**Experiment No: 9**

**AIM:** Implementation OBST (Dynamic Programming) and obtaining its step count.

**THEORY:**

Given a fixed set of identifiers, we can create a binary search tree organisation. There can be different search trees for the same set of identifiers, having distinct performance characteristics. Optimal Binary Search Tree Algorithm is used to obtain a binary search tree, for this set, in which the total cost of all the searches is as small as possible.

*What is the cost of a BST?*

The cost of a BST node is the product of the level of the node and its frequency.

For Example:

Input: keys[] = {10, 12}, freq[] = {34, 50}

There can be following two possible BSTs

10 12

\ /

12 10

I II

Frequency of searches of 10 and 12 are 34 and 50 respectively.

The cost of tree I is 34\*1 + 50\*2 = 134

The cost of tree II is 50\*1 + 34\*2 = 118

**ALGORITHM:**

AlgorithmOBST(p,q,n)

// Given n distinct identifiers a1<a2<….<an and probabilities

// p[i], 1<i <n, and q[i], 0 <i <n, this algorithm computes

// the cost c[i,j] of optimal binary search trees tij for identifiers

// ai+I,…,aj. It also computes r[i,j], the root of tij.

// w[i,j] is the weight of tij.

{

**for** i :=0 to n-1 **do**

{

// Initialize

w[i,i]:=q[i]; r[i,i]:=0; c[i,i]:=0.0;

// Optimal trees with one node

w[i,i+ i]:=q[i]+q[i +l]+p[i+ l]i

r[i,i+l]:=i+ 1;

c[I,i+1] :=q[i]+q[i + 1]+p[i +1];

}

w[n,n] :=q[n]; r[n,n] :=0; c[n,n]:=0.0;

**for** m :=2 to n **do** // Find optimal trees with m nodes.

**for** i :=0 to n-m **do**

{

j :=i +m;

w[i,j] :=w[i,j-1]+ p[j] + q[j];

k :=Find(c,r,i,j);

// A value of I in the range[i,j-1]<l

// <r[i+1,j] that minimizes c[i,I-1] + c[l,j]

c[i,j]:=w[i,j]+c[i,k-1]+c[k,j];

r[i,j]:=k;

}

write (c[0,n], w[0,n],r[0,n]);

}

AlgorithmFind(c,r,i, j)

{

min:=∞;

**for** m :=r[i,j-1] to r[i+ 1,j]**do**

if (c[i,m-1]+c[m,j]) < min **then**

{

min:=c[i,m-1]+c[m,j];I :=m;

}

return I;

