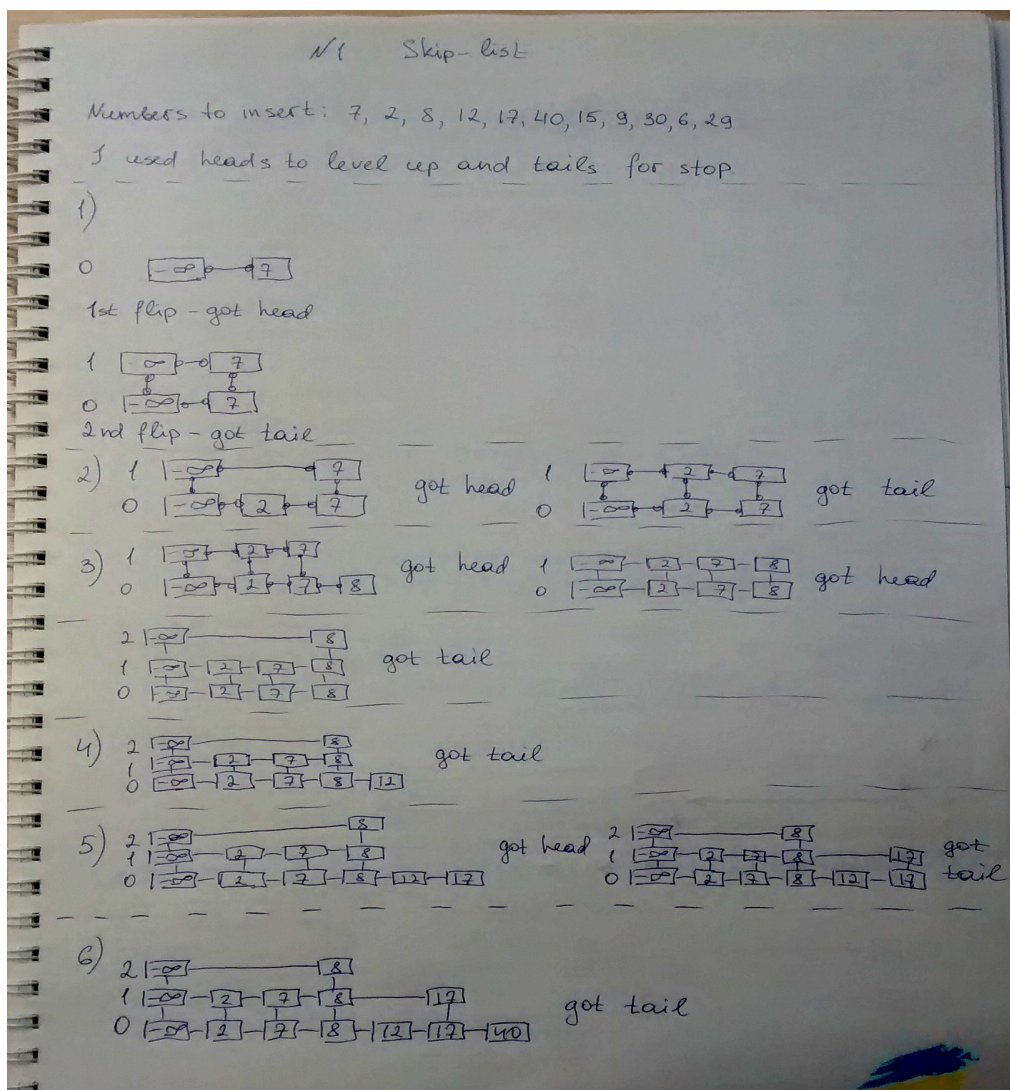


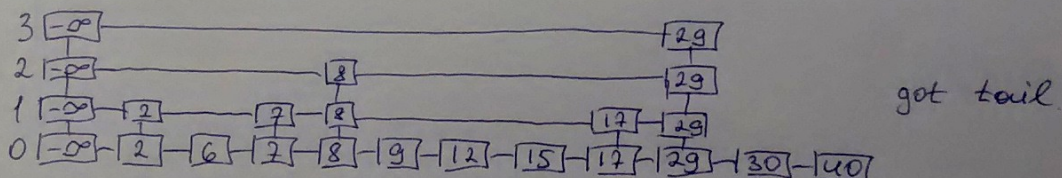
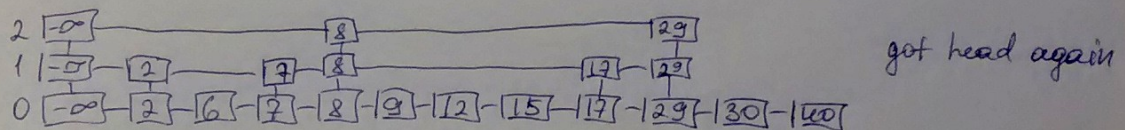
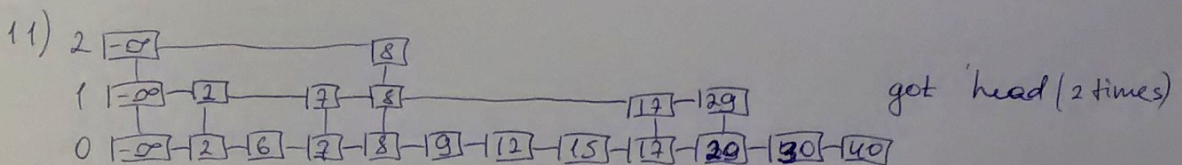
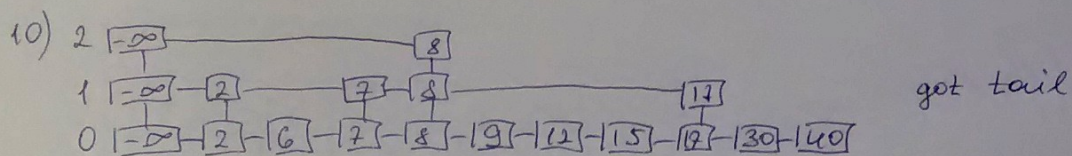
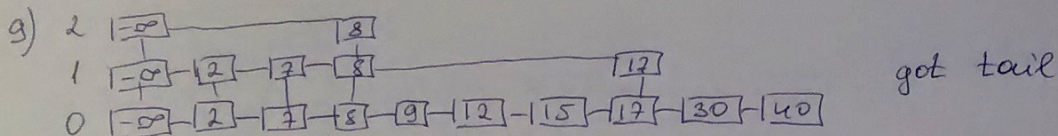
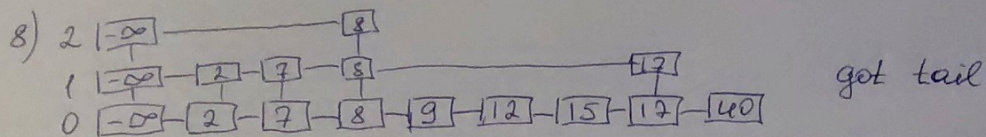
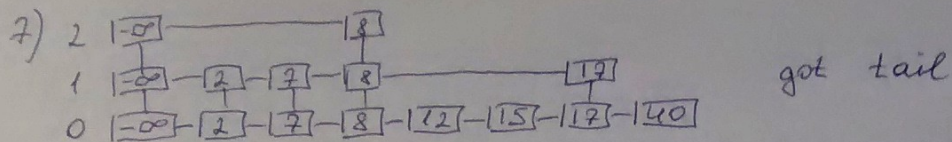
Homework #3

Skiplists, trees

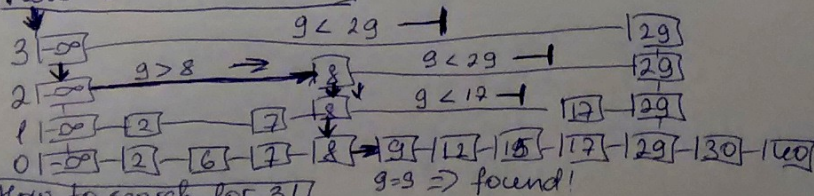
Anastasiia Konoplina

1. Simulate the insertion of values 7, 2, 8, 12, 17, 40, 15, 9, 30, 6, 29 into the skip-list by drawing it on paper. Use a coin for tossing head or tail. Show, how to search for 9 and 31 in that skip-list. Then delete 15.

