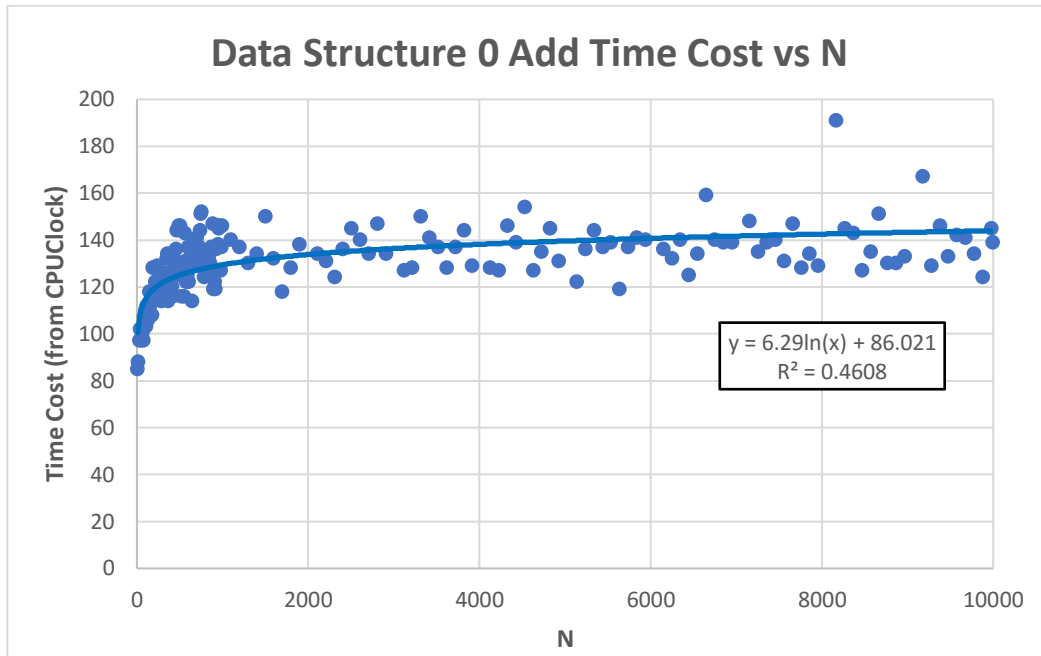
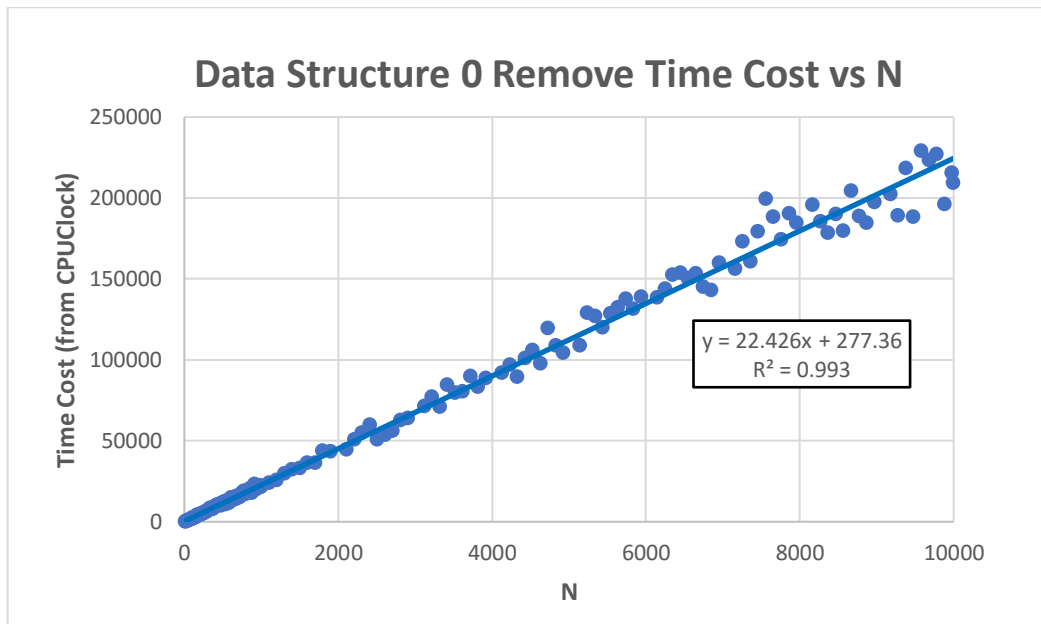


