Benchmark Report

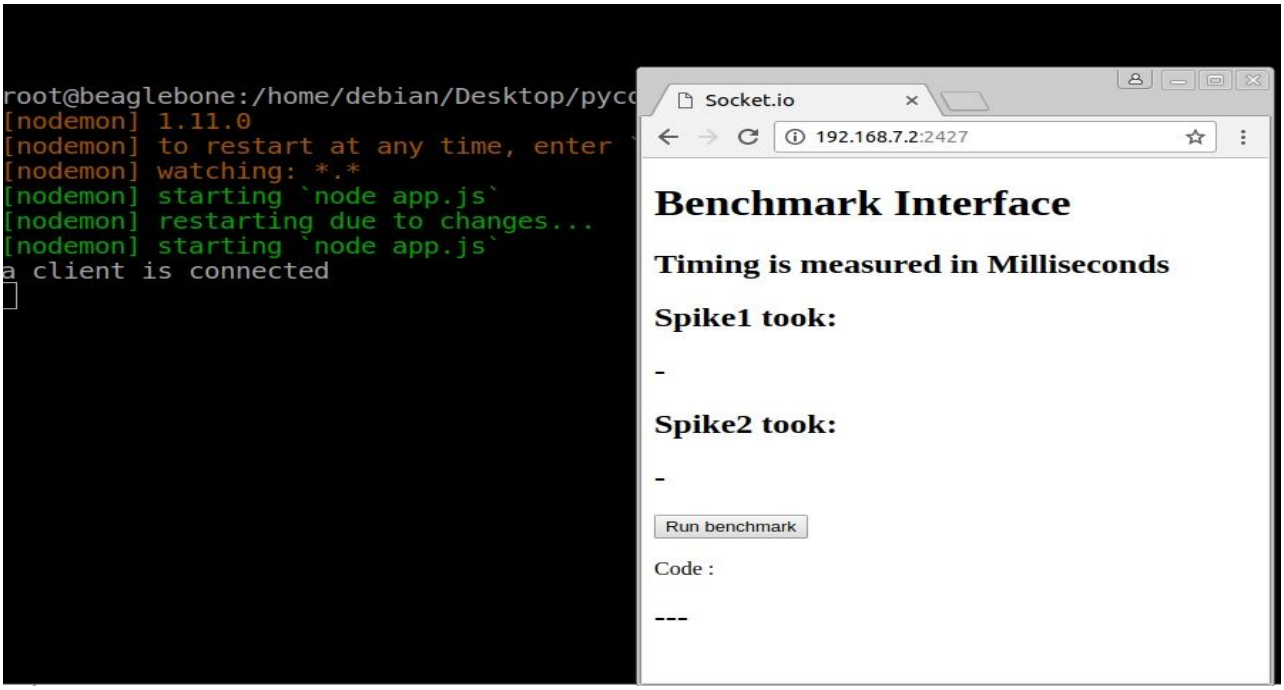
Introduction

The Morse code application works by getting motion data in the form of “1” indicating a long motion or “0” indicating a short motion from the PIR motion sensor to form a combination. Then, this combination is passed to a dictionary, which checks if the given combination matches any of the code. If so, the dictionary will return a letter which is presented to the user. In this report, we will compare the performance of two data structures to store the dictionary and the motion combination, the first benchmark will be using a string to store the codes and the second one will be using an array.

