Height-Balanced Binary Search Tree

A binary search tree is a very basic and useful data structure. Unlike a hash table or array, it does not require the user to know the size of the set it will contain to efficiently build or resize a container. It has much better search speed than an unsorted array or a linked list, although generally slower than a hash table for equality search. It sorts the keys, and so can provide a sorted listing, a range of included key objects, or an adjacent key either higher or lower. A sorted array can be searched as easily as a binary tree, with less storage (no pointers/references, height or other balance values) but is very expensive to resize. If the set once built will be entirely static, one can sort the keys and put the data in a sorted array for the same bisection search speed as a tree search and no pointer/reference and balance storage space overhead.

The weakness of a binary tree is balance. Without a balance mechanism, a tree built with ordered keys becomes a linked list, but with twice as much pointer/reference storage space. A practical binary tree needs some sort of balance mechanism, preferably low overhead in storage and processing. A balanced binary tree does not need to be perfectly balanced. Low balance overhead is generally a much more significant factor than the perfection of the balance. A height balanced tree is probably the simplest and least overhead balanced tree. One 8 bit byte is sufficient to record the height of the sub-tree at each node, as even a worst case height balanced tree of 255 levels has a unreachable large size. This is the same storage overhead as a red-black tree or an AVL tree with binary relative height indicators, as their balance storage is also 1 byte. Such a height balanced tree is a simplified AVL tree.

Duplicate keys are a concern when using any keyed set or map. Depending on the application, you can choose to:

- ignore them, discarding any insert data,
- update the entire keyed object, discarding the old data,
- update the content of the keyed object, such as incrementing a count of that word, by returning a pointer/reference to the caller,
- have the keyed object be some sort of list of all objects for that key, by returning a pointer/reference to the caller so it can be added to the list,
- place the item on the right or left of the identically keyed item, for height balance trees chosen by height, which affects many aspects of tree processing. The set or map now becomes a multi-set or multi-map, respectively.

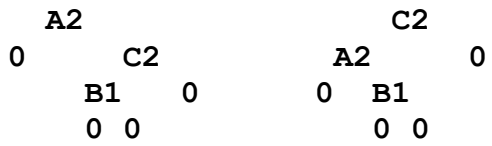
With low balance overhead, a height balanced tree can be kept in balance with every insertion or deletion, both ready for any search and to efficiently support the next insertion or deletion. Repairing the balance of a tree that is far out of balance is a much harder task, and so not a good approach. Balance and height needs to be checked and adjusted for any sub-tree where a node is added or

removed. Balance processing needs to be done leaf to root, so every node being balanced has two balanced sub-tree children, keeping the whole tree balanced.

The height of every node is 1 greater than the height of the higher child. If the difference in height of any node's two children exceeds one, it needs to be balanced. Balance improvement is most efficiently achieved by rotation. With just the writing of 3 pointers/references, large amounts of data can be re-positioned.

However, rotation comes with a catch. In rotation, the greater height side inside grandchild or quarter moves to the opposite side as the inside child of the old root, grandchild of the new root. If the inside quarter is the source of the imbalance, the problem will just swaps sides of the root of this sub-tree. So, while a balance difference of 1 is acceptable at the local root, for rotation, any greater height side inside sub-tree cannot be greater in height than the greater height side outside sub-tree. Before you rotate to balance a local root, you check the greater height side for a greater height inside, and if present, rotate that child toward the outside. Then you can safely rotate the parent.

> Consider this set of key-heights after an insert B. The new B1 raises its parent to a C2, but still in a legal balance on C2. The root A2 (root height has not been reset) has children 0 and 2, so a balance rotate right to left is desired, but without first doing a child rotate, things are still exactly as bad:



The procedure for an insert is to search the tree recursively until you find a null pointer/reference, and install the new node there as height 1. Having 2 null pointers/references as children says you are a leaf, so a null pointer/reference is considered a sub-tree of height 0, and the leaf height is 1. If you find a duplicate key, you can design your tree to ignore the update, replace the key object, or leave the object unmodified. The final insert call will return a pointer/reference or Boolean value, both to indicate if the tree needs balance and height processing, and to allow the inserting process to deal with a key collision, such as by adding a duplicate key object to a list of objects at that key, or updating that object, such as incrementing an internal count of duplicate keys. When the insert call of a sub-tree returns, if it indicates a real insert, the current node must recheck its balance, possibly rotate to restore adequate balance, always recalculate its height, and finally return the same value as returned from the call it made. This both balances the tree and sets height values, leaf to root.

