DESIGN AND ANALYSIS OF ALGORITHMS

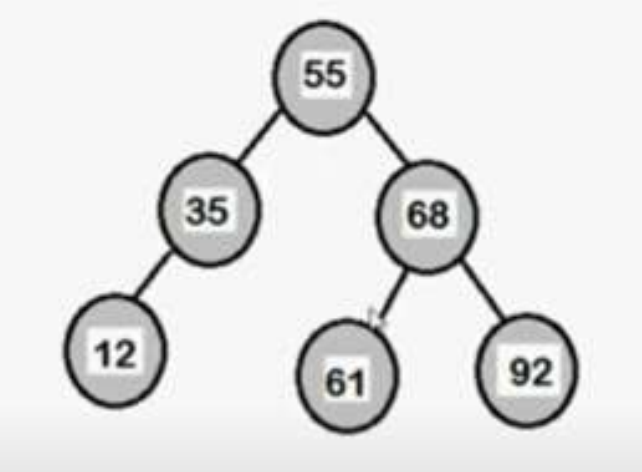
Session -20

OBST

OPTIMAL BINARY SEARCH TREE(OBST)

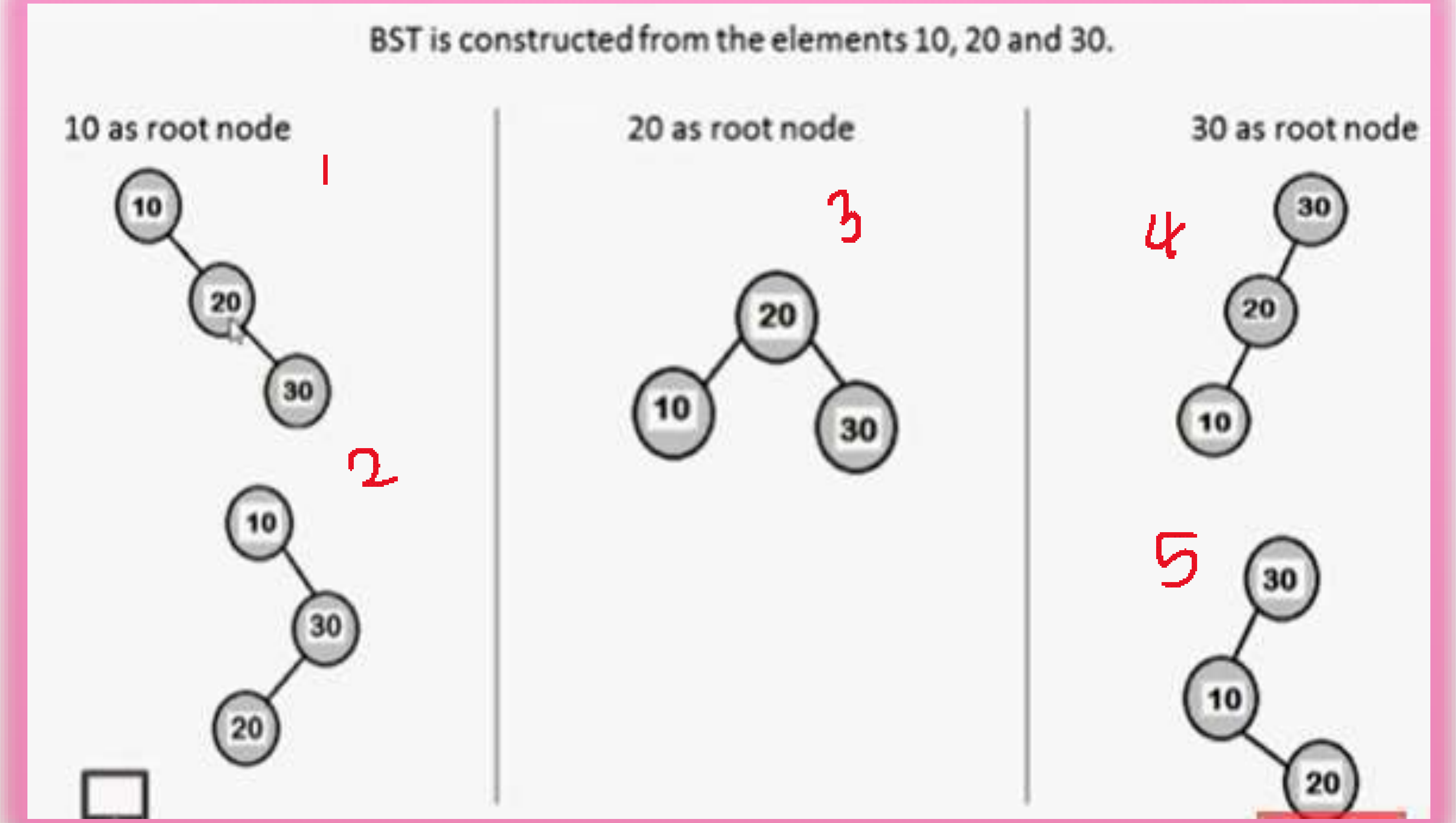
A binary search tree T is a binary tree, either it is empty or each node in the tree contains an identifier and,

