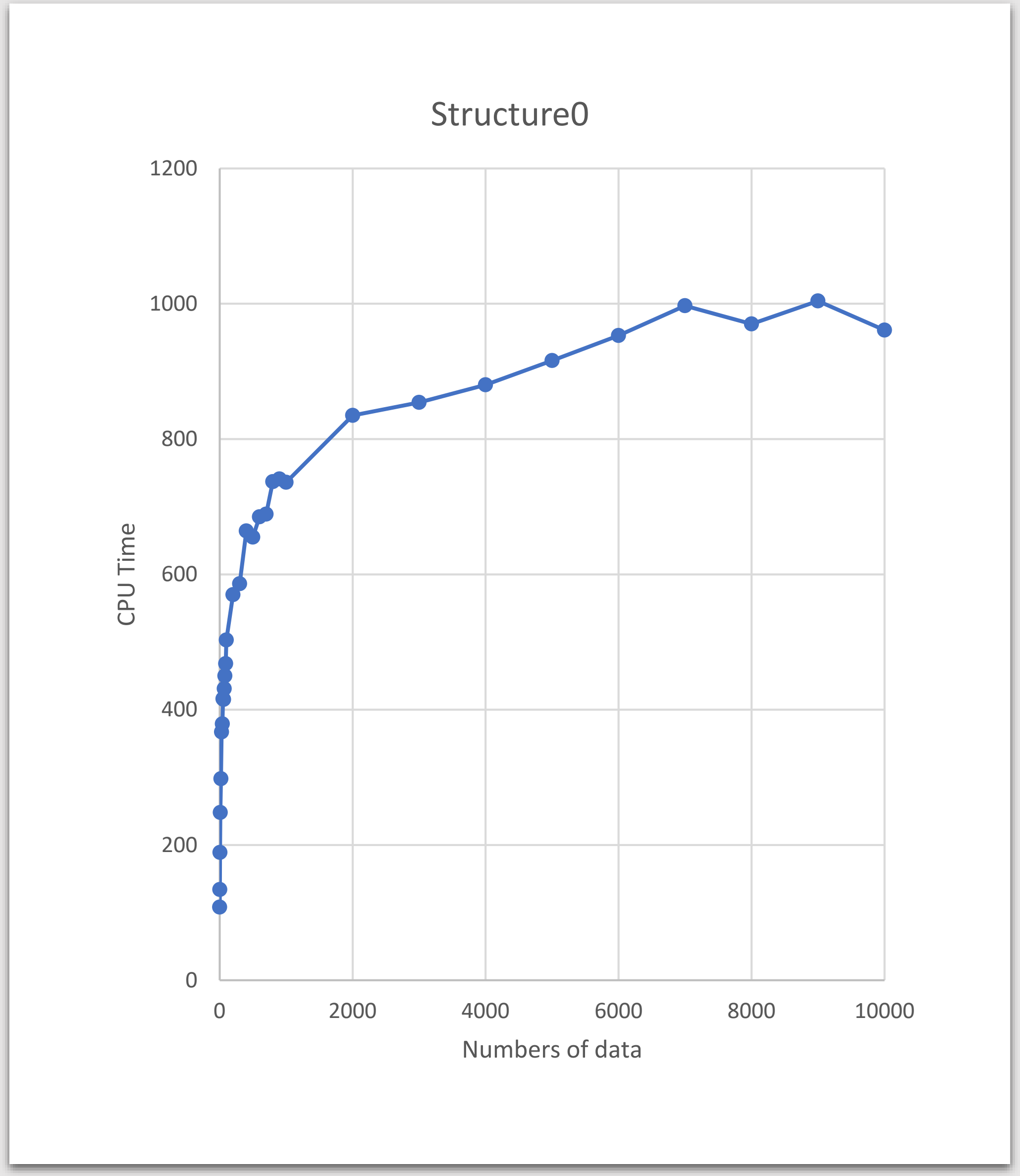
Project 4

Kepei Lei & PJ Mara

Procedure

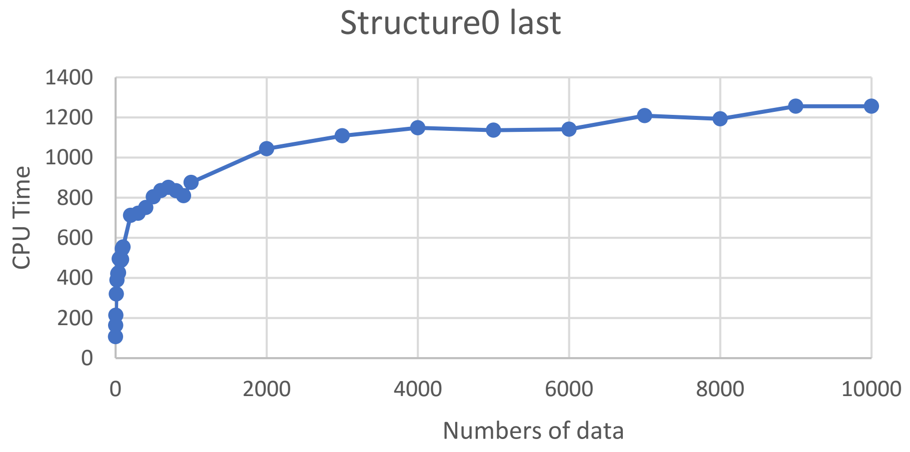
- Each structure of size n (taken from array of n’s defined as a constant at the top of the file) is populated with numbers from 0-> n-1
- Add method- Adds a new value (n) into the structure
- Remove Method- Best case removes the value 0. Worst Case is that the value is not in the array- uses value n. Average case picks a number randomly between 0 and n-1.
- Contains Method- Best, Worst, and average are the same as remove- however, we also include a test for a heap by looking for the n-1 value, which will be much faster because it is the largest in the structure.
- Key- “Worst” = D.N.E = does not exist (in structure). “Best” = smallest value in structure “Biggest” = n-1 value in structure.

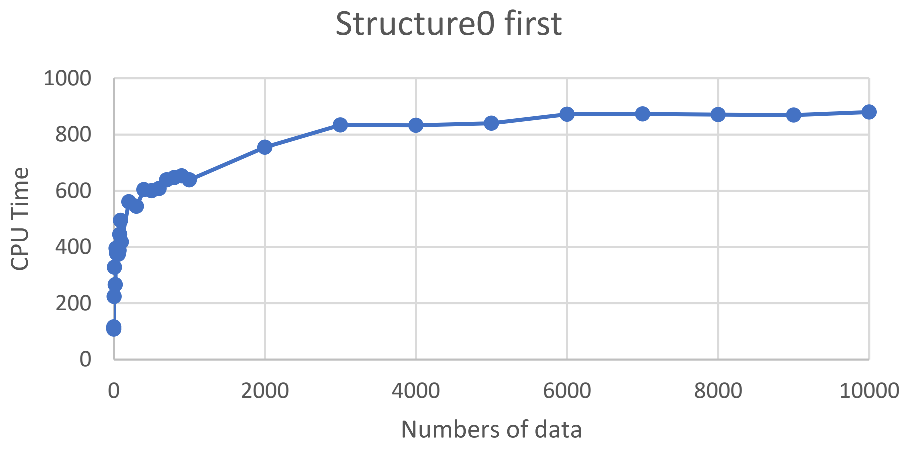


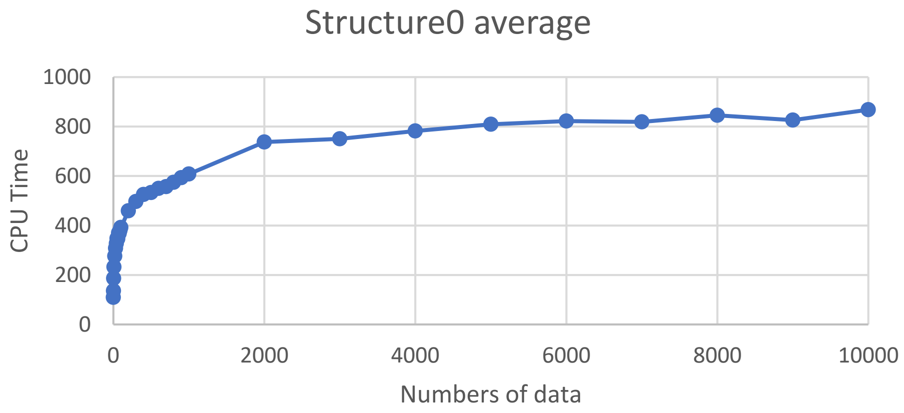
Structure 0: add

O(log(n))