The process for a delete is very similar. A search for the target either returns either a Boolean, a null pointer/reference, or a pointer/reference to the deleted

node or keyed object, indicating if a real deletion has occurred and the tree height and balance need to be reprocessed. If the key has a hit, the node needs to be removed from the tree. If either child pointer/reference is null, you replace the node pointer/reference to this node with the other child. For nodes with two children, if one child has more height, then a successor needs to be elevated from the higher child, else either child can provide a successor. The successor of a left child is the maximum key of that sub-tree, else it is the minimum key of the right sub-tree. As the successor is removed and returned to the recursive callers. the balance and height of those sub-trees are reprocessed at each parent node. The successor is installed in place of the deleted key, the balance and height reprocessed, and the pointer related to the deletion is returned. As each delete call returns, if there was a real delete, each level reprocesses its balance and height.

Both insert and delete can take log2(N) compares to find the key, like 20 compares in a 2^20 -1 or 20 level, 10 meg., perfectly full tree. If there is an actual insert or delete, there will be log2(N) balance checks, possible balance repairs, and height adjustments. If a deleted key is high in the tree, there are fewer compares to find it but more to find the successor, and vice versa, but in any case the balance and level is reprocessed from successor up to root. Inserting ordered data into this tree, a worst case without balance, in amounts 2^N – 1, results in a perfectly balance tree of height N.

While some may obsess about perfect tree balance, the worst case height balanced tree may have much more height than a optimally balanced tree, but the actual performance is virtually identical. The average number of comparisons to find a key is generally no more than one more. The average number of comparisons to find a key is not present is also no more than one more. The greater overhead to create a tree or better balance is huge compared to any difference in search speed. The storage space of this tree is identical.

Below are calculations of worst case height balanced tree sizes, comparison counts, and calculations of binary full tree sizes and comparison counts. For comparison, a perfect full binary tree of 20 levels has over 19 average compares to find a key, and exactly 20 to find not in set, for over a million keys. A worst case height balanced tree has 28 to 29 levels, 19.5 to 21.2 comparisons. The chance, with random activity, of a worst case height balanced tree is as unlikely as achieving a perfectly balance tree, so actual performance is usually much better than worst case.

Height Balanced Worst Case Trees

Levels	WSize	WAvg=D	WAvg!=D
1	1	1.0000000000000000	1.0000000000000000
2	2	1.5000000000000000	1.6666666666666667
3	4	2.0000000000000000	2.3999999999999999
4	7	2.5714285714285716	3.1250000000000000
5	12	3.1666666666666665	3.8461538461538463
6	20	3.7999999999999998	4.5714285714285712
7	33	4.4545454545454541	5.2941176470588234
8	54	5.1296296296296298	6.0181818181818185
9	88	5.8181818181818183	6.7415730337078648
10	143	6.5174825174825175	7.4652777777777777
11	232	7.2241379310344831	8.1888412017167376
12	376	7.9361702127659575	8.9124668435013259
13	609	8.6518883415435148	9.6360655737704910
14	986	9.3701825557809322	10.3596757852077008
15	1596	10.0902255639097742	11.0832811521602999
16	2583	10.8114595431668601	11.8068885448916401
17	4180	11.5334928229665064	12.5304950968667779
18	6764	12.2560615020697821	13.2541019955654100
19	10945	12.9789858382823216	13.9777087520555447
20	17710	13.7021456804065505	14.7013155665970299
21	28656	14.4254606365159130	15.4249223575391703
22	46367	15.1488774343822108	16.1485291580400272
23	75024	15.8723608445297497	16.8721359546817737
24	121392	16.5958877026492679	17.5957427528770189
25	196417	17.3194428180860100	18.3193495504485320
26	317810	18.0430162675812582	19.0429563482698825
27	514228	18.7666015852890169	19.7665631459913769
28	832039	19.4901945702064445	20.4901699437527043
29	1346268	20.2137924989675177	21.2137767414981688
30	2178308	20.9373936100863602	21.9373835392499394
31	3524577	21.6609967664204817	22.6609903369992089
32	5702886	22.3846012352342321	23.3845971347494697
33	9227464	23.1082065451569356	24.1082039324993360
34	14930351	23.8318123934259809	24.8318107302493623
35	24157816	24.5554185858522978	25.5554175279993210
36	39088168	25.2790249980505592	26.2790243257493081
37	63245985	26.0026315504454537	27.0026311234992846
38	102334154	26.7262381921875267	27.7262379212492647
39	165580140	27.4498448908184294	28.4498447189992412
40	267914295	28.1734516256402081	29.1734515167492212
41	433494436	28.8970583834667707	29.8970583144992013
42	701408732	29.6206651559051295	30.6206651122491813
43	1134903169	30.3442719376176129	31.3442719099991578
44	1836311902	31.0678787252123350	32.0678787077491378
45	2971215072	31.7914855165355057	32.7914855054991179
46	4807526975	32.5150923102204743	33.5150923032490979

