```
1: //
   Assignment no-5
2: // Name- Prerana Rajesh Gajare      Class-SEIT
    RollNo-SI41
3: /*PROBLEM STATEMENT:-
4:         Implement binary search tree and perform
   the following operations:
5:             a) Insert (Handle insertion of
   duplicate entry)
6:             b) Delete
7:             c) Search
8:             d) Display tree (Traversal)
9: */
10: //Source Code:-
11: #include <iostream>
12: using namespace std;
13:
14: //Class bst
15: class bst
16: {
17:     int data;//Integer data
18:     bst *left;//Left address pointer
19:     bst *right;//Right address pointer
20:     public:
21:         //Function declaration
22:         bst* create(int);
23:         int search(bst*, int);
24:         bst* insert(bst*,bst*);
25:         bst* findMaximum(bst*);
26:         bst* del(bst*,int);
27:         void inorder(bst*);
28: };
29:
30: // To create a node for accepted data item
```

```
31: bst* bst::create(int item)
32: {
33:     bst *newnode = new bst;//Create newnode using
    new keyword
34:     newnode->data = item;//Store key value in
    newnode of data
35:     newnode->left = NULL;//Initialize left and
    right pointer as null
36:     newnode->right = NULL;
37:     return newnode;
38: }
39:
40: // To search an element(key) in tbt
41: int bst::search(bst *root, int key)
42: {
43:     bst *curr;
44:     curr = root;//Store root value in curr
45:     // If the element to be search is curr(root)
    itself, return 1.
46:     if(key == curr->data)
47:         return 1;
48:     else
49:     {
50:         //While we reach to the leaf node
51:         while(curr!=NULL)
52:         {
53:             //If accepted element is less than
    curr of data,move towards left side
54:             if(key < curr->data)
55:             {
56:                 curr = curr->left;
57:             }
58:             //If accepted element is greater than
    curr of data,move towards right side
```

```
59:             else if(key > curr->data)
60:             {
61:                 curr = curr->right;
62:             }
63:             else
64:             {
65:                 return 1;
66:             }
67:         }
68:         //If tree is empty ,return 0
69:         if(curr == NULL)
70:         {
71:             return 0;
72:         }
73:     }
74: }
75:
76: //To perform insertion in tbt
77: bst* bst::insert(bst*root, bst*parent)
78: {
79:     bst *newnode;
80:     bst *curr;
81:     int key, valid;
82:     //Accepting the data to be inserted
83:     cout<<"\nEnter the element:";
84:     cin>>key;
85:     newnode = create(key);//Create node for
    accepted element and store it in newnode
86:     //If tree is empty,set newnode as root
87:     if(root == NULL)
88:         root = newnode;
89:     else
90:     {
91:         curr = root;//Store root value in curr
```

```
 92:         valid = search(root, key);//To search
     wheather element is already present or not in the
     tree
 93:         if(valid == 0)
 94:         {
 95:             //While we reach tothe leaf node
 96:             while(curr!=NULL)
 97:             {
 98:                 //Store curr vlaue in parent
 99:                 parent = curr;
100:
101:                 //If accepted element is less than
     curr of data,move towards left side
102:                 if(key < curr->data)
103:                     curr = curr->left;
104:                 else
105:                     //move towards right side
106:                     curr = curr->right;
107:             }
108:                 //If accepted element is less than
     parent of data
109:                 if(key < parent->data)
110:                     parent->left = newnode;//Store
     the value of newnode in parent's left side
111:                 else
112:                     parent->right =
     newnode;                                        //Store the v
     right side
113:         }
114:         else
115:         {
116:             //If element to be inserted is already
     present
117:             cout<<"\nDuplicate Entry";
```

