```
public static<T> void remove(Stack<T> st, T e) {
	Stack<T> tmp = new Stack<T>();

	while(!st.empty()) {
		T x = st.pop();
		if(!x.equals(e))
			tmp.push(X);
	}

	while(!tmp.empty()) {
		st.push(tmp.pop());
	}
}

public static<T> Stack<T> concatenate(Stack<T> s1, Stack<T> s2) {
	Stack<T> s3 = new Stack<T>();
	Stack<T> tmp = new Stack<T>();

	while(!s1.empty()) {
		T x = s1.pop();
		s3.push(x);
		tmp.push(x);
	}

	while(!tmp.empty())
		st1.push(tmp.pop());

	while(!s2.empty()) {
		T x = s2.pop();
		s3.push(x);
		tmp.push(x);
	}

	while(!tmp.empty())
		st2.push(tmp.pop());

	while(!s3.empty())
		tmp.push(s3.pop());

	return tmp;
}

// memeber of class BST
public boolean searchReplace(int k, T e) {
	BSTNode<T> tmp = root;
	while(tmp != null) {
		if(tmp.key == k) {
			tmp.data = e;
			return true;
		}
		else if(k < tmp.key)
			tmp = tmp.left;
		else
			tmp = tmp.right;
	}
```

```
        return false;
}

// Recursive method, return true if BT n1 matches BT n2, false otherwise
public boolean equals(BTNode<T> n1, BTNode<T> n2) {
        if(n1 == null && n2 == null)
                return true;
        else if(n1 == null || n2 == null)
                return false;
        else if(!n1.data.equals(n2.data))
                return false;
        else
                return equals(n1.left, n2.left)
                        && equals(n1.right, n2.right);
}
```