47	7778742048	33.2386991054006486	34.2386991009990780
48	12586269024	33.9623059015268680	34.9623058987490509
49	20365011073	34.6859126982513502	35.6859126964990310
50	32951280098	35.4095194953539618	36.4095194942490110
51	53316291172	36.1331262926954579	37.1331262919989911
52	86267571271	36.8567330901878023	37.8567330897489711
53	139583862444	37.5803398877753452	38.5803398874989512
54	225851433716	38.3039466854229502	39.3039466852489241
55	365435296161	39.0275534831084414	40.0275534829989041
56	591286729878	39.7511602808178068	40.7511602807488842
57	956722026040	40.4747670785422144	41.4747670784988642
58	1548008755919	41.1983738762761007	42.1983738762488443
59	2504730781960	41.9219806740159555	42.9219806739988243
60	4052739537880	42.6455874717595691	43.6455874717488044
61	6557470319841	43.3691942695055488	44.3691942694987773
62	10610209857722	44.0928010672530064	45.0928010672487574
63	17167680177564	44.8164078650014090	45.8164078649987374
64	27777890035287	45.5400146627503943	46.5400146627487175
65	44945570212852	46.2636214604997491	47.2636214604986975
66	72723460248140	46.9872282582493384	47.9872282582486775
67	117669030460993	47.7108350559990697	48.7108350559986576
68	190392490709134	48.4344418537488863	49.4344418537486376
69	308061521170128	49.1580486514987740	50.1580486514986106
70	498454011879263	49.8816554492486901	50.8816554492485906
71	806515533049392	50.6052622469986346	51.6052622469985707
72	1304969544928656	51.3288690447485862	52.3288690447485507
73	2111485077978049	52.0524758424985592	53.0524758424985308
74	3416454622906706	52.7760826402485250	53.7760826402485037
75	5527939700884756	53.4996894379984909	54.4996894379984838
76	8944394323791463	54.2232962357484709	55.2232962357484638
77	14472334024676220	54.9469030334984438	55.9469030334984438
78	23416728348467684	55.6705098312484239	56.6705098312484239
79	37889062373143905	56.3941166289984039	57.3941166289984039
80	61305790721611590	57.1177234267483840	58.1177234267483769
81	99194853094755496	57.8413302244983640	58.8413302244983569

Perfect full binary tree statistics

Size	depth	avgd
1	1	1.00000000000000
3	2	1.66666666666667
7	3	2.42857142857143
15	4	3.26666666666667
31	5	4.16129032258065
63	6	5.09523809523809
127	7	6.05511811023622
255	8	7.03137254901961
511	9	8.01761252446184
1023	10	9.00977517106549
2047	11	10.00537371763556
4095	12	11.00293040293040
8191	13	12.00158710780125
16383	14	13.00085454434475
32767	15	14.00045777764214
65535	16	15.00024414435035
131071	17	16.00012970069657
262143	18	17.00006866481272
524287	19	18.00003623969315
1048575	20	19.00001907350452
2097151	21	20.00001001358510
4194303	22	21.00000524520999
8388607	23	22.00000274181399
16777215	24	23.00000143051156
33554431	25	24.00000074505808
67108863	26	25.00000038743020
134217727	27	26.00000020116568
268435455	28	27.00000010430813
536870911	29	28.00000005401671
1073741823	30	29.00000002793968
2147483647	31	30.00000001443550
4294967295	32	31.00000000745058
8589934591	33	32.00000000384171
17179869183	34	33.00000000197906
34359738367	35	34.00000000101863
68719476735	36	35.00000000052387
137438953471	37	36.00000000026921
274877906943	38	37.00000000013824
549755813887	39	38.00000000007094
1099511627775	40	39.00000000003638
2199023255551	41	40.00000000001864
4398046511103	42	41.00000000000955
8796093022207	43	42.00000000000489
17592186044415	44	43.00000000000250
35184372088831	45	44.00000000000128
70368744177663	46	45.00000000000065

