In the insertion sort, we iterate the list excluding the first element. The iterated part of the list is sorted among itself. As we iterate, we find the true place on the sorted list to the current element and we put it there.

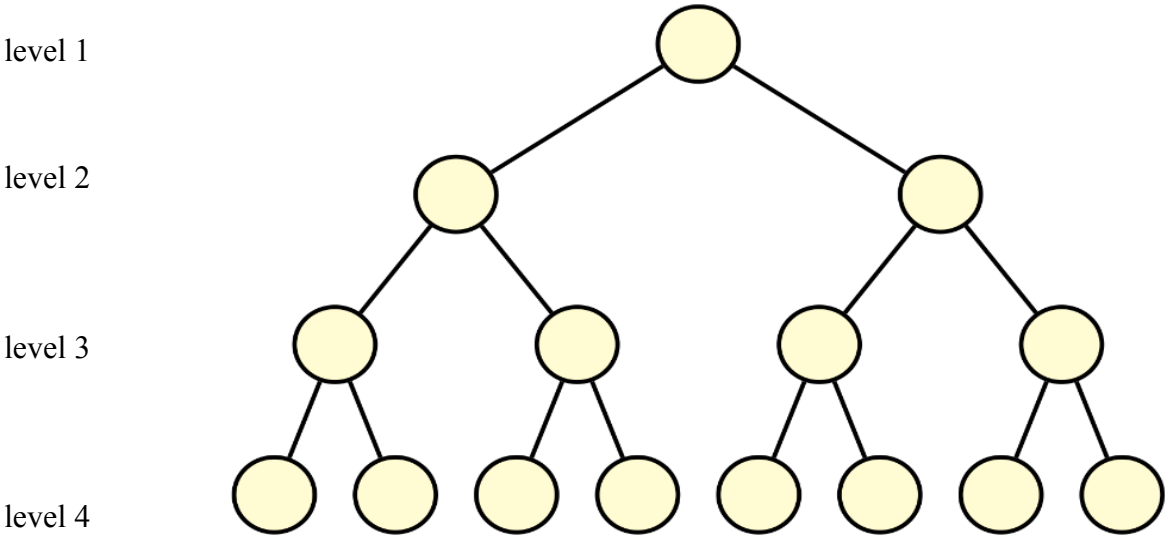
In order to reduce the number of comparisons in the insertion sort. We can find the place to put the current element by running a binary search on the sorted part of the list. This algorithm is called *Binary Insertion Sort*.

```
function BinaryInsertionSort (L[0:n-1])
    for i←1 to n-1 do
        current←L[i]
        left ← 0
        right ← i-1
        while (right ≥ left) do
            position ← ⌊(left+right) / 2⌋
            if (L[position] < current) then
                left ← position + 1
            else
                right ← position - 1
            endif
        endwhile
        L[left+1:i] ← L[left:i-1] //in slicing both indices are included in the operation
        L[left] ← current
    endfor
end
```

Insertion sort makes $O(n^2)$ comparisons on the average case. Now, we have to find the number of comparisons on the average case $A(n)$ for binary insertion sort.
Let's call τ_i the number of comparisons at step i (i represents the index/iteration count on the for loop). So, we have to find the expected value of τ_i and take a summation of it.

$$
\begin{aligned}
A(n) &= E[\tau] \\
&= E[\tau_1] + E[\tau_2] + E[\tau_3] + \cdots + E[\tau_{n-1}] \\
&= \sum_{i=1}^{n-1} E[\tau_i]
\end{aligned}
$$

To make a simplification, let us consider a 14 item list and 1 item to be inserted. So, all the nodes on the below graph represent a slot that the new item can be inserted.



If we apply our loop to every possible case, we will see that 4 comparisons are made in every possibility. Further investigation with other numbers can also be made. However, since the complexity of the binary search algorithm is a well-known topic. I will assume this will be enough.

So, it will be convenient to say $E[\tau_i] = log_2(i + 1)$ since all the indexes are equally likely.
If we plug-in these equation we reach

$$\sum_{i=1}^{n-1} E[\tau_i] = \sum_{i=1}^{n-1} log_2(i + 1) = log_2(n!)$$

Now we have to find the complexity

$$(n/2) * log_2(n/2) \leq log_2(n/2) + \cdots log_2(n) \leq log_2(n!) = log_2(1) + \cdots log_2(n)$$
$$log_2(n!) = log_2(1) + \cdots log_2(n) \leq n * log_2(n)$$

Using these two inequalities we can conclude that our function makes $\theta(n * log(n))$ comparisons.

This method has some drawbacks, some of which are

- Although there are less comparisons, as we insert the item to its place we will have to move all the items greater than the item. As a result, we will have to iterate all the items greater than our current number just as in the insertion sort besides the binary search part.
- Best-case of insertion sort is $O(n)$ which is the sorted list case, but this method has no best or fast cases. All cases behave the same.
- This method is also harder to implement since it is a mixture of two algorithms.