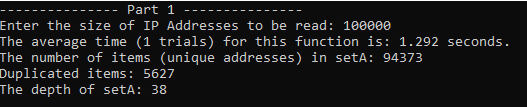
Q1.

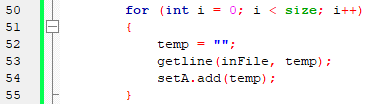
a ) Time required:

|  |  |
| --- | --- |
| Size (n) | Time required (s) |
| 10,000 | 0.085 |
| 100,000 | 1.292 |
| 1,000,000 | 9.441 |



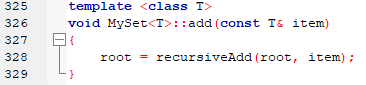
*Time required for size 100,000 for part 1a.*

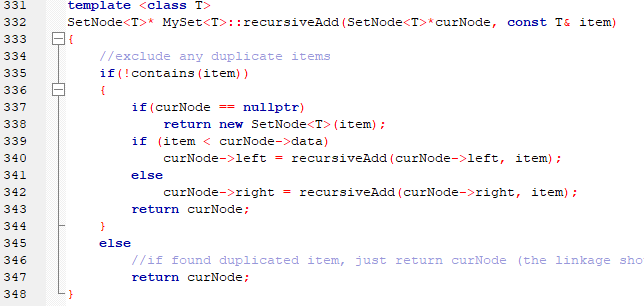
b )



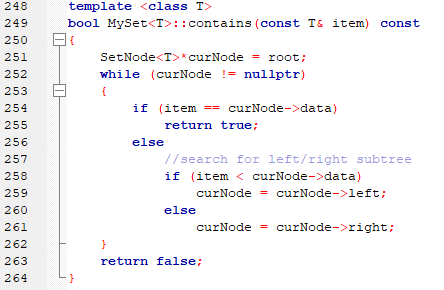
*Code for read in the data for the set*

**BigO for add() = BigO of recursiveAdd()**





*Code for add() and recursiveAdd()*



*Code for contains()*

**BigO for contains(): Average O(logn); (assume balanced BST)**

**BigO for recursiveAdd(): BigO of contains() + Line 337 – 348**

**= O(logn) + O(logn) = 2O(logn) = average O(logn); (assume Balanced BST)**

**BigO for add() = average O(logn); (assume Balanced BST)**

**Time required for reading 10,000,000 records into set:**

**= n \* O(logn)**

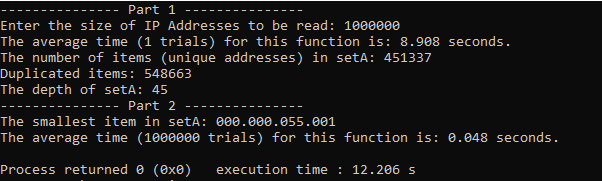
**= 10,000,000 \* log(10,000,000) \* 9.441 / 1,000,000 \* log(1,000,000)**

**= 70,000,000 \* 9.441 / 6,000,000**

**= 110.15s**

Q2.

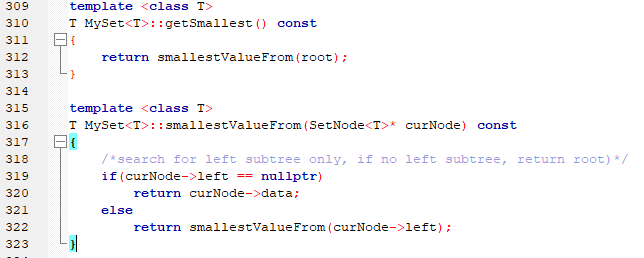
a )



*Code for testing getSmallest for size 1,000,000 for 1,000,000 trials*

Time required for getSmallest(): 0.048s.

b )



*Code for getSmallest() and smallestValueFrom()*

**BigO for smallestValueFrom(): Average log(n); as determined by the depth of the tree**

**BigO for getSamllest(): Average log(n)**

**Time required for getSamllest() to read 10,000,000,000 items:**

**= 10,000,000,000 \* 0.048 / 1,000,000**

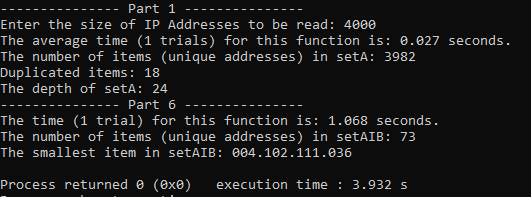
**= 480s**

Q6.

