lAiman Abdullah

1001472426

Assignment ID: 02

Q\ 1

(A)

To know the maximum number of nodes in Binary search Tree with height K. We need to

Calculate the height of Tree. To do that Let’s consider the height Root will start from 1. And level of BST will start at Level 0

So will have:

Examples:

A close up of text on a whiteboard

Description automatically generated

(B)

A close up of text on a whiteboard

Description automatically generated

(C)

In Order- Traverse:

9 -> 12 -> 14 -> 17 -> 19 -> 23 -> 50 -> 54 -> 67 -> 72 -> 76

Pre Order- Traverse:

50 -> 17 -> 12 -> 9 -> 14 -> 23 -> 19 -> 72 -> 54 -> 67 -> 76

Post Order- Traverse:

1. -> 14 -> 12 -> 19 -> 23 -> 17 -> 67 -> 54 -> 76 -> 72 -> 50

Q\ 2

(A)

The time complexity of Heap is O (N log N). Heap is kind of Balance tree but have unsorted data.

So We will building Heap either Max Heap or Min Heap. Both are same steps except Max heap will have max value in Root, in other side Min heap will have lowest value in Root. Let’s say we will build in a Max- Heap way. So, the data will be input one by one and each node inserted after the Root node will checked the parent node to see the value is smaller or bigger than value of node itself if value is smaller will swap values and

Keep checking with parents until hit the root to maintain the Max Heap properties.

Finally, insert one node will take O(log N) but building heap tree will take O(N log N) where N= no. of operations , log N = time complexity of insert one node.

(B)

K=3 which K is height

For the Minimum number of points heap can have: We can find it by equation (2^k). So, 2^3 = 8 that max heap can have.

For Maximum number of points heap can have: We can find it by equation (2^k-1)+1. So, (2^3-1)+1 = 15 that max heap can have.

(C)

A close up of text on a white background

Description automatically generated

Q\3

(A)

|  |  |
| --- | --- |
| **AVL Tree** | **BST** |
| **Insert**:  O (Log N) | **Insert**:  O (N) |
| **Delete:**  O (Log N) | **Delete:**  O (N) |
| **Search:**  O (Log N) | **Search:**  O (N) |

**AVL Tree**: For Insert, Delete is better than BST because is Very Fast.

AVL Tree maintain Balance, mean left subtree and right subtree will not different in length more than 1 node. Which lead to fast searching than BST. Therefore, AVL will not take more or same time of BST to implement operations.

**BST**: Is maintain node depending on Root value, so any values less than Root will be in left subtree and values greater than Root will be in Right subtree. For insertion will need to travel from root to leaf will take O(h) which h is height of the tree. For delete same thing to remove leaf node need O(h). For searching need to traverse O(h) to find the node.

(B)

A close up of text on a white background

Description automatically generated

(C)

A close up of text on a white background

Description automatically generated

Q\4

(A)

**Red – Black Tree:**

Its useful more than BST in meanwhile its BST. Red black tree is faster than BST and with operations insertion, searching and deletion need O (log N), read black tree guaranteed time complexity of insert, delete, search will be O(log N). Also, its self-balanced tree mean it maintain height of tree by recoloring and rotation sometimes. This mean does not need more time to traverse to do search.

**BST**: with Binary Search Tree there is no self-Balancing as Red-Black tree mean it could be long chain of nodes which will need more time to do searching. In BST we search, insert and delete in O(N) which is more time need to perform operations. BST could be

(B)

A close up of text on a white background

Description automatically generated

(C)

A close up of text on a white background

Description automatically generated