David Hamlin

Program 1

	N = 100000	N = 200000	N = 400000
BubbleSort	47.014	188.056	752.224
CocktailSort	39.957	159.828	639.312
RadixSort	.072	.144	.288

BubbleSort = O(N^2). This algorithm iterates through and compares the value at index i with the next value. If it is bigger, then it switches it. This means that after each run through, the largest value will end up at the end. In mine, I made it so after each run-through, the amount it iterates by decreases by 1 because the largest value will be in the end of the array anyways. The runtime makes sense, as n is 100000 and it is O(n^2).

CocktailSort = O(N^2). This is exactly like BubbleSort, except as well as iterating through forwards and comparing, it iterates through backwards at the same time. The runtime makes sense, because while it is also O(N^2), it is considerably less than BubbleSort due to being two tailed.

RadixSort = O(Nk). This algorithm does not compare values. Instead, it goes by each digit in the longest element, which is k characters long. In this case, k = 6, so the algorithm would iterate 6 times. If k = 255, then the runtime would be considerably longer. Each value of k, it does count sort, which in the end puts all of the elements in order. The runtimes are linear, as each time N doubles, the runtime also doubles, hence O(Nk).

Predicting the Runtime using = a * O(N)

BubbleSort = 47.014 = a * 100000^2
A = 4.7014e-9
A * 200000^2 = 188.056
A * 400000^2 = 752.224

CocktailSort = 39.957 = a * 100000^2
A = 3.9957e-9
A * 200000^2 = 159.828
A * 400000^2 = 639.312

RadixSort = .072 = a * 6 * 100000
A = 1.2e-7
A * 200000*6 = .144
A * 400000*6 = .288