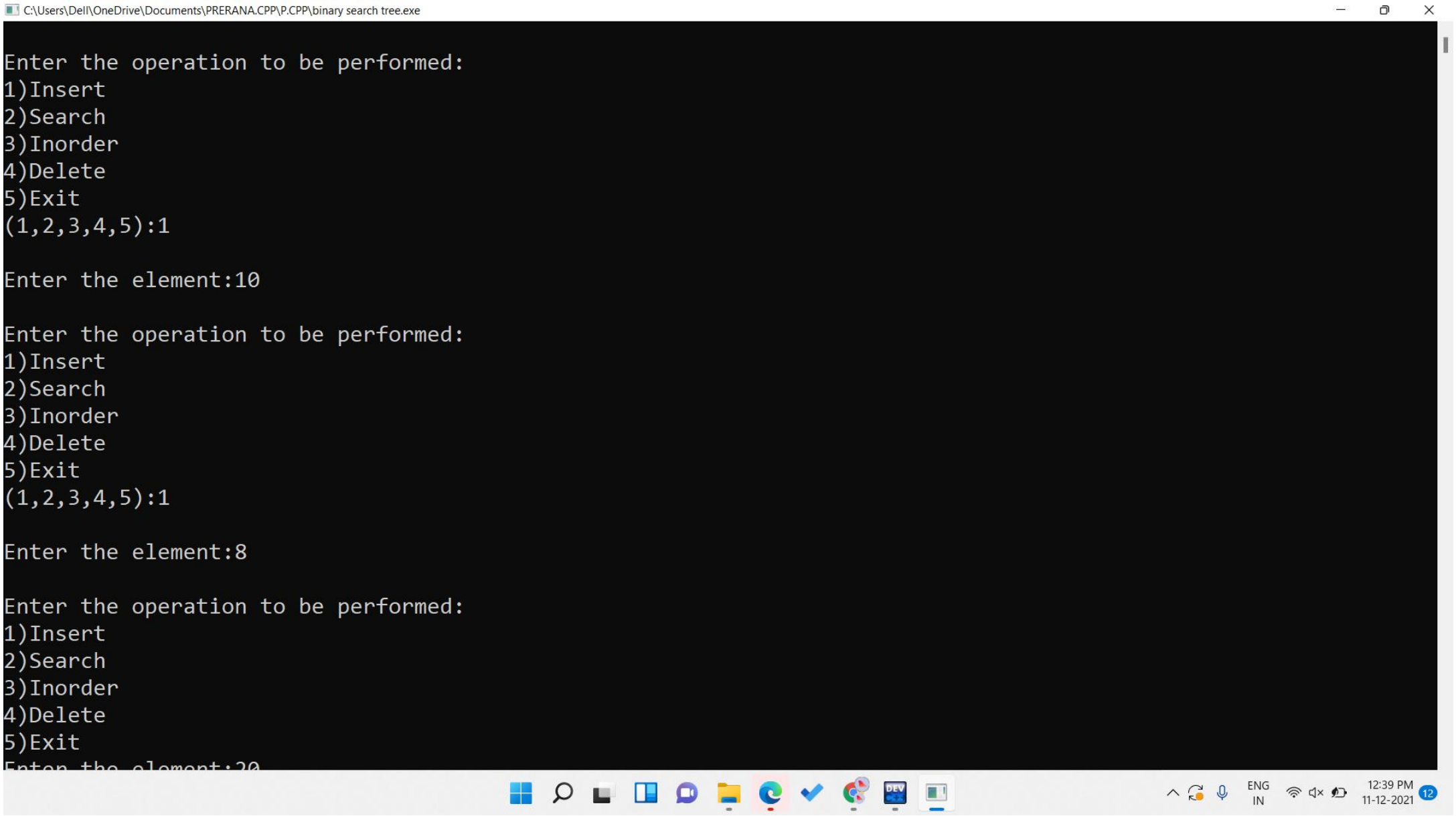
```
118:                 return root;
119:             }
120:         }
121:         return root;
122: }
123:
124: //To find maximum element in left sub tree
125: bst* bst :: findMaximum(bst* curr)
126: {
127:         curr = curr->left;
128:         while(curr->right != NULL) {
129:             curr = curr->right;
130:         }
131:         return curr;
132: }
133:
134: //To perform inorder traversal of bst
135: void bst::inorder(bst*root)
136: {
137:
138:         if(root == NULL)
139:             return;
140:         inorder(root->left);
141:         cout<<"\t"<<root->data;
142:         inorder(root->right);
143: }
144:
145: bst* bst::del(bst*root,int key)
146: {
147:         if(root ==NULL)//if tree is empty
148:         {
149:             cout<<"\nElement not found";
150:             return root;
151:         }
```

