```cpp
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

class BTNode{

    private:
        int nodeid;
        int data;
        int levelNum;
        BTNode* leftChildPtr;
        BTNode* rightChildPtr;

    public:

        BTNode(){}

        void setNodeId(int id){
            nodeid = id;
        }

        int getNodeId(){
            return nodeid;
        }

        void setData(int d){
            data = d;
        }

        int getData(){
            return data;
        }

        void setLevelNum(int level){
            levelNum = level;
        }

        int getLevelNum(){
            return levelNum;
        }

        void setLeftChildPtr(BTNode* ptr){
            leftChildPtr = ptr;
        }

        void setRightChildPtr(BTNode* ptr){
            rightChildPtr = ptr;
        }

        BTNode* getLeftChildPtr(){
            return leftChildPtr;
        }

        BTNode* getRightChildPtr(){
            return rightChildPtr;
        }

        int getLeftChildID(){
            if (leftChildPtr == 0)
                return -1;

            return leftChildPtr->getNodeId();
        }
```

```
65
66          int getRightChildID(){
67              if (rightChildPtr == 0)
68                  return -1;
69
70              return rightChildPtr->getNodeId();
71          }
72  };
73
74
75
76  class BinarySearchTree{
77
78      private:
79          int numNodes;
80          BTNode* arrayOfBTNodes;
81          int rootNodeID;
82
83
84      public:
85
86          BinarySearchTree(int n){
87              numNodes = n;
88              arrayOfBTNodes = new BTNode[numNodes];
89
90              for (int index = 0; index < numNodes; index++){
91
92                  arrayOfBTNodes[index].setNodeId(index);
93                  arrayOfBTNodes[index].setLeftChildPtr(0);
94                  arrayOfBTNodes[index].setRightChildPtr(0);
95                  arrayOfBTNodes[index].setLevelNum(-1);
96
97              }
98          }
99
100
101         void setLeftLink(int upstreamNodeID, int downstreamNodeID){
102             arrayOfBTNodes[upstreamNodeID].setLeftChildPtr(&arrayOfBTNodes[
                downstreamNodeID]);
103         }
104
105         void setRightLink(int upstreamNodeID, int downstreamNodeID){
106             arrayOfBTNodes[upstreamNodeID].setRightChildPtr(&arrayOfBTNodes[
                downstreamNodeID]);
107         }
108
109
110         void selectionSort(int *array, int arraySize){
111
112             for (int iterationNum = 0; iterationNum < arraySize-1; iterationNum++){
113
114                 int minIndex = iterationNum;
115
116                 for (int j = iterationNum+1; j < arraySize; j++){
117
118                     if (array[j] < array[minIndex])
119                         minIndex = j;
120
121                 }
122
123                 // swap array[minIndex] with array[iterationNum]
124                 int temp = array[minIndex];
125                 array[minIndex] = array[iterationNum];
126                 array[iterationNum] = temp;
```

```
127
128             }
129
130         }
131
132         void constructBSTree(int* array){
133
134             int leftIndex = 0;
135             int rightIndex = numNodes-1;
136             int middleIndex = (leftIndex + rightIndex)/2;
137
138             rootNodeID = middleIndex;
139             arrayOfBTNodes[middleIndex].setData(array[middleIndex]);
140
141             ChainNodes(array, middleIndex, leftIndex, rightIndex);
142
143         }
144
145
146         void ChainNodes(int* array, int middleIndex, int leftIndex, int rightIndex){
147
148
149             if (leftIndex < middleIndex){
150                 int rootIDLeftSubtree = (leftIndex + middleIndex-1)/2;
151                 setLeftLink(middleIndex, rootIDLeftSubtree);
152                 arrayOfBTNodes[rootIDLeftSubtree].setData(array[rootIDLeftSubtree]);
153                 ChainNodes(array, rootIDLeftSubtree, leftIndex, middleIndex-1);
154             }
155
156
157             if (rightIndex > middleIndex){
158                 int rootIDRightSubtree = (rightIndex + middleIndex + 1)/2;
159                 setRightLink(middleIndex, rootIDRightSubtree);
160                 arrayOfBTNodes[rootIDRightSubtree].setData(array[rootIDRightSubtree]);
161                 ChainNodes(array, rootIDRightSubtree, middleIndex+1, rightIndex);
162             }
163
164
165         }
166
167
168         void printLeafNodes(){
169
170             for (int id = 0; id < numNodes; id++){
171
172                 if (arrayOfBTNodes[id].getLeftChildPtr() == 0 && arrayOfBTNodes[id].
                    getRightChildPtr() == 0)
173                     cout << arrayOfBTNodes[id].getData() << " ";
174             }
175
176             cout << endl;
177         }
178
179
180         void InOrderTraversal(int nodeid){
181
182             if (nodeid == -1)
183                 return;
184
185
186
187             InOrderTraversal(arrayOfBTNodes[nodeid].getLeftChildID());
188             cout << arrayOfBTNodes[nodeid].getData() << " ";
189             InOrderTraversal(arrayOfBTNodes[nodeid].getRightChildID());
```

```
190
191         }
192
193
194         void PrintInOrderTraversal(){
195
196             InOrderTraversal(rootNodeID);
197             cout << endl;
198
199         }
200
201
202         int getKeyIndex(int searchKey){
203
204             int searchNodeID = rootNodeID;
205
206             while (searchNodeID != -1){
207
208                 if (searchKey == arrayOfBTNodes[searchNodeID].getData())
209                     return searchNodeID;
210                 else if (searchKey < arrayOfBTNodes[searchNodeID].getData())
211                     searchNodeID = arrayOfBTNodes[searchNodeID].getLeftChildID();
212                 else
213                     searchNodeID = arrayOfBTNodes[searchNodeID].getRightChildID();
214
215             }
216
217             return -1;
218
219         }
220
221 };
222
223
224     void selectionSort(int *array, int arraySize){
225
226         for (int iterationNum = 0; iterationNum < arraