Answer -1:

```
PruneAndSearch (A, low, high)

    if (low < high)
    {
        mid = (low + high) / 2;
        $m_1$ = (low + mid) / 2;
        $m_2$ = (mid+1+high) / 2;
        if(comparator(A[$m_1$] , A[$m_2$])
         {
            PruneAndSearch(A, low, $m_1$-1);
            PruneAndSearch(A, $m_1$+1, high);
             PruneAndSearch(A, mid+1, $m_2$-1);
             PruneAndSearch(A, $m_2$+1, high);
         }
        else if ($m_1$ < n and comparator(A[$m_1$] , A[$m_1$+1]) or (o < $m_1$ and comparator(A[$m_1$] , A[$m_1$-1]))
         {
                return A[$m_1$];
        }

       else
       {
                return A[$m_1$];
       }
  }
```

T(n) = T(n/2) + T(n/2) + k

As we are dividing the problem into half and solving each half time complexity reduces to two times of T(n/2) and a constant time to do the comparison.

$T(n) = 2.T(n/2) + k$

As per the master's theorem ➔ $a = 2$, $b = 2$, $f(n) = k.\, n^0$

$c = 0$

$$\begin{aligned} f(n) &= O(n^{\log_2 2 - \varepsilon}),\ \varepsilon = 1 \\ &= O(n^{1-1}) \\ &= O(n^0)\ \text{(True)} \end{aligned}$$

So, $T(n) = \theta(n)$

Answer - 2:
To Compute mingap, Redblack tree can be used by adding additional fields min,max and mingap to each node.

Values are computed using below. Mingap of leaf node is ∞

$$\min[x] = \begin{cases} \min[\text{left}[x]] \text{ if left child exists} \\ \text{key}[x] \text{ otherwise} \end{cases}$$

$$\max[x] = \begin{cases} \max[\text{right}[x]] \text{ if right child exists} \\ \text{key}[x] \text{ otherwise} \end{cases}$$

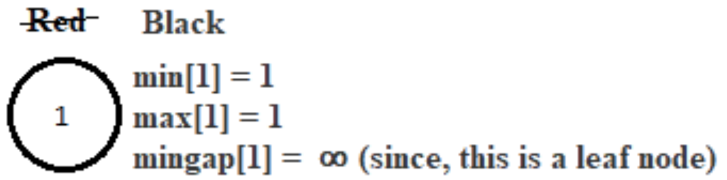
$$\text{mingap}[x] = \min \begin{cases} \text{mingap}[\text{left}[x]] \text{ if right child exists} \\ \text{key}[x] \text{ otherwise} \end{cases}$$

$$\text{mingap}[x] = \begin{cases} \text{mingap}\big[\text{left}[x]\big] \text{if right child exists} \\ \text{key}[x] \text{ otherwise} \\ \text{key}[x] - \text{max}\big[\text{left}[x]\big] \text{ if left} \\ \text{min}\big[\text{right}[x]\big] - \text{key}[x] \text{ if right} \end{cases}$$

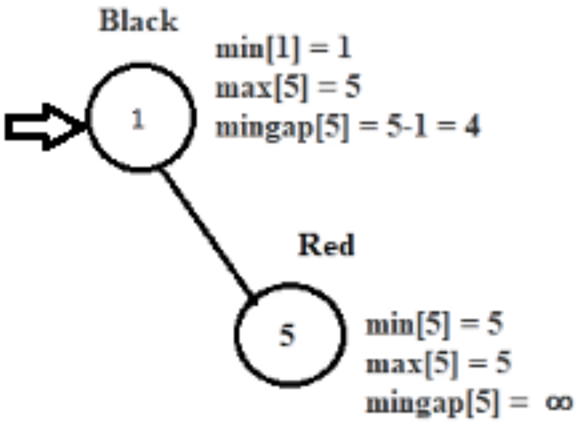
After each operation and Red Black Tree update min, max and mingap values from newly inserted/deleted to the root as long as there is change in the values.

Q = {1,5,9,15,18,22}

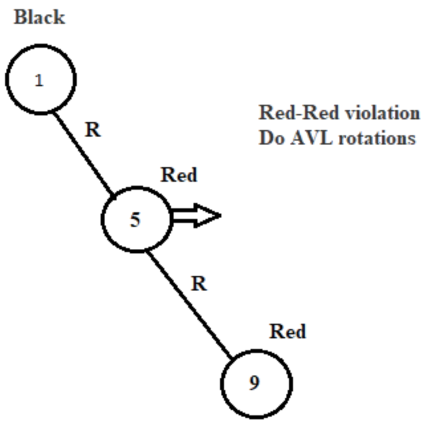
Insert 1 in Red black tree

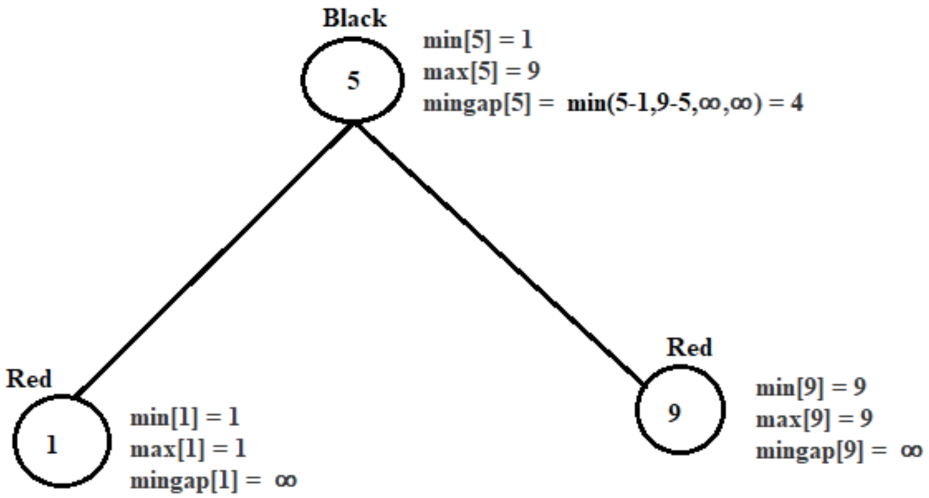


Insert 5 in Reb black tree

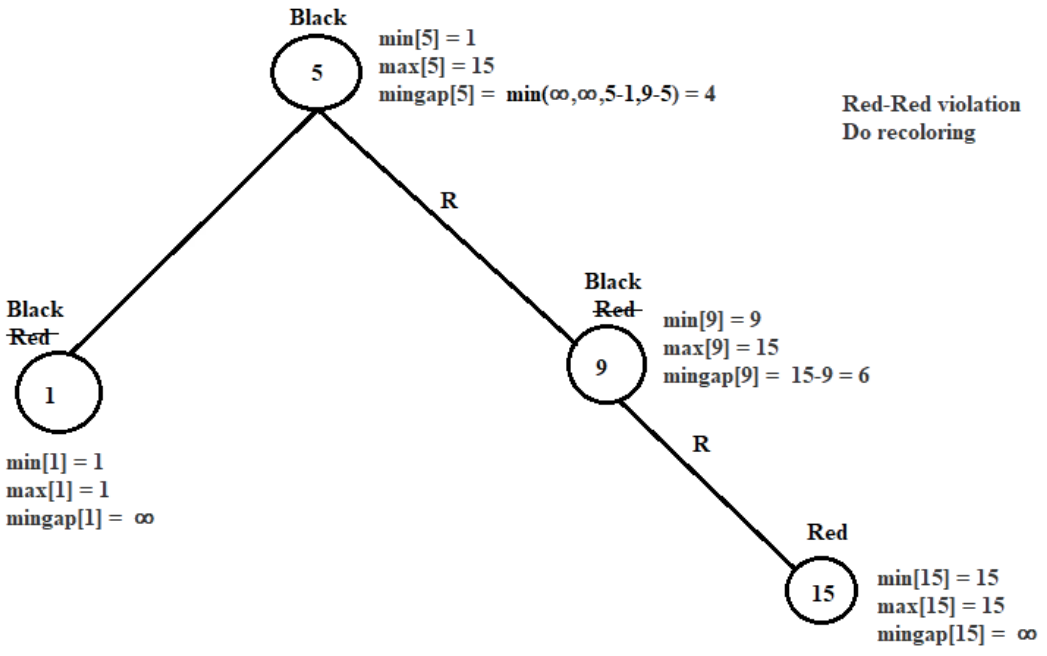


Insert 9 in Red black tree

