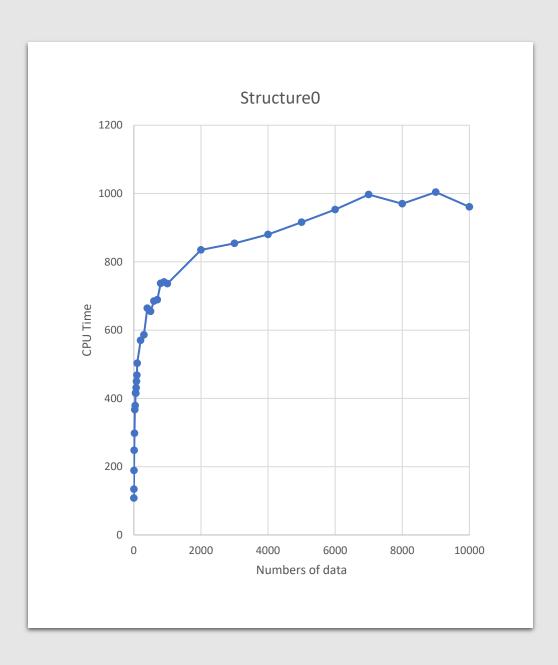
## Project 4

Kepei Lei & PJ Mara

## Procedure

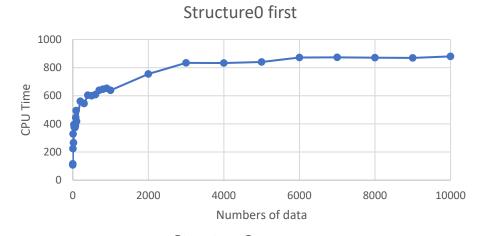
- Each structure of size n (taken from array of n's defined as a constant at the top of the file) is populated with numbers from 0-> n-1
- Add method- Adds a new value (n) into the structure
- Remove Method- Best case removes the value 0. Worst Case is that the value is not in the array- uses value n. Average case picks a number randomly between 0 and n-1.
- Contains Method- Best, Worst, and average are the same as removehowever, 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.

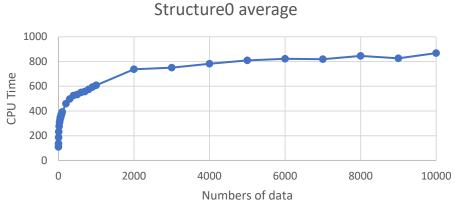


## Structure 0: add

O(log(n))

## Structure0 last 1400 1200 1000 800 400 200 0 2000 4000 8000 10000 Numbers of data





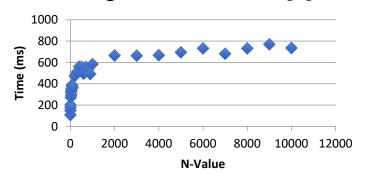
### Structure 0: remove

• Last: O(log(n))

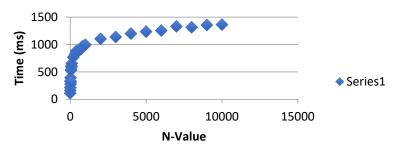
• First: O(log(n))

• Average: O(log(n))

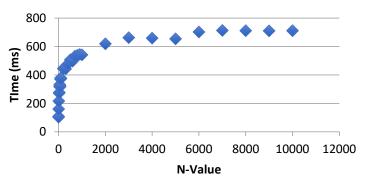
#### **Average Case- Structure[0]**



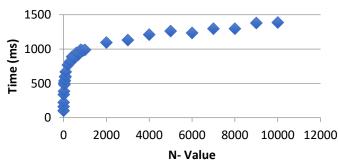
#### **Biggest - Structure[0]**



#### D.N.E - Structure[0]



#### Smallest- Structure [0]



## Structure 0: contains

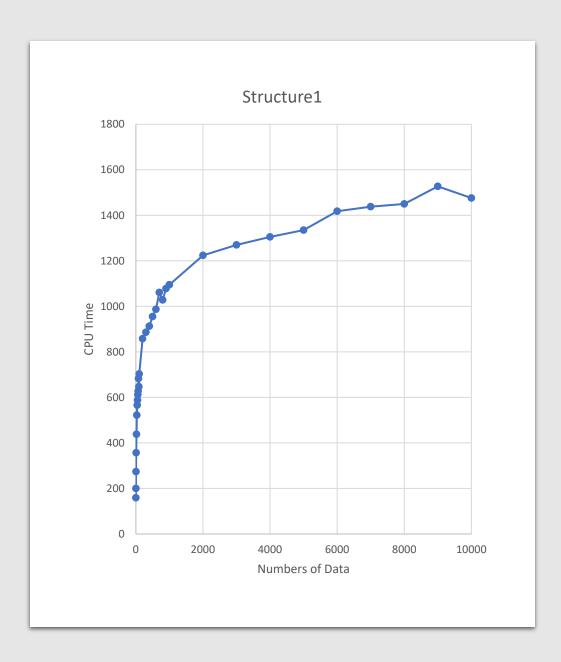
Average: O(log(n))

