Bryan Arnold

CSE 3500

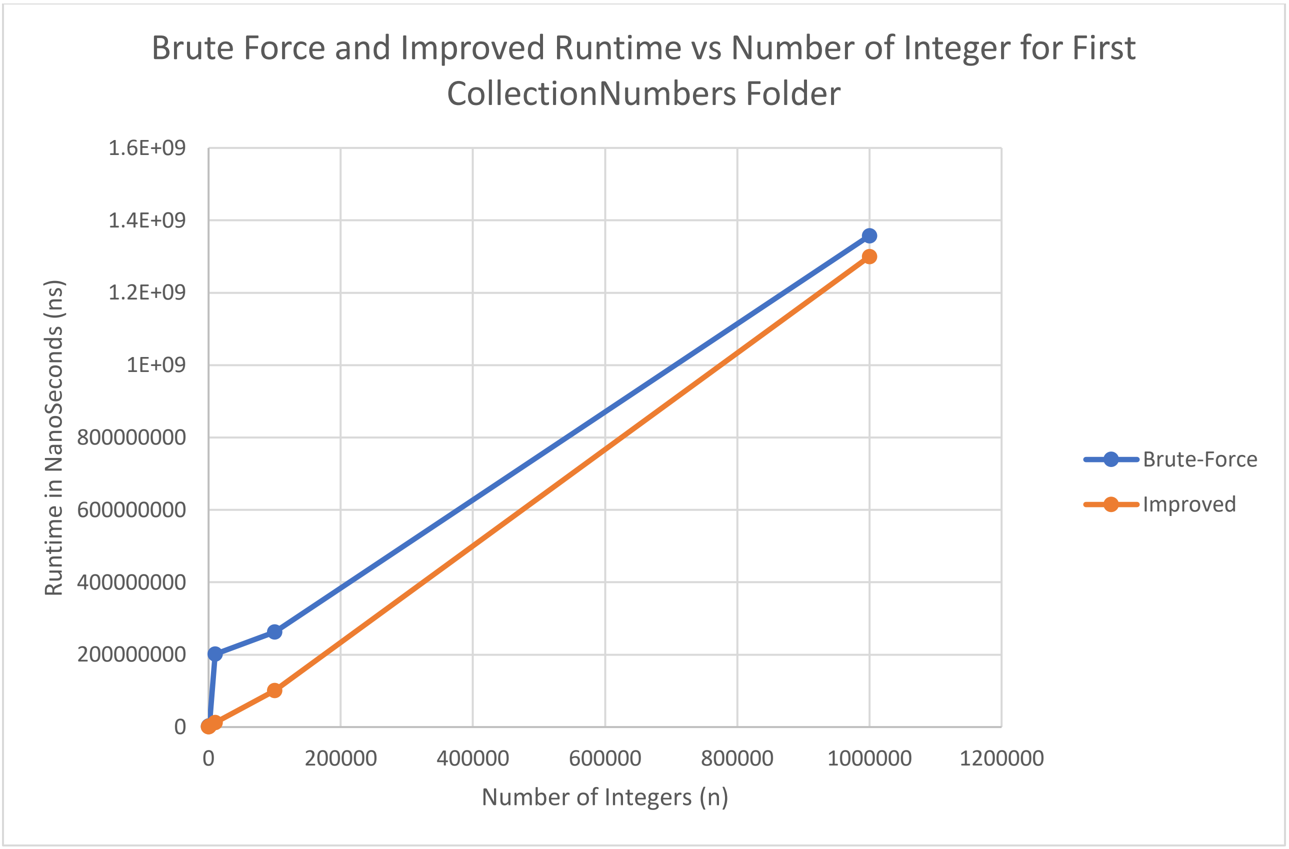
Assignment 1 Report

2/8/17

Settings: The language I used was Java in the Eclipse compiler given by Oracle. My machine is an Asus Laptop with 64-bit operating system (Windows 10), Intel(R) Core™ i7-4710HQ CPU running at 2.50 GHz. My RAM is 16.00 GB and my hard drive has 1.395 TB of memory (with 943 GB remaining available).

