Algorithms and Problem Solving (15B11CI411)

T1 examination Even 2021

Q1. **[CO1] [1 mark]** How many times L and k would be compared and how many times function F1() will be called in the following code?

```
for (int L=1; L<k; L=L*3)
        F1();
```

Soln: Half mark for each case (0.5 x 2 = 1 mark)

Case 1 (K>1): L and K would be compared $\lceil \log_3 K \rceil + 1$ times and F1 will be called $\lceil \log_3 K \rceil$ times for all the values of K>1.

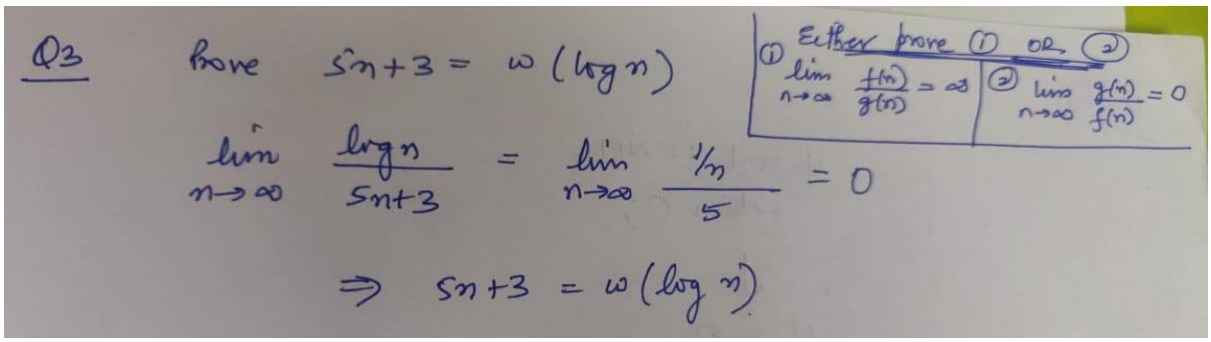
Case 2 (K<=1): Otherwise, L and K would be compared only once with no call to F1 function.

Q2. **[CO1] [1 mark]** Provide a recurrence relation for the execution time of the following function:

```
F2(K)
{       If(K>1)
                {
                  j= F2(K-1);
                  return j++;
                }
        else
                return 1;
}
```

Solution: T(K) = T(K-1) + Const. (0 or 1 mark, no partial)

Q3. **[CO1] [1 mark]** Prove that: 5n+3 = ω(log(n)). (0 or 1 mark, no partial)



Q4. **[CO1] [1 mark]** Solve the given recurrence using Master Theorem.

$T(n) = 9T(n/3) + n^2$, (T(1) = 1)

Solution (0 or 1 mark, no partial)

$T(n) = 9T(n/3) + n^2$

A=9, b=3, $f(n)= n^2$

$n^{\log_b a} = n^{\log_3 9}$; $n^2 = f(n)$ This case 2

So $T(n) = O(n^{\log_b a} \log_b n) = O(n^2 \log_3 n)$

Q5. **[CO3] [1 mark]** Amazon has decided to use counting sort algorithm to sort millions of products by their 15-digit serial numbers. Is it the right decision? Justify your answer.

Solution: not right. As count sort performs sorting based on number of digits like one place, tens place, hundredth place etc it complexity will be very high. So it's not suitable. (0 or 1 mark, no partial).

Q6. **[CO1] [1 mark]** Professor Sujata uses the following algorithm for merging k sorted lists, each having n/k elements. She takes the first list and merges it with the second list using a linear-time algorithm for merging two sorted lists, such as the merging algorithm used in merge sort. Then, she merges the resulting list of 2n/k elements with the third list, merges the list of 3n/k elements that results with the fourth list, and so forth, until she ends up with a single sorted list of all elements. The worst-case running time of the professor's algorithm in terms of n and k is?

Solution (0 or 1 mark, no partial).