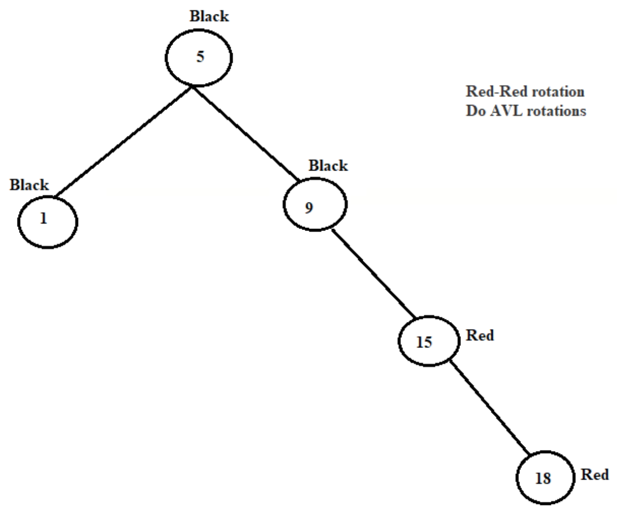


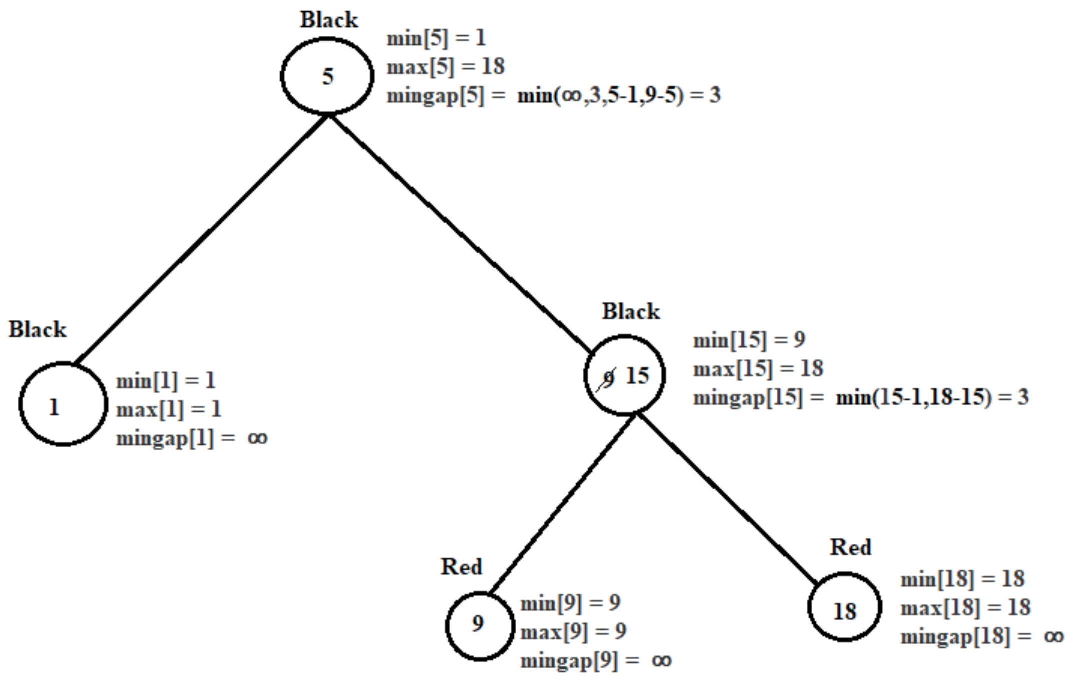


Insert 15 in Red black tree

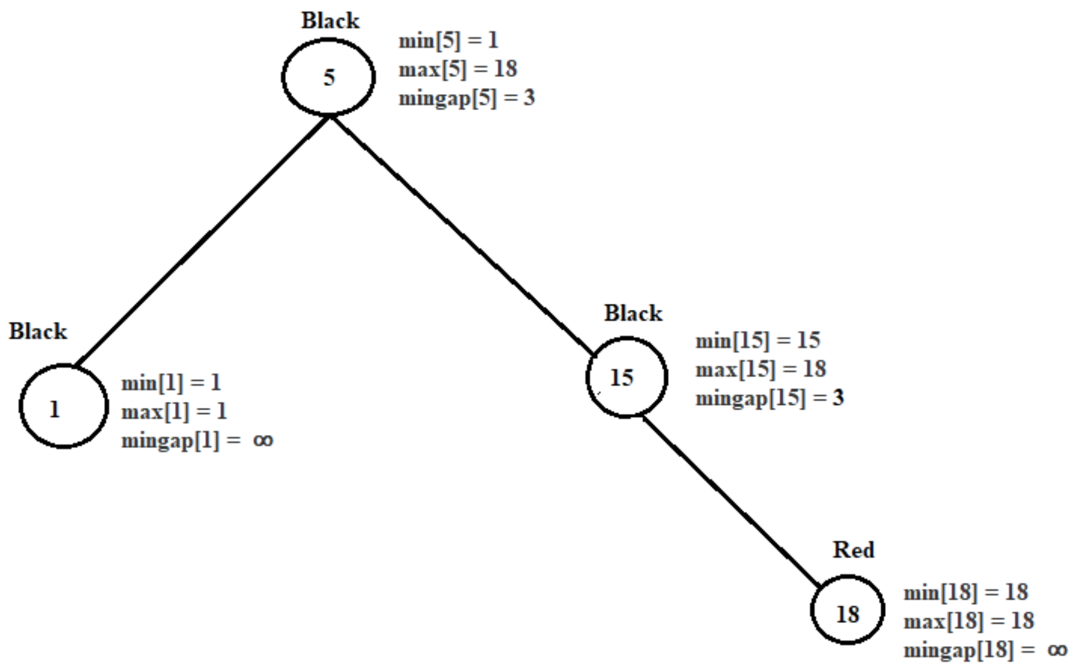


Insert 18 in Red black tree

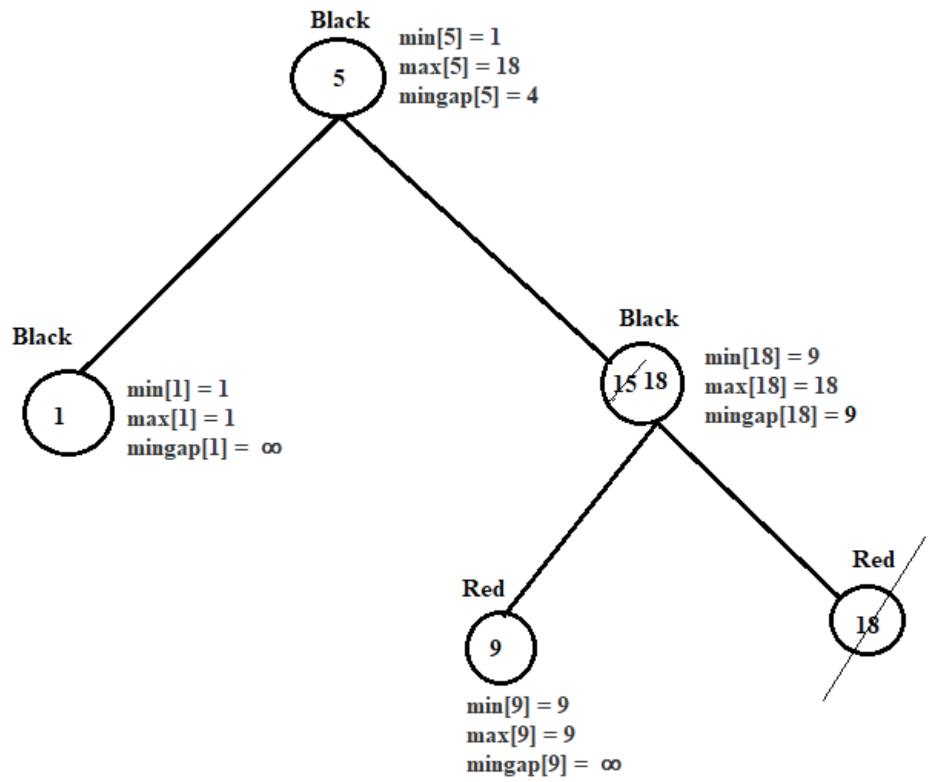



Black
5
min[5] = 1
max[5] = 18
mingap[5] = min(∞,3,5-1,9-5) = 3
Black
1
min[1] = 1
max[1] = 1
mingap[1] = ∞
Black
9 15
min[15] = 9
max[15] = 18
mingap[15] = min(15-1,18-15) = 3
Red
9
min[9] = 9
max[9] = 9
mingap[9] = ∞
Red
18
min[18] = 18
max[18] = 18
mingap[18] = ∞