140737488355327	47	46.00000000000033
281474976710655	48	47.00000000000017
562949953421311	49	48.00000000000009
1125899906842623	50	49.00000000000004
2251799813685247	51	50.00000000000002
4503599627370495	52	51.00000000000001
9007199254740991	53	52.00000000000001
18014398509481983	54	53.00000000000000
36028797018963967	55	54.00000000000000
72057594037927935	56	55.00000000000000
144115188075855871	57	56.00000000000000

```
Testing a height balanced integer set loaded with 2^n - 1 random integers, timed,
then two deletes, with smaller trees dumped for validation:

Load 1
Elapsed: 2e-06
root 0, level 1, btree level 1, avg depth 1, avg null depth 1
ptree:
0
Del 2 time: 0

Load 3
Elapsed: 0
root 0, level 2, btree level 2, avg depth 1.66667, avg null depth 2
ptree:
0       -4
        3
Del 2 time: 0

Load 7
Elapsed: 1e-06
root 3, level 3, btree level 3, avg depth 2.42857, avg null depth 3
ptree:
3       -9      -12
                -5
        10      9
                11
Del 2 time: 1e-06

Load 15
Elapsed: 4e-06
root -3, level 5, btree level 4, avg depth 3.23077, avg null depth 3.92857
ptree:
-3      -7      -21     -28
                        -20
                -4
        7       3
                        5
                18      10
                        25
                                28
Del 2 time: 1e-06

Load 31
Elapsed: 8e-06
root -15, level 6, btree level 5, avg depth 4.10714, avg null depth 4.93103
ptree:
-15     -42     -50     -57     -61
                                -51
                        -43     -47
                -27     -35     -39     -41
                                -34
                        -18
                                -17
        17      5       0       -11
                                4
                        16      10
                37      32      31
                                35
                        60      59
```

```
Del 2 time: 1e-06

Load 63
Elapsed: 1.6e-05
root -2, level 7, btree level 6, avg depth 5.08772, avg null depth 5.98276
ptree:
-2      -58     -97     -108    -121    -124
                                        -112
                                -102
                                        -99
                        -71     -86     -92
                                        -73
                                -59     -66
                -31     -41     -52
                                -35
                        -15     -21     -27
                                -10
                                        -4
        55      20      8       4       0
                                        7
                                9
                                        16
                        26      25      24
                                38      27
                                        42      41
                93      73      60      57
                                        70
                                87      82
                                                86
                                        90
                        113     98      97      96
                                        107     99
                                                108
                                114
                                        119
Del 2 time: 1e-06

Load 127
Elapsed: 5e-05
root -58, level 8, btree level 7, avg depth 6.0354, avg null depth 6.97368
ptree:
-58     -149    -197    -235    -236    -252
                                -204    -205    -213
                                        -202
                        -176    -182    -188    -189
                                                -185
                                        -180
                                -166    -172
                                                -167
                                        -157    -163
                                                -154
                -112    -128    -138    -144    -145
                                        -130
                                -124    -127
                                                -125
                                        -116    -121
                                                -113
                        -80     -91     -104    -109
                                                        -107
```

```
                                                -94     -103
                                        -89
                                                -85
                                -70     -77     -78
                                                -73
                                        -67
        82      12      -13     -45     -54     -55
                                                -47
                                        -39     -42
                                                        -40
                                                -28     -34
                                                        -19
                                -3      -4      -7      -9
                                                0
                                        8       7
                                                10
                        51      21      17      14
                                                19
                                        37      30
                                                39
                                                        40
                                -67     61      57
                                        77      76
                                                80
                164     121     95      88
                                        109     108
                                                118
                                -149    136     128
                                                        131
                                                138
                                        153
                                                154
                        208     170     169     168
                                        183     182
                                                203     186
                                -249    224     214
                                                227
                                                        245
                                        251
                                                252
Del 2 time: 0

Load 255
Elapsed: 9.7e-05
root 147, level 9, btree level 8, avg depth 7.01818, avg null depth 7.9819
ptree:
147     -253    -382    -436    -465    -493    -501    -507
                                                -496
                                        -470    -473
                                -452    -458    -464
                                        -449    -450
                                                -448
                                                        -439
                                -428    -431    -434
                                                -430
                                        -400    -419
                                                -393    -398
                                                        -388
                        -323    -354    -366    -372    -377
```

