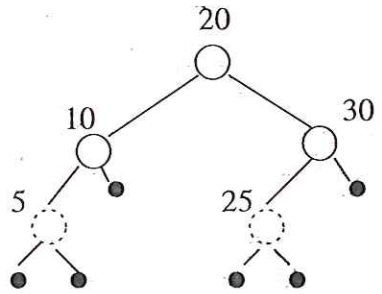
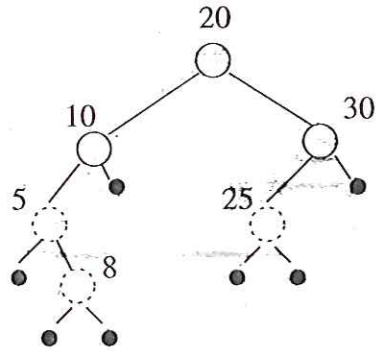
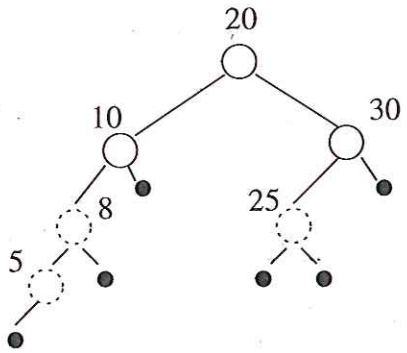
1. Start with the following tree:



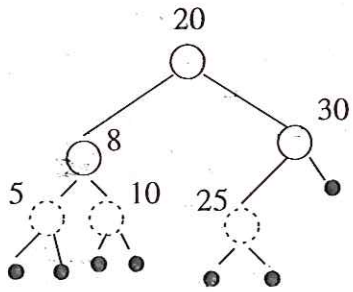
Insert 8. This gives us the following tree.



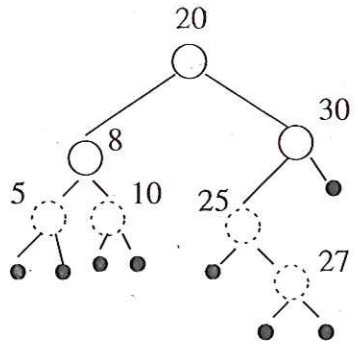
This puts us in case 2 (aunt is black and we have the zig-zag pattern). We first left rotate to get this tree:



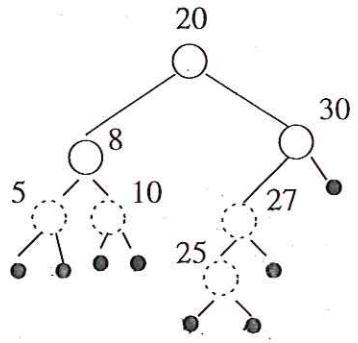
This moves us to case 3 which we fix by doing a right rotate and a recolor:



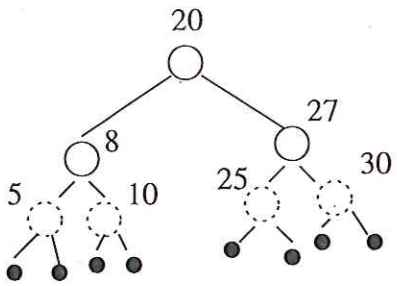
Now we insert 27:



Again we are in case 2. We first left rotate:



Now we right rotate and recolor:



2. If a node is red, then both of its children are black. From this property we can say $b \geq h/2$

where b = black height of the RB tree
h = height of the RB tree

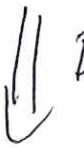
(a) Now for maximum number of internal nodes the RB tree will be a complete binary tree with alternating black and red level of nodes and we have $b = h/2$

