ComS 311
Recitation 3, 2:00 Monday
Homework 3

Sean Gordon

September 30, 2019

1a) $T(n) \leq 3T(\frac{n}{2}) + Cn^2, \quad T(2) \leq c$

$3[3T(\frac{n}{4}) + c(\frac{n}{2})^2] + cn^2$
$9T(\frac{n}{4}) + 3c(\frac{n}{2})^2 + cn^2$

$9[3T(\frac{n}{8}) + c(\frac{n}{4})^2] + 3c(\frac{n}{2})^2 + cn^2$
$27T(\frac{n}{8}) + 9c(\frac{n}{4})^2 + 3c(\frac{n}{2})^2 + cn^2$

$27[3T(\frac{n}{16}) + c(\frac{n}{8})^2] + 9c(\frac{n}{4})^2 + 3c(\frac{n}{2})^2 + cn^2$
$81T(\frac{n}{16}) + 27c(\frac{n}{8})^2 + 9c(\frac{n}{4})^2 + 3c(\frac{n}{2})^2 + cn^2$

$3^4T(\frac{n}{2^4}) + 3^3c(\frac{n}{2^3})^2 + 3^2c(\frac{n}{2^2})^2 + 3^1c(\frac{n}{2^1})^2 + 3^0c(\frac{n}{2^0})^2$
$3^4T(\frac{n}{2^4}) + \frac{3^3}{2^{3*2}}cn^2 + \frac{3^2}{2^{2*2}}cn^2 + \frac{3^1}{2^{1*2}}cn^2 + \frac{3^0}{2^{0*2}}cn^2$

Final term is $3^kT(\frac{n}{2^k})$. Assuming that $n/2^k = 2$ so that T(2) = c
$\frac{n}{2^k} = 2 \quad \Rightarrow \quad n = 2^k * 2 \quad \Rightarrow \quad n = 2^{k+1} \quad \Rightarrow$
$log(n) = k + 1 \quad \Rightarrow \ k = log(n) - 1$
$\therefore$ end term == $3^{log(n)-1} * c$

Full: $3^{log(n)-1} * c + cn^2 \sum_{k=0}^{log(n)-1} (\frac{3}{2^2})^k$
However, $\lim_{k\to\infty} (\frac{3}{2^2})^k = \frac{1}{1-3/4}$
$\Rightarrow c * 3^{log(n)-1} + cn^2 \frac{1}{1-3/4}$

1b) $T(n) \leq 2T(\frac{n}{2}) + Cnlog(n), \quad T(2) \leq c$

$2[2T(\frac{n}{4}) + c(\frac{n}{2})log(\frac{n}{2})] + cnlog(n)$
$4T(\frac{n}{4}) + 2c(\frac{n}{2})log(\frac{n}{2}) + cnlog(n)$

$4[2T(\frac{n}{8}) + c(\frac{n}{4})log(\frac{n}{4})] + 2c(\frac{n}{2})log(\frac{n}{2}) + cnlog(n)$
$8T(\frac{n}{8}) + 4c(\frac{n}{4})log(\frac{n}{4}) + 2c(\frac{n}{2})log(\frac{n}{2}) + cnlog(n)$

$8[2T(\frac{n}{16}) + c(\frac{n}{8})log(\frac{n}{8})] + 4c(\frac{n}{4})log(\frac{n}{4}) + 2c(\frac{n}{2})log(\frac{n}{2}) + cnlog(n)$
$16T(\frac{n}{16}) + 8c(\frac{n}{8})log(\frac{n}{8}) + 4c(\frac{n}{4})log(\frac{n}{4}) + 2c(\frac{n}{2})log(\frac{n}{2}) + cnlog(n)$

End term: $2^kT(\frac{n}{2^k})$
Assuming $\frac{n}{2^k} = 2$ so that $T(\frac{n}{2^k}) = T(2) = c$

$\frac{n}{2^k} = 2 \quad \Rightarrow \quad n = 2^k * 2 = 2^{k+1} \quad \Rightarrow$
$log(n) = k + 1 \quad \Rightarrow \quad k = log(n) - 1$

Full: $2^{log(n)-1} * c + cn\sum_{k=0}^{log(n)-1} log(\frac{n}{2^k}) \Rightarrow$
$2^{log(n)-1} * c + cn\sum_{k=0}^{log(n)-1} log(n) + cn\sum_{k=0}^{log(n)-1} log(2^k)$
$2^{log(n)-1} * c + cnlog(n) + cn\sum_{k=0}^{log(n)-1} k$

However, $\sum_{k=0}^{log(n)-1} k = \frac{(log(n))(log(n)+1)}{2} = \frac{2log(n)+log(n)}{2}$

$\therefore$ Full: $c * 2^{log(n)-1} + cn\frac{2log(n)+log(n)}{2} \Rightarrow$

2) Given a k-sorted arrayList...

```
kSort(int k, List<> kSorted){

    //Make two temporary arrays with their own ks...
    int k1 = k-(k/2)
    int k2 = k/2

    List<> arr1 = new ArrayList();
    List<> arr2 = new ArrayList();


    // Split the kSorted array into two
    // {1, 4, 7, 3, 5,  3, 9, 12, 6, 19,  7, 83, 14}   k=5
    // ==>
    // {1, 4, 7,        3, 9, 12,         7, 83, 14}   k=3
    // {         3, 5,            6, 19             }   k=2

    // ==>>
    // {1, 4, 7, 3, 9, 12, 7, 83, 14} k=3
    // ==>
    // {1, 4,    3, 9,     7, 83    } k=2
    // {      7,       12,        14} k=1

    // etc...


    //Copy the data to the temp arrays...

    int i, j;
    int x = 0, y = 0;
    for (i = 0; x < kSorted.size(); i++){
        x = i + k2*(i/k1);

        arr1.add(kSorted.get(x));
    }
    for (j = 0; y < kSorted.size(); j++){
        y = i + k1*(i/k2) + k1;

        arr2.add(kSorted.get(y));
    }


    //If k1 > 1, call again
    if(k1 > 1)
        arr1 = kSort(k1, arr1);

    //If k2 > 1 call again
    if(k2 > 1)
        arr2 = kSort(k2, arr2);


    //Merge the two arrays in a merge-sort fashion,
    //and return the result
    return merge(arr1, arr2);

}
```

This algorithm runs in some amount of time, of which I have none of to calculate.

3)

```
Given an array of points...
Declare an empty array of purple points.

Sort the array by the y values of each point.

Inside the funct(arr){
    Split the array at the midpoint into two arrays, smaller and larger.

    Find the largest x-value of the larger array.
    Compare this value to each value in the smaller array, adding each smaller
    point to the array of purple values.

    Recursively call this function with the larger array.
}
```

As we divide the array in half each time, there are log(n) recurrence levels. Therefore time is $cn * \sum_{k=0}^{log(n)} (1/2)^k \Rightarrow O(n)$
Multiplied by the sort in the beginning, the algorithm takes O(nlog(n) time.