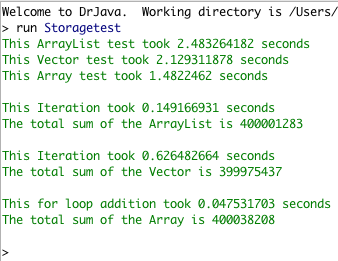
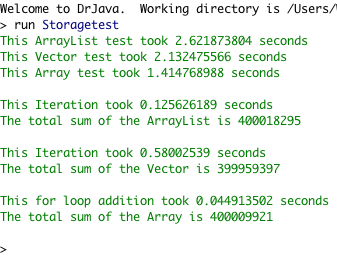
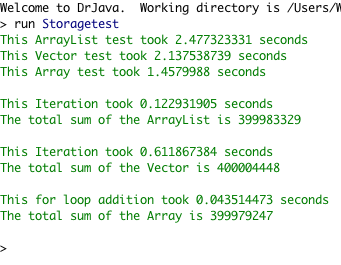
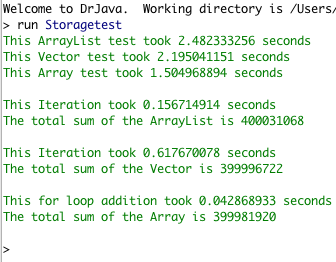
Storage Analysis

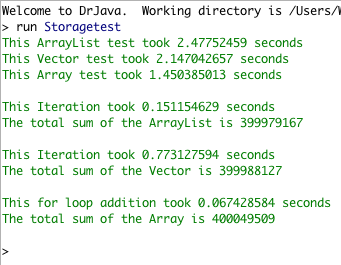
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Attempt | Storage time (seconds) | | | Addition time (seconds) | | |
|  | 100000000 numbers | 150000000 numbers | 200000000 numbers | 100000000 numbers | 150000000 numbers | 200000000 numbers |
| ArrayList | 2.621873804 | 3.716740692 | 5.239470118 | 0.125626189 | 0.183326237 | 0.229150543 |
|  | 2.477323331 | 3.661434603 | 5.32234802 | 0.122931905 | 0.164926678 | 0.242024812 |
|  | 2.483264182 | 3.747025711 | 5.279090871 | 0.149166931 | 0.183588017 | 0.215265117 |
|  | 2.482333256 | 3.740975216 | 5.390120807 | 0.156714914 | 0.226398635 | 0.217971414 |
|  | 2.47752459 | 3.886739059 | 5.224701724 | 0.151154629 | 0.202950985 | 0.229667488 |
| Vector | 2.132475566 | 3.617996047 | 6.980501657 | 0.58002539 | 0.884696853 | 1.134622541 |
|  | 2.137538739 | 3.456857301 | 24.696971459 | 0.611867384 | 0.912935027 | 1.147658437 |
|  | 2.129311878 | 3.628462375 | 6.94227647 | 0.626482664 | 0.89170476 | 1.161246556 |
|  | 2.195041151 | 3.378951662 | 7.106537779 | 0.617670078 | 0.916687427 | 1.18546939 |
|  | 2.147042657 | 3.89563115 | 26.359870209 | 0.773127594 | 0.952667849 | 1.124716991 |
| Array | 1.414768988 | 31.65704702 | 2.804104284 | 0.044913502 | 0.06152839 | 0.074102791 |
|  | 1.4579988 | 29.061236093 | 2.748688965 | 0.043514473 | 0.058452753 | 0.078104009 |
|  | 1.4822462 | 18.936930529 | 2.727189901 | 0.047531703 | 0.059625779 | 0.077325574 |
|  | 1.504968894 | 21.062018663 | 2.726672591 | 0.042868933 | 0.061434458 | 0.079209754 |
|  | 1.450385013 | 31.968872852 | 2.725143713 | 0.067428584 | 0.092188678 | 0.07826707 |

The purpose of this assignment question was to code and analyse three different java storage devices and measure their time duration of adding large collections of numbers. After I coded the assignment my results turned out to be quite suprising. What surprised me the most were the outliers in my data. For example when adding 200 000 000 numbers to a vector most of the time it took roughly 7 seconds to implement, however there were times, as you can see, where it would take upward of 25 seconds. After strenuous and repetitive testing the overall conclusion of the data collected points to an ArrayList being the most efficient method of number storage. As you can see by the graph below ArrayList has a steady increase in time it took based on an increase in numbers added. As opposed to the other forms of storage which had a very unreliable timing to collection size correlation. Seeing as most storage solutions are not working within a finite basis and should always be flexible to grow and be reliable while doing so our overall tip to designers would be to stick with an ArrayList, as made very clear by the data given below

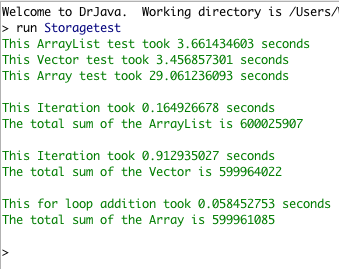
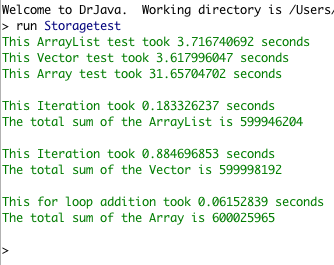
**This set of screenshots contains all the outputs for 100 000 000 number storage**

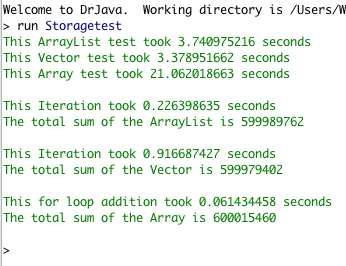
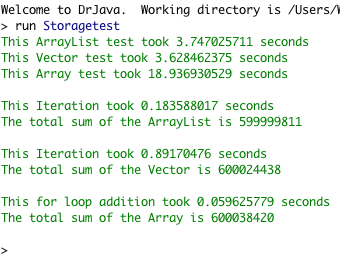


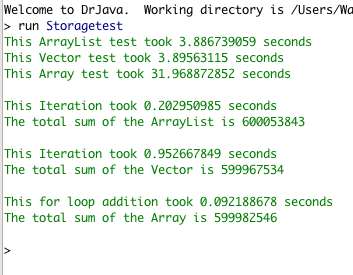
 



**These next set of screen shots contain all the outputs for my 150 000 000 number storage**







**These next set of screen shots contain all the outputs for my 200 000 000 number storage**