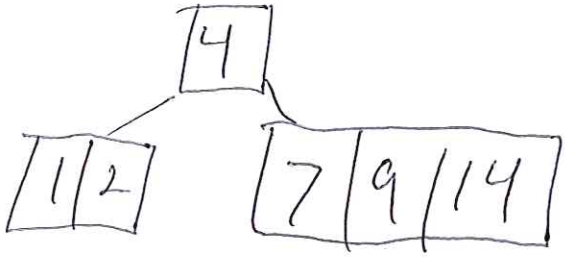
Total number of internal nodes for this kind of tree is $2^h - 1 = 2^{2b} - 1$

(b) For minimum number of internal nodes the RB tree will be a binary tree consisting of all black nodes and we have $b = h$

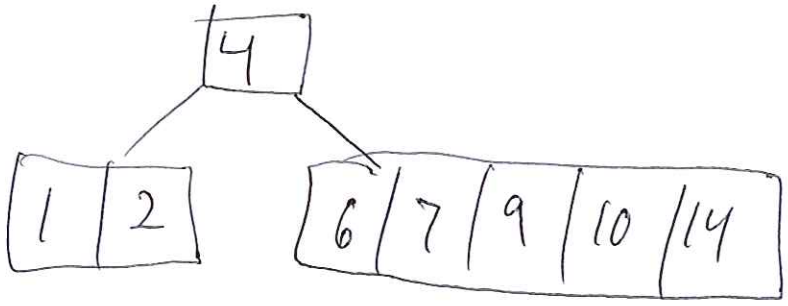
Total number of internal nodes for this kind of tree is $2^h - 1 = 2^b - 1$

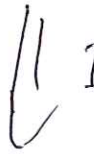
3. First 5 inserts:

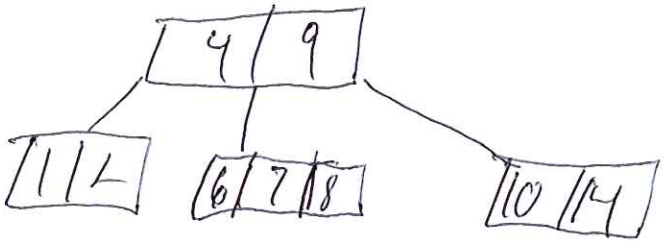




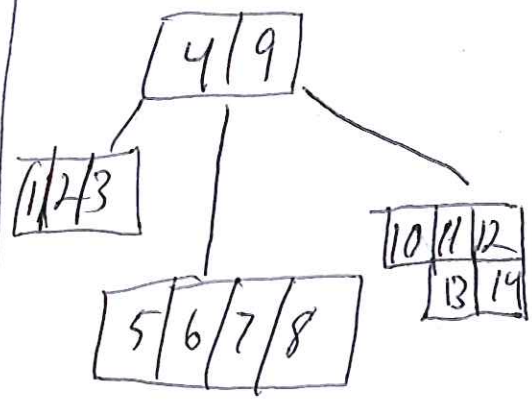
Before Inserting 8:







Final Tree:



(9)

```
// S = Red-black tree containing all interval
// root = root of tree S
// i = interval to search
// interval_set = A set containing all
                  intervals that match i

find_All (S, root, i)
{
    interval_set = null;
    while (true)
    {
        interval = Interval_search (S, root, i
                   // returns the first interval
                   // in S that matches i
        if (interval == null)
            return interval_set;
        else
        {
            interval_set = interval_set ∪
                           interval;
            S = S - interval; // deletes the
                // node from the tree
                // that matches interval;
            if (S is empty)
                return interval_set;
        }
    }
}
```

Running time analysis :

- For each iteration of while loop the function call Interval_Search() will return exactly one interval that overlaps i
- At each iteration the interval that overlaps i will be deleted from the tree so no Interval_Search() will return any duplicate interval.
- So the while loop will iterate at most k times if there are k intervals that overlap with i.
- cost of Interval_Search() is $\log n$
- cost of deleting node from Red-Black tree is $\log n$
- Total cost $= O(k(\log n + \log n))$
$= O(k \log n)$