a ) Time required:

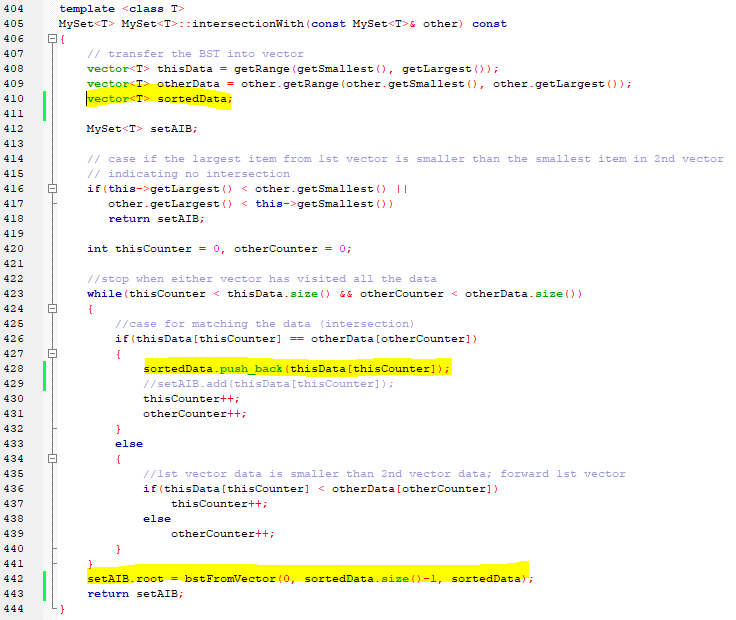
|  |  |
| --- | --- |
| Size (n) | Time required (s) |
| 2,000 | 0.275 |
| 4,000 | 1.068 |
| 8,000 | 3.854 |

\*\* when counting the time required for the function inserctionWith(), the copy constructor is also included in this exercise to return the intersection between 2 sets.

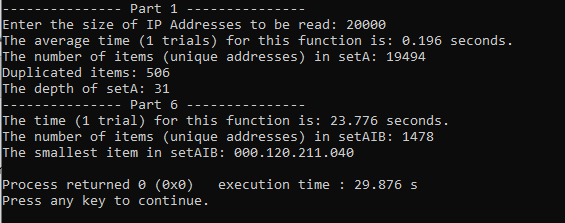


*Time required for size 4,000 for part 6a.*

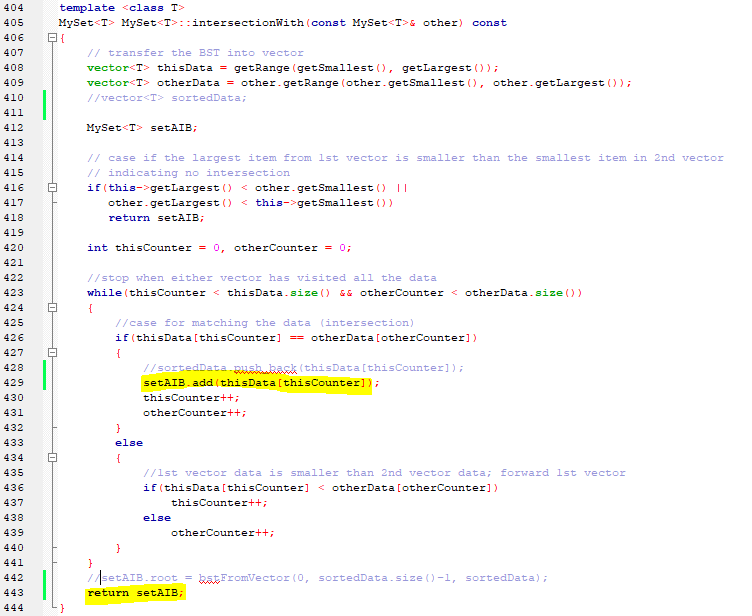
b )



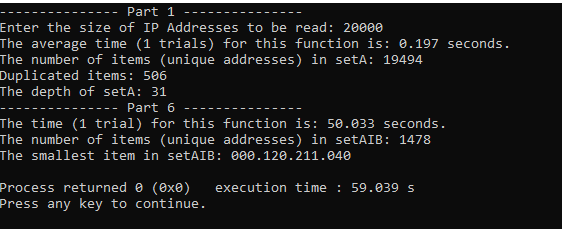
***Code for part 6 (ver.A using sorted data to form BST) .***



