<u>Report on Assignment</u>

<u>Code Details</u>

I have basically mainly used three maps for this assignment. The first is a map which stores the person Ids as a key and the person struct as the value(name :persons).The major operations like get titles, get name and get salary all have accessed the person using their person ids. Using an unordered_map for this purpose has thus been beneficial.

The map can be accessed by the key. Since the unordered map is keyed by personID it can be accessed by constant time.

I have also used two multimaps. One multimap will have salary as the keys and the Person struct as the value(name :personOrderSalary). Another multimap will have the names as the key and the person struct as the value(name :personOrderName). A multimap has been used as they can have multiple elements with same key values. Also a multimap has the keys automatically sorted. This takes each entry a O(nlogn) time.Since these map are already sorted. The alphalist list and salarylist method can be called on these methods and can return vectors in O(n) time. The max and min also can be obtained in Constant time O(1).Median First Quartile and third Quartile can be obtained through either constant time O(1) or Maximum of O(n) time.

The method find ceo has a worst case time complexity of O(n). It iterates through the unordered_map and check for the person who doesn't have any boss.Change salary and Changes names edit both the multimaps and the unordered_map. It has a time complexity of O(n).They edit the names or salaries in the unrdered_map and (salary keyed map or name keyed map).
Remove element has the complexity of O(n). It takes the child elements of the elements to be removed. Adds those elements as the child of its boss. The boss id of the child elements are also changed to the boss id of the element to be removed.

Personnel_with_title and find_name method both require O(nlogn) time as both of the vectors to be returned need to be sorted by personID.Thus the matches are found through linear iteration and then inserted into a map which gets them sorted.

Add boss takes a constant time complexity O(1).
Underlings also takes a constant time complexity of O(1). The element is searched in the unordered map through the personID and the children attribute of person struct is returned. I have used a method ranking which has a time complexity of O(n) which recursively is used to rank the level of every person .
This is used in higher_lower_ranks to return the number of element with higher or lower ranks.This also has a time complexity of O(n).

The nearest common boss can have a maximum time complexity of the depth of the tree. O(n).

The implemented methods for HW assignment 3 are

- add_friendship
- remove_friendship
- get_friends
- all_friendships
- shortest_friendpath
- check_boss_hierarchy
- cheapest_friendpath

add_Friendship: This method is basically used to add a Friendship struct to the already existing persons unordered_map.The Friendship struct has two attributes , a person ID and corresponding cost. The persons objects contains an attribute which is vector of Friendships. This is used to keep track of the friends of a person. This operation has a time complexity of around O(n).

remove_friendship: This method is as from the name is used to remove friendships. It removes a Friendship struct from the friends attribute of a person. It has a simple linear time complexity of aroun O(n).

get_friends: This method has finds all the Friendship objects for the given id sent as argument. It also finds all the other ids which has this id in their friends attribute of objects structs. This has a time complexity of O(n*n).

shortest_friendpath:This method finds the shortest path to reach from fromID to toID. This does not consider the weights but only the number of nodes that has to be traversed. This simply used a breadth first search to solve this problem. For the breadth First logic another custom method has been added which though a nested loop iterates every level at once and check if the match is found. Also at every level all the node are marked with the predecessor values. This enables to track the shortest path when the toid has finally been obtained,
Time Complexity of BFS Gives a **O(|V|+|E|)** time complexity.

check_boss_hierarchy: This method check first of there are only one boss . If so then it checks if there are any cycles using the method checkCyclic().The complexity of this method is O(n*n).

cheapest_friendpath: This uses dijkstra's algorithm to find the least weighted path.Algorithm is used here is with Time complexity of O(V^2).The additional supplementary methods used are minUnvisited and DJKalgorithm.