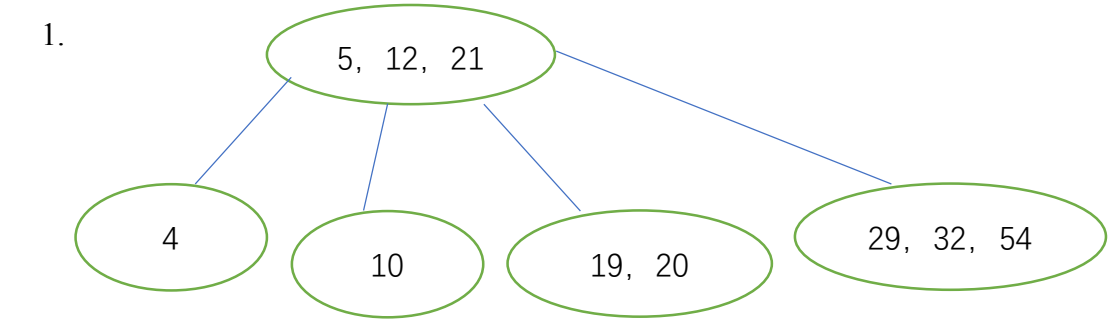
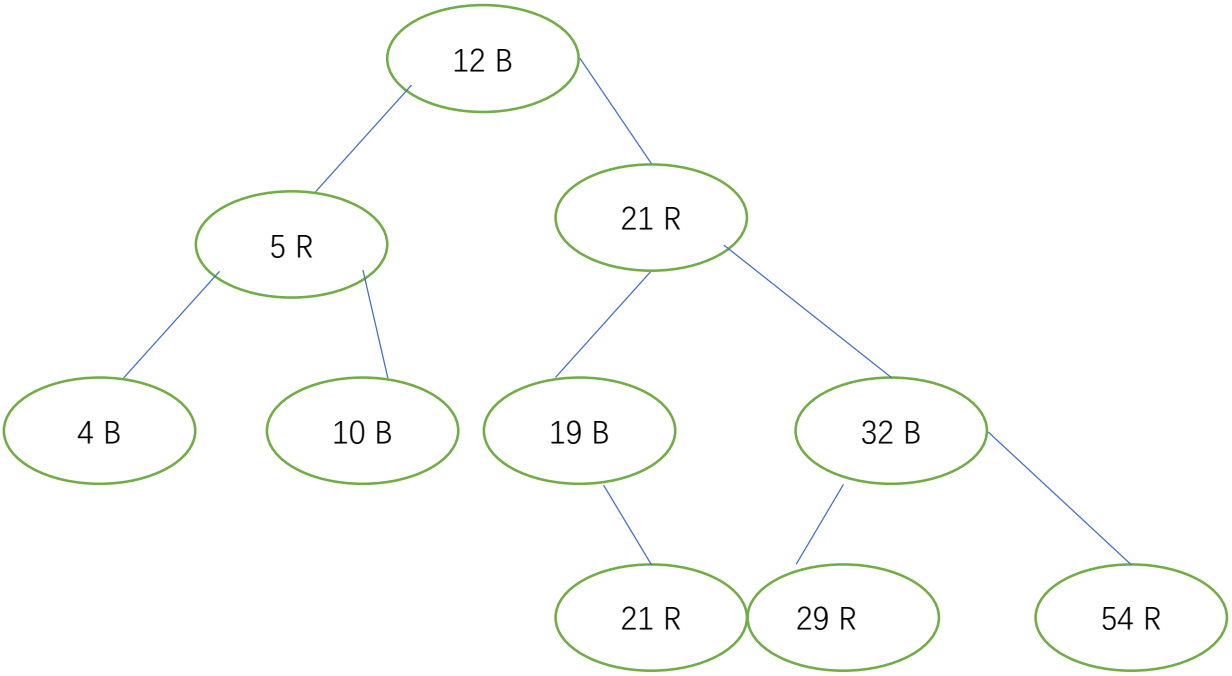
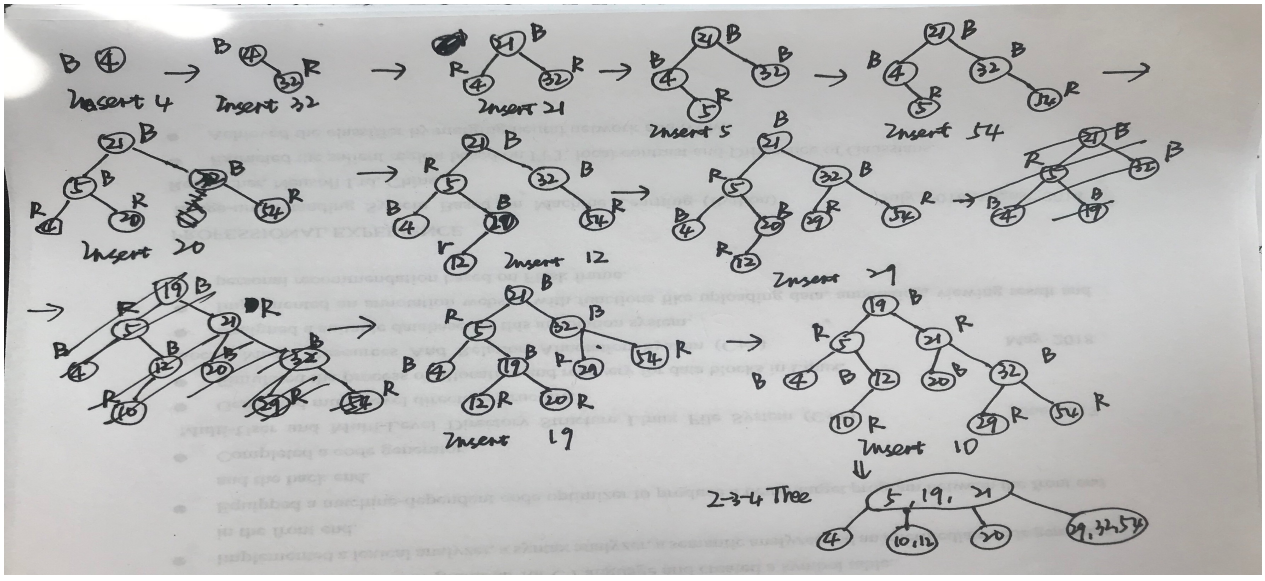
1.