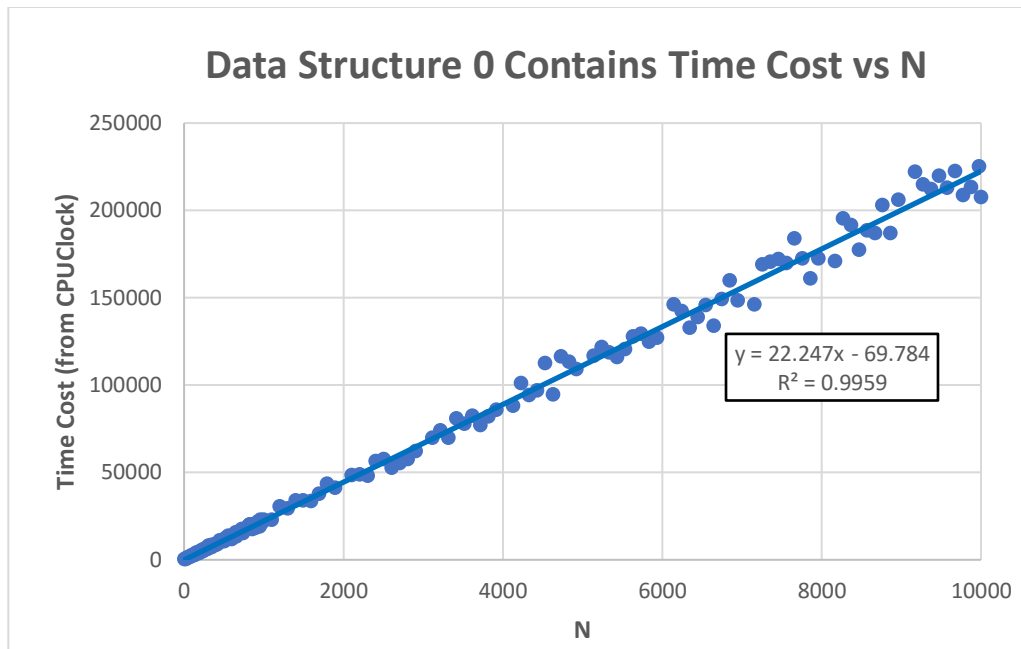
## Mystery Data Structure 0



**Graph 1:** The graph of the time cost versus n of using add() with mystery data structure 0.



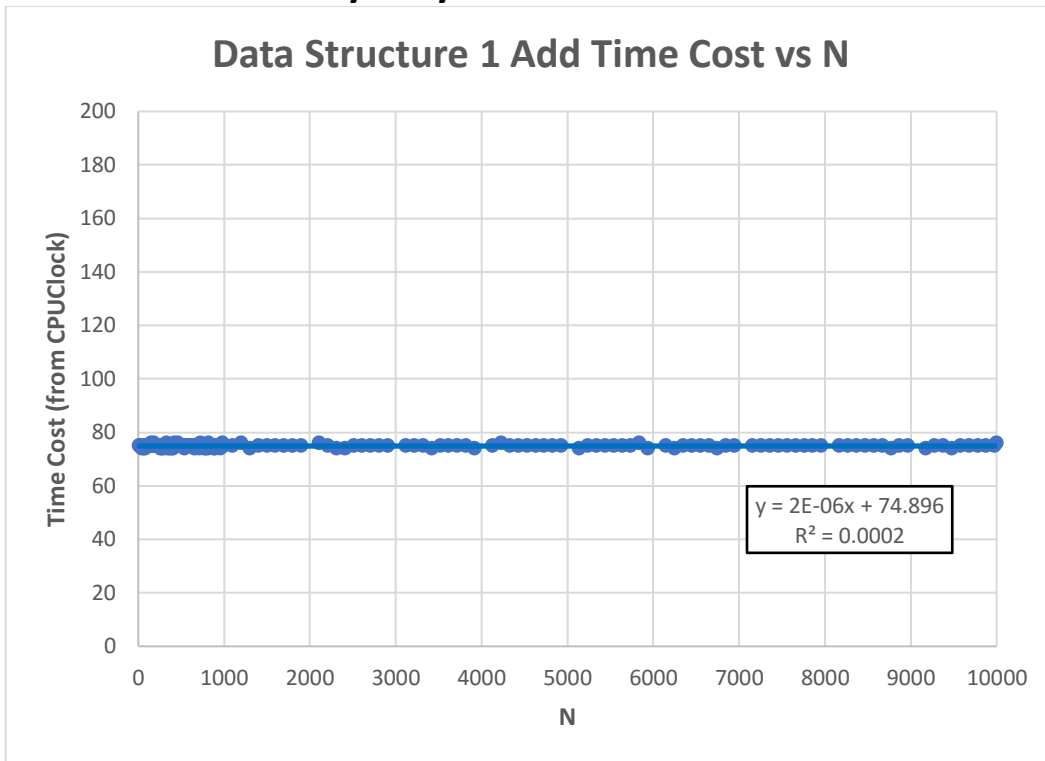
**Graph 2:** The graph of the time cost versus n of using remove() with mystery data structure 0.