Benchmarking procedure

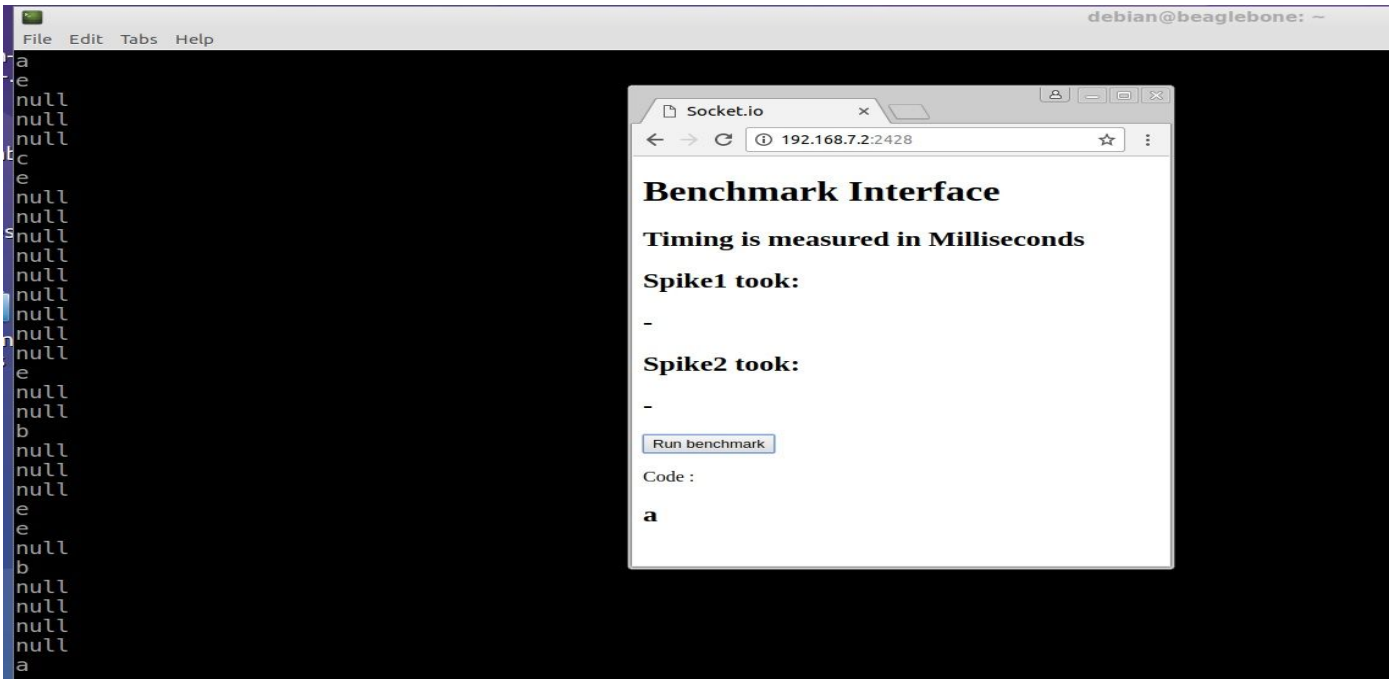
The benchmark is done in three rounds, having **10,000**, **50,000** and **100,000** of input data in a list to feed the dictionary. It is assumed that you already have a dictionary table that has at least five letters and their respective code in the script. Firstly, we append n random data, in the format of ones and zeros “0110”, to the feeding list where n is the size depending on the round. Then, record the starting time and for each data in the list, check if the data matches any code in the dictionary. If a match is found, we print the letter received to the console and send the letter to the client to be presented on a webpage, else repeat the previous step with a ‘null’(refer to “In process” photo below). Wait until all the data in the input list is passed before recording the ending time and calculating the duration in milliseconds for spike 1 by using the formula duration = end time – start time. Currently the test for the first benchmark (using string) is done. Hence, we convert the data in the input list and the codes in the dictionary to a list instead of a string. For example, ‘0010’ will be converted to [‘0’,’0’,’1’,’0’]. After the conversion is done, we do the same procedures as for the first benchmark. Finally, the duration of both benchmarks is printed in the console and sent to the client side to be viewed on the web page (refer to “Result” photo below).

Client connects:



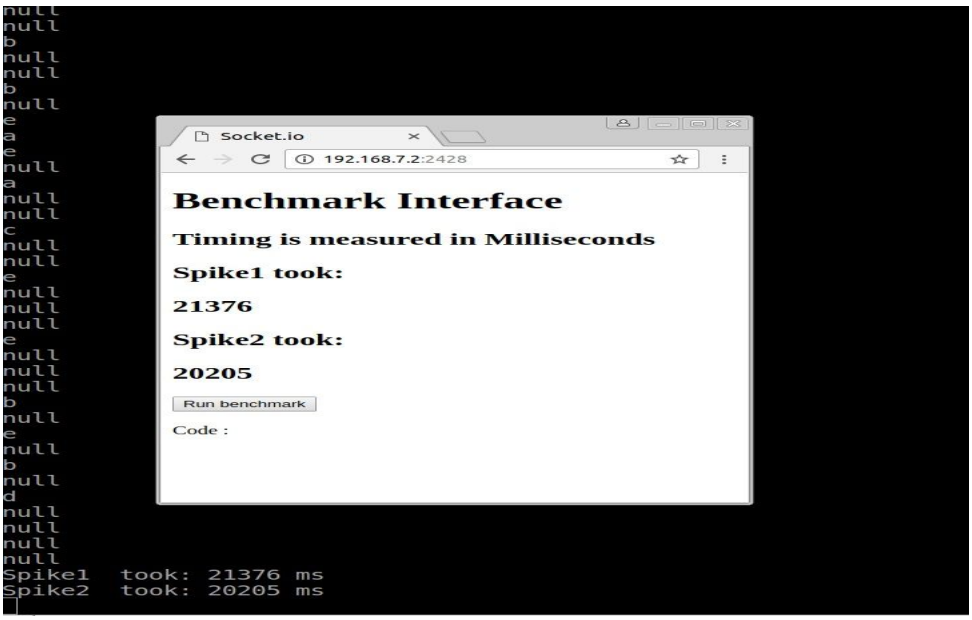
This photo shows what happens when you run app.js and a client connects to localhost:2427