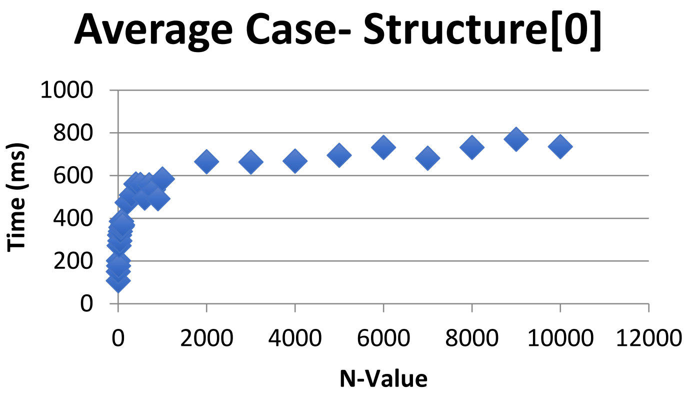
Structure 0: remove

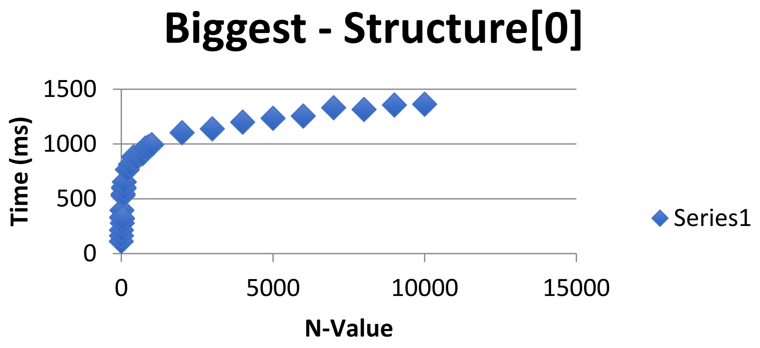






- Last: O(log(n))
- First: O(log(n))
- Average: O(log(n))





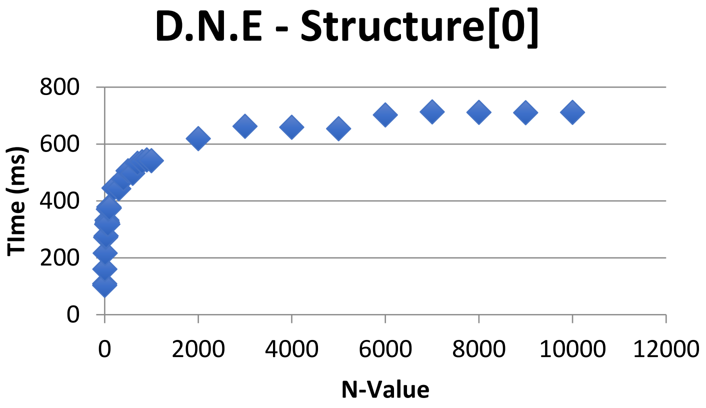
Structure 0: contains

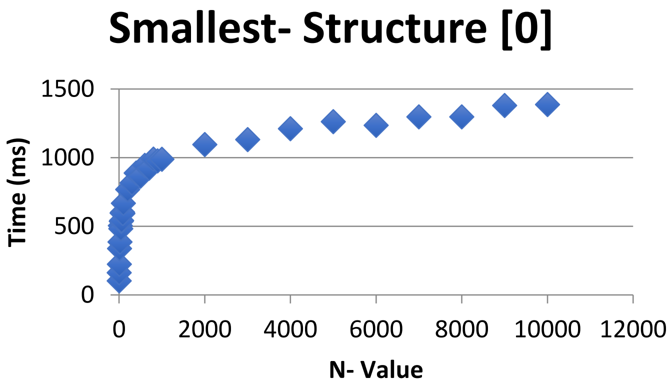
Average: O(log(n))

Last(biggest): O(log(n))

First(smallest): O(log(n))

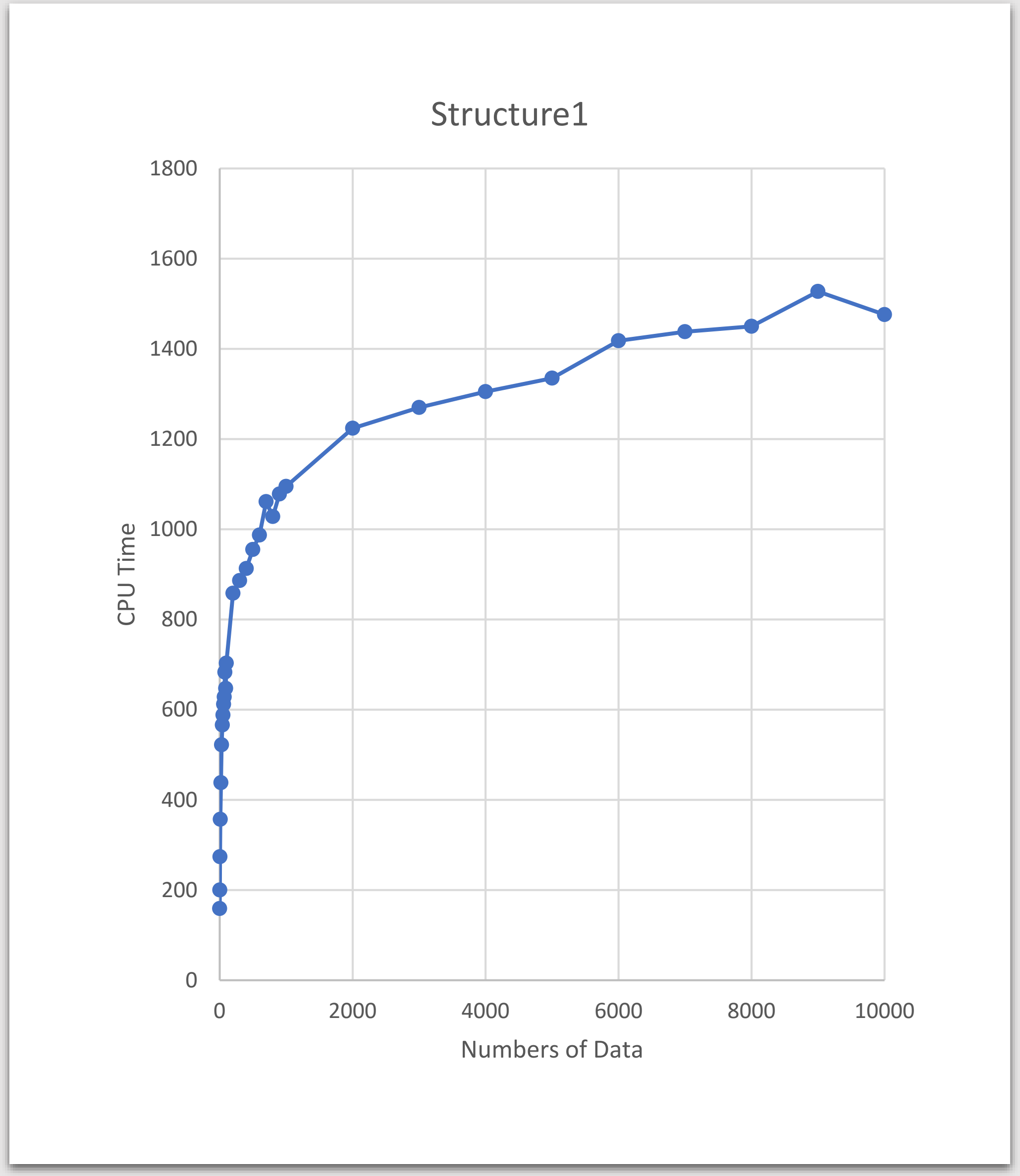
N+1(D.N.E): O(log(n))





