**AIM** :

To find the minimal cost of Optimal Binary Search Tree.

**DESCRIPTION** :

The problem has optimal substructure. We have seen that the problem can be broken down into smaller subproblems, which can further be broken down into yet smaller subproblems, and so on. The problem also exhibits overlapping subproblems, so we will end up solving the same subproblem over and over again. We know that problems with optimal substructure and overlapping subproblems can be solved by dynamic programming, where subproblem solutions are memoized rather than computed and again. With the help of optimal binary search tree we can build a tree which consists of nodes that are frequently accessed with the help of their respective frequencies.

So that the searching time decreases further than what we get from normal binary search tree.

**Algorithm** :

Algorithm(freq,n)

//freq is an array consisting of frequencies and n is the number of elements

{

for i:=0 to n do cost[i][i]:=0

for L:=2 to n do

{

for i:=0 to n-L+1 do

{

j=n-L+1

cost[i][j]=INT\_MAX

for r:=i to j do

{

c=((r > i) ? cost[i][r - 1] : 0) + ((r < j) ? cost[r + 1][j] : 0) + sum(freq, i, j);

if (c < cost[i][j]) then cost[i][j] = c;

}

}

}

return c[0][n-1]

}

**Code** :

#include <bits/stdc++.h>

using namespace std;

int sum(int freq[], int i, int j);

int optimalSearchTree(int freq[], int n)

{

int cost[n][n];

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

cost[i][i] = freq[i];

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

{

for (int i = 0; i <= n - L + 1; i++)

{

int j = i + L - 1;

cost[i][j] = INT\_MAX;

for (int r = i; r <= j; r++)

{

int c = ((r > i) ? cost[i][r - 1] : 0) +

((r < j) ? cost[r + 1][j] : 0) +

sum(freq, i, j);

if (c < cost[i][j])

cost[i][j] = c;

}

}

}

return cost[0][n - 1];

}

int sum(int freq[], int i, int j)

{

int s = 0;

for (int k = i; k <= j; k++)

s += freq[k];

return s;

}

int main()

{

int n;

cout<<"Enter number of keys : ";

cin>>n;

int freq[n];

cout<<"Enter frequencies : ";

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

cin>>freq[i];

}

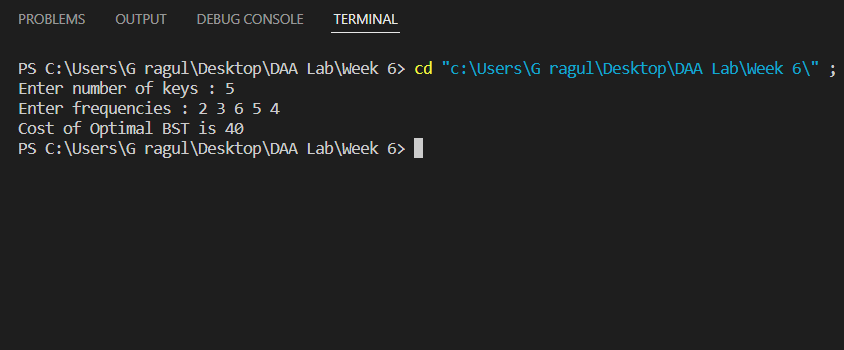
int num = sizeof(freq) / sizeof(freq[0]);

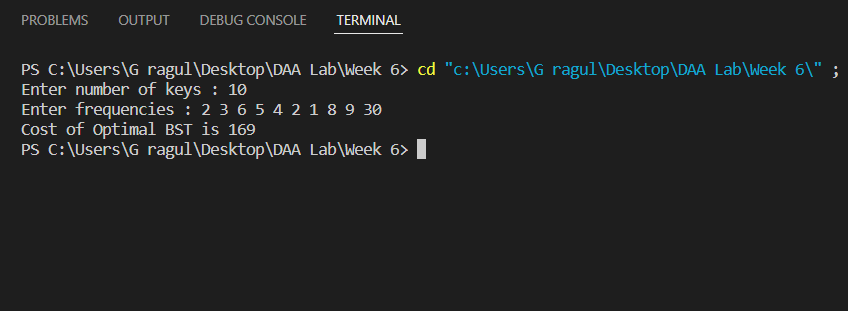
cout << "Cost of Optimal BST is " << optimalSearchTree(freq, num);

