Design and Analysis of Algorithm (DAA)

Dynamic Programming

(Optimal Binary Search Tree)

Dr. Dayal Kumar Behera

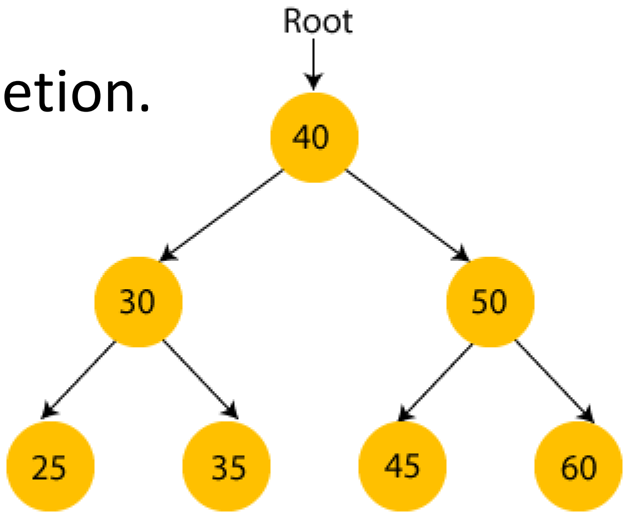
School of Computer Engineering
KIIT Deemed to be University, Bhubaneswar, India

Binary Search Tree (BST)

A tree where each node has a value, and all nodes in the left subtree of a node have smaller values, while nodes in the right subtree have larger values.

left_subtree (value) < node (value) < right_subtree (value)

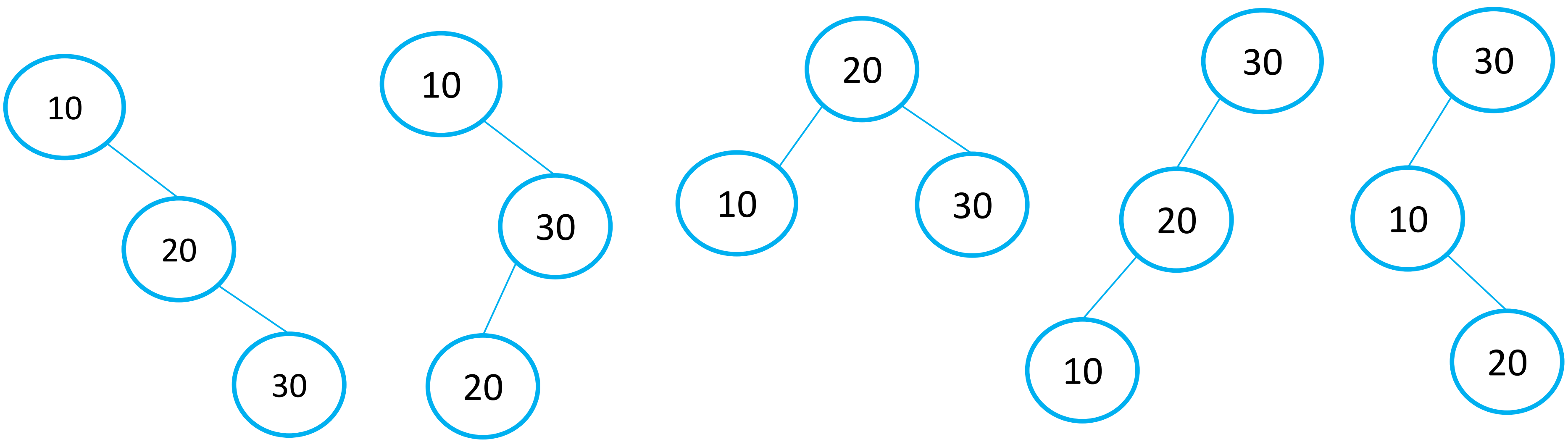
Data structure for fast search, insertion, deletion.



Binary Search Tree (BST)