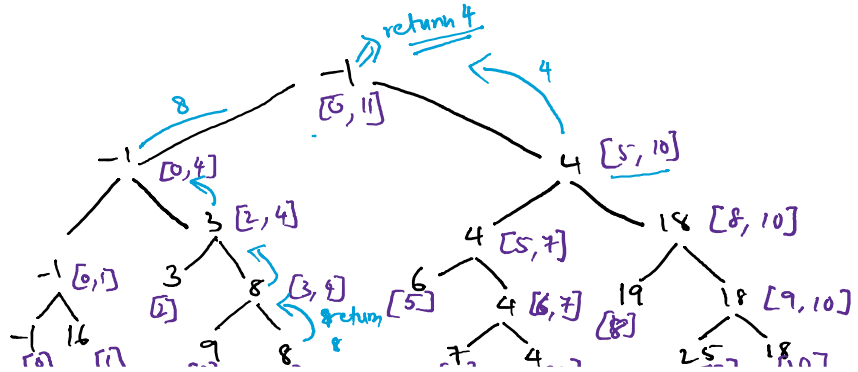
Solution: Merging the first two lists,each of n/k elements, takes $2n/k$ time. Merging the resulting $2n/k$ elements with the third list of n/k elements takes $3n/k$ time, and so on. Thus for a total of k list, we have:

$$\begin{aligned} Time &= \frac{2n}{k} + \frac{3n}{k} + \cdots + \frac{kn}{k} \\ &= \sum_{i=2}^{k} \frac{in}{k} \\ &= \frac{n}{k} \sum_{i=2}^{k} i \\ &= \frac{n}{k} \frac{(k+2)(k-1)}{2} \\ &= \theta(nk) \end{aligned}$$

Q7. **[CO2] [1 mark]** You are given a set of values as shown: A= [-1,16,3,9,8,6,7,4,19,25,18]

Suppose you constructed the segment tree for minimum in a range query in A. After constructing the required segment tree, how many numbers of comparisons will be required if you want to find what is the minimum in the range of [4,10].

Solution: 6 (0 or 1 mark, no partial).



Q8. **[CO2] [1 mark]** Suppose you created a segment tree for finding sum in a given range queries and inserted the following elements in it.

-1,16,3,9,8,6,7,4,19,25,18.

What will be the root value of the created segment tree? (0 or 1 mark, no partial).

Solution : 114 (it is simply the sum of all the elements of the array as root represents the entire range).

Q9. **[CO2] [1 mark]** Parent and children of a node in the segment tree can be found at what index value if it is implemented using static array and how much size of memory would be allocated for such segment tree?

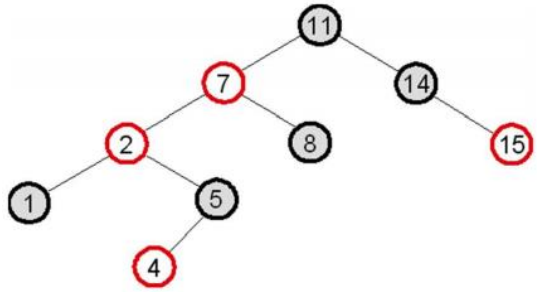
Parent and child node index: (0.5 mark) : An array representation of tree is used to represent Segment Trees. For each node at index i, the left child is at index 2*i+1, right child at 2*i+2 and the parent is at (i-1)/2.

Segment tree size (0.5 mark) : **(any explanation is correct)**

If n is a power of 2, then there are no dummy nodes. So the size of the segment tree is 2n-1 (n leaf nodes and n-1) internal nodes. If n is not a power of 2, then the size of the tree will be 2*x – 1 where x is the smallest power of 2 greater than n. For example, when n = 10, then size of array representing segment tree is 2*16-1 = 31.

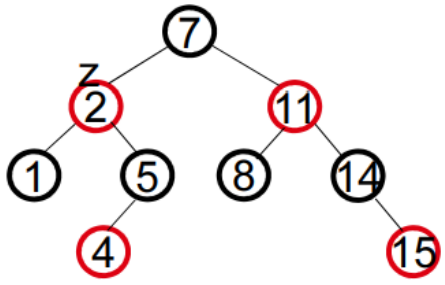
An alternate explanation for size is based on height. Height of the segment tree will be $\lceil \log_2 n \rceil$. Since the tree is represented using array and relation between parent and child indexes must be maintained, size of memory allocated for segment tree will be $2 * 2^{\lceil \log_2 n \rceil} - 1$.

Q10. **[CO2] [1 mark]** What red-black tree property is violated in the tree below? Show the tree after you restore the red-black tree property in this case.



Solution: (0 or 1 mark, no partial).

