HW 2 Writeup

Task 1:

Table 1 – Results of <size> 10000000 <experiment> 1 <seed> 42

Repetition	*Regular (seconds)*	*Volatile (seconds)*	*Average Regular Sum*	*Average Volatile Sum*
1	0.00591	0.04512	-5000000.00	-5000000.00
2	0.00598	0.04495	-5000000.00	-5000000.00
3	0.00597	0.04539	-5000000.00	-5000000.00
4	0.00620	0.04608	-5000000.00	-5000000.00
5	0.00670	0.04489	-5000000.00	-5000000.00

$$Regular\ Time\ Average = \frac{(0.00591 + 0.00598 + 0.00597 + 0.00620 + 0.00670)\ secs}{5} = \frac{0.03076\ secs}{5} = 0.006152\ secs$$

$$Volatile\ Time\ Average = \frac{(0.04512 + 0.04495 + 0.04539 + 0.04608 + 0.04489)\ secs}{5} = \frac{0.22643\ secs}{5} = 0.045286\ secs$$

$$Average\ Time\ Ratio = \frac{0.045286\ secs}{0.006152\ secs} \approx 7.3612$$

Table 2 – Results of <size> 20000000 <experiment> 1 <seed> 42

Repetition	*Regular (seconds)*	*Volatile (seconds)*	*Average Regular Sum*	*Average Volatile Sum*
1	0.00933	0.08933	-10000000.00	-10000000.00
2	0.00958	0.08958	-10000000.00	-10000000.00
3	0.00896	0.08967	-10000000.00	-10000000.00
4	0.00994	0.08873	-10000000.00	-10000000.00
5	0.00903	0.08824	-10000000.00	-10000000.00

$$Regular\ Time\ Average = \frac{(0.00933 + 0.00958 + 0.00896 + 0.00994 + 0.00903)\ secs}{5} = \frac{0.04684\ secs}{5} = 0.009368\ secs$$

$$Volatile\ Time\ Average = \frac{(0.08933 + 0.08958 + 0.08967 + 0.08873 + 0.08824)\ secs}{5} = \frac{0.44555\ secs}{5} = 0.08911\ secs$$

$$Average\ Time\ Ratio = \frac{0.08911\ secs}{0.009368\ secs} \approx 9.5122$$

$$Average\ Time\ Ratio\ of\ Table\ 1\ \&\ 2 = \frac{7.3612 + 9.5122}{2} \approx 8.4367$$

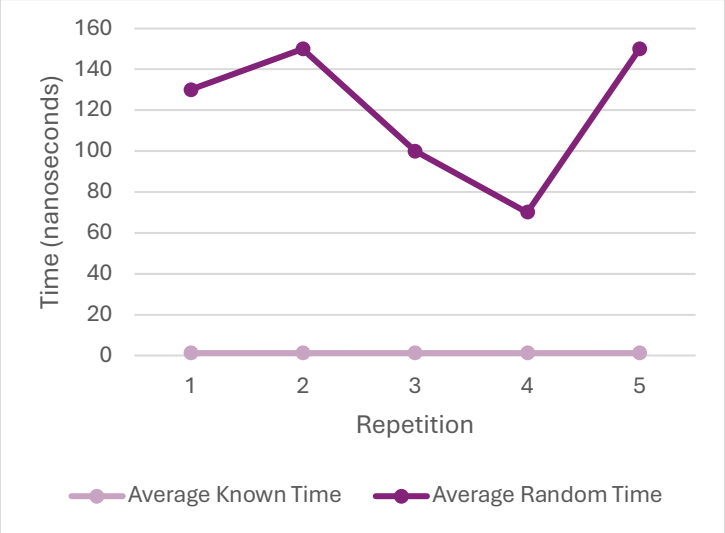
$$Regular\ Time\ Ratio\ of\ Table\ 1\ \&\ 2 = \frac{0.009368\ secs}{0.006152\ secs} \approx 1.5228$$

$$Volatile\ Time\ Ratio\ of\ Table\ 1\ \&\ 2 = \frac{0.08911\ secs}{0.045286\ secs} \approx 1.9676$$

