Note: I used http://www.ics.uci.edu/~pattis/ICS-23/lectures/notes/Sorting3.txt for reference.

What algorithm did you use?
I used radix sort. It's like bucket sort for the digits in a number and I used it because high scores are very applicable. High scores are often broken into digits, from the largest digit to the smallest, which is perfect for radix sort.

How does this algorithm scale?
Radix sort goes through the "digits" in a number, from the ones place to however high and the numbers are sorted by place, at the final digit, all the numbers will be sorted. Here's how it works.

If I have an array of high scores:
535, 235, 745, 633, 754, 252, 632

I would have another array for the places 0 - 10, with a queue in each slot, and add the numbers from left to right based on the "ones place"

0	1	2	3	4	5	6	7	8	9
		252	633	754	535				
		632			235				
					745				

Then I would put it back into the original array from slots 0 - 9 and repeat the process for the "tens place"

252, 632, 633, 754, 535, 235, 745

0	1	2	3	4	5	6	7	8	9
			632	745	252				
			633		754				
			535						
			235						

Put them back into the array.
632, 633, 535, 235, 745, 252, 754

Now do process again for the "hundreds" place.

0	1	2	3	4	5	6	7	8	9
		235			535	632	745		
		252				633	754		

Put back into array.
235, 252, 535, 632, 633, 745, 754

In the end, the numbers are sorted. What's important to note that, if the places don't change for the number, those numbers in the array will remain in the same order, so the sorting is stable. Radix sort requires O(N) of space for holding the data in the queue and also the original array.

For the running time, the sort needs to do N movements for the values to the array with the queues and then another N to move it back to the original array. All this times the number of digits. For the highscore program, if we have N numbers to sort, from 1 to N, then there are log10 N digits, meaning the operations are done log10 N times resulting in the complexity class to be O(N log10 N)

So radix sort is very good for this, better than a lot of other sorting methods such as selection sort O(n^2), merge sort O(n log2 n), though quick sort would probably be quicker.

Can you reduce its time and space complexity?
Radix sort needs O(N) for the array and O(N) for the array with queues.
The number of loops above can be cut in half if we have a radix of 100 instead of 10, meaning it goes by two digits at a time. This would increase space in the queues though. Obviously, it would make sense to have larger radixes for highscores with larger numbers of digits. An issue

I had was dealing with decimals, and the way I dealt with that was to multiply the numbers by 100 then divide by 100 later on. That right there is another O(2N).