Assignment 2
CS 3340B

Name: Maryam Karimi Firozjaei
Student #: 250999590

1) 8.2-1

Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array A = {6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2}

	1	2	3	4	5	6	7	8	9	10	11
A	6	0	2	0	1	3	4	6	1	3	2

	0	1	2	3	4	5	6
C	2	2	2	2	1	0	2

(a)

	0	1	2	3	4	5	6
C	2	4	6	8	9	9	11

(b)

	1	2	3	4	5	6	7	8	9	10	11
B						2					

	0	1	2	3	4	5	6
C	2	4	5	8	9	9	11

(c)

	1	2	3	4	5	6	7	8	9	10	11
B						2		3			

	0	1	2	3	4	5	6
C	2	4	5	7	9	9	11

(d)

	1	2	3	4	5	6	7	8	9	10	11
B				1		2		3			

	0	1	2	3	4	5	6
C	2	3	5	7	9	9	11

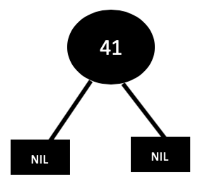
(e)

	1	2	3	4	5	6	7	8	9	10	11
B	0	0	1	1	2	2	3	3	4	6	6

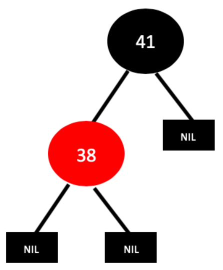
(f)

2) 13.3-2

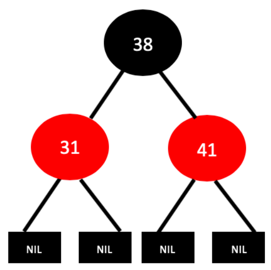
Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.