						-369		
					-356	-364		
						-355		
				-343	-346			
						-345		
					-324	-326		
			-289	-301	-307	-313	-315	
						-304		
					-297	-300		
						-294		
				-266	-279	-288		
					-261	-265		
						-258		
	-86	-150	-200	-230	-246	-250		
							-248	
						-242		
					-219	-228		
							-225	
						-204	-215	
							-202	
				-172	-183	-199		
							-185	
						-181		
					-156	-164	-165	
							-157	
						-154		
			-118	-133	-135	-138		
						-134		
					-128	-129		
						-124		
				-107	-111	-113		
						-109		
					-104			
						-87		
		24	-52	-56	-70	-79	-83	
							-72	
						-61	-63	
							-58	
					-55			
						-54		
				-5	-18	-30	-36	
						-6		
					12	3	1	
							4	
						17		
							23	
			92	54	39	35	33	
						42		
					85	74	55	
							77	
						87		
				117	110	105	103	
							106	
						116	111	
					128	126	118	
							127	
						139	138	
							145	
347	247	195	167	159	150	148		

```
                                                  153
                                        164       163
                                                  165
                              177       171
                                                  173
                                        179       178
                                                  193
                    224       208       202       198
                                        213       212
                                                  221       215
                              240       235       228
                                        243       241
          307       279       267       258       250
                                        270
                              295       291       290
                                                  293
                                        303
                    324       318       308
                                        323
                              334       331       328
                                        341
     429       392       363       351       349
                                        362
                              369       366
                                        380
                    417       412       404       396
                                        414
                              420       418
                                        424
          471       443       434       432
                              463       448
                    483       482       480
                              495       488
                                        508
Del 2 time: 5e-06

Load 511
Elapsed: 0.000203
root 239, level 11, btree level 9, avg depth 8.06154, avg null depth 9.04167
Del 2 time: 0

Load 1023
Elapsed: 0.000408
root 509, level 12, btree level 10, avg depth 9.08231, avg null depth 10.0711
Del 2 time: 1e-06

Load 2047
Elapsed: 0.000897
root 640, level 13, btree level 11, avg depth 10.1102, avg null depth 11.1042
Del 2 time: 1e-06

Load 4095
Elapsed: 0.001851
root 421, level 14, btree level 12, avg depth 11.0772, avg null depth 12.0739
Del 2 time: 5e-06

Load 8191
Elapsed: 0.004135
root 2640, level 16, btree level 13, avg depth 12.0949, avg null depth 13.0931
```

```
Del 2 time: 1e-06

Load 16383
Elapsed: 0.009065
root 1685, level 16, btree level 14, avg depth 13.0789, avg null depth 14.0779
Del 2 time: 4e-06

Load 32767
Elapsed: 0.034264
root 4897, level 18, btree level 15, avg depth 14.1334, avg null depth 15.1329
Del 2 time: 3e-06

Load 65535
Elapsed: 0.064853
root -11533, level 19, btree level 16, avg depth 15.1504, avg null depth 16.1501
Del 2 time: 4e-06

Load 131071
Elapsed: 0.158064
root -62394, level 20, btree level 17, avg depth 16.1909, avg null depth 17.1907
Del 2 time: 4e-06

Load 262143
Elapsed: 0.375258
root -15007, level 21, btree level 18, avg depth 17.1539, avg null depth 18.1539
Del 2 time: 8e-06

Load 524287
Elapsed: 0.896229
root -185508, level 23, btree level 19, avg depth 18.2003, avg null depth 19.2003
Del 2 time: 9e-06

Load 1048575
Elapsed: 2.15594
root -465574, level 24, btree level 20, avg depth 19.2169, avg null depth 20.2169
Del 2 time: 1.3e-05

Load 2097151
Elapsed: 5.21541
root -800002, level 25, btree level 21, avg depth 20.2426, avg null depth 21.2426
Del 2 time: 1e-05

Load 4194303
Elapsed: 11.5176
root 288181, level 26, btree level 22, avg depth 21.2271, avg null depth 22.227
Del 2 time: 7e-06

Load 8388607
Elapsed: 26.6298
root 479211, level 27, btree level 23, avg depth 22.2598, avg null depth 23.2598
Del 2 time: 7e-06

Load 16777215
Elapsed: 64.9325
root -559343, level 29, btree level 24, avg depth 23.2524, avg null depth 24.2524
Del 2 time: 1.4e-05
```