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

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

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

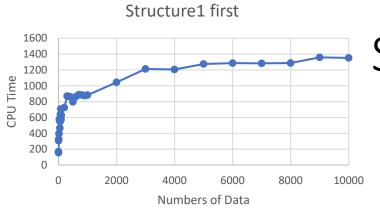
- Structure 0 is a self-balancing Binary Search Tree
- Everything is O(log(n)), regardless of max, min, or average, in the remove or contains case. This is enough to prove self-balancing BST.



## Structure 1: add

O(log(n))

## Structure1 last 2000 1500 500 0 2000 4000 6000 8000 10000 Numbers of Data

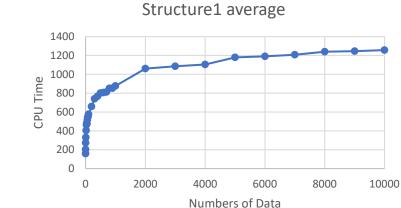


## Structure 1: remove

• Last: O(log(n))

• First: O(log(n))

Average: O(log(n))



D.N.E. - Structure[1] Time Cost (ticks) 2000 1000 500 500 First - Structure[1] 10000 12000 **Structure Size** Last - Structure[1] **Structure Size** Time Cost (ticks)
000
008
40
20 Average - Structure[1] **Structure Size** Time Cost (ticks) 

0 +

**Structure Size** 

## Structure 1: contains

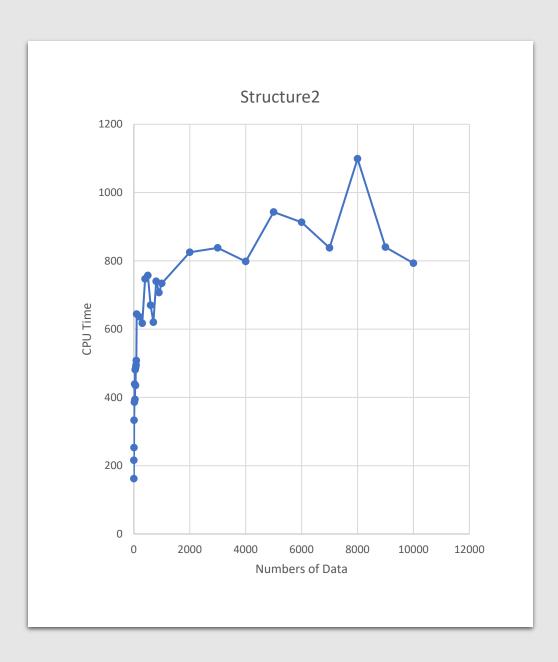
Average: O(log(n))

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

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

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

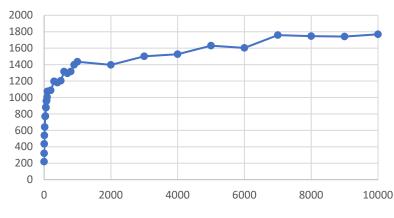
- Structure 1 is a self-balancing Binary Search Tree
- Everything is O(log(n)), regardless of max, min, or average, in the remove or contains case. This is enough to prove self-balancing BST.



## Structure 2: add

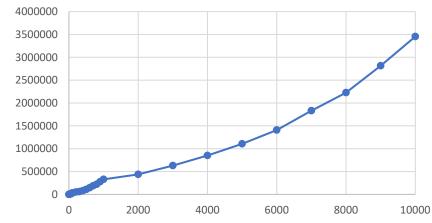
O(log(n))

#### Structure2 last



Structure2 average

#### Structure2 first



## Structure 2: remove

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

D.N.E. - Structure[2] First - Structure[2] **Structure Size** Last - Structure[2] **Structure Size Average Case- Structure[2] Structure Size** 

**Structure Size** 

## Structure 2: contains

Average: O(n)

(with a jump)

