# Minimum number of items in an AVL balanced binary tree

```
height      min size    growth
------    ----------   ------
     1    1.000000e+00      inf
     2    2.000000e+00 2.000000
     3    4.000000e+00 2.000000
     4    7.000000e+00 1.750000
     5    1.200000e+01 1.714286
     6    2.000000e+01 1.666667
     7    3.300000e+01 1.650000
     8    5.400000e+01 1.636364
     9    8.800000e+01 1.629630
    10    1.430000e+02 1.625000
    11    2.320000e+02 1.622378
    12    3.760000e+02 1.620690
    13    6.090000e+02 1.619681
    14    9.860000e+02 1.619048
    15    1.596000e+03 1.618661
    16    2.583000e+03 1.618421
    17    4.180000e+03 1.618273
    18    6.764000e+03 1.618182
    19    1.094500e+04 1.618125
    20    1.771000e+04 1.618090
    21    2.865600e+04 1.618069
    22    4.636700e+04 1.618056
    23    7.502400e+04 1.618047
    24    1.213920e+05 1.618042
    25    1.964170e+05 1.618039
    26    3.178100e+05 1.618037
    27    5.142280e+05 1.618036
    28    8.320390e+05 1.618035
    29    1.346268e+06 1.618035
    30    2.178308e+06 1.618034
    31    3.524577e+06 1.618034
    32    5.702886e+06 1.618034
    33    9.227464e+06 1.618034
    79    3.788906e+16 1.618034
    80    6.130579e+16 1.618034
    81    9.919485e+16 1.618034
    82    1.605006e+17 1.618034
    83    2.596955e+17 1.618034
    84    4.201961e+17 1.618034
    85    6.798916e+17 1.618034
    86    1.100088e+18 1.618034
    87    1.779979e+18 1.618034
    88    2.880067e+18 1.618034
    89    4.660047e+18 1.618034
    90    7.540114e+18 1.618034
    91    1.220016e+19 1.618034
    92    1.974027e+19 1.618034
    93    3.194043e+19 1.618034
    94    5.168071e+19 1.618034
    95    8.362114e+19 1.618034
    96    1.353019e+20 1.618034
    97    2.189230e+20 1.618034
    98    3.542248e+20 1.618034
    99    5.731478e+20 1.618034
```

$$1.618034 = \frac{\sqrt{5}+1}{2} = \phi$$

actual Number of items in tree of Height H

$$N = 1.17 \times \phi^H$$

$$\log_2 N = \log_2 1.17 + \log_2 \phi^H$$

Ⓐ $\log_2 N - 0.22 = \log_2 \phi^H$

$= H \log_2 \phi$

Ⓑ $= 0.696H$

$$H = \frac{\log_2 N}{0.696} + \frac{0.22}{0.696}$$

$$\underline{H = 1.44 \log_2 N} + 0.317$$

= worst possible case for (-1,0,+1) balanced tree

= 44% more than best possible case for full binary tree

(but the operations are <u>slightly</u> more complex)

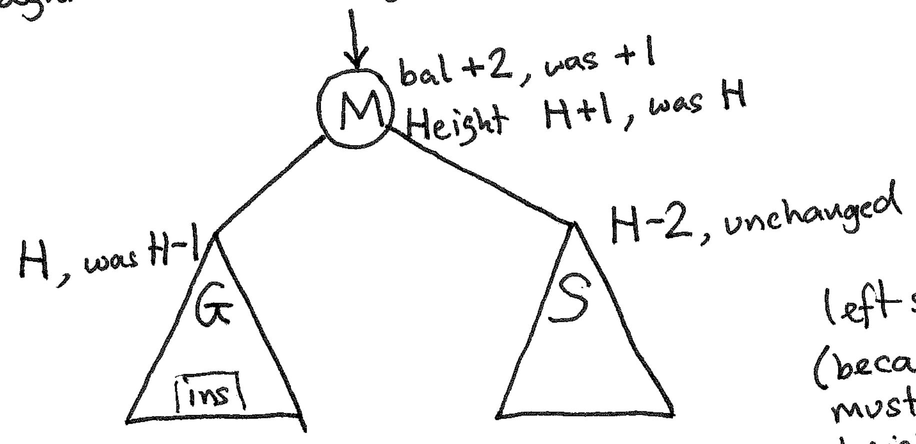
Ⓐ $\log_2 1.17 = \frac{\log 1.17}{\log 2} = \frac{0.068}{0.301} = 0.22$

Ⓑ $\log_2 \phi = \frac{\log \phi}{\log 2} = \frac{0.209}{0.301} = 0.696$

Somewhere in the tree, just after an insertion, looking at the first unbalanced node on the return path from the insertion to the root.

