- –
  –
  – Define $N(h)$ to be the least number of nodes in a height-h foo tree. One subtree must have height at least $h-1$, the other at least $h-3$. Therefore we have the following recursive definition for $N(h)$.

$$N(h) = \begin{cases} 1 + N(h-1) + N(h-3) & h \geq 3 \\ 3 & h = 2 \\ 2 & h = 1 \\ 1 & h = 0 \end{cases}$$

Since $N(h-1) \geq N(h-3)$ and $1 > 0$ we have $N(h) > 2N(h-3)$. By successive application of this relation we get

$$N(h) > 2N(h-3) > 4N(h-6) > 8N(h-9) > \cdots > 2^i N(h-3i).$$

For $i = \lfloor h/3 \rfloor$ we have $0 \leq h - 3i \leq 2$ and $N(h-3i) \geq 1$. Thus we have $N(h) > 2^{\lfloor h/3 \rfloor}$. Now consider an arbitrary foo tree T with n nodes and let h be the height of T. Since T is a height-h foo tree it contains at least $N(h)$ nodes. Therefore we have $n \geq N(h) > 2^{\lfloor h/3 \rfloor} \Rightarrow \lfloor h/3 \rfloor < \log_2 n \Rightarrow h/3 - 1 < \log_2 n \Rightarrow h < 3\log_2 n + 3 \Rightarrow h \in O(\log n)$.

–

```
height(T)
T: the root of an AVL tree
1.   if T.left = nil then
2.       return T.balance
3.   else
4.       return 1 + height(T.left) + ⌈T.balance/2⌉
```

If $T.left = nil$ then $T.balance$ is either 0 or 1, the height of the tree. This shows that the correct answer is always returned at line 2. Now use induction on h, the height of the tree. For the base case $h = 0$, we have $T.left = nil$, and we have already noted that the correct answer will be returned at line 2. Assume, to apply the principle of strong mathematical induction, that the algorithm is correct for all trees of height up to $h-1$, some $h > 0$, and let T be a tree of height h. If $T.balance \leq 0$ then h is equal to 1 plus the height of $T.left$. If $T.balance = 1$ then h is equal to 2 plus the height of $T.left$. In both of these cases the correct answer is returned at line 4 since $\lceil -1/2 \rceil = 0$, $\lceil 0/2 \rceil = 0$ and $\lceil 1/2 \rceil = 1$.

The nonrecursive work done during each call to *height* is $O(1)$, and the number of recursive calls is bounded by h. Since $h = O(\log n)$ the overall running time is $O(\log n)$.

- –

  -
- -
  -
  -
  - $h = \lfloor \log_2(n+1) \rfloor - 1$
- -
  - B-tree of height 2 on 9 nodes, all nodes with 2 KVPs except root which only has one
  -

- The main idea of preprocessing is to initialize an array C such that $C[i]$ is equal to the number of elements in A whose value is at most i.

```
preprocess(A, k)
A: array of size n,   k: positive integer
1.  C ← array of size k, filled with zeros
2.  for i ← 0 to n − 1 do
3.      increment C[A[i]]
4.  for i ← 1 to k − 1 do
5.      C[i] ← C[i] + C[i − 1]
```

Line 1 takes time $O(n)$. Line 3 takes constant time and is iterated n times. The total running time of lines 2-3 is thus $O(n)$. Line 5 takes constant time and is iterated $O(k)$ times. The total running time of lines 4-5 is $O(k)$. Thus, the total running time of preprocessing algorithm is $O(n+k)$. After this preprocessing step, $C[a]$ contains the number of elements of A that are at most a for each integer a, $0 \leq a < k$.

```
query(a, b)
1.  if a = 0 then
2.      return C[b]
3.  else
4.      return C[b] − C[a − 1]
```

The query algorithm takes constant time.