# Activity 1: Red/Black Tree

1. (10, 5, 50, 35, 40, 15, 95, 65, 20)

Diagram

Description automatically generated

Tree Generated from Demo App.

Diagram

Description automatically generated

1. (5, 10, 20, 30, 40, 50, 60, 70)

A picture containing clock

Description automatically generated

Tree Generated from Demo App.

Background pattern

Description automatically generated

1. (100, 90, 80, 70, 60, 50, 40, 30, 20, 10, 5)

Diagram

Description automatically generated

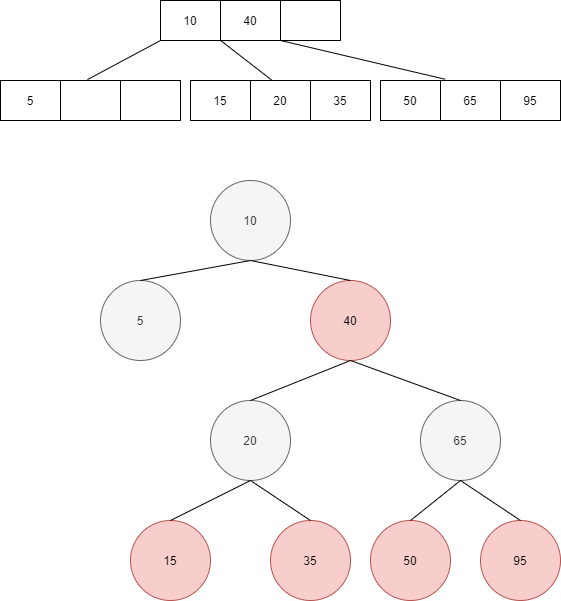
Tree Generated from Demo App.

Background pattern

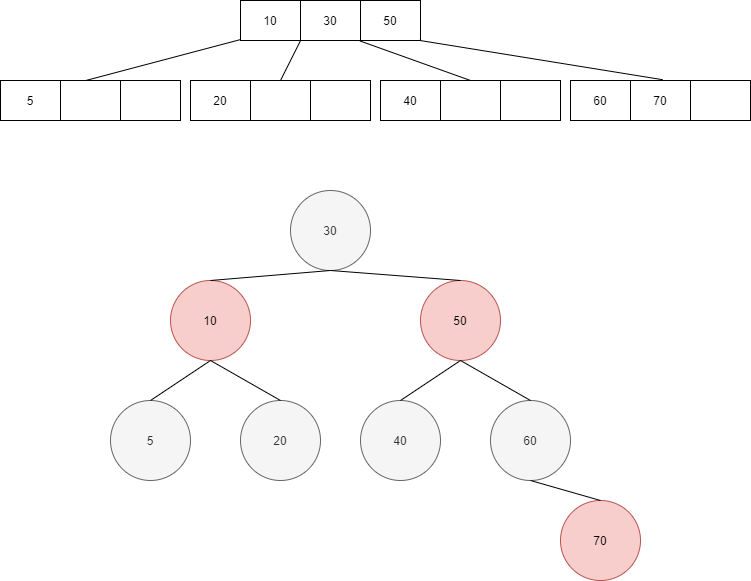
Description automatically generatedNo differences.