Assume insertion was to the left (to the right is just ~~the~~ mirror image of this, so nothing is lost)

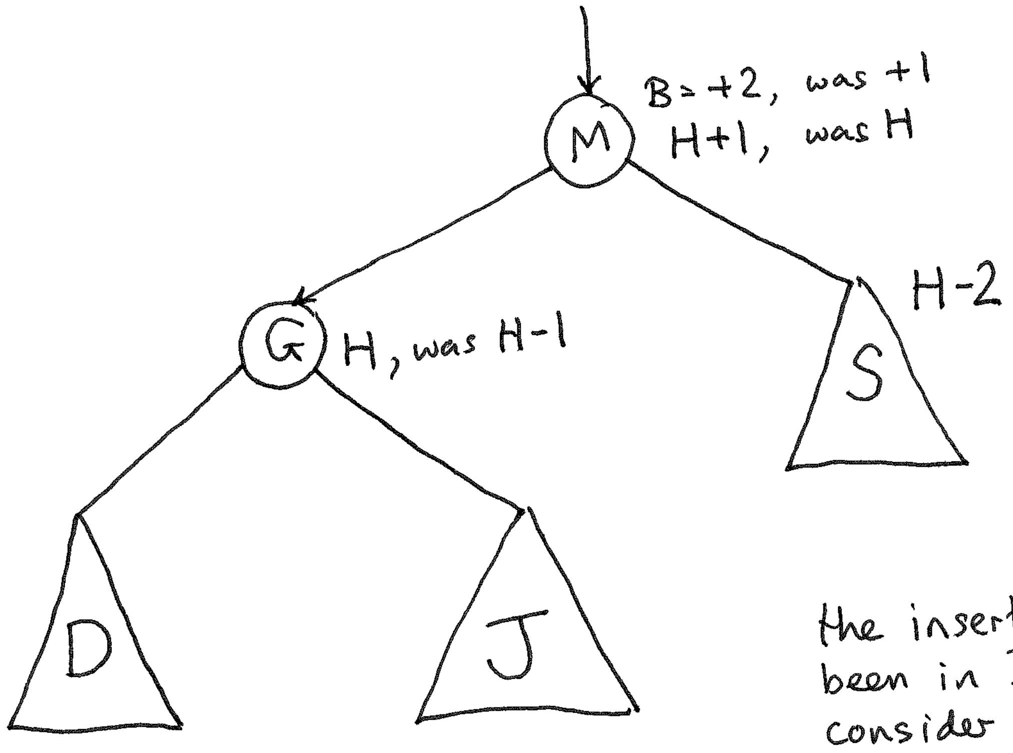
Tree was balanced before, and only one thing was inserted, so only possibility is balance was +1 before insertion, but +2 now, and height has increased by one.



left subtree G is biggest (because bal is +2) so must be one less than height of M.

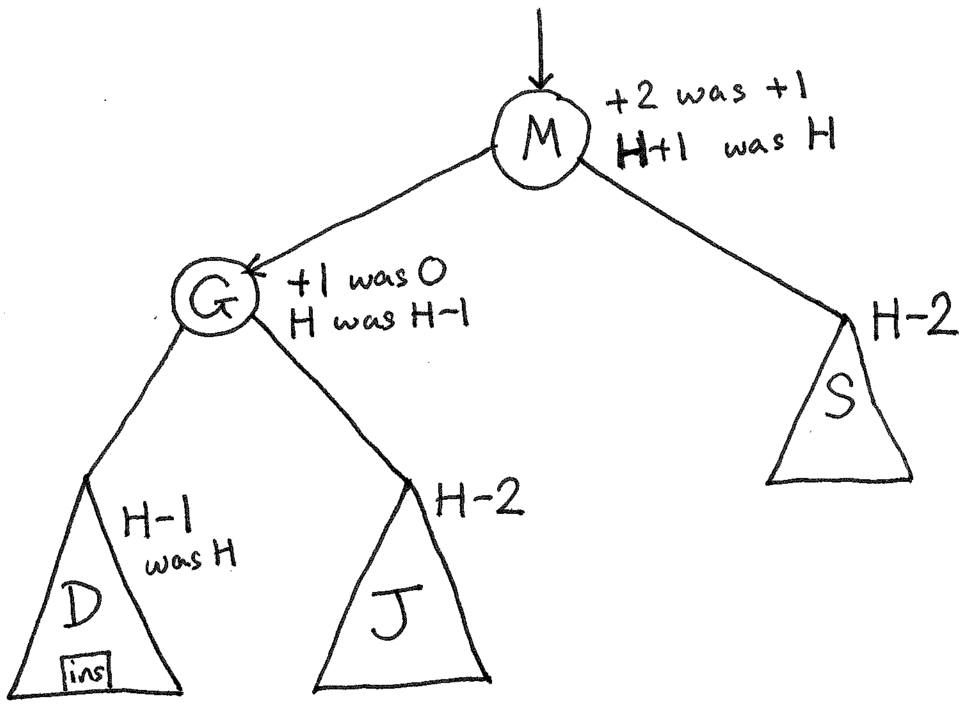
right subtree S must be two less than G because bal is +2.

not enough information yet, but we know G has at least one node so can expand

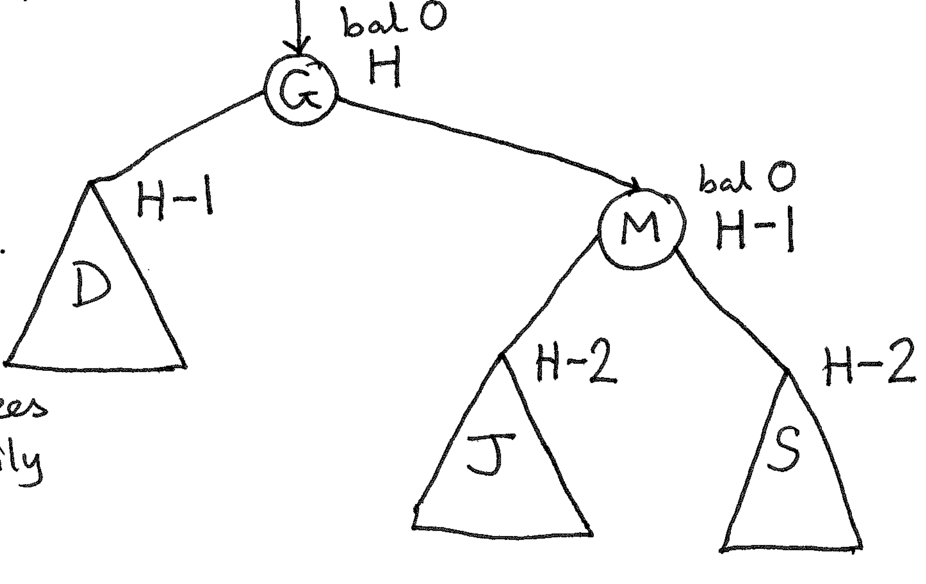


the insertion could have been in D or in J, consider each case in turn....

case 1 – insertion was in D.
insertion increased G's height
∴ must also have increased D's height
and J's height was not bigger than D's height
G is still balanced (because M is lowest unbalanced)
∴ G's balance is now +1 but was 0.
∴ D's height is H−1
and J's height is H−2



reassembling with G at top and maintaining correct left to right ordering



heights of subtrees D, J, S will not change, they are just being moved.

so heights and balances of G and M are easily recalculated.