- All identifiers in the left subtree of T are *less than* the identifier in the *root* node of T.
- All identifiers in the right subtree of T are *greater than* the identifier in the *root* node of T.
- The *left and right* subtree of T are also *binary search* trees.



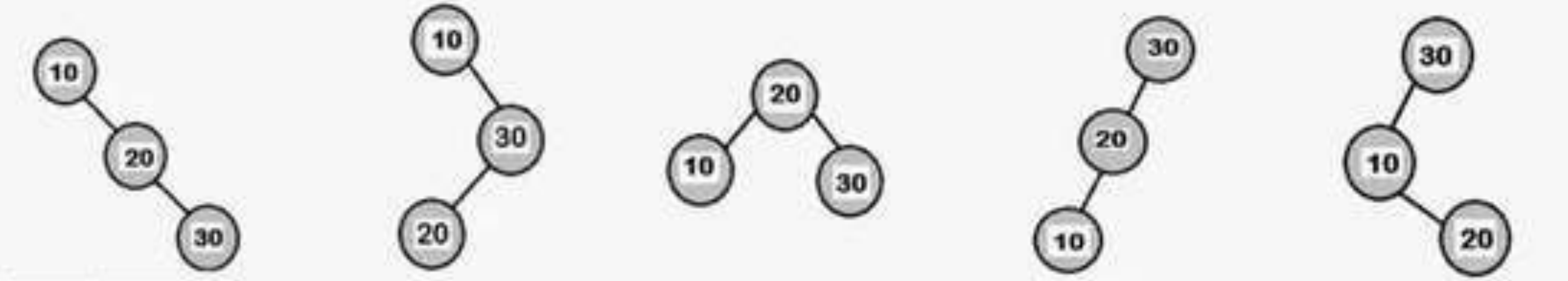
Binary Search Tree – Examples (No. of Possible BSTs)

- Ex :- $(a_1, a_2, a_3) = (\ 10, 20, 30\ )$



Binary Search Tree (BST) – COST OF SEARCHING A KEY

- Cost of searching any key is dependent on comparisons required for searching any key element in the tree.



Key	C
10	1
20	2
30	3
Avg	6/3=2

Key	C
10	1
20	3
30	2
Avg	6/3=2

Key	C
10	2
20	1
30	2
Avg	5/3=1.66

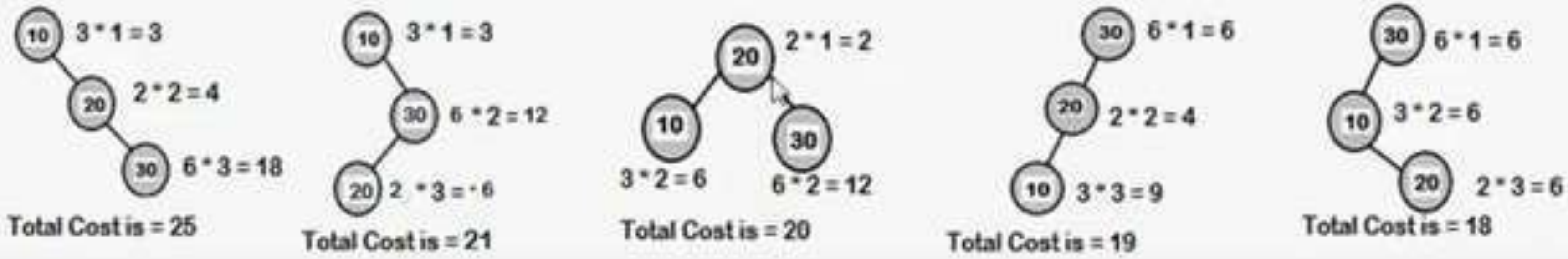
Key	C
10	3
20	2
30	1
Avg	6/3=2

Key	C
10	2
20	3
30	1
Avg	6/3=2

$$\sum_{i=0}^{n-1} \max[No.of\ Cmparisions]$$

Binary Search Tree (BST) – COST OF SEARCHING A KEY by giving the Frequency of Searching

