Contains Average Time Chart:

Container	Time (N = 1000)	Time (N = 2000)	Time (N = 4000)	Time (N = 8000)
Inventory<Comparator> (Uses a vector)	0.00156338 ms average	0.00324058 ms average	0.00556057 ms average	0.00741921 ms average
Inventory<Comparator, std::list> (Uses a linked-list)	0.00268577 ms average	0.00575064 ms average	0.00814874 ms average	0.0118023 ms average
Inventory<Comparator, std::unordered_set<Item>>	0.00020041 ms average	0.000202515 ms average	0.0002206 ms average	0.000213575 ms average
Inventory<Comparator, Tree>	0.000282505 ms average	0.00035665 ms average	0.000428365 ms average	0.000433315 ms average

Query Average Time Chart:

Container	CompareItemName Time	CompareItemWeight Time
Inventory<Comparator> (Uses a vector)	(N = 1000):0.170529 (N = 2000):0.291554 (N = 4000):0.581375 (N = 8000):0.745391	(N = 1000):0.0110041 (N = 2000):0.0223083 (N = 4000):0.0348166 (N = 8000):0.0435124
Inventory<Comparator, std::list> (Uses a linked-list)	(N = 1000):0.161208 (N = 2000):0.240542 (N = 4000):0.393008 (N = 8000):0.611875	(N = 1000):0.0116916 (N = 2000):0.0226832 (N = 4000):0.0281999 (N = 8000):0.0366706
Inventory<Comparator, std::unordered_set<Item>>	(N = 1000):0.1589 (N = 2000):0.270009 (N = 4000):0.633608 (N = 8000):1.12709	(N = 1000):0.0112749 (N = 2000):0.0243418 (N = 4000):0.0420126 (N = 8000):0.0706584
Inventory<Comparator, Tree>	(N = 1000):0.1183126 (N = 2000):0.361875 (N = 4000):0.986791 (N = 8000):1.63146	(N = 1000):0.000792 (N = 2000):0.000833 (N = 4000):0.000875 (N = 8000):0.000958

Conclusion on Findings:
Overall, most of the tests I saw are what I expected though a couple of the tests did surprise me. The contains test for vector being faster than the linked list made sense because it's a lot easier to traverse through the vector since it's a contiguous block of memory, while linked lists are a little less efficient in traversal, though they weren't too different from one another. However the opposite occurs in the query test which was pretty surprising, again the differences weren't too drastic from one another. As for the hashTable and the Tree, I was not surprised that the hashTable times were all consistent with each other because the time complexity for searching an element in a hashTable is O(1). The Tree's times were also expected because it's an AVL tree that's average case for searching is O(log n). For their query tests, the HashTable was still pretty good but the tree container seemed to be the same and sometimes a bit better than the hashTable, its most likely because HashTables are not the best when it comes to traversing and are mainly good at finding a specific element rather than finding a boundary range. In terms of when it is better to use a HashTable over a tree, you use it when you are checking to see if there is an element contained in the container since as said previously, the case is O(1) while the tree needs to go node by node to find the element. Lastly in terms of when a tree is better than a Hashtable, when you want to get a boundary of elements since trees (especially AVL trees) are better at traversing than keeping ranges secure through balance than Hashtables.