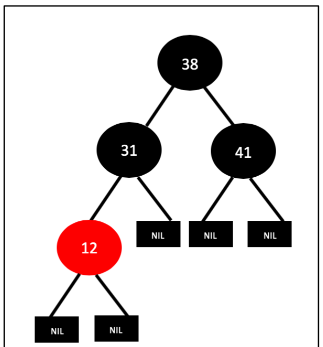
1) Insert 41



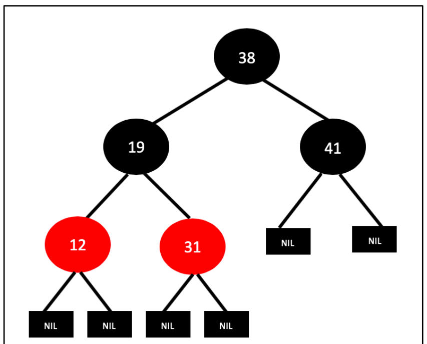
2) Insert 38



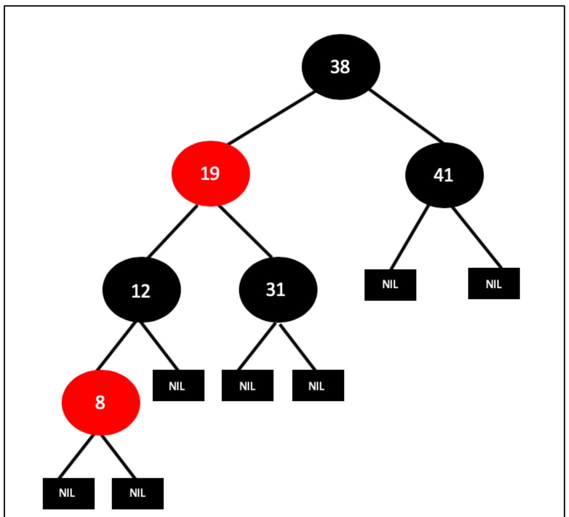
3) Insert 31



4) Insert 12



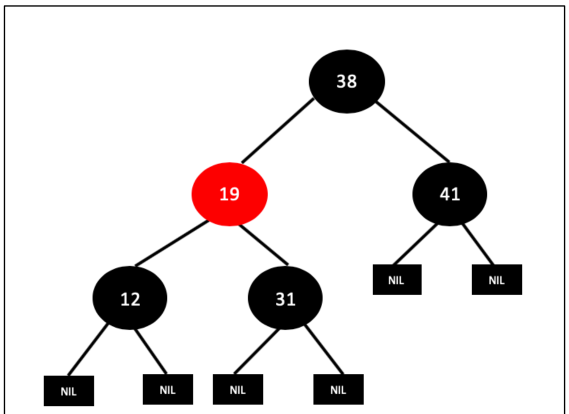
5) Insert 19



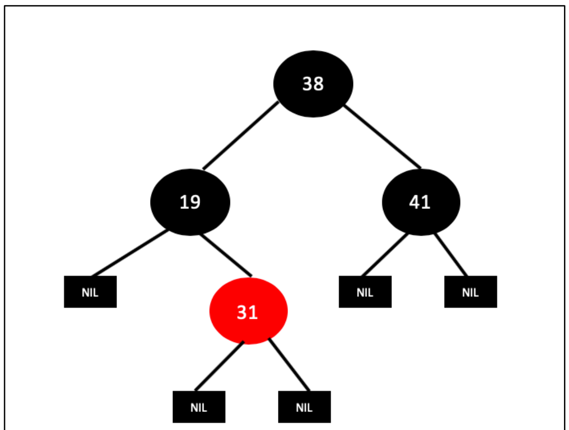
6) Insert 8

3) 13.4-3

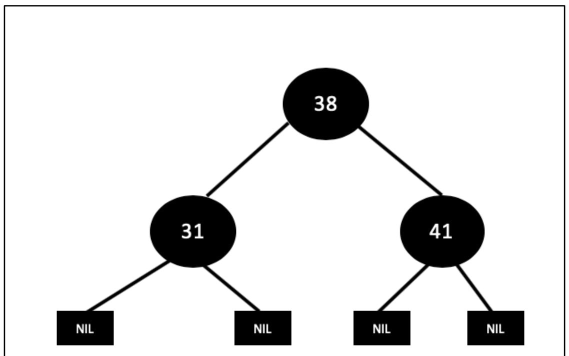
In Exercise 13.3-2, you found the red-black tree that results from successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty tree. Now show the red-black trees that result from the successive deletion of the keys in the order 8, 12, 19, 31, 38, 41.



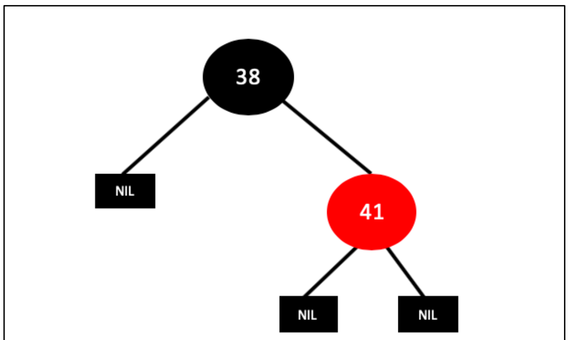
1) Delete 8



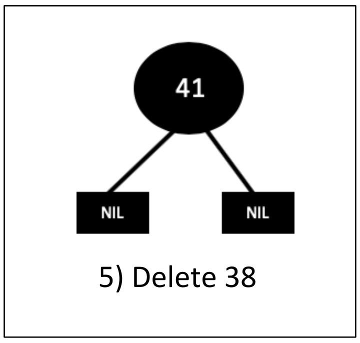
2) Delete 12



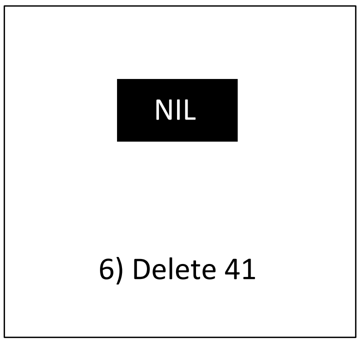
3) Delete 19



4) Delete 31



5) Delete 38



6) Delete 41

4) Given k sorted sequences each of which has n elements, design an efficient algorithm to sort all kn elements into one sorted sequence. What is the time complexity of your algorithm?