For “n” nodes, No. of Binary Search Tree: $\frac{2n_{C_n}}{n+1}$

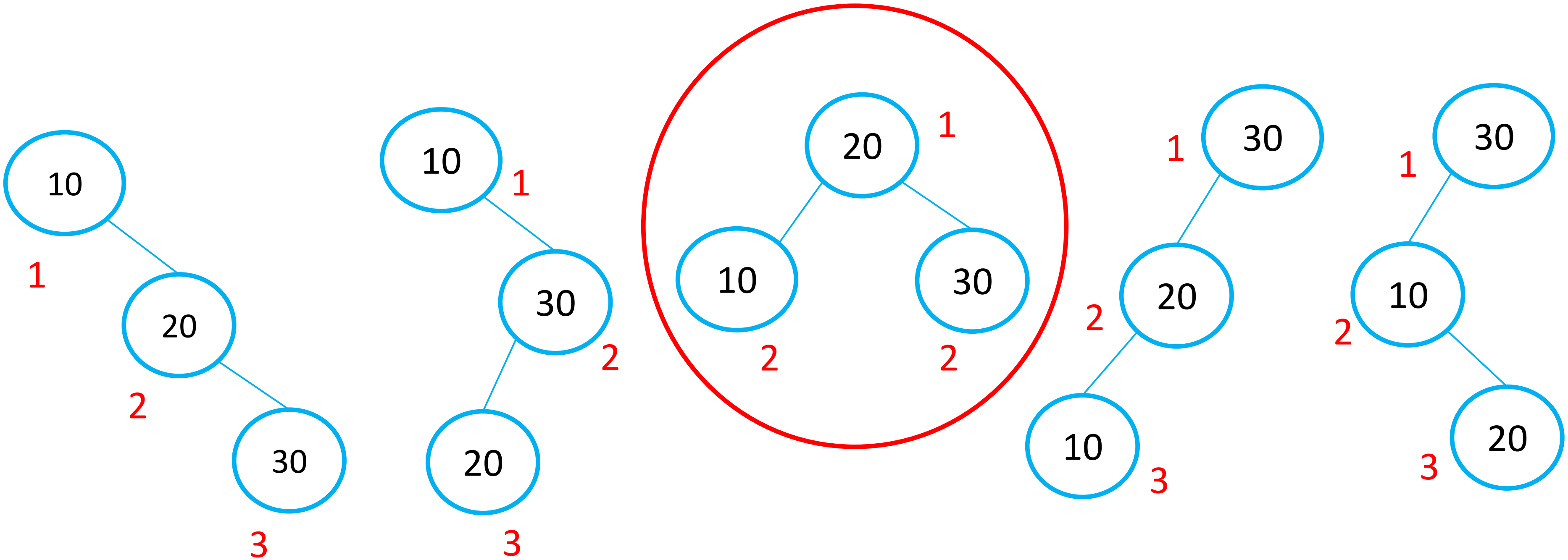
Let n =3, Keys = [10, 20, 30]



No. of Binary Search Tree: $\frac{6_{C_3}}{3+1} = 5$

Binary Search Tree (BST)

No of Comparison:



No of Comparison = (1+2+3) = 6

(1+2+3) = 6

(1+2+2) = 5

(1+2+3) = 6

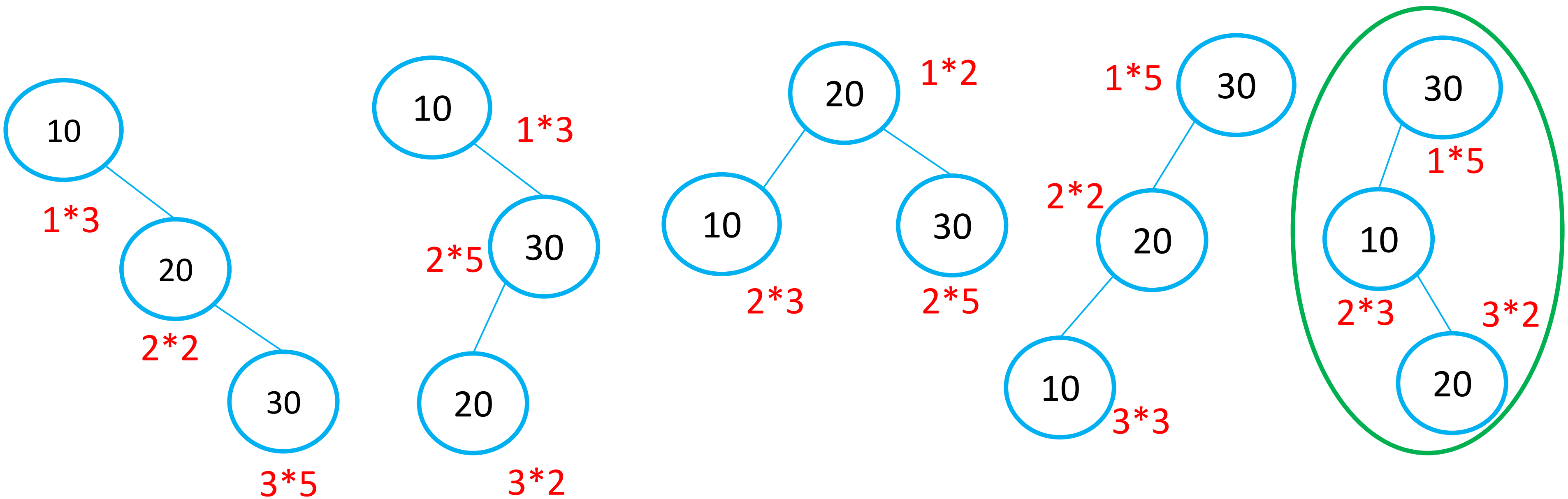
(1+2+3) = 6

The search cost of a node in a BST is proportional to its level (depth).

