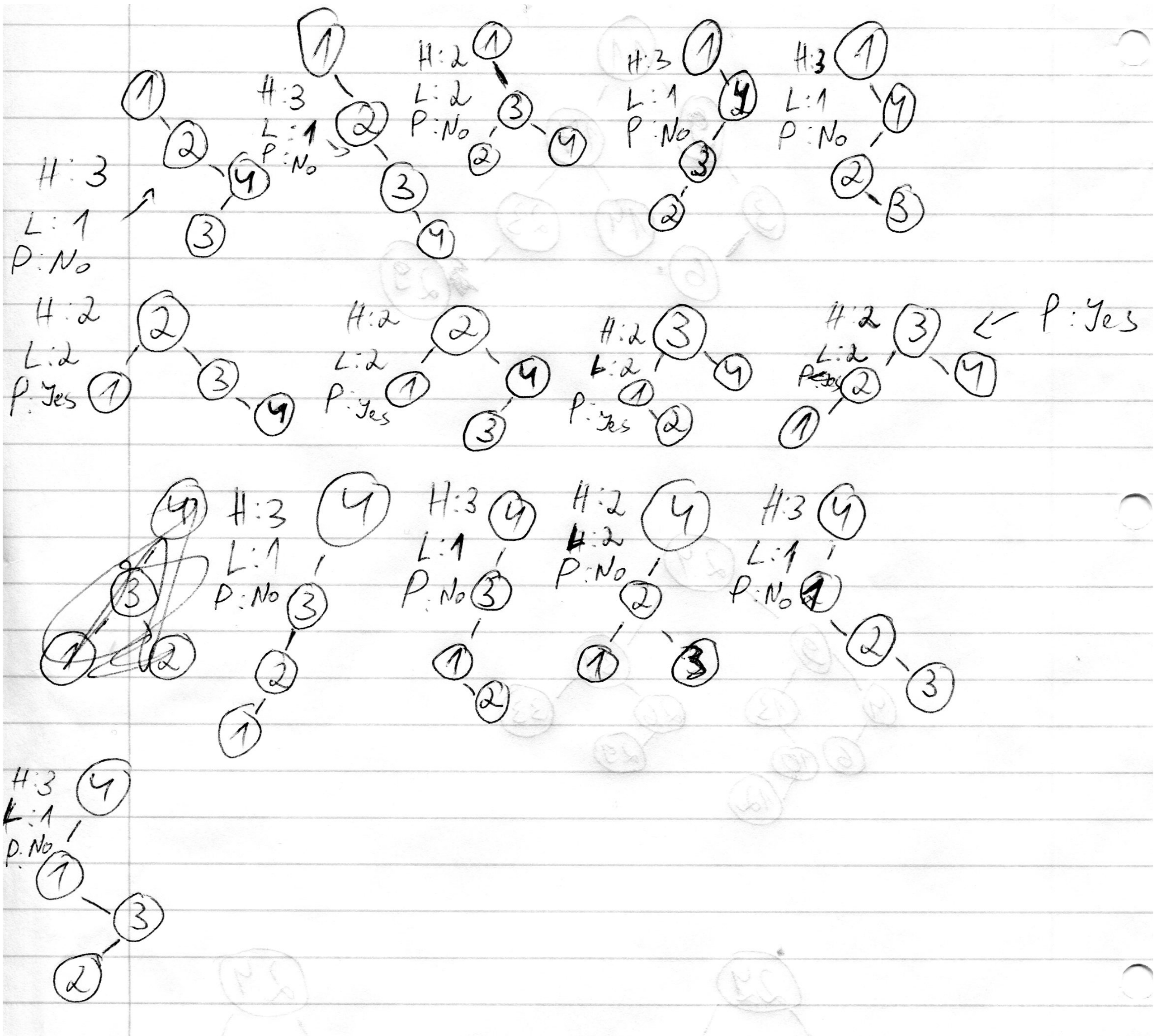
**Student ID:** 1422087

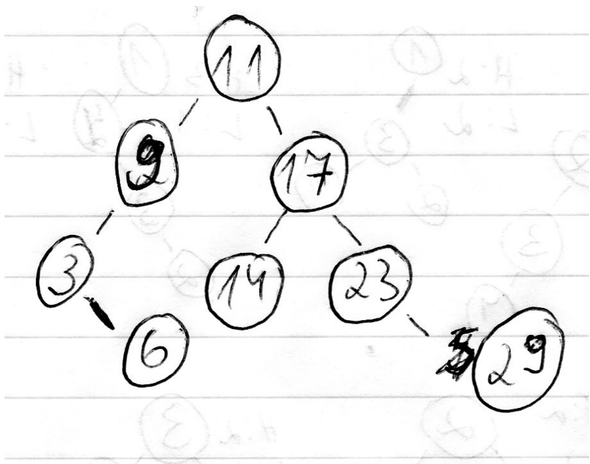
**1.** There are 4*3*2*1 = 24 different orderings of the numbers {1, 2, 3, 4} available. They can be represented by the following binary trees:



On the right of every tree are written its height, number of leaves and if it's perfectly balanced (H – height, L – number of leaves and P – is it perfectly balanced).

