Yeseul An
CSS 342
Professor: Erika Parson
Program 5

Report on Program5(Skip List)

a) Output of driver.cpp Execution

```
#faculty members: 10
contents:
-inf   -inf   -inf   -inf   -inf   -inf
berger  berger  berger
cioch
erdly   erdly   erdly   erdly   erdly
fukuda
jackels
olson   olson   olson
stiber
sung
unknown unknown
zander  zander
+inf   +inf   +inf   +inf   +inf   +inf

deleting unknown
#faculty members: 9
contents:
-inf   -inf   -inf   -inf   -inf   -inf
berger  berger  berger
cioch
erdly   erdly   erdly   erdly   erdly
fukuda
jackels
olson   olson   olson
stiber
sung
zander  zander
+inf   +inf   +inf   +inf   +inf   +inf

finding stiber = 1

create another list
finding stiber = 1
#faculty members: 9

cost of find = 104
```

b) Performance Results (Comparison between Doubly-linked, MTF, Transposed, and Skip List)

```
dlist's find cost = 6491439
mtflist's find cost = 66448
translist's find cost = 6422436
skip's find cost = 2013580
```

c) Performance Consideration (Doubly-linked, Skip List, Transposed, MTF)

When it comes to finding an item in the **doubly linked list**, it takes O(n) because it has to linearly traverse each node in the list to find the item. Doubly linked list doesn't allow random access with indices like an array so that access of a random item in for the doubly linked list takes linear traversal of the list.

On the other hand, **skip list** improves the linear traversal of the doubly linked list in that it has multiple layers of lists. The finding an item starts very top left corner of the skip list. The upper layers of lists in the skip list plays a role as an express lane, which checks the main items in the lists (if the item's value is less than the item that is being searched for, move right). If it couldn't find the item in that level of the list, move down to the lower level to find the item. This process doesn't involve traversal of every single node like doubly linked list does, so the random access of an item in the skip list is faster than the doubly linked list. The item to be added, removed, and accessed in skip list is O(logn) time.

The **transpose list** has similar aspect in that it has to traverse the list linearly in order to find the item for the first time. However, one thing that is different from the doubly linked list is that it moves the found item to one position closer to the header. This involves the process of swapping found item with the item that is previous to the found item. This would allow the next search of that same item to be faster. Accordingly, the more frequently accessed items will move closer to the head and the items in the list will be ordered based on their access frequency. For the random access of an item, if randomly accessed item has higher precedence because it has been searched several times, it would be moved closer to the header and the finding time would become faster. In the performance result, transpose list's outcome time is slightly better than the doubly linked list because it moved found items closer to the header by 1 for certain time which is 1000 out of 10000 items.

The **move-to-front list** allows the frequently sought, high-probability item to stay at the front of the list. The searching time of the MTF list is faster than others because it moves the found items to the front of the list. If the randomly finding item is sought before, it would have faster finding time because it was moved and stayed closer to the head node. In the performance result, move-to-front list has the fastest finding time than other lists because the founded items are jumping ahead to the front so when next time they are to be found they are quickly to be found.