C code of AVL worst case analysis:

```
#include <stdio.h>

int main(){
      long l; // levels
      long a = 1; // size a
      long b = 2; // size b
      long c ; //    size c
      long nda = 2 ; // total null depth sum a
      long ndb = 5 ; // total null depth sum b
      long ndc ;     // totol null depth sum next
      long tda = 1 ; // total value depth a
      long tdb = 3 ; // total value depth b
      long tdc ;     // total value depth next

      printf( "AVL Worst Case Trees\nLevels\tWSize\tWAvg=D\tWAvg!=D\n" );

      for ( l = 1 ; l < 128 ; l++ ){
            if ( nda < 0 ) break ; // so high it went negative
            printf( "%ld\t%ld\t%.16lf\t%.16lf\n",
                  l,
                  a,
                  1.0 * tda / a,
                  1.0 * nda / (a + 1) );
            ndc = nda + a + ndb + b + 2 ; // null depth sum
            tdc = tda + a + tdb + b + 1 ; // next total value depth
            c = a + b + 1 ; // next tree has prior 2 trees as children
            nda = ndb ;
            ndb = ndc ;
            tda = tdb ;
            tdb = tdc ;
            a = b;
            b = c;
      }

      return 0;
}
```

C code of binary search tree average find comparison count.

```
#include <stdio.h>

int main(){
      long s = 1, l = 1, td = 1, d = 1;

      printf( "Size\t\t\tdepth\tavgd\n" );

      while (td > 0 && s > 0) {
            printf( "%18ld\t%5ld\t%02.14lf\n", s, d, td * 1.0 / s );
            s = s + s + 1 ;
            d++ ;
            l <<= 1 ;
            td += l * d ;
      }

      return 0;
}
```

