# Performance Characteristics of Standard Containers

The three standard containers that will be compared will be the std::map, std::list and std::unordered\_map. Their worst-case performance will be highlighted when considering their complexity.

## std::map

A map is implemented with a Red-Black Binary tree (cppreference.com 2017). The lookup only looks at half of the data set at each recursion, meaning that the data set which is looked at shrinks by half each time giving a time complexity of O(log n). When searching a binary tree, only part of the data in considered because of the way that it is implemented meaning less elements to search and therefore, a better performance (Horowitz and Mehta 2006).

Maps are used when the data set requires a container to store a key value pair. Elements have a key that is mapped to a value and is referenced by its key rather than an indexing value (cplusplus.com 2016b). Although they are generally slower than unordered\_maps at accessing elements, which as a O(1) time complexity on average, they have the ability to allow direct iteration on subsets because of the specific order that they are stored in (cplusplus.com 2016b).

## std::list

A list is implemented using a doubly-linked list (cplusplus.com 2016a). Each node is connected to the next node and also the previous through a pointer. When inserting a node at any known index, the time complexity is O(1) since the list can easily adjust the pointers to the next and previous nodes and do the same with the neighbouring nodes without having to change the memory location of each of the other nodes, which is costly. When finding an element within the list, its time complexity is O(n) since the list isn’t indexed sequentially in memory as a vector is and therefore finding an element would require traversal from the front to the end of the list.

The list would be used to store an array of singular data, the order of which will constantly change. What is meant by this can be explained by the following example, if data *y* needed to be inserted after position *x*, *x* would change its pointer to *y* and *y* to *x* and whatever formerly followed *x*. The list allows for the data to be in a specific order without compromising resources when data is inserted into the middle frequently.

## std::unordered\_map

An unordered map is a hash table that stores a key-value pair (cplusplus.com 2016c). has a constant time operation O(1) on all operations, provided that there are no collisions (Ahlgren 2013). However, if there are collisions, The unordered map uses the bucket implementation. This then changes to a O(n) time complexity in worst-case when traversing the map since the buckets are essentially linked lists.

An unordered maps are generally faster than maps at accessing individual elements by their key, although “they are generally less efficient for range iteration” (cplusplus.com 2016c) and therefore would be used in a case where elements are individually selected by their key. For example, if a program were to conduct lots of lookup operations but limited number of range iterations, an unordered map should be considered.

# Three Million Men: Predicting Performance

The searchable data structure requires both lookup and traversal. The issue is which will be used more than the other? There is a 50:50 chance for either depending on what the first name arbitrarily selected is, if it is closer to the westerly point then the answer would require faster individual and if it was closer to the easterly point then the answer would be faster traversal since the names would need to be found from the value rather than key. The list would be the first thing to disregard as this is better suited to a random access use case rather than a traversal.

Since the pieces of paper contained two values that can be used usefully, the natural thought would drift to either of the maps, since they are implemented in a way where the containers are referenced by their key and not their index. Looking at the worst case search complexity, a Standard Template Library (STL) map boasts an impressive O(log n) whereas a STL unordered map has a slightly worse time complexity of O(n) (Ahlgren 2013). Even though the unordered map does not perform as well in a worst case scenario, the choice of container used will be the STL unordered map as this, on average, has a time complexity of O(1) for both searching and access whereas the STL map is O(log n).

# References

Ahlgren, J., 2013. *STL Container Performance* [online]. john-ahlgren.blogspot.co.uk. Available at: <http://john-ahlgren.blogspot.co.uk/2013/10/stl-container-performance.html> [Accessed 12 January 2017].

cplusplus.com, 2016a. *list - C++ Reference* [online]. cpluscplus.com. Available at: <http://www.cplusplus.com/reference/list/list/> [Accessed 12 January 2017].

cplusplus.com, 2016b. *map - C++ Reference* [online]. cplusplus.com. Available at: <http://www.cplusplus.com/reference/map/map/> [Accessed 12 January 2017].

cplusplus.com, 2016c. *unordered\_map - C++ Reference* [online]. cplusplus.com. Available at: <http://www.cplusplus.com/reference/unordered_map/unordered_map/> [Accessed 12 January 2017].

cppreference.com, 2017. *std::map - cppreference.com* [online]. cppreference.com. Available at: <http://en.cppreference.com/w/cpp/container/map> [Accessed 12 January 2017].

Horowitz, E., and Mehta, D., 2006. *Fundamentals of data structures in C.* Galgotia Publications.