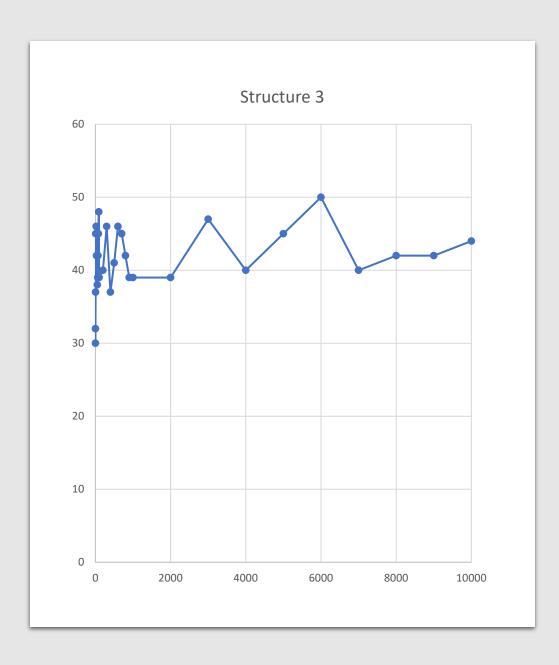
Last(biggest): O(1)

First(smallest): O(n)

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

10000 12000

- 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



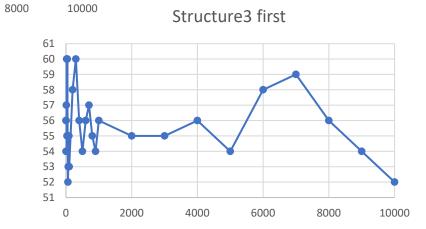
## Structure 3: add

## Structure3 last 61 60 59 58 57 56 55 54 53 52

6000

4000

2000

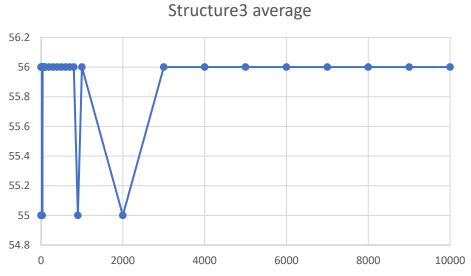


### Structure 3: remove

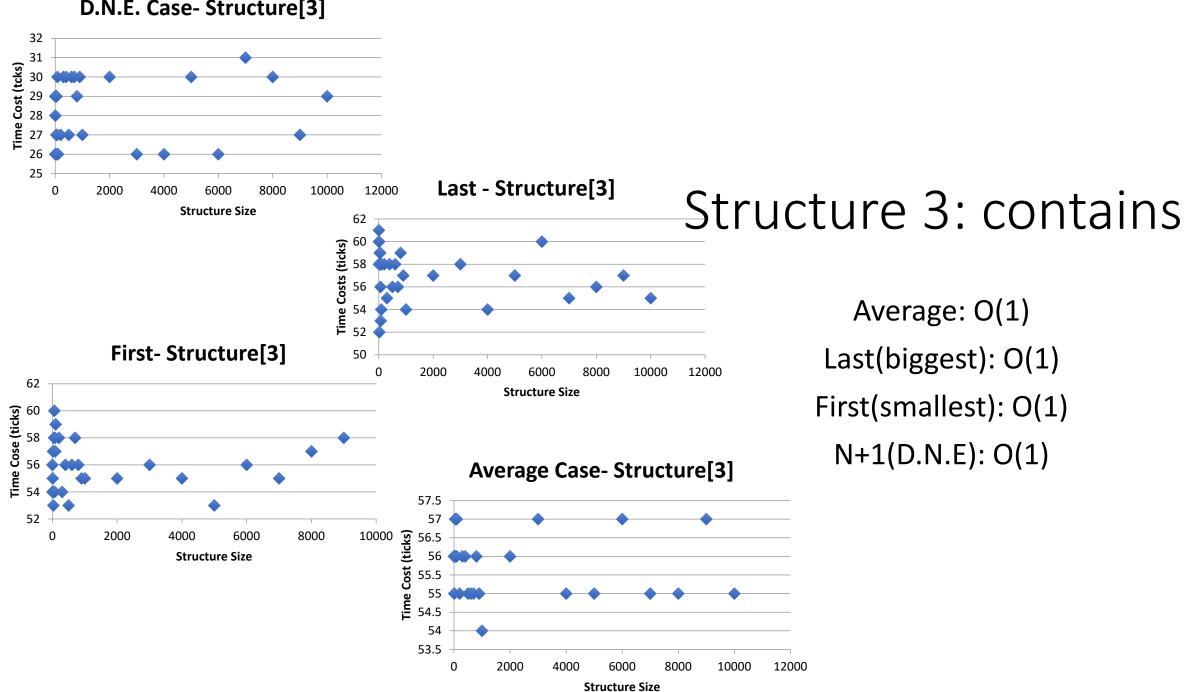
• Last: O(1)

• First: O(1)

• Average: O(1)