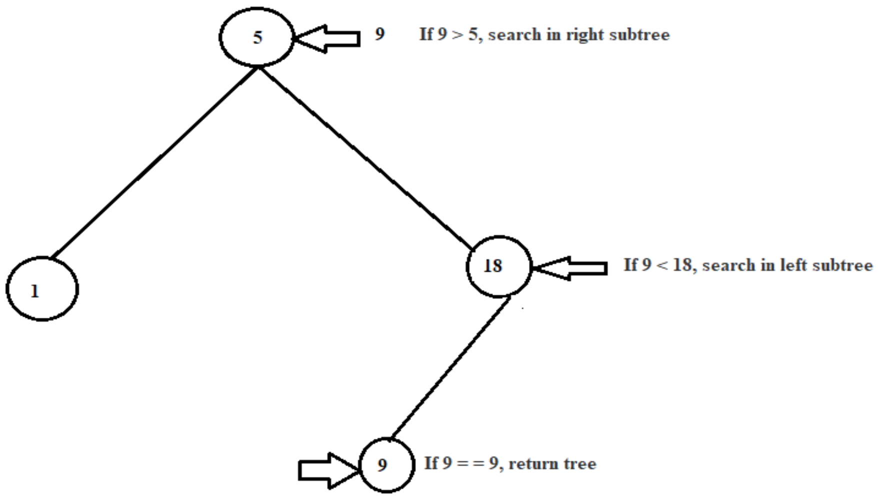
Delete 9


Black
5
min[5] = 1
max[5] = 18
mingap[5] = 3
Black
1
min[1] = 1
max[1] = 1
mingap[1] = ∞
Black
15
min[15] = 15
max[15] = 18
mingap[15] = 3
Red
18
min[18] = 18
max[18] = 18
mingap[18] = ∞

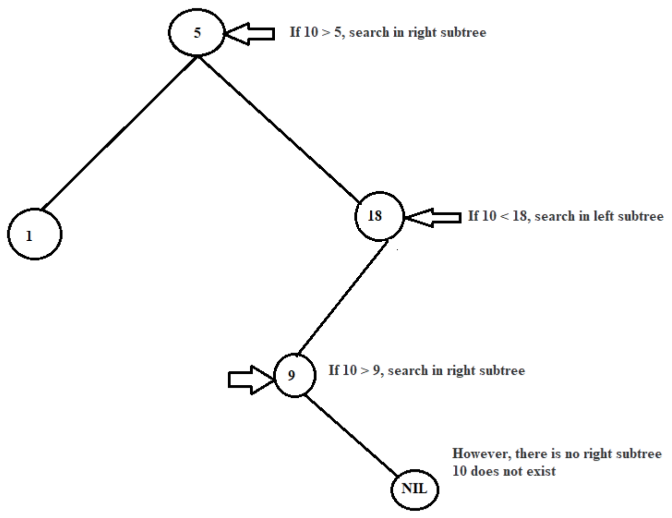
Red node is deleted. So, no recoloring/balancing is needed.



Search 9


5
9
If 9 > 5, search in right subtree
1
18
If 9 < 18, search in left subtree
9
If 9 = = 9, return tree

Search 10

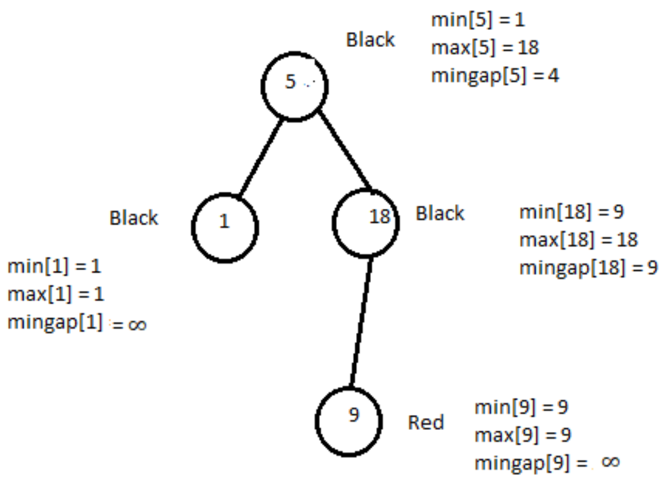

5
If 10 > 5, search in right subtree
1
18
If 10 < 18, search in left subtree
9
If 10 > 9, search in right subtree
However, there is no right subtree
10 does not exist
NIL

Time to insert, delete and search depends on the height of the tree and constant number of rotations of insert and delete and time to compute min, max value which also depends on height of tree.

