1 Homework 4

Name: Kangdong Yuan

1.1 problem1

a).I did not work in a group.
b).I did not consult without anyone my group members
c).I did not consult any non-class materials.

1.2 problem2

1). First, divide the array into two same length array A1 and A2, then we get the majority of the sub-lists l and r. if l or r is the majority element of whole list, return l or r. And if the l and r is same, this element is the majority element of whole list. And we do this divide and combine process recursively to find whether the array has majority element.

The algorithm in pseudo-code.

```
program majority element:
Input: array A[1...n]
Output: Majority element of array
if n=None:
        return None
if n=1:
        return A[1]
half = n // 2
left = majority(A[1...half+1])
right = majority(A[half+1...n])
if left == right:
        return left
if A.count(left)>half:
        return left
if A.count(right)>half:
        return right
return None
```

Count() return the number of specified element in given array.
b). If A has a majority element, the majority element m must also be a majority element of A1 or A2 or both. If the count of m is not majority in each sub array, it must not be the majority in total array. If both parts have the same majority element, it is the majority element for total array. If one

of the sub array has a majority element, count occurrence of combined array (O(n) time) to see whether it is a majority element. If both parts have a majority, the algorithm check those two elements and return the result (O(n) running times). And this divide algorithm perform recursively, the running time in each level is O(n), and the recursive tree has nlog(n) levels.
c). this algorithm is every function call two another functions, and the time complexity in each level is O(n), because the count() function go through every element in array.
$T(n) = 2T(n/2) + n$, using the master theorem, $a = 2$, $,b = 2$, $c = 1$, $log_a(b) = log_2(2) = 1 = c$
The time complexity is $T(n) = O(n^{log_a(b)} * log(n)) = O(n * log(n))$

1.3 problem3

a). For graph stored in adjacency list format, if we want to return the Reverse graph G^R, we need to do below algorithm.

The algorithm in pseudo-code.
program Reverse graph:
Input: graph: G (adjacency list format)
Output: Reverse graph: G^R (adjacency list format)
create an empty adjacency list G^R
for each vertex u in V
    add the new vertex u to G^R
    for each edge (u, w) in E
        add new edge (w,u) to G^R
return changed graph G^R
When add a vertex or edge, it take O(1) time. the total number of add is constant, since there are only $|V| + |E|$ vertices and edges in graph G, so the running time is also $O(|V| + |E|)$

b)For graph stored in adjacency matrix format, if we want to return the Reverse graph G^R, we need to do below.
for the adjacency matrix format, the number of columns and rows are equal to number of vertices. We assume the number of vertices is $|V|$
The algorithm in pseudo-code.
program Reverse graph:
Input: graph: G (adjacency matrix format)
Output: Reverse graph: G^R (adjacency matrix format)
create a new matrix G^R with dimension $|V| * |V|$

for i in rows of G
    for j in columns of G
        if edge G[i,j] exist
            add edge to $G^R[j,i]$
return G^R

we assume deal with a condition and add a edge all take O(1) time. There are $|V|^2$ conditions and $|E|$ adds, so the running time of conditions and adds are $O(|V|^2 + |E|)$. And create a empty matrix take $O(|V|^2)$ times. Adding all step together, $O(2*|V|^2+|E|) = O(|V|^2)$, so, the running time is $O(|V|^2)$

1.4 problem4

a). the adjacency list for graph 1:
$A->\{B,E\}$ $B->\{G,D\}$ $D->\{E\}$ $E->\{D\}$ $G->\{D,F\}$
the adjacency list for graph 2:
$C->\{H,I\}$ $H->\{I\}$ $I->\{\}$
b). In undirected graph, each edge connect to two vertices and no parallel edge, and the number of self loop edge is $|v|$. The number of edges is the number of subsets of size 2 chosen from the set of vertices. Since the set of vertices has size $|V|$, the number of such subsets is given by the binomial coefficient $C(|V|,2) = \frac{|V|*(|V|-1)}{2}$. The possible max number of edge in undirected graph is $\frac{|V|*(|V|-1)}{2} + |V| = \frac{|V|^2+|V|}{2}$
c). for each undirected edge, it incident to two vertices, so the sum of d_i cause by one edge is 2. We assume there are n edge in graph, those edges incident to the vertices in graph. Since one edge only can incident to two vertices, the sum of d_i is $2n$. So, whatever the graph is, the sum of d_i must be the even number.
d).The number of edge in this graph is inherited the result in question b, the upper bound of $|E| = \frac{|V|^2+|V|}{2}$, and lower bound of $|E| < |V|$
if $|E| > |V|log(|V|)$, then the $\Theta(|V|^2log(|V|))$ run faster than $\Theta(|V||E|)$
if $|E| < |V|log(|V|)$, then the $\Theta(|V||E|)$ run faster than $\Theta(|V|^2log(|V|))$
e). for any undirected garph in question b, the upper bound of $|E| = \frac{|V|^2+|V|}{2}$, and lower bound of $|E| < |V|$.
So,if $|E| > |V|$, the $\Theta(|E|log(|V|))$ run faster $\Theta(|E|log(|E|))$.
But, if $|E| < |V|$, the $\Theta(|E|log(|E|))$ run faster $\Theta(|E|log(|V|))$.