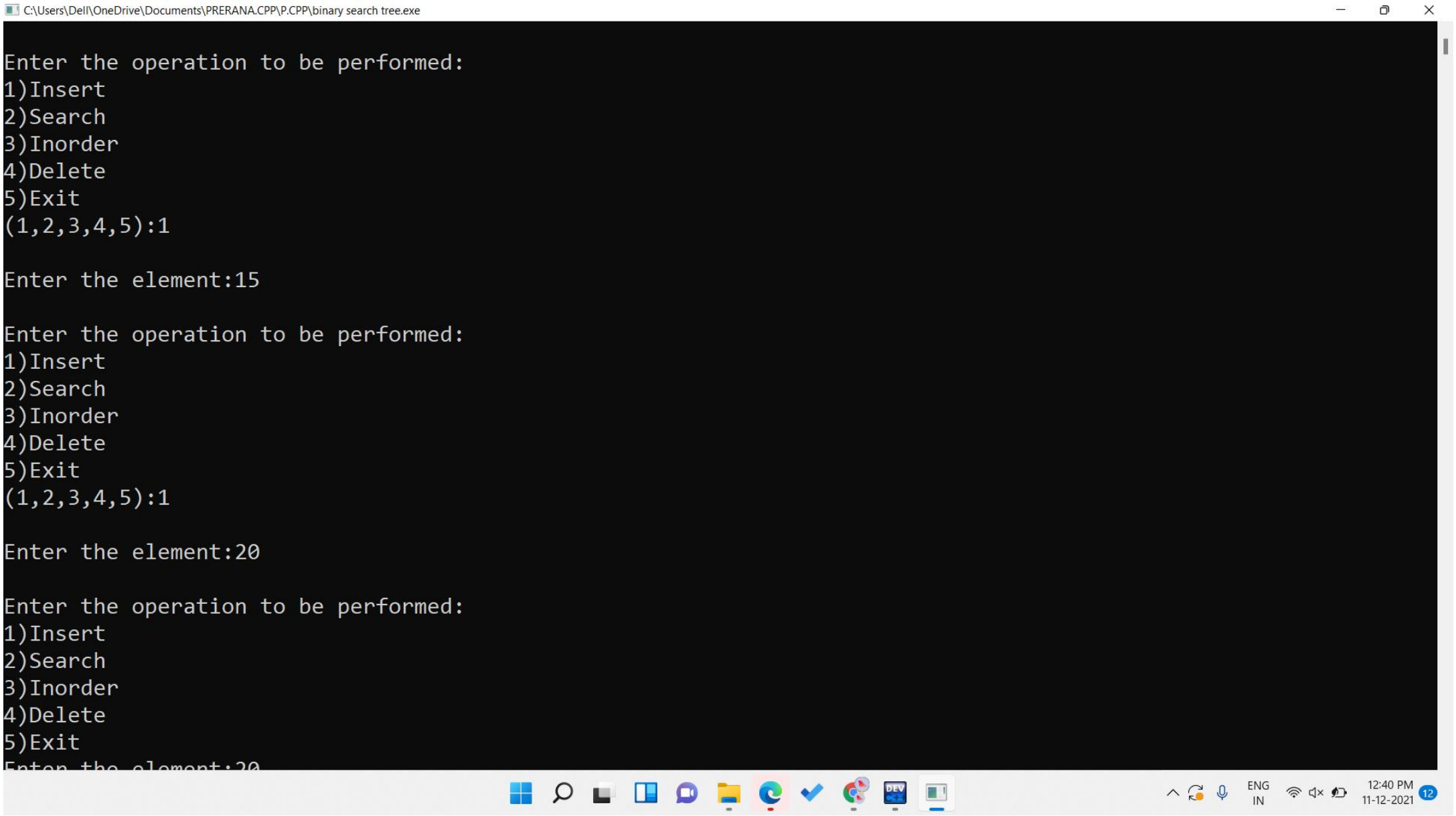
```
152:     else if(key <root->data)
153:     {
154:         root->left=del(root->left,key);//delete
     operation will return the modified address of the
155:         //root of the left sub tree and store  in
     root of left
156:         return root;
157:     }
158:     else if(key >root->data)
159:     {
160:         root->right=del(root->right,key);//delete
     operation will return the modified address of the
161:         // root of the right sub tree and store in
     root of right
162:         return root;
163:     }
164:     else
165:     {
166:         //leaf node case
167:         if(root->left == NULL && root->right==
     NULL)
168:         {
169:
170:             delete root;
171:             root= NULL;
172:
173:         }
174:         //1 children
175:         else if(root->left ==NULL)
176:         {
177:             bst *temp= root;//Store the element to
     be deleted in temp
178:             root=root->right;//Store right child
     of root in root
```

