**Task 2a.**

For each

Typical data structures (describe):

Suitable functions for searching and insertion (time complexity guarantees, & how data structures used enable the guarantees):

std::list

A doubly linked list is used to implement the std::list container. This data structure is composed of a list of nodes each of which is linked to its previous and preceding node within the list. This allows for bi-directional traversal through the list.

Searching : linear, iterating through each item. Unless first and last item.

Inserting : linerar, iterating through each item.

Unless at beginning or end… then O(1)

std::map

The container std::map has the underlying data structure of a red-black tree. A red-black tree is a self-balancing binary search tree (BST). BST’s are composed of nodes which hold values, each node has a left and right child node. Left child nodes have a lower value than that of their parent, while right child nodes have a higher value. This means nodes are kept in sorted order by their values. The self-balancing aspect of a red-black tree attempts to ensure there are as many nodes on the left of the root node as there are on the right. This allows for the fastest average search time.

Searching : thanks to layout of nodes searching skips about half of the tree so that each lookup, insertion or deletion takes time proportional to the logarithm of the number of items stored in the tree. (wikipedia). This is much better than the linear time required by std::lists;

Insertion : just use the .insert function (look this up). logarithm of the number of items

Searching: just use the .find function. logarithm of the number of items

std::unordered\_map

A hash table data structure is used in the implementation of std::unordered\_map. A hash table uses a hash function to calculate the index of an array of buckets or slots in which values are stored.

Searching: O(1) just use the .find function

Insertion : O(1) : just use the .insert function (look this up)

**Task 2b.**

Random order of pairs of names

Small amount of ram

Put in order

Data structure for inserting random list & then searching from (insertion & searching time important)

Data structure for holding resulting sequence (insertion time important)

std::list & std::map

std::list & std::unordered\_map

std::map & std::unordered\_map

Since the container used to store results requires searching and inserting elements at its front and back, of the three std::list seems a perfect choice. This is because of the doubly linked list data type used to implement the container which requires head and tail pointers.