Seconds to **sort**	Program 0	Program 3
125,000 Elements	2.956	0.01
250,000 Elements	11.894	0.021
500,000 Elements	48.597	0.42

Seconds to **search** for y elements in a collection of	Program 1	Program 2	Program 4
800,000 Elements	0.003	2.999	0.001
1,600,000 Elements	0.004	7.298	0.001
3,200,000 Elements	0.04	14.85	0.001

Program 0: Selection Sort
Best: n^2 Worst: n^2 Avg: n^2
Program 3: Built-in Java array sort (quicksort for primitives)
Best: n log n Worst: n^2 Avg: n log n
Program 1: Binary Search
Best: 1 Worst: log n Avg: log n
Program 2: Linear Search
Best: 1 Worst: n Avg: n/2
Program 4: Hash set search (Constant)
Best: 1 Worst: 1 Avg: 1

Searches were each run a handful of times in order to get non-outlier data that roughly matches up with the average result. The final data for each program matches up with the expected results based on their Big O notation.

Program 0 is selection sort, which isn't particularly efficient and struggles with large sets of data, as the number of elements sorted increases, the number of comparisons that need to be made increases exponentially.

Program 3 is Java's built in array sort, which in this example would be quicksort as the lists being sorted are primitive type. This sorting algorithm scales much more gradually than something like selection sort, and only matches selection sort's efficiency in the worst case-scenario.

Program 1 is a binary search, a very standard and relatively efficient sorting algorithm.

Program 2 is just a linear search, which sequentially checks each item in the list for a match. This algorithm also has troubles dealing with large data sets, and decreases in efficiency linearly as the number of elements increases. On average it has to check half the elements in a data set, which isn't ideal when you have a list or database with millions of entries.

Program 4 is a hash set search. This one isn't a search algorithm so to say, as it relies on a specific kind of data set to work. This "search" is able to point to the element immediately, by using it's unique key in the hash set, and thus can find the desired element in the same amount of time, no matter the size of the hash set.