5, 12, 21
4
10
19, 20
29, 32, 54


12 B
5 R
21 R
4 B
10 B
19 B
32 B
21 R
29 R
54 R

2 .

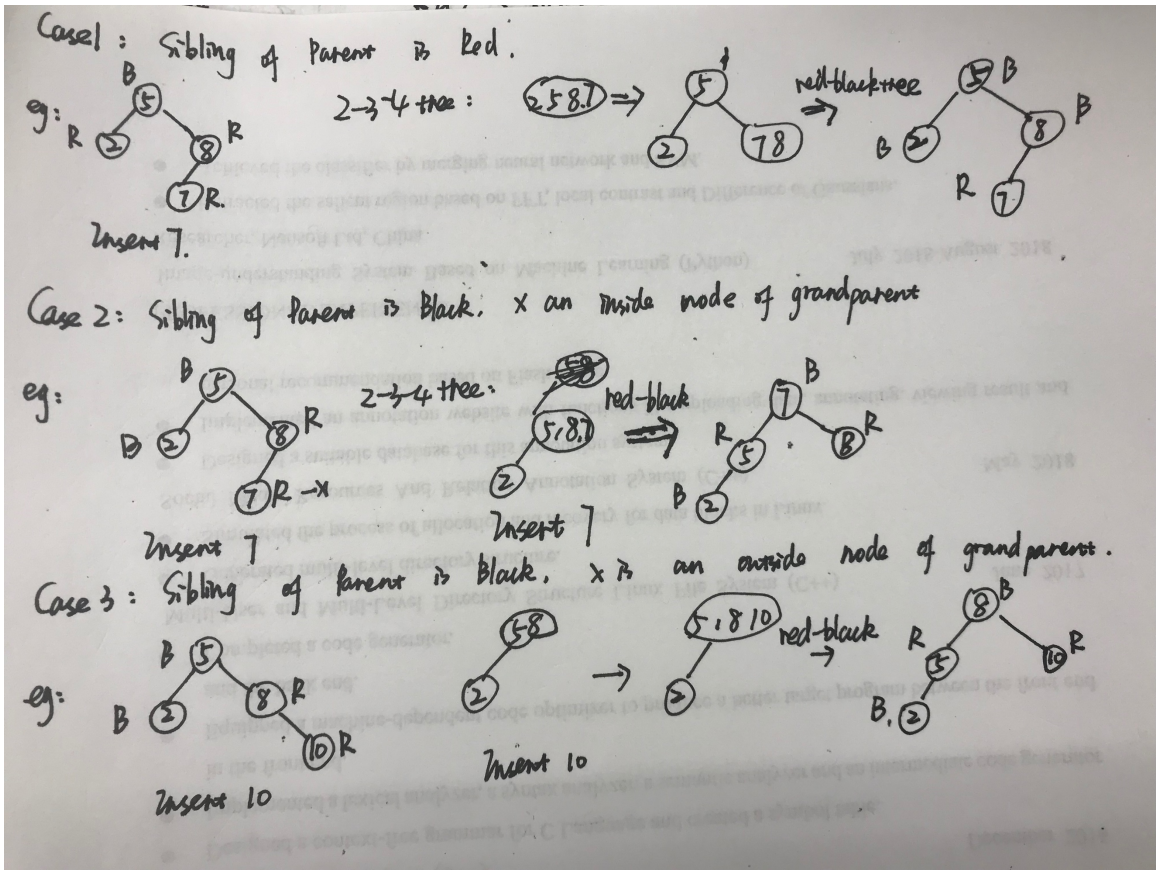

Insert 4
Insert 32
Insert 21
Insert 5
Insert 54
Insert 20
Insert 12
Insert 19
Insert 10
2-3-4 Tree
5, 19, 21
4
10,12
20
29,32,54

3.The maximum height of 2-3-4 tree is $\lfloor \log_2 n \rfloor + 1$, because there are at least two nodes in 2-3-4 tree, if all nodes in 2-3-4 tree is two nodes, it will have the maximum height which is $\lfloor \log_2 n \rfloor + 1$.