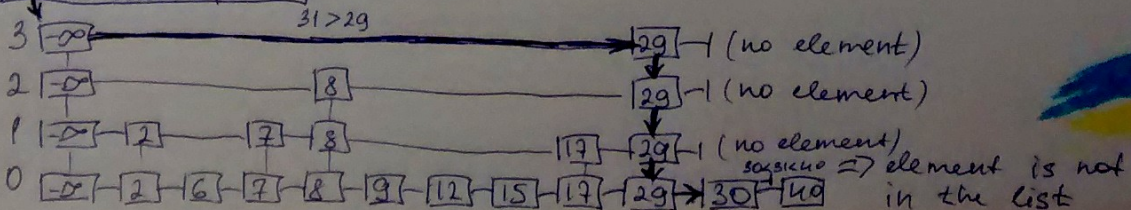




How to search for 9

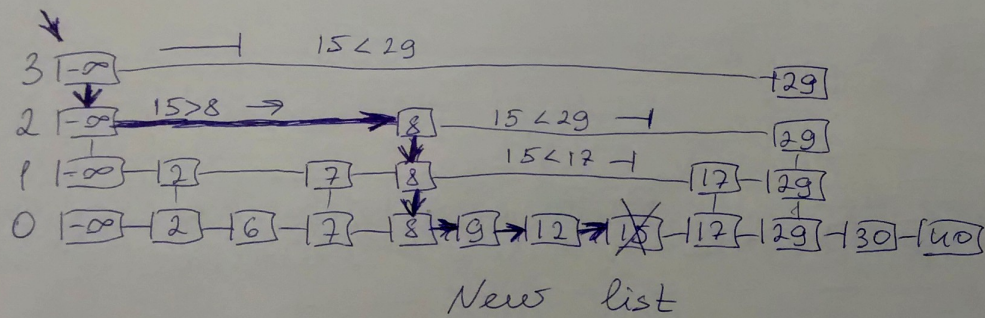


How to search for 31

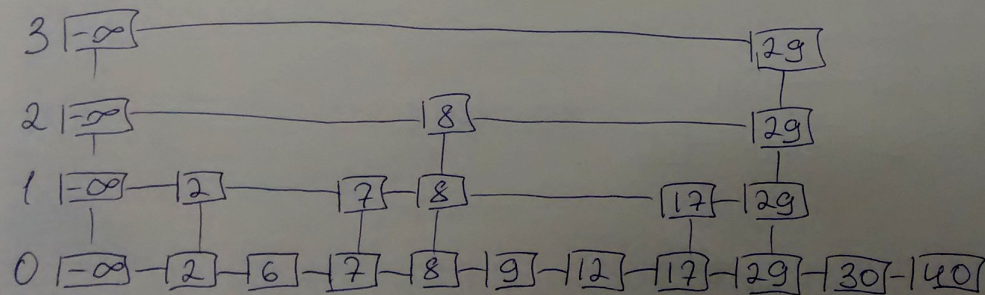


How to delete 15:

- 1) Find element 15 in the list
- 2) Remove from all levels



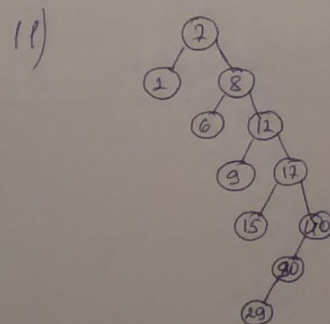
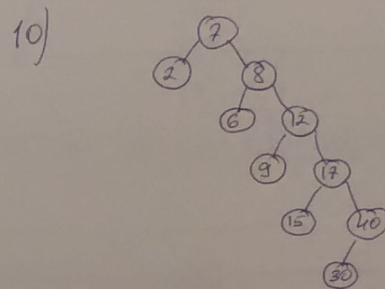
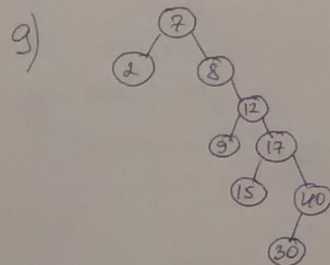
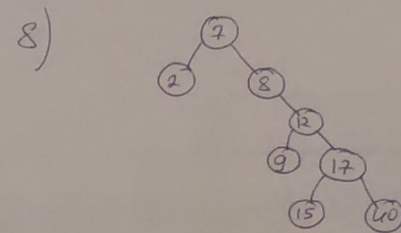
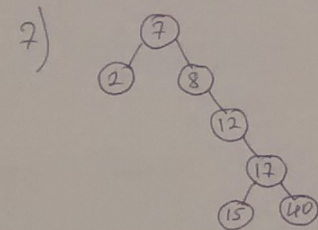
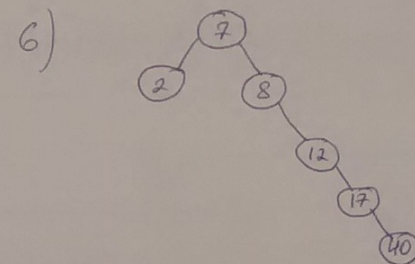
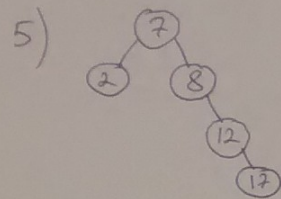
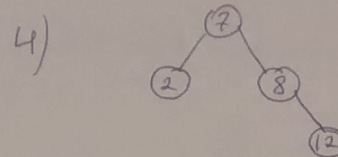
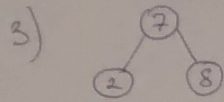
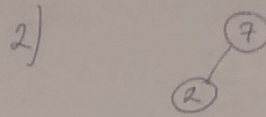
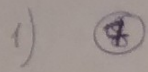
New list