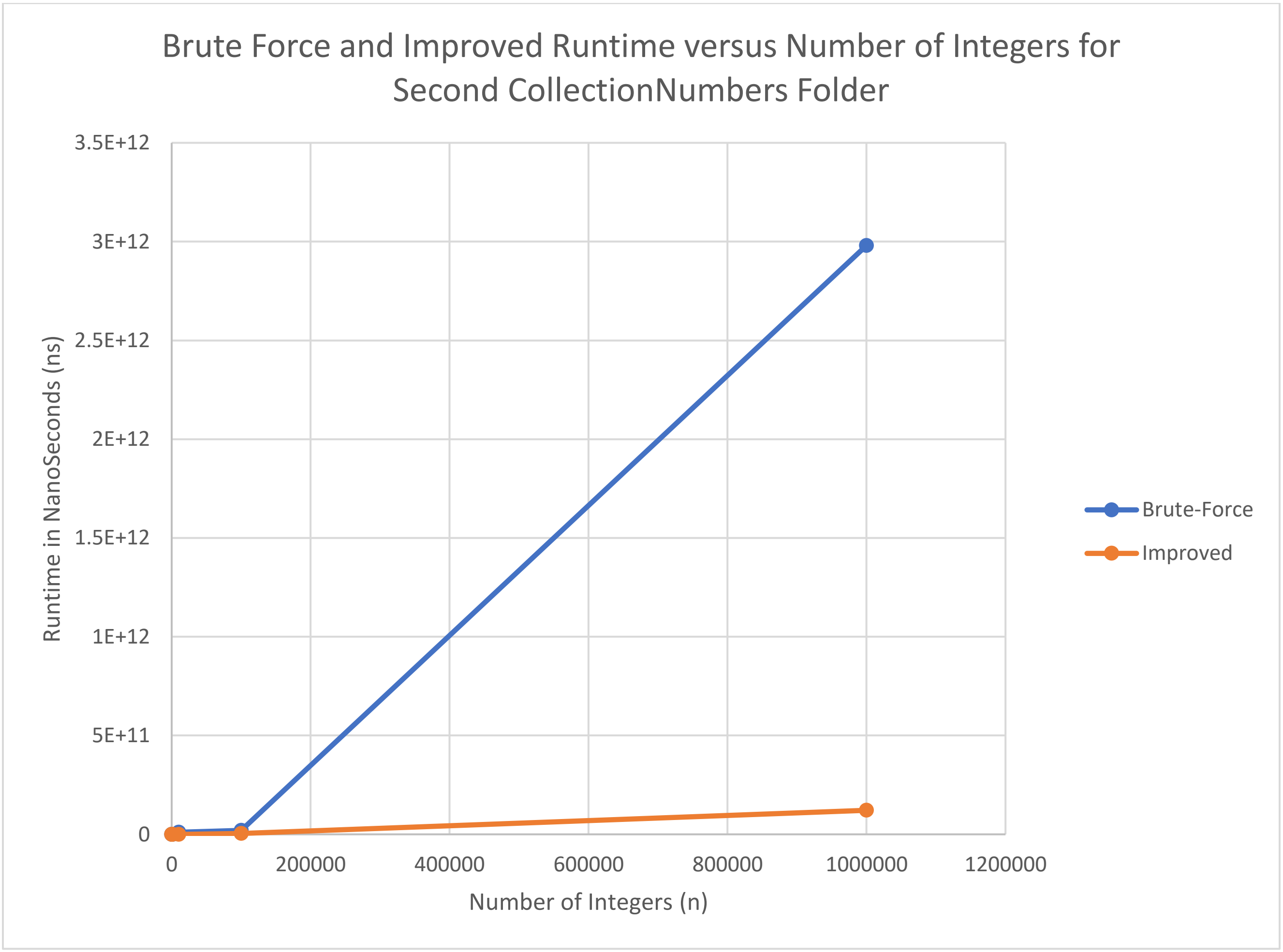
Results:

The following graphs are my results for solving the two integers sum problem on question 3 of homework two for two different algorithms, brute-force style and binary search style:



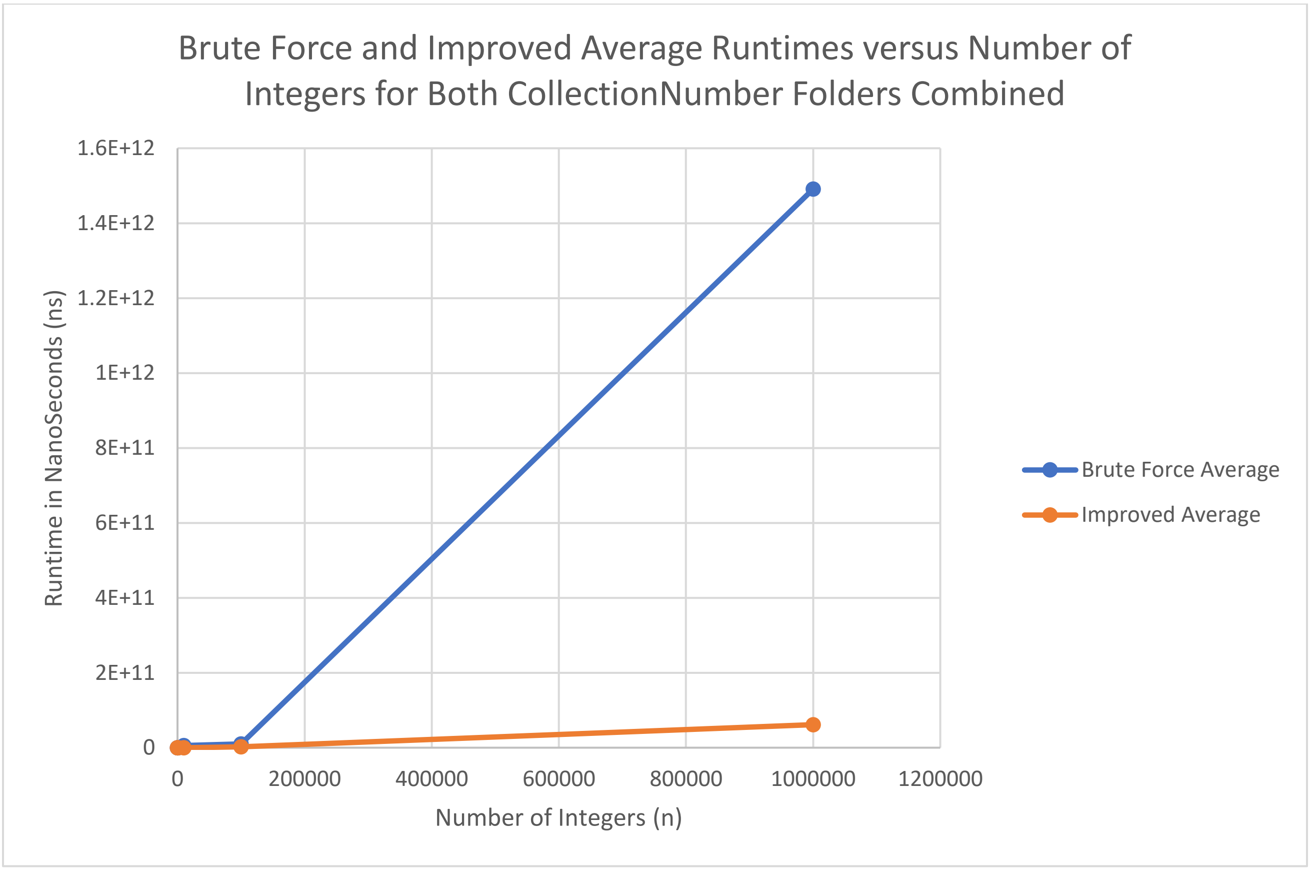
Graph 1: This graph represents the runtime for both algorithm designs versus the number of integers in the given array for the first collection folder. As you can see, the improved method is faster, although not by a huge margin. This can probably be attributed to the fact the solutions

array given only has a few integers to find compared to the next collections folder. Each method only had to find a few integers that have a sum in the solutions array, so naturally the runtime will not be as high. The brute force method runs at O(n^2) time, while the improved version runs at O(nlogn) time. This means the graphs are correct in how they look, as the theoretical runtime matches the found runtimes, and O(n^2) will run longer than O(nlogn). Note: a factor that can lead to more runtime is each time I display true or false for each number. This isn't necessary, but I chose to keep it for being a useful observation tool, and will increase the runtime by some factor. Also, note: the runtime varied slightly each run-in Nano seconds, so some random error exists.