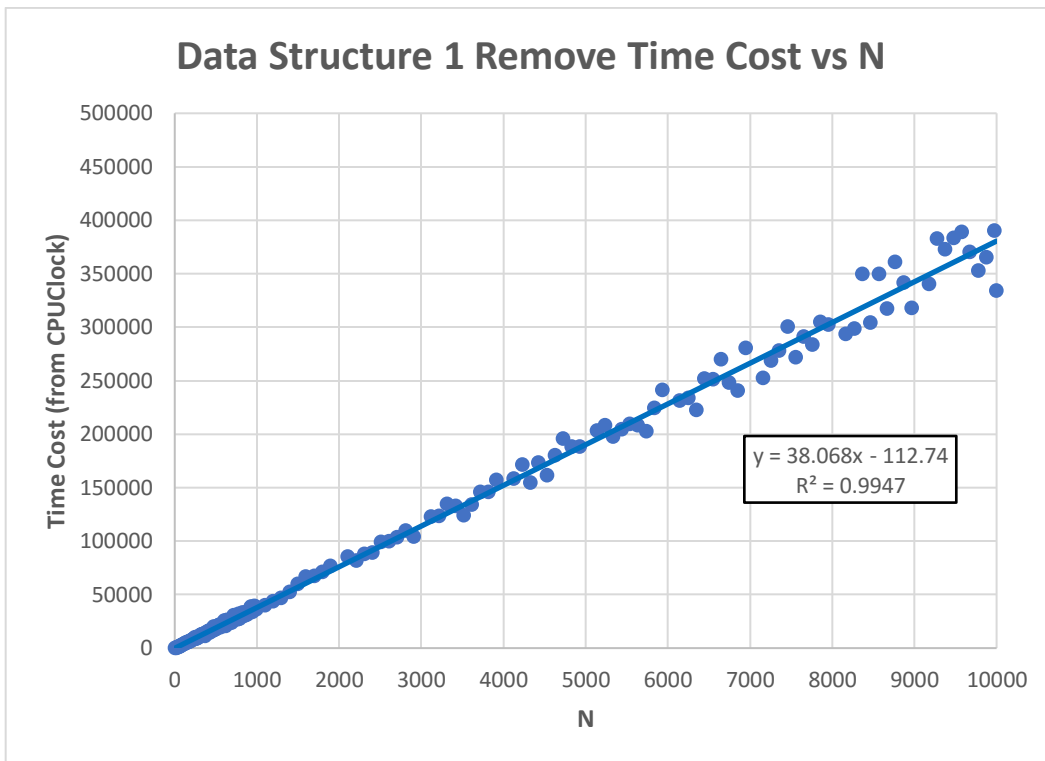
**Graph 3:** The graph of the time cost versus n of using contains() with mystery data structure 0.

We have deemed data structure 0 to be a heap. **Graph 1** shows a (rough) logarithmic shape, as the rate of change slows as N increases, which matches the  $O(\log n)$  time cost of the add method for a heap. Furthermore, **Graph 2** and **Graph 3** clearly show a linear shape, which again conforms to the time costs of a heap.