Conclusion

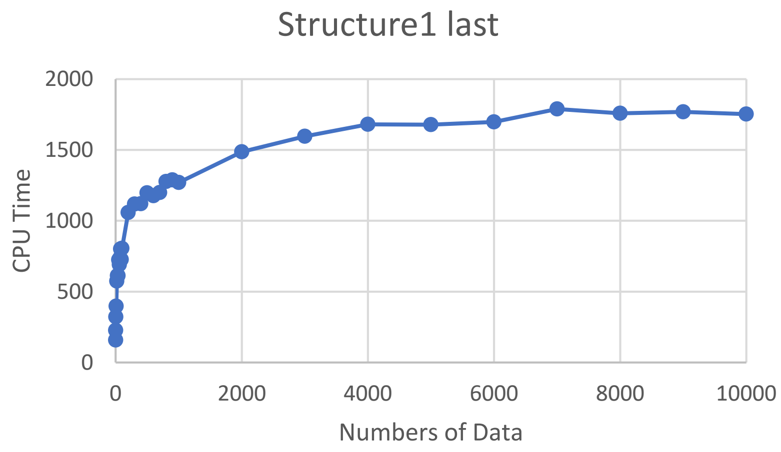
- Structure 0 is a self-balancing Binary Search Tree
- Everything is O(log(n)), regardless of max, min, or average, in the remove or contains case. This is enough to prove self-balancing BST.

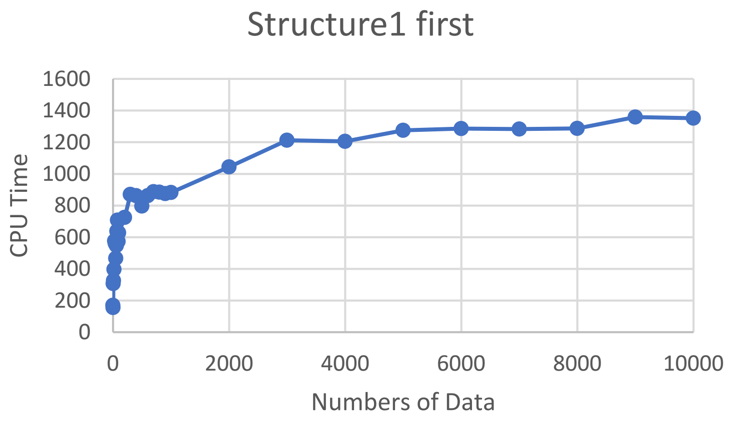


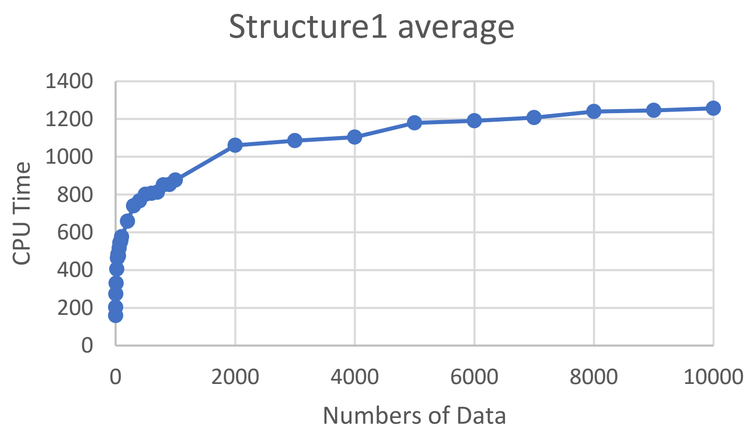
Structure 1: add

O(log(n))

Structure 1: remove

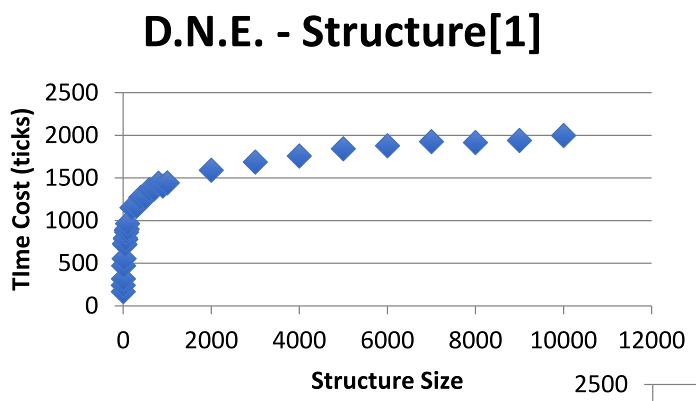


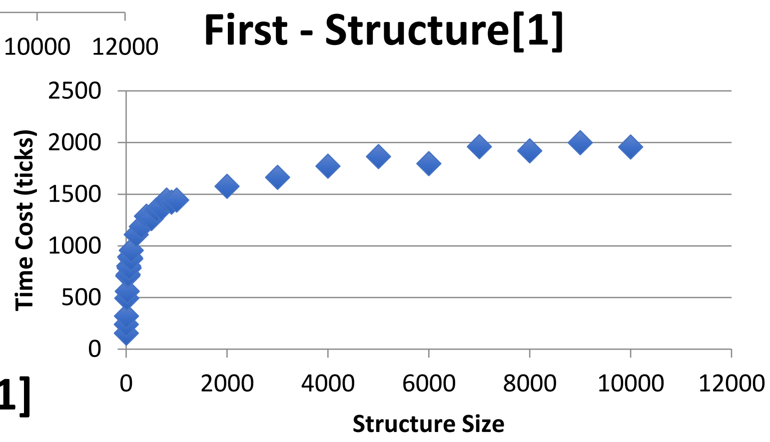


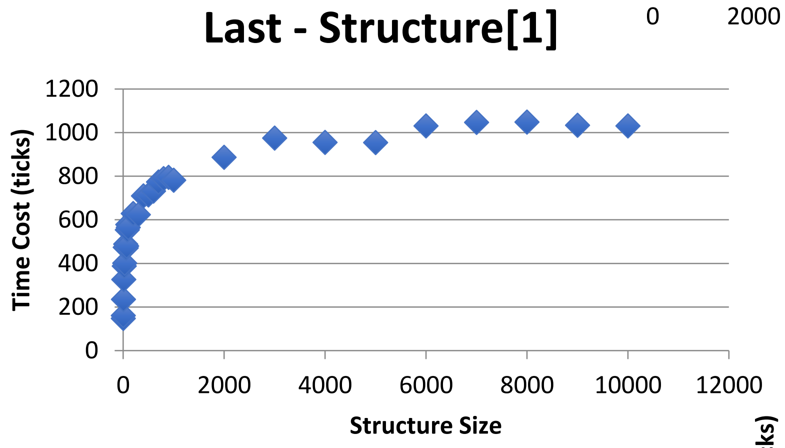


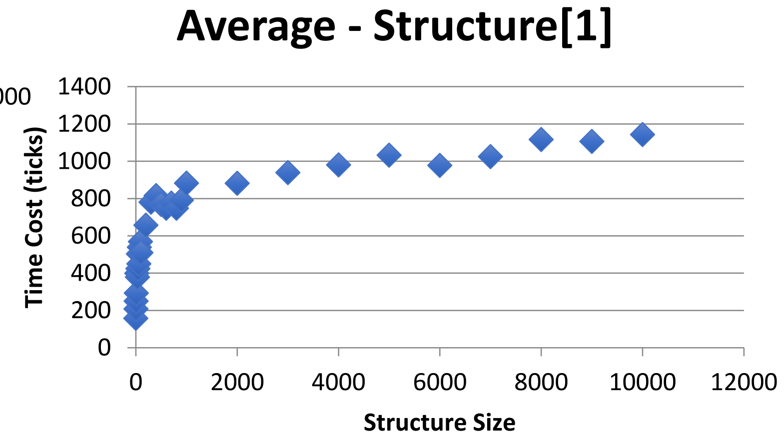
- Last: O(log(n))
- First: O(log(n))
- Average: O(log(n))

