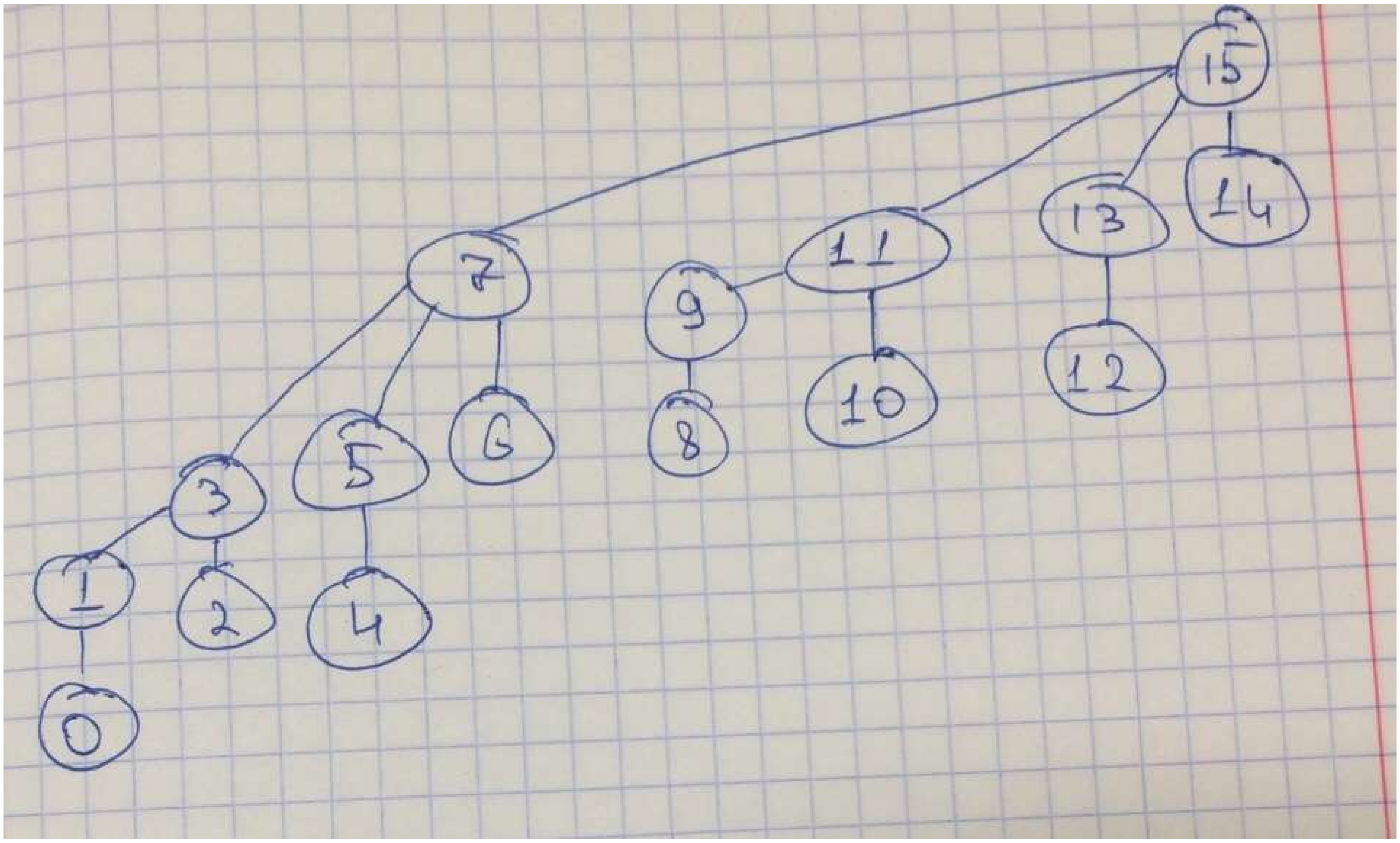
EX1: This is the binomial heap of degree 4. The values to nodes I gave by using postorder method (according to it (Left, Right, Root), we should fill left, then right node and root at the end).



This Binomial heap B4 contains 4 sub-heaps B0, B1, B2, B3. Array representation of B4 is as below, where the array index represents the number which I gave to the node before:

$$[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$$

We have 2^k nodes and as the degree equals to 4, the amount of nodes is 16.

The root has degree k, and as the children are numbered from left to right by k-1,k-2, k-3, ...,0,child i the root of subtree B.

After considering all of these I tried to find the mathematical operations to access first child, parent and next sibling. For the first child I got $i – 2^{(k-1)}$, where i is node and k is the degree of tree. If we check this for 7, we will get that $7-2^{3-1}$ = 3 which is first child of 7.