Keys	10	20	30
Frequencies for Searching	3	2	6



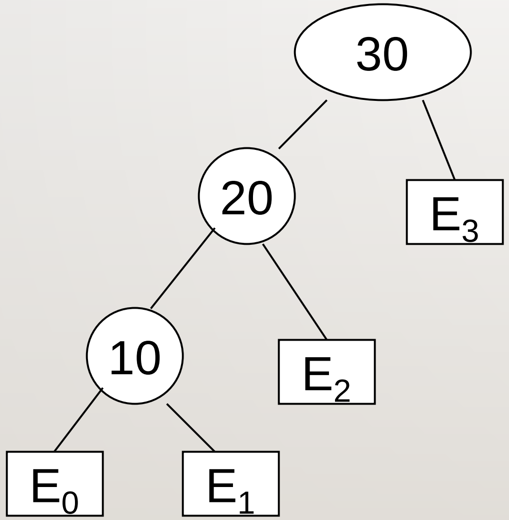
- Minimum searching cost is low means it's a Optimal BST.
- Tree 5 is having minimum searching cost = 15.
- Tree 5 is OBST .
- Though it is not height balanced, tree 5 is OBST which is based on frequencies the cost of BST is minimum.

OPTIMAL BINARY SEARCH TREE (OBST)

- **Weight-balanced Binary Search Tree**
- It is a **Binary Search Tree** which provides the **smallest possible search time (or expected search time)** for a given sequence of accesses .
- Optimal BSTs are generally divided into two types: **Static** and **Dynamic**.
- In the **Static Optimality** problem,
  - the tree **cannot be modified** after it has been constructed.
  - In this case, there exists some layout of the nodes of the tree which provides the **smallest expected search time** for the given access probabilities.
  - Various algorithms exist to construct or approximate the statically optimal tree.
- In the **Dynamic Optimality** problem,
  - the tree **can be modified at any time**, typically by permitting tree rotations.
  - The tree is considered to have a cursor starting at the root which it can move or use to perform modifications.
  - In this case, there exists some **minimal-cost sequence** of these operations which causes the cursor to visit every node in the target access sequence in order.

OPTIMAL BINARY SEARCH TREE (OBST)

- Problem
  - Given sequence of identifiers ($a_1, a_2, \ldots, a_n$) with $a_1 < a_2 < \cdots < a_n$.
  - Let ***p(i)* be the probability** *with which we* ***search for*** a_i
  - Let **q(i) be the probability** with which we **search for an identifier x** such that $a_i < x < a_{i+1}$.
  - We must build a Binary Search Tree (BST) **with minimum expected search cost.**



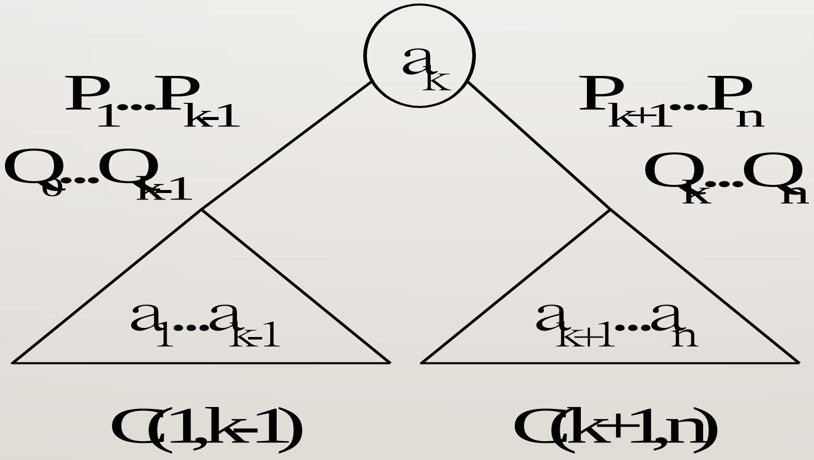
- Identifiers : 10, 20, 30
- Internal node : successful search, p(i)
- External node : unsuccessful search, q(i)

The expected cost of a binary tree:

$$\sum_{n=1}^{n} p_i * \text{level}(a_i) + \sum_{n=0}^{n} q_i * (\text{level}(E_i) - 1)$$

The dynamic programming approach

- Make a decision as which of the a_i's should be assigned to the root node of the tree.
- If we choose a_k, then it is clear that the internal nodes for $a_1, a_2, \ldots\ldots, a_{k-1}$ as well as the external nodes for the classes $E_0, E_1, \ldots., E_{k-1}$ will lie in the left subtree *l* of the root. The remaining nodes will be in the right subtree r.