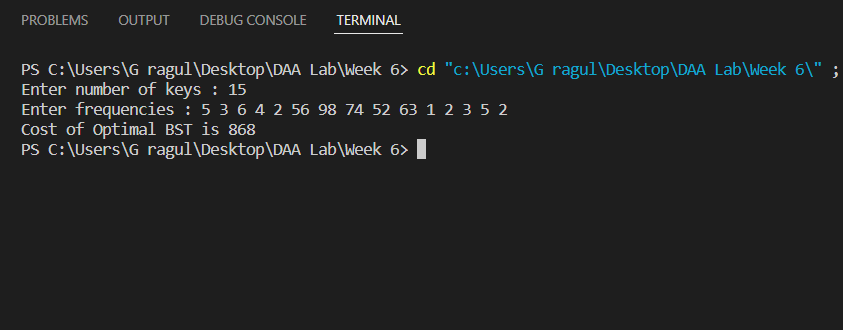
return 0;

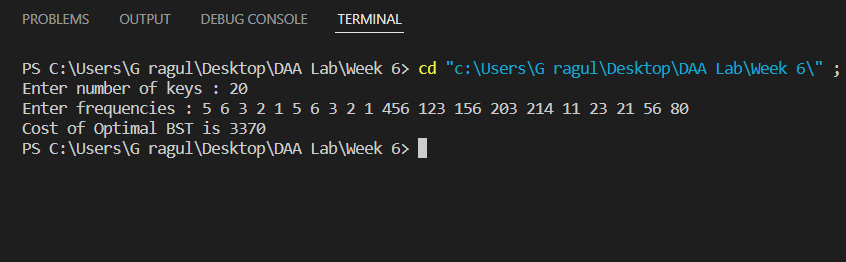
}

**RESULT ANALYSIS** :









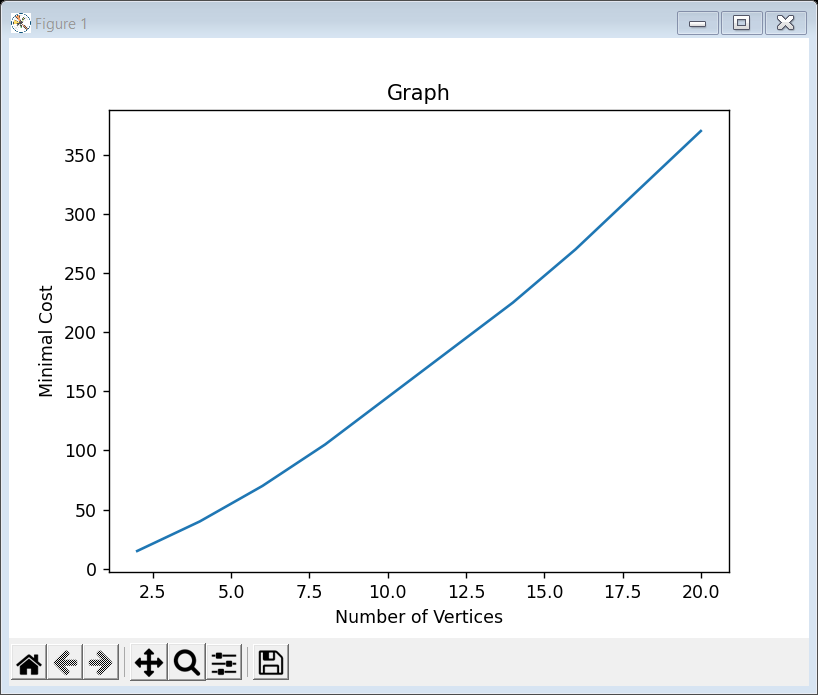
**Time Complexity** :

The above code has a time complexity of O(n^3) because of 3 inner for-loops.

**Space Complexity** :

O(n^2) in order to store the costs in a 2D array.

**Graph** :



**Conclusion** :

As the number of inputs increases the time taken by the code to get executed increases.

Also, if we keep the frequencies of the node constant and increase the vertices, the minimal cost of OBST increases.

**References** :

* Geeks for Geeks