion sort
#include <stdio.h>
#include <math.h>


using std::chrono::system_clock;

void merge(int arr[], int l, int m1, int m2, int m3, int r)
{
	int i1, i2, i3, i4, k;
	int n1 = m1 - l + 1;
	int n2 = m2 - m1;
	int n3 = m3 - m2;
	int n4 = r - m3;

	/* create temp arrays */
	int* A1 = new int[n1];
	int* A2 = new int[n2];
	int* A3 = new int[n3];
	int* A4 = new int[n4];
```

```
	/* Copy data to temp arrays */
	for (i1 = 0; i1 < n1; i1++)
		A1[i1] = arr[l + i1];
	for (i2 = 0; i2 < n2; i2++)
		A2[i2] = arr[m1 + 1 + i2];
	for (i3 = 0; i3 < n3; i3++)
		A3[i3] = arr[m2 + 1 + i3];
	for (i4 = 0; i4 < n4; i4++)
		A4[i4] = arr[m3 + 1 + i4];

	/* Merge the temp arrays back into arr[l..r]*/
	i1 = 0; // Initial index of first subarray
	i2 = 0;
	i3 = 0;
	i4 = 0;
	k = l; // Initial index of merged subarray

	while (true)
	{
		int min = 0;
		bool found_any = false;
		int* imin = NULL;
		if ((i1 != n1) && (!found_any || A1[i1] < min)) {
			found_any = true;
			min = A1[i1];
			imin = &i1;
		}

		if ((i2 != n2) && (!found_any || A2[i2] < min)) {
			found_any = true;
			min = A2[i2];
			imin = &i2;
		}

		if ((i3 != n3) && (!found_any || A3[i3] < min)) {
			found_any = true;
			min = A3[i3];
			imin = &i3;
		}

		if ((i4 != n4) && (!found_any || A4[i4] < min)) {
			found_any = true;
			min = A4[i4];
			imin = &i4;
		}

		if (!found_any) {
			return;
		}
		arr[k] = min;
		(*imin)++;
		k++;
	}
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergesort(int arr[], int l, int r)
```

```
{
	if (l < r)
	{
		// Same as (l+r)/2, but avoids overflow for
		// large l and h
		int m = l + (r - l) / 2;
		int m1, m2, m3;
		m1 = l + (m - l) / 2;
		m2 = m;
		m3 = m + (r - m) / 2;

		// Sort first and second halves
		mergesort(arr, l, m1);
		mergesort(arr, m1 + 1, m2);
		mergesort(arr, m2 + 1, m3);
		mergesort(arr, m3 + 1, r);

		merge(arr, l, m1, m2, m3, r);
	}
}

// A utility function to print an array of size n
void printArray(std::ostream &stream, int arr[], int n)
{
	for (int i = 0; i < (n - 1); i++)
	{
		stream << arr[i] << " ";
	}
	stream << arr[n - 1];
	stream << std::endl;

}



int main()
{
	srand(time(NULL));


	for (size_t size = 5000; size <= 50000; size = size + 5000)
	{
		int* array = new int[size];

		for (size_t j = 0; j < size; j++)
		{
			array[j] = rand() % 10001;
		}

		system_clock::time_point start = system_clock::now();
		mergesort(array, 0, size - 1);
		system_clock::time_point end = system_clock::now();

		std::chrono::duration<double> duration = end - start;

		std::cout << size << " " << duration.count() << std::endl;
	}
```

```
std::cin.get();
}
```