Thomas Haddy
Com S 311
Section 19
TA: Trent Muhr
3/21/18

Homework 4

1.

a) T(n) = 3T(n/2) + n; T(1) = 1

T(n) = 3[3T(n/4) + n/2] + n
T(n) = 9[3T(n/8) + n/4 + n/2] + n
T(n) = $3^k T(1) + n[1 + \frac{3}{2} + \frac{3^2}{2} + \cdots + \left(\frac{3}{2}\right)^{k-1}]$
Choose k = $\log_2 n$
T(n) = $3^{\log_2 n}(1) + n[1 + \frac{3}{2} + \frac{3^2}{2} + \cdots + \left(\frac{3}{2}\right)^{\log_2(n)-1}]$
T(n) = $n^{\log_2 3} + n[\frac{1-(3/2)^{\log_2 n}}{1-(\frac{3}{2})}] < 2n[\left(\frac{3}{2}\right)^{\log_2 n} - 1]$
T(n) < $n^{\log_2 3} + 2n(n)^{\log_2(\frac{3}{2})}]$
T(n) = $n^{\log_2 3} + 2n(n)^{\log_2(3)-1}]$
T(n) = (1 + 2) $n^{\log_2 3}$
T(n) = $n^{\log_2 3} \in O(n^{1.58\ldots})$

b) T(n) = T(n/8) + n; T(1) = 1

T(n) = T(n/64) + n + n/8
T(n) = T(n/512) + n + n/8 + n/64
T(n) = T(n/8^k) + n[1 + $(1/8)^1$ + $(1/8)^2$ + … + $(\frac{1}{8})^{k-1}$]
Choose k = $\log_8 n$
T(n) = T(n/$8^{\log_8 n}$) + n[1 + $(1/8)^1$ + $(1/8)^2$ + … + $(\frac{1}{8})^{\log_8 n-1}$]
T(n) = T(n/$8^{\log_8 n}$) + n[$\sum_{i=0}^{\log_8(n)-1}(1/8)^i$]
T(n) = T(1) + n[$\sum_{i=0}^{\log_8(n)-1}(1/8)^i$]
T(n) = 1 + n($\log_8 n$)
$\in O(n \log_8 n)$

2. Assume r is the root of the BST being constructed from the sorted int array arr. If you want to access the BST, call this method with following: root r = sortedArrToBST(arr)

```
sortedArrToBST(int[] arr) {
        if (arr equals null)
                return null
        r = middle element of arr
        r.left = sortedArrToBST(new array: arr[0] to arr[middle element-1])
        r.right = sortedArrToBST (new array: arr[middle element+1] to arr[length-1])
        return r
}
```

So, this algorithm goes first sets the root for the BST. Then it recursively makes the left subtree of r and continues to make subtrees. Next it moves to the right side of the arr and recursively makes right subtrees of root r. Lastly, it returns the root which contains the contents of arr but as a BST.

Recurrence: $T(n) = 2T(n/2) + O(1); T(1) = 1$

$T(n) = 2[2T(n/4)] + 1$

$T(n) = 4[2T(n/8)] + 2$

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + k$

Choose $k = \log_2(n)$ $so\ T\left(\frac{n}{2^{\log_2(n)}}\right) = T\left(\frac{n}{n}\right) = T(1) = 1$

Note: $2^{\log_2 n} = n$

$T(n) = 2^{\log_2 n}(1) + \log_2 n$

$T(n) = n + \log_2 n \in O(n)$

3.

```
mergeSets(LinkedList l1, LinkedList l2) {
        let mergedList be an empty LinkedList
        let cur1 be the head Node of l1
        let cur2 be the head Node of l2
        while (cur1 != tail of l1 and cur2 != tail of l2)
                let Point tmp be the smaller x value between cur1 and cur2
                if (mergedList is empty or y value of tmp is less than mergedList's tail)
                        add tmp to the end of mergedList
                update smaller x value Node to equal the next Node in that LinkedList
                if (either cur1 or cur2 equals a tail)
                        while (the other cur does not equal its tail)
                                add other cur to LinkedList merged
                                other cur = next Node in other LinkedList
}

FindNewSetS(LinkedList s) {
        if (s's size equals 1)
                return LinkedList containing s's single element
```

let l1 be a LinkedList with s's first half of elements
let l2 be a LinkedList with s's second half of elements
let result be a LinkedList with mergeSets(l1, l2)
return result
}

Recurrence: $T(n) = 2T(n/2) + O(n)$; $T(1) = 1$

$T(n) = 2[2T(n/4)] + cn(1+1)$

$T(n) = 4[2T(n/8)] + cn(1+1+1)$

$T(n) = 2^k T\left(\frac{n}{2^k}\right) + cn(k)$

Choose $k = \log_2(n)$ $so\ T\left(\frac{n}{2^{\log_2(n)}}\right) = T\left(\frac{n}{n}\right) = T(1) = 1$

Note: $2^{\log_2 n} = n$

$T(n) = 2^{\log_2 n}(1) + cn(\log_2 n)$

$T(n) = n + cn \log_2 n \in O(n \log_2 n)$

4.

```
Graph2(Graph g) {
    for each vertex v in g's V
        let adj_v be the current vertex v's adjacency list
            for each vertex u in adj_v
                for each vertex x in u's adjacency list
                    if (x is not in v)
                        add w to a new adjacency list result

    return result
}
```

My algorithm will go through each vertex in graph G and create a new adjacency list of length 2 between *u* and *v* in graph g.

Runtime: *m* is number of edges in G and *n* is number of vertices in G. In the worst case, the second for loop will run 2*m* times because it will look at every edge twice.

$$\sum_{i=1}^{n} \sum_{j=1}^{2m} 1 = \sum_{i=1}^{n} 2m = 2m \sum_{i=1}^{n} 1 = 2\text{m(n)} \in O(\text{mn})$$