- $\text{cost}(l) = \sum_{1 \le i < k} p(i) * \text{level}(a_i) + \sum_{0 \le i < k} q(i) * (\text{level}(E_i) - 1)$

- and
- $\text{cost}(r) = \sum_{k < i \le n} p(i) * \text{level}(a_i) + \sum_{k < i \le n} q(i) * (\text{level}(E_i) - 1)$

- In both the cases the level is measured by considering the root of the respective sub tree to be at level 1.

- we obtain the following as the expected cost of the above search tree.

  p(k) + cost(l) + cost(r) + w(0,k-1) + w(k,n)

- If we use $c(i,j)$ to represent the cost of an optimal binary search tree t_{ij} containing $a_{i+1},\ldots\ldots,a_j$ and $E_i,\ldots,E_j$, then

  $cost(l) = c(0,k-1)$, and $cost(r) = c(k,n)$.

- For the tree to be optimal, we must choose k such that

  $p(k) + c(0,k-1) + c(k,n) + w(0,k-1) + w(k,n)$ is minimum.

Hence, for $c(0,n)$ we obtain

$$c(0,n) = \min_{0<k\leq n} \left\{ c(0,k-1) + c(k,n) + p(k) + w(0,k-1) + w(k,n) \right\}$$

We can generalize the above formula for any $c(i,j)$ as shown below

$$c(i,j) = \min_{i<k\leq j} \left\{ c(i,k-1) + c(k,j) + \underbrace{p(k) + w(i,k-1) + w(k,j)} \right\}$$

$$c(i, j) = \min_{i<k\leq j} \left\{ cost(i,k-1) + cost(k,j) + w(i, j) \right\}$$

- Therefore, c(0,n) can be solved by first computing all
- c(i, j) such that j – i = 1, next we compute all c(i , j) such
- that j – i = 2, then all c(i , j) with j – i = 3, and so on.
- During this computation we record the root r(i, j) of each tree t_{ij}, then an optimal binary search tree can be constructed from these r(i, j).
- r(i, j) is the value of k that minimizes the cost value.

Note:1. c(i,i) = 0, w(i, j) = q(i), and r(i, j) = 0 for all $0 \leq i \leq n$

2. w(i, j) = p(j) + q(j) + w(i, j-1)

Ex 1: Let n=4, and (a_1,a_2,a_3,a_4) = (do, if, int, while).

Let p(1 : 4) = (3, 3, 1, 1) and q(0: 4) = (2, 3, 1,1,1).

p's and q's have been multiplied by 16 for convenience.

Then, we get

The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

c(0,4) + w(0.4)

c(0,0)+ c(1,4) **c(0,1)+c(2,4)** **c(0,2)+c(3,4)** **c(0,3)+c(4,4)**

The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

w(i, i) = q(i) c(i, i)= 0 and r(i, i)= 0, 0 <i <4.

w(i, j)= P(j) + q(j) + w(i, j-1)

The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

w(i, i) = q(i) c(i, i)= 0 and r(i, i)= 0, 0 <i <4.

w(i, j)= P(j) + q(j) + w(i, j-1)

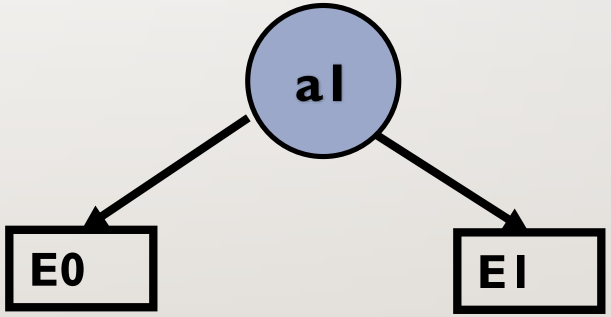
$$
\begin{aligned}
w(0,1) &= p(1) + q(1) + w(0,0) = 8 \\
c(0,1) &= w(0,1) + \min\{c(0,0) + c(1,1)\} = 8 \\
r(0,1) &= 1
\end{aligned}
$$