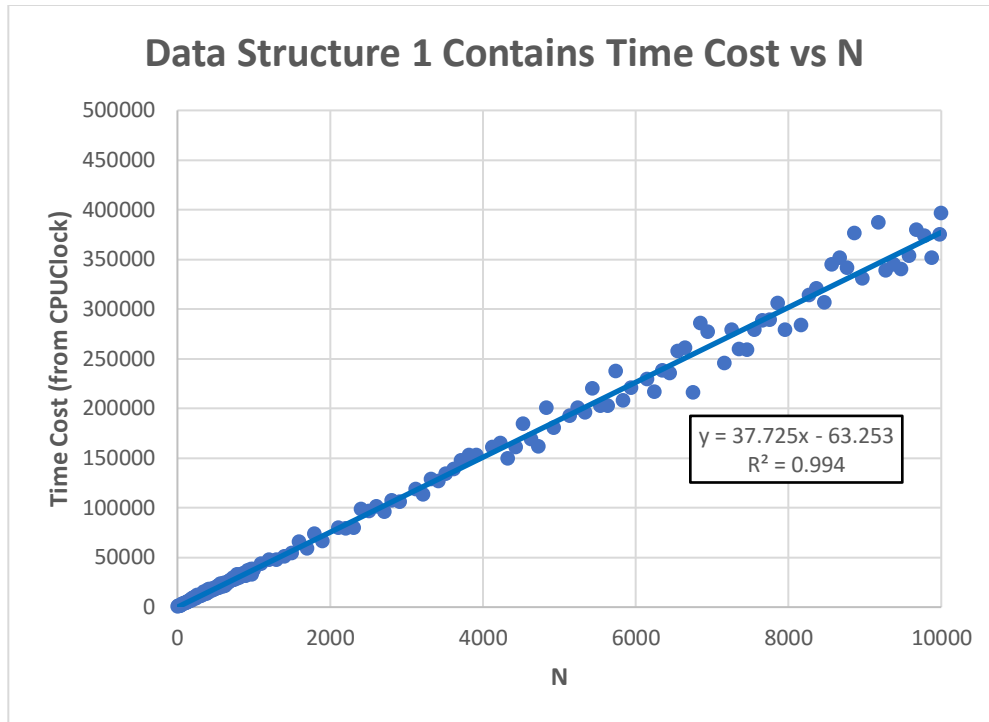
## Mystery Data Structure 1



**Graph 4:** The graph of the time cost versus n of using add() with mystery data structure 1.



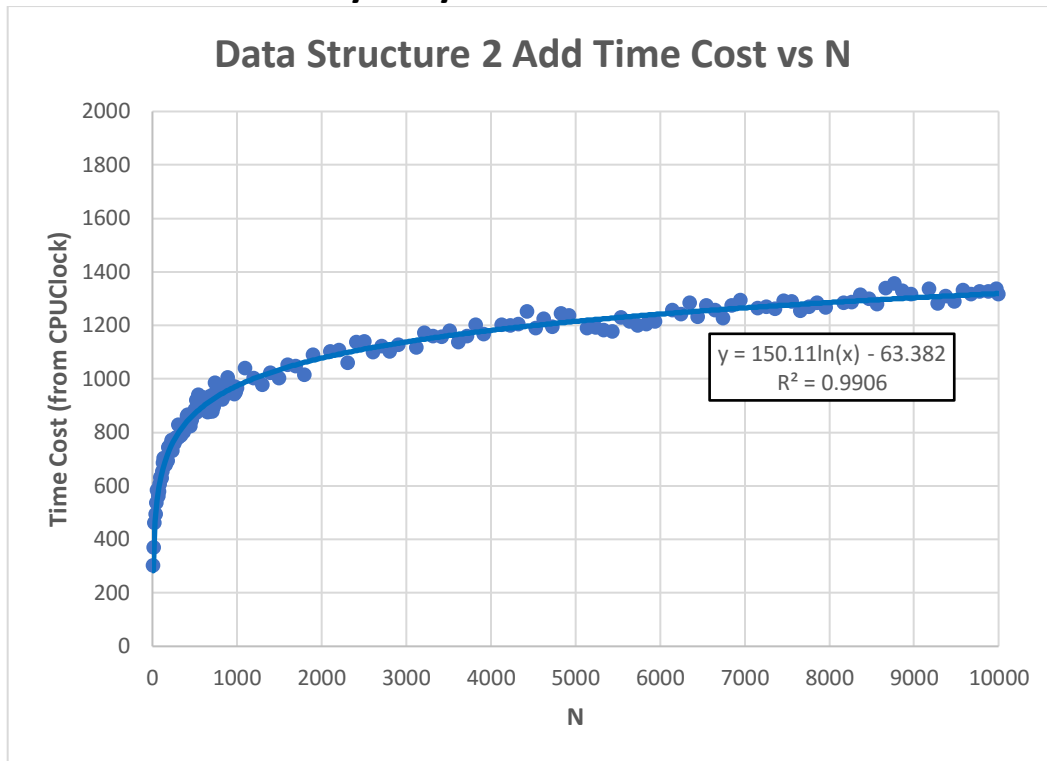
**Graph 5:** The graph of the time cost versus n of using remove() with mystery data structure 1.



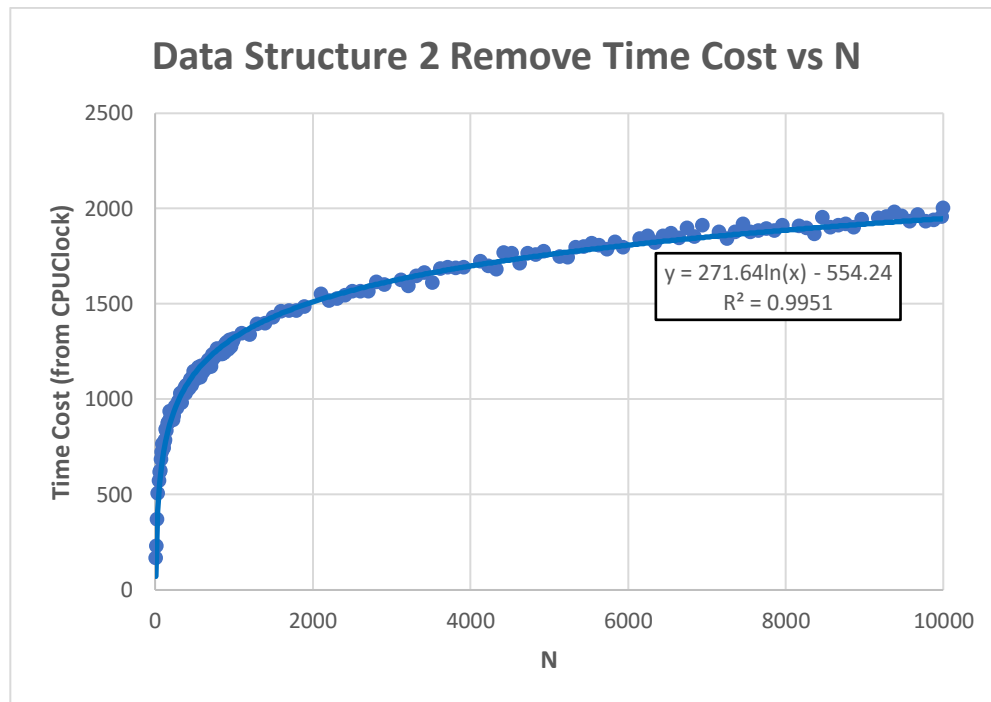
**Graph 6:** The graph of the time cost versus n of using contains() with mystery data structure 1.

