Neha Maddali

Problem 1:

a. Using the master theorem case 1, if f(n) = $O(n^{\log 4(5) - \epsilon})$ for some constant $\epsilon > 0$, then T(n) = $\Theta(n^{\log b(a)})$ = Θ(n^1.161)
b. Using the master theorem case 3, if f(n) = $\Omega(n^{\log 2(16) + \epsilon})$ for some constant $\epsilon > 0$, and where c = 0.75, 16*f(n/2) ≤ 0.75*f(n) satisfies the regularity condition for some constant c < 1 and all sufficiently large n, then T(n) = Θ(f(n)) = Θ(n^4)
c. Using the master theorem case 3, if f(n) = $\Omega(n^{\log 3(9)) + \epsilon})$ for some constant $\epsilon > 0$, and where c = 0.75, 9*f(n/3) ≤ 0.75*f(n) satisfies the regularity condition for some constant c < 1 and all sufficiently large n, then T(n) = Θ(f(n)) = $\Theta(n^2(\log n)^2)$
   Recursion tree proves this as well.
   $n^2 \log n$ —> $n^2 \log n$
   (n/3)^2log(n/3)----> ≤ $n^2 \log n$
   T(n) = $\Theta(n^2(\log n)^2)$

Problem 2:

a. T(n) = 3T(n/3)+c
b. Using master theorem case 1, if f(n) = $O(n^{\log 3(3) - \epsilon})$ for some constant $\epsilon > 0$, then T(n) = $\Theta(n^{\log b(a)})$ = Θ(n)
c. T(n) = 3T(n/4) + c
d. Using master theorem case 1, if f(n) = $O(n^{\log 4(3) - \epsilon})$ for some constant $\epsilon > 0$, then T(n) = $\Theta(n^{\log b(a)})$ = Θ(n^0.79248)

Problem 3:

```
Search(first, last, A[ ])
        first = A[0]
        last = A[n]
        mid = (first+last)/2
        if(first > last)
                return false
        if(A[mid] == mid)
                return true
        else if (A[mid] < mid)
                A[ ] = A[first, … mid-1]
                return Search(first, mid-1, A[ ])
        else
                A[ ] = A[mid+1, … last]
                return Search(mid+1, last, A[ ])
```

Correctness of Algorithm: consider any array A having n integers. Array A is sorted from low to high. Let first = A[0] and last = A[n].

- For the base case, if the first element is greater than the last element, we have reached the last element without finding A[i] = i, so return 0.

- Now our middle element in the array A ‘mid’ = (first+last)/2. If the middle element is equal to i then A[i] = i is found and returns mid.
- Otherwise, if the middle element is less than mid, the element A[i] has to be somewhere to the left of A[mid]. So the first element remains ‘first’ and the last element is A[mid-1]. Now call Search(first, mid-1, A[])
- Else, if the middle element is greater than ‘mid’ the element A[i] has to be to the right of A[mid]. Now the first element is A[mid+1] and the last element is ‘last’. Now call Search (mid+1, last, A[])

Running time: This algorithm running time is O(log n)

Problem 4:

a. By trying to loop through each value of I and I^0 and increasing the counter each time the values of I and I^0 are equal to each other. Then, the number of comparisons that are being counted in I*I^0
   $\Sigma^{n}_{i=0}$ 1 ($\Sigma^{n}_{j=0}$ 1+c+c) = n(n)
   Therefore, the runtime is O(n^2)
b. The algorithm provided makes comparisons between the left and right sides of the array after splitting the array. However, the answer is not correct because it does not compare the overlapping of the left and right sides.
c. First, given two intervals [a,b] and [c,d], compute the length of their overlap like so
   overlap([a,b],[c,d])
   ```
   if b < c then return 0
   else
       if b ≤ d then return b – c + 1
       if b > d then return d – c + 1
   ```
   Then, to find the greatest overlap of two intervals such that they come from different sets sorted by starting point …
   overlapbetween(([[a_1, b_1], … [a_x, b_x]], [[c_1, d_1], … [c_k, d_k]]
   ```
   if k==0 or x == 0 then return 0
   minc = c_1
   maxb = 0
   olap = 0
   for i from 1 to x
       if maxb < b_i
           maxb = b_i
   for j from 1 to k
       if olap < overlap([minc,maxb],[c_k, d_k])
           olap = overlap([minc,maxb],[c_k, d_k])
   return olap
   ```
   Finally, given a list of intervals, find the length of the greatest overlap between two intervals.
   overlapAlgo (sort([a_1, b_1], … [a_n, b_n]))
   ```
   if n==1 then return 0
   mid = Ln/2⌋
   ```

LS = [[a_1, b_1], … [a_{mid}, b_{mid}]]
RS = [[a_{mid+1}, b_{mid+1}], … [a_n, b_n]]
olap1 = overlapAlgo(LS)
olap2 = overlapAlgo(RS)
olap3 = overlapbetween(LS,RS)
return max(olap1,olap2,olap3)

d. T(n) = 2T(n/2)+n
e. Θ(nlogn) using the master theorem case 3.