The minimum height of 2-3-4 tree is $\lfloor \log_4 n \rfloor$, because there are at most four nodes in 2-3-4 tree, if all nodes in 2-3-4 tree is four nodes, it will have the minimum height which is $\lfloor \log_4 n \rfloor$.

4. From the red-black properties, we have that every simple path from node x to a descendant leaf has the same number of black nodes and that red nodes do not occur immediately next to each other on such paths. Then the shortest possible simple path from node x to a descendant leaf will have all black nodes, and the longest possible simple path will have alternating black and red nodes. Since the leaf must be black, there are at most the same number of red nodes as black nodes on the path.

5.



In each case, I firstly convert it into 2-3-4 tree. And then I insert the node into 2-3-4 tree,

and according to 2-3-4 tree insertion rules I adjust the 2-3-4 tree, finally I convert the 2-3-4 tree to a red-black tree which is consistent with the result of red-red violation's solution.

6. In this problem, firstly I want to put all drives in a Red_Black tree in order by their remaining storage capacity, and then I store images in drives. Every time finding a drive for the image from the root of RB tree, if the image's size is larger then the root, then goes to its right child, if the image's size is smaller than its left child, then goes to its left child, until the image's size is smaller than the node' storage. Next, I delete the node which stores the new image, insert it again after renewing it remaining storage.

7.def conversion(A,T)

```
size =A.length
T.root=A[size/2]
conversion(A[1..size/2-1],T.left)
conversion(A[size/2+1..size], T.right)
```

I would like to take the half element of A to be the root of T, then split the array into three pieces A[1..size/2-1], root, A[size/2+1..size], finally recursively execute the algorithm conversion, so the running time is O(n).

8.(1) if $\max\{size(t.left), size(t.right)\} \leq \frac{2}{3} size(t)$, then every node's height will be O(logn) ,so it must be a balanced tree.

(2) LEFT_ROTATE(T,x)

```
    y=x.right
    x.right=y.left
    if y.left≠T.nil
        y.left.p=x
    y.p=x.p
    if x.p==T.nil
        T.root=y
    Elseif x==x.p.left
        x.p.left=y
    else
        x.p.right=y
```

x.left=x

y.size=x.size

x.size=x.left.size+x.right.size+1

RIGHT_ROTATE(T,x)

y = x.left

x.left=y.right

if y.right≠T.nil

y.right.p=x

y.p=x.p

if x.p==T.nil

T.root=y

else if x==x.p.left

x.p.left=p

else

x.p.right=y

y.right=x

x.p=y

y.size=x.size

x.size=x.left.size+x.right.size+1

(3)def function(root)

if root.left=NULL &&root.right=NULL

return

function(root.left)

function(root.right)

if size(root.left)>$\lfloor size(root)/2 \rfloor$

right_rotation(root)

else if size(root.right)> $\lfloor size(root)/2 \rfloor$

left_rotation(root)

9.

Firstly, I design a data structure which include a guest's name, home planet, alliances, favorite color and kompromat.

Then, construct a Red_Black Tree in dictionary order by guests' name.

10.

```
data structure node {
      distance;
      roll;
      size; // the number of subtree rooted at this node
      sum; // the sum of toll booth whose distance is less than or equal to this node.
}
```

(a)
```
Def Insert(d ,r){
    Node x =new node();
    x.distance=d;
    x.roll=r;
    RB_INSERT(T, x);
    }
```

RB_INSERT(T,x) is the same with the algorithm in the lecture which augment size to the data structure, and the sum attribute will calculate its left subtree's toll.

(b)
```
def delete(k){
    x=T.root;
 j=OS_SELECT(x,k)
 n=j.left
while(n.right)
      n=n.right //find j predecessor n
if j.p.left=j      // replace j with n
      j.p.left=n
else j.p.right=n
n.p=j.p
if(n.left)      // if predecessor has a left child, replace n with its chid.
      n.p.right=n.left
```

n.left=j.left

j.left.p=n

n.right=j.right;

j.right.p=n

(3) toll(i,j)

Because there is an attribute sum in data structure, I can calculate toll(i,j) by calculate $\frac{\mathrm{sum(j)-sum(i-1)}}{j-i+1}$.