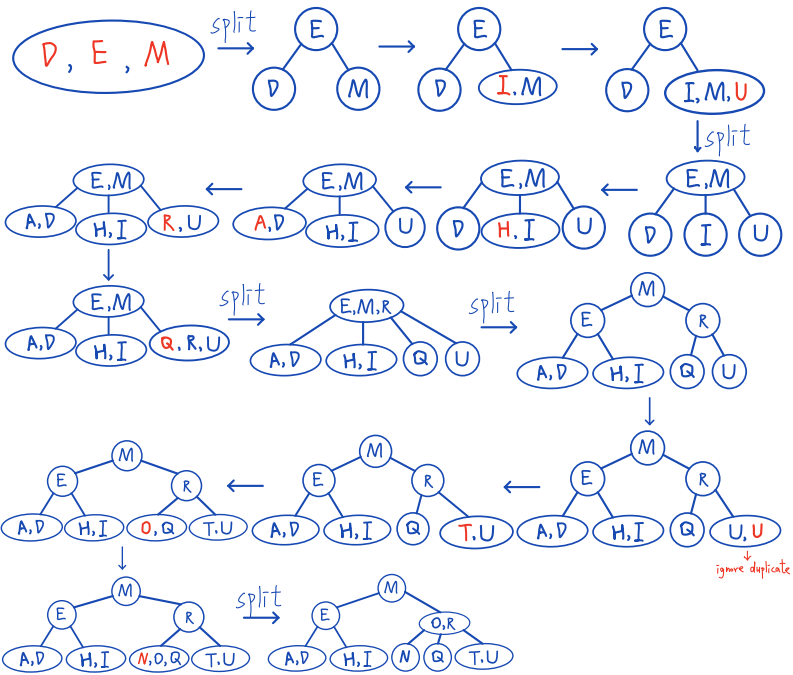


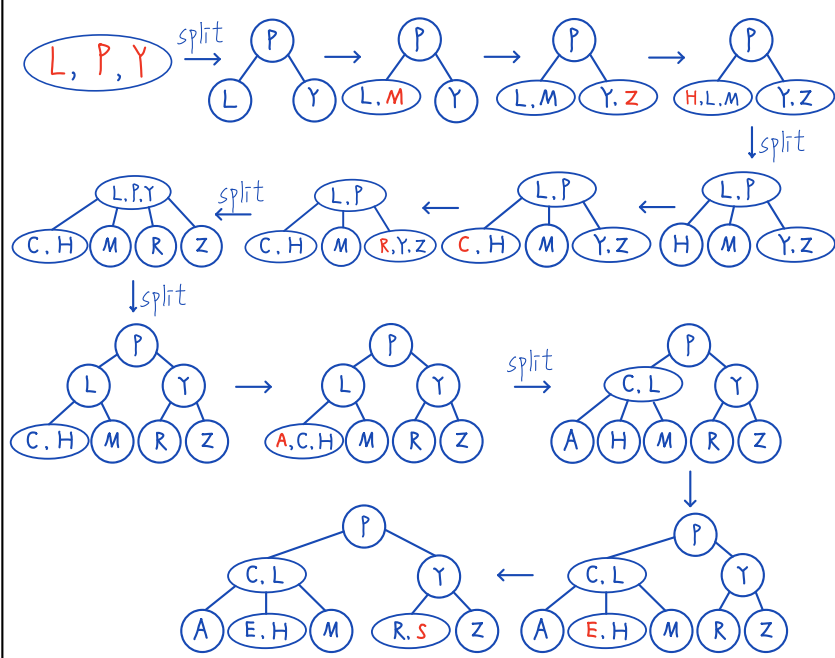
1. My Steps:

✂ = insert

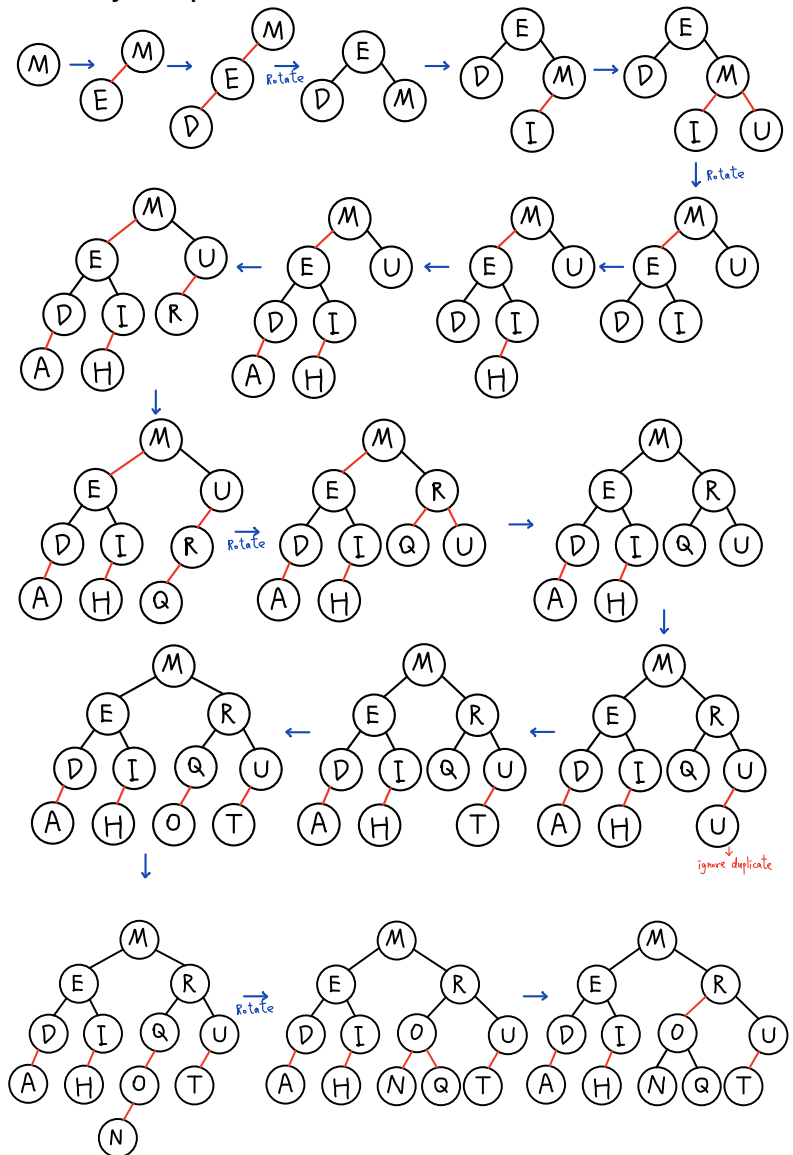


2. My Steps:

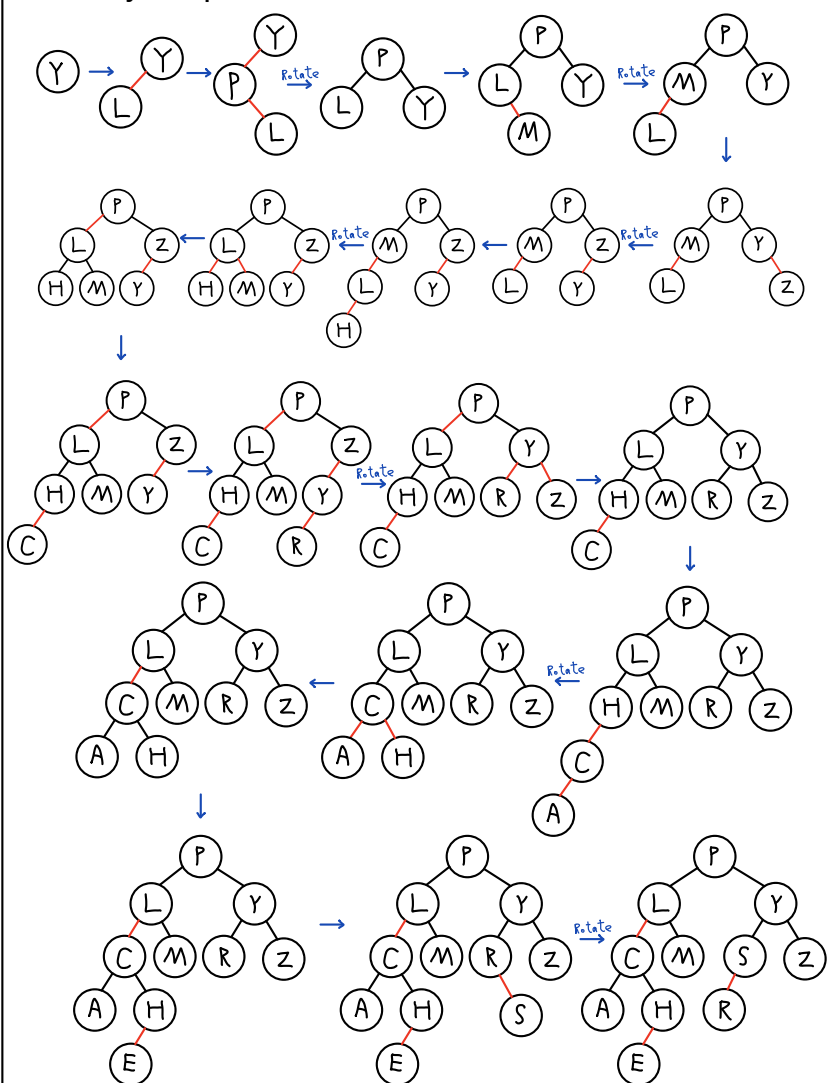
✂ = insert



3. My Steps:



4. My Steps:



5. My Python Code & explain:

I didn't complete this part, too difficult.

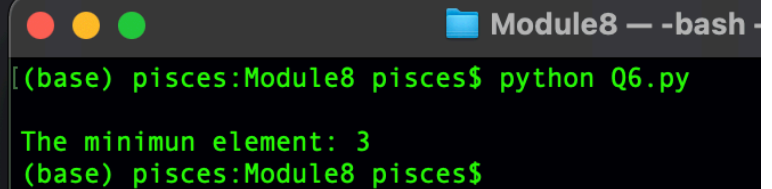
6. My Answer: $O(\log n)$

My reason:

Because Left-Leaning Red/Black tree (LLRB) belong to a Binary Search Tree (BST), so we can find the minimum element by looking for the most left node in the tree. So I used a recursive method to find the minimum element in the left node, and return the result. In conclusion, the time complexity of find minimum element is $O(\log n)$.

My Python Code:

```
1 # Hsuan-You Lin Module 8 Problem Set Question 6.
2
3 class LLRB_Node():
4     def __init__(self, val):
5         self.val = val
6         self.left = None
7         self.right = None
8
9     def insert(node, num):
10
11         if node is None:
12             return (LLRB_Node(num))
13
14         else:
15             # 2. Otherwise, recur down the tree
16             if num <= node.val:
17                 node.left = LLRB_Node.insert(node.left, num)
18             else:
19                 node.right = LLRB_Node.insert(node.right, num)
20
21             # Return the (unchanged) node pointer
22             return node
23
24     def Find_Minimum(root):
25         while root.left is not None:
26             root = root.left
27         return root.val
28
29
30 if __name__ == "__main__":
31     root = None
32     root = LLRB_Node.insert(root, 3)
33     LLRB_Node.insert(root, 9)
34     LLRB_Node.insert(root, 10)
35     LLRB_Node.insert(root, 4)
36     LLRB_Node.insert(root, 6)
37     LLRB_Node.insert(root, 5)
38
39     print("\nThe minimum element: %d" % (LLRB_Node.Find_Minimum(root)))
40
```



```
Module8 - bash
(base) pisces:Module8 pisces$ python Q6.py
The minimum element: 3
(base) pisces:Module8 pisces$
```

7. My Answer: $O(n)$

My reason:

In this case, we could use Morris Inorder Traversal to count the number of the nodes in the tree (which is LLRB_Node.Count_Nodes function in my Python code), then easily find the median element by returns the average of the numbers in the result list. Therefore, the overall time complexity of find median element is $O(n)$, because it's based on Inorder Traversal algorithm.

