

```
1  #include <iostream>
2  #include <chrono>
3  #include <math.h>
4  #include <random>
5  #include <algorithm>
6
7  #include <vector>
8  #include <unordered_map>
9
10 // Lets look at searching lists vs vectors vs maps
11
12 /* Each function needs to be modified to return the time it took to      ↗
   complete
13 * the necessary searches. Do not include setup in the testing time.    ↗
   */
14 class Node {
15 private:
16     Node* _next;
17     int _data;
18
19 public:
20     Node(int data) : _next(nullptr), _data(data) {}
21
22     void setNext(Node* next) { _next = next; }
23     void setData(int data) { _data = data; }
24     Node* getNext() { return _next; }
25     int getData() { return _data; }
26 };
27
28 class LinkedList {
29 private:
30     int _size;
31     Node* _head;
32
33 public:
34     LinkedList() : _size(1), _head(new Node(_size)) {};
35     ~LinkedList() {
36         reset();
37         delete _head;
38     }
39
40     void addNode() {
41         _size++;
42         Node* traversal_ptr = _head;
43         Node* new_node = new Node(_size);
44
45         while (traversal_ptr->getNext() != nullptr) {
46             traversal_ptr = traversal_ptr->getNext();
47         }
```

```
48
49     traversal_ptr->setNext(new_node);
50 }
51
52 // This is the function which needs to be timed.
53 std::chrono::duration<double> searchNodeByValue(int value) {
54     int idx = 0;
55     int search_value = value + 1;
56
57     auto start = std::chrono::steady_clock::now();
58     Node* traversal_ptr = _head;
59
60     while (traversal_ptr->getNext() != nullptr) {
61         if (traversal_ptr->getData() == search_value) {
62             auto end = std::chrono::steady_clock::now();
63             return end - start;
64         }
65
66         idx++;
67         traversal_ptr = traversal_ptr->getNext();
68     }
69
70     if (traversal_ptr->getNext() == nullptr &&
71         traversal_ptr->getData() == search_value) {
72         auto end = std::chrono::steady_clock::now();
73         return end - start;
74     }
75
76     auto bad_end = std::chrono::steady_clock::now();
77     return start - bad_end; // Neg value for err
78 }
79
80 void reset() {
81     Node* traversal_ptr = _head;
82     Node* current_ptr = nullptr;
83
84     while (traversal_ptr->getNext() != nullptr) {
85         current_ptr = traversal_ptr;
86         traversal_ptr = traversal_ptr->getNext();
87         delete current_ptr;
88     }
89
90     _head = traversal_ptr;
91     _size = 1;
92     _head->setData(_size);
93 }
94 };
95
96 // Takes the position starting from 0
```

```
197 std::chrono::duration<double> testVectorSearch(int value, int value_pos,  ↗
    int vec_size) {
198     int search_value = value;
199     int search_value_pos = value_pos;
200     int vector_size = vec_size;
201
202     std::vector<int> integer_vector;
203     integer_vector.reserve(vector_size);
204
205     for (int idx = 0; idx < vector_size; idx++) {
206         integer_vector.push_back(0);
207         if (idx == search_value_pos) { integer_vector.push_back(value); }
208     }
209
210     //Time below here
211     auto start = std::chrono::steady_clock::now();
212     auto integer_it = std::find(integer_vector.begin(), integer_vector.end  ↗
        (), search_value);
213     auto end = std::chrono::steady_clock::now();
214
215     if (integer_it != integer_vector.end()) {
216         return end - start;
217     }
218
219     return start - end; // negative numbers means i know there's an err
220 }
221
222 std::chrono::duration<double> testMapSearch(std::string key, int value,  ↗
    int value_pos, int map_size) {
223     // Really quick and dirty map setup
224     std::string search_key = key;
225     int search_value = value;
226     int search_value_pos = value_pos;
227     int umap_size = map_size;
228
229     std::unordered_map<std::string, int> str_int_umap;
230
231     for (int idx = 0; idx < umap_size; idx++) {
232         str_int_umap.insert({ "Default", 0 });
233         if (idx == value) { str_int_umap.insert({search_key,  ↗
            search_value}); }
234     }
235
236     //Time below here
237     auto start = std::chrono::steady_clock::now();
238     int found_value = str_int_umap[search_key];
239     auto end = std::chrono::steady_clock::now();
240
241     return end - start;
```

```

142 }
143
144 std::vector<std::chrono::duration<double>> singleSearchStructureTest
    (LinkedList* list, int search_value_pos, int search_range, bool
    show_results) {
145     LinkedList* linked_list = list;
146     int value_pos = search_value_pos;
147     int range = search_range;
148     bool display_results = show_results;
149
150     std::vector<std::chrono::duration<double>> test_time_data;
151     double map_ns, vec_ns, list_ns;
152
153     // Start at 1 cuz linked linked constructor says so
154     for (int idx = 1; idx <= range; idx++) { linked_list->addNode(); }
155
156     std::chrono::duration<double> list_search_time = linked_list-
        >searchNodeByValue(value_pos);
157     std::chrono::duration<double> vec_search_time = testVectorSearch(1,
        value_pos, range);
158     std::chrono::duration<double> map_search_time = testMapSearch
        ("Target", 1, value_pos, range);
159
160     map_ns = map_search_time.count() * pow(10, 9);
161     vec_ns = vec_search_time.count() * pow(10, 9);
162     list_ns = list_search_time.count() * pow(10, 9);
163
164     test_time_data.push_back(map_search_time);
165     test_time_data.push_back(vec_search_time);
166     test_time_data.push_back(list_search_time);
167
168     if (display_results) {
169         std::cout << std::endl;
170         std::cout <<
            "-----"
            << std::endl;
171         std::cout << "                Single Test Results                "
            << std::endl;
172         std::cout <<
            "-----"
            << std::endl;
173         std::cout << "This test iterated " << value_pos + 1 << " times
            through structures containing " << range << " elements." <<
            std::endl;
174         std::cout << ">> Unordered Map search time:\t" <<
            map_search_time.count()
            << " == " << map_ns << " ns" << std::endl;
175         std::cout << ">>                Vector search time:\t" <<
            vec_search_time.count()

```

```

177         << " == " << vec_ns << " ns" << std::endl;
178         std::cout << ">>          List search time:\t" <<
            list_search_time.count()
179         << " == " << list_ns << " ns" << std::endl;
180     }
181
182     return test_time_data;
183 }
184
185 void manyRandomValueTest(LinkedList* list, int search_reps, int max_range,
    bool show_results) {
186     LinkedList* linked_list = list;
187     std::vector<std::chrono::duration<double>> single_test_data;
188     std::vector<std::vector<std::chrono::duration<double>>>
        multiple_test_data;
189     int repitions = search_reps, maximum_range = max_range;
190     bool display_results = show_results;
191
192     int range, value;
193     double sum_map = 0.0, sum_vec = 0.0, sum_list = 0.0;
194
195     // Run the test
196     for (int search_rep = 0; search_rep < repitions; search_rep++) {
197         srand((unsigned)time(NULL) + std::rand()); // Reseed the random
            generator
198         range = (std::rand() % maximum_range);
199         value = std::rand() % range;
200
201         single_test_data = singleSearchStructureTest(linked_list, value,
            range, false);
202         multiple_test_data.push_back(single_test_data);
203     }
204
205     // Calculate the results
206     for (int idx = 0; idx < repitions; idx++) {
207         sum_map += multiple_test_data[idx][0].count();
208         sum_vec += multiple_test_data[idx][1].count();
209         sum_list += multiple_test_data[idx][2].count();
210     }
211
212     double avg_map = sum_map / repitions;
213     double avg_vec = sum_vec / repitions;
214     double avg_list = sum_list / repitions;
215     double avg_map_ns = avg_map * pow(10, 9);
216     double avg_vec_ns = avg_vec * pow(10, 9);
217     double avg_list_ns = avg_list * pow(10, 9);
218
219     // Display the results
220     if (display_results) {

```

```

221     std::cout << std::endl;
222     std::cout <<
        "-----"
        << std::endl;
223     std::cout << "           Multiple Test Results"
        " << std::endl;
224     std::cout <<
        "-----"
        << std::endl;
225     std::cout << "Over " << repitions << " repitions, the average time
        recorded is: " << std::endl;
226     std::cout << ">> Unordered Map avg search time:\t" << avg_map
227         << " == " << avg_map_ns << " ns" << std::endl;
228     std::cout << ">>           Vector avg search time:\t" << avg_vec
229         << " == " << avg_vec_ns << " ns" << std::endl;
230     std::cout << ">>           List avg search time:\t" << avg_list
231         << " == " << avg_list_ns << " ns" << std::endl << std::endl;
232 }
233 }
234
235 // Will search through more and more data
236 void singleRampUpTests(LinkedList* list, int max_ramps, bool show_results)
{
237     std::vector<std::vector<std::chrono::duration<double>>>
        ramp_test_data;
238     LinkedList* linked_list = list;
239     int maximum_ramps = max_ramps;
240     bool display_results = show_results;
241
242     for (int current_ramp = 1; current_ramp <= maximum_ramps; current_ramp
        ++){
243         srand((unsigned)time(NULL) % std::rand()); // Reseed
244
245         int container_size = pow(10, current_ramp);
246         int search_value = container_size - 1; // Final value
247
248         std::vector<std::chrono::duration<double>> single_ramp_data =
249             singleSearchStructureTest(linked_list, search_value,
                container_size, false);
250
251         ramp_test_data.push_back(single_ramp_data);
252     }
253
254     if (display_results) {
255         std::cout << std::endl;
256         std::cout <<
            "-----"
            << std::endl;
257         std::cout << "           Ramp Test Results"
            " <<

```

```

        std::endl;
258     std::cout <<
        "-----"
        << std::endl;
259     std::cout << "The test ran " << maximum_ramps << " ramps... " <<
        std::endl;
260
261     for (int idx = 1; idx <= maximum_ramps; idx++) {
262         double sci_map = ramp_test_data[idx - 1][0].count(), sci_vec =
        ramp_test_data[idx - 1][1].count(),
263         sci_list = ramp_test_data[idx - 1][2].count(),
264         ns_map = sci_map * pow(10, 9), ns_vec = sci_vec * pow(10,
        9), ns_list = sci_list * pow(10, 9);
265
266         std::cout << std::endl;
267         std::cout << "Ramp " << idx << ": Searched " << pow(10, idx)
        << " elements." << std::endl;
268         std::cout << ">> Unordered Map avg search time:\t" << sci_map
        << "\t== " << ns_map << " ns." << std::endl;
269         std::cout << ">> Vector avg search time:\t" << sci_vec
        << "\t== " << ns_vec << " ns." << std::endl;
270         std::cout << ">> List avg search time:\t" << sci_list
        << "\t== " << ns_list << " ns." << std::endl;
271     }
272 }
273 }
274
275
276 int main(){
277     LinkedList* linked_list = new LinkedList();
278     manyRandomValueTest(linked_list, 4, 1000, true);
279     linked_list->reset();
280     singleRampUpTests(linked_list, 4, true);
281     linked_list->reset();
282     return 0;
283 }

```