In Task 1, I executed the program ten times, dividing it into two sections of five runs each, with each group using a different input size. Table 1 shows the results for an array size of 10,000,000 elements, and Table 2 shows the results for an array size twice that of Table 1, with 20,000,000 elements. Within each group of five runs, the only parameter that changed among the tables was the size, while all other parameters remained the same, with experiment being 1 and seed being 42. For each run, I recorded the time in seconds it takes for the regular and volatile loops to complete, and the average sums of each to confirm consistency. For Table 1, the regular loop was approximately 7.4 times faster than the volatile loop, while for Table 2, the regular loop was approximately 9.5 times faster than the volatile loop. When comparing Table 1 and 2, the regular loop time doesn't double with its size, but instead increases by approximately 1.5 times, while the volatile loop increases by approximately 2.0 times. Overall, volatile loops increase the time it takes for loops to complete by approximately 8.5 times due to inherently informing the compiler that the variable shouldn't be cached and that accesses should invariably go to main memory.

Task 2:

Graph 1 – Results of <size> 10000000 <experiment> 10 <seed> 42



Repetition	*Average Known Time – First 10% (nanoseconds)*	*Average Random Time – Last 10% (nanoseconds)*	*Sum*
1	1.17	130.00	-43178617617.80
2	1.18	150.00	-43178617617.80
3	1.17	100.00	-43178617617.80
4	1.17	70.00	-43178617617.80
5	1.18	150.00	-43178617617.80

$$Average\ Known\ Time\ Average = \frac{(1.17 + 1.18 + 1.17 + 1.17 + 1.18)\ ns}{5} = \frac{5.87\ ns}{5} \approx 1.174\ ns$$

$$Average\ Random\ Time\ Average = \frac{(130.00 + 150.00 + 100.00 + 70.00 + 150.00)\ ns}{5} = \frac{600\ ns}{5} = 120.00\ ns$$

$$Average\ Time\ Average\ Ratio = \frac{120.00\ ns}{1.174\ ns} \approx 102.21$$

Graph 2 – Results of <size> 10000000 <experiment> 100 <seed> 42

Repetition	*Average Known Time – First 10% (nanoseconds)*	*Average Random Time – Last 10% (nanoseconds)*	*Sum*
1	0.17	25.00	-43944508456.53
2	0.17	24.00	-43944508456.53
3	0.17	26.00	-43944508456.53
4	0.17	24.00	-43944508456.53
5	0.17	26.00	-43944508456.53

$$Average\ Known\ Time\ Average = 0.17\ ns$$

$$Average\ Random\ Time\ Average = \frac{(25.00 + 24.00 + 26.00 + 24.00 + 26.00)\ ns}{5} = \frac{125\ ns}{5} = 25.00\ ns$$

$$Average\ Time\ Average\ Ratio = \frac{25.00\ ns}{0.17\ ns} \approx 147.06$$

$$Average\ Time\ Ratio\ of\ Graph\ 1\ \&\ 2 = \frac{102.21 + 147.06}{2} \approx 124.64$$

$$Average\ Known\ Time\ Ratio\ of\ Graph\ 1\ \&\ 2 = \frac{1.174\ ns}{0.17\ ns} \approx 6.91$$

$$Average\ Random\ Time\ Ratio\ of\ Graph\ 1\ \&\ 2 = \frac{120.00\ ns}{25.00\ ns} \approx 4.80$$

In Task 2, I executed the program ten times, dividing it into two sections of five runs each, with each group using a different input experiment. Graph 1 shows the results for an experiment size of 10, and Graph 2 shows the results for an experiment size of 100. Within each group of five runs, the only parameter that changed among the graphs was the experiment size, while all other parameters remained the same, with size being 10,000,000 and seed being 42. For

each run, I recorded the time in nanoseconds it takes for memory access latency, and the average sum of each to confirm consistency. For Graph 1, the average time for accessing a known element in the first 10% was approximately 102 times faster than the average time for accessing a random element. In contrast, for Graph 2, the average time for accessing a known element in the first 10% was approximately 147 times faster than the average time for accessing a random element. When comparing Graphs 1 and 2, the average time for accessing an element decreases as the experiment size increases. Graph 2's average time for accessing a known element was approximately 6.9 times faster than was approximately 4.8 times average time for accessing a random element than Graph 1. Overall, accessing an element is approximately 125 times quicker when you know the element you are looking for, as it's likely already cached. If not, the compiler knows what it's looking for and can access it more quickly.

