Computer & Systems Engineering Department

Data Structures and Algorithms

Implementing Self Balanced BSTs (AVL & Red Black)

Contributors:

	Name	ID
1	Adel Mahmoud Mohamed Abdelrahman	20010769
2	Amr Ahmed Abdelazim	20011037
3	Marwan Essam Eldin	20011859
4	Mohamed Amr Abdelfattah	20011675

Time Analysis of the project parts:

1. Time Complexity of AVL basic functions:

Function	Time Analysis
Insert	O ($\log_2 n$)
Delete	O ($\log_2 n$)
Search	O ($\log_2 n$)
Size	O (1)
Height	O (1)

2. Time Complexity of Red-Black basic functions:

Function	Time Analysis
Insert	O ($\log_2 n$)
Delete	O ($\log_2 n$)
Search	O ($\log_2 n$)
Size	O (1)
Height	O (n)

3. Time Complexity of English dictionary basic operations:

Function	Time Analysis
Initialize	O (1)
Insert	O ($\log_2 n$)
Delete	O ($\log_2 n$)
Search	O ($\log_2 n$)
Batch insert	O (n $\log_2 n$)
Batch delete	O (n $\log_2 n$)
Size	O (1)
Height	O (n) **(for RB)** O (1) **(for AVL)**

Comparison between AVL and RB under different test sizes measuring different aspects:

1. Time taken for insertion, deletion, and height of n elements:

N	AVL Tree			Red-Black Tree		
	Insert	Delete	Height	Insert	Delete	Height
100	11982 µs	2472.8 µs	8	9386.8 µs	2228 µs	8
1000	22543.2 µs	3755.9 µs	12	15842.4 µs	5587.3 µs	12
5000	31261 µs	21753.7 µs	15	25717.2 µs	11199.6 µs	15
10,000	45668.5 µs	23836.5 µs	16	30798.9 µs	15571.9 µs	16
50,000	131903 µs	87064.9 µs	19	71527.2 µs	31281.8 µs	22
100,000	171664 µs	114667µs	20	102510.5 µs	53099.4 µs	26
200,000	283562µs	158707 µs	21	172087 µs	102067.4 µs	25
400,000	545516 µs	291288 µs	22	316833.6 µs	135412.3 µs	27
500,000	583974 µs	339629 µs	23	378528.4 µs	176426 µs	28
600,000	749134 µs	427172 µs	23	454226.7 µs	229001.5 µs	28
800,000	927828 µs	591314 µs	23	526490.6 µs	269388.7 µs	29
1,000,000	1048402 µs	675250 µs	24	624711.4 µs	343832.9 µs	30

2. Time taken for search in n elements:

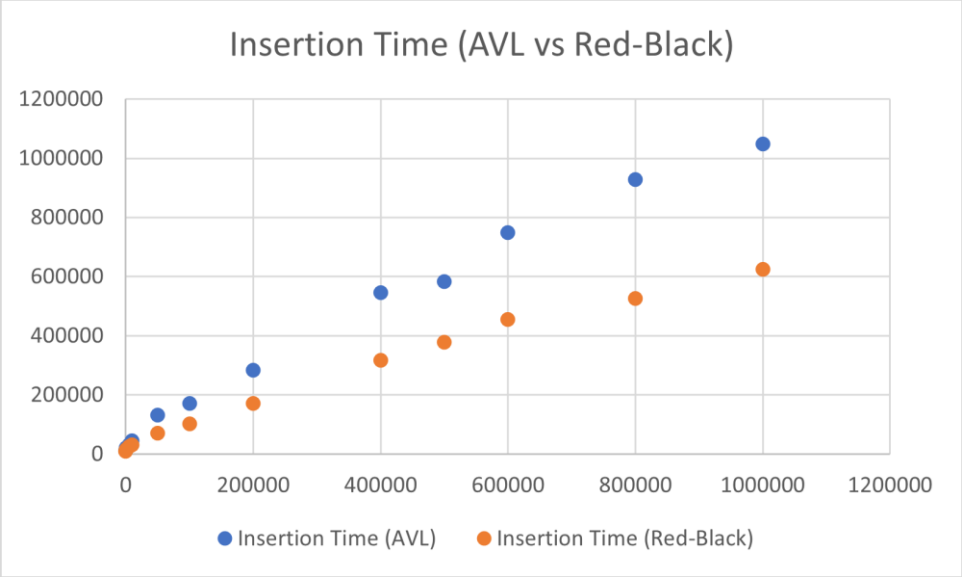
N	AVL tree	RB tree
100	**68.275 µs**	**67.55 µs**
1000	**55.525 µs**	**63.35 µs**
10000	**54.6 µs**	**65.35 µs**