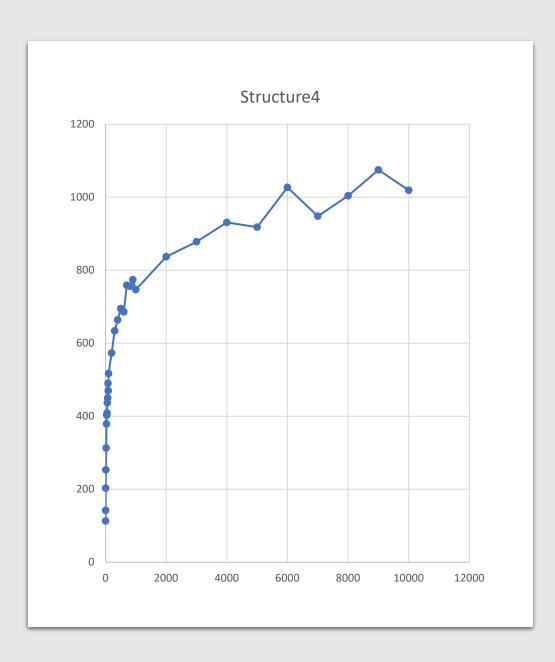
D.N.E. Case- Structure[3]



Average: O(1)

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

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

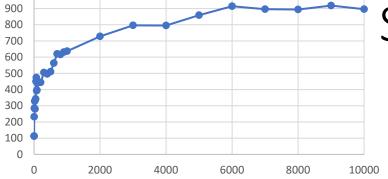


## Structure 4: add

O(log(n))

# Structure4 last 1400 1200 1000 800 600 400 0 2000 4000 6000 8000 10000

## Structure 4: remove

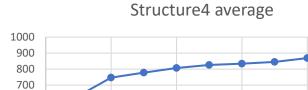


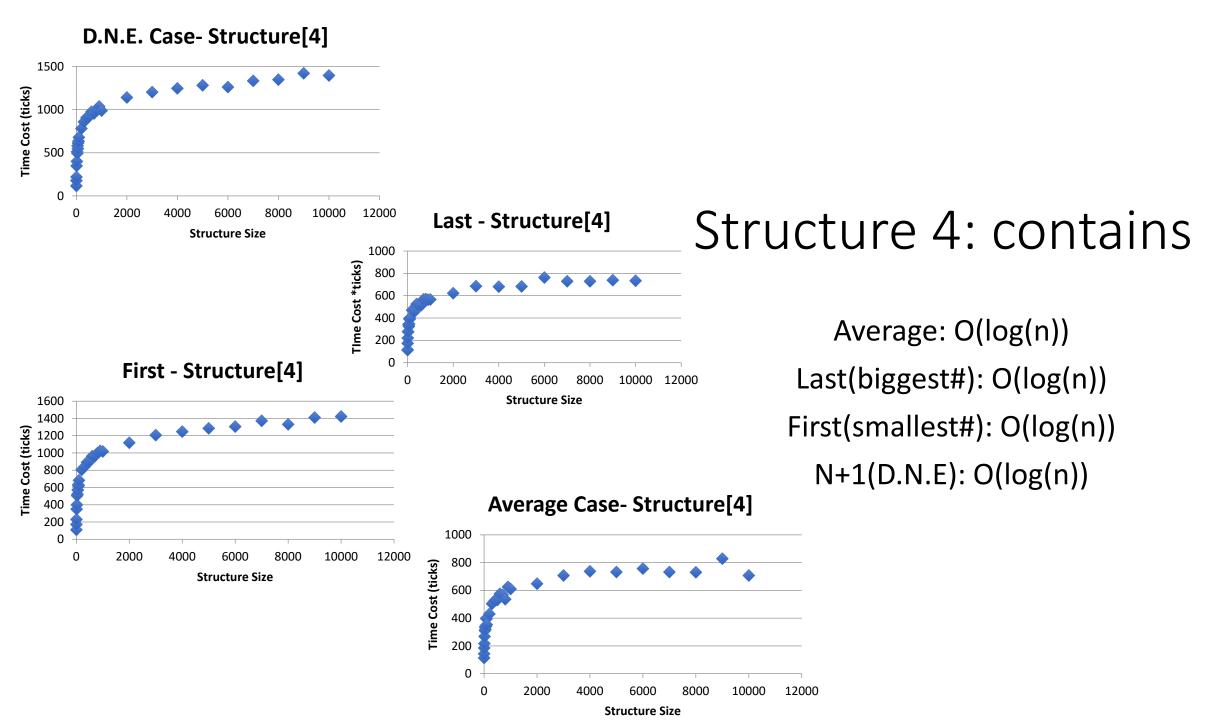
Structure4 first

• Last: O(log(n))

• First: O(log(n))

Average: O(log(n))





- 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