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 SEARCH(root,interval)        ▷ interval to be searched has attributes low and high
2:     if root = NULL then return NULL
3:     if Interval[root].low ≤ interval.high and interval.low ≤ Interval[root].high then   ▷ Checking for overlaps return Interval[root]
4:     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
5:
6: procedure LEFT-ROTATE(T,x)
7:     y ← right[x]
8:     right[x] ← left[y]
9:     if left[y] ≠ NIL then
10:        parent[left[y]] ← x
11:    parent[y] ← parent[x]
12:    if parent[x] ← NIL then
13:        root[T] ← y
14:    else if x = left[parent[x]] then
15:        left[parent[x]] ← y
16:    else
17:        right[parent[x]] ← y
18:    left[y] ← x
19:    parent[x] ← y
20:    max[x] = MAX(max[left[x]], max[right[x]], high[x])   ▷ reconfiguring the augmented values
21:    max[y] = MAX(max[left[y]], max[right[y]], high[y])
22:    max[parent[y]] = MAX(max[left[parent[y]]], max[right[parent[y]]], high[parent[y]])
23:
24: procedure RIGHT-ROTATE(T,y)           ▷ analogous to LEFT-ROTATE
25:    x ← left[y]
26:    left[y] ← right[x]
27:    if right[x] ≠ NIL then
28:        parent[right[x]] ← y
29:    parent[x] ← parent[y]
30:    if parent[y] ← NIL then
31:        root[T] ← x
32:    else if y = right[parent[y]] then
33:        right[parent[y]] ← x
34:    else
35:        left[parent[y]] ← x
36:    right[x] ← y
37:    parent[y] ← x
38:    max[x] = MAX(max[left[x]], max[right[x]], high[x])
39:    max[y] = MAX(max[left[y]], max[right[y]], high[y])
40:    max[parent[x]] = MAX(max[left[parent[x]]], max[right[parent[x]]], high[parent[x]])
41:
```

```
42: procedure BST-INSERT(T,z)
43:     y ← NIL
44:     x ← root[T]
45:     while x ≠ NIL do
46:         y ← x
47:         if low[z] < low[x] then
48:             x ← left[x]
49:         else
50:             x ← right[x]
51:     parent[z] ← y
52:     if y = NIL then
53:         root[T] ← z                                   ▷ z is the only node
54:     else if low[z] < low[y] then                      ▷ setting up the pointers to z
55:         left[y] ← z
56:     else
57:         right[y] ← z
58:     max[z] = high[z]                                  ▷ setting up augmented value of inserted node
59:
60: procedure RB-INSERT(T,x)                              ▷ inserting a node in interval(Red Black) tree
61:     BST − INSERT(T, x)                                ▷ First insert x as normally inserted into BST and color it red
62:     color[x] ← RED
63:     while x ≠ root[T] and color[parent[x]] = RED do
64:         if parent[x] = left[parent[parent[x]]] then
65:             y ← right[parent[parent[x]]]              ▷ uncle
66:             if color[y] = RED then                    ▷ Case a
67:                 color[parent[x]] ← BLACK
68:                 color[y] ← BLACK
69:                 color[parent[parent[x]]] ← RED
70:                 x ← parent[parent[x]]                 ▷ Change x to grandparent
71:             else if x = right[parent[x]] then         ▷ Case b:Left Right Case
72:                 x ← parent[x]
73:                 LEFT − ROTATE(T, x)
74:                 color[parent[x]] ← BLACK              ▷ Follow Case b: Left Left Case
75:                 color[parent[parent[x]]] ← RED
76:                 RIGHT − ROTATE[T, parent[parent[x]]]
77:             else                                      ▷ Case b: Left Left Case
78:                 color[parent[x]] ← BLACK
79:                 color[parent[parent[x]]] ← RED
80:                 RIGHT − ROTATE[T, parent[parent[x]]]
81:         else
82:             (do the same thing in then in line 64 clause with "right" and "left" swapped)   ▷ Case b: Right Left and Right Right Case
83:     color[root[T]] ← BLACK                            ▷ Since root is always black
84:
```

```
85: procedure RB-DELETE(T,z)                                   ▷ Deleting a node in RB-Tree
86:     if left[z] = nil[T] or right[z] = nil[T] then          ▷ z has no or 1 child
87:         y ← z
88:     else
89:         y ← RB − SUCCESSOR(z)                              ▷ z has 2 children
90:     if left[y] ≠ nil[T] then
91:         x ← left[y]
92:     else
93:         x ← right[y]
94:     parent[x] ← parent[y]                                  ▷ y gets removed
95:     max[parent[x]] = MAX(high[x], high[parent[x]])         ▷ changed the augmented value
96:     if parent[y] = nil[T] then
97:         root[T] ← x
98:     else if y = left[parent[y]] then                       ▷ reconfiguring the pointers to x
99:         left[parent[y]] ← x
100:    else
101:        right[parent[y]] ← x
102:    if y ≠ z then                                          ▷ z had 2 children
103:        low[z] ← low[y]                                    ▷ changed the augmented and key values
104:        high[z] ← high[y]
105:        max[z] ← MAX(high[z], max[left[z]], max[right[z]])
106:    if color[y] = BLACK then                               ▷ no change in black height for deleting red
107:        RB − DELETE − CORRECTION(T, x)                     ▷ if deleted black, need to check for violations
        return y
108:
109: procedure RB-SUCCESSOR(x)                                 ▷ helper for finding successor for a node in tree
110:    if right[x] ≠ NIL then return RB − MINIMUM(right[x])
111:    y ← parent[x]
112:    while y ≠ NIL and x = right[y] do
113:        x ← y
114:        y ← parent[y]
        return y
115:
116: procedure RB-MINIMUM(x)                                   ▷ helper for finding minimum in tree
117:    while left[x] ≠ NIL do
118:        x ← left[x]
        return x
119:
```

```
120: procedure RB-DELETE-CORRECTION(T,x)
121:     while x ≠ root[T] and color[x] = BLACK do
122:         if x = left[parent[x]] then                          ▷ assume x has double black
123:             w ← right[parent[x]]                            ▷ Old Sibling
124:             if color[w] = RED then
125:                 color[w] ← BLACK                            ▷ Recolour old sibling and parent
126:                 color[parent[x] ← RED
127:                 LEFT − ROTATE(T, parent[x])
128:                 w ← right[parent[x]]
129:             if color[left[w]] = BLACK and color[right[w]] = BLACK then   ▷ both the children of siblings are black
130:                 color[w] ← RED
131:                 x ← parent[x]                               ▷ will recur for parent
132:             else if color[right[w]] = BLACK then            ▷ one of the children of sibling is red
133:                 color[left[w]] ← BLACK                      ▷ Right Left Case
134:                 color[w] ← RED
135:                 RIGHT − ROTATE(T, w)
136:                 w ← right[parent[x]]
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                                            ▷ Right Right Case
143:                 color[w] ← color[parent[x]]
144:                 color[parent[x]] ← BLACK
145:                 color[right[w]] ← BLACK
146:                 LEFT − ROTATE(T, parent[x])
147:                 x ← root(T)
148:         else
149:             (do the same thing in then in line 122 clause with "right" and "left" swapped)   ▷ Case b: Left Left and Left Right Case
150:     color[x] ← BLACK
151:
```