Structure 1: contains









Average: O(log(n))

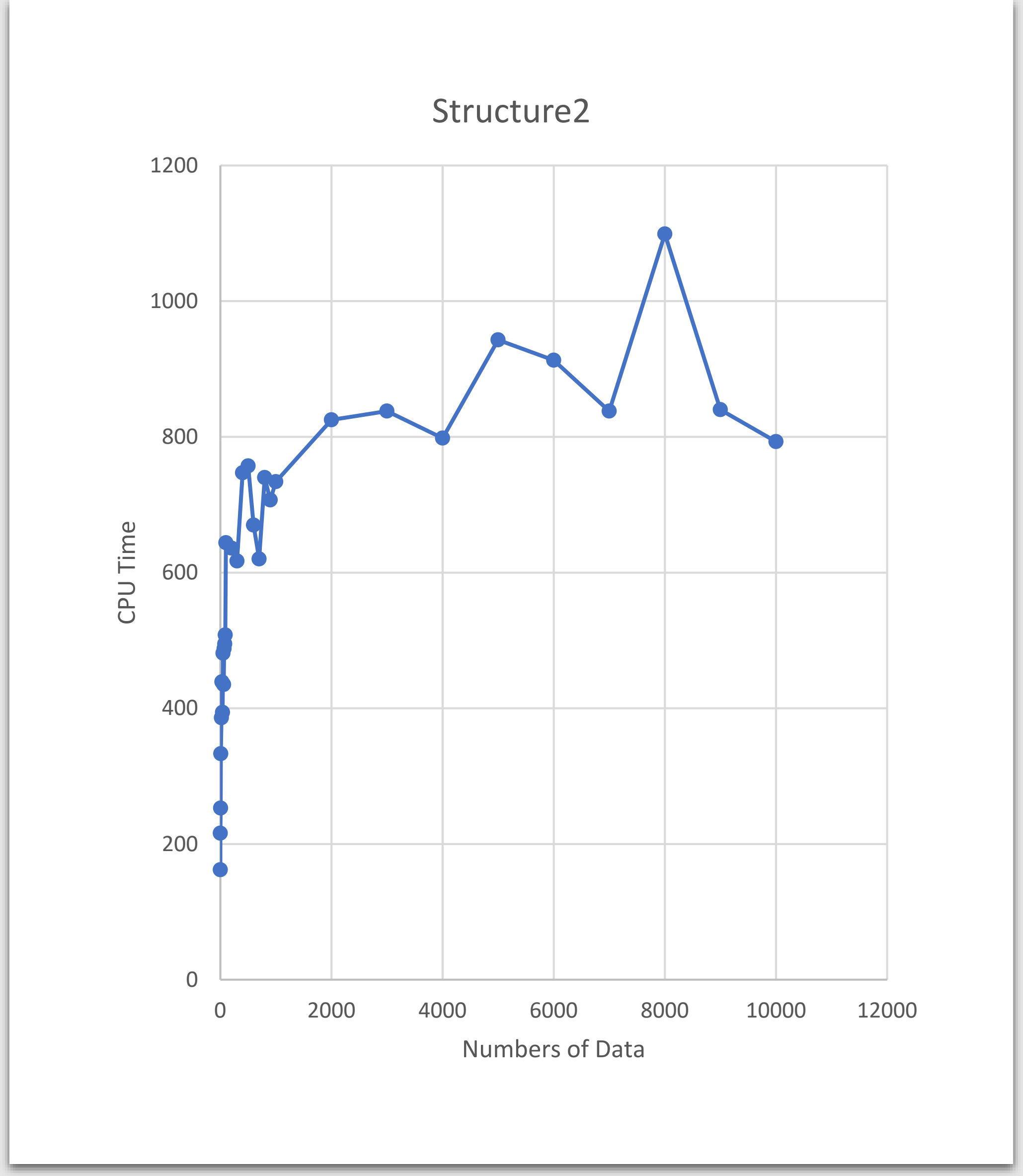
Last(biggest#): O(log(n))

First(smallest#): O(log(n))

N+1(D.N.E): O(log(n))

Conclusion

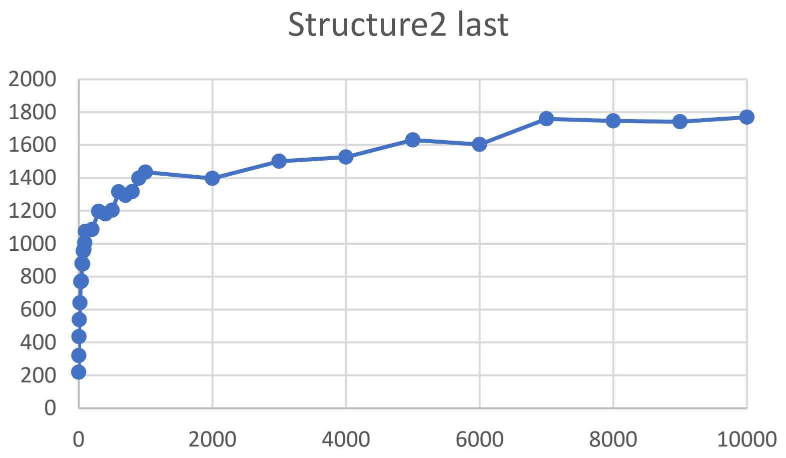
- Structure 1 is a self-balancing Binary Search Tree
- Everything is O(log(n)), regardless of max, min, or average, in the remove or contains case. This is enough to prove self-balancing BST.

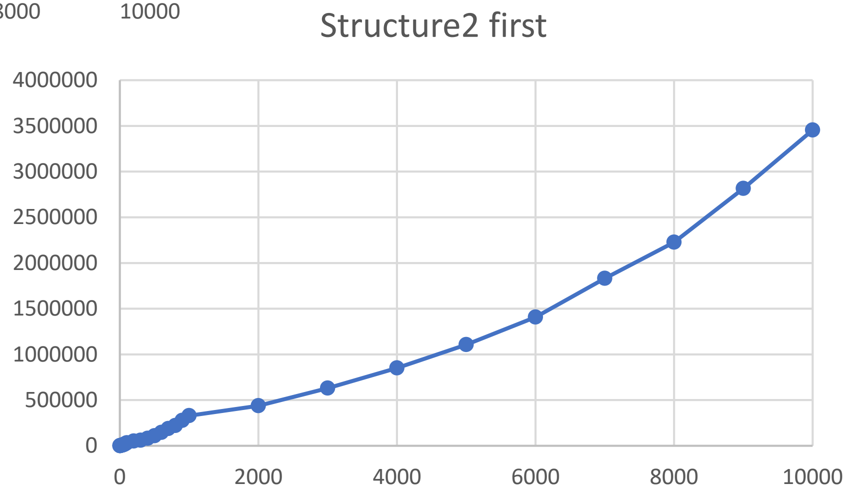


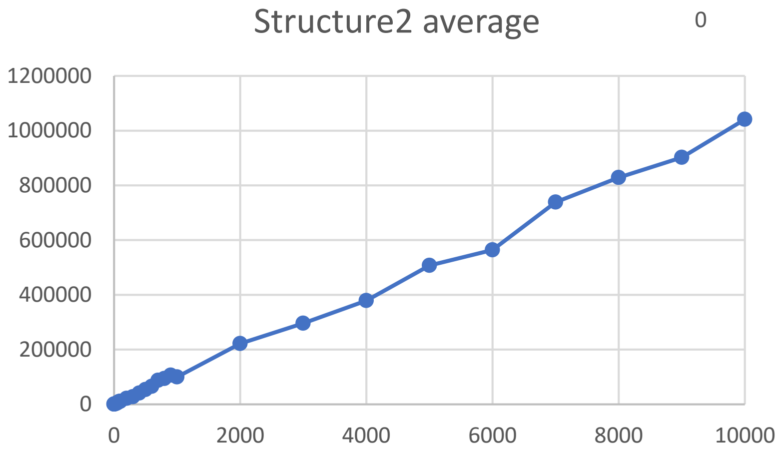
Structure 2: add

O(log(n))