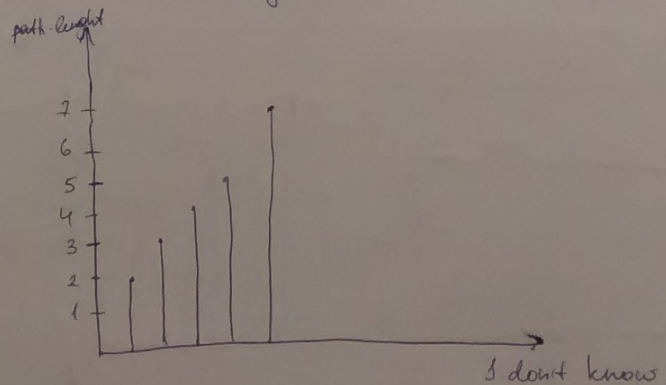
2. Store the same values in binary search tree. How unbalanced it is? Now insert the same numbers in the same order but simulate the AVL tree construction. How many rotations and how many operations to maintain the AVL height difference counting in the tree?

Binary tree

Numbers to insert: 7, 2, 8, 12, 17, 40, 15, 9, 30, 6, 29




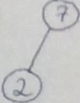
Path-length to leaves distribution




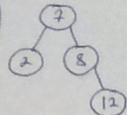
AVL tree

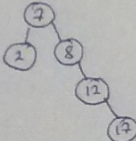
Numbers to insert: 7, 2, 8, 12, 17, 40, 15, 9, 30, 6, 29

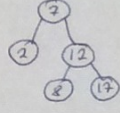
1)  Balance(7)=0
Balanced

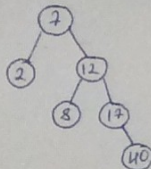
2)  balanced
Balance(7)=1

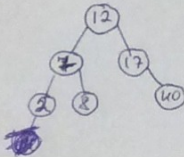
3)  Balance(7)=0
Balanced

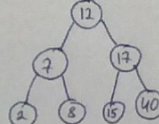
4)  Balance(8)=-1
Balanced

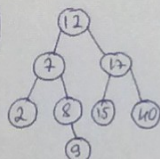
5)  Balance(12)=-1
Balance(8)=-2
Unbalanced

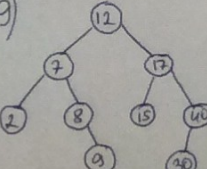
\Rightarrow  Balance(12)=0
Balance(7)=-1
Balanced
#rotations=1

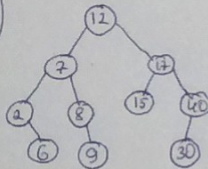
6)  Balance(17)=-1
Balance(12)=-1
Balance(7)=-2
Unbalanced

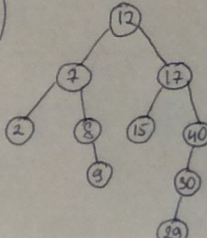
\Rightarrow  Balance(7)=0
Balance(17)=-1
Balance(12)=0
Balanced
#rotations=2