Property violated: if a node is red, both its children are black – Fixup: color 7 black, 11 red, then right-rotate around 11



Q11. **[CO3] [2 marks]** You are given two databases, where each database contains n numerical values. Further assume that the values are distinct and sorted. We are also given a function f() to access the numerical values from any one of the two databases. For the given database and value k, Function f() outputs the k^{th} smallest value that the given database contains. The function f() takes O(1). We would like to determine the n^{th} smallest value, or in other words median, out of these 2n values. Give an algorithm which finds the median value using O(logn) queries only.

Solution: Suppose we first query each database for its median—that is, we query for element n/2. Let m1 be the median from database 1 and let m2 be the median from database 2. Suppose, without loss of generality, that m1 < m2. We know that n/2 elements in database 1 are less than or equal to m1, and therefore must also be less than m2. We don't know how the remaining n/2 elements in database 1 are ordered with respect to m2. Similarly, we know that n/2 elements in database 2 are greater than m2 and therefore also greater than m1, but we don't know how the remaining n/2 elements in database 2 are ordered with respect to m1. We can say for sure that the median of all 2n elements lies somewhere in either the largest n/2 elements in database 1, or the smallest n/2 elements in database 2. In fact, we can say that the median of all 2n elements is the median of the largest n/2 elements in database 1 and the smallest n/2 elements in database 2, since we have removed from consideration the same number of elements in both databases. So we can view these subsets as two "smaller databases," and recursively find the median of these smaller databases.

Here's some pseudocode:

```
FindMedian(D1, d1^ℓ, d1^h, D2, d2^ℓ, d2^h)
    if d1^ℓ == d1^h
        m1 = query(D1, 1)
        m2 = query(D2, 1)
        return min(m1, m2)
    m1 = query(D1,(d1^ℓ + d1^h − 1)/2)
    m2 = query(D2,(d2^ℓ + d2^h − 1)/2)
    if m1 < m2
        return FindMedian(D1, m1+1, d1^h, D2, d2^ℓ, m2)
    else
        return FindMedian(D1, d1^ℓ, m1, D2, m2+1, d2^h)
```

If we imagine the elements of each database as being stored in an array (indexed starting at 1) in increasing order, d_i^ℓ and d_i^h represent the subarray to which we are restricting our attention, and `query(D1, j)` looks up the jth smallest element in the database (i.e., the element in position j in our imagined array). Our initial call is to `FindMedian(D1, 1, n, D2, 1, n)`.