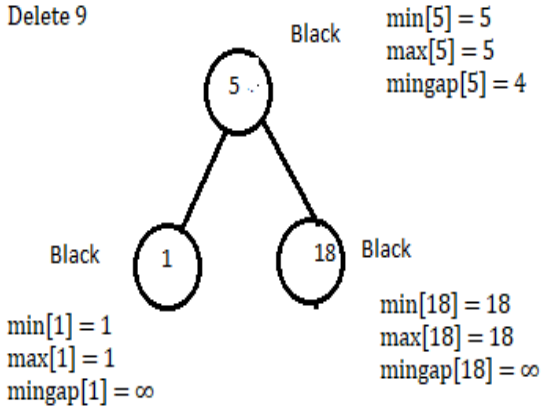
As red black tree is balanced, tree is O(h) ≅ O(logn).

mingap value is stored at root.

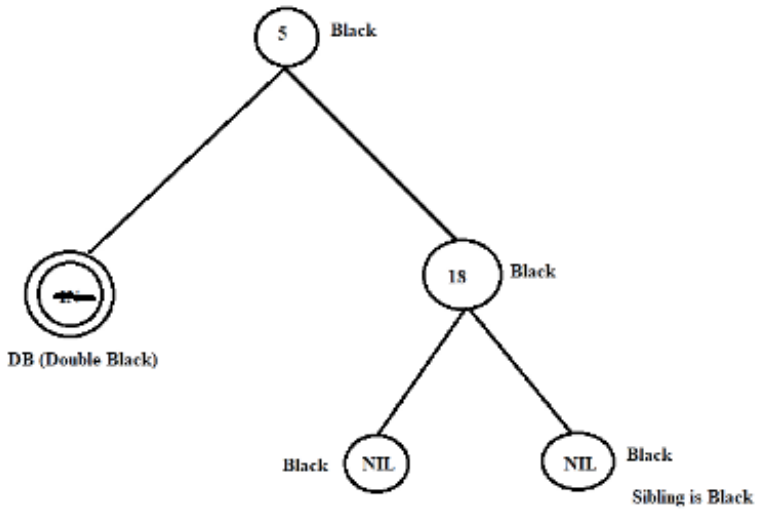
So, time complexity is O(1).



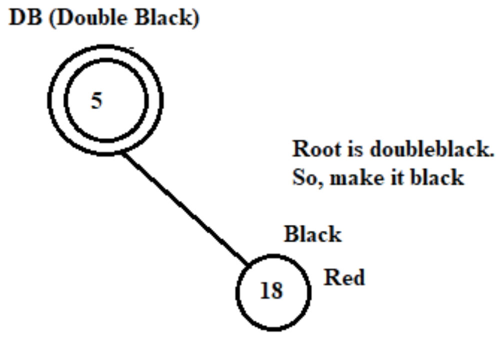
Delete 9

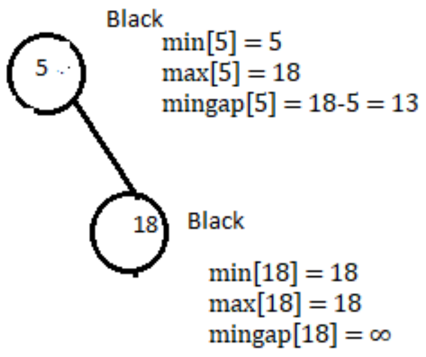


There are no children left for 9. So, simly delete it and update min, max and mingap values.



Sibling is black and both the children are black. So, make the parent as black and sibling as red. As parent is already black, it becomes double black.





min[5] = 5
max[5] = 18
mingap[5] = 18-5 = 13

min[18] = 18
max[18] = 18
mingap[18] = ∞