The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

$$
\begin{aligned}
w(0,1) &= p(1) + q(1) + w(0,0) = 8 \\
c(0,1) &= w(0,1) + \min\{c(0,0) + c(1,1)\} \quad = \quad 8 \\
r(0,1) &= 1
\end{aligned}
$$

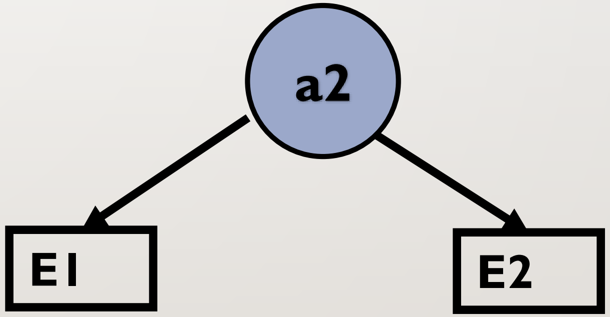


The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

$$\begin{aligned} w(1,2) &= p(2)+q(2)+w(1,1) = 7 \\ c(1,2) &= w(1,2)+\min\ \{c(1,1)+c(2,2)\} = 7 \\ r(0,2) &= 2 \end{aligned}$$

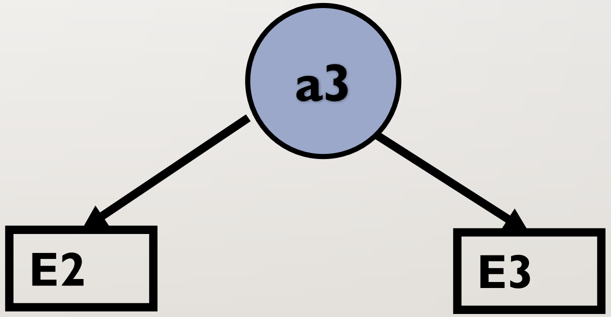


The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

$$
\begin{aligned}
w(2,3) &= p(3)+q(3)+w(2,2) = 3\\
c(2,3) &= w(2,3)+\min\ \{c(2,2)+c(3,3)\} = 3\\
r(2,3) &= 3
\end{aligned}
$$

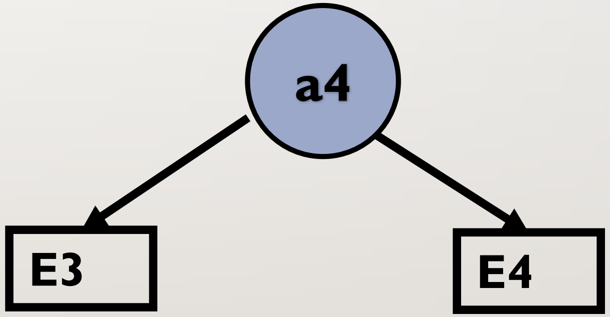


The dynamic programming approach

Let n = 4 and (a1, a2, a3,a4) = (do, if, int, while)

p(l:4) = (3,3,1,1) and q(0 :4) = (2, 3, 1, 1, 1)

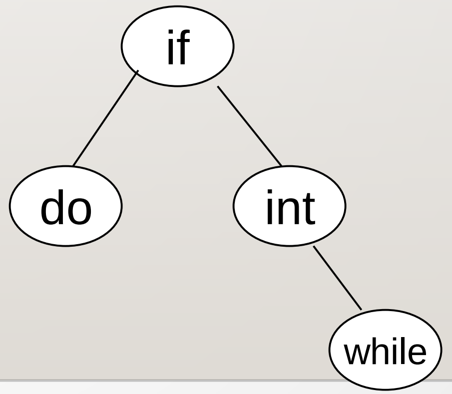
$$
\begin{aligned}
w(3,4) &= p(4) + q(4) + w(3,3) = 3 \\
c(3,4) &= w(3,4) + \min\ \{c(3,3) + c(4,4)\} = 3 \\
r(3,4) &= 4
\end{aligned}
$$



