George Gabricht
56735102 - ggabrich

ICS 46 - HW 6 Report

```
^C
ggabrich@andromeda-4 16:18:49 ~/ics46/hw/ggabrich_hw6
$ make
echo ---------compiling testTree.cpp to create executable program test_tree---------
---------compiling testTree.cpp to create executable program test_tree---------
g++ -ggdb -std=c++11 -Wpedantic -Wall -Wextra -Werror -Wzero-as-null-pointer-constant testTree.cpp -
test_tree
ggabrich@andromeda-4 16:19:04 ~/ics46/hw/ggabrich_hw6
$ test_tree
File: random.txt. Num Words: 4500.       Function: insertAllWords.      Time: 0.006339s
File: random.txt. Num Words: 4500.       Function: findAllWords.        Time: 0.006011s
File: random.txt. Num Words: 4500.       Function: removeAllWords.      Time: 0.006856s
File: random.txt. Num Words: 9000.       Function: insertAllWords.      Time: 0.013811s
File: random.txt. Num Words: 9000.       Function: findAllWords.        Time: 0.013047s
File: random.txt. Num Words: 9000.       Function: removeAllWords.      Time: 0.014749s
File: random.txt. Num Words: 13500.      Function: insertAllWords.      Time: 0.020391s
File: random.txt. Num Words: 13500.      Function: findAllWords.        Time: 0.020426s
File: random.txt. Num Words: 13500.      Function: removeAllWords.      Time: 0.022043s
File: random.txt. Num Words: 18000.      Function: insertAllWords.      Time: 0.019436s
File: random.txt. Num Words: 18000.      Function: findAllWords.        Time: 0.018765s
File: random.txt. Num Words: 18000.      Function: removeAllWords.      Time: 0.021099s
File: random.txt. Num Words: 22500.      Function: insertAllWords.      Time: 0.024799s
File: random.txt. Num Words: 22500.      Function: findAllWords.        Time: 0.02404s
File: random.txt. Num Words: 22500.      Function: removeAllWords.      Time: 0.027333s
File: random.txt. Num Words: 27000.      Function: insertAllWords.      Time: 0.029021s
File: random.txt. Num Words: 27000.      Function: findAllWords.        Time: 0.029372s
File: random.txt. Num Words: 27000.      Function: removeAllWords.      Time: 0.032832s
File: random.txt. Num Words: 31500.      Function: insertAllWords.      Time: 0.037039s
File: random.txt. Num Words: 31500.      Function: findAllWords.        Time: 0.052612s
File: random.txt. Num Words: 31500.      Function: removeAllWords.      Time: 0.058221s
File: random.txt. Num Words: 36000.      Function: insertAllWords.      Time: 0.062649s
File: random.txt. Num Words: 36000.      Function: findAllWords.        Time: 0.060624s
File: random.txt. Num Words: 36000.      Function: removeAllWords.      Time: 0.054311s
File: random.txt. Num Words: 40500.      Function: insertAllWords.      Time: 0.047167s
File: random.txt. Num Words: 40500.      Function: findAllWords.        Time: 0.04596s
File: random.txt. Num Words: 40500.      Function: removeAllWords.      Time: 0.051561s
File: random.txt. Num Words: 45000.      Function: insertAllWords.      Time: 0.051973s
File: random.txt. Num Words: 45000.      Function: findAllWords.        Time: 0.052133s
File: random.txt. Num Words: 45000.      Function: removeAllWords.      Time: 0.057555s

 Words in random_small.txt of:
        length 2: 1 words
        length 3: 2 words
        length 4: 10 words
        length 5: 10 words
        length 6: 12 words
        length 7: 24 words
        length 8: 27 words
        length 9: 22 words
        length 10: 15 words
        length 11: 5 words
        length 12: 2 words
        length 13: 2 words
        length 14: 1 words
        length 15: 1 words
        length 16: 1 words
ggabrich@andromeda-4 16:19:05 ~/ics46/hw/ggabrich_hw6
$ valgrind test_tree
==1641== Memcheck, a memory error detector
==1641== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1641== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1641== Command: test_tree
==1641==
```