*Time required to do 20,000 sample size for intersectionWith() for ver. A*



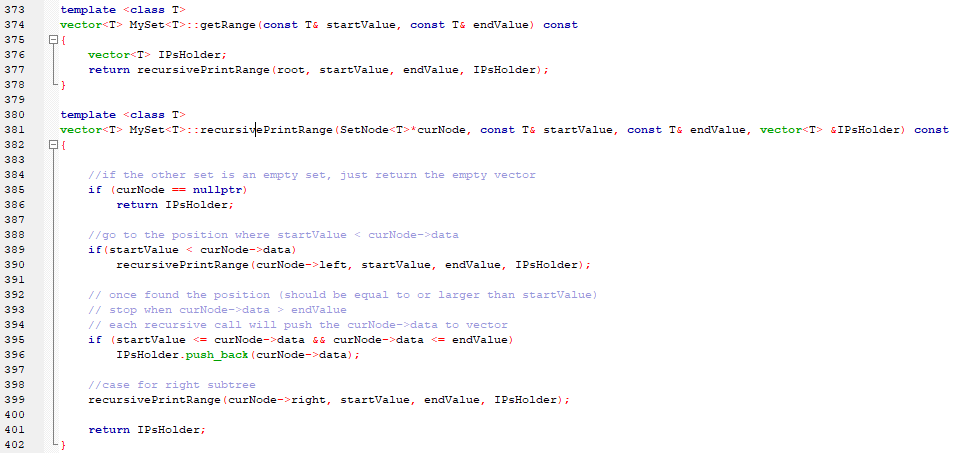
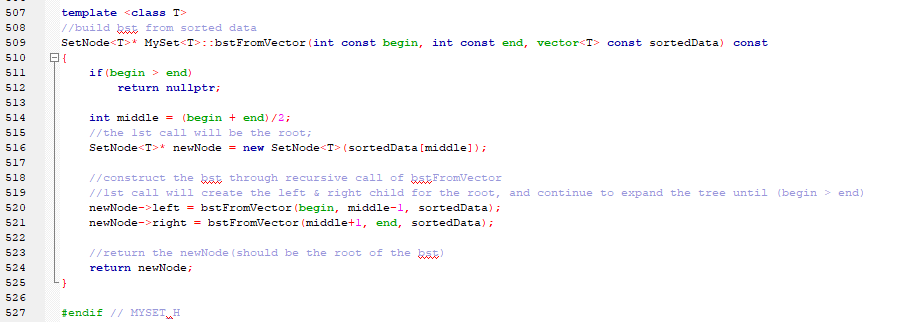
***Code for part 6 (ver.B BigO: O(nlogn) using add() to form BST) .***



*Time required to do 20,000 sample size for intersectionWith() for ver. B*

From the above, I have implemented both ver.A and ver.B for intersectionWith() for comparison. ver.A (using sorted data to form BST) runs faster than ver.B (using add() function to form BST) when the sample size is 20,000.

**BigO for other helper functions:**

1. **BigO for getSmallest() : O(logn) as determined above**
2. **BigO for getLargest() : O(logn); similar to getSmallest()**
3. **BigO for getRange() : O(n)**
   1. **BigO for recursivePrintRange() = 2 \* O(n/2) + c = O(n); the worst case is to travel to the smallest to the largest item (go through all nodes between 2 sides of subtrees)**
4. **BigO for bstFromVector(): 2 \* O(n/2) + c = O(n); as half of the data in vector is in left-subtree, the middle will be the root of the BST, and the half of the data is for right-subtree.**

**Therefore BigO for ver.A intersectionWith():**

**Line 408-409: 2 \* O(n) = O(n)**

**Line 423: O(m+n); here m refers to no. of data in setA, n refers to no. of data in setB**

**Line 426-441: constants**

**Line 442: O(n)**

**Overall BigO for ver.A intersectionWith(): O(m+n)**

**(Optional) BigO for ver.B intersectionWith():**

**Line 408-409: 2 \* O(n) = O(n)**

**Line 423-441: O((m+n)\*logX); here m refers to no. of data in setA, n refers to no. of data in setB; X refers to the intersection data (can also be interpreted as the depth of the intersection BST)**

**\*\* constants are ignored**

**Overall BigO for ver.B intersectionWith(): O((m+n)\*logX)**

**It can be explained that why ver.A is faster than ver.B. when there are large intersection data (e.g. 1478 when the overall input size is 20,000). logX will there be significant comparing to ver.A to explain the difference between the time taken between 2 versions.**