O(log n) solution : 2 marks

Correct solution with more complexity than O(logn): 0.5 mark

Incorrect solution : 0 mark

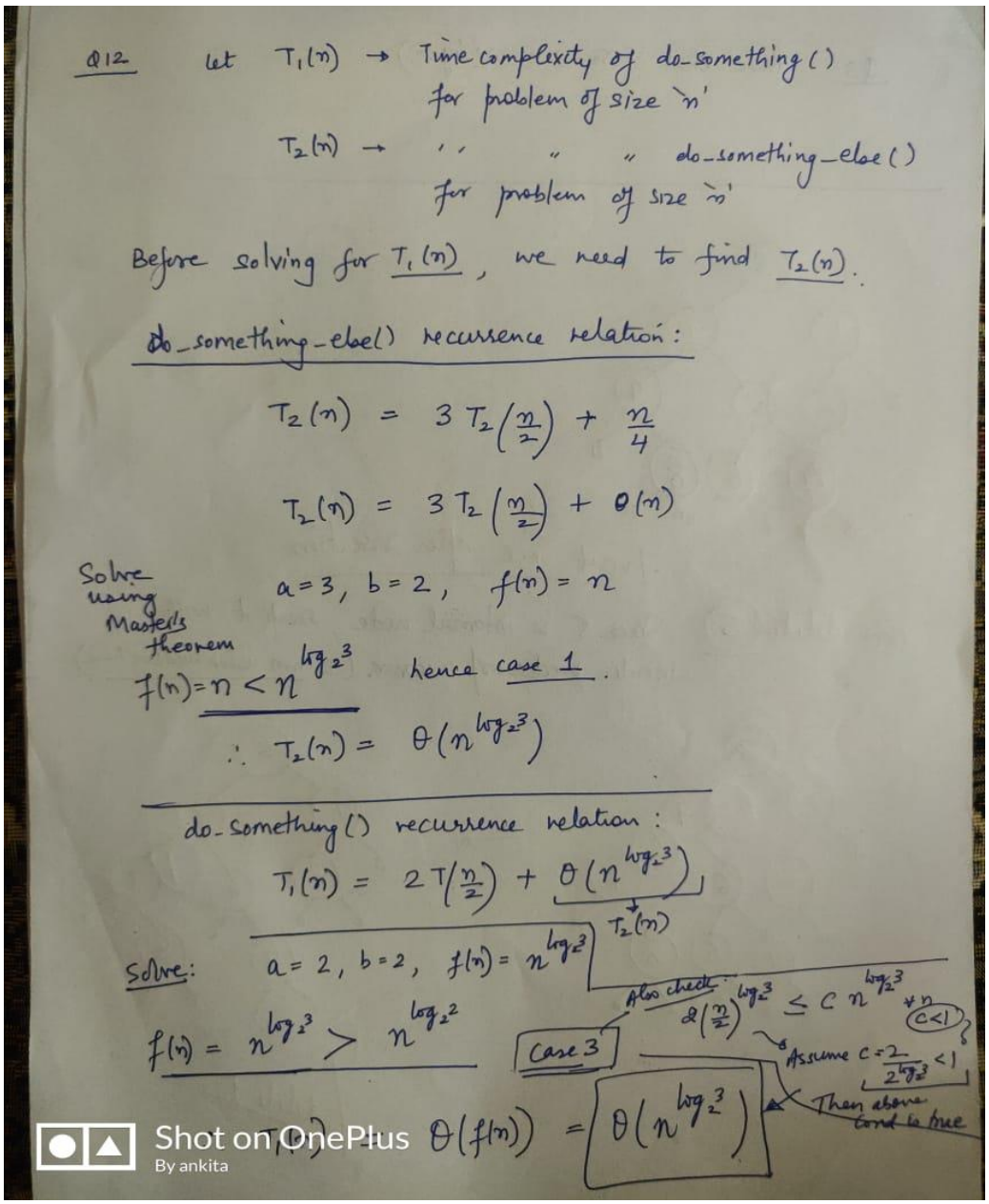
Q12. **[CO1] [2 marks]** Write the recurrence relation and analyse the time complexity (in terms of Big O) of recursive algorithm **do_something()** given below.

```
void do_something (A[1..n])
{
   If (n>1)
      do_something (A[1…n/2]);
      do_something (A[n/2+1…n]);
      do_something_else(A[1..n]);
}
```

```
void do_something_else(A[1..n])
{
     If(n==2)
       If(A[1]>A[2])
          swap A[1] and A[2];
     else
     {     for i= 1 to n/4
                swap A[i + n/4] and A[i + n/2]
           // end of for loop

           do_something_else (A [ 1.. n/2]);
           do_something_else (A [ n/2 +1 … n]);
           do_something_else (A [n/4 +1 …3n/4]);
     }
}
```

1 mark for do_something_else() complexity + 1 mark for do_something() complexity.



Q13. **[CO4] [3 marks]** JIIT wants to prepare an interesting application for JIIT Alumni. Its functionality is explained as follows: The application requires to store a data set "D" of all JIIT faculty till date (from the beginning) with following fields: Faculty_ID, Faculty_name, JIIT joining year and JIIT leaving year (this field is equal to current year if faculty is still working with JIIT). This data set D needs to be stored in such a data structure such that thousands of Alumni students can perform following types of queries/searches on this data set in an efficient manner.

Query 1: An alumni wants to know who all faculty were there in JIIT during his stay at JIIT. So, he enters his batch like 2013-2017 batch and application returns all JIIT faculty details who worked with JIIT in that duration.

Query 2: An alumni wants to know that during his particular year (second year or third year etc) at JIIT, who all faculty were there in JIIT. So, he enters a year like 2015 and application returns all JIIT faculty details who worked with JIIT in 2015.

a) **[0.5 mark]** Propose an appropriate Data structure to store faculty dataset D so that above queries are efficiently processed. Write the Node structure of your proposed Data structure.
b) **[1.5 mark]** Propose an efficient algorithm for Query 2.
c) **[1 mark]** Show the complexity analysis of your proposed algorithm.

Solution:

a) **(0 or 0.5 mark)** Interval tree is suitable for the given scenario with node structure as follows:

```
struct Interval
{
    int low, high;
};
struct ITNode
{
    String Fac_name;
    int faculty_id;
    Interval *i;
    int max;
    char color;
    ITNode* parent;
    ITNode *left, *right;
};
```

Note:

➔ Low and high represent JIIT joining and leaving year respectively.
➔ I have used RB Tree as augmenting tree here, hence included color attribute. If students use AVL then they should include Balance factor as the attribute.

