1. In 1-2 paragraphs, describe the approach that you took for this assignment.

This practicum requires us to have a linked hashmap so the elements are in orders. This will be a combination with doubly linked list and hashmap. The main structure will be the hashmap as the previous homework, but we will have the head node, tail node and count in hashmap. Inside the node structure, we will have next node to link the single linked list for hashmap, before node and after node to link the doubly linked list. Before node will point to the node inserted before and it works as the previous node in DLL. After node will point to the node inserted after and it works as the next node in DLL.

When an element is inserted, there are two changes. One is in the hashmap and other is in the DLL. This program will analyze the key of the element and put it to the correct bucket and link it with other elements. Also, it will add the element to the back of DLL. When it element is inserted with lexicographical order, I will first compare the key with the head and tail by using strcpy() function to see if we could insert it into the front or back of the DLL. If the key should be inserted in the middle of DLL, I will use iterator to compare the element from the head until the key of this element is less than the key of iterator. Then, we will insert the element before the iterator. When the element is removed, it should be removed from the single linked list of hashmap and also the DLL.

2. Consider the implementation of your linked hash map in Part 1. For each of the functions in your implementation, list what the big-Oh complexity of that function is and give a brief (1 sentence) explanation.

Hashmap_create : Time complexity is O(1), space complexity is O(n).

Create_node: Time complexity is O(1).

Free_node: Time complexity is O(1).

Hashmap_delete: Time complexity is O(n). we will need to delete every node in the hashmap.

Hashmap_hasKey: time complexity is O(n). If the element is at the end of linked list, we will need to go through all items to find it.

Hashmap_insert(insertion order): the worst time complexity is O(n), the best time complexity is O(1). The worst case is we will need to go through to see if the key already exists and insert to DLL is O(1) so the time complexity is depending on the hashmap.

Remove_node: time complexity is O(1). We already know the node to remove, only need to disconnect it from DLL.

Hashmap_getFirst: time complexity is O(1). Space complexity is O(1). This will remove the first node from DLL so we do not need to go through other nodes.

Hashmap_getLast: time complexity is O(1). Space complexity is O(1). This will remove the last node from DLL so we do not need to go through other nodes.

Hashmap_getValue: time complexity is O(n). Space complexity is O(1). We may need to search all elements to find the key and return value.

Hashmap_removeKey: time complexity is O(n). Similar as the search function, we have to go through all elements to find the key in worst case.

Hashmap_removeKey: time complexity is O(n). Same as the search function.

Hashmap_printKey: time complexity is O(n). We will need to print every elements from the head to tail.

3. After we have made the modifications to the ordering in Part 3, which of the functions from Part 2 now have a different big-Oh complexity? For each of these functions, list what the new complexity is and give a brief (1 sentence) explanation of the difference.

   The insert function will have slightly difference in my implementation, but the time complexity is still O(n). Because when we insert the element, we not only need to search the linked list in hashmap but also need to search the DLL to find the position, all the elements are searched twice in the worst case.