COMP 370 - Assignment 1

Friesen, Barbara (300184589)
Gill, Riya (300176410)
Shergill, Amar (300130878)
Nand, Ryan (300138674)
Dedhia, Vatsal (300183606)

1. Performance Analysis of Design Options

Introduction

In order to determine whether polar coordinates or cartesian coordinates be stored in a program that would require extensive rotations of point and computations of distance we designed 4 different designs to analyze which would be the most efficient. The first design only stores polar coordinates (PointCPD2), the second stores both (PointCPD4), and the other two were based on the first with an abstract superclass. For PointCPD4 and PointCPD54's constructor we decided to only ask for one type of coordinate although it stores both, and depending on what was entered the constructor would calculate and store the other coordinate type to ensure accuracy of the point values. We decided that for each design we would run it at 4 different loop sizes (100000, 1000000, 10000000, and 100000000) 6 times each. Doing 6 trials for each loop size helps to validate the average that we found by limiting the chance of fluke factors such as the processor being busy with other tasks which could increase the processing time. We also ensured that all design processes were run on the same computer to ensure the same environment and machine specifications that could also impact the accuracy of our findings.

In order to automate our loops we generated a random number for generation of new 'points' which we just used rho (0-200) and theta (0-360) for the generation of new objects because that is a common aspect for all designs. To automate the loop calculating the distance we created both points before the loop and the timer began so that the time we got would not include object construction and we could just focus on the calculation time. This means that the same distance was calculated over and over again in the loop though we decided that the difference of time we would get through having different distances would be non-existent given the large loop sizes. For the rotate point function we created a new point and a random theta value before the loop and timer began, however the calculations were different for each loop due to the returning point from the function overwriting the point that was rotated. We believe these automated methods were accurate to measure the processes of interest in order to find which design is the most efficient for the intended usage

PointCPD2							
Calculate Distance (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	8	8	8	8	9	8	8.17
1000000 Loops	53	74	55	56	54	64	59.33
10000000 Loops	493	519	499	500	498	496	500.83
100000000 Loops	4989	4999	5037	4992	5064	4989	5011.67
Rotate Point (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	23	22	24	24	26	22	23.5
1000000 Loops	179	207	185	180	180	184	185.83
10000000 Loops	1740	1810	1745	1733	1776	1800	1767.33
100000000 Loops	16533	16659	16675	16717	16699	17323	16767.67

Design 2 only stored the polar coordinates which in both cases of distance calculation and rotation point increase linearly.

PointCPD4							
Calculate Distance (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	3	2	2	4	2	3	2.67
1000000 Loops	5	6	6	8	5	6	6
10000000 Loops	5	5	5	4	5	5	5.17
100000000 Loops	6	5	5	5	5	5	5.17
Rotate Point (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	21	21	22	23	21	21	21.5
1000000 Loops	144	147	144	161	145	146	147.83
10000000 Loops	1400	1402	1395	1391	1440	1405	1405.5
100000000 Loops	12813	12827	12853	12761	12826	12930	12835

Design 4 stores both polar and cartesian coordinates. Design 4 runs much faster than Design 2 in terms of calculating distance. However, it’s exponential increase is not as large as Design 2.

PointCPD52							
Calculate Distance (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	8	8	8	8	10	8	8.33
1000000 Loops	53	54	53	53	52	54	53.17
10000000 Loops	495	496	505	492	522	496	501
100000000 Loops	4952	5005	5077	5011	5162	5018	5037.5
Rotate Point (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	22	24	23	24	26	25	24
1000000 Loops	179	180	178	178	178	180	178.83
10000000 Loops	1738	1782	1752	1745	1745	1762	1754
100000000 Loops	16548	16770	16632	16750	16794	16692	16697.67

The following dataset represents Design 2 but using an abstract superclass. Which comparatively run similarly when calculating the distance however Design 5.2 is more efficient at at point rotation.