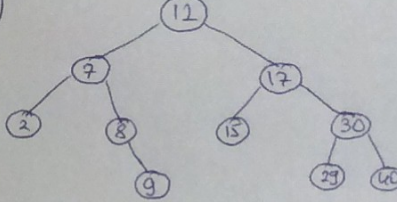
7)  Balance(7)=0
Balance(17)=0
Balance(12)=0
Balanced

8)  Balance(8)=-1
Balance(7)=-1
Balance(12)=-1
Balanced

9)  Balance(40)=1
Balance(17)=-1
Balance(12)=0
Balanced

10)  Balance(2)=-1
Balance(7)=0
Balance(12)=0
Balanced

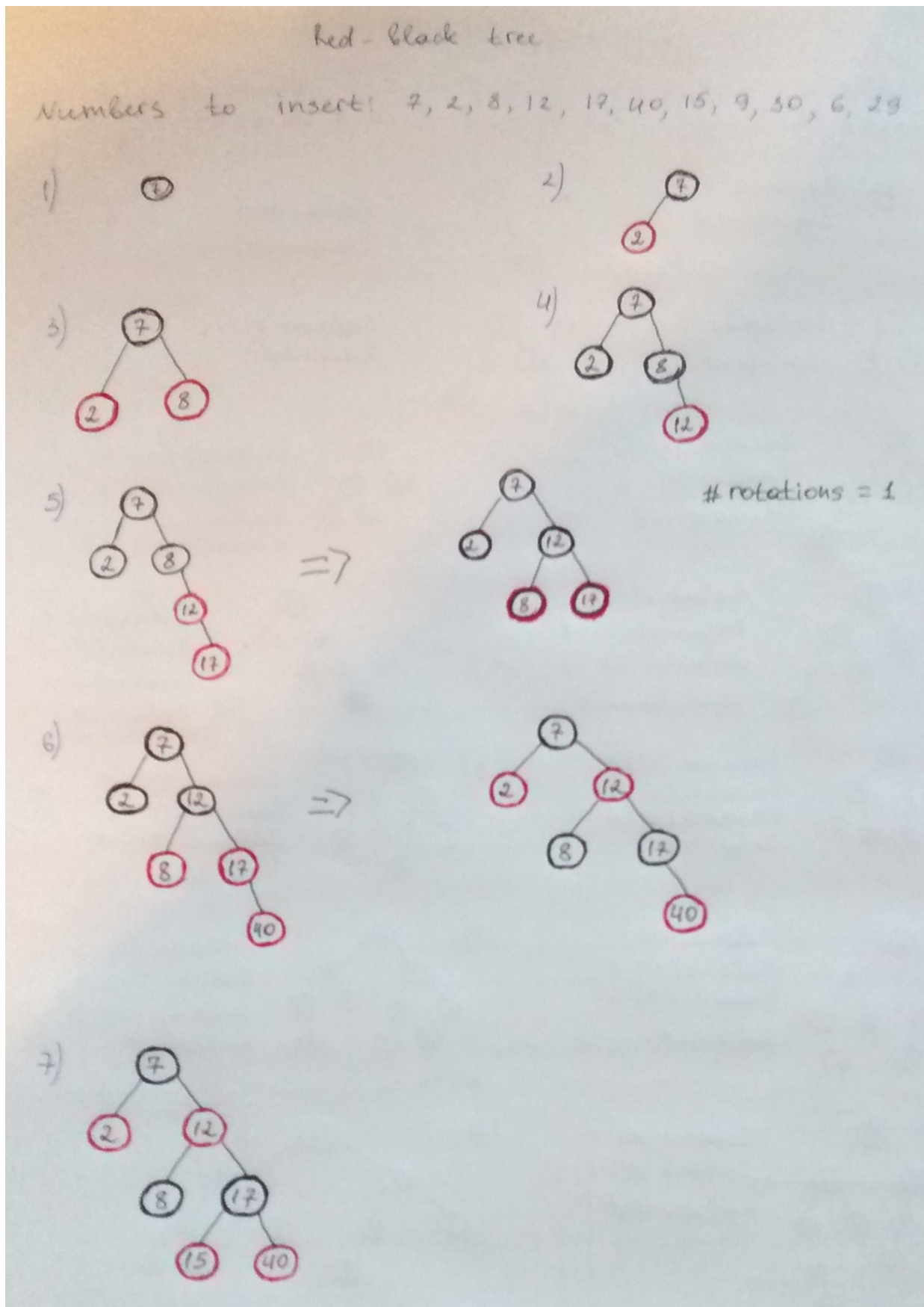
11)  Balance(30)=1
Balance(40)=2
Unbalanced