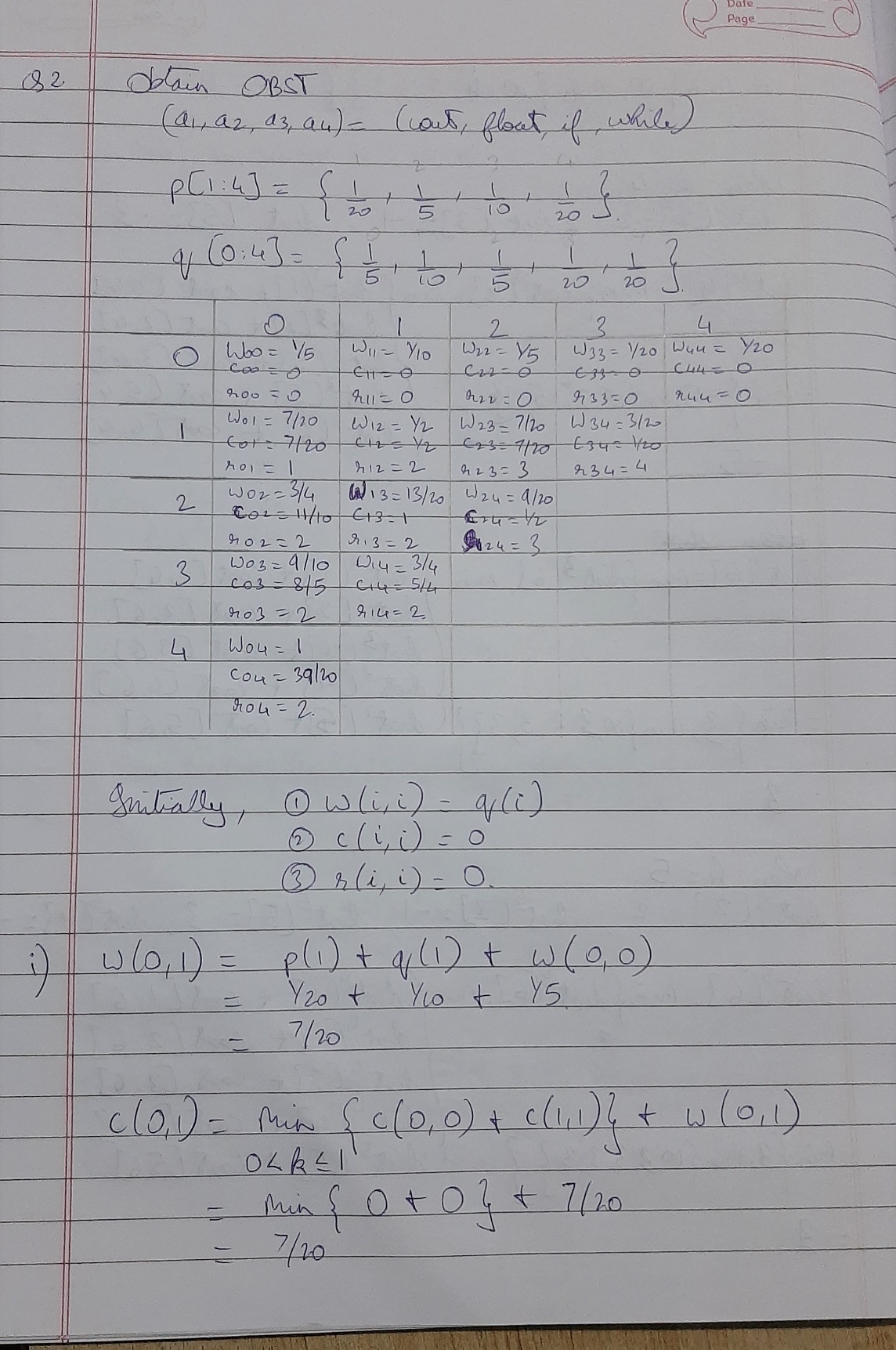
}

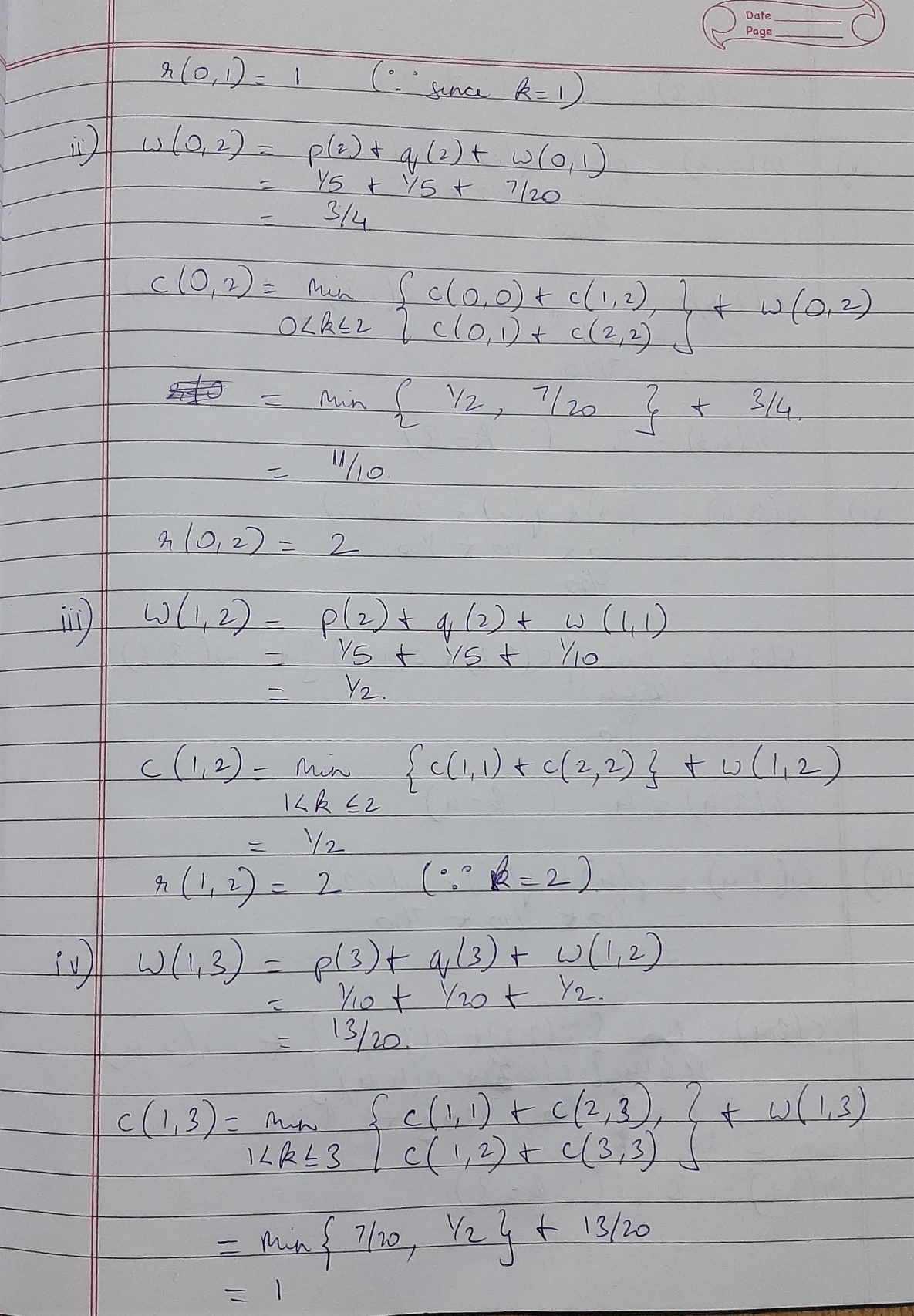
*Choose the root k such that |w(0,k-1) – w(k,n)| is as small as possible. Repeat the procedure to find the left and right sub trees of the root.*