C++ code of a height balanced set exercised with sequential or random integers:

```
#include <iostream>
#include <stdio.h>      /* printf, scanf, puts, NULL */
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <sys/time.h>       /* gettimeofday */

using namespace std;

template <class T>
struct avl_node {
      struct avl_node<T> * left, * right ;
      char level ;
      T value ;
};

template <class T>
class avl_tree {
      private:
      public:
            avl_node<T> * root = nullptr ;
            size_t size = 0 ;

      bool add( T &t ){ // returns true for update
            return add( &root, t );
      }

      bool add( avl_node<T> ** rootp, T &t ){
            avl_node<T> * root = *rootp ;

            if ( root == nullptr ){
                  *rootp = root = new avl_node<T>();
                  root->left = root->right = nullptr ;
                  root->level = 1 ;
                  root->value = t ;
                  size++ ;
                  return false ;    // not an update
            }

            // auto ans = t <=> root->value ;
            bool ret ;

            if ( t < root->value ){
                  ret = add( &root->left, t );
            } else if ( t > root->value ){
                  ret = add( &root->right, t );
            } else { // equal t
                  root->value = t ; // update
                  ret = true ;
            }

            if ( !ret ) avl( rootp ); // ensure balance after insert
            return ret ;
      }

      // achieve balance and set new level
      void avl( avl_node<T> ** rootp ){
            avl_node<T> * root = *rootp ;
```

```
// cout << ( root ? root->value : 0 ) << " avl\n" ;
            if ( !root ) return ;

            char avlr = root->right ? root->right->level : 0 ;
            char avll = root->left ? root->left->level : 0 ;
            short balance = avlr - avll ;

            if ( balance < -1 ){
                  // is a right rotate needed before the left rotate?
                  avl_node<T> * root2 = root->left ;
                  avlr = root2->right ? root2->right->level : 0 ;
                  avll = root2->left ? root2->left->level : 0 ;

                  if ( avll < avlr ){ // inside heavier
                        rrotate( &root->left );
                        avl( &root->left );
                  }

                  lrotate( rootp );
                  avl( rootp );
            } else if ( balance > 1 ){
                  // is a left rotate needed before the right rotate?
                  avl_node<T> * root2 = root->right ;
                  avlr = root2->right ? root2->right->level : 0 ;
                  avll = root2->left ? root2->left->level : 0 ;

                  if ( avlr < avll ){ // inside heavier
                        lrotate( &root->right );
                        avl( &root->right );
                  }

                  rrotate( rootp );
                  avl( rootp );
            } else { // balanced, just calc level
                  root->level = 1 + (avlr > avll ? avlr : avll) ;
            }
      }

      // rotate right to left
      void rrotate( avl_node<T> ** rootp ){
// cout << "rr\n" ;
            avl_node<T> * t1, * t2 ;
            t1 = *rootp ;                 // save old root
            t2 = *rootp = t1->right ;     // hang right as root
            t1->right = t2->left ;        // move rl to lr
            t2->left = t1 ;               // hang old root on left
            avl( &t2->left );       // balance new node
      }

      // rotate left to right
      void lrotate( avl_node<T> ** rootp ){
// cout << "lr\n" ;
            avl_node<T> * t1, * t2 ;
            t1 = *rootp ;                 // save old root
            t2 = *rootp = t1->left ;      // hang left as root
            t1->left = t2->right ;        // move lr to rl
            t2->right = t1 ;        // hang old root on right
            avl( &t2->right );            // balance new node
      }
```

```
      void clear(){
            clear( root );
            root = nullptr ;
            size = 0 ;
      }

      void clear( avl_node<T> *root ){
            if ( root ){
                  clear( root->left );
                  clear( root->right );
                  delete root ;
            }
      }

      bool del( T &t ){
// cout << "del\n";
            return del( &root, t );
      }

      bool del( avl_node<T> ** rootp, T &t ){
            avl_node<T> * root = *rootp ;

            if ( ! root ) return false ;

            bool ret ;

            if ( root->value > t ){
                  ret = del( &root->left, t );
            } else if ( root->value < t ){
                  ret = del( &root->right, t );
            } else { // equal
                  ret = true ;
                  size-- ;

                  if ( root->right == nullptr ){
                        *rootp = root->left ;
                        delete root ;
                  } else if ( root->left == nullptr ){
                        *rootp = root->right ;
                        delete root ;
                  } else if ( root->right->level >= root->left->level ){
                        root->value = delmin( &root->right );
                  } else root->value = delmax( &root->left );
            }

            if ( ret ) avl( rootp );
            return ret ;
      }

      T delmin( avl_node<T> ** rootp ){