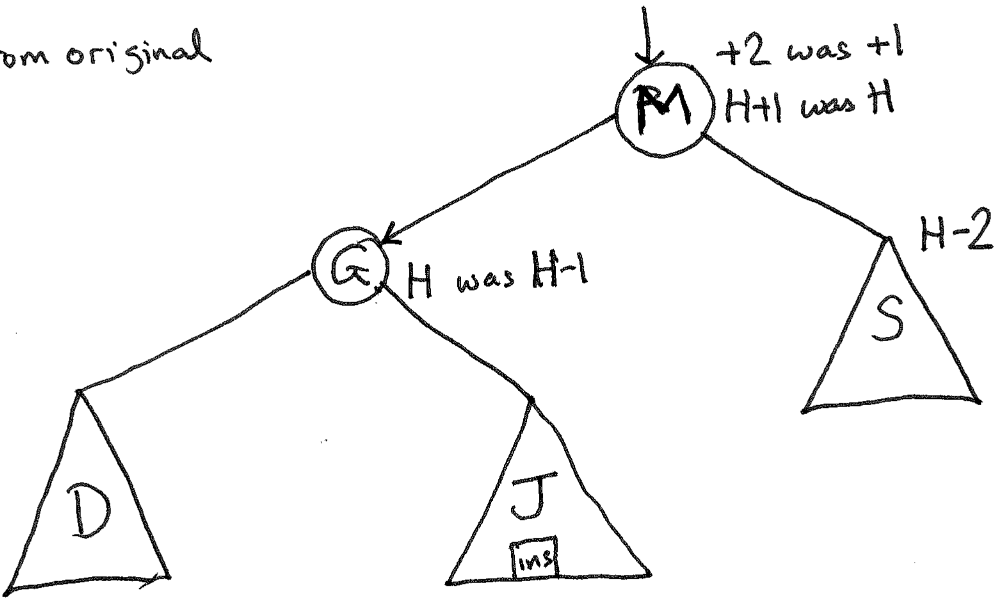
The balance of this portion of the tree is better than it was before the insertion.

The height of this portion of the tree has been restored to its pre-insertion value of H.
it has not grown, ∴ nothing above it can possibly have its balance changed, so no further checks are needed.

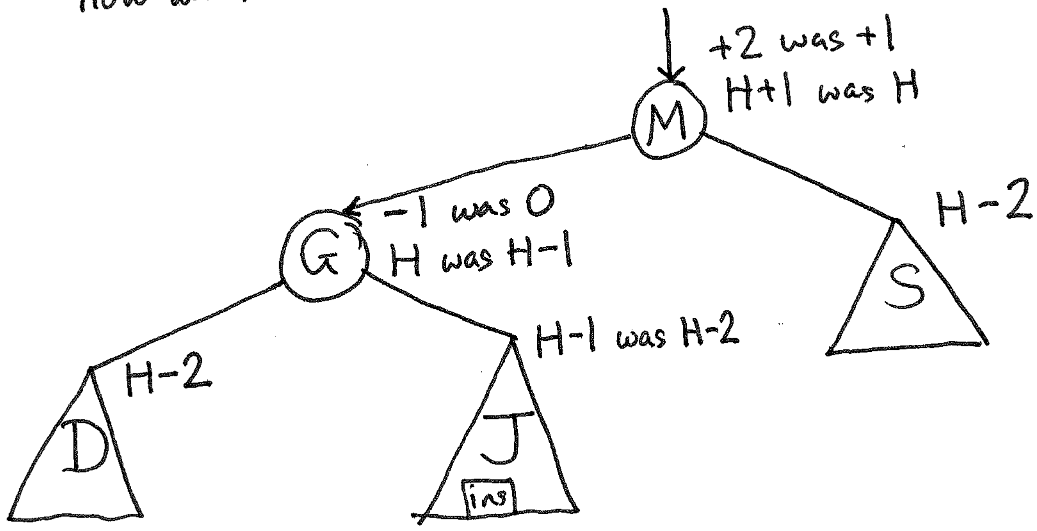
---

Case 2 - insertion was in J.

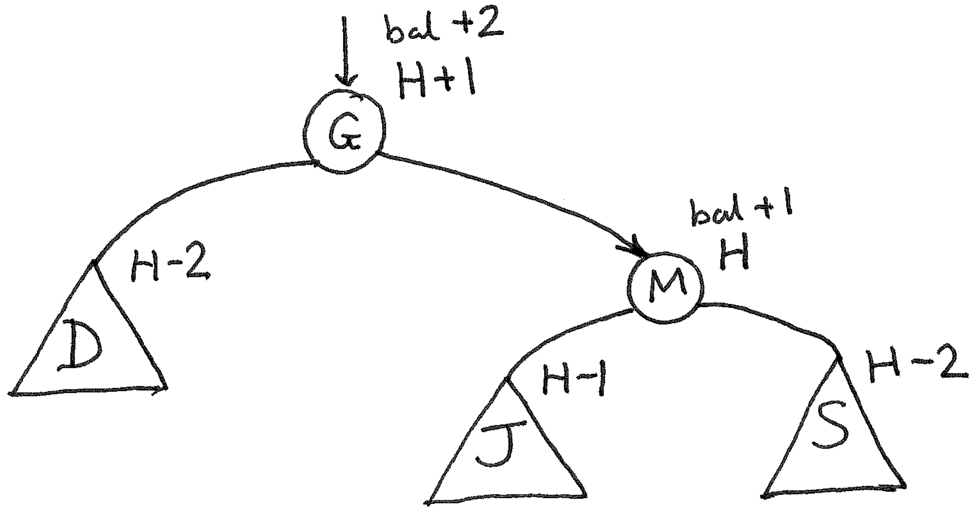
from original



same reasoning as before
J must have grown, and for that to have affected G,
D can not have been bigger and is now smaller than J.
G is still balanced ∴ size difference is only 1.
now we know all the heights and balances.