PointCPD54							
Calculate Distance (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	3	3	2	3	3	2	2.67
1000000 Loops	5	5	5	6	5	5	5.17
10000000 Loops	5	5	5	5	6	5	5.17
100000000 Loops	5	6	6	6	5	5	5.5
Rotate Point (ms)							
	Runs						
Loops	1	2	3	4	5	6	Avg.
100000 Loops	21	21	22	20	21	21	21
1000000 Loops	145	143	146	147	144	147	145.33
10000000 Loops	1387	1414	1388	1393	1406	1387	1395.83
100000000 Loops	12762	12879	12864	13640	12737	12837	12953.17

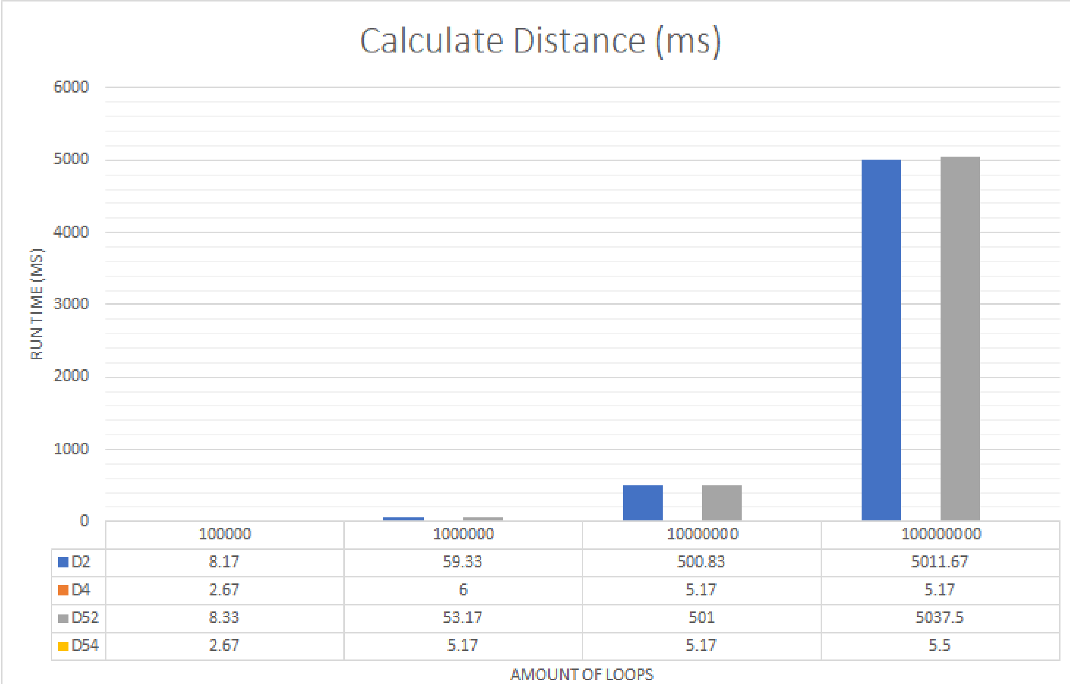
Likewise this dataset is Design 4 though uses an abstract super class. Once again the distance calculation time is similar however, the rotation point run time performs much more poorly than the original Design 4

Sample Output

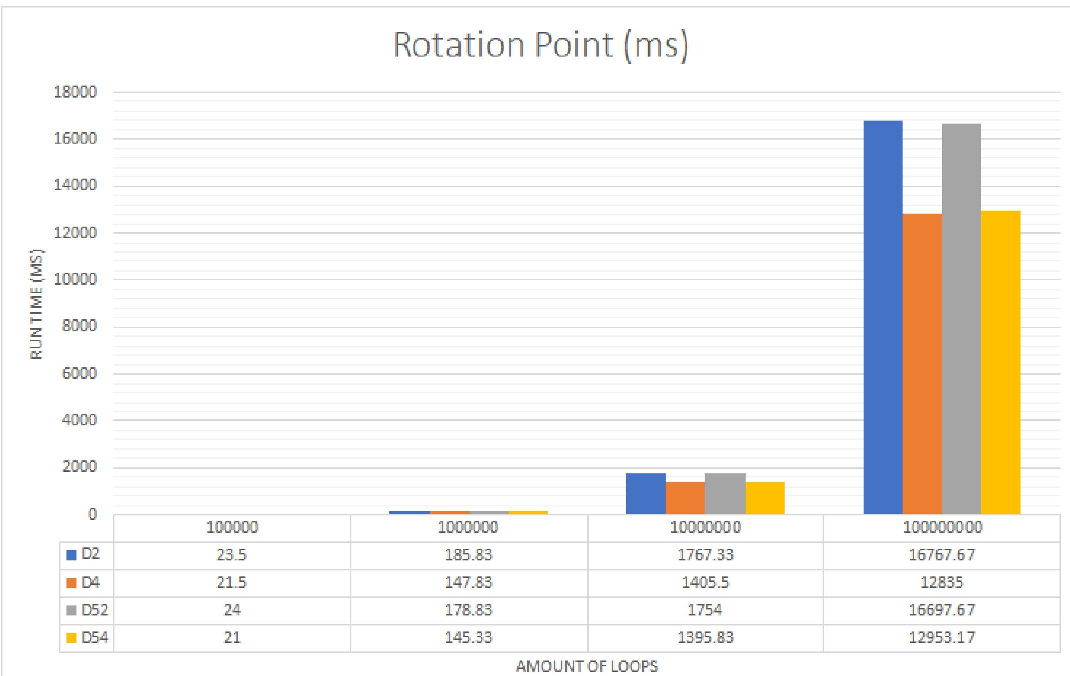
```
Distance time is: 5005
Rotation time is: 16957
```

