Kelvin Yu

Vector

Runtime - O(N^2)

Precise Runtime - O(N^2) *but lower_bound is a O(log N) function but should not really affect it

Insertion and erase are precisely O(N), but with them being nested inside the main for loop, our overall becomes O(N^2). It will run the third fastest because O(Nlog N) is faster than O(N^2) for larger sample sizes. Vectors are still faster than Lists due it being a contiguous array.

Inserting at the proper place rather than sorting then inserting would be much faster and simpler.

List

Runtime - O(N^2)

Precise Runtime - O(N^2)

Linked list is much slower, due to cache misses. When cpu communicates to RAM, it takes a big chunk and saves to the cache. Since vectors are arrays, their elements are contiguous. Nodes on the other hand could be in different spots so it would have to do much more checking. List is the slowest, but I could have hyper-optimized it to make it constant time. However I wasn’t sure if that was the purpose. Very bad for large sample sizes.

Heap

Runtime - O(Nlog N)

Precise Runtime - O(Nlog N)

Heaps are the fastest. Because our implementation is an array, inserting is O(1) and built in Linear time. Using two heaps, and switching their priorities made it simple to find and pop the median. We also aren’t using any nodes, so again it’s not prone to cache misses.

AVL tree

Runtime - O(Nlog N)

Precise Runtime - O(Nlog N)

AVL trees came in second place but not by a long margin. Only reason is because traversal is more costly and has to account for the left and right, while a heap implemented as an array is more linear. Nlog N is just much closer to N than N^2, and so that’s why even with cache misses it is still fast.