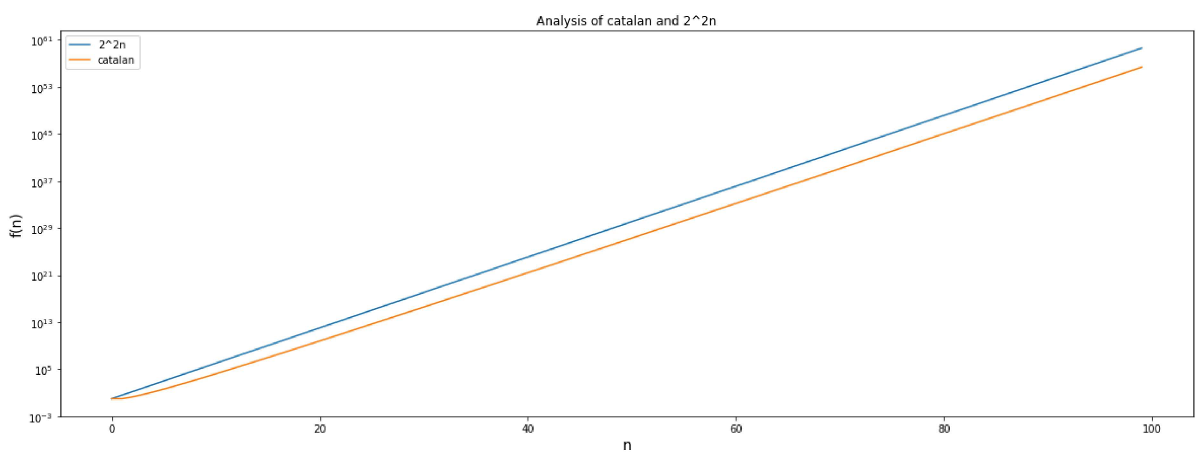
This the formula for parent: $i+2^k$. For example, for 8 as its degree is 0, I get $8+2^0 \ =9$ and it is its parent. For sibling, $i+2^{k-1}$ . . If we check this for 5 and 5 is in 1 degree tree, We will get that $5+2^{1-1}$ = 6 which is sibling of 5.

EX2: As we know, there are less than 2^{2n} distinct binary trees on n nodes. Catalan numbers describe that how many binary trees there are for n nodes. The equation of Catalan numbers is:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \qquad \text{for } n \geq 0.$$

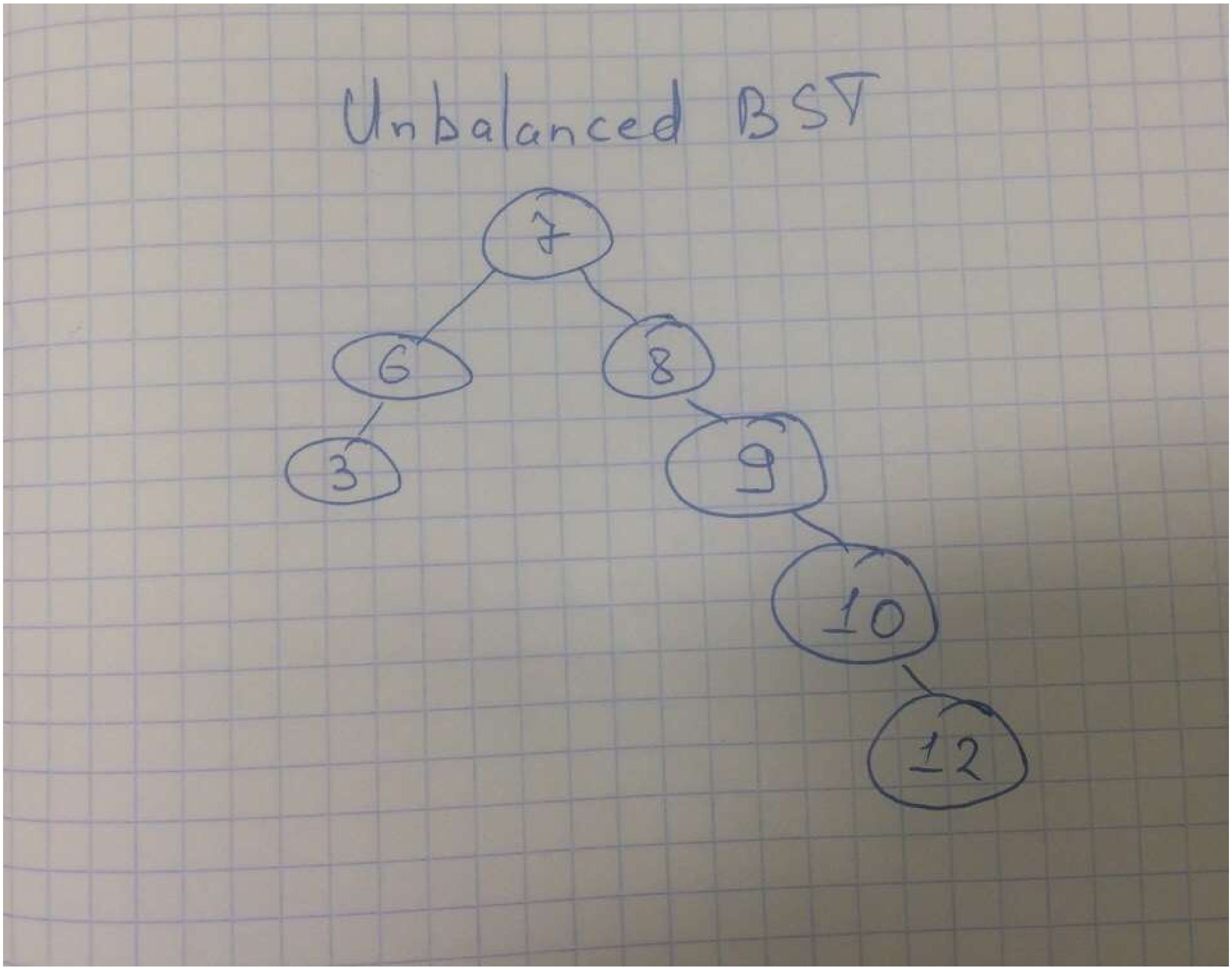
In the code below I plotted the 2^{2n} and the equation of catalan numbers in order to compare them. Also I put them in logarithmic scale to see the difference and check if they are in Theta() relationship.