This algorithm takes an array of k sorted arrays (A). Each of the sorted arrays has n elements

```
MergeKArrays($A$ , $n$)
    $k = A.size$
    let $Array[0..n*k]$
    MergeTwoArrays($A[0]$,$Array$,$Array$,$n$,0) //Merge empty Array and A0
    for $j$ = 1 to $A.size$-1 // A.size is our k
        MergeTwoArrays($A[j]$,$Array$,$Array$,$n$,$j*n$)
    end for
    return $Array$
end
```

This algorithm merges two sorted Arrays called A1(size n1) and A2(size n2), and stores the results in A3

```
MergeTwoArrays($A1$,$A2$,$A3$,$n1$,$n2$) //n1 is length of A1 and n2 is length of A2

    $i = 0$
    $j = 0$
    $k = 0$
    while ($i<n1$ && $j<n2$)
        if $A1[i] < A2[j]$
            $A3[k] = A1[i]$
            $k = k + 1$
            $i = i + 1$
        else
            $A3[k] = A2[j]$
            $k = k + 1$
            $j = j + 1$
    end while


    while ($i$<$n$1)
        $A3[k] = A1[i]$
        $k = k + 1$
        $i = i + 1$
    end while

    while ($j$<$n$2)
        $A3[k] = A2[j]$
        $k = k + 1$
        $j = j + 1$
    end while

end
```

Time complexity:

Here, I use MergeTwoArrays procedure to merge the first array with an empty array, then merge it with the second array, then the third, and so on.
MergeTwoArrays takes O(n1+n2) time. Therefore, the first merge takes O(n), the second takes O(n+n), the second takes O(2n+n), and so on.
So, the running time is:
$(n) + (n + n) + (2n + n) + (3n + n) + ... + ((k\text{-}1)n + n))$
$= n + 2n + 3n + 4n + ... + kn$
$= (1+2+3+...+k)n$
$= \mathbf{O\ (k^2n)}$

---

5) (optional for bonus credit) **16.3-6**

Suppose we have an optimal prefix code on a set C = {0, 1, ... , n-1} of characters and we wish to transmit this code using as few bits as possible. Show how to represent any optimal prefix code on C using only 2n -1+ n $\lceil \lg n \rceil$ bits. (Hint: Use 2n-1 bits to specify the structure of the tree, as discovered by a walk of the tree.)

We know that every full binary tree has exactly 2n-1 nodes. The structure of this type of tree can be encoded by performing a preorder traversal as follows:
Start a preorder traversal on tree
For each node,
    write 1 if the node is a leaf
    write 0 if the node is internal
We can specify the structure of the tree this way because it is a full binary tree.

Since any character of C can be encoded using $\lceil \lg n \rceil$ bits and we have n characters, we can encode it using $n\lceil \lg n \rceil$ bits. This comes to a total of $2n$-1+ $n\lceil \lg n \rceil$ bits.

---

6) **21.4-2**

Prove that every node has rank at most $\lfloor$lg n$\rfloor$.

Proof by strong Induction:

We prove by induction on the number of nodes.

When $n = 1$, then the node has rank equal to $\lfloor \lg 1 \rfloor = 0$

Assume that this holds for $k \leq n$.

Suppose we have two disjoint sets with a and b nodes respectively such that a,b $\leq n$.

Based on our assumption, the root of the first set has rank at most $\lfloor$lg a$\rfloor$ and the root of the second set has rank at most $\lfloor$lg b$\rfloor$.

Given n+1 nodes, perform a union operations on the previously mentioned disjoint sets. Suppose the ranks are equal. (if the ranks are not equal, we are done). Then the rank of the union will be increased by 1, and the set will have rank of:

$\lfloor$lg b$\rfloor$ + 1 $\leq$ $\lfloor$lg (n+1)/2$\rfloor$ + 1 = $\lfloor$lg (n+1)$\rfloor$ □

Therefore, every node has rank at most $\lfloor$lg n$\rfloor$.

---

7) **21.4-3**

In light of Exercise 21.4-2, how many bits are necessary to store *x.rank* for each node *x*?

Every rank is at most $\lfloor \lg n \rfloor$, therefore, $\Theta(\lg(\lg(n)))$ bits are necessary to store the rank for each node.