This block of code is responsible for getting the data from the file, and it will take O(n*m*LOG(n)), where (n) is the number of the lines at the file, and (m) is the length of the ID

```
while(getline(myfile,s))                    // read information from file.
{
    int u = 0,v = 0, tmp = 0;
    for(int i = 0 ;i < s.size() ;++i)       // Iterating over the line readed ( O(s.size()) )
    {
        if(s[i] == ',')
        {
            u = tmp;                        // save first ID
            tmp = 0;
            continue;
        }
        tmp = (tmp * 10) + (s[i] - '0');
    }
    v = tmp;                                // save second ID
    follow[u].insert(v);                    // make directed edge between first ID and second ID ( O(1) )
    followed[v].insert(u);                  // reverse edge between first ID and second ID ( O(1) )
}
```

this block of code will cost O(followed.size())

```
for(auto it:followed)                           // iterate over reversed graph ( O(followed.size()) )
    p.push_back({it.second.size(),it.first});    // save the number of followers of each ID and ID
```

this sorting will cost O(p.size()*LOG(p.size))

```
sort(p.rbegin(),p.rend());                  // sort vector in Descending Order ( O(p.size() * LOG (p.size())) )
```

this block of code will take O(follow[id].size()*follow[it].size()*LOG(follow[it2].SIZE())

```
unordered_map<int,int> freq;
vector<pair<int,int>> mx_freq;
for(auto it:follow[id])                     // Iterating over the the IDs followed by input ID ( O(follow[id].size()) )
{
    for(auto it2:followed[it]){             // Iterating over the the IDs follow child of input ID ( O(follow[it].size()) )
        if(follow[it2].count(id) || it2 == id) continue;        // check if specific ID follow input ID O( LOG(follow[it2].SIZE()
        freq[it2]++;
    }
}
```

This for loop will take O(freq.size())

```
for(auto it:freq)                              // Iterating over mutual friends O(freq.size())
    mx_freq.push_back({it.second,it.first});
```

This sort function will cost O(mx_freq.size() * LOG (mx_freq.size()))

```
sort(mx_freq.rbegin(),mx_freq.rend());      // sort vector in Descending Order ( O(mx_freq.size()
```

So, we can say that the complexity of that code will be O(n*m*LOG(n)), where (n) is the number of the lines at the file, and (m) is the length of the ID