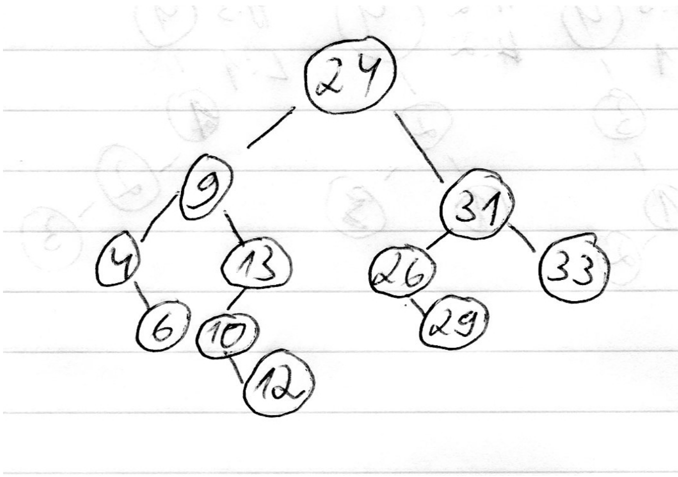
There are 14 unique trees.

**2.** Here's the tree that is formed if number 6 is the root:

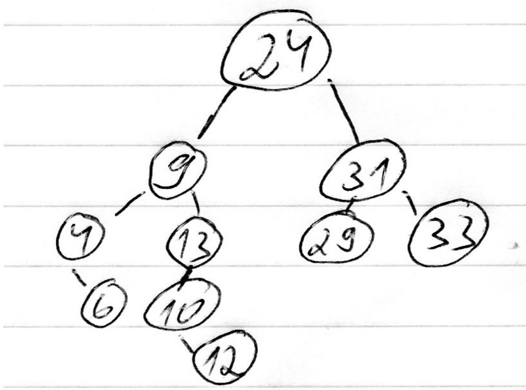



This is a valid search binary tree because on the left side we always have values smaller than the root and on the right – values always bigger than the node.

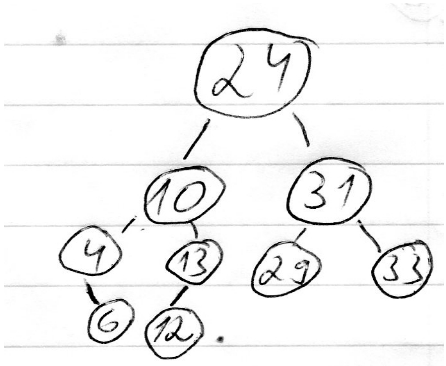
**3.** The tree which is a result of inserting the items [24, 9, 13, 4, 31, 26, 6, 10, 29, 33, 12] in that order into an empty tree is:



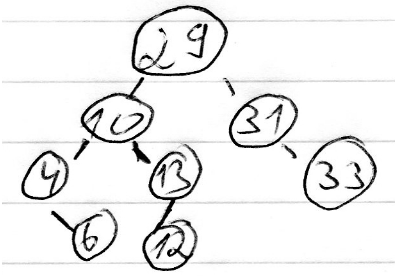

When we remove item 26 we get:



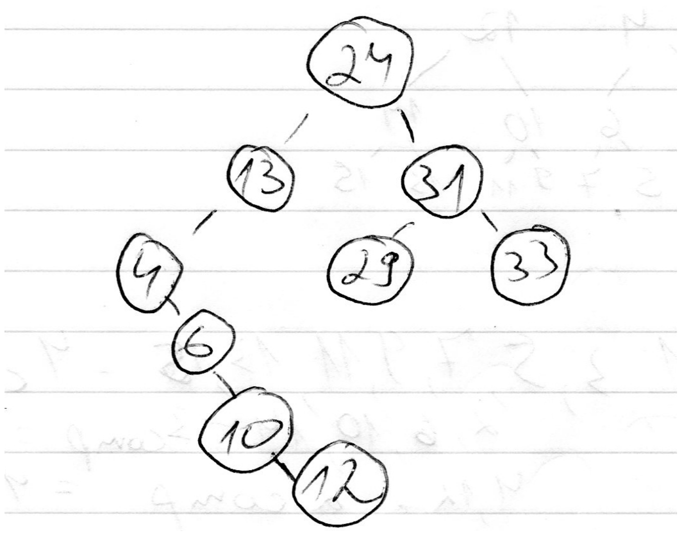
And when we delete 9 from the new tree:



