# Name : Neron Parmar

# Student ID : 171690217

# File : a2.docx

# Date : 11/26/23

# All the work in the assignment is done by my own and no part of the assignment is copied or shared with anyone.

# **PART – A: ANALYSIS**

Here’s the analysis for the member functions of the Sorted Table:

1. **def insert(self, key, value):**

* The insertion function begins with a search for an existing key. This results in the time complexity of O(n).
* However, if the key is not found then it appends the new key value pair at the end of the lost, leading to an average time complexity of O(1).
* Yet, the subsequent sorting using a bubble sort algorithm causes the overall time complexity to be O(n^2), which is highly inefficient.
* Therefore, to improve this, considering a more efficient sorting algorithm, such as merge sort or quick sort would probably enhance the performance.

1. **def modify(self, key, value):**

* Modification function sequentially searched for a given key in a loop which results in a time complexity of the O(n).
* While searching, if the key is found then the modification operation is straight forward, with a constant time complexity of O(1).
* To enhance this function, we can replace the linear search with the binary search as this will result in the reduced time complexity of O(log n), making it more efficient.

1. **def remove(self, key):**

* This function is like the modify function, the removal function has the time complexity of O(n) because of the sequential search for the key.
* After the location of the key, the subsequent shift operation has an additional time complexity.
* However, to improve this, we can implement the binary search and optimize the removal process such as the marking of the elements for deletion and performing a cleanup after, this would likely enhance the efficiency.

1. **def search(self, key):**

* The search function sequentially looks for the key in the list and this results in the time complexity of the O(n).
* To improve this, we can implement a binary search which would significantly reduce the time complexity to O(log n), making it more efficient.

1. **def capacity(self):**

* This member function capacity has a constant time complexity of O(1) as it directly returns the precomputed capacity.

1. **def \_\_len\_\_(self):**

* The length function counts the non-none elements sequentially that results in the time complexity of the O(n).
* To improve this, maintaining the counter that is updated during insertions and removals would provide a constant time retrieval of length which makes it more efficient.

# **PART – B: SUGGESSTIONS**

Looking into the code, the best suggestions I that could be done and I found were in the insert, modify, remove and search functions. The suggestions are as follows:

1. **Optimizing the Sorting in the Insert member function:**

* The implementation of the insert function utilizes a bubble sort algorithm for sorting which results in the time complexity of O(n^2).
* This can be improved by replacing the bubble sort with a more efficient sorting algorithm such as merge sort or the quick sort algorithm this will significantly reduce the time complexity to O(n log n).

And the second suggestion I found that could help in improving is,

1. **Implementing the binary search for the modify, remove, and search member function:**

* In these functions for the current code the linear search hase been used which results in the time complexity of O(n).
* To improve this, we can implement binary search instead of linear search to improve the search efficiency. Thus, this will change the time complexity to O(log n) which provides a more efficient way to locate and modify records. Moreover, the binary search is beneficial in these functions because it can deal more efficiently with the sorted data and we have the case of the SortedTable class.