References:

1. https://www.cs.purdue.edu/homes/ayg/CS251/slides/chap13c.pdf
2. Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Algorithm 1 INSERT,DELETE,SEARCH INTERVAL TREES

```
1: procedure LEFT-ROTATE(T,x)
2:     y ← right[x]
3:     right[x] ← left[y]
4:     if left[y] ≠ NIL then
5:         parent[left[y]] ← x
6:     parent[y] ← parent[x]
7:     if parent[x] ← NIL then
8:         root[T] ← y
9:     else if x = left[parent[x]] then
10:        left[parent[x]] ← y
11:    else
12:        right[parent[x]] ← y
13:    left[y] ← x
14:    parent[x] ← y
15:    max[x] = MAX(max[left[x]], max[right[x]], high[x])        ▷ reconfiguring the augmented values
16:    max[y] = MAX(max[left[y]], max[right[y]], high[y])
17:    max[parent[y]] = MAX(max[left[parent[y]]], max[right[parent[y]]], high[parent[y]])
18:
19: procedure RIGHT-ROTATE(T,y)                                 ▷ analogous to LEFT-ROTATE
20:    x ← left[y]
21:    left[y] ← right[x]
22:    if right[x] ≠ NIL then
23:        parent[right[x]] ← y
24:    parent[x] ← parent[y]
25:    if parent[y] ← NIL then
26:        root[T] ← x
27:    else if y = right[parent[y]] then
28:        right[parent[y]] ← x
29:    else
30:        left[parent[y]] ← x
31:    right[x] ← y
32:    parent[y] ← x
33:    max[x] = MAX(max[left[x]], max[right[x]], high[x])
34:    max[y] = MAX(max[left[y]], max[right[y]], high[y])
35:    max[parent[x]] = MAX(max[left[parent[x]]], max[right[parent[x]]], high[parent[x]])
36:
37: procedure BST-INSERT(T,z)
38:    y ← NIL
39:    x ← root[T]
40:    while x ≠ NIL do
41:        y ← x
42:        if low[z] < low[x] then
43:            x ← left[x]
44:        else
45:            x ← right[x]
46:    parent[z] ← y
47:    if y = NIL then
48:        root[T] ← z                                          ▷ z is the only node
49:    else if low[z] < low[y] then                             ▷ setting up the pointers to z
50:        left[y] ← z
51:    else
52:        right[y] ← z
53:    max[z] = high[z]                                         ▷ setting up augmented value of inserted node
```

55: **procedure** RB-INSERT(T,x) ▷ inserting a node in interval(Red Black) tree
56: $BST - INSERT(T, x)$ ▷ First insert x as normally inserted into BST and color it red
57: $color[x] \leftarrow RED$
58: **while** $x \neq root[T]$ *and* $color[parent[x]] = RED$ **do**
59: **if** $parent[x] = left[parent[parent[x]]]$ **then**
60: $y \leftarrow right[parent[parent[x]]]$ ▷ uncle
61: **if** $color[y] = RED$ **then** ▷ Case a
62: $color[parent[x]] \leftarrow BLACK$
63: $color[y] \leftarrow BLACK$
64: $color[parent[parent[x]]] \leftarrow RED$
65: $x \leftarrow parent[parent[x]]$ ▷ Change x to grandparent
66: **else if** $x = right[parent[x]]$ **then** ▷ Case b:Left Right Case
67: $x \leftarrow parent[x]$
68: $LEFT - ROTATE(T, x)$
69: $color[parent[x]] \leftarrow BLACK$ ▷ Follow Case b: Left Left Case
70: $color[parent[parent[x]]] \leftarrow RED$
71: $RIGHT - ROTATE[T, parent[parent[x]]]$
72: **else** ▷ Case b: Left Left Case
73: $color[parent[x]] \leftarrow BLACK$
74: $color[parent[parent[x]]] \leftarrow RED$
75: $RIGHT - ROTATE[T, parent[parent[x]]]$
76: **else**
77: (do the same thing in **then in line 59** clause with "right" and "left" swapped) ▷ Case b: Right Left and Right Right Case
78: $color[root[T]] \leftarrow BLACK$ ▷ Since root is always black
79:
80: **procedure** RB-DELETE(T,z) ▷ Deleting a node in RB-Tree
81: **if** $left[z] = nil[T]$ *or* $right[z] = nil[T]$ **then** ▷ z has no or 1 child
82: $y \leftarrow z$
83: **else**
84: $y \leftarrow RB - SUCCESSOR(z)$ ▷ z has 2 children
85: **if** $left[y] \neq nil[T]$ **then**