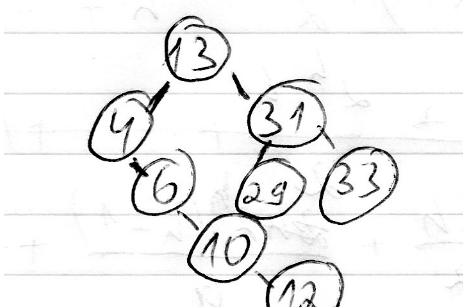

And after we remove 24:



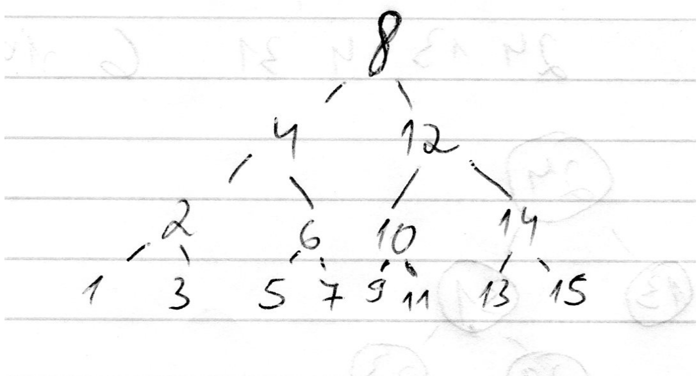

All new trees are valid binary search trees. Not always though the same trees would form if we remove some of numbers – if we only remove 26 the tree would be the same, but if we remove 9 too then the “tree from scratch” would be:




After we remove 24 the tree from building a new binary search would again be different:



**4.** The tree looks like this:




The leaves (1, 3, 5, 7, 9, 11, 13 and 15) need the most comparisons – 4. Then the level above them requires one comparison less, therefore 2, 6, 10 and 14 require 3 comparisons each. Using the same logic we can see that 4 and 12 need 2 comparisons and 8 needs only one. We can say that the number of comparisons required is equal to l+1 where l is the level on which the node is situated.

Using these numbers, if we want to search for all the numbers the total number of comparisons C required for this would be 8*4 + 4*3 + 2*2 + 1 = 49. Therefore the average number of comparisons required for searching a number should be 49 / 15 ≈ 3.267 so around 3-4 comparisons.

**5.** As we saw in Question 4 the total number of comparisons C required is the sum of the comparisons needed to search all the numbers on a given level. So if we get the tree from Question 4 as an example we can have these sums:

- For Level 3 (the lowest one): 8 elements * 4 comparisons needed for each = 32 comparisons
- For Level 2: 4 elements * 3 comparisons = 12 comparisons
- For Level 1: 2 elements * 2 comparisons = 4 comparisons
- For Level 0: 1 element * 1 comparison = 1 comparison

We can see that the elements on each level are 2 at the power of l ("l" being the level we're computing). We can also see that the comparisons needed are always l+1 for each element. Therefore we can derive the equation for a level:

- C(l) = (2^l)*(l+1)

The overall C is the sum of all sums of the levels, so the whole equation for C should be:

- C(h) = (2^h)*(h+1) + (2^(h-1))*h + … + (2^0)*1

We've seen that in Question 4 we computed A(h) by dividing the total number of comparisons (C(h)) by the total number of elements. In the case of Question 4 the exact numbers were 49 / 15.

Using this we can make a default equation for computing A(h). If we have a full binary search tree then the number of elements on each level should be 2^l where l is the level we're computing for. Therefore the whole number of elements i.e. the size S(h) of a tree should be:

- S(h) = 2^h + 2^(h-1) + … + 2^0

Therefore we can write A(h) as:

- A(h) = C(h) / S(h) = ((2^h)*(h+1) + (2^(h-1))*h + … (2^0)*1) / (2^h + 2^(h-1) + … + 2^0)

A clearer version:

$$C(h) = 2^h(h+1) + 2^{h-1} \cdot h + \dots + 2^0 \cdot 1$$

$$A(h) = \frac{2^h(h+1) + 2^{h-1} \cdot h + \dots + 2^0 \cdot 1}{2^h + 2^{h-1} + \dots + 2^0}$$

We can see that for large trees the average amount of comparisons needed shouldn't increase quickly – both C(h) and S(h) increase exponentially, but the C(h) is divided by S(h) and therefore the increase isn't massive.