Whether height balanced tree is Optimal Binary Search Tree?

Binary Search Tree (BST)

Consider Keys = [10, 20, 30] with search frequency = [3, 2, 5], resp.



No of Comparison
= (3+4+15) = 22

19

18

18

17

Optimal Binary Search Tree

OBST Problem

Given:

Keys: $k_1, k_2 \ldots, k_n$ sorted in increasing order.

Probabilities: Each key k_i has a probability p_i of being accessed (search frequency).

Goal: Build a BST where the total search cost (expected search time) is minimized.

Understanding Cost in OBST

Search Cost: The search cost of a node in a BST is proportional to its level(depth).

Expected Search Cost: Assuming only successful search

$$Expected\ Cost = \sum_{i=1}^{n} p_i * level(k_i)$$

DP Approach to OBST

Dynamic Programming can be used to find the OBST by breaking down the problem into smaller subproblems, which are combined to obtain the final solution.

Steps:

1. Define Subproblems:

Let C[i][j] represent the minimum search cost for the subtree containing keys ki to kj.

W[i][j] is the sum of probabilities from 'i' to 'j'. $W[i][j] = \sum_{k=i}^{j} p_k$

R[i][j] stores the root of the optimal subtree containing keys ki to kj.

DP Approach to OBST

2. Base Case Initialization:

For single nodes (When i==j): C$[i][j] = \ p_i$ and $\mathrm{R}[i][i] = i$

For nodes beyond the range (i.e., empty subtree): C$[i][i-1] = 0$

DP Approach to OBST

3. **Recursive Formula to Compute Optimal Cost:**

For each range of keys k_i to k_j (for all $j \geq i$):

$$C[i][j] = \min_{r=i\ to\ j} \{ C[i][r-1] + C[r+1][j] + W[i][j] \}$$

Explanation: Choose each key k_r as the root, then recursively find the cost of left and right subtrees.

Add $W[i][j]$ to account for the cumulative search cost when all nodes from i to j are in subtree.

DP Approach to OBST

4. Constructing the DP Table:

Fill $\mathrm{C}[i][j]$ in increasing order of length $L = j - i + 1$

For each length L compute $\mathrm{C}[i][j]$ for all valid i and j.

Keep track of the root in $\mathrm{R}[i][j]$ for each subtree to construct the Optimal BST.

5. Extract the Result:

The minimum cost of the OBST for all keys is stored in $\mathrm{C}[1][n]$

Example

Keys (k_i)	10	20	30
Frequency or Probabilities (p_i)	4	2	6

Algorithm

Algorithm OBST(keys, p)

```
n = length(keys)

// Initialize DP tables C, W, and R
for i = 1 to n do
    C[i][i] = p[i]          // Cost of a single key as its own subtree
    W[i][i] = p[i]          // Weight is the probability itself
    R[i][i] = i             // Root of a single key is the key itself
    C[i][i-1] = 0           // Base case for empty subtree
```

```
// Fill DP tables for increasing lengths of subtrees
for L = 2 to n do                    // L is the length of the subtree
   for i = 1 to n - L + 1 do      // Start of the subtree
      j = i + L - 1              // End of the subtree

      // Compute W[i][j] as the sum of probabilities from i to j
      W[i][j] = W[i][j-1] + p[j]

      // Initialize C[i][j] with a large value for comparison
      C[i][j] = INF

      // Calculate the minimum cost for each possible root in the range i to j
      for r = i to j do            // Try each key as the root
         cost = C[i][r-1] + C[r+1][j] + W[i][j]
         if cost < C[i][j] then
            C[i][j] = cost       // Update minimum cost for this range
            R[i][j] = r          // Store the root of the optimal subtree

// The minimum cost of the entire OBST is stored in C[1][n]
return C[1][n], R
```

Complexity Analysis

Time Complexity: O (n^3)

Nested loops for length and root choice.

Space Complexity: O (n^2)

for C[i][j], W[i][j], R[i][j]

"Each of your actions will have an impact on your future.

Thank you