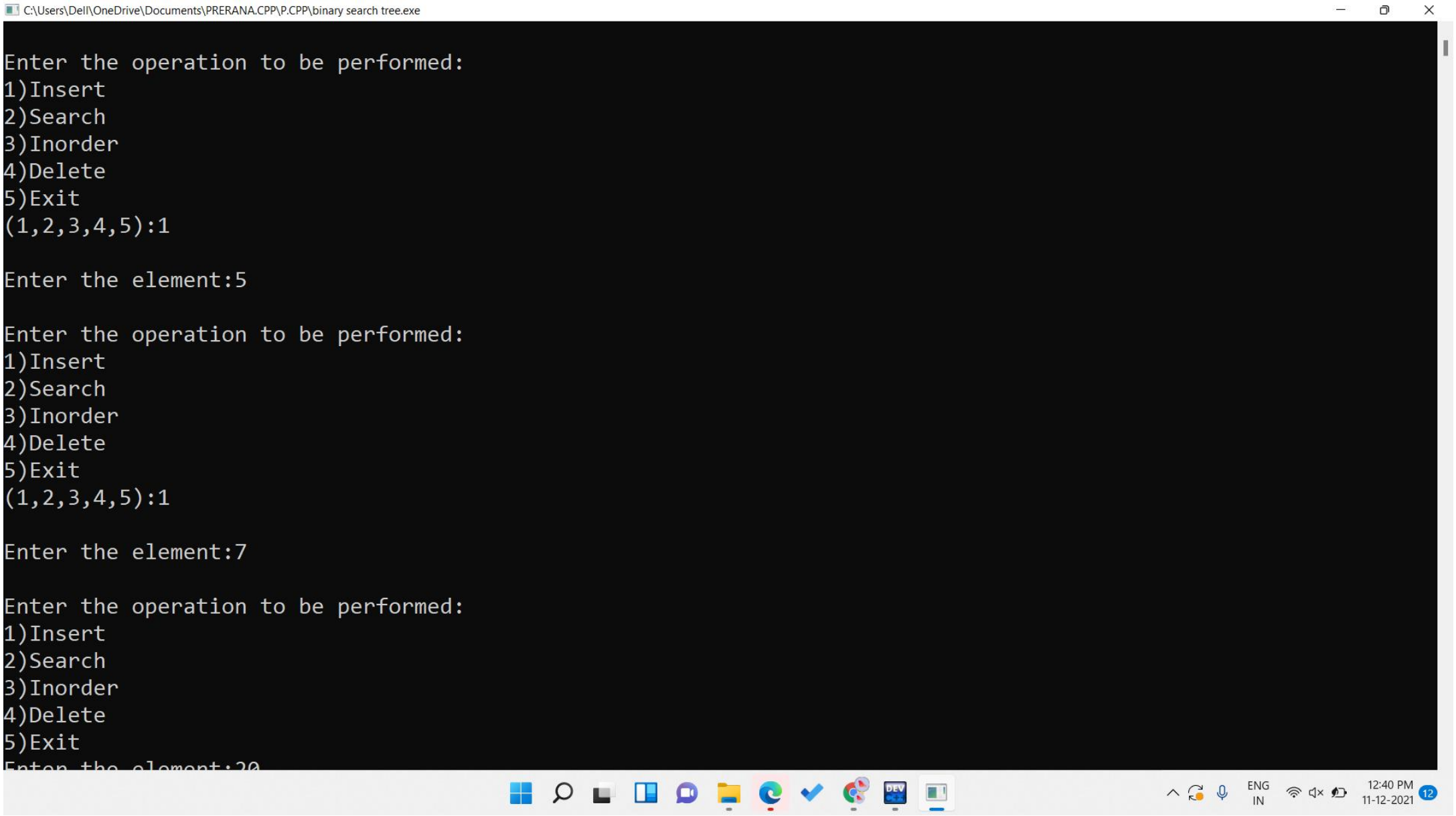
```
179:                delete temp;
180:
181:            }
182:            else if(root->right== NULL)
183:            {
184:                bst *temp=root;//Store the element to
    be deleted in temp
185:                root=root->left;//Store left child of
    root in root
186:                delete temp;
187:            }
188:            //2 children
189:            else
190:            {
191:                bst *temp=findMaximum(root);//Find
    maximum element in left subtree and store in temp
192:                root->data=temp->data;//Store temp
    value in root
193:                root->left=del(root->left,temp-
    >data);                                          //Recur
    temp value
194:
195:            }
196:
197:        }
198:        return root;
199:
200: }
201:
202:
203: int main()
204: {
205:     bst t;
206:     int l, key, valid;
```

```
207:     bst *root = NULL;//Initiaze  root value as null
208:     bst *parent;
209:         do
210:     {
211:         cout<<"\nEnter the operation to be
    performed:\n1)Insert\n2)Search\n3)Inorder\n4)Delete
    2,3,4,5):";
212:         cin>>l;
213:         switch(l)
214:         {
215:             case 1:
216:                 root = t.insert(root,
    parent);//Calling insert function and store in root
217:                 break;
218:             case 2:
219:                 cout<<"\nEnter the element to
    search: ";
220:                 cin>>key;
221:                 valid = t.search(root, key);//Call
    Search function
222:                 if(valid == 0)
223:                     cout<<"\nElement not found";
224:                 else
225:                     cout<<"\nElement found";
226:                 break;
227:             case 3:
228:                 t.inorder(root);//Calling inorder
    function
229:                 break;
230:             case 4:
231:                 cout<<"\nEnter the element to
    delete: ";
232:                 cin>>key;
233:                 t.del(root,key);//Calling delete
    function
```

```
234:                 break;
235:             case 5:
236:                     cout<<"\nThe end";
237:                     break;
238:             default :
239:                 cout<<"\nWrong choice";
240:         }
241:     }while(l!=5);
242: }
```