12) 

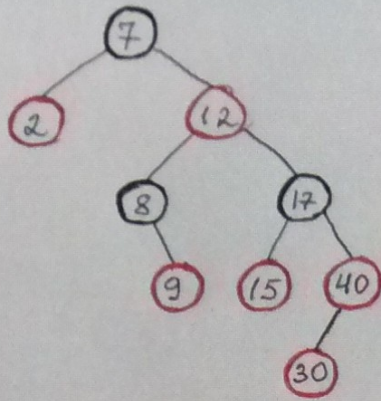
Balance(30)=0
Balance(17)=-1
Balance(8)=-1
Balance(7)=-1
Balance(12)=0
Balanced!!!

#rotations=3

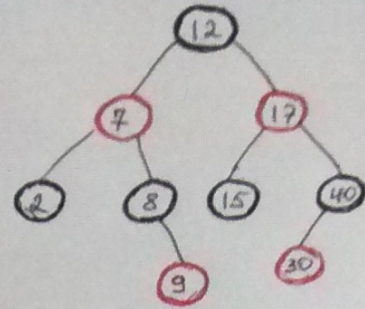
3. Repeat the same simulations for red-black tree.



8)

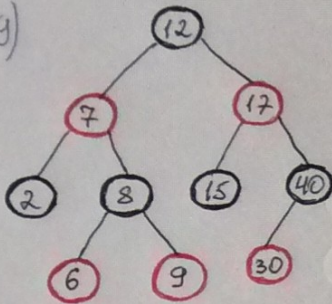


=>

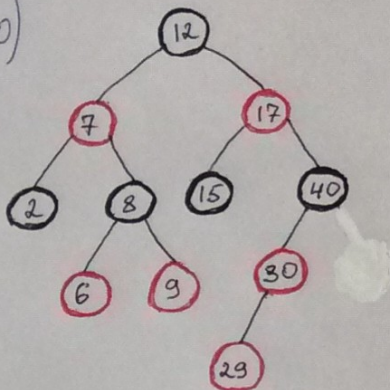


rotations = 2

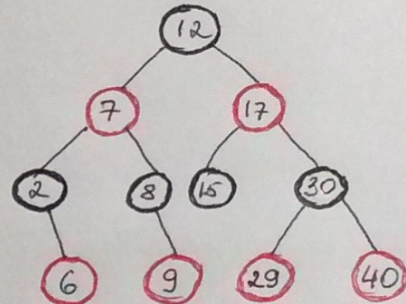
9)



10)