Full marks (0.5) only if all parameters (including RB/AVL related) are given…. Otherwise award 0

b) Algorithm for query 2 (this is not inorder/preorder traversal as all the nodes are not visited).
This solution is O(log n + k) solution. : 1.5 mark
If O(n) solution is proposed: then give 0.5 mark.

```
bool findAlloverlap_point(ITNode *root, int t)
{
    if (root == NULL)
        return 0;

    bool flag = 0;

    if (root->left != NULL && root->left->max >= t)
        flag = findAlloverlap_point(root->left, t);
    else
        findAlloverlap_point(root->right,t);

    if (flag)
    {
     flag = flag | findAlloverlap_point(root->right, t);
    }

    if (doOVerlap_point(*root->i, t))
{
    cout << root->i->low << " - " << root->i->high << "\n";
    flag = 1;
}
    return flag;
}
```

```
bool doOVerlap_point(Interval i1, int i2)
{
    if (i1.low <= i2 && i2 <= i1.high)
        return true;
    return false;
}
```

c) Complexity analysis:
0.5 mark for O(n) solution, if and only if its complexity analysis properly explained.
1 mark for O(log n + k) solution's complexity analysis properly explained.

We are rejecting one half of the tree based on the condition:
(root->left != NULL && root->left->max >= t)

If this condition is false, we do not go to left subtree and hence half tree is rejected. We only go to right side.

If this condition is true, we go to left subtree. On left side, there are two possibilities again:

- ➔ if we do not find any overlapping interval on left side(we make flag=0) and hence there is no need to go to right side also. (again half subtree is rejected).

- ➔ If we find any overlapping interval on left side , then we should go right as well (flag==1). In this case , half subtree is not rejected because there is possibility to get another overlapping interval on right side.

Hence , we will go to other half only when we have overlapping interval there. In best case , we are rejecting half subtree every time and going till height of the tree which is O($\log_2$ n).

In worst case, the complexity is proportional to the number of overlapping intervals.

Overall complexity : O(log n + k) where k is the number of overlapping intervals.

Q14. **[CO2] [3 marks]** You have been given a scenario below. Read the scenario carefully and apply the mentioned operations.

Scenario: Airport authority of India wants to store the details of flights (like flight number, source, destination etc) in a non-linear data structure and after that perform some queries like searching a flight, inserting a new flight, deleting/cancelling a flight. In the pandemic times, few people search for the flights (hence 'search a flight' query is less frequent). Whereas, a large number of flights get cancelled (deleted) and added (inserted) daily in the chosen data structure.