C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):1
Enter the element:10
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):1
Enter the element:8
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
ENG
IN
12:39 PM
11-12-2021


C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):1
Enter the element:15
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):1
Enter the element:20
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
ENG
IN
12:40 PM
11-12-2021


C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):1
Enter the element:5
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):1
Enter the element:7
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
ENG
IN
12:40 PM
11-12-2021

C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe

```
Enter the element:17

Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):2

Enter the element to search: 20

Element found
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):2

Enter the element to search: 24

Element not found
Enter the operation to be performed:
1)Insert
2)Search
Enter the element:20
```

C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe

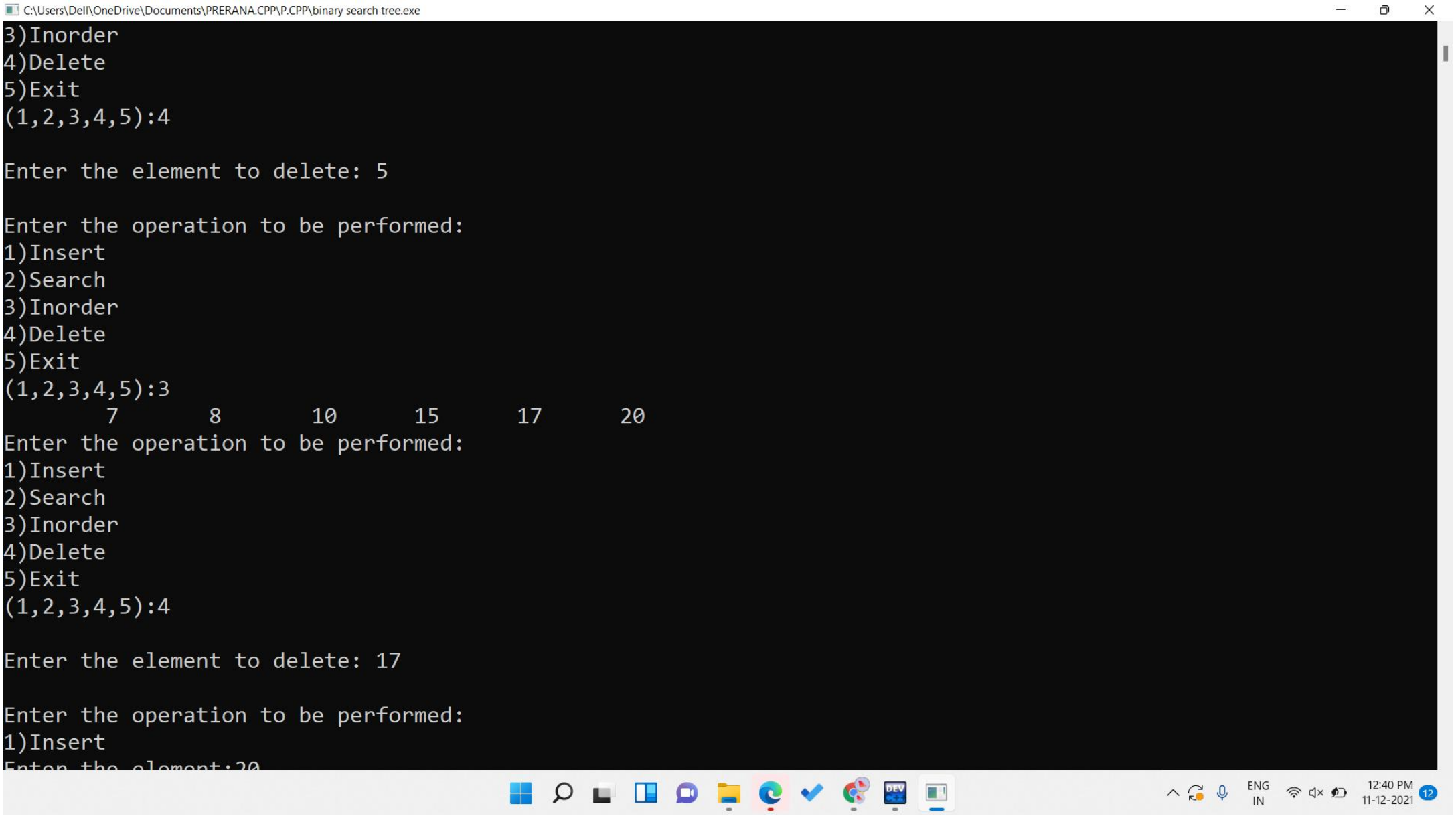
```
(1,2,3,4,5):2

Enter the element to search: 24

Element not found
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):3
       5       7       8       10      15      17      20
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):4

Enter the element to delete: 5

Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
Enter the element:20
```


C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):4
Enter the element to delete: 5
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):3
7 8 10 15 17 20
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):4
Enter the element to delete: 17
Enter the operation to be performed:
1)Insert
ENG
IN
12:40 PM
11-12-2021

C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe

```
5)Exit
(1,2,3,4,5):4

Enter the element to delete: 17

Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):3
       7       8       10      15      20
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):4

Enter the element to delete: 10

Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
Enter the element:20
```

C:\Users\Dell\OneDrive\Documents\PRERANA.CPP\P.CPP\binary search tree.exe

```
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):4

Enter the element to delete: 10

Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):3
        7       8       15      20
Enter the operation to be performed:
1)Insert
2)Search
3)Inorder
4)Delete
5)Exit
(1,2,3,4,5):5

The end
--------------------------------
Process exited after 126.5 seconds with return value 0
Enter the element:20
```