In process:



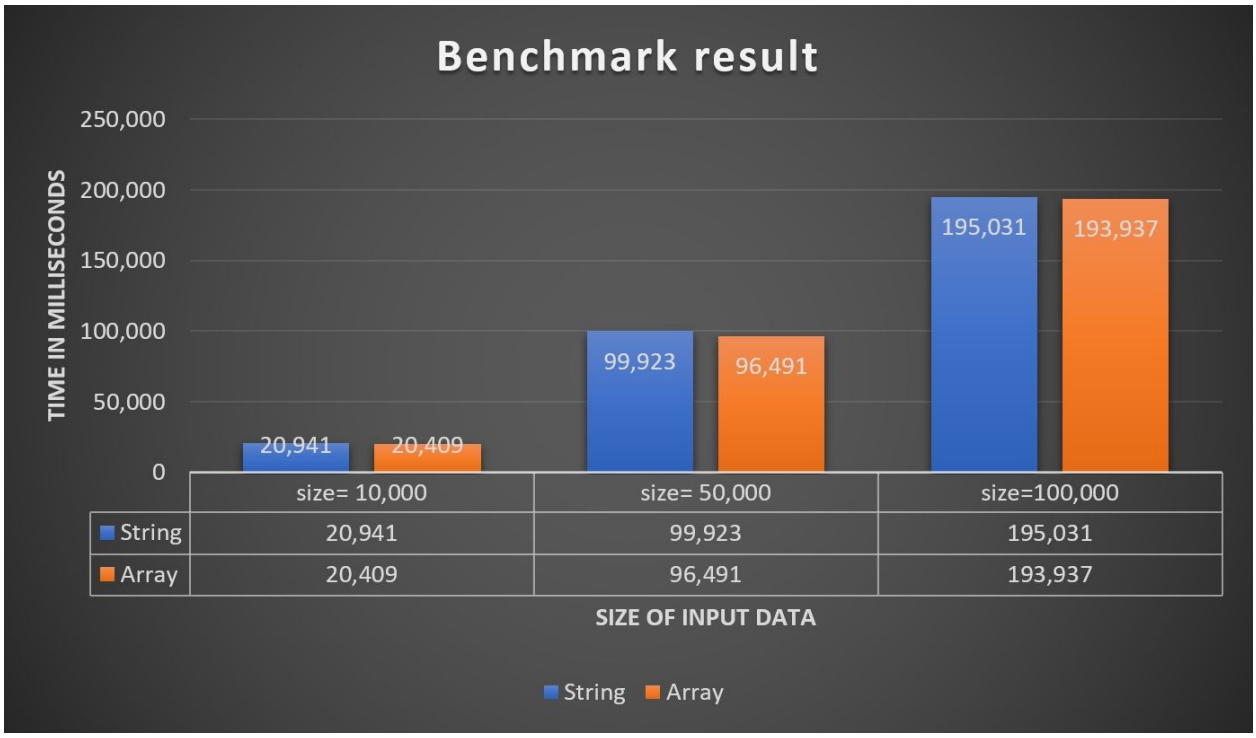
This photo shows what the interface and console look like during the process of benchmarking

Result:



This photo is a screenshot of the state after the two benchmark is finished running

Benchmark results



The benchmark graph above illustrates the all runtime durations in *milliseconds*.However, for simplicity, the graph will be explained in seconds.When the dataset size is 10,000, storing the data as strings ('0010') or as an array (['0','0','1','0']) would not make a significant difference in performance as the string method takes 20.941 seconds while the array Is slightly better as it takes 20.409 seconds. However, as the dataset size gets larger, the difference gap gets wider as the time difference between them is more significant. For example, the comparison between the two methods when the datasize is 50,000 shows that array method is 3.432 seconds faster than the string method as it only takes 96.491 seconds to execute while the string method takes 99.923 seconds. Finally, when the dataset size is 100,000 the time difference between the two methods becomes more significant as the string method takes 195.031 seconds to compute while the array method only takes 193.937 seconds.

Test conclusion

As illustrated in the result section, storing the input information and the codes in the dictionary as arrays yield faster processing time especially when the datasize is large. Hence, it is recommended to use store the data in arrays for the upcoming morse code app.