CS721 Exam - 2

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

1. A binary search algorithm aims to find a target value within a sorted array by splitting the sorted input array into two subsets of almost equal size and recursively searches in one of these two subsets after making one comparison. Consider the following two modifications of binary search algorithm.

a) Consider the modified binary search algorithm (called ternary search) so that it splits the sorted array not into two subsets of almost-equal sizes, but into three subsets of sizes approximately one-third and then recursively searches in one of these three subsets after doing appropriate comparisons. If this ternary search algorithm needs to recursively search in one of these three subsets of approximately one-third size, how many comparisons need to be made in each step? Write down the recurrence for this ternary search algorithm and find the asymptotic runtime of this algorithm.

Answer:

Like binary search instead of splitting array in to 2 subsets, here we split the array into 3 subsets. As we have 3 sub array's we need maximum of 2 comparisons to identify in which subarray the element is present. The comparisons we have are one with maximum of A1 that's last element in A1, if element is greater than that we have another comparison with largest element in A2 and finally we will get know in which subarray we need to perform search.

By this we can define the recurrence relation as: -

$T(n) = T(n/3) + 2$

$T(n) = T(n/3^2) + 2*2$

$T(n) = T(n/3^3) + 3*2$

After doing for k times it becomes

$T(n) = T(n/3^k) + k*2$

As, $n/3^k = 1$

$3^k = n$

$k = \log_3 n$

Therefore, $T(n) = T(1) + 2*\log_3 n$

As T(1) is constant we can say, time complexity as **T(n) is O(log3 n)**

b) Consider another variation of the binary search algorithm so that it splits an input sorted array into two subsets of sizes approximately one-third and two-thirds respectively and then recursively searches in one of these two subsets. Write down the recurrence for this search algorithm when (i)

the search always selects the smaller subset and (ii) the search always selects the larger subset. Find the asymptotic runtime of this algorithm for both these cases.

Answer:

Like the above problem here array A of size n is divided into two subarrays A1 and A2 with sizes n/3 and 2n/3 respectively. To perform search, we need only one comparison which is with largest element in A1 to identify in which subarray the element is present.

(i) The search when we select the smaller subset A1 becomes as: -

T(n) = T(n/3) + 1

T(n) = T(n/3^2) + 2*1

T(n) = T(n/3^3) + 3*1

After doing for k times it becomes

T(n) = T(n/3^k) + k*1

As, n/3^k = 1

3^k = n

k = log3 n

Therefore, T(n) = T(1) + log3 n

As T(1) is constant we can say, time complexity as **T(n) is O(log3 n)**

(ii) The search when we select the bigger subset A2 becomes: -

T(n) = T(2n/3) + 1

T(n) = T((2/3)^2*n) + 2*1

T(n) = T((2/3)^3*n) + 3*1

After doing for k times it becomes

T(n) = T((2/3)^k*n) + k*1

As, (2/3)^k*n = 1

(2/3)^k = n

k = log3/2 n

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

Therefore, T(n) = T(1) + log3/2 n

As T(1) is constant we can say, time complexity as **T(n) is O(log3/2 n)**

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.

Answer:

A red-black tree must satisfy these properties:

1. The root is always black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. Every node is either red or black

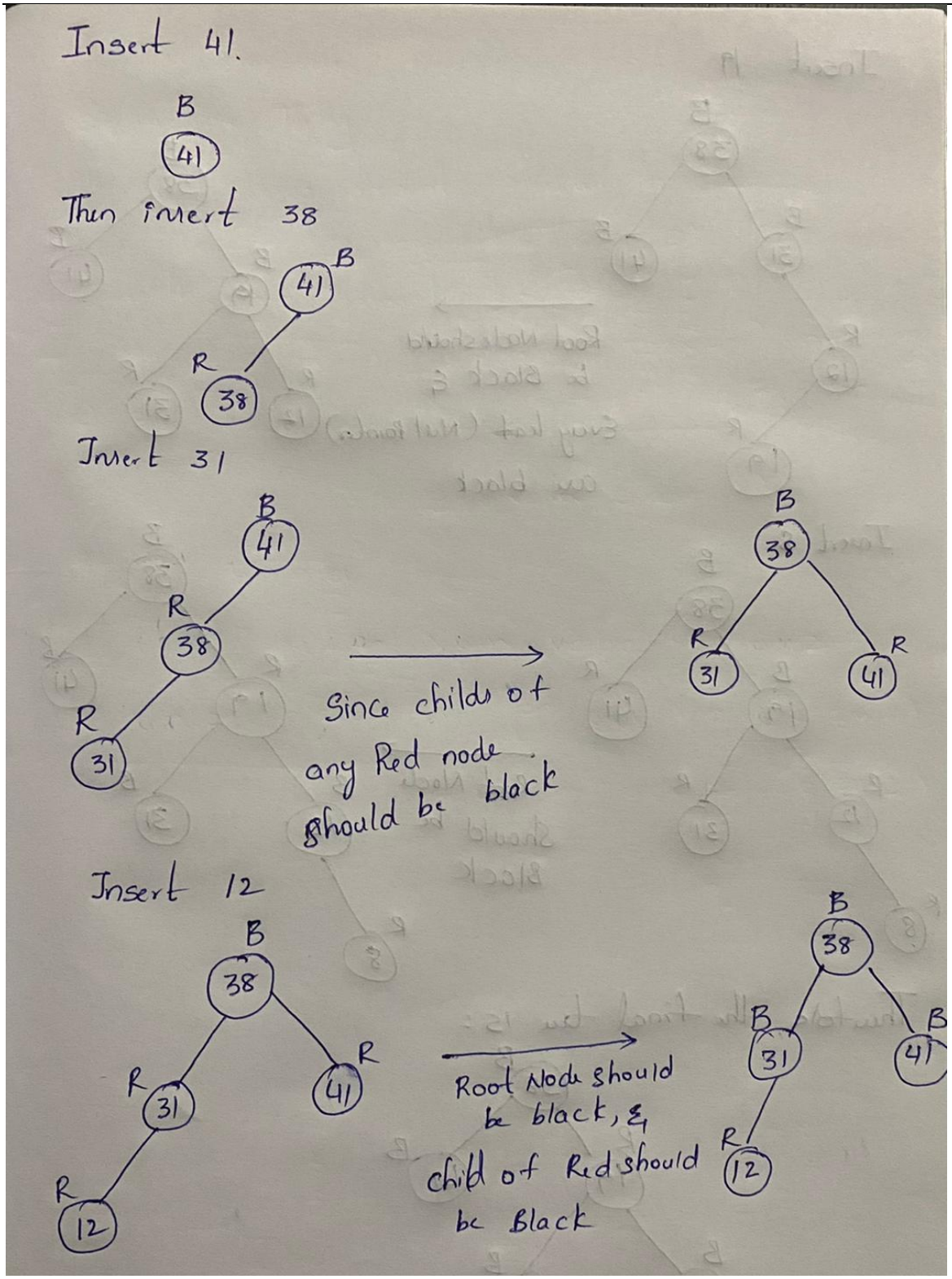
B – denotes black

R – denotes Red

By using above properties insertion of keys would be as follows:

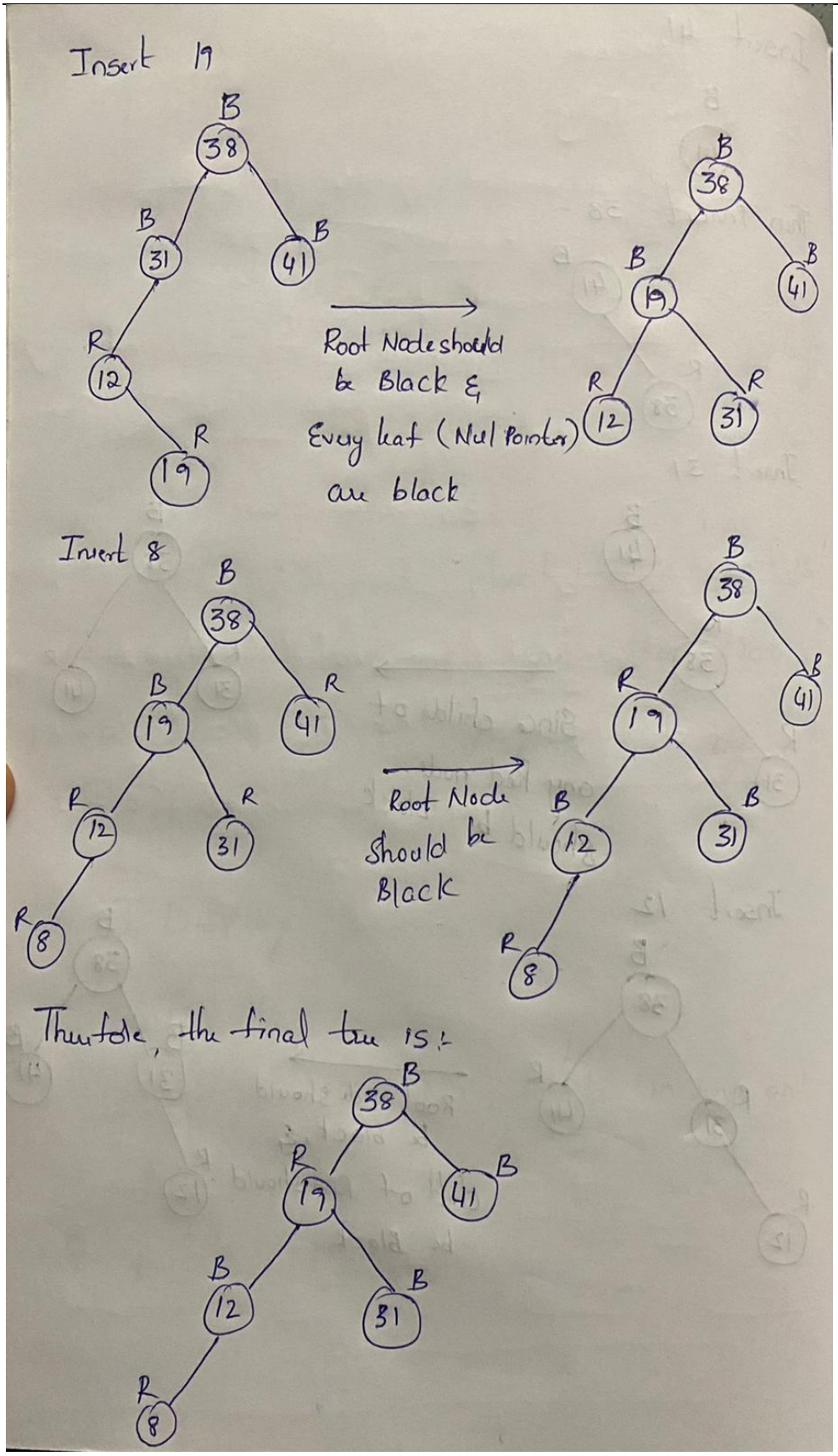
CS721 Exam - 2

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624



CS721 Exam - 2

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624



CS721 Exam - 2

Name: Jogesh Venkata Surya Prakash Devathi
WSU ID: n659a624

3. Modify MEMOIZED-CUT-ROD to return not only the value but the actual solution, too.

Answer:

```
def memoized_cut_rod_sub(p, n, r, s):
    if r[n] >= 0:
      return r[n]
    r[n] = 0
    for i in range(1, n + 1):
        ret = p[i] + memoized_cut_rod_sub(p, n - i, r, s)
        if r[n] < ret:
            r[n] = ret
            s[n] = i
    return r[n]


def memoized_cut_rod(p, n):
    r = [-1 for _ in xrange(n + 1)]
    s = [i for i in xrange(n + 1)]
    memoized_cut_rod_sub(p, n, r, s)
    r = r[n]
    subs = []
    while n > 0:
        subs.append(s[n])
        n -=s[n]
    return r, subs
```