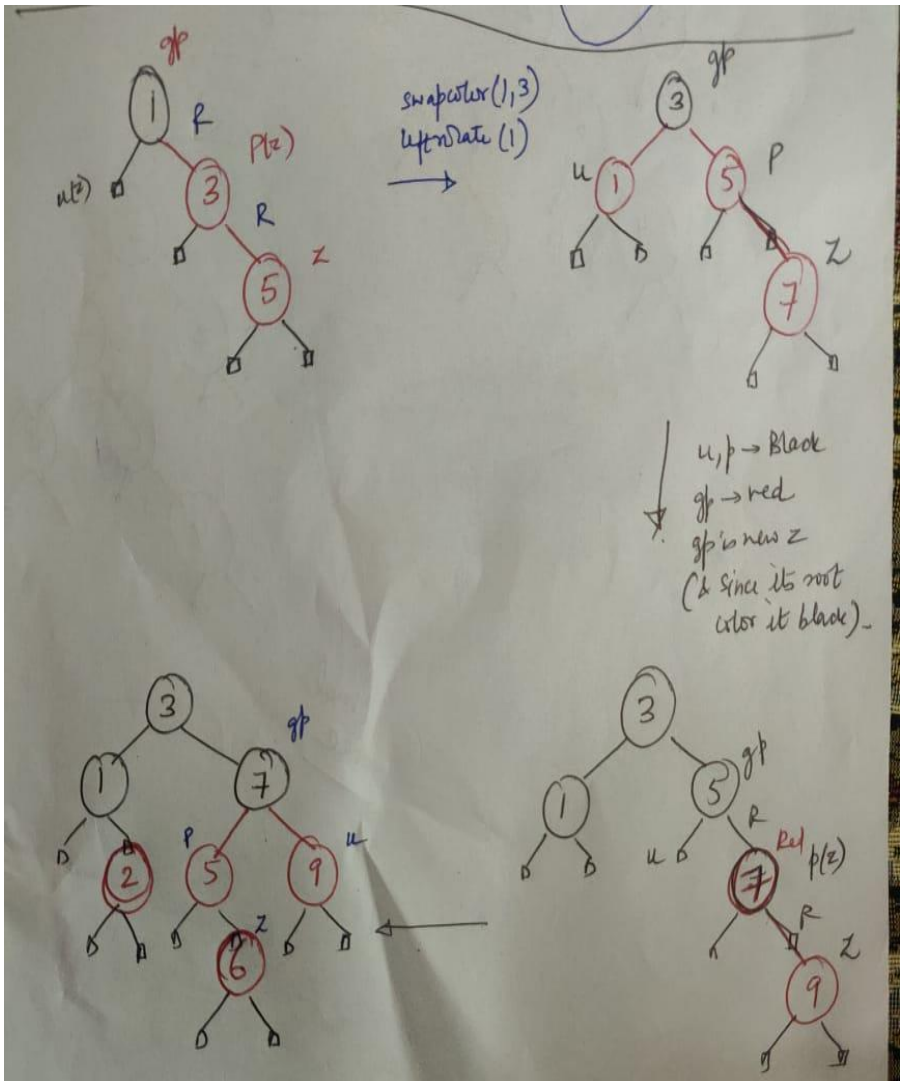
Which data structure would you suggest to store the flight details? For flight numbers given below, perform the following operations on chosen data structure in the given order (show all the steps).

