Quiz 5 (Oct 18)

By taking this quiz, you agree to adhere to the honor code of the class.

Name: netid:

Write your name and netid on **both** sides of the paper. Write your solution **first on this side**. If space is not enough, write to the other side. You can ask for extra paper if necessary.

Name: netid:

In `UnsortedMap<K, V>` we maintain a list of entries, `entrylist` , and perform `get(K key)` by searching this list for an entry with a key that matches `key` ; on Monday we'll also implement `put(K key, V value)` by adding an entry to the end of entrylist. Imagine we implemented a map `SortedMap<V>` that had integer keys and implemented `put(Integer key, V value)` such that the keys of the entries in `entrylist` are increasing. In this implementation we can write a more efficient implementation of `V get(Integer key)` by performing binary search. Implement this method recursively. Assume the value doesn't contain `null`.
If a map has N entries, what is the complexity of `get` for `UnsortedMap<K, V>` and `SortedMap<V>` ? Why?

Signatures

```
    public V get(K key) {
        Position<Entry<K,V>> current_pos = entrylist.first();
        for (int i = 0; i< entrylist.size(); i++) {
            Entry<K, V> entry = current_pos.getElement();
            if (entry.getKey() == key) return entry.getValue();
            current_pos = entrylist.after(current_pos);
        }
        return null;
    };
```

```
public class UnsortedMap<K, V> implements Map<K, V>{
    private int size;
    private PositionList<Entry<K, V>> entrylist;
    private class UnsortEntry<K, V> implements Entry<K, V>{
        K key;
        V value;
        UnsortEntry(K key, V value){
            this.key = key;
            this.value = value;
        }

        public K getKey() {return key;};
        public V getValue() {return value;};
    }

    public UnsortedMap(){
        entrylist = new DoublyLinkedList<Entry<K, V>>();
    }
```

```
public interface PositionList<E> extends List<E>{
        public Position<E> first();
        public Position<E> last();
        public Position<E> after(Position<E> p);
        public Position<E> before(Position<E> p);
        public void addAfter(Position<E> p, E e);
        public void addBefore(Position<E> p, E e);
        public void set(Position<E> p, E e);
        public E remove(Position<E> p);
}
```

Further assume there is a method `public Position<E> getAtIndex(int i)` runs in O(1).

Reference solution

```
public V get(Integer key) {
    return get(key, 0, entrylist.size());
}

public V get(Integer key, int l, int r) {
    if (l>r) return null;
    int mid = (l+r)/2;
    Entry<K, V> midEntry = entrylist.getAtIndex(mid);
    if (midEntry.getKey() == key) {
        return midEntry.getValue();
    } else if (midEntry.getKey() > key) {
        return get(key, l, mid-1);
    } else {
        return get(key, mid+1, r);
    }
}
```