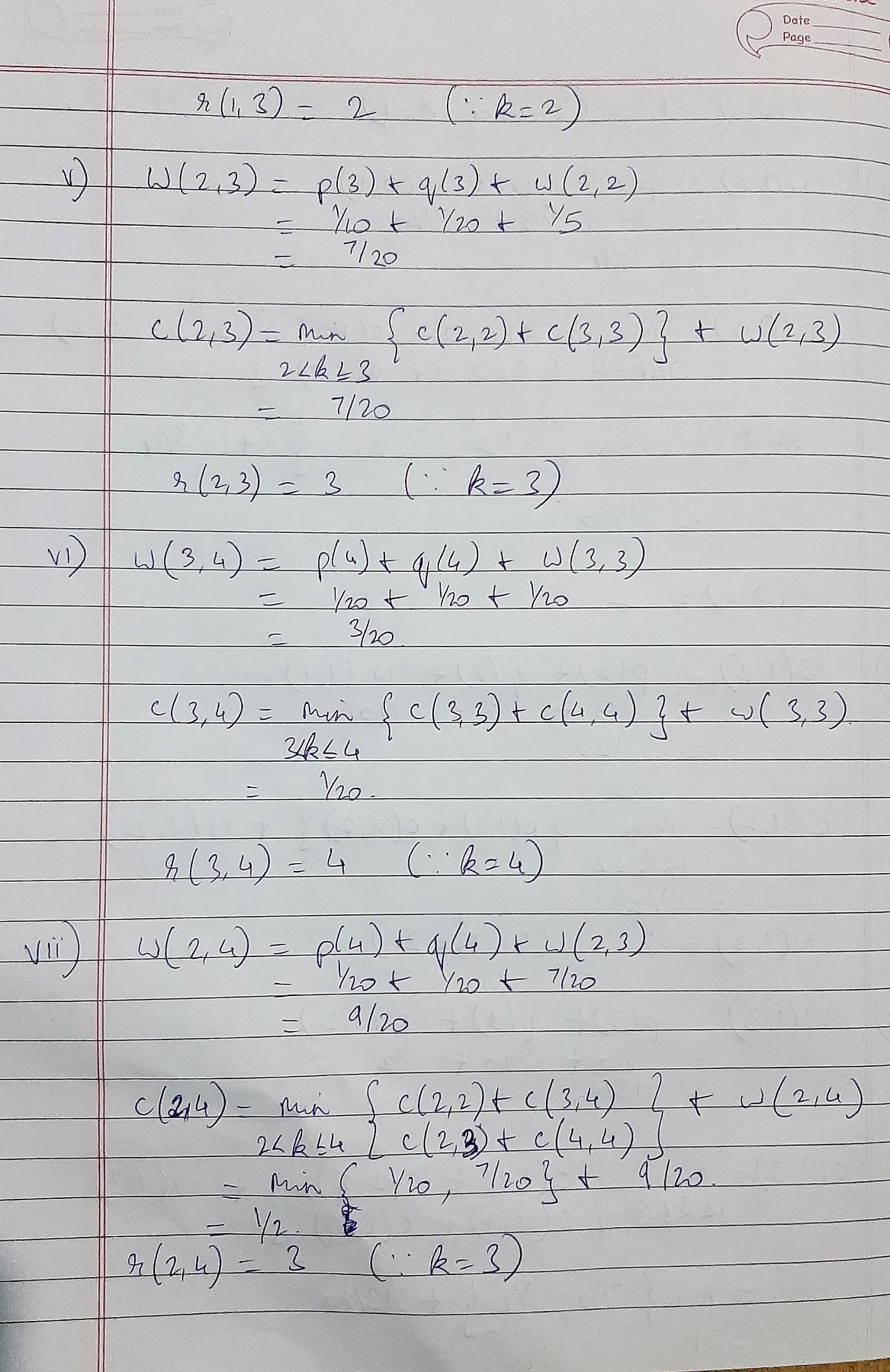
*Time Complexity*

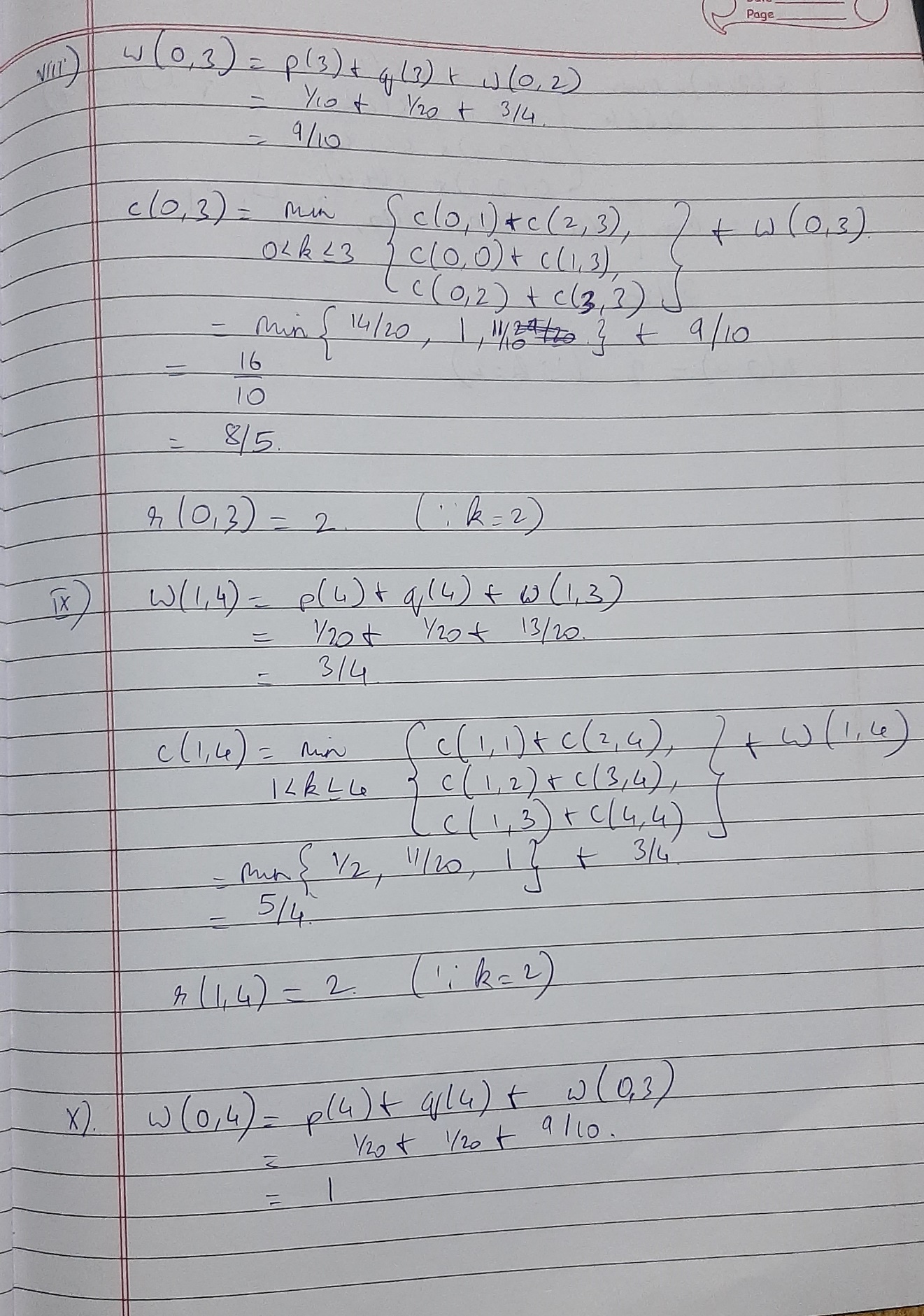
• The algorithm requires O () time, since two nested for loops are used. Each of these loops takes on at most n values.