86: $x \leftarrow left[y]$
87: **else**
88: $x \leftarrow right[y]$
89: $parent[x] \leftarrow parent[y]$ ▷ y gets removed
90: $max[parent[x]] = MAX(high[x], high[parent[x]])$ ▷ changed the augmented value
91: **if** $parent[y] = nil[T]$ **then**
92: $root[T] \leftarrow x$
93: **else if** $y = left[parent[y]]$ **then** ▷ reconfiguring the pointers to x
94: $left[parent[y]] \leftarrow x$
95: **else**
96: $right[parent[y]] \leftarrow x$
97: **if** $y \neq z$ **then** ▷ z had 2 children
98: $low[z] \leftarrow low[y]$ ▷ changed the augmented and key values
99: $high[z] \leftarrow high[y]$
100: $max[z] \leftarrow MAX(high[z], max[left[z]], max[right[z]])$
101: **if** $color[y] = BLACK$ **then** ▷ no change in black height for deleting red
102: $RB - DELETE - CORRECTION(T, x)$ ▷ if deleted black, need to check for violations
return y

```
103: procedure RB-SUCCESSOR(x)                          ▷ helper for finding successor for a node in tree
104:     if right[x] ≠ NIL then return RB − MINIMUM(right[x])
105:     y ← parent[x]
106:     while y ≠ NIL and x = right[y] do
107:         x ← y
108:         y ← parent[y]
         return y
109:
110:     procedure RB-MINIMUM(x)                        ▷ helper for finding minimum in tree
111:         while left[x] ≠ NIL do
112:             x ← left[x]
             return x
113:
114: procedure RB-DELETE-CORRECTION(T,x)
115:     while x ≠ root[T] and color[x] = BLACK do
116:         if x = left[parent[x]] then                ▷ assume x has double black
117:             w ← right[parent[x]]                   ▷ Old Sibling
118:             if color[w] = RED then
119:                 color[w] ← BLACK                   ▷ Recolour old sibling and parent
120:                 color[parent[x] ← RED
121:                 LEFT − ROTATE(T, parent[x])
122:                 w ← right[parent[x]]
123:             if color[left[w]] = BLACK and color[right[w]] = BLACK then   ▷ both the children of siblings are black
124:                 color[w] ← RED
125:                 x ← parent[x]                      ▷ will recur for parent
126:             else if color[right[w]] = BLACK then   ▷ one of the children of sibling is red
127:                 color[left[w]] ← BLACK             ▷ Right Left Case
128:                 color[w] ← RED
129:                 RIGHT − ROTATE(T, w)
130:                 w ← right[parent[x]]
131:                 color[w] ← color[parent[x]]
132:                 color[parent[x]] ← BLACK
133:                 color[right[w]] ← BLACK
134:                 LEFT − ROTATE(T, parent[x])
135:                 x ← root(T)
136:             else                                   ▷ Right Right Case
137:                 color[w] ← color[parent[x]]
138:                 color[parent[x]] ← BLACK
139:                 color[right[w]] ← BLACK
140:                 LEFT − ROTATE(T, parent[x])
141:                 x ← root(T)
142:         else
143:             (do the same thing in then in line 116 clause with "right" and "left" swapped)   ▷ Case b: Left Left and Left Right Case
144:     color[x] ← BLACK
145:
```

```
146: procedure SEARCH(root,interval)        ▷ interval to be searched has attributes low and high
147:     if root = NULL then return NULL
148:     if Interval[root].low ≤ interval.high and interval.low ≤ Interval[root].high then        ▷ Checking for overlaps return Interval[root]
149:     if left[root] ≠ NULL and max[left[root]] ≥ interval.low then return SEARCH(left[root], interval)▷ interval may overlap with an interval in left subtree
         return SEARCH(right[root], interval)        ▷ Otherwise recur for right subtree
```