We have deemed data structure 1 to be a doubly-linked list. **Graph 4** clearly depicts a  $O(1)$  time cost, which matches the time cost of the add method for a doubly linked list. Furthermore, **Graph 5** and **Graph 6** clearly show a linear shape, which again conforms to the time costs of a doubly linked list.

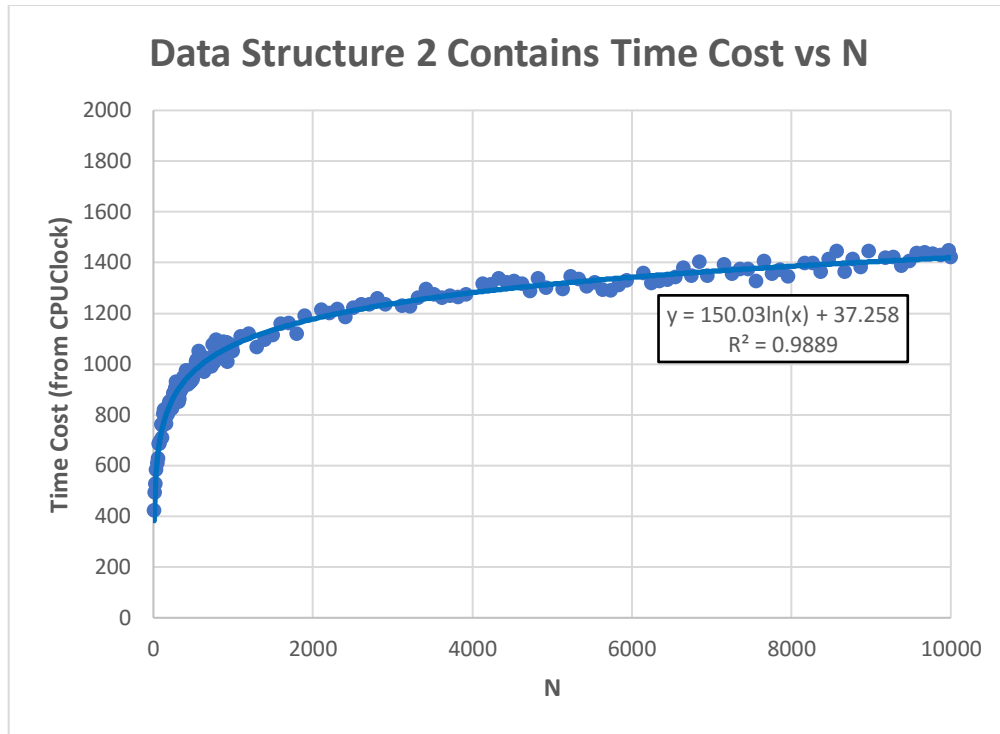
## Mystery Data Structure 2



**Graph 7:** The graph of the time cost versus n of using add() with mystery data structure 2.



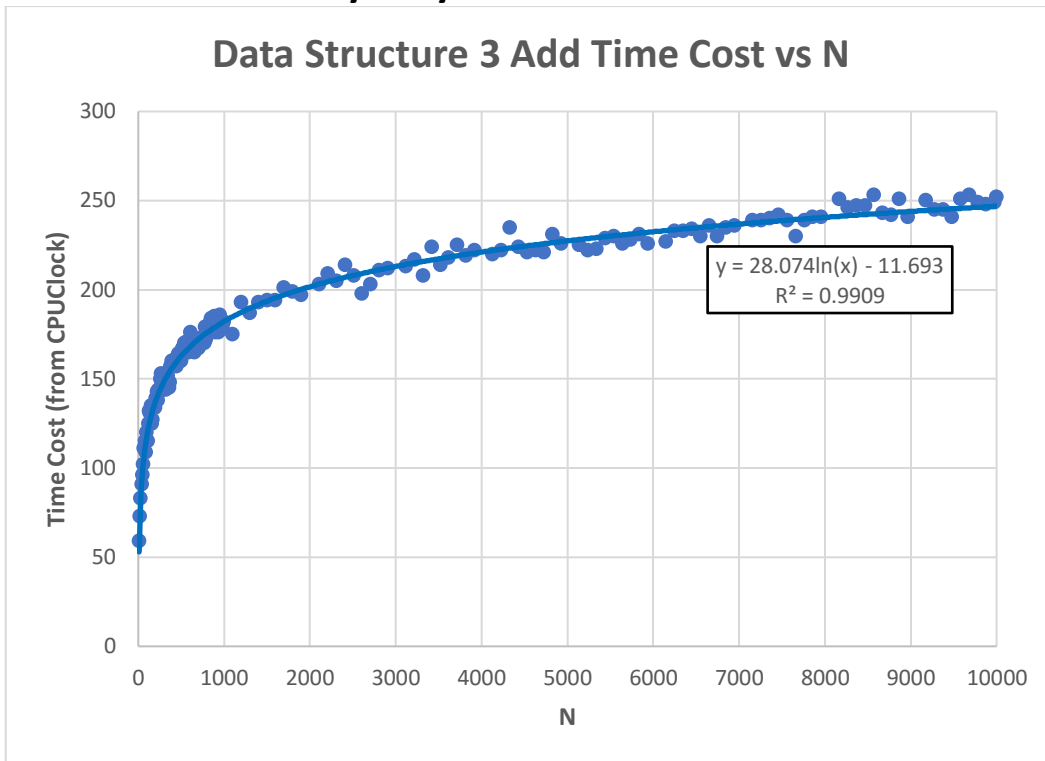
**Graph 8:** The graph of the time cost versus n of using remove() with mystery data structure 2.



