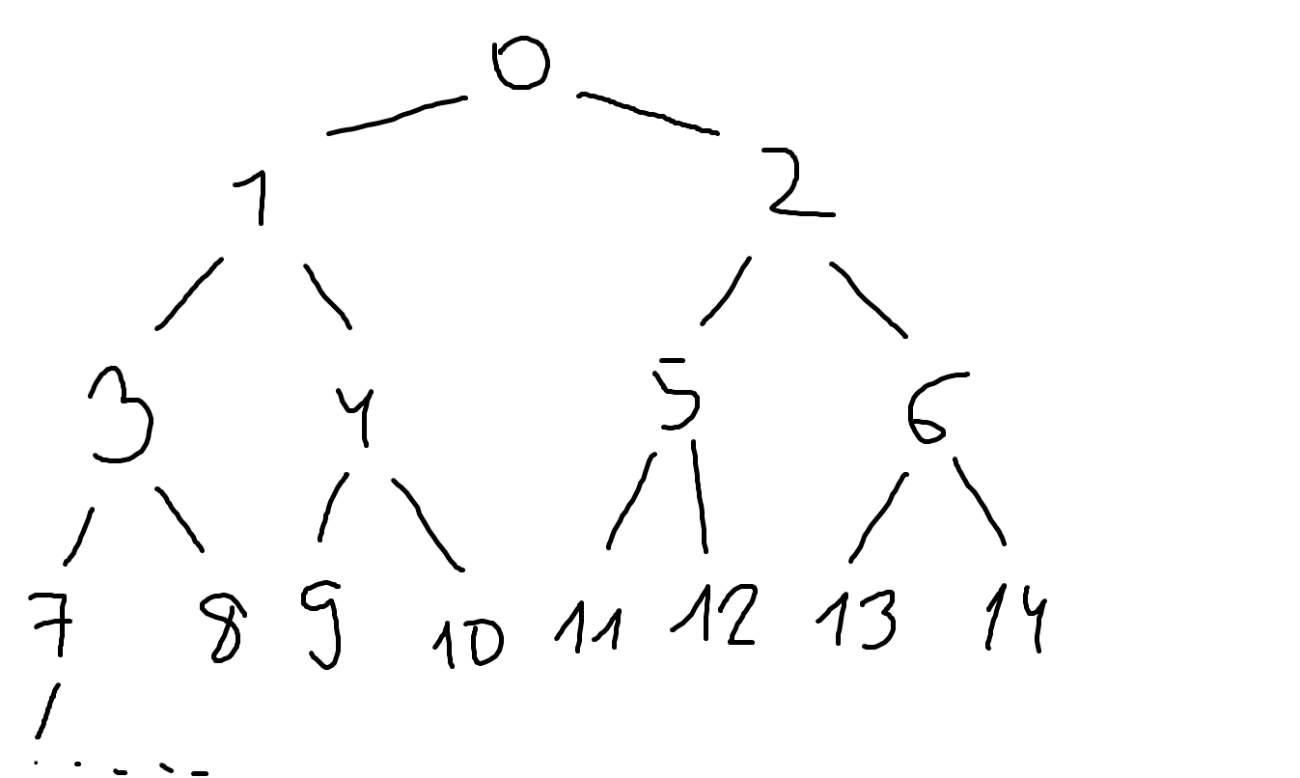
1. При запуске f(0, 7, 13) сделаются 14 рекурсивных вызовов:

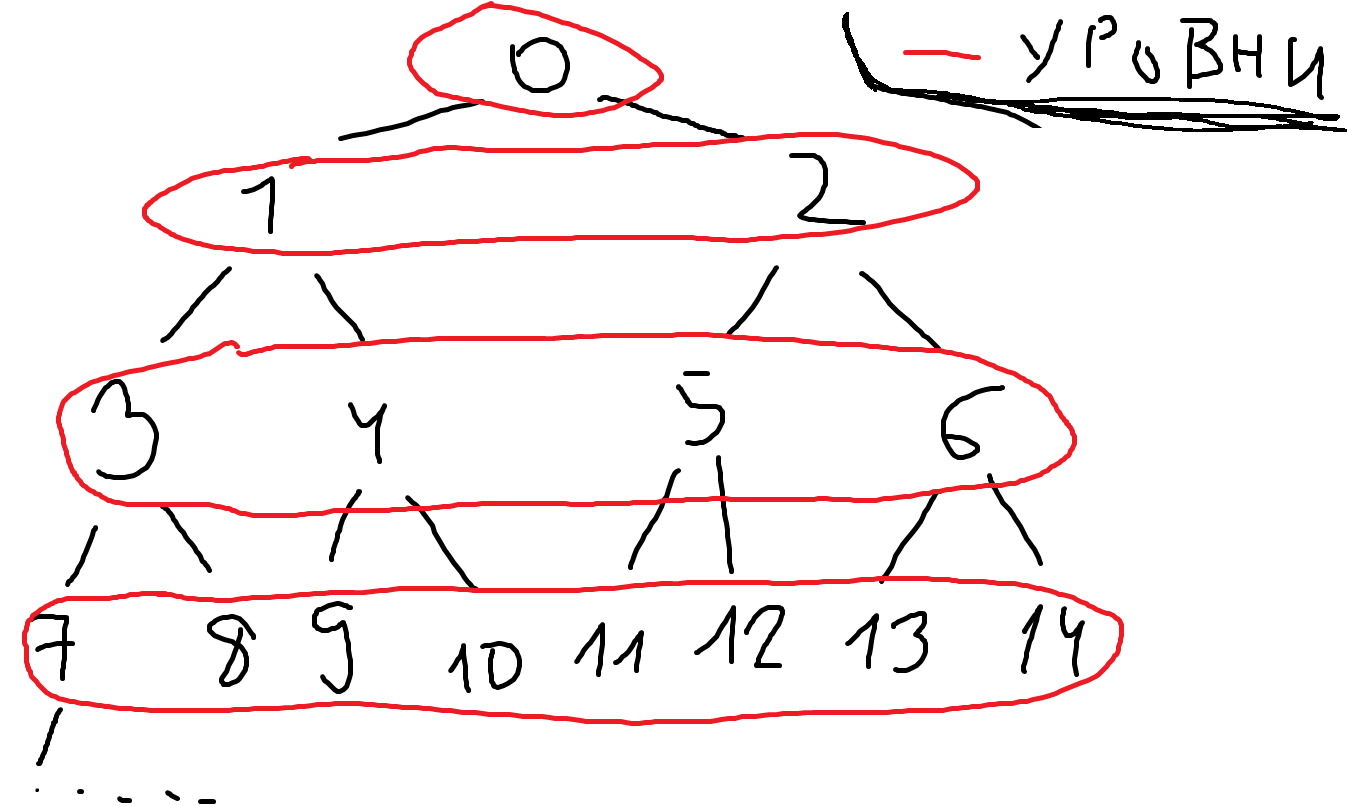
1 7 13, 3 7 13, 7 7 13, 8 7 13, 4 7 13, 9 7 13, 10 7 13, 2 7 13, 5 7 13, 11 7 13, 12 7 13, 6 7 13, 13 7 13, 14 7 13

2. При запуске f(0, 1000000000, 123456789) вернётся единственное число 0.

Происходит это по тому, что в единственное условие, при котором рекурсивный вызов вернёт 1, программа не зайдёт, потому что a > b.

3. Представим себе двоичное дерево, значения которого совпадают с индексами дерева отрезков. Элементов в дереве бесконечно много, из нуля можно дойти до любого числа, двигаясь по рёбрам к сыновьям.



Элементы дерева разделены на «уровни».

Рассмотрим несколько случаев для этой задачи.

а) a > 0, b >= a.

1) b > 2a. Так как рекурсивная функция зайдёт в а (n = a), она не может зайти дальше, чем 2a (n <= 2a). Вследствие того, что при n = a, дальше вызовов не делается, наибольшее значение n, при котором ещё будет делаться вызов, это a – 1. При этом их два: 2 \* (a – 1) + 1 и 2 \* (a – 1) + 2 => 2a – 1 и 2a.   
Поэтому в случае b > 2a ответ будет равен 2a – a + 1 = a + 1.  
2) b <= 2a. В этом случае рекурсивная функция вернёт значение 1 на отрезке [a, b]. То есть ответ равен b – a + 1.

В итоге получаем ответ: min(a + 1, b – a + 1). Почему же такой ответ? Очевидно, что, если b > 2a, то b – a + 1 > a + 1, тогда ответ у нас будет правильный: а + 1.  
Наоборот, если b < 2a, то b – a + 1 < a + 1, поэтому мы выберем правильный ответ. Если же b = 2a, то ответ равен b – a + 1 = a + 1.

б) a > 0, b < a.

В этом случае рекурсивный вызов всегда будет возвращать 0, так как программа не зайдёт в условие a <= n <= b, при заходе которого возвращает 1. Просто складываются нули.

в) a <= 0, b >= 0.

В этом случае программа зайдёт в условие, которое вернёт 1: a <= n <= b.

г) a <= 0, b < 0

В этом случае программа зайдёт в условие, которое вернёт 0: n > b. (n = 0, b < 0)