Homework 6 COM S 311

Alec Meyer

October 24, 2020

Question 1

a.
$T(n) = T(\frac{n}{2}) + cn^2$
$T(n) = T(\frac{n}{4}) + c(\frac{n}{2})^2 + cn^2$
$T(n) = T(\frac{n}{4}) + c(\frac{n}{2})^2 + cn^2$
$T(n) = T(\frac{n}{2^k}) + c(\frac{n}{(2^k)-1})^2 + ...cn^2$
$n/(2^k) = 1$
$k = logn$
$T(n) = cn^2(\sum_{k=1}^{n}(\frac{1}{(2^{k-1})^2})$
Runtime $= O(n^2)$

b.
$T(n)$
$T(\frac{n}{2}) - -T(n-1)$
$c + c = 2c$
...
$k = logn$ – $2^k c$
$T(n) = \sum_{k=0}^{n} 2^k c$
Runtime $= O(n)$

c.
$T(n) = 3T(\frac{n}{3}) + cn$
$T(n) = 3[3T(\frac{n}{3^2}) + c(\frac{n}{3^1})] + cn$
$T(n) = 3^k T(\frac{n}{3^k}) + \sum_{k=1}^{n} \frac{n}{3^k} = 1$
$n = 3^k$
$k = logn$
$T(n) = kcn$
$T(n) = cnlogn$
Runtime $= O(nlogn)$

Question 2

If we have two differing MSTs T and S with edge weight contained in only T and S. The edge Te only appears in T. If this is the case then $S \cup Te$

contains a cycle. We can call an edge of this cycle Se which is not in T. Se is a distinct edge from Te which is contained in S or T its weight must be less than Se. This would mean the weight of T is less than S which is a contradiction since it was assumed S and T were MSTs.

Question 3

Psuedo code:
1. initialize N and an array of houses houseArray
2. sort houses in descending order with largest distance from east coming first
3. left, right = 0, 0;
4. Traverse array of houses
5. if r ¡ houseArray[i] then build a tower
6. if we are at extreme east or west add a tower

Proof:
since we traverse each house in the array we are going a greedy route for the algorithm. Therefore we will be able to test each location and decide whether or not there needs to be a tower. This through results in a runtime of O(n) since we have to traverse the entire array.

Question 4

```
Test(Cards C)
        n = C.length;
        if(n == 1)
                return C;
        if(n == 2)
                if (C[0] && C[1])
                        reutn C[0];
        CLeft = left half of C;
        CRight = right half of C;
        if(Test(CLeft) != NULL)
                check returned card against rest of the array;
        if(Test(CRight) != NULL)
                check returned card against rest of the array;
```

```
return card if more than n/2 cards equivalent;
```

We use the recursive function twice per call. We then divide each array by 2 resulting in an O(nlogn) runtime