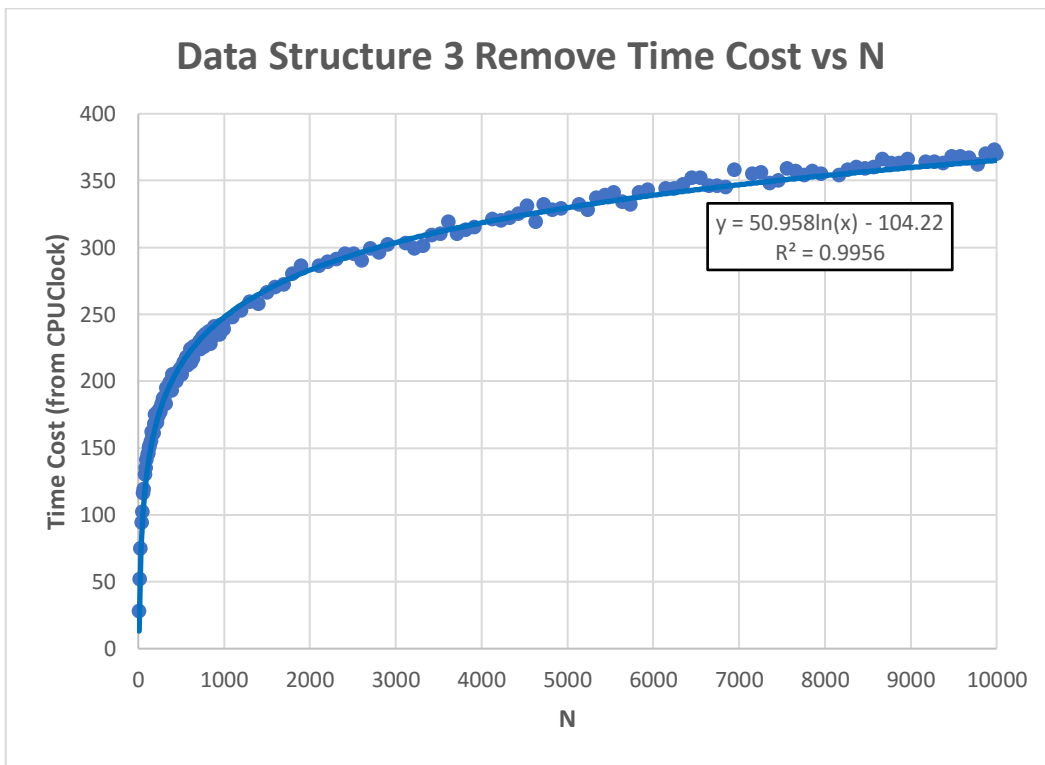
**Graph 9:** The graph of the time cost versus n of using contains() with mystery data structure 2.

We have deemed data structure 2 to be a binary search tree. **Graph 7**, **Graph 8**, and **Graph 9** show a logarithmic relationship between N and the asymptotic time cost, which matches the known time cost of a binary search tree for all 3 main methods:  $O(\log n)$ . In all three graphs, this self-balancing binary search tree shows a very accurate logarithmic curve with little variation.