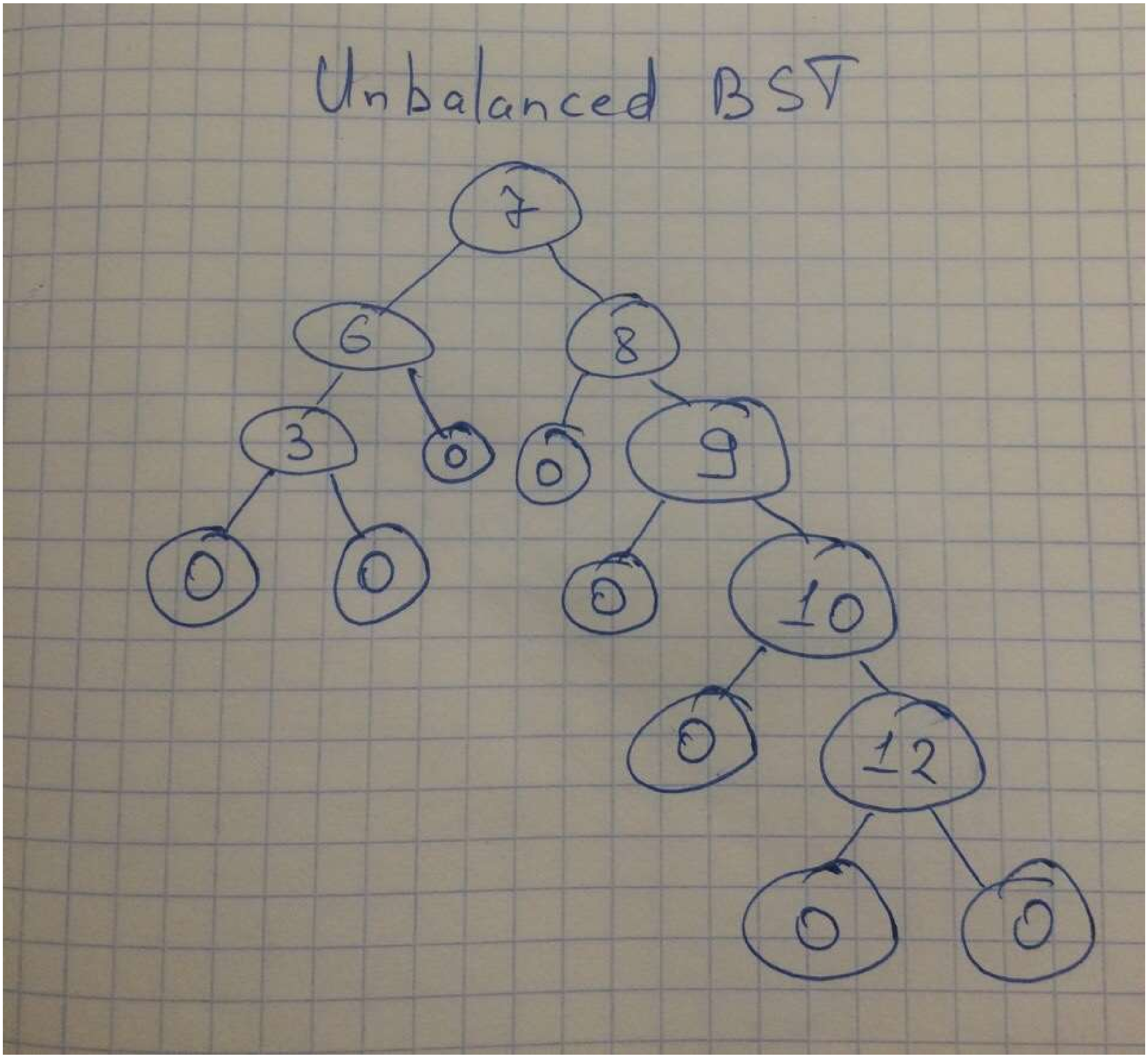
In [23]:
```
import random, sys, time, numpy, math
import matplotlib.pyplot as plott

measures_exponent = []
measures_catalan = []
def catnumber(n):
    ans = 1.0
    for k in range(2,n+1):
        ans = ans *(n+k)/k
    return ans

for i in range(100):
    measures_exponent.append(2**(2*i))
    measures_catalan.append(catnumber(i))

plott.figure(figsize=(20,7))
plott.title("Analysis of catalan and 2^2n")
plott.ylabel('f(n)', fontsize=14)
plott.xlabel('n', fontsize=14)

plott.plot(measures_exponent, label='2^2n')
plott.plot(measures_catalan, label='catalan')
plott.legend()
plott.yscale('log')
plott.show()
```