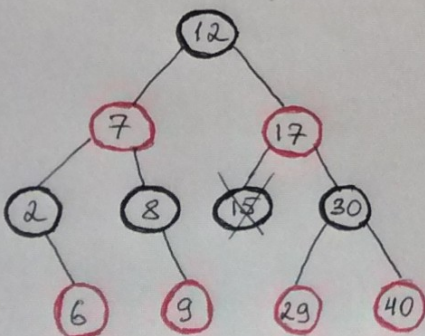


=>

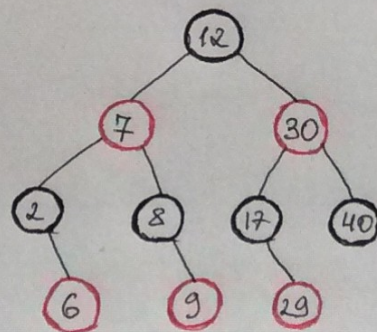


rotations = 3

Delete 15



=>

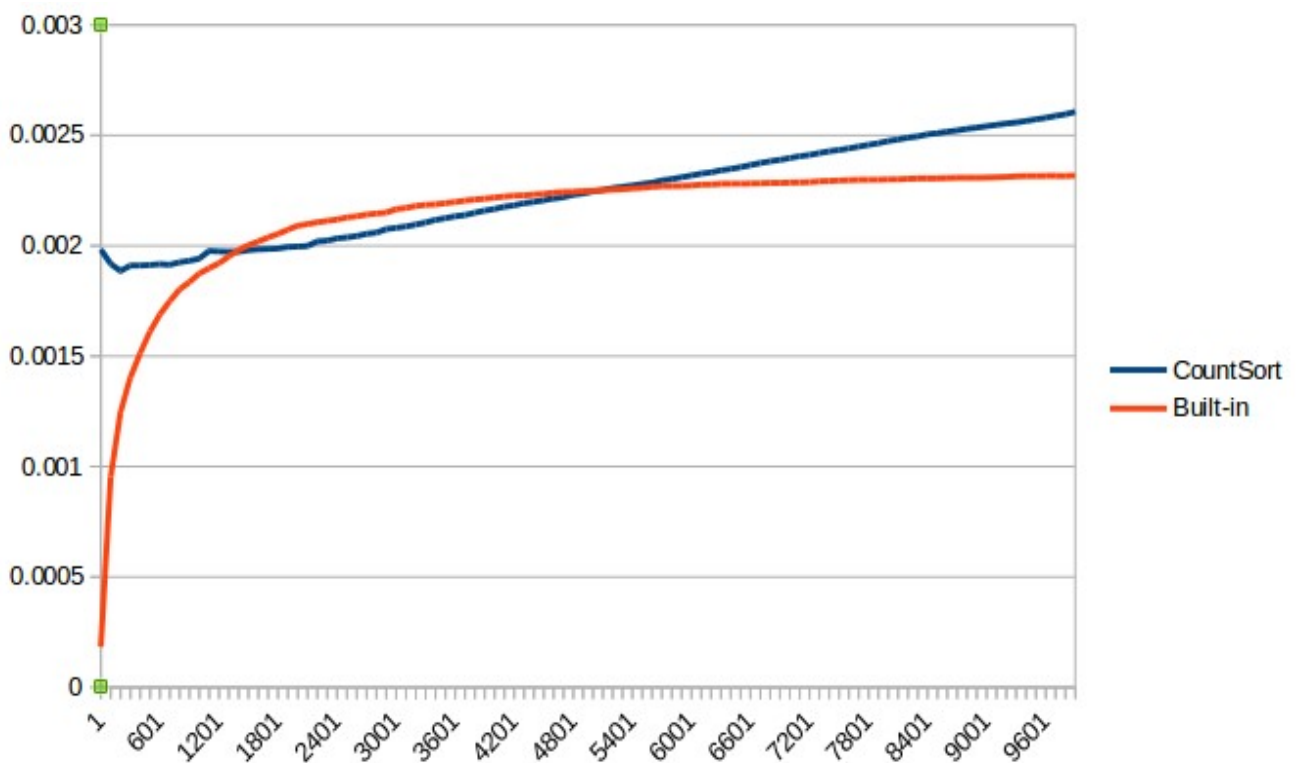


4. Beat the speed of a built-in sort for 64-bit unsigned integers. Hint: "cheat" by generating only small integers, e.g. 0..255 and apply for example a linear-time counting sort. How large integers can you generate to still be competitive with built-in sort?

I used counting sort, the code of the function is below:

```
def CountSort(array, k):  
    count = 0  
    res = [0] * len(array)  
    h_array = [0] * (k + 1)  
    for i in array:  
        h_array[array[i]] += 1  
    for l in range(1, len(h_array)):  
        while h_array[l] > 0:  
            res[count] = l  
            h_array[l] -= 1  
            count += 1  
    return res
```

For testing I generated array of 10 000 elements of different range of integers (from 1 to 10000). I expected CountingSort function to work faster with array of numbers [0; 255], but it was not like that. On small numbers built-in function worked much faster, but on some interval of integers CountingSort function worked a little bit faster (see picture below). For all spots mean of 10 values were taken.



5. Estimate how many cycles of count-sort you can afford to perform with radix-sort to beat the built-in sort on some large data. Try to measure the time of one count sort for different bit-lengths. To sort based on some "middle bits" - you would need to extract those first. How to get the integer value of e.g. 10 bits from 30..21, fast? (21st bit would be the least significant bit of the 10-bit integer, and 30th the most significant bit).

I understood the task fully when it was too late to implement it :)