

# Testing Report (User 12)

## Overview

Methods that Failed:

- `insert()`
- `remove()`
- `toString()`
- `split()`
- `join()`
- `iterator()`
- Custom edge cases

## Testing Methodology

### `insert()`

I tested this method by first inserting random integer keys and values into your `TreapMap`. I stored the values I inserted in a hash map. After inserting 1000 elements, I made sure that every element was found in the `HashMap` was found in the `TreapMap` and vice versa. I also made sure that the treap satisfied the binary tree property and the heap property.

### `remove()`

I tested this method similarly to how I tested `insert()`, except I gave the treap a 33% chance of removing an element. I made sure that when an element was removed, when trying to look it up again, null would be returned. I did the same checks as I did in `insert()`.

### `split()` and `join()`

I tested this method by inserting elements randomly into the `TreapMap` with sizes of powers of 2 until 1024. I split the Treap, made sure that the left Treap had elements strictly less than the key, and made sure that the right Treap had elements greater than or equal to the key. I also made sure both Treaps had the same keys as the original Treap. I then joined the subtrees and made sure it had the same elements as the original treap. For the subtrees and the joined treap, I made sure they satisfied the binary tree property and the heap property.

### `iterator()`

I tested this method by constructing a Treap similarly to the Treap constructed in `remove()` and made sure that each element as it was iterating was greater than its previous element. I made sure that each element that was iterated through was present in the `TreapMap`. I also made sure that `ConcurrentModificationException` was thrown when the `TreapMap` was altered. I made sure that operations that did not affect the `TreapMap`, such as removing a node that wasn't found or passing in nulls for all operations, did not affect the iterator.

### `toString()`

This was indirectly tested through my testing of the heap property and the bst property. I built a parser to build a tree from string output.

## Specific Edge Cases

The edge cases I tested were as follows. I made sure that on an empty Treap, `insert` and `join` returned null while `split` led to an array of empty strings when calling `Arrays.toString()` and `join` with an empty Treap led to another

empty Treap. I also made sure that when you pass in null to any of these methods, the methods return null and does not alter the Treap. I also made sure that when all entries were removed, the treap was properly reset to an empty Treap.

## Notes

- Null pointer exception when trying to remove an element from an empty TreapMap.
- Infinite loop when trying to split an empty TreapMap and when trying to *split()* in general.
- Many of your methods seem to have an infinite loop in it (i.e. *toString()*).
- Iterator iterates through nodes instead of the type of the keys.
- Null pointer exceptions when passing in null as an argument
  - *remove()*
  - *insert()* for both the key and value
  - *split()*
- Seems to be an infinite loop when deleting the root
- When inserting a bunch of values into a treap, not all of the values could be found in the treap when *lookup()* was called
- I could not check that you properly threw *ConcurrentModificationException* due to the above bugs
- Recommendations: look into making sure that while you are updating the treap that it handles edge cases surrounding the root and leaf nodes. Also, make sure your constructor is interpretable. For example, provide a default constructor.