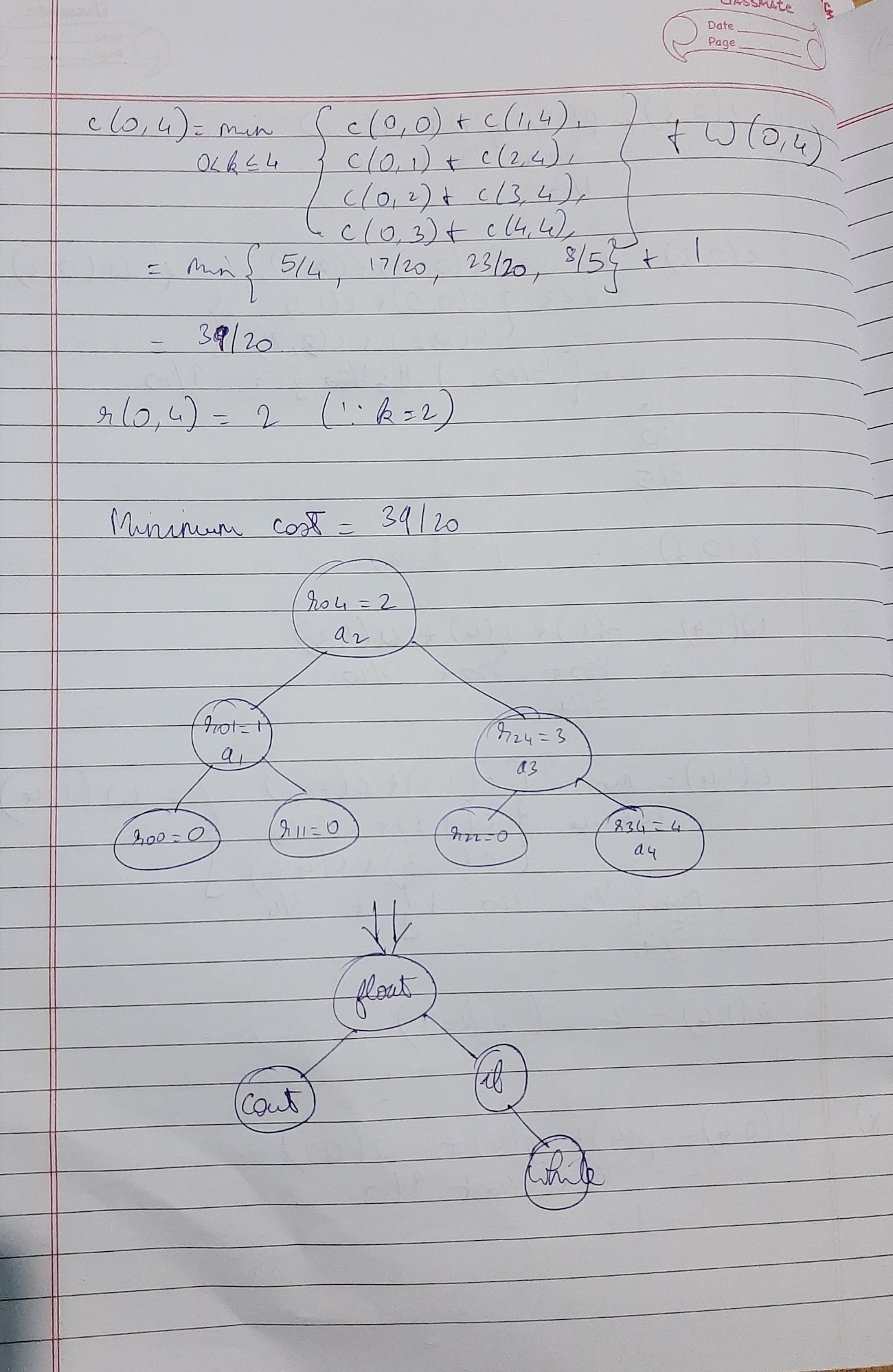
*Problem Tracing*











PROGRAM IMPLEMENTATION:

#include<iostream>

#include<iomanip>

using namespace std;

static int ctr;

#define N INT\_MAX;

#define v 10

static float c[v][v], w[v][v];

static int r[v][v];

int n;

void tree(int r[v][v],int n,int i, int j)

{

ctr++;

int k= r[i][j]; ctr++;

ctr++;

if(i==j)

{

ctr++;

return;

}

cout<<"\n\nLeft Child of "<<k<<" : "<<r[i][k-1]; ctr++;

cout<<"\tRight Child of "<<k<<" : "<<r[k][j]; ctr++;

tree(r,n,i,k-1);

tree(r,n,k,j);

}

int Find(float c[v][v], int r[v][v], int i, int j)

{

ctr++;

float min = N; int index;

ctr++;

for(int m= i+1; m<=j; m++)

{

ctr++;

ctr++;

if((c[i][m-1] + c[m][j])<min)

{

ctr++;

min = c[i][m-1] + c[m][j]; ctr++;

index = m; ctr++;

}

}

ctr++;

return index;

}

void OBST(float p[v], float q[v], int n)

{

ctr++;

int j, k;

//initialise

ctr++;

for(int i=0; i<n; i++)

{

ctr++;

w[i][i] = q[i]; ctr++;

r[i][i] = 0; c[i][i]=0.0; ctr++;