Structure 2: remove

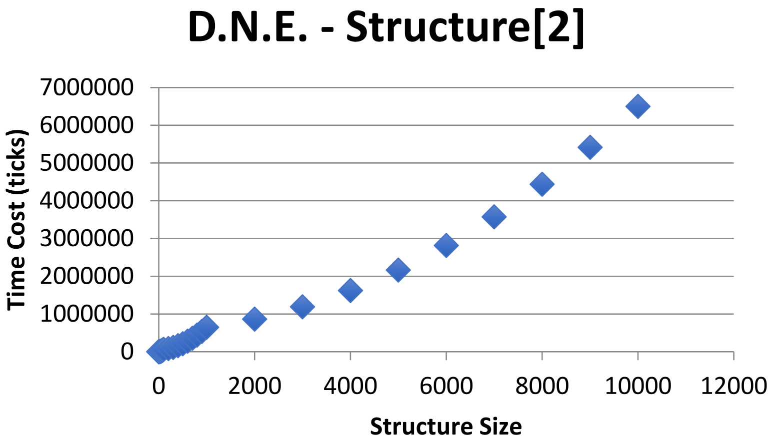


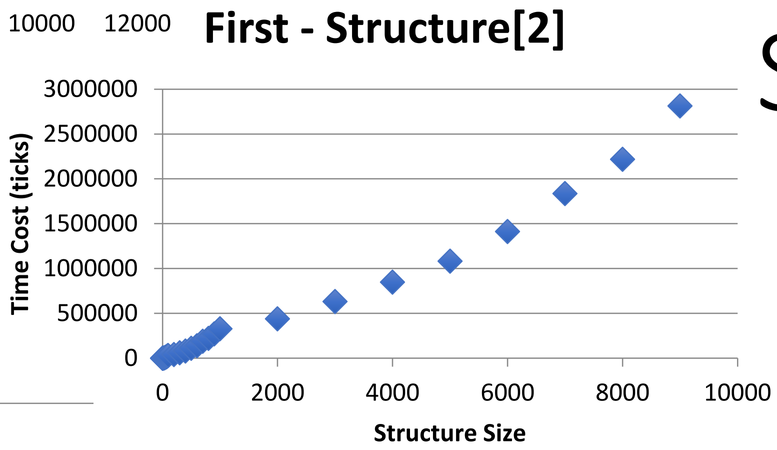


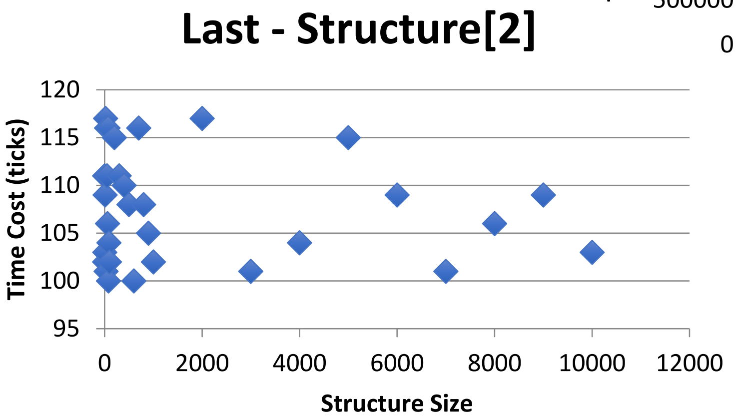


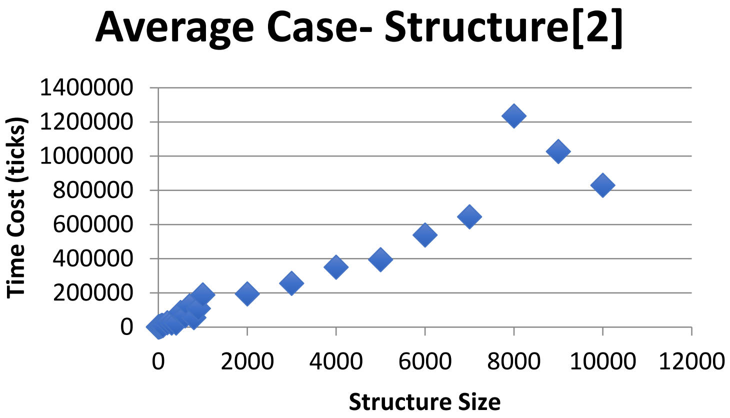
- Last: O(log(n))
  - First: O(n)
- Average: O(n)

Structure 2: contains









Average: O(n)

(with a jump)

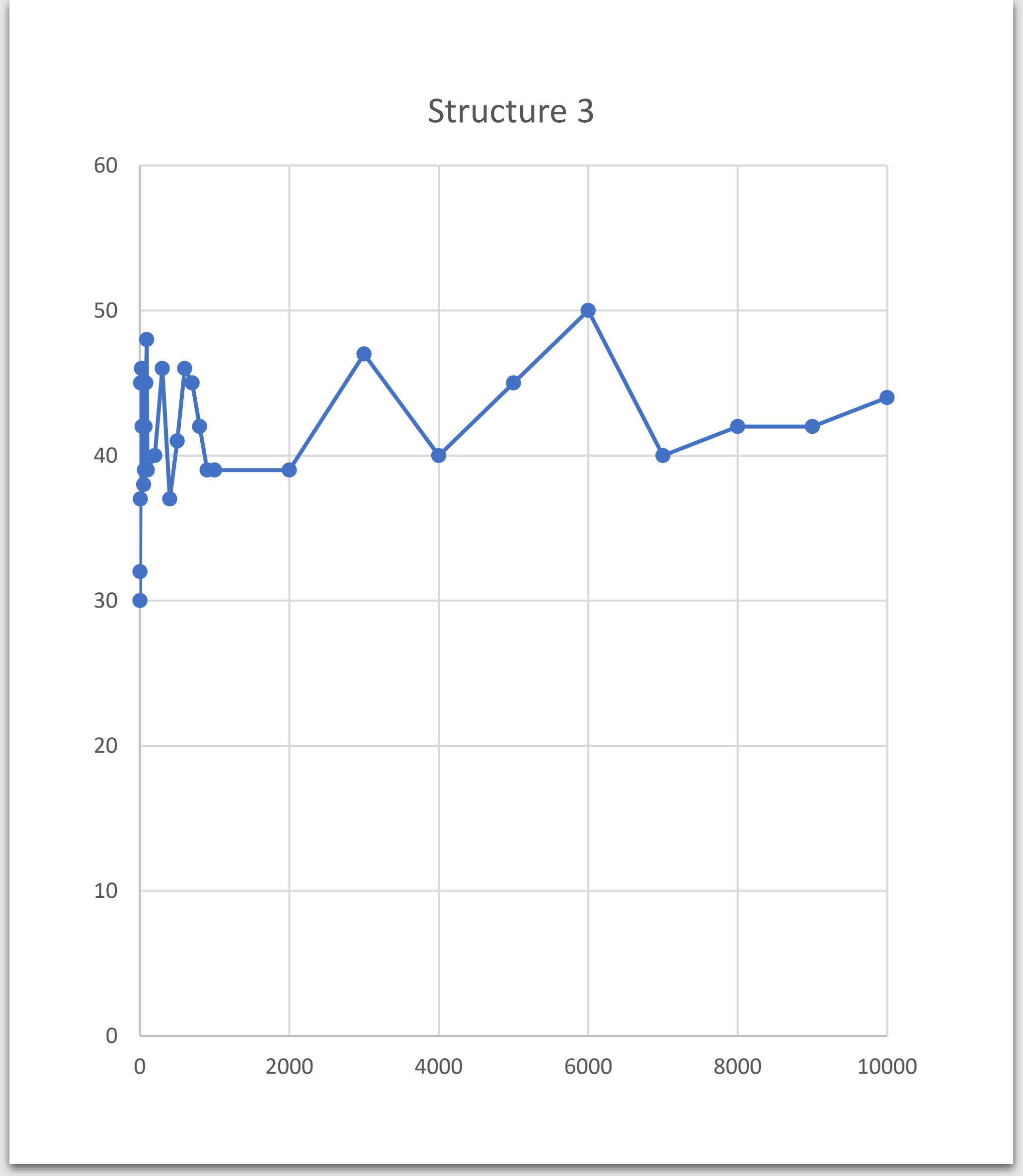
Last(biggest): O(1)