The dynamic programming approach

	0	1	2	3	4
0	$w_{00} = 2$ $c_{00} = 0$ $r_{00} = 0$	$w_{11} = 3$ $c_{11} = 0$ $r_{11} = 0$	$w_{22} = 1$ $c_{22} = 0$ $r_{22} = 0$	$w_{33} = 1$ $c_{33} = 0$ $r_{33} = 0$	$w_{44} = 1$ $c_{44} = 0$ $r_{44} = 0$
1	$w_{01} = 8$ $c_{01} = 8$ $r_{01} = 1$	$w_{12} = 7$ $c_{12} = 7$ $r_{12} = 2$	$w_{23} = 3$ $c_{23} = 3$ $r_{23} = 3$	$w_{34} = 3$ $c_{34} = 3$ $r_{34} = 4$	
2	$w_{02} = 12$ $c_{02} = 19$ $r_{02} = 1$	$w_{13} = 9$ $c_{13} = 12$ $r_{13} = 2$	$w_{24} = 5$ $c_{24} = 8$ $r_{24} = 3$		
3	$w_{03} = 14$ $c_{03} = 25$ $r_{03} = 2$	$w_{14} = 11$ $c_{14} = 19$ $r_{14} = 2$			
4	$w_{04} = 16$ $c_{04} = 32$ $r_{04} = 2$				

- From the table we can see that $c(0,4)=32$ is the minimum cost of a binary search tree for (a1, a2, a3, a4).
- The root of tree t_{04} is a_2.
- The left subtree is t_{01} and the right subtree t_{24}.
- Tree t_{01} has root a_1; its left subtree is t_{00} and right subtree t_{11}.
- Tree t_{24} has root a_3; its left subtree is t_{22} and right subtree t_{34}.
- Thus we can construct *OBST*.



OBST ALGORITHM


```
Algorithm OBST(p, q, n)
{
      for i:= 0 to n-1  do
      {                                                        //  initialize.
            w[ i, i ] :=q[ i ]; r[ i, i ] :=0;  c[ i, i ]=0;
             //  Optimal trees  with one node.
             w[ i, i+1 ]:= p[ i+1 ] + q[ i+1 ] + q[ i ] ;
             c[ i, i+1 ]:= p[ i+1 ] + q[ i+1 ] + q[ i ] ;
             r[ i, i+1 ]:= i + 1;
     }
   w[n, n] :=q[ n ]; r[ n, n ] :=0;  c[ n, n ]=0;
```

```
// Find optimal trees with m nodes.
for m:= 2 to n do
{
        for i := 0 to n – m do
        {
                j:= i + m ;
                w[ i, j ]:= p[ j ] + q[ j ] + w[ i, j -1 ];
                // Solve using Knuth's result
                x := Find( c, r, i, j );
                c[ i, j ] := w[ i, j ] + c[ i, x -1 ] + c[ x, j ];
                r[ i, j ] :=x;
         }
}
```

```
Algorithm  Find( c, r, i, j )
{
            for k := r[ i, j -1 ] to r[ i+1, j ] do
            {          min :=∞;
            if ( c[ i, k -1 ]  +c[ k, j ] < min ) then
                 {
                 min := c[ i, k-1 ] + c[ k, j ];
                 y:= k;
                 }
      }
 return y;
}
```

TIME COMPLEXITY OF ABOVE PROCEDURE TO EVALUATE THE *C'S* AND *R'S*

- Above procedure requires to compute $c(i, j)$ for

  $(j - i) = 1,2,\ldots\ldots,n$.

- When $j - i = m$, there are $n\text{-}m+1$ $c(i, j)$'s to compute.

- The computation of each of these $c(i, j)$'s requires to find m quantities.

- Hence, each such $c(i, j)$ can be computed in time $o(m)$.

- The total time for all *c(i,j)'s* with *j – i= m* is

$$= m(n-m+1)$$

$$= mn-m^2+m$$

$$=O(mn-m^2)$$

- Therefore, the *total time* to evaluate all the *c(i, j)'s* and *r(i, j)'s* is

$$\sum_{1 \le m \le n} (mn - m^2) = O(n^3)$$

- We can reduce the *time complexity* by using the observation of *D.E. Knuth*

- *Observation:*
  - The optimal k can be found by limiting the search to the range $r(i, j-1) \leq k \leq r(i+1, j)$

- In this case the *computing* time is $O(n^2)$.

Ex 2: Let $n=4$, and $(a_1,a_2,a_3,a_4) =$ (count, float, int,while).

Let $p(1:4) = (1/20, 1/5, 1/10, 1/20)$ and

$q(0:4) = (1/5, 1/10, 1/5, 1/20, 1/20)$.

Using the r(i, j)'s construct an optimal binary search tree.

THANK YOU