Berke Belgin

1-

```
private ListIterator<Entry<K,V>> find(K key){
 int index = key.hashCode() % CAPACITY;
 LinkedList<Entry<K,V>> temp = new LinkedList<Entry<K,V>>();
 ListIterator<Entry<K,V>> tempItr = table[index].listIterator();
 while(tempItr.hasNext()){
    Entry<K,V> e = tempItr.next();
    if(e.getKey().equals(key)) temp.add(e);
 }
 return temp.listIterator();
}
```

2-

a) Linked List is preferable in the situations where we make so much insertions and removals to a linear data structure since there is no need to shifting operation unlike arrays.
If there is a situation where there will be so much get operations to the data structure, it may not be preferable since it is not possible to binary search a linked list or get an element in a constant time except root.

b) It is preferable when we hold a comparable data. where we hold our data in a binary search tree but it also rebalances itself. It has a dynamic size and can get elements in log$_2$ operations.

Berke Bekşi
BB

c) We use hashing when we are able to generate distinct codes from some property of elements. By this distinct codes, we can determine the index in an array. So in the situations where the datas are as much as unique, it is preferable to use hashing since we can reach element with knowing only the object itself by getting index with its hashcode. It is not preffered if the elements are not distinct or memory size is limited since hashing requires large memory spaces to avoid collisions.

d) We use heap where we want to represent our Binary Tree in an array. We prefer heap when we want to make Breadth-first-search operation for example, since getting any element in heap is constant when we know its index in the array. We can search, traverse tree freely without recursions etc.