//for optimal trees with 1 node

w[i][i+1] = q[i+1] + p[i+1] + q[i]; ctr++;

c[i][i+1] = q[i+1] + p[i+1] + q[i]; ctr++;

r[i][i+1] = i+1; ctr++;

}

w[n][n]= q[n]; ctr++;

c[n][n]= 0.0; ctr++;

r[n][n]= 0; ctr++;

//for optimal tree with 'm' nodes

ctr++;

for(int m=2; m<=n; m++)

{

ctr++;

ctr++;

for(int i=0; i<=n-m; i++)

{

ctr++;

j=i+m; ctr++;

w[i][j]= w[i][j-1] + p[j] + q[j]; ctr++;

k = Find(c, r, i, j); ctr++;

c[i][j] = w[i][j] + c[i][k-1] + c[k][j]; ctr++;

r[i][j]=k; ctr++;

}

}

//printing table

cout<<"\n";ctr++;

cout<<" |";

for (int i = 0; i <= n; i++){

cout<<setw(13)<<i<<" |";ctr++;

cout<<" ";

}

ctr++;

cout<<"\n"<<"---------------------------------------------------------------------------------"<<"\n";

for(int m=0;m<=n;m++){

ctr++;

cout<<m<<"|";ctr++;

for(int i=0,j=m;i<=n && j<=n;i++,j++){

ctr++;

cout<<"w(" << i << "," << j << ")= " << w[i][j]<<"\t"<<"|";ctr++;

}

ctr++;

cout<<"\n";ctr++;

cout<<" |";ctr++;

for(int i=0,j=m;i<=n && j<=n;i++,j++){

ctr++;

cout<<"c(" << i << "," << j << ")= " << c[i][j]<<"\t"<<"|";ctr++;

}

ctr++;

cout<<"\n";ctr++;

cout<<" |";ctr++;

for(int i=0,j=m;i<=n && j<=n;i++,j++){

ctr++;

cout<<"r(" << i << "," << j << ")= " << r[i][j]<<"\t"<<"|";ctr++;

}

ctr++;

cout<<"\n"<<"---------------------------------------------------------------------------------\n";ctr++;

}

ctr++;

cout<<"\nMinimum Cost: "<<c[0][n];ctr++;

cout<<"\nRoot of OBST: "<<r[0][n];ctr++;

tree(r,n,0,n);ctr++;

cout<<"\n\nNOTE: 0 == NULL"<<endl; ctr++;

}

int main()

{

ctr++;

cout << "Enter number of identifiers: ";ctr++;

cin >> n;ctr++;

float p[n + 1], q[n + 1];

p[0]=-1;ctr++;

cout << "\nEnter probabilities of successful search(p): ";ctr++;

for (int i = 1; i <= n; i++)

{

ctr++;

cin >> p[i];ctr++;

}

ctr++;

cout << "\nEnter probabilities of unsuccessfull search(q): ";ctr++;

for (int i = 0; i <= n; i++)

{

ctr++;

cin >> q[i];ctr++;

}

OBST(p, q, n);ctr++;

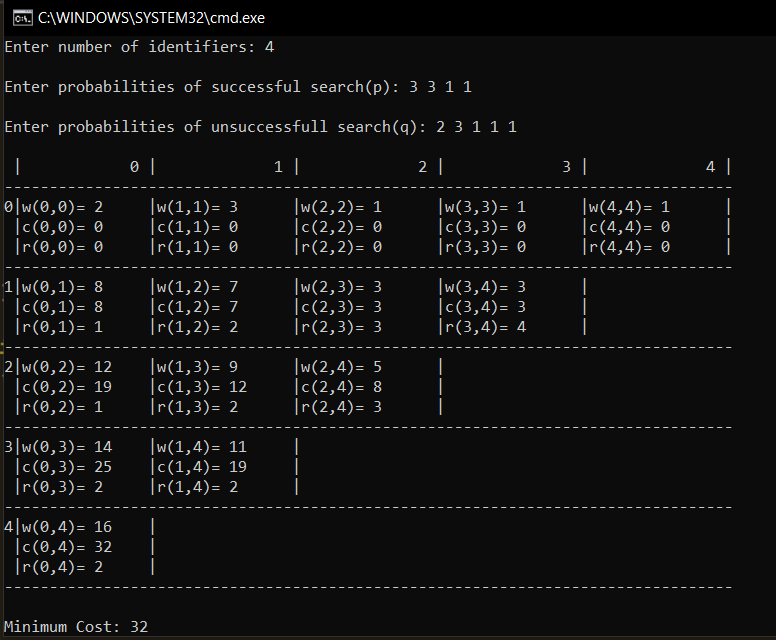
cout<<"\n\nStepcount: "<<ctr<<endl;