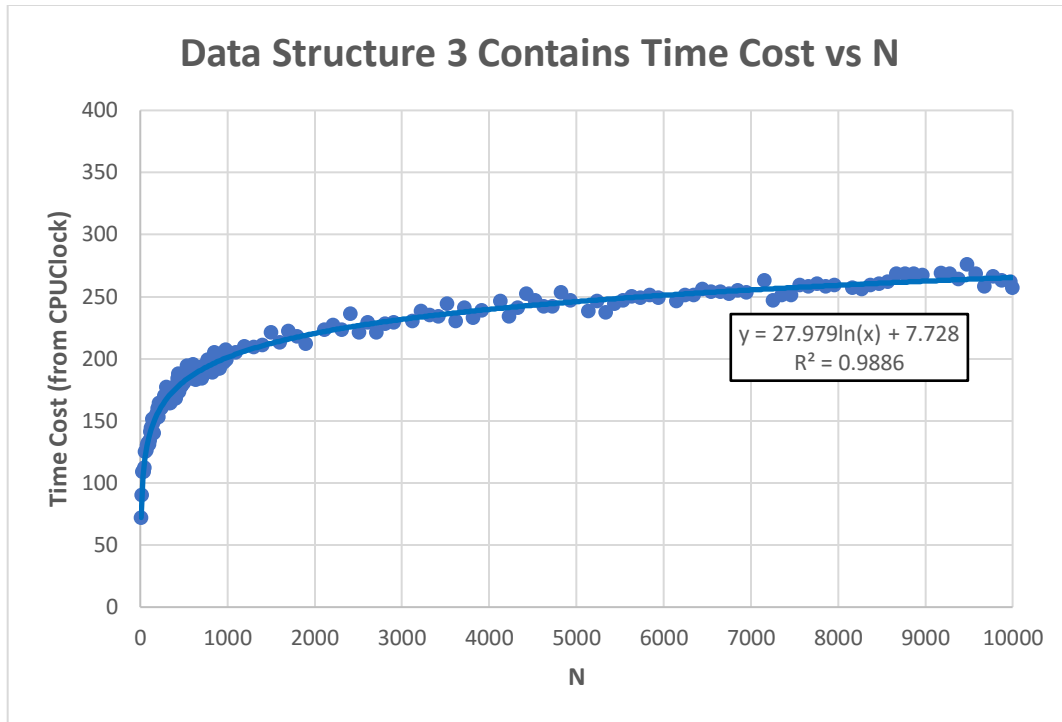
## Mystery Data Structure 3



**Graph 10:** The graph of the time cost versus n of using add() with mystery data structure 3.



**Graph 11:** The graph of the time cost versus n of using remove() with mystery data structure 3.

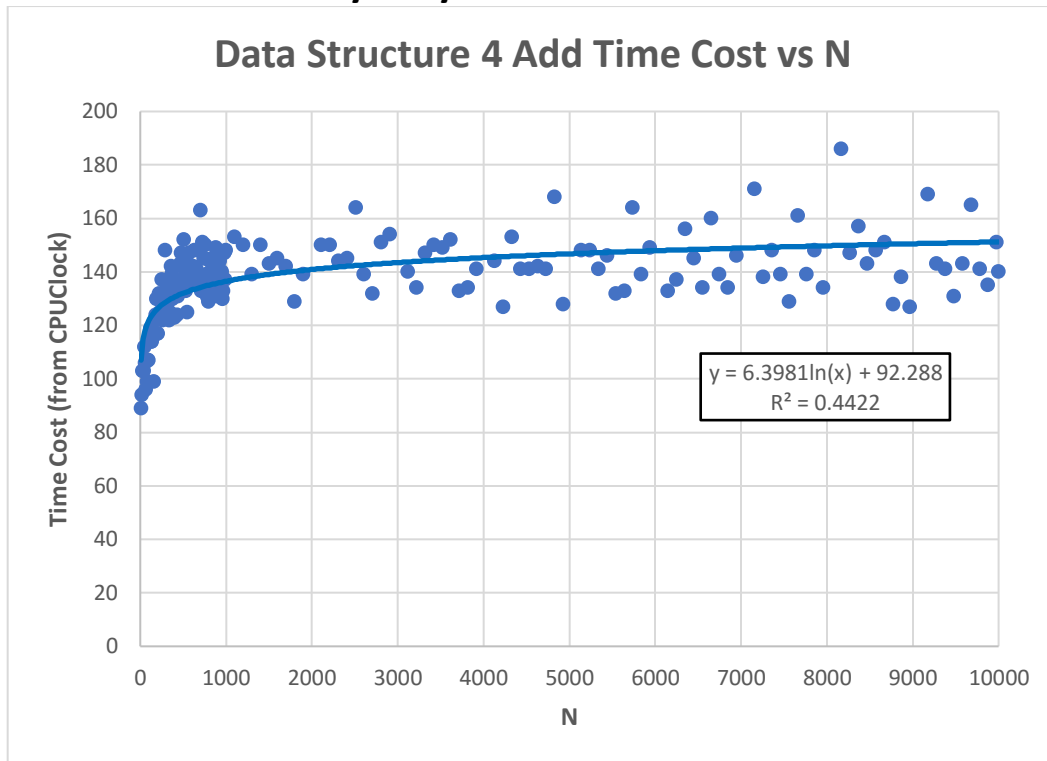


**Graph 12:** The graph of the time cost versus n of using contains() with mystery data structure 3.

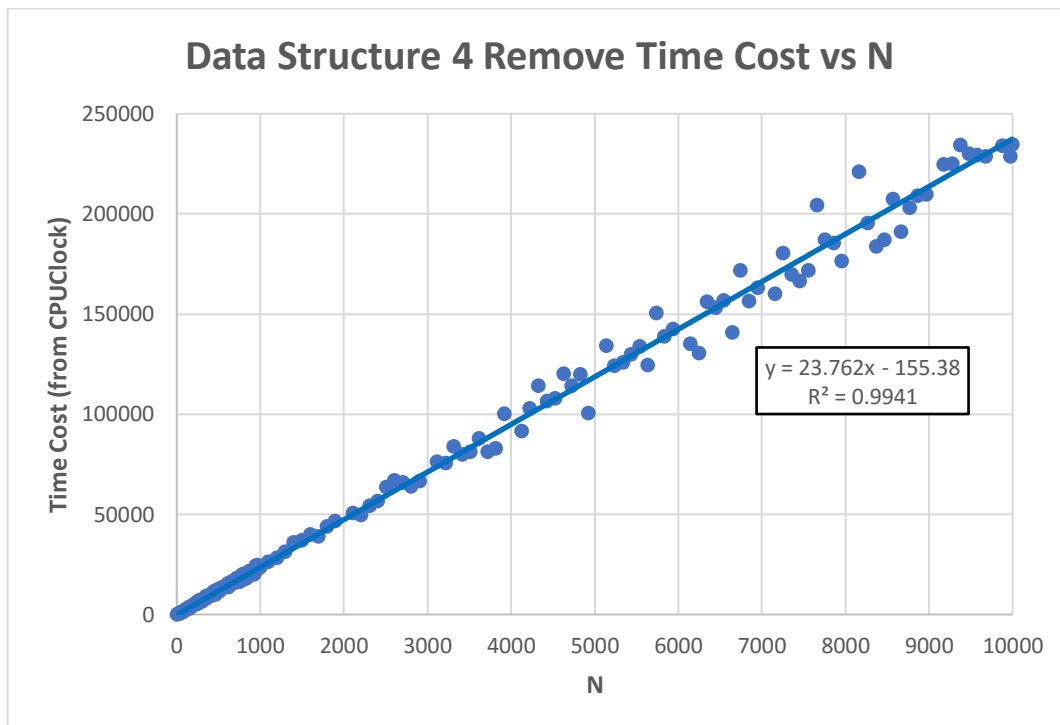
We have deemed data structure 3 to be a binary search tree. **Graph 10**, **Graph 11**, and **Graph 12** show a logarithmic relationship between N and the asymptotic time cost, which matches the known time cost of a binary search tree for all 3 main methods:  $O(\log n)$ . In all three graphs, this self-balancing binary search tree shows a very accurate logarithmic curve with little variation as all three  $R^2$  are greater than 0.98.



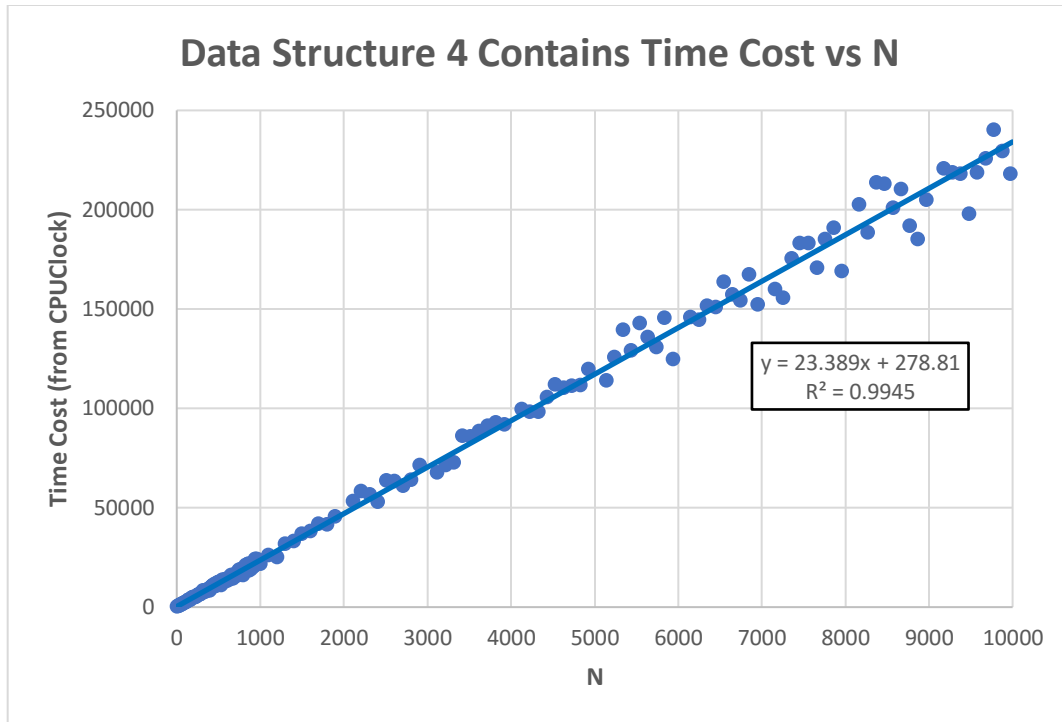
## Mystery Data Structure 4



**Graph 13:** The graph of the time cost versus n of using add() with mystery data structure 4.



**Graph 14:** The graph of the time cost versus n of using remove() with mystery data structure 4.



**Graph 15:** The graph of the time cost versus  $n$  of using `contains()` with mystery data structure 4.

We have deemed data structure 4 to be a heap, the same as data structure 0. **Graph 13** shows a (rough) – again very similar to data structure 0 – logarithmic shape, as the rate of change slows as  $N$  increases, which matches the  $O(\log n)$  time cost of the `add` method for a heap. Furthermore, **Graph 14** and **Graph 15** clearly show a linear shape, which again conforms to the time costs of a heap. Because the  $R^2$  value of **Graph 13** is quite low (high variance) for both data structure 0 and this data structure, we have determined data structure 4 to be another occurrence of a heap.