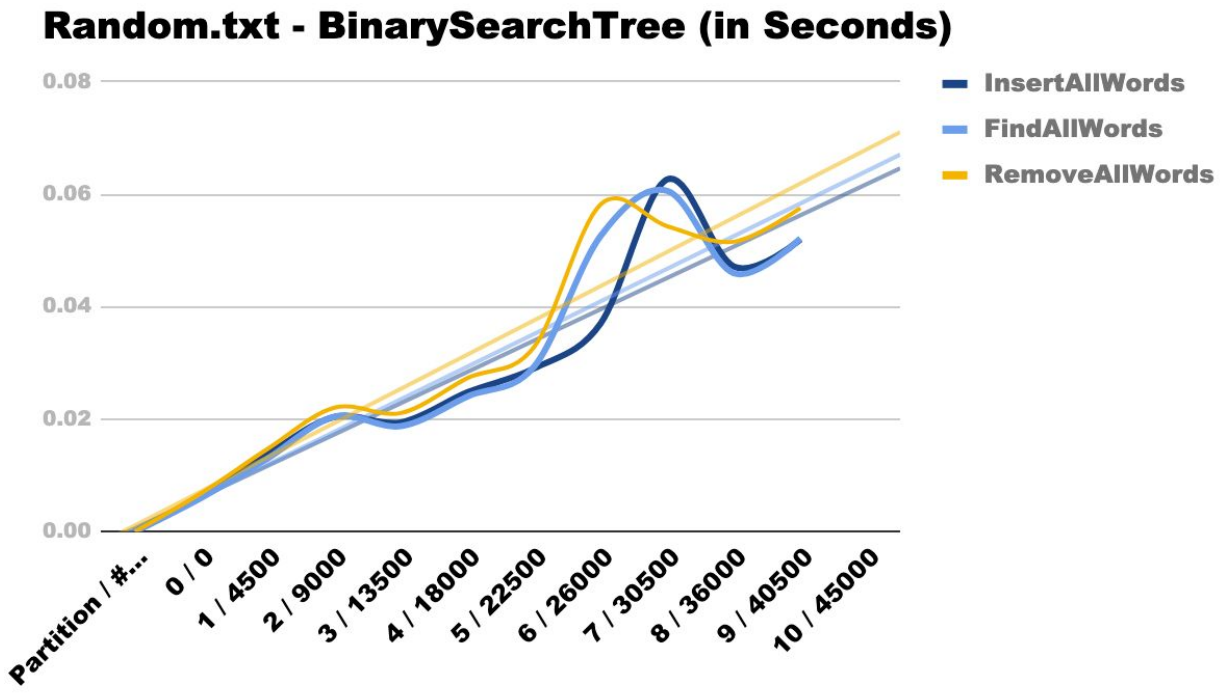
George Gabricht
56735102 - ggabrich

```
ggabrich@andromeda-4 16:19:05 ~/ics46/hw/ggabrich_hw6
$ valgrind test_tree
==1641== Memcheck, a memory error detector
==1641== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1641== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==1641== Command: test_tree
==1641==
File: random.txt. Num Words: 4500.       Function: insertAllWords.      Time: 0.131955s
File: random.txt. Num Words: 4500.       Function: findAllWords.        Time: 0.10517s
File: random.txt. Num Words: 4500.       Function: removeAllWords.      Time: 0.130078s
File: random.txt. Num Words: 9000.       Function: insertAllWords.      Time: 0.246343s
File: random.txt. Num Words: 9000.       Function: findAllWords.        Time: 0.211826s
File: random.txt. Num Words: 9000.       Function: removeAllWords.      Time: 0.259486s
File: random.txt. Num Words: 13500.      Function: insertAllWords.      Time: 0.381875s
File: random.txt. Num Words: 13500.      Function: findAllWords.        Time: 0.367636s
File: random.txt. Num Words: 13500.      Function: removeAllWords.      Time: 0.394025s
File: random.txt. Num Words: 18000.      Function: insertAllWords.      Time: 0.513562s
File: random.txt. Num Words: 18000.      Function: findAllWords.        Time: 0.445972s
File: random.txt. Num Words: 18000.      Function: removeAllWords.      Time: 0.533073s
File: random.txt. Num Words: 22500.      Function: insertAllWords.      Time: 0.654068s
File: random.txt. Num Words: 22500.      Function: findAllWords.        Time: 0.566017s
File: random.txt. Num Words: 22500.      Function: removeAllWords.      Time: 0.673076s
File: random.txt. Num Words: 27000.      Function: insertAllWords.      Time: 0.797472s
File: random.txt. Num Words: 27000.      Function: findAllWords.        Time: 0.686749s
File: random.txt. Num Words: 27000.      Function: removeAllWords.      Time: 0.813831s
File: random.txt. Num Words: 31500.      Function: insertAllWords.      Time: 0.935771s
File: random.txt. Num Words: 31500.      Function: findAllWords.        Time: 0.808115s
File: random.txt. Num Words: 31500.      Function: removeAllWords.      Time: 1.01001s
File: random.txt. Num Words: 36000.      Function: insertAllWords.      Time: 1.07133s
File: random.txt. Num Words: 36000.      Function: findAllWords.        Time: 0.932843s
File: random.txt. Num Words: 36000.      Function: removeAllWords.      Time: 1.11644s
File: random.txt. Num Words: 40500.      Function: insertAllWords.      Time: 1.21612s
File: random.txt. Num Words: 40500.      Function: findAllWords.        Time: 1.06257s
File: random.txt. Num Words: 40500.      Function: removeAllWords.      Time: 1.25267s
File: random.txt. Num Words: 45000.      Function: insertAllWords.      Time: 1.36487s
File: random.txt. Num Words: 45000.      Function: findAllWords.        Time: 1.1916s
File: random.txt. Num Words: 45000.      Function: removeAllWords.      Time: 1.40062s

 Words in random_small.txt of:
        length 2: 1 words
        length 3: 2 words
        length 4: 10 words
        length 5: 10 words
        length 6: 12 words
        length 7: 24 words
        length 8: 27 words
        length 9: 22 words
        length 10: 15 words
        length 11: 5 words
        length 12: 2 words
        length 13: 2 words
        length 14: 1 words
        length 15: 1 words
        length 16: 1 words
==1641==
==1641== HEAP SUMMARY:
==1641==     in use at exit: 0 bytes in 0 blocks
==1641==   total heap usage: 263,060 allocs, 263,060 frees, 14,494,556 bytes allocated
==1641==
==1641== All heap blocks were freed -- no leaks are possible
==1641==
==1641== For counts of detected and suppressed errors, rerun with: -v
==1641== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

George Gabricht
56735102 - ggabrich

Random.txt - BinarySearchTree (in seconds)			
Partition / # Words	InsertAllWords	FindAllWords	RemoveAllWords
1 / 4500	0.006339	0.006011	0.006865
2 / 9000	0.013811	0.013047	0.014749
3 / 13500	0.020391	0.020426	0.022043
4 / 18000	0.019436	0.018765	0.021099
5 / 22500	0.024799	0.02404	0.027333
6 / 26000	0.029021	0.029372	0.032832
7 / 30500	0.037039	0.052612	0.058221
8 / 36000	0.062649	0.060624	0.054311
9 / 40500	0.047167	0.04596	0.051561
10 / 45000	0.051973	0.052133	0.057555



George Gabricht
56735102 - ggabrich

The Code:

```
// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	static TreeNode * binarySearch (KeyType key, TreeNode *node) { // O(Log N)
		TreeNode * cur = node; // 1
		while (cur != nullptr) { // log N
			if (key < cur->key) {
				if (cur->left == nullptr) {
					return cur;
				} else if (cur->left->key == key) {
					return cur;
				}
				cur = cur->left;
			} else if (key > cur->key) {
				if (cur->right == nullptr) {
					return cur;
				} else if (cur->right->key == key) {
					return cur;
				}
				cur = cur->right;
			} else {
				return cur;
			}
		}
		return cur;
	}

// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	static TreeNode * findPred (TreeNode *node) { // O(Log N)
		TreeNode *cur = node->left;
		while (cur->right != nullptr) {
			cur = cur->right;
		}
		return cur;
	}

	// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	static TreeNode * findSuc (TreeNode *node) { // O(Log N)
		TreeNode *cur = node->right;
		while (cur->left != nullptr) {
			cur = cur->left;
		}
		return cur;
	}
```

George Gabricht
56735102 - ggabrich

```
// Always - O(1)
	static TreeNode * newNode (KeyType k, ElementType v, TreeNode *l = nullptr, TreeNode *r =
nullptr) {
		return new TreeNode<KeyType, ElementType>(k, v, l, r);
	}

// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	void insert (KeyType key, ElementType value) {
		root = TreeNode<KeyType, ElementType>::insert(key, root, value);
	}

// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	static TreeNode * insert (KeyType k, TreeNode *node, ElementType v = ElementType()) {
		TreeNode *result = binarySearch(k, node);
		if (result == nullptr) {
			return newNode(k, v);
		} else if (k < result->key) {
			if (result->left && k == result->left->key) {
				result->left->value = v;
			} else {
				result->left = newNode(k, v);
			}
		} else if (k > result->key) {
			if (result->right && k == result->right->key) {
				result->right->value = v;
			} else {
				result->right = newNode(k, v);
			}
		} else {
			result->value = v;
		}
		return node;
	}

	// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	ElementType find (KeyType key) {
		TreeNode <KeyType, ElementType> *result = TreeNode <KeyType, ElementType>::find(key,
root);
		ElementType val = result->getValue();
		return val;
	}

// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	static TreeNode * find (KeyType k, TreeNode *node) {
```

```
            TreeNode *result = binarySearch(k, node);
            if (result == nullptr) {
                  return nullptr;
            } else if (k != result->key) {
                  if (k < result->key && result->left) {
                        return result;
                  } else if (k > result->key && result->right) {
                        return result;
                  }
                  return nullptr;
            }
            return result;
      }

// Worst Case - O(N) when inserted in nearly sorted order
      // Typical Case - O(Log N)
      ElementType & operator[] (KeyType key) {
            TreeNode <KeyType, ElementType> *result = TreeNode<KeyType, ElementType>::find(key,
root); // log N
            if (result == nullptr) {
                  root = TreeNode <KeyType, ElementType>::insert(key, root);
                  result = root;
            } else if (key < result->getKey()) {
                  result = result->getLeft();
            } else if (key > result->getKey()) {
                  result = result->getRight();
            }
            if (result == nullptr) { // 1
                  try {
                        root = TreeNode<KeyType, ElementType>::insert(key, root); // log N
                        return TreeNode<KeyType,ElementType>::find(key, root)->getValue(); // log N
                  } catch (...) {
                        cout << "Could not insert on [] operator" << endl;
                        TreeNode<KeyType, ElementType>::deleteTree(root);
                        exit(-1);
                  }
            }
            return result->getValue();
      }

// Worst Case - O(N) when inserted in nearly sorted order
      // Typical Case - O(Log N)
      void remove (KeyType key) {
            TreeNode<KeyType, ElementType>::remove(key, root);
      }
```

George Gabricht
56735102 - ggabrich

```
// O(N) - because traverses whole tree
void countLengths() {
	int lens[100];
	for (int ndx = 0; ndx < 100; ndx++) {
		lens[ndx] = 0;
	}
	TreeNode<KeyType, ElementType>::countLengths(lens, root);
	for (int ndx = 0; ndx < 100; ndx++) {
		if (lens[ndx] > 0) {
			// handle print output
			cout << "\tlength " << ndx + 1 << ": ";
			cout << lens[ndx] << " words" << endl;
		}
	}
}

// O(N) - Always because traversing whole list (N items)
static void countLengths(int * lens, TreeNode * node) {
	if (node == nullptr) {
		return;
	} else {
		countLengths(lens, node->left);
		lens[node->key.size() - 1]++;
		countLengths(lens, node->right);
	}
}

// Worst Case - O(N) when inserted in nearly sorted order
	// Typical Case - O(Log N)
	static TreeNode * remove (KeyType k, TreeNode *node) {
		TreeNode *result = binarySearch(k, node), *cur; // log N
		if (result == nullptr) { // 1
			return result;
		} else if (result == node) { // 1
			if (k < result->key) { // 1
				cur = result->left;
			} else if (k > result->key) { // 1
				cur = result->right;
			} else { // 1
				cur = result;
			}
		} else if (k < result->key) { // 1
			if (result->left == nullptr) { // 1
				return node;
			}
			cur = result->left;
		} else if (k > result->key) { // 1
			if (result->right == nullptr) { // 1
```

George Gabricht
56735102 - ggabrich

```
                return node;
            }
            cur = result->right;
        }
        // Awaiting answers on Piazza...
        // no children
        if (cur->left == nullptr) { // 1
            if (cur->right == nullptr) { // 1
                if (cur == result->left) { // 1
                    result->left = nullptr;
                } else { // 1
                    result->right = nullptr;
                }
                deleteNode(cur);
            } else { // 1
                if (cur == result->left) { // 1
                    result->left = cur->right;
                } else { // 1
                    result->right = cur->right;
                }
                deleteNode(cur);
            }
        } else if (cur->right == nullptr) { // 1 - 1 child
            if (cur == result->left) { // 1
                result->left = cur->left;
            } else { // 1
                result->right = cur->left;
            }
            deleteNode(cur);
        } else { // 1 - 2 children
            TreeNode *pred = findPred(cur);
            /*if (pred->right == nullptr) {
                suc = findSuc(cur);
                cur->key = suc->key;
                cur->value = suc->value;
                remove(k, suc);
            }*/
            KeyType tempK = cur->key;
            ElementType tempV = cur->value;
            cur->key = pred->key;
            cur->value = pred->value;
            pred->key = tempK;
            pred->value = tempV;
            remove(k, cur);
        }
        return node;
    }
```