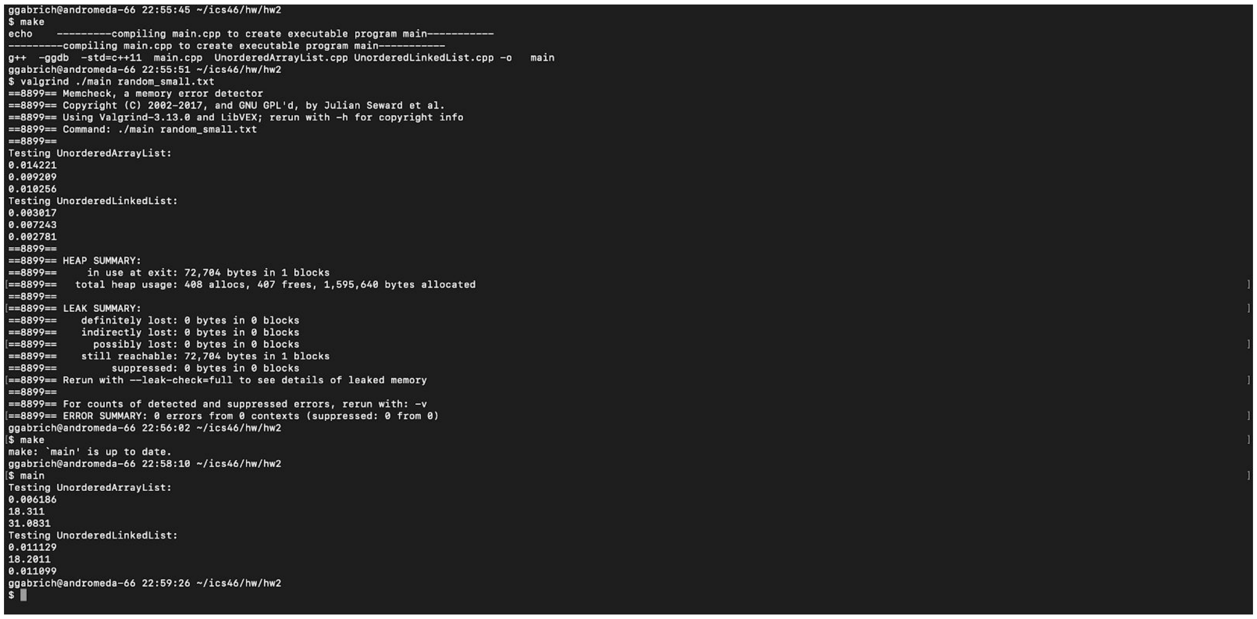

```
ggabrich@andromeda-66 22:55:45 ~/ics46/hw/hw2
$ make
echo    ---------compiling main.cpp to create executable program main-----------
---------compiling main.cpp to create executable program main-----------
g++  -ggdb  -std=c++11  main.cpp  UnorderedArrayList.cpp UnorderedLinkedList.cpp -o   main
ggabrich@andromeda-66 22:55:51 ~/ics46/hw/hw2
$ valgrind ./main random_small.txt
==8899== Memcheck, a memory error detector
==8899== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8899== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8899== Command: ./main random_small.txt
==8899==
Testing UnorderedArrayList:
0.014221
0.009209
0.010256
Testing UnorderedLinkedList:
0.003017
0.007243
0.002781
==8899==
==8899== HEAP SUMMARY:
==8899==     in use at exit: 72,704 bytes in 1 blocks
==8899==   total heap usage: 408 allocs, 407 frees, 1,595,640 bytes allocated
==8899==
==8899== LEAK SUMMARY:
==8899==    definitely lost: 0 bytes in 0 blocks
==8899==    indirectly lost: 0 bytes in 0 blocks
==8899==      possibly lost: 0 bytes in 0 blocks
==8899==    still reachable: 72,704 bytes in 1 blocks
==8899==         suppressed: 0 bytes in 0 blocks
==8899== Rerun with --leak-check=full to see details of leaked memory
==8899==
==8899== For counts of detected and suppressed errors, rerun with: -v
==8899== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ggabrich@andromeda-66 22:56:02 ~/ics46/hw/hw2
$ make
make: `main' is up to date.
ggabrich@andromeda-66 22:58:10 ~/ics46/hw/hw2
$ main
Testing UnorderedArrayList:
0.006186
18.311
31.0831
Testing UnorderedLinkedList:
0.011129
18.2011
0.011099
ggabrich@andromeda-66 22:59:26 ~/ics46/hw/hw2
$ 
```

```
// inserts string word into the array list
void UnorderedArrayList::insert(string word) { // O(N)
	if (size == capacity) {
		buf = resizeArray(capacity, buf);
		capacity *= 2;
	}
	buf[size++] = word;
}

void insert_all_words(string file_name, UnorderedArrayList & L) // O(N^2)
{
	double eTime;
	Timer t;// declare timer object
	ifstream wordFile;
	string newWord;
	wordFile.open(file_name);
	if (wordFile.is_open()) {
		t.start();// start timer
		while (getline(wordFile, newWord)) {
			L.insert(newWord);
		}
		t.elapsedUserTime(eTime);// stop timer
		wordFile.close();
	}
	cout << eTime << endl;// report time
}
```

```
// finds string word in the array list.
// If found, returns true, else false.
bool UnorderedArrayList::find(string word) { // O(N)
	for (int i = 0; i < size; i++) {
		if (buf[i] == word) {
			return true;
		}
	}
	return false;
}

void find_all_words(string file_name, UnorderedArrayList & L) // O(N^2)
{
	double eTime;
	Timer t;// declare timer object
	ifstream wordFile;
	string newWord;
	wordFile.open(file_name);
	if (wordFile.is_open()) {
		t.start();// start timer
		while (getline(wordFile, newWord)) {
			L.find(newWord);
		}
		t.elapsedUserTime(eTime);// stop timer
		wordFile.close();
	}
	cout << eTime << endl;// report time
}

// removes string word from the array list.
void UnorderedArrayList::remove(string word) { // O(N^2)
	if (isEmpty()) {
		throw 0;
	}
	for (int i = 0; i < size; i++) {
		if (buf[i] == word) {
			for(int j = i; j < size - 1; j++) {
				buf[j] = buf[j + 1];
			}
			size--;
			return;
		}
	}
	throw 0;
}
```

```
void remove_all_words(string file_name, UnorderedArrayList & L) // O(N^3)
{
    double eTime;
    Timer t;// declare timer object
    ifstream wordFile;
    string newWord;
    wordFile.open(file_name);
    if(wordFile.is_open()) {
        t.start();// start timer
        while (getline(wordFile, newWord)) {
            L.remove(newWord);
        }
        t.elapsedUserTime(eTime);// stop timer
        wordFile.close();
    }
    cout << eTime << endl;// report time
}

        // inserts a new node at the front of the list and returns new head.
        static ListNode* insert(string word, ListNode *L) { // O(1)
            ListNode *newNode = new ListNode(word, L);
            return newNode;
        }

// inserts a new node at head of linked list with value of string word,
// sets head to this new node.
void UnorderedLinkedList::insert(string word) { // O(1)
    head = ListNode::insert(word, head);
}

void insert_all_words(string file_name, UnorderedLinkedList & L) // O(N)
{
    double eTime;
    Timer t;// declare timer object
    ifstream wordFile;
    string newWord;
    wordFile.open(file_name);
    if (wordFile.is_open()) {
        t.start();// start timer
        while (getline(wordFile, newWord)) {
            L.insert(newWord);
        }

        t.elapsedUserTime(eTime);// stop timer
        wordFile.close();
    }
    cout << eTime << endl;// report time
}
```

```
        // finds a node with info value of string word,
        // returns true if found else false.
        static bool find(string word, ListNode *L) { // O(N)
            for (ListNode *cur = L; cur != nullptr; cur = cur->next) {
                if (cur->info == word) {
                    return true;
                }
            }
            return false;
        }

// searched through the linked list for a node with info == string word.
// returns true if found else false.
bool UnorderedLinkedList::find(string word) { // O(N)
    return ListNode::find(word, head);
}

void find_all_words(string file_name, UnorderedLinkedList & L) // O(N^2)
{
    double eTime;
    Timer t;// declare timer object
    ifstream wordFile;
    string newWord;
    wordFile.open(file_name);
    if (wordFile.is_open()) {
        t.start();// start timer
        while (getline(wordFile, newWord)) {
            L.find(newWord);
        }
        t.elapsedUserTime(eTime);// stop timer
        wordFile.close();
    }
    cout << eTime << endl;// report time
}
```

```
 // removes node with info value of string word.
        static ListNode* remove(string word, ListNode *L) { // O(N)
            if (L->info == word) {
                ListNode *temp = L;
                L = L->next;
                delete temp;
                return L;
            } else {
                for (ListNode *cur = L; cur->next != nullptr; cur++) {
                    if (cur->next->info == word) {
                        ListNode *temp = cur->next;
                        cur->next = cur->next->next;
                        delete temp;
                        return L;
                    }
                }
            }
            throw 0;
        }

// searched through the linked list for node with info == string word
// and deleted the node, handling edge cases appropriately
void UnorderedLinkedList::remove(string word) { // O(N)
    if (isEmpty()) {
        throw 0;
    }
    head = ListNode::remove(word, head);
}

void remove_all_words(string file_name, UnorderedLinkedList & L) // O(N^2)
{
    double eTime;
    Timer t;// declare timer object
    ifstream wordFile;
    string newWord;
    wordFile.open(file_name);
    if(wordFile.is_open()) {
        t.start();// start timer
        while (getline(wordFile, newWord)) {
            L.insert(newWord);
        }
        t.elapsedUserTime(eTime);// stop timer
        wordFile.close();
    }
    cout << eTime << endl;// report time
}
```