First(smallest): O(n)

N+1(D.N.E): O(n)

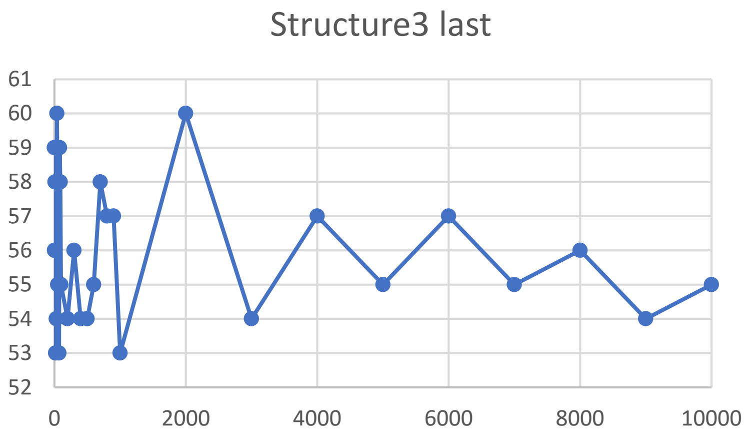
Conclusion

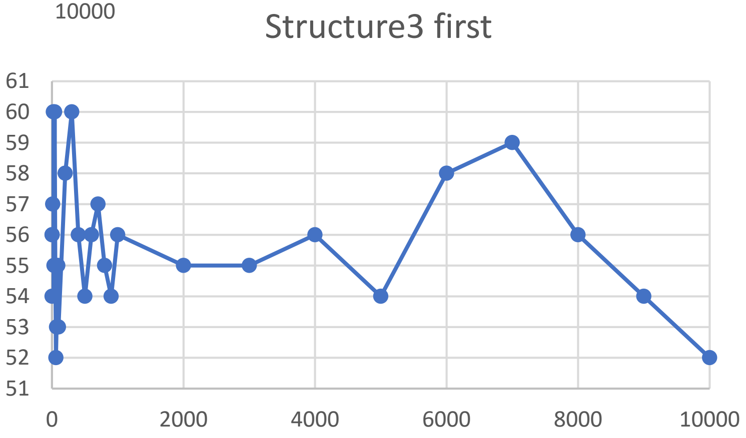
- Structure 2 is a Heap
- O(1) for finding the largest- means that it finds the largest element first found, which is how a maxheap works. This alone could prove the data structure given that the other contains are not O(1).
- O(log n) for removing the largest
- O(n) for other find and remove cases



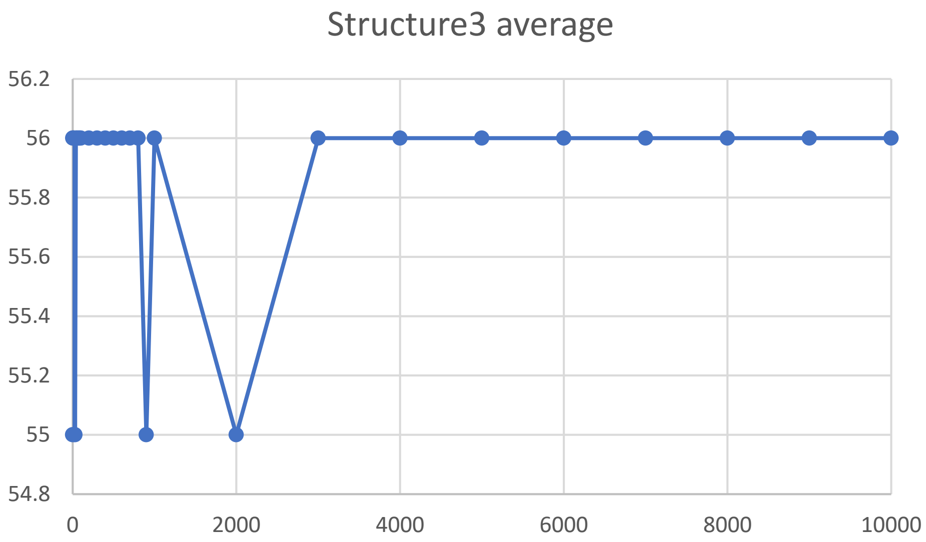
Structure 3: add

O(1)



