Assignment 4: Binary Search Trees

ID: 2017120486 Name: 서천함

In each round of exam, I tested various erase/find ratios at 20:80, 40:60, 50:50, 60:40, 80:20 of AVL Tree and Search Tree respectively.

In order to call erase and find in a random order, I initialized the array and assigned 20%/40%/50%/60%/80% of the array elements with value 1. Then I used random_shuffle to randomize the order of 1 and 0. When the foo[i] == 1, we call function erase; else we call function find. Therefore various ratios can be implemented.

```
vector<int> foo(nElem, 0);
for(int i = 0; i < 0.8 * nElem; i++){        // 0.2, 0.4, 0.5, 0.6, 0.8
    foo[i] = 1;
}
random_shuffle(foo.begin(), foo.end());        'random_shuffle<std::__1::__
```

Test results are as follows. The measurement unit is second.

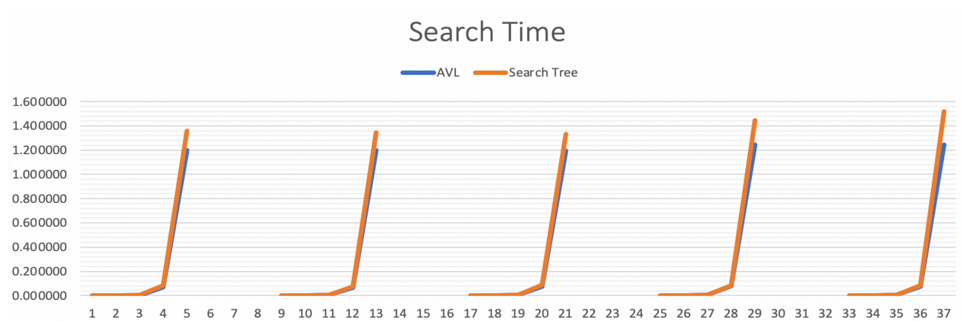
			AVL				**Search Tree**	
Experiment 1								
	Size	Insert	Find	Erase:Find(20:80)		Insert	Find	Erase:Find(20:80)
	100	0.00013	0.000025	0.000034		0.000042	0.000023	0.000023
	1000	0.001625	0.000297	0.000286		0.000467	0.000339	0.000344
	10000	0.017693	0.003605	0.003315		0.005065	0.004502	0.004307
	100000	0.243048	0.070628	0.080457		0.089898	0.082296	0.083821
	1000000	3.836982	1.199586	1.199299		1.241744	1.356995	1.400103
Experiment 2								
	Size	Insert	Find	Erase:Find(40:60)		Insert	Find	Erase:Find(40:60)
	100	0.000127	0.00002	0.000019		0.000036	0.000024	0.000024
	1000	0.00156	0.000275	0.000277		0.000443	0.000317	0.000324
	10000	0.018365	0.003664	0.003812		0.005741	0.0044	0.004608
	100000	0.258873	0.064705	0.070822		0.084654	0.07373	0.064809
	1000000	3.761787	1.199047	1.255865		1.289704	1.343575	1.333388
Experiment 3								
	Size	Insert	Find	Erase:Find(50:50)		Insert	Find	Erase:Find(50:50)
	100	0.000151	0.000018	0.000016		0.000028	0.000019	0.000018
	1000	0.00147	0.000289	0.000287		0.000494	0.000346	0.000371
	10000	0.019516	0.004005	0.003913		0.005619	0.004744	0.004599
	100000	0.239671	0.073114	0.074079		0.093379	0.087272	0.069876
	1000000	3.800337	1.196332	1.211943		1.229243	1.33113	1.396156
Experiment 4								
	Size	Insert	Find	Erase:Find(60:40)		Insert	Find	Erase:Find(60:40)
	100	0.000126	0.00002	0.000021		0.000033	0.000022	0.000021
	1000	0.0015	0.000275	0.000274		0.000447	0.000332	0.000337
	10000	0.019036	0.003984	0.004315		0.005832	0.004635	0.004813
	100000	0.27544	0.080991	0.083484		0.095466	0.085467	0.072311
	1000000	4.054769	1.244169	1.275241		1.352083	1.446585	1.447609
Experiment 5								
	Size	Insert	Find	Erase:Find(80:20)		Insert	Find	Erase:Find(80:20)
	100	0.000126	0.000019	0.00002		0.000033	0.000022	0.000023
	1000	0.001459	0.000306	0.000308		0.000467	0.000305	0.000279
	10000	0.020229	0.004188	0.004494		0.006624	0.005161	0.005177
	100000	0.272846	0.074101	0.073569		0.086978	0.08305	0.082348
	1000000	3.86674	1.242701	1.25912		1.417735	1.517694	1.458753

Tree type		Average	Worst
Binary search tree	Space	Θ(n)	O(n)
	Insert	Θ(log n)	O(n)
	Search	Θ(log n)	O(n)
	Delete	Θ(log n)	O(n)
AVL tree	Space	O(n)	O(n)
	Insert	O(log n)	O(log n)
	Search	O(log n)	O(log n)
	Delete	O(log n)	O(log n)

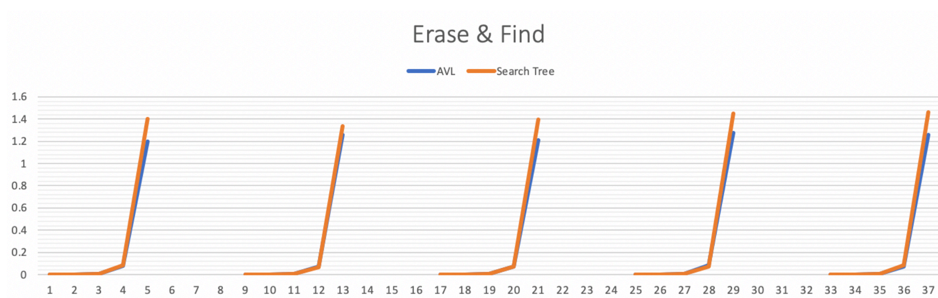
Theoretically, BST and AVL Tree has similar insert, search, delete performance of O(log n) on average. However, based on the results of 5 experiments, we can find that:

1. **Insertion**: Search Tree performs better than AVL tree in terms of as size increases. The reason might be that we tested random insertion here so AVL tree has to constantly balance itself. Constant balancing adds heavy overhead to the AVL tree algorithm. Therefore search tree outperforms AVL tree in the case of random insertion.

2. **Search:** Basically their performances are similarly. However, we can see that when the size becomes larger, AVL tree performs slightly better than search tree.



3. **Mix of Erase & Find**: Both perform similarly. As size increases, AVL tree performs slightly better than search tree.



5. When the ratio of erase increases, both AVL Tree and Search Tree shows insignificant difference in the performances.