Note: This was taken from execution of PointCPD2Test with loop size of 100,000,000. This data does not appear in the table due to forgetting to take a screenshot earlier. However as displayed the values shown are within the range of the values recorded. The other test classes have the same display of output except of course the time values.

Graphical Representation



	100000	1000000	10000000	100000000
D2	8.17	59.33	500.83	5011.67
D4	2.67	6	5.17	5.17
D52	8.33	53.17	501	5037.5
D54	2.67	5.17	5.17	5.5



	100000	1000000	10000000	100000000
D2	23.5	185.83	1767.33	16767.67
D4	21.5	147.83	1405.5	12835
D52	24	178.83	1754	16697.67
D54	21	145.33	1395.83	12953.17

Conclusion

The various designs store the coordinates using different techniques, yielding assorted running times in terms of their rotation point and distance calculation speed. Firstly we will go over the distance calculations, as seen in the datasets and graphs Design 4 has a much faster running time than Design 2. This is due to Design 2 having to compute the cartesian coordinates on demand, which takes more time than Design 4 which simply returns the values. The abstract classes follow concurrently with their original, the time it takes to calculate the distance is not much different.

As for the rotation point running times, Design 2 and 4 behave similarly in the smaller loop amounts though differently when dealing with larger amounts of rotations. Design 2 increases exponentially more than Design 4, this may be once again caused by the on demand computation of the cartesian coordinates in Design 2. As for the abstract superclass designs(5.2 and 5.4), they compute much more efficiently than their originals. This is due to the nature of the abstract class. It is reusable code in which it doesn't need to be reinstated over and over again, thus saving a significant amount of time when computing. And as before we Design 5.4 is more efficient than Design 5.2 that has to compute the cartesian coordinates on demand.

As the data presents, it is seen that having the polar and cartesian coordinates simply return is a much better option for efficiency in terms of calculating distance and rotation point. Not having to compute coordinates results in a lower running time. That combined with abstract superclasses that don't have to be reinstated every time a class is invoked produces the most ideal results for computations running time. Thus Design 5.4 is the most ideal design in this scenario.