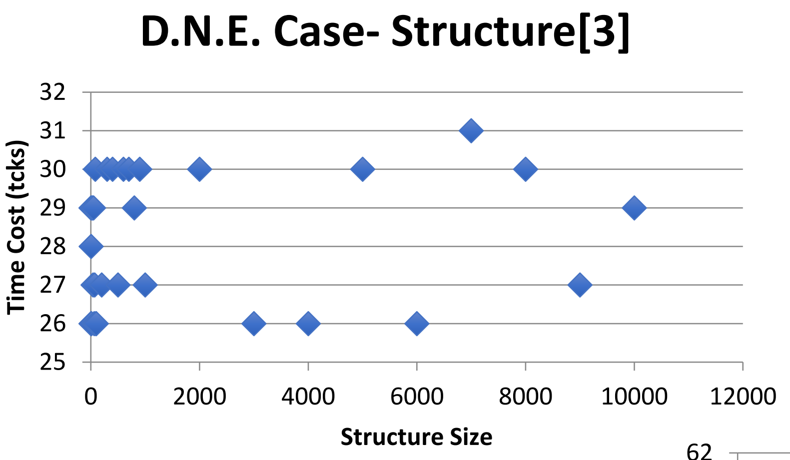


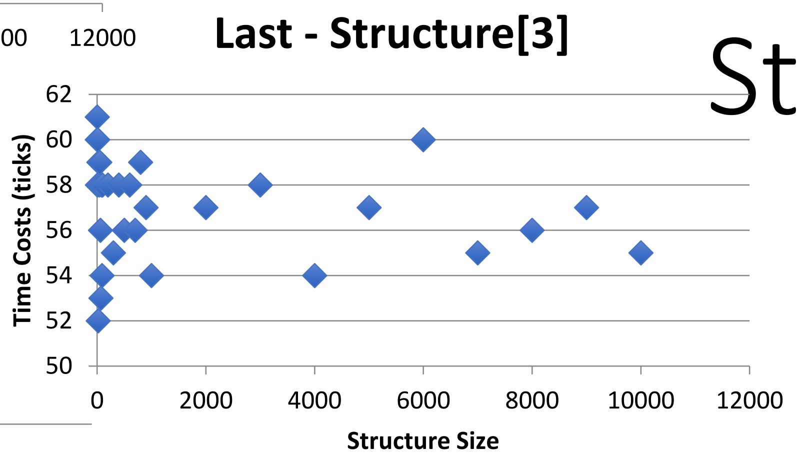
Structure 3: remove

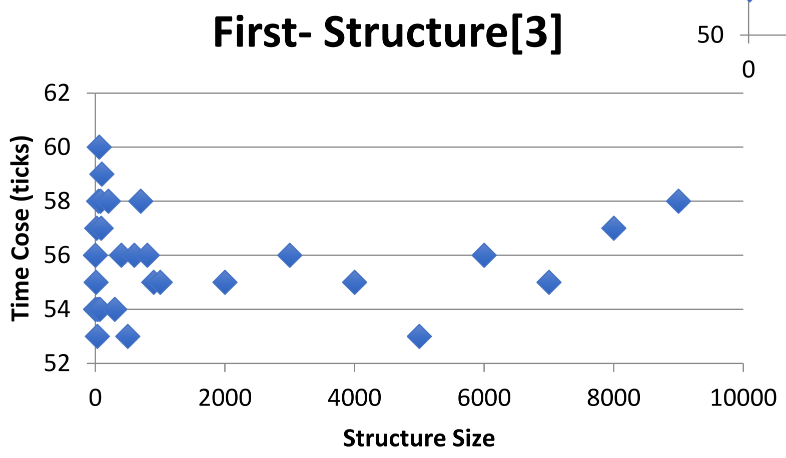


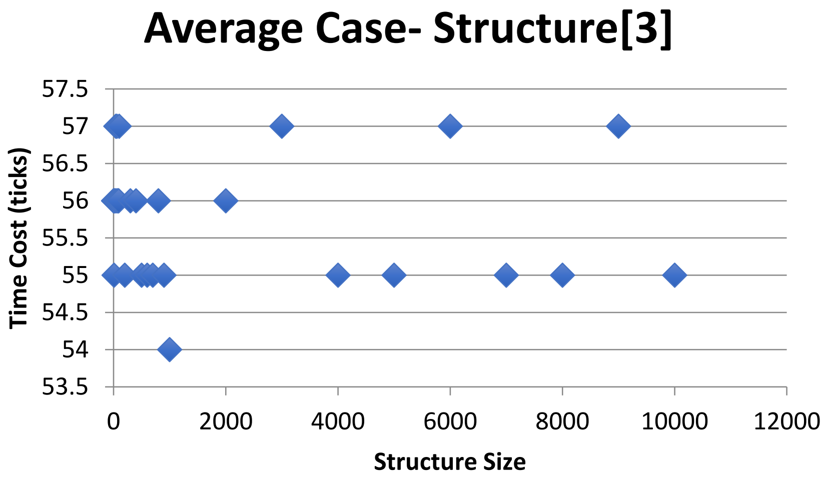
- Last: O(1)
- First: O(1)
- Average: O(1)

Structure 3: contains









Average: O(1)

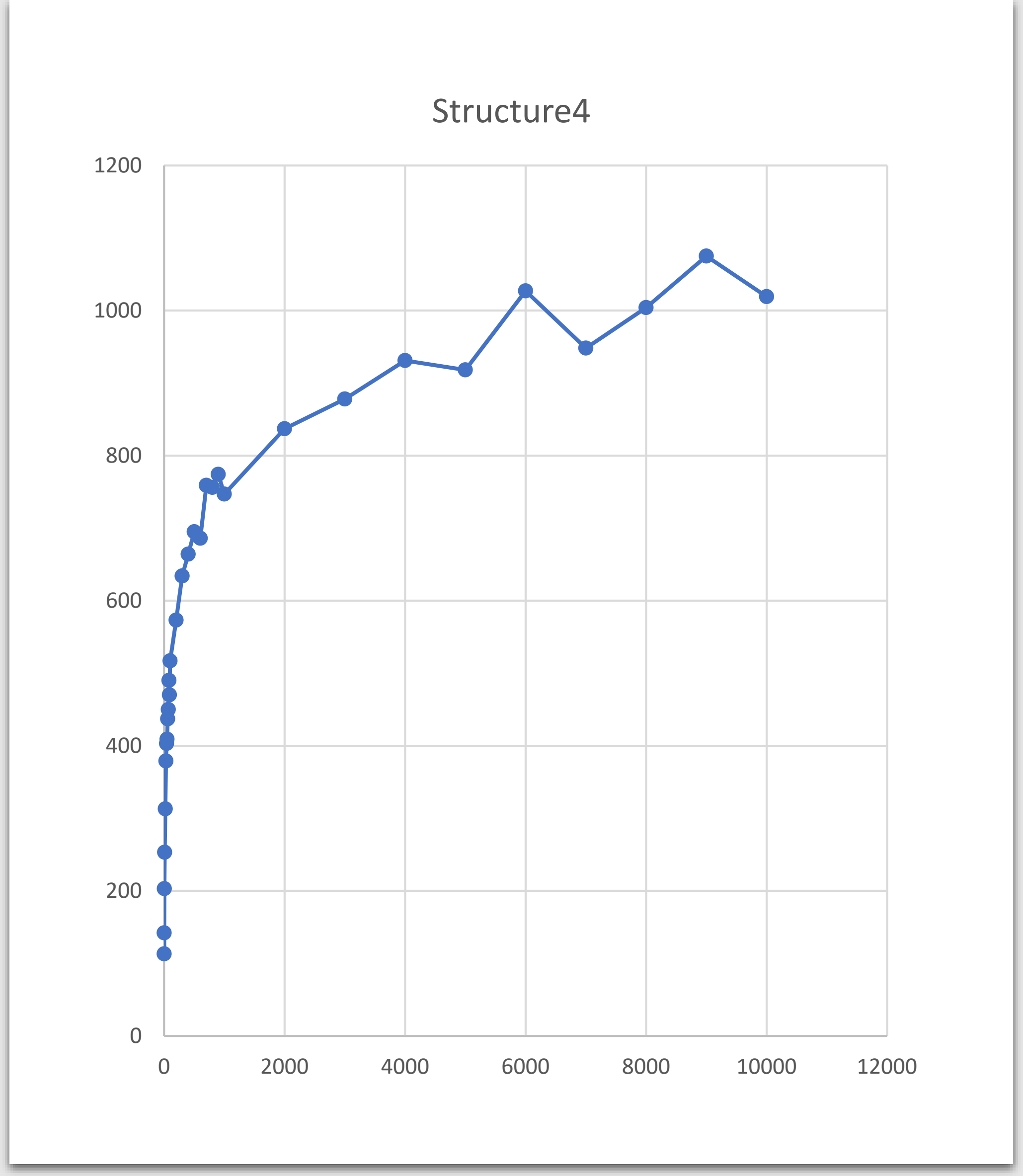
Last(biggest): O(1)

First(smallest): O(1)

N+1(D.N.E): O(1)

Conclusion

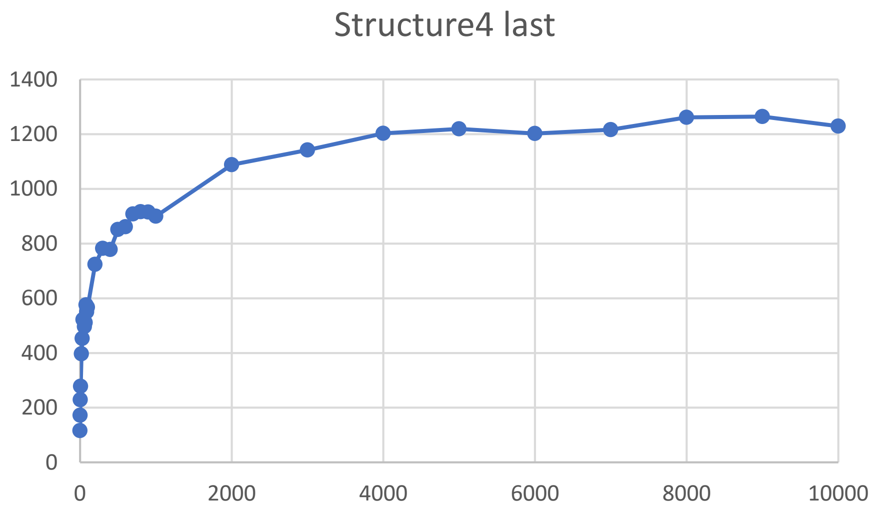
- Structure 3 is a HashSet
- Everything is O(1)- this is fairly straightforward and implicit.