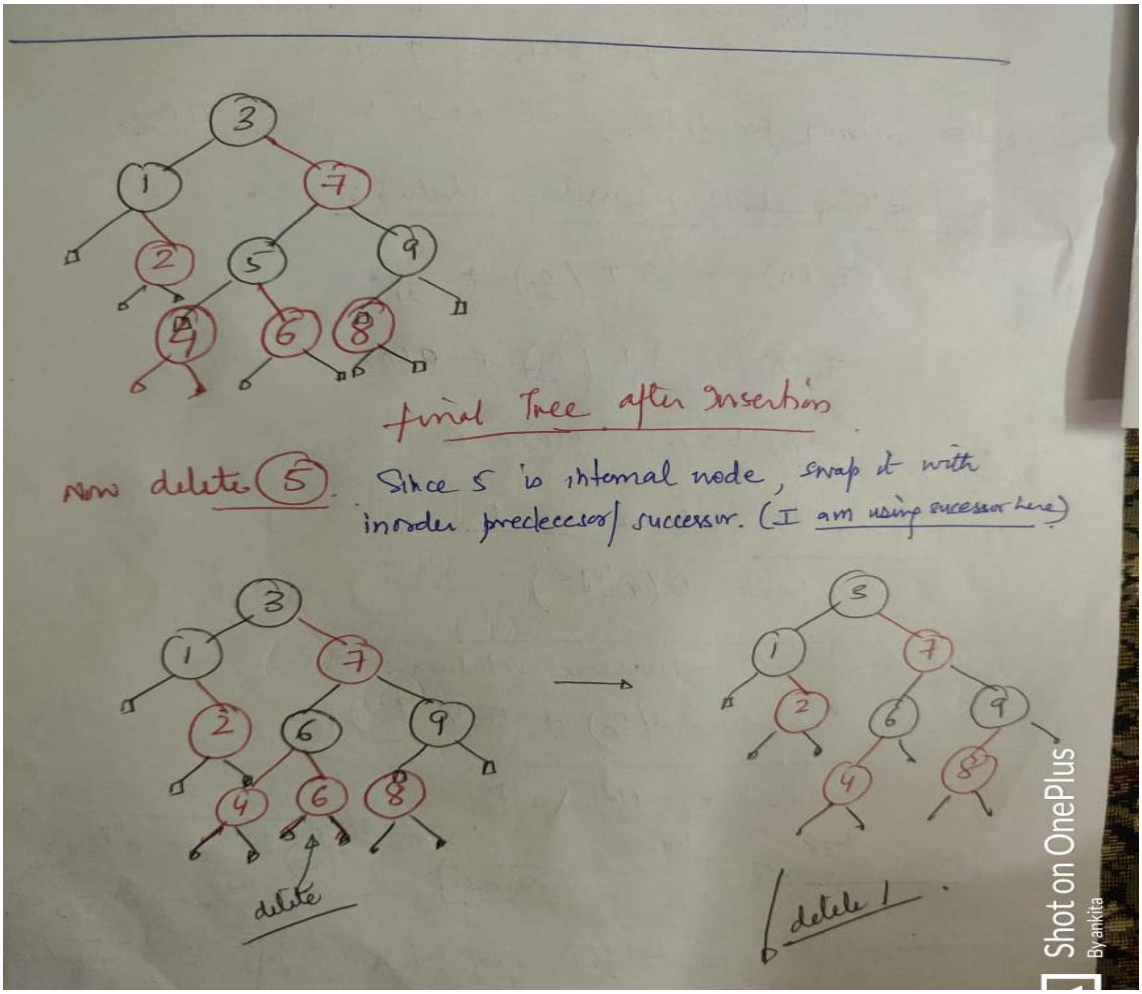
Insert: 1, 3, 5, 7, 9, 2, 6, 8, 4

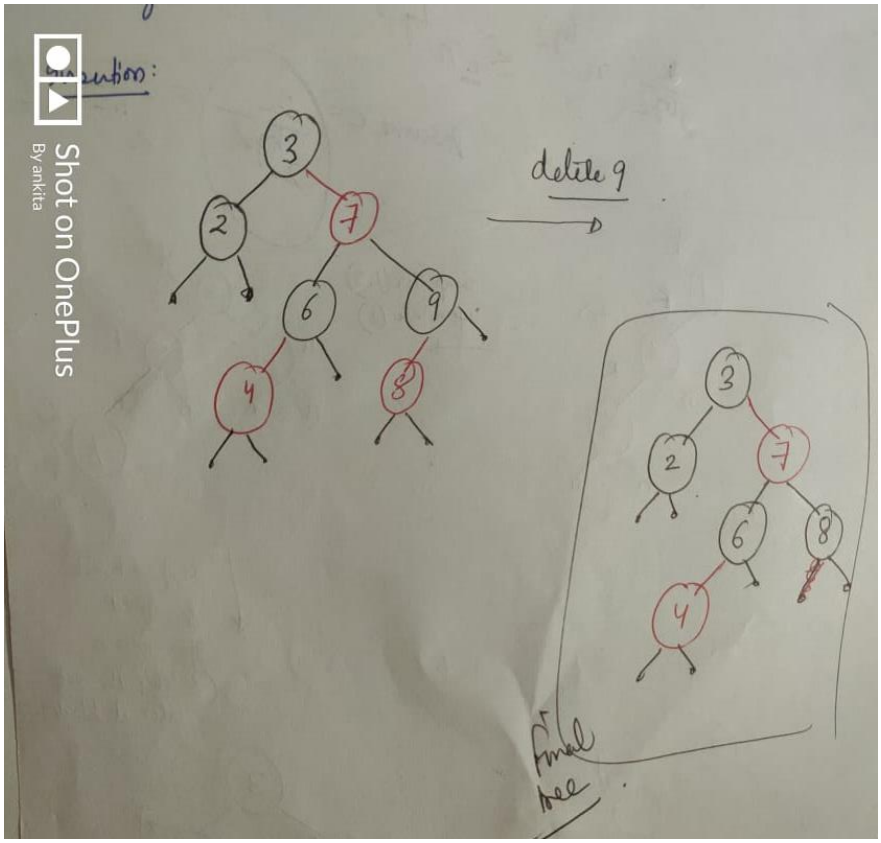
Delete: 5, 1, 9

Solution: Red Black Tree is the suitable data structure for given scenario.

Identifying RB tree is suitable - 0.5 mark **Insertions- 1.5 mark** **Deletions- 1 mark**




3
1
7
2
5
9
4
6
8
final Tree after insertion
now delete 5. Since 5 is internal node, swap it with inorder predecessor/ successor. (I am using successor here)
3
1
7
2
6
9
4
6
8
delete
5
1
7
2
6
9
4
8
delete 1
Shot on OnePlus
By ankita


Shot on OnePlus
By ankita
3
2
7
6
9
4
8
delete 9
3
2
7
6
8
4
final tree