For this project I have used four different STL containers which are:

“unordered_map<PersonID, Person*>”, “multimap<string, PersonID>”, “multimap<Salary, PersonID>”, and vector<Person*>. I have used the unordered_map for storing pairs of IDs and pointers to a type of struct-which could hold each person’s information. I have used this container because all operations on this kind of container are done within a reasonable complexity and I did not need the pairs to be sorted by the key so I used unordered maps. I used the other two containers which are multimaps to store pairs of name/salary and IDs. This approach helped me to save some time for sorting because those containers are some maps which are always sorted by their keys and they also could have non-unique keys. The last vector is used for putting all pointers -which point to people- in a vector so we can use this vector in finding the nth-element with a better performance.

The complexity of each operation is listed below:

- Constructor: Nothing ☺
- Deconstructor: O(n)
- add_person: O(n)
- remove_person: O(n)
- get_name: O(n) - Theta(1)
- get_title: O(n) - Theta(1)
- get_salary: O(n) - Theta(1)
- find_persons: O(log(n))
- personnel_with_title: O(n)
- change_name: O(n)
- change_salary: O(n)
- add_boss: O(n)
- size: O(constant)
- clear: O(n)
- underlings: max[O(n), O(m*log(m))] and ‘m’ here is the size of ‘underlings’ vector of each person and ‘n’ is all the people in our unordered_map.
- personnel_alphabetically, personnel_salary_order: O(n)
- find_ceo: O(n)
- nearest_common_boss: O(m*k) in which 'm' is number of direct or undirect bosses of each person and 'k' is the size of boss vector first person.
- higher_lower_ranks: O(n*m) in which 'n' is number of all employees and 'm' is number of direct or undirect bosses of each person.
- min_salary and max_salary: O(constant)
- median_salary, first_quartile_salary, and third_quartile_salary: O(log(n))