Task 3:

Table 1 – Results of <size> 25000000 <experiment> 1 <seed> 30

Repetition	*Average TreeSet Time (nanoseconds)*	*Average LinkedList Time (nanoseconds)*
1	39100.00	79449900.00
2	49700.00	74290900.00
3	42900.00	78639000.00
4	36100.00	101664700.00
5	41800.00	87910100.00

$$Average\ TreeSet\ Time\ Average = \frac{(39100.00 + 49700.00 + 42900.00 + 36100.00 + 41800.00)\ ns}{5} = \frac{209600\ ns}{5}$$
$$= 141920\ ns$$

$$Average\ LinkedList\ Time\ Average$$
$$= \frac{(79449900.00 + 74290900.00 + 78639000.00 + 101664700.00 + 87910100.00)\ ns}{5}$$
$$= \frac{421954600\ ns}{5} = 84390920\ ns$$

$$Average\ Time\ Average\ Ratio = \frac{84390920\ ns}{141920\ ns} \approx 594.64$$

Table 2 – Results of <size> 25000000 <experiment> 1 <seed> 300

Repetition	*Average TreeSet Time (nanoseconds)*	*Average LinkedList Time (nanoseconds)*
1	119300.00	47561900.00
2	34000.00	40752300.00
3	41400.00	40034500.00
4	41000.00	38452300.00

5	88100.00	39117300.00

$$Average\ TreeSet\ Time\ Average = \frac{(119300.00 + 34000.00 + 41400.00 + 41000.00 + 88100.00)\ ns}{5} = \frac{323800\ ns}{5}$$
$$= 64760\ ns$$

$$Average\ LinkedList\ Time\ Average$$
$$= \frac{(47561900.00 + 40752300.00 + 40034500.00 + 138452300.00 + 39117300.00)\ ns}{5}$$
$$= \frac{305818300\ ns}{5} = 61163660\ ns$$

$$Average\ Time\ Average\ Ratio = \frac{61163660\ ns}{64760\ ns} \approx 944.47$$

$$Average\ Time\ Ratio\ of\ Table\ 1\ \&\ 2 = \frac{594.64 + 944.47}{2} \approx 769.56$$

$$Average\ TreeSet\ Time\ Ratio\ of\ Table\ 1\ \&\ 2 = \frac{141920\ ns}{64760\ ns} \approx 2.19$$

$$Average\ LinkedList\ Time\ Ratio\ of\ Table\ 1\ \&\ 2 = \frac{84390920\ ns}{61163660\ ns} \approx 1.38$$

In Task 3, I executed the program ten times, dividing it into two sections of five runs each, with each group using a different input seed. Table 1 shows the results for a seed size of 30, and Table 2 shows the results for a seed size of 300. Within each group of five runs, the only parameter that changed among the tables was the seed size, while all other parameters remained the same, with size being 25,000,000 and experiment being 1. For each run, I recorded the time in nanoseconds it takes for memory access latency in a TreeSet and LinkedList. For Table 1, the average time for accessing an element in a LinkedList is approximately 594.6 times slower than the average time for accessing an element in a TreeSet. In contrast, for Table 2, the average time for accessing an element in a LinkedList is approximately 944.5 times slower than the average time for accessing an element in a TreeSet. When comparing Tables 1 and 2, the average time for accessing an element in a TreeSet increases by approximately 2.2 times when the seed is increase 900%; however, the average time for accessing an element in LinkedList increases approximately 1.4 times when the seed is increases 900%. Overall, accessing an element in a TreeSet is approximately 769.6 times faster than accessing an element in a LinkedList due to the balanced of the binary trees that in turn are quicker to search.