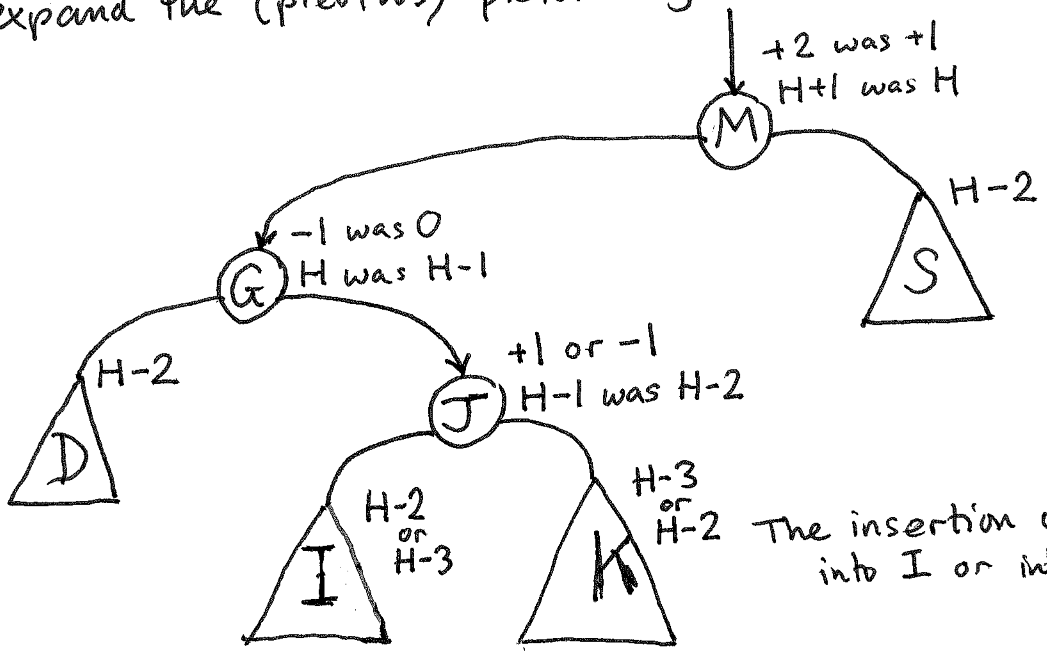


trying out the solution that worked for case 1:



it didn't work. G is still unbalanced.

But we know J has at least one node, so we can expand the (previous) picture again



