Pre-lab Part 1

1.

```
def bf_insert(self, key):
        index1 = hash(salt1, key) % self.size
        bv_set_bit(self, index1)
        index2 = hash(salt2, key) % self.size
        bv_set_bit(self, index1)
        index3 = hash(salt3, key) % self.size
        bv_set_bit(self, index1)
        return
def bf_delete(self):
        bv_delete(self)
        return
```

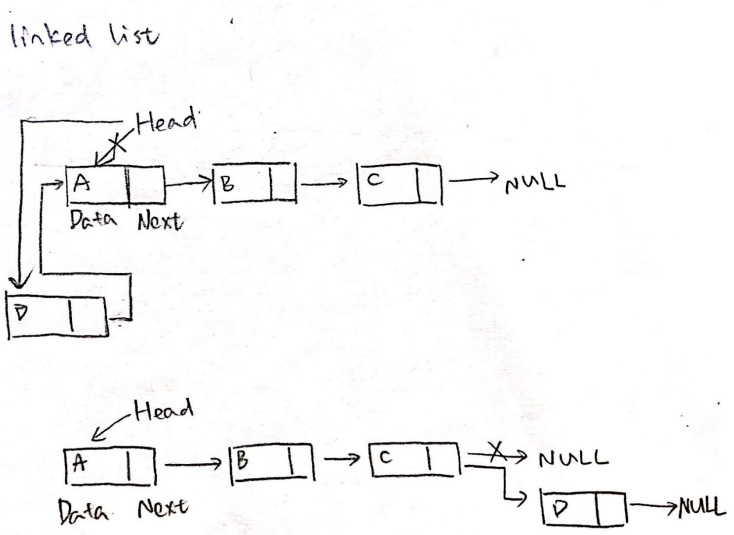
2.

insertion and search: O(k)

space of the actual data structure: O(m)

Pre-lab Part2

1.



2.

```
def ll_insert(self, newData):
        if move_to_front:
                newNode = ll_create(newData)
                newNode.next = self.head
                self.head = newNode
        else
```

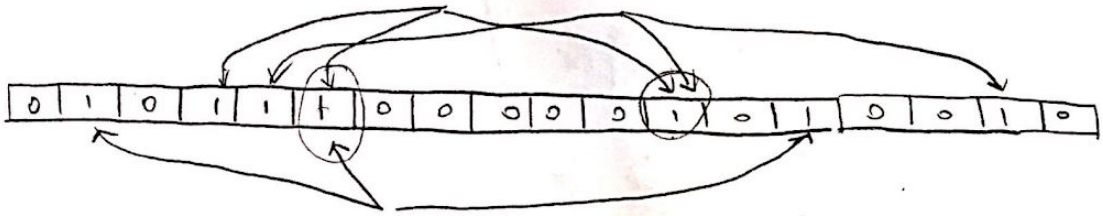
```
newNode = ll_create(newData)
temp = self.tail
temp.next = newNode
self.tail = newNode
newNode.next = NULL
```

bloom filter. (m bits and k hash functions)

operation: add
search/query.

time/space complexicity
insertion and search: O(k)
space of actul data structure
O(m)

k = 3 hash functions: h_1 h_2 h_3



pass {x, y, z} through 3 hash functions and set the corresponding bits in the bloom filter.

hash functions

hashes keys to generate indices for each value.

a salt

take in two inputs : a key, a salt

Struct HatterSpeak.

HatterSpeak. *gs: → oldspeak.
→ hatterspeak.