Graph 2: This graph represents the runtime for both algorithm designs versus the number of integers in the given array for the second collection folder. As you can see, the improved method is significantly faster for the integers given in the second collection. This can be attributed to the fact the solutions array given only has many more integers in it to check if the given array has a sum equal to a solution than to the first collections folder. Each method only had to find a lot more integers that have a sum in the solutions array, so naturally the runtime will much higher. The brute force method runs at O(n^2) time, while the improved version runs at O(nlogn) time. This means the graphs are correct in how they look, as the theoretical runtime matches the found

runtimes, and O(n^2) will run longer than O(nlogn). Note: a factor that can lead to more runtime is each time I display true or false for each number. This isn't necessary, but I chose to keep it for being a useful observation tool, and will increase the runtime by some factor. Also, note: the runtime varied slightly each run in Nano seconds, so some random error exists.



Graph 3: this graph is the average of the runtimes from both collection folders versus the number of integers in the given array. This graph is almost the same as the second graph, which is expected, showing that the improved method is much quicker than the brute force method as the number of integers used gets bigger. The brute force method runs at O(n^2) time, while the improved version runs at O(nlogn) time. This means the graphs are correct in how they look, as the theoretical runtime matches the found runtimes, and O(n^2) will run longer than O(nlogn). Note: a factor that can lead to more runtime is each time I display true or false for each number. This isn't necessary, but I chose to keep it for being a useful observation tool, and will increase the runtime by some factor. Also, note: the runtime varied slightly each run in Nano seconds, so some random error exists.

Here are my data tables used to create these graphs, they are self-explanatory:

Collection Folder	Brute Force			Improved	
(1 or 2)	NanoSeconds (ns)	Number of Integers (n)		NanoSeconds (ns)	Number of Integers (n)
Collection Folder 1	1580194	10		849831	10
Collection Folder 1	2179182	100		1090001	100
Collection Folder 1	3142734	1000		2511313	1000
Collection Folder 1	201555808	10000		11913652	10000
Collection Folder 1	262092302	100000		100757582	100000
Collection Folder 1	1357287223	1000000		1299890015	1000000
Collection Folder 2	4433492	10		3136987	10
Collection Folder 2	4711561	100		3581198	100
Collection Folder 2	113640534	1000		10663127	1000
Collection Folder 2	11035019216	10000		110679672	10000
Collection Folder 2	19408905345	100000		4578909981	100000
Collection Folder 2	2.98056E+12	1000000		1.21658E+11	1000000

Collection Folder	Brute Force (Average ns)	Improved (Average ns)	Number of Integers (n)
(1 or 2)	NanoSeconds (ns)	NanoSeconds (ns)	
Collection Folder 1 + 2	3006843	1993409	10
Collection Folder 1 + 2	3445371.5	2335599.5	100
Collection Folder 1 + 2	58391634	6587220	1000
Collection Folder 1 + 2	5618287512	61296662	10000
Collection Folder 1 + 2	9835498824	2339833782	100000
Collection Folder 1 + 2	1.49096E+12	61478845882	1000000

Conclusion:

The reason why the choice of an algorithm matters can clearly be seen in the graphs and data tables. The larger the data size got, the longer it took. The brute force method was significantly longer than the improved version, O(n^2) vs O(nlogn), so the time in which the overall program ran was significantly longer. We want fast programs that are accurate as well. So, in choosing an efficient algorithm that is more complex to design, yet is much faster, we create an overall program. Brute force methods are easy to make, but are far to slow for what we want. So, in conclusion, the algorithm design choice matters significantly because we want efficient, fast, and accurate running programs to run tasks, so one has to be created to maximize these desires.