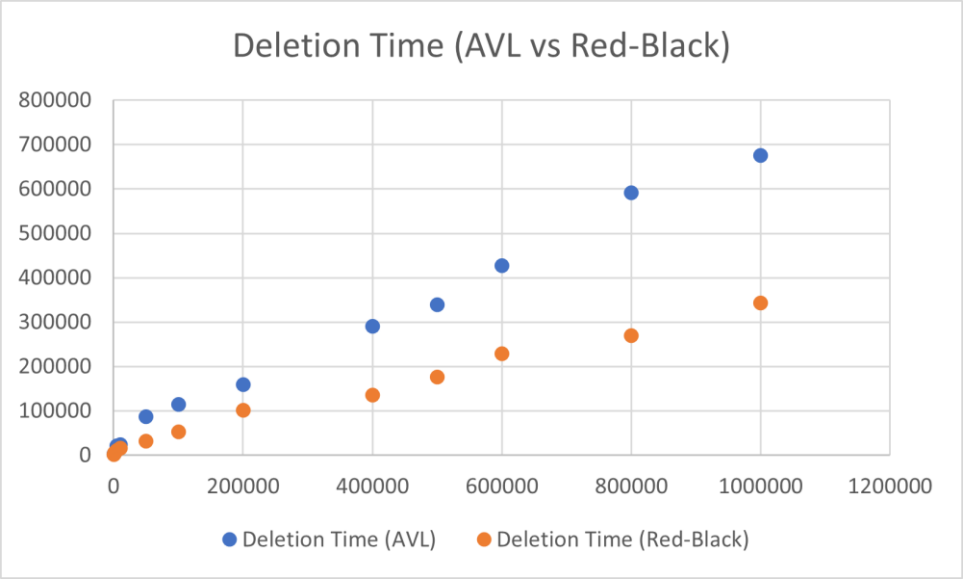
So now we can get the weighted avg of the time as follows:

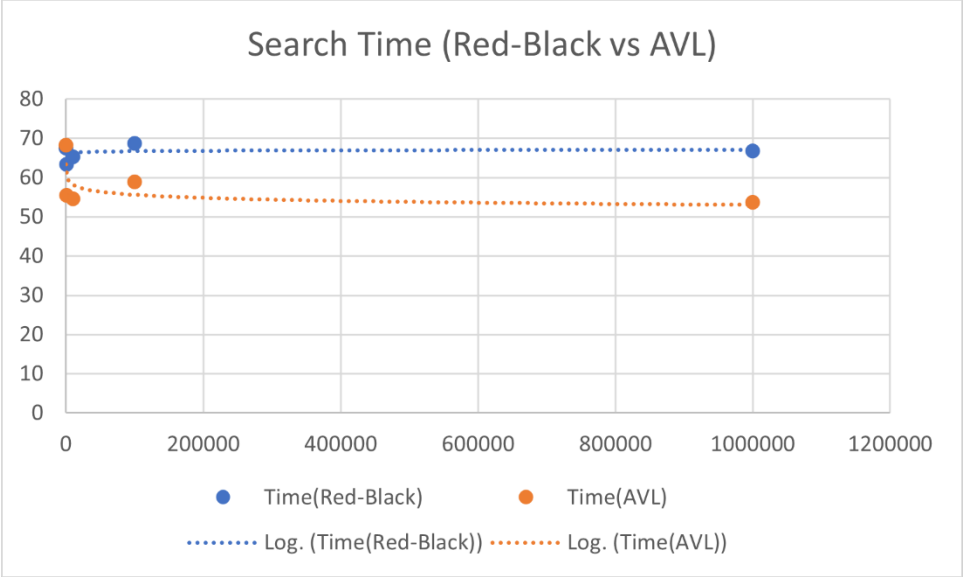
(Time / one operation)

