3-4.

```
class Dictionary {
        private int[] arr;
        public Dictionary(int n) {
          this.arr = new int[n];
        }
        public void insert(int el) {
          this.arr[el - 1] = el;
        }
        public int search(int el) {
          return this.arr[el - 1];
        }
        public void delete(int el) {
          this.arr[el - 1] = -1;
        }
}
```

3-6.

In each node we store the predecessor and successor, which allows us to retrieve the successor predecessor in constant time. These only need to be updated when the tree is modified, that is an insertion or deletion operation was performed. And so, we only modify these operations.

Insertion:

After inserting a node x in O(log n), we must now find its predecessor and successor. This can be done in O(h) = O(log n) time. Then we set the successors predecessor and the predecessors successor to be x. This overhead takes O(log n) but as insertion was already O(log n) the time complexity has not changed.

Deletion:

Before deleting node x, we set the predecessor to the successor of x to be the predecessor of x, and similarly set the successor to the predecessor of x to be the successor of x. This overhead takes O(1) time, and so deletion remains O(log n).

3-10.

Create a tree with an empty bin as the root element

Take the first object and fit it into the bin

If the bin is filled up; remove it from the tree and jump to the beginning of the algorithm again

Otherwise, set the bin empty space to 1 - w

If the next object can't fit into the existing bin

Create a new bin

Assign that object to the bin

Set the bin's empty space to 1 - w

Insert this bin into the right position in the tree

Find the bin with the smallest space left after fitting the object into the bin; since this is a tree, this would imply finding the leftmost child that can contain that object

3-27.

A loop can be detected efficiently using the fast and slow pointer algorithm, where the fast pointer moves by two nodes and the slow pointer move by one node at a time. You can think of this as a tortoise and hare problem. Once the tortoise and hare mee, we know there's a loop. You can put a new pointer (tortoise) at the front and let both tortoise move until they meet. They'll always meet at the front of the loop.