File: bst.cpp

```
#include<stdlib.h>
#include<iostream>
#include<mpi.h>
#include<cmath> //ceil function
using namespace std;

struct node {
     int data;
     struct node *left,*right;
};

struct node *leaf(int d) {
    struct node *temp =  (struct node *)malloc(sizeof(struct node));
    temp->data = d;
    temp->left = temp->right = NULL;
    return temp;
}

struct node *create(struct node *n,int data) {
     if(n==NULL) {
          return leaf(data);
     }
     if(n->data > data) {
          n->left = create(n->left,data);
     }
     else if(n->data < data) {
          n->right = create(n->right,data);
     }
     return n;
}

int search(struct node *root,int data) {
     int flag=0;
     while(root!=NULL && flag==0) {
          if(root->data==data) {
               flag=1;
               return 1;
          }
          else {
               if(root->data<data) {
                    root=root->right;
               }
               else {
                    root=root->left;
               }
          }
     }
     return 0;
}

int main(int argc,char* argv[]) {
     int rank,nproc;
     MPI_Status stat;

     MPI_Init (&argc, &argv);
     MPI_Comm_rank (MPI_COMM_WORLD, &rank);
     MPI_Comm_size (MPI_COMM_WORLD, &nproc);
```

```
     int d[10],ans=1;
     struct node *root = NULL;
     int n;
     int no;
     int searched[10];
     int se[5];

     //accepts required data
     if(rank==0) {
          cout<<"Enter Number of elements:";
          cin>>n;
          int c=0;
          cout<<"\nEnter "<<n<<" Elements: ";
          do {
               cin>>d[c];
               c++;
          }while(c!=n);

          cout<<"\nEnter Number of Elements to be Searched: ";
          cin>>no;

          cout<<"\nEnter Elements to be Searched: ";
          for(int i=0;i<no;i++) {
               cin>>searched[i];
          }
     }

     //Broadcast data
     MPI_Bcast(&no,1,MPI_INT,0,MPI_COMM_WORLD);
     MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
     MPI_Bcast(d,n,MPI_INT,0,MPI_COMM_WORLD);

     int count=ceil((float)no/nproc);

     //divide the elements to be searched on available no of cores

MPI_Scatter(searched,count,MPI_INT,se,count,MPI_INT,0,MPI_COMM_WORLD);

     //Tree creation
     int c=0;
     do {
          root = create(root,d[c]);
          c++;
     }while(c!=n);

     int temp = no % nproc;
     if(rank >= temp)
          count--;
     //Searching element
     int l;
     for(int k=0; k<count; k++) {
          l=0;
          l=search(root,se[k]);
          if(l==1) {
               cout<<"\nProcessor "<<rank<<": Found Data =
"<<se[k]<<endl;
          }
          else {
```

```
                cout<<"\nProcessor "<<rank<<": Data = "<<se[k]<<", Not
Found\n";
            }
        }

        MPI_Finalize();
        return 0;
}
```

#OUTPUT: