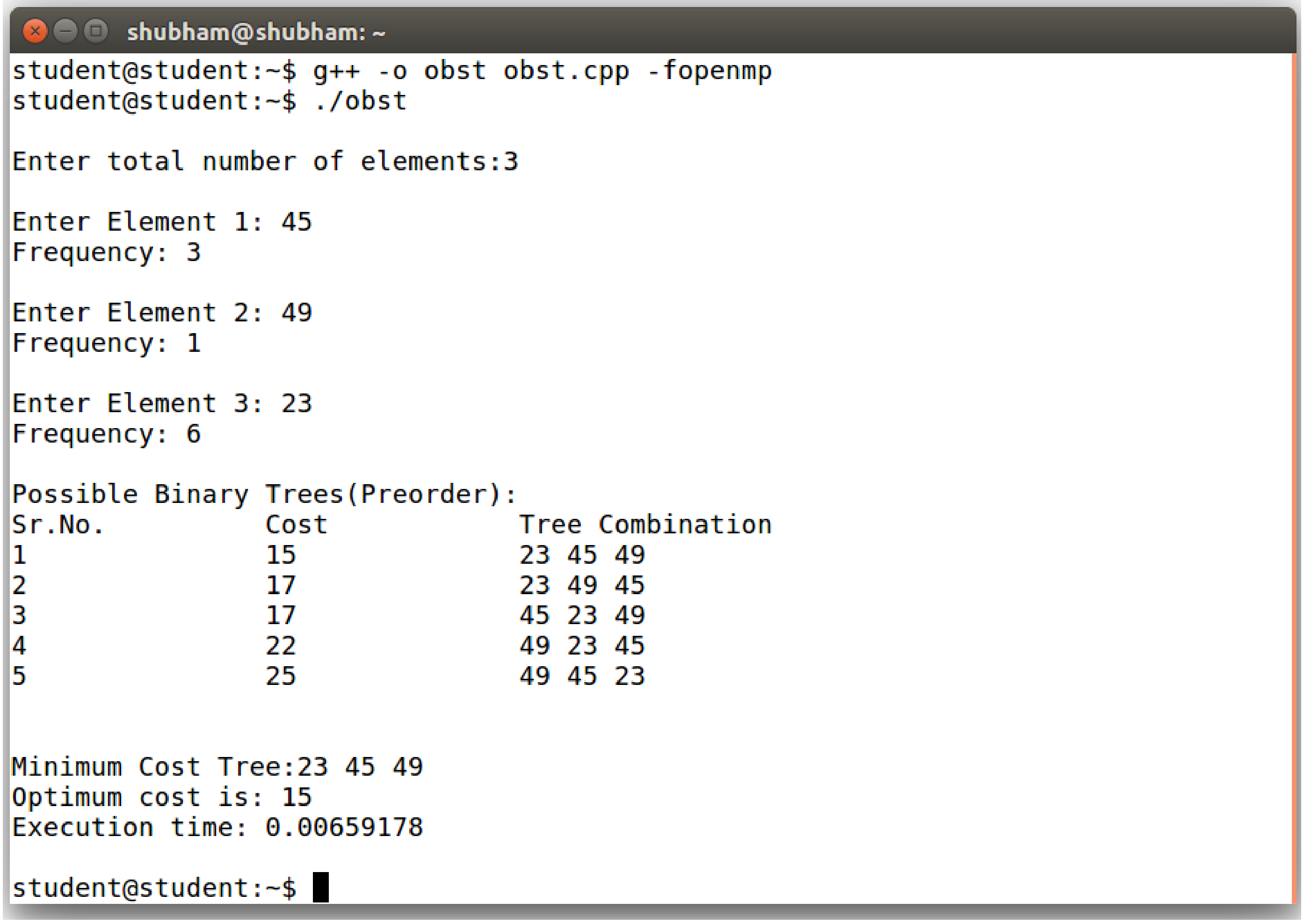
File: obst.cpp

```
#include <iostream>
#include <omp.h>
#include<vector>
using namespace std;

struct Node { //structure of node
     int key;
     int frequency;
     struct Node *left;
     struct Node *right;
};

class BST {
     int n;
     int input_arr[40];
     int frequency_array[40];
     int cost_per_tree[40];
     int cost,level;
     vector<Node *> totalTrees;

     public:
          void input();
          struct Node *newnode(int item, int frequency);
          void preorder(Node *root);
          vector<Node *> constructTrees(int arr[], int
frequency_array[], int start, int end);
          void create_tree();
          void display();
          int computeCost(Node *temp, int level);
};

void BST :: input() {
     cout<<"\nEnter total number of elements:";
     cin>>n;
     for(int i=0;i<n;i++) {
          cout<<"\nEnter Element "<<i+1<<": ";
          cin>>input_arr[i];
          cout<<"Frequency: ";
          cin>>frequency_array[i];
     }
     /*Sorting the input in ascending order of elemnts*/
     int temp1,temp2;
     for(int i=0;i<n;i++) {
          for(int j=0;j<n-1;j++) {
               if(input_arr[j] > input_arr[j+1]) {
                    //Swapping Logic=== to eliminate same trees
                    temp1 = input_arr[j];
                    input_arr[j] = input_arr[j+1];
                    input_arr[j+1] = temp1;
                    temp2 = frequency_array[j];
                    frequency_array[j] = frequency_array[j+1];
                    frequency_array[j+1]=temp2;
               }
          }
     }

}
```

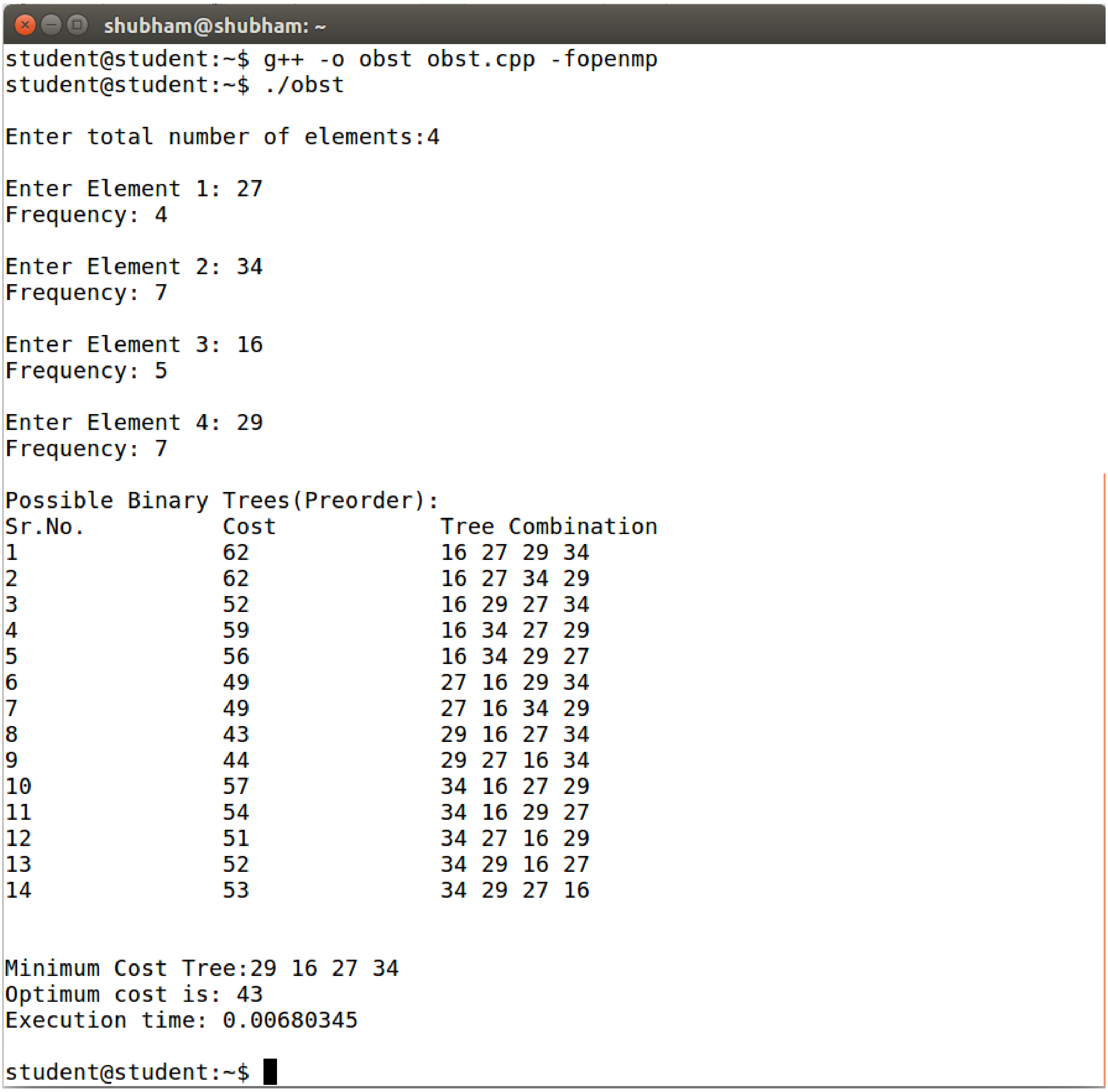
```
struct Node * BST :: newnode(int item, int frequency) {
     struct Node *temp = new Node;
     temp->key = item;
     temp->frequency = frequency;
     temp->left = temp->right = NULL;
     return temp;
}

int BST::computeCost(Node *temp, int level) {
     if (temp != NULL) {
          cost=cost + level*temp->frequency;
          computeCost(temp->left,level+1);
          computeCost(temp->right,level+1);
     }
     return cost;
}

void BST :: preorder(Node *root) {
     if (root != NULL) {
          cout<<root->key<<" ";
          preorder(root->left);
          preorder(root->right);
     }
}

vector<Node *> BST :: constructTrees(int arr[],int frequency_array[],
int start, int end) {
     // List to store all possible trees
     vector<Node *> trees;
     vector<Node *> lefttrees;
     vector<Node *> righttrees;
     /* if start > end then subtree will be empty so returning NULL in
the list */
     if (start > end) {
          trees.push_back(NULL);
          return trees;
     }

     for (int i = start; i <= end; i++) {
          #pragma omp parallel sections
          {
               #pragma omp section
               {
                    /* Constructing left subtree */
                    lefttrees = constructTrees(arr, frequency_array,
start, i-1);
               }

               #pragma omp section
               {
                    /* Constructing right subtree */
                    righttrees = constructTrees(arr, frequency_array,
i+1, end);
               }
          }
          for (int j = 0; j < lefttrees.size(); j++) {
               for (int k = 0; k < righttrees.size(); k++) {
                    // Making arr[i] as root
                    Node * Node = newnode(arr[i], frequency_array[i]);

                    // Connecting left subtree
                    Node->left = lefttrees[j];

                    // Connecting right subtree
                    Node->right = righttrees[k];

                    // Adding this tree to list
                    trees.push_back(Node);
               }
          }
     }
     return trees;
}

void BST :: create_tree() {
     totalTrees = constructTrees(input_arr, frequency_array, 0, n-1);
}

void BST :: display() {
     int cost_array[40];
     int c;
     cout<<"\nPossible Binary Trees(Preorder):";   //display all
possible combinations
     cout <<"\nSr.No.\t\tCost\t\tTree Combination\n";
     for (int i = 0; i < totalTrees.size(); i++) {
          cost=0;
          level=1;
          c = computeCost(totalTrees[i],level);
          cout<<i+1<<"\t\t"<<c<<"\t\t";
          preorder(totalTrees[i]);
          cost_array[i] = c;

          cout<<endl;
     }
     int x=cost_array[0],index=0;
        for(int j=0; j<totalTrees.size()-1;j++) { //get minimum cost
tree
          if(x>cost_array[j+1]) {
               x=cost_array[j+1];
               index=j+1;
          }
     }

     cout<<"\n\nMinimum Cost Tree:"; //display final optimal tree
        preorder(totalTrees[index]);
     cout<<"\nOptimum cost is: "<<cost_array[index]<<"\n";
}
```

```
int main() {
     BST b;
     b.input();

     double start = omp_get_wtime();

     b.create_tree();

     double end = omp_get_wtime();

     b.display();
     cout<<"Execution time: "<<end-start<<"\n\n";
     return 0;
}
```

#OUTPUT:



```
student@student:~$ g++ -o obst obst.cpp -fopenmp
student@student:~$ ./obst

Enter total number of elements:3

Enter Element 1: 45
Frequency: 3

Enter Element 2: 49
Frequency: 1

Enter Element 3: 23
Frequency: 6

Possible Binary Trees(Preorder):
Sr.No.          Cost            Tree Combination
1               15              23 45 49
2               17              23 49 45
3               17              45 23 49
4               22              49 23 45
5               25              49 45 23


Minimum Cost Tree:23 45 49
Optimum cost is: 15
Execution time: 0.00659178

student@student:~$ █
```



```
student@student:~$ g++ -o obst obst.cpp -fopenmp
student@student:~$ ./obst

Enter total number of elements:4

Enter Element 1: 27
Frequency: 4

Enter Element 2: 34
Frequency: 7

Enter Element 3: 16
Frequency: 5

Enter Element 4: 29
Frequency: 7

Possible Binary Trees(Preorder):
Sr.No.          Cost            Tree Combination
1               62              16 27 29 34
2               62              16 27 34 29
3               52              16 29 27 34
4               59              16 34 27 29
5               56              16 34 29 27
6               49              27 16 29 34
7               49              27 16 34 29
8               43              29 16 27 34
9               44              29 27 16 34
10              57              34 16 27 29
11              54              34 16 29 27
12              51              34 27 16 29
13              52              34 29 16 27
14              53              34 29 27 16


Minimum Cost Tree:29 16 27 34
Optimum cost is: 43
Execution time: 0.00680345

student@student:~$ 
```