The insertion was either into I or into K

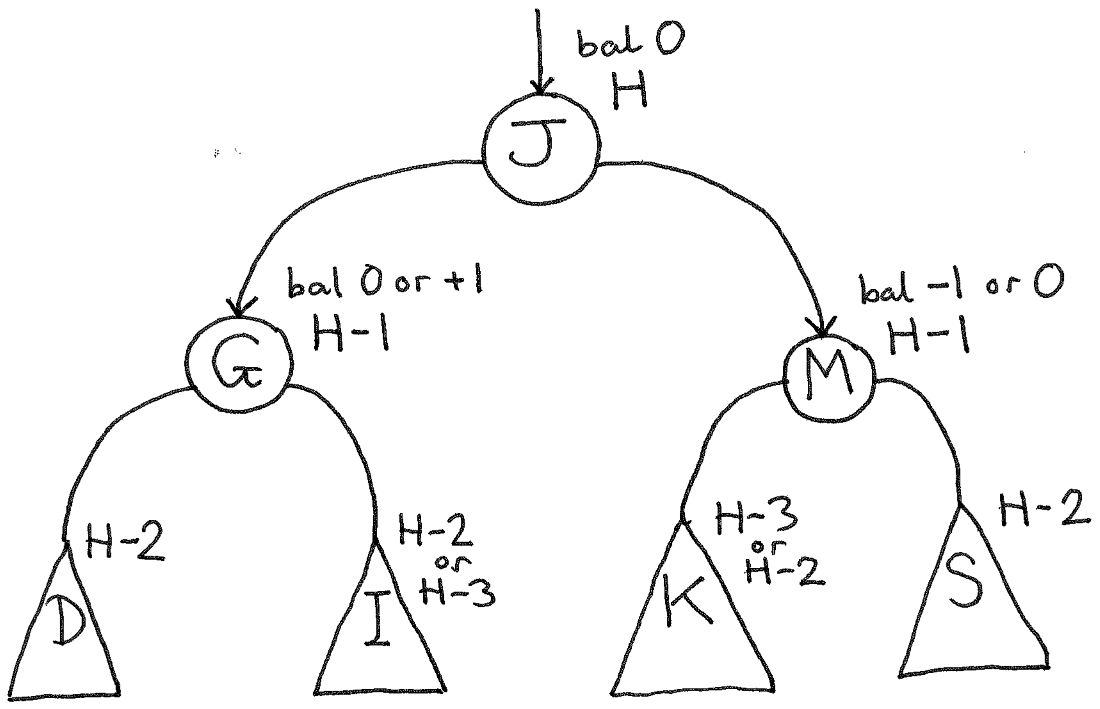
wherever the insertion was (I or K) it grew so its new height must be H-2, one less than J's height.

the other one (K or I) can not have been bigger to start with because it would have been responsible for J's height and J would not have grown

but it can not have been smaller either, otherwise J would now be unbalanced

∴ one of I, K has height H-2 and the other H-3.
It turns out we don't care which.

Of the three nodes we know of (G, J, M) J is in the middle, so let's try it as the new root. Moving D, I, K, and S will not change their sizes, so the height and balance of G, J, M can easily be worked out.



The whole thing is balanced and back to its pre-insertion height of H, so once again, nothing higher in the tree can be affected, no further checks are needed.

In summary

At the lowest unbalanced node in the tree
    if its balance is -2, do the mirror image of the following.
its balance is +2
look at the node to its left
    if its balance is +1, do the transformation ("rotation") we worked out for case 1.
    if its balance is -1, do the transformation we just worked out for case 2.
    anything else means you've done something wrong.

A random binary tree, no attempt at balancing

```
struct node
{ int val;
  node * left, * right;
  node(int v) { val = v; left = NULL; right = NULL; } };

void insert(node * & t, int v)
{ if (t == NULL)
    t = new node(v);
  else if (v < t->val)
    insert(t->left, v);
  else
    insert(t->right, v); }

int total_depth(node * t, int depth_here = 1)
{ if (t == NULL)
    return 0;

  int left_depth  = total_depth(t->left,  depth_here + 1);
  int right_depth = total_depth(t->right, depth_here + 1);

  return depth_here + left_depth + right_depth; }

int main()
{ srandomdev();
  node * t = NULL;
  int n = 1 << 20;

  for (int i=0; i<n; i+=1)
    insert(t, random());
  int tot = total_depth(t);

  printf("total dept %d for %d items\n", tot, n);
  printf("average %f\n", tot/(double)n); }
```

adding up the depths of every single node.

```
$ a.out
total dept 27675409 for 1048576 items
average 26.393327
$ a.out
total dept 27679123 for 1048576 items
average 26.396869
$ a.out
total dept 27225243 for 1048576 items
average 25.964015
$ a.out
total dept 27156735 for 1048576 items
average 25.898681
$ a.out
total dept 26395019 for 1048576 items
average 25.172252
$ a.out
total dept 27114779 for 1048576 items
average 25.858668
$ a.out
total dept 26941356 for 1048576 items
average 25.693279
$ a.out
total dept 26602916 for 1048576 items
average 25.370518
$ a.out
total dept 27773280 for 1048576 items
average 26.486664
```

this is $2^{20} = N$

so $\log_2 N = 20$

the average is $1{\cdot}3 \times \log_2 N$

Search times for binary tree with N items

Ordinary, no balancing done

| | |
|---|---|
| best possible | $\log_2 N$ |
| average random | $1.3 \log_2 N$ |
| worst possible | $N$ |

AVL balanced

| | |
|---|---|
| best possible | $\log_2 N$ |
| worst possible | $1.44 \log_2 N$ |