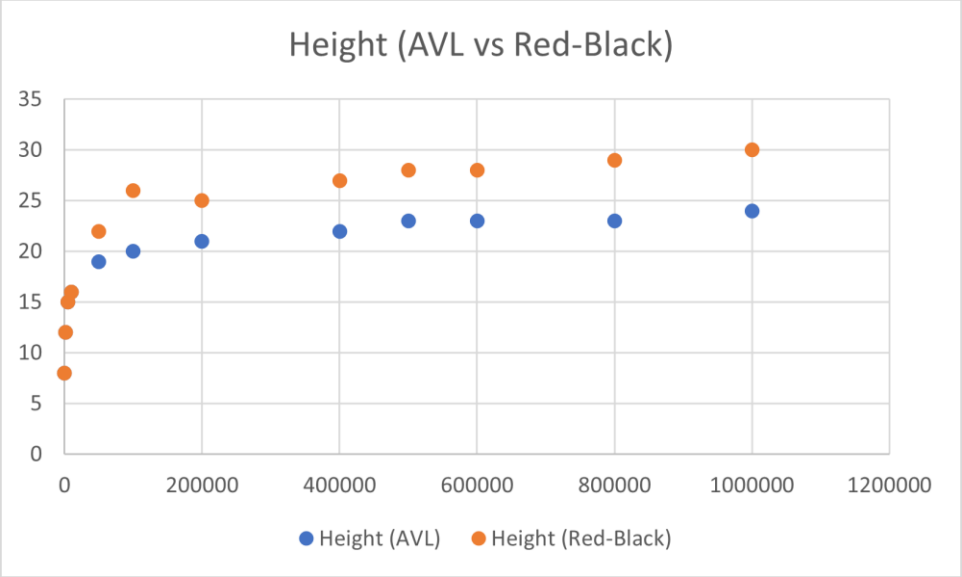
	AVL	Red-Black
Insertion	**13.745 µs**	**10.403 µs**
Deletion	**3.536 µs**	**2.924 µs**
Searching (for one element)	**0.14887568 us**	**0.149227895 us**

Statistical confirmation of the results we got:




Deletion Time (AVL vs Red-Black)
800000
700000
600000
500000
400000
300000
200000
100000
0
0
200000
400000
600000
800000
1000000
1200000
Deletion Time (AVL)
Deletion Time (Red-Black)


Search Time (Red-Black vs AVL)
80
70
60
50
40
30
20
10
0
0
200000
400000
600000
800000
1000000
1200000
Time(Red-Black)
Time(AVL)
Log. (Time(Red-Black))
Log. (Time(AVL))



Conclusion:

From the above results, it has been evident that the AVL tree involves more rotations in insertion/deletion while preserving a shorter height. Therefore, AVL trees are well suited for situations where a lot of search operations are required while having occasional insertion/deletion operations. This was evident in the dictionary application as the AVL tree was faster at searching but slower at insertion/deletion.

On the other hand, Red-Black trees involve less rotation operations in insertion/deletion as Red-Black tree conditions are more relaxed. Due to the same reason, the red-black tree has a longer height, where one path can reach at most twice the length of the other path in the tree. This makes the Red-Black tree a candidate for situations where a lot of insertion/deletion operations are required with less frequent search operations. This appeared in the dictionary application when the Red-Black tree insertion/deletion operations were faster but slower at searching.

We can say that there is a tradeoff between maintaining a minimal tree height and minimizing the number of rotation operations. This depends on the nature of

the application in which we are using the trees, i.e., the frequency of deletions/insertions [writes] vs the frequency of searches [reads].