Although the difference is very small on the graph we should take into account that we are in logarithmic scale and even small differences make a sense. As we see, 2^{2n} is greater than Catalan numbers. They are not in Theta relationship. Catalan numbers = O(2^{2n})

EX3: This is the unbalanced tree:


Unbalanced BST
7
6
8
3
9
10
12


Unbalanced BST
7
6
8
3
0
0
9
0
0
0
10
0
12
0
0

This tree can be represented as an array with indexes:

[1,1,1,1,0,0,1,0,0,0,1,0,1,0,0]

An n node binary tree can be represented in 2n+1 bits. We have 7 nodes and 15 bits.

The formulas to find

1.left child(x) = [2x]

2.right child(x) = [2x+1]

3.parent(x) = [⌊x/2⌋]

The child of 9:

9 is the 5th node in the tree, so: left child: $2 * 5 = 10$. This node holds 0, so 9 does not have left child. right child: $2 * 5 + 1 = 11$. In 11th node we have 10, so 10 is the only child of 9.

Parent of 10:

10 is 11th node: ⌊11/2⌋ = 5. And in the 5th node we have 9.

EX4: I used level-order degree sequence. Our nodes: 7, 8, 6, 9, 10, 3, 12. Unary: 1 1 0 1 0 1 0 0 1 0 1 0 0.

So, the level-wise representation is:

- {1, 0,1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0}

Parenthesis representation:

- ((())(((()))))

The second representation is more natural and visible as it gives a depth-first flattening of the tree. It has a huge advantage over the other representation, as it can be parsed purely with a stack-based parser.

1.Operations for level-order:

9 is in 8th order. parent(8) = 3. So when k is 8, we get the number of 0's 3 and 3rd node in the tree is 8. It means that 8 is parent of 9.

children of k are stored after the k-th 0. 12 is 7th node in the tree and k = 7. We should find the children after 7th 0 in the bits representation but we don't have it.

2.Parenthesis.

parent – enclosing parenthesis:

To find parent(9) we should follow closing parenthesis which opened before to find enclosing paranthesis. The enclosing parenthesis belongs to 8. So, it means 8 is the parent of 9.

first child – next parenthesis (if 'open')

In parenthesis representation there is no next parenthesis after 12. So 12 has no children.

EX5: Subtree size is the number of '(' before the enclosing ')' corresponding to our position. We push and pop as always until we are back at a empty stack and count the number of open parenthesis.

Subtree size of '8': $((())\big((1(2(3)))\big)) = 3$

Lowest common ancestor is found by popping the number of open parenthesis before x until being a parent of y. Let's take 6 and 9 nodes to find lowest common ancestor for them. First, we should find the parent of 9 which is 8. And now we should find the enclosing parenthesis for 8 and 6 which is 7. It means that 7 is lowest common ancestor for 6 and 9. It is found in O(1) time.

EX6:

In [24]:
```
import math
a = int("0b" + "1" * 64, 2)
tmp = a

def tmpp(a):
    a = math.ceil(a/2)
    global tmp
    tmp = tmp + a
    if a > 1:
        tmpp(a)
    else:
        return tmp

tmpp(a)
print("The data structure would be larger", tmp, "bits with 1 million random v
alues in 64 bit")
```

```
The data structure would be larger 36893488147419103230 bits with 1 million r
andom values in 64 bit
```