// cout << "delmin\n";
            avl_node<T> * root = *rootp ;

            if ( root->left ){
                  T ret = delmin( &root->left );
                  avl( rootp );
                  return ret ;
            }
```

```
            *rootp = root->right ; // snap me out of the tree
            avl( rootp ); // adjust tree levels
            T ret = root->value ;
            delete root ;
            return ret ;
      }

      T delmax( avl_node<T> ** rootp ){
// cout << "delmax\n";
            avl_node<T> * root = *rootp ;

            if ( root->right ){
                  T ret = delmin( &root->right );
                  avl( rootp );
                  return ret ;
            }

            *rootp = root->left ; // snap me out of the tree
            avl( rootp ); // adjust tree levels
            T ret = root->value ;
            delete root ;
            return ret ;
      }

      bool contains( T t ){
            return contains( root, t );
      }

      bool contains( avl_node<T> * root, T t ){
            if ( !root) return false ;

            if ( root->value < t ){
                  return contains( root->right, t );
            } else if ( root->value > t ){
                        return contains( root->left, t );
                } else  return true; // equal
      }

      void inorder( avl_node<T> * root ){
            if ( root ){
                  inorder( root->left );
                  cout << root->value << ' ' << (int)root->level << endl ;
                  inorder( root->right );
            }
      }

      void ptree(){
            if ( root ){
                  ptree( root, false, 0 ) ;
            }
      }

      void ptree( avl_node<T> * root, bool right, int rl ){
            for ( int i = rl ; right && i ; i-- ){
                  cout << '\t' ;
            }

            cout << root->value ;
```

```
            if ( root->left ){
                  cout << '\t' ;
                  ptree( root->left, false, rl + 1 );
            } else cout << endl ;

            if ( root->right ){
                  ptree( root->right, true, rl + 1 );
            }
      }

      float avgdepth(){ // root is 1, children are 2, etc.
            if ( !size ) return 0.0 ;
            return avgdepth( root, 1 ) * 1.0 / size ;
      }

      long avgdepth( avl_node<T> * root, int depth ){
            if ( !root ) return 0 ;

            long l = avgdepth( root->right, depth + 1 )
                 + avgdepth( root->left, depth + 1 )
                 + depth ;
            // cout << l << " avg\n";
            return l;
      }

      float avgndepth(){ // root is 0, children are 1, 2, etc.
            return avgndepth( root, 0 ) * 1.0 / ( 1 + size ) ;
      }

      long avgndepth( avl_node<T> * root, int depth ){
            if ( !root ) return depth ;

            long l = avgndepth( root->right, depth + 1 )
                 + avgndepth( root->left, depth + 1 );
            // cout << l << " avgn\n";
            return l;
      }
};

int randint(){
      static bool first = true ;
      static int bitct = 0;
      static long bits = 0L;
      int ret ;

      if ( first ){
            srand (time(NULL));
            first = false ;
      }

      while ( bitct < 32 ){
            bitct += 15 ;
            bits = (bits << 15) ^ rand();
      }

      ret = bits & 0xffffffff ;
      bits >>= 32 ;
      bitct -= 32 ;
```

```
      return ret ;
}

double timer( string desc ){
      static struct timeval start ;
      struct timeval end ;
      double ret ;

      if ( ! desc.size() ){
            gettimeofday( &start, NULL );
            return 0.0;
      }

      gettimeofday( &end, NULL );
      ret = (end.tv_usec - start.tv_usec) * .000001
          + (end.tv_sec - start.tv_sec) ;
      cout << desc << ": " << ret << endl;
      return ret ;
}

int main( int argc, char **argv ){
      avl_tree<int> tree;
      float bf;
      int i, rs, ls, bl = 0;

      for ( int j = 1 ; j > 0 ; j = j + j + 1 ){
      tree.clear();
      int * rands = new int[j];
      if ( argc > 1 ) for ( i = 0 ; i < j ; i++ ){
            rands[i] = randint() % (j << 1) ;
      }
      cout << "\nLoad " << j << endl ;
      timer( "" );
      for ( i = 1 ; i < j+1 ; i++ ){
            if ( argc == 1 ) tree.add( i );
            else {
                  tree.add( rands[i-1] );
            }
      }
      timer( "Elapsed" );
      delete[] rands;
      bl++ ; // perfect binary levels

      // bf is low size / high size
      // ls = tree.root->value - 1 ;
      // rs = j - tree.root->value ;
      // bf = rs > ls ? 1.0 * ls / rs : 1.0 * rs / ls ;

      // cout << "after " << j << ", root " << tree.root->value << ", level " <<
(int)tree.root->level << ", bf = " << bf << ' ' << ", lf = " << lf << endl ;
      cout << "root " << tree.root->value << ", level " << (int)tree.root->level <<
", btree level " << bl << ", avg depth " << tree.avgdepth() << ", avg null depth "
<< tree.avgndepth() << endl ;
      if ( j < 500 ){ cout << "ptree:\n"; tree.ptree(); }
      // if ( j < 100 ){ cout << "inorder:\n"; tree.inorder( tree.root ); }
      timer( "" );
      int k = j / 3 ;
      // cout << k << " del\n" ;
      tree.del( k );
```

```
        k = (j * 2) / 3 ;
        // cout << k << " del\n" ;
        tree.del( k );
        timer( "Del 2 time" );
        }

        return 0;
}
```