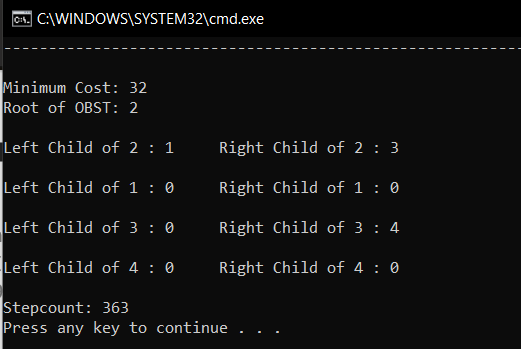
}

OUTPUTS:

1. When n=4

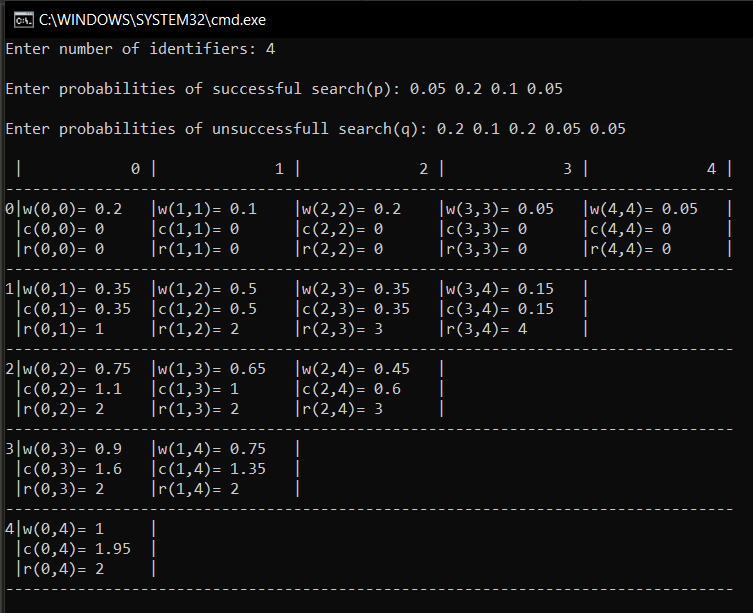
**Count=363**

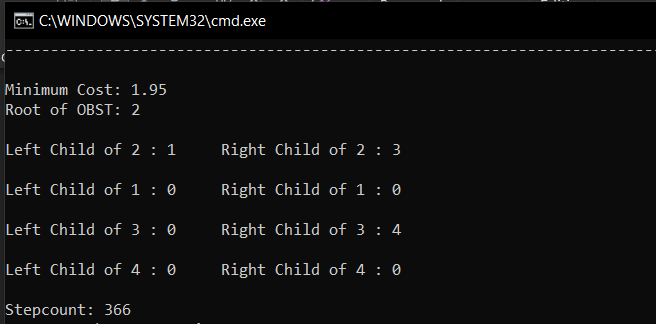




1. When n=4 (with fractional values)

**Count=366**

****

****

**Conclusion**:

* **If we know the probability or frequency of nodes in the tree we can find**

**Optimal Binary Search Tree Using Dynamic Programming.**

* **Time complexity is found to be O (n2**