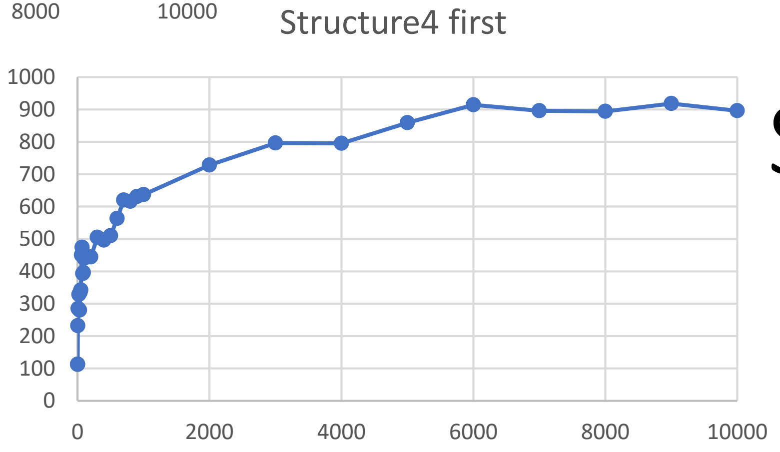


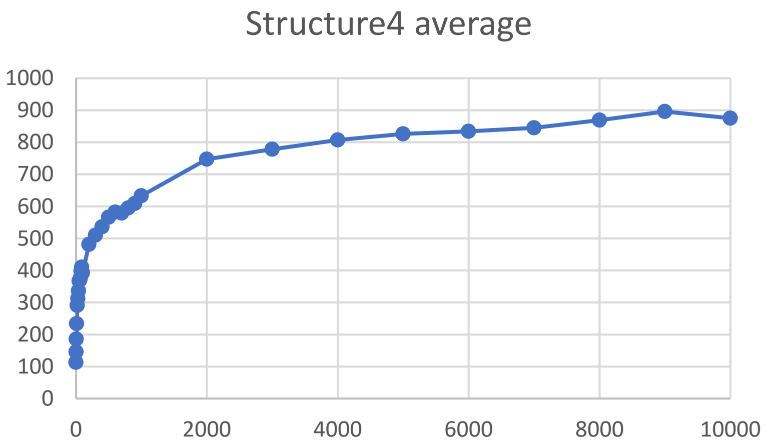
Structure 4: add

O(log(n))

Structure 4: remove

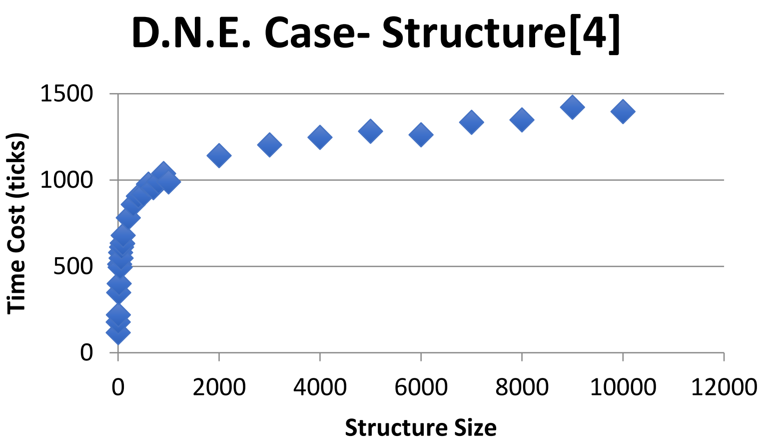


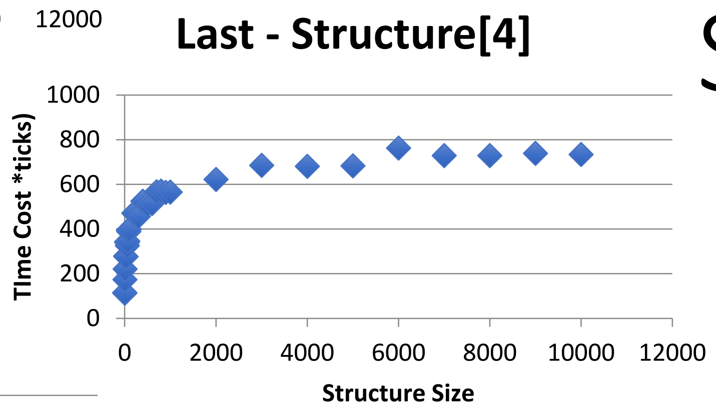


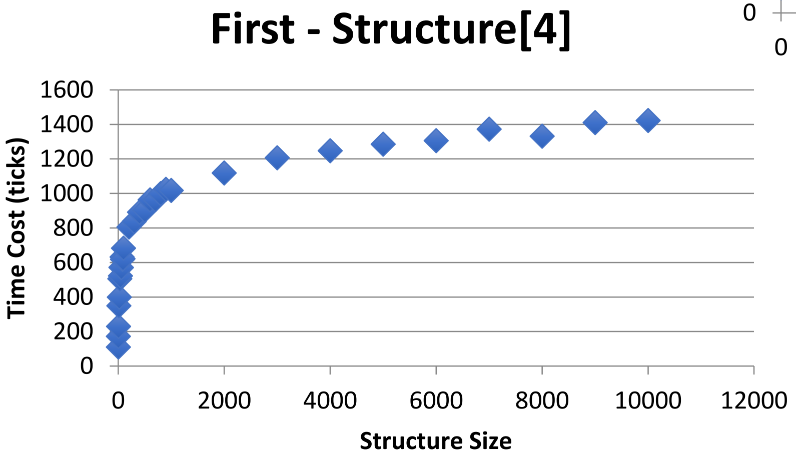


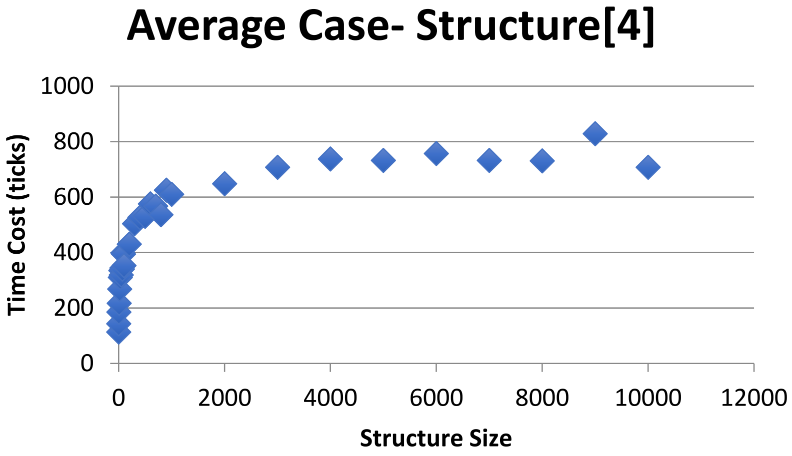
- Last: O(log(n))
- First: O(log(n))
- Average: O(log(n))

Structure 4: contains









Average: O(log(n))

Last(biggest#): O(log(n))

First(smallest#): O(log(n))

N+1(D.N.E): O(log(n))

Conclusion

- Structure 4 is a Binary Search Tree
- Everything is O(log(n)), regardless of max, min, or average, in the remove or contains case. This is enough to prove self-balancing BST.

Final List of Structures:

- Structure[0] = Self-Balancing BST
- Structure[1] = Self- Balancing BST
- Structure[2] = Heap
- Structure[3] = HashSet
- Structure[4] = Self- Balancing BST