My Python Code:

```
Q7 > Find_Median

41 def Find_Median(root):
42     if root is None:
43         return 0
44     count = LLRB_Node.Count_Nodes(root)
45     n = 0
46
47     while (root != None):
48         if (root.left == None):
49             n += 1
50             if (count % 2 != 0 and n == (count + 1) // 2):
51                 return prev.val
52             elif (count % 2 == 0 and n == (count // 2) + 1):
53                 return (prev.val + root.val) // 2
54             root = root.right
55         else:
56             pre = root.left
57             while (pre.right != None and pre.right != root):
58                 pre = pre.right
59             if (pre.right == None):
60                 pre.right = root
61                 root = root.left
62             else:
63                 pre.right = None
64                 prev = pre
65                 n += 1
66                 if (count % 2 != 0 and n == (count + 1) // 2):
67                     return root.val
68
69                 elif (count % 2 == 0 and n == (count // 2) + 1):
70                     return (prev.val + root.val) // 2
71                 n = root.right
72
73 if __name__ == "__main__":
74     root = LLRB_Node(3)
75     LLRB_Node.insert(root, 9)
76     LLRB_Node.insert(root, 10)
77     LLRB_Node.insert(root, 4)
78     LLRB_Node.insert(root, 6)
79     LLRB_Node.insert(root, 5)
80     LLRB_Node.insert(root, 12)
81     print("\nThe median element: %d" % (LLRB_Node.Find_Median(root)))
```

```
Module8 — -bash — 66x7
(base) pisces:Module8 pisces$ python Q7.py
The median element: 6
(base) pisces:Module8 pisces$
```

8.1 My final hash table:

0	14
1	20
2	
3	
4	16, 5
5	88, 11
6	94, 39
7	12, 34, 23
8	
9	
10	

8.2 My Answer:

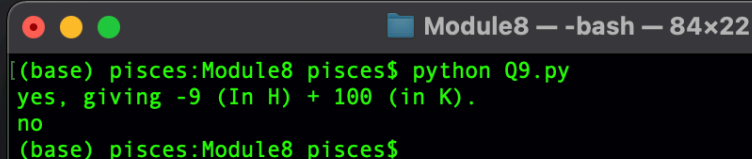
The final hash table is shown in below by using linear probes, it will search start from the beginning location of the hash.

0	14
1	39
2	20
3	5
4	16
5	88
6	94
7	12
8	34
9	23
10	11

9. My Python Code & explain:

Whether an element is inside the set will have $O(1)$ expected time, which means this algorithm have $O(n)$ expected time.

```
1 # Hsuan-You Lin Module 8 Problem Set Question 9.
2 def Q9(H, J, K):
3     HashMap = set(H)
4     for i in J:
5         if K - i in HashMap:
6             return "yes, giving -9 (In H) + 100 (in K)."
7     return "no"
8
9 if __name__ == "__main__":
10     H = [4, 5, 7, -1, -9, 22]
11     J = [0, -5, 10, 44, 100, -12]
12     K = 91
13     print(Q9(H, J, K))
14     K = 92
15     print(Q9(H, J, K))
16
```



```
(base) pisces:Module8 pisces$ python Q9.py
yes, giving -9 (In H) + 100 (in K).
no
(base) pisces:Module8 pisces$
```

10.

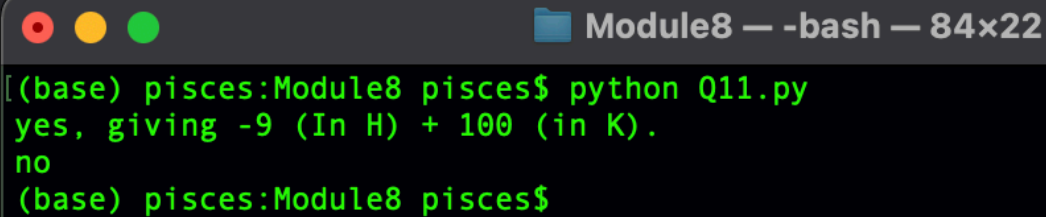
(a) My Answer: The worst-case complexity of my problem 9 algorithm is $O(n)$.

(b) My Answer: When the program traverses all of the elements it will becomes linear search, so the worst-case complexity will have $O(n)$.

11. My Python Code & explain:

The difference between problem 9 and problem 11 is problem 11 need to created an empty set, then during each iteration in the for-loop, there must exists an previously element and their sum is K, Thus, the complexity of my Python code is $O(n)$.

```
1 # Hsuan-You Lin Module 8 Problem Set Question 11.
2 def Q11(H, K):
3     HashMap = set()
4     for i in H:
5         if K - i in HashMap:
6             return "yes, giving -9 (In H) + 100 (in K)."
7         HashMap.add(i)
8     return "no"
9
10 if __name__ == "__main__":
11     H = [4, 5, 7, -1, -9, 22]
12     K = -5
13     print(Q11(H, K))
14     K = 7
15     print(Q11(H, K))
16
```



```
Module8 — -bash — 84x22
(base) pisces:Module8 pisces$ python Q11.py
yes, giving -9 (In H) + 100 (in K).
no
(base) pisces:Module8 pisces$
```