# Activity 2-3-4 Trees

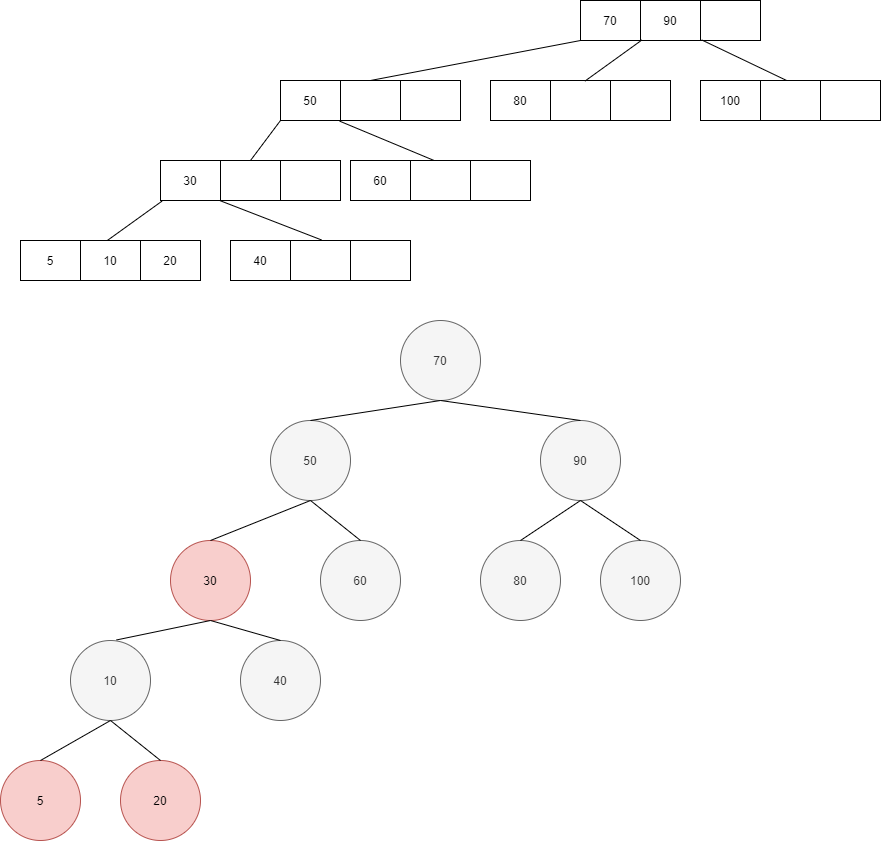
1. (10, 5, 50, 35, 40, 15, 95, 65, 20)



1. (5, 10, 20, 30, 40, 50, 60, 70)

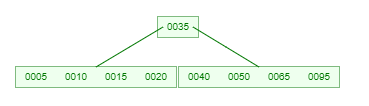


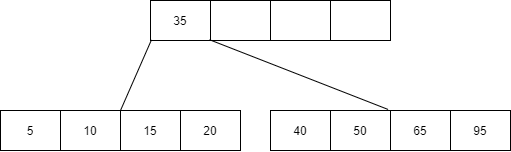
1. (100, 90, 80, 70, 60, 50, 40, 30, 20, 10, 5)



# Activity B-Trees

1. (10, 5, 50, 35, 40, 15, 95, 65, 20)

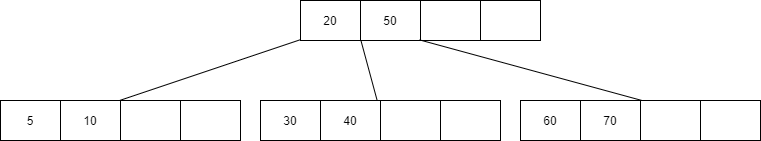




1. (5, 10, 20, 30, 40, 50, 60, 70)

Timeline

Description automatically generated



Its not any different.

# Activity Reflection

1. All three resultant trees have a less height in comparison to a Binary Search Tree for the same values, this is especially evident when entering values in ascending and descending values when the height of a BST becomes (h = n; where h is height of tree and n is the number of elements)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Insert | Find | Delete |
| Red/Black Trees | Check Parents, Grand Parents, Switch Colors, Change nodes. | Same as Binary Search Tree | Check Parents, Grand Parents, Switch Colors, Change nodes |
| 2-3-4 Trees | Insert into Leaf, if leaf is full, create new leaf and do additional linking among the leafs from the top up. | Same as Binary Search Tree but must travel across the index’s in the leaf so added cost. | Search for Entry, delete entry and then reshuffle inside the leaf and update the links. |
| B-Trees | Insert into Leaf, if leaf is full, create new leaf and do additional linking among the leafs from the top up. | Same as Binary Search Tree but must travel across the index’s in the leaf so added cost. | Search for Entry, delete entry and then reshuffle inside the leaf and update the links. |

1. Binary Search Tree would be the easiest to implement, followed by the B-trees since there is little to no reshuffling of the elements upon entering whereas Red/Black trees require you to constantly check for the status of Parents/Grandparents/Uncle Nodes.
2. 1. Red/Black Tree, the inorder traversal will work exactly like the Binary Search Tree where it is Left Index, Node, Right Index recursively.
   2. Both the 2-3-4 trees and the B-trees will work in such a way where they will recursively go down the leftmost link, left index, go down the path of the link to the immediate right of the left index and repeat until the end of the leaf, recursively. Picture attached for simplicity.