2. Arrays

Introduction

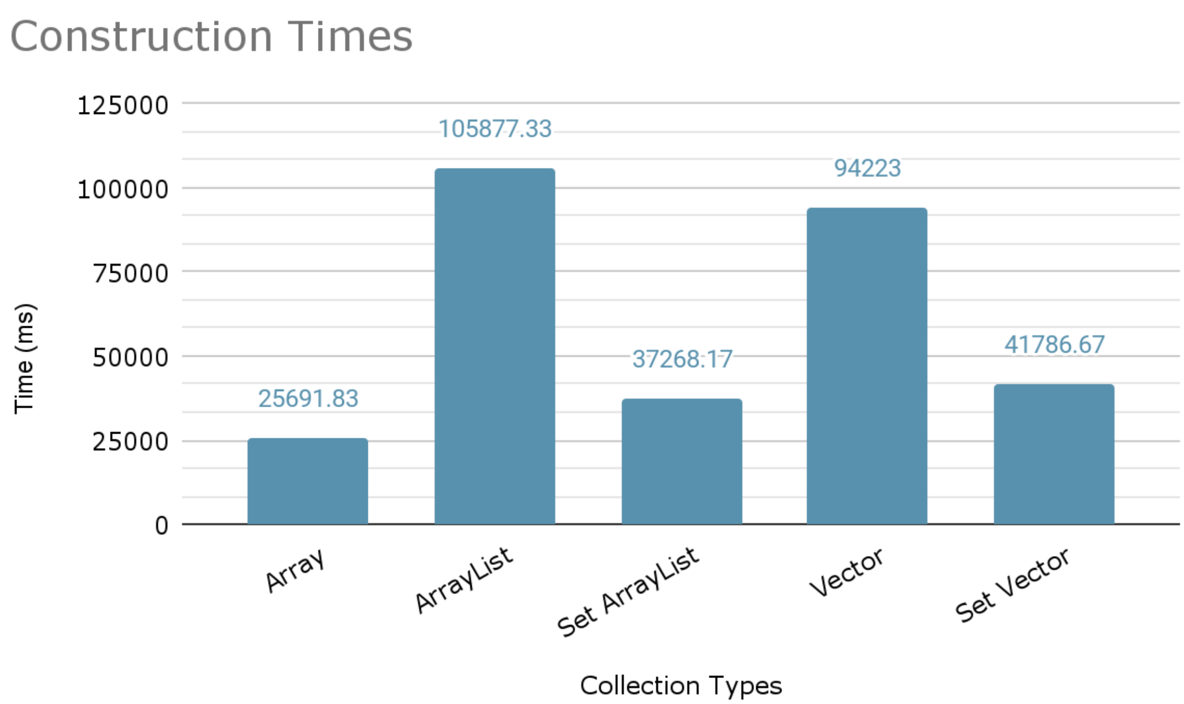
Here we assessed the execution time of three data structures: Arrays, ArrayList and Vector. We accomplished this by creating a large collection of random integers running in value from zero to nine which were equal for each of our three data structures. We measured the construction time of the Array, ArrayList, and Vector, though since there are two different ways to instantiate the ArrayList and Vector (grow with addition or create a set size), we measured and recorded both. We then proceeded to run the program several times on the same computer in order to better obtain a more appropriate average without risk of having machine specifications mess with the times. We decided to do 6 runs for each process in order to get an accurate average, doing 6 runs lessens the chance of random factors like busy processors that can affect the average.

In our design we created a large collection of random integers, 0-9, for our three data structures. We then used a for loop to sum the Array and an iterator to sum the ArrayList and Vector. After we calculated the elapsed time by subtracting the start time from the stop time using the currentTimeMillis() Java method. We created two separate methods in our code, one that would time construction of a collection for each of the 5 ways mentioned before and print out the result, and the second function that would construct the three collections and then time the addition of the collections and print out the result. Although it is a bit redundant to have the collection constructed twice, we feel that this method is the easiest way to prevent external variables from affecting our processing times and get the most accurate results.

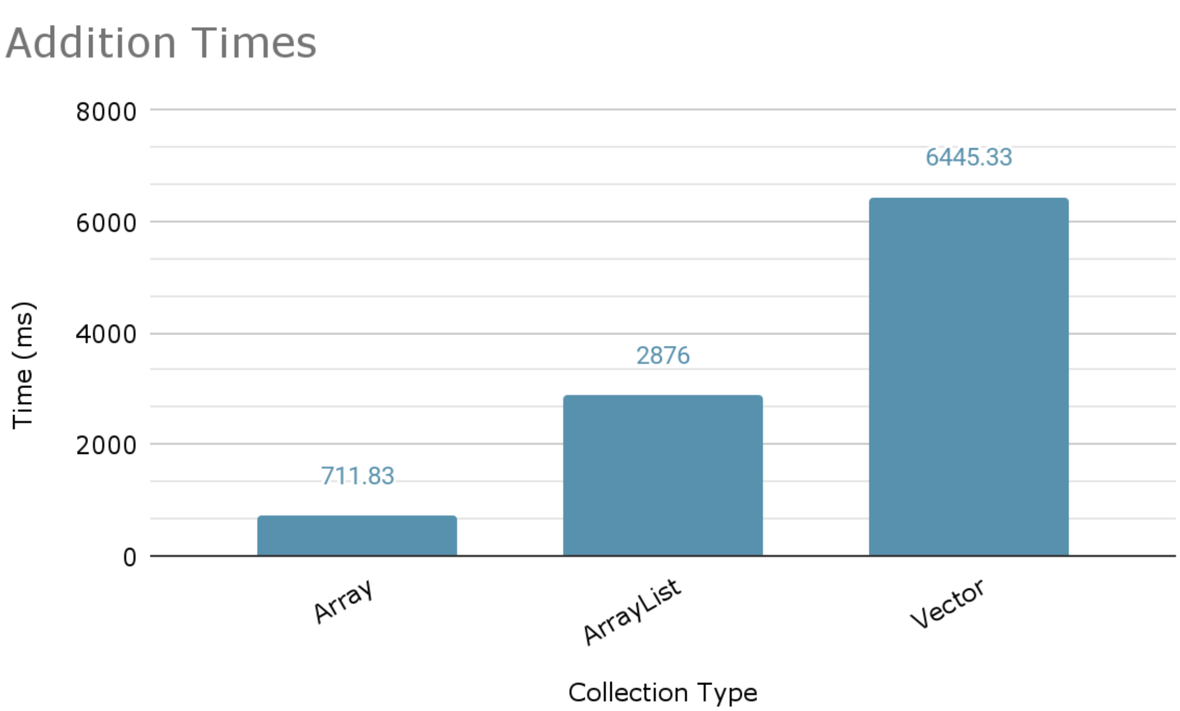
Collections		**Runs (ms)**						
Type	Timing	1	2	3	4	5	6	Avg
Array	Construction	25714	25726	25728	25659	25666	25658	25691.83
	Addition	704	712	709	712	714	720	711.83
ArrayList	Construction	105892	105489	106362	106017	105510	105994	105877.33
	Set Size Construction	36918	37238	37476	37391	37390	37196	37268.17
	Addition	2864	2864	2887	2880	2879	2882	2876
Vector	Construction	90286	93962	92003	99109	94303	95675	94223
	Set Size Construction	41472	41690	42079	42013	41592	41874	41786.67
	Addition	6420	6495	6469	6413	6418	6457	6445.33

Graphical Representation:

1. Bar Graph depicting the construction time of the three data structures:



2. Bar graph to show the time taken by each data structure to add the elements



Conclusion

Comparing the construction time of the three data structures, it can be seen that arrays are constructed faster than both arrayList and Vector. This is because for adding an element at the end of an array, the time complexity,i.e, the big O notation is O(1).
For ArrayLists and Vectors, the big O notation is O(n), where n is the number of elements. Although the time complexity for both the data structures is same, arrayLists take less time, as can be seen from the construction time given above. This is because ArrayLists are non-synchronized and vectors are synchronized.

From the data above, it can also be inferred that arrays take the least amount of time to add all the elements present in the data structure as compared to ArrayList and Vector. But, arrays are not recommended to be used when large data is involved. The time for arrays is the fastest because arrays are of constant size.The size of an array cannot be changed whereas ArrayList and Vectors can change in size. Thus, arrays are not preferred when the number of elements is not known, or when the developer does not know till what size the array can increase.When comparing ArrayLists with vectors, it can be seen that ArrayList is faster than vector because it is non-synchronized and vectors are synchronized. Thus, for developers, ArrayList would be recommended. The size of an ArrayList increases by only 50% as compared to the double increase in size for vectors. ArrayLists also have the added advantage that these can be made synchronized externally.

For 2 b) the time complexity is similar to 2 a). The most efficient data structure for summation is accomplished using a simple array. The 2nd most efficient is the ArrayList and the least efficient is the Vector. One possible reason why the vector possessed better construction execution times over the ArrayList is because vectors can make use of both enumeration and iteration while traversing over elements while ArrayList can only make use of iteration for traversing over elements.

Participation Journal

Barbara Friesen (300194589)

- Contributed to coding of PointCPD2
- Contributed to coding of PointCPD4
- Contributed to coding of PointCPD52
- Contributed to coding of PointCPD54
- Coded PointCPD2Test
- Wrote introduction for question 1
- Ran Processes and created tables for all experiments
- Sample output
- Contributed to writing introduction for question 2
- Final edits and formatting

Riya Gill (300176410)

- contributed to the coding of question 2
- writing conclusion for question 2
- Adding visual representation of data of question 2

Ryan Nand (300138674)

- contributed to the coding of question PointCPD4Test
- contributed to the coding of question PointCPD52Test
- contributed to the coding of question PointCPD54Test
- Q1 statements
- Q1 Graphical representation
- Q1 Conclusion
- Final edits and formatting

Amar Shergill (300130878)

- Contributed to coding of question 2
- Contributed to writing introduction for question 2
- Contributed to writing the conclusion for question 2

Vatsal Dedhia (300183606)

- Contributed to coding of PointCPD2
- Contributed to coding of PointCPD4
- Contributed to coding of PointCPD4Test