Part A) contains( ) Timing in milliseconds

Container	N = 1000	N = 2000	N = 4000	N = 8000
Vector	0.00181351 ms	0.0040415 ms	0.00631671 ms	0.00994213
List	0.00545519 ms	0.0107181 ms	0.0108542 ms	0.0293915 ms
Unordered set	0.0053852 ms	0.0171217 ms	0.0250688 ms	0.056156 ms
Tree	0.00726038 ms	0.0131254 ms	0.0271573 ms	0.0603916 ms

Part B) query( ) Timing in milliseconds

CompareItemName test

Container	N = 1000	N = 2000	N = 4000	N = 8000
Vector	0.144916ms	0.254429ms	0.443696ms	0.709925ms
List	0.143325ms	0.207271ms	0.328567ms	0.714142ms
Unordered set	0.144716ms	0.181491ms	0.350246ms	0.758633ms
Tree	0.125242ms	0.149946ms	0.281658ms	0.620667ms

CompareItemWeight test

Container	N = 1000	N = 2000	N = 4000	N = 8000
Vector	0.0393416ms	0.0670166ms	0.0934375ms	0.141167ms
List	0.03225ms	0.0522125ms	0.0670375ms	0.136629ms
Unordered set	0.0341961ms	0.0480667ms	0.0761625ms	0.160083ms
Tree	0.00055ms	0.000475ms	0.000429ms	0.0004708ms

As expected, the results show the vector and list containers have linear time growth when they call contains() and query(). The unordered_set container performed pretty much just as fast as the vector and the list, which was surprising because I was expecting an O(1) efficiency. The tree also had a linear growth for contains(), which was expected since we have to check every item in the AVL to see if a name matches since we used CompareItemName for this test. All had the same behavior for query() when using the comparator CompareItemName. This was expected again of vector and list inventories and for the tree inventory due to the way it was implemented for this project to search for a name. Again, I expected the hash inventory to run faster, O(1), but I am guessing there might have been collisions affecting its performance. The tree container shows a notably fast performance for query() using comparator CompareItemWeight, likely because it takes advantage of the tree's ordered structure for comparisons. The rest of the containers had a linear growth. I would say use a Hash-based inventory when you prioritize fast exact-match lookups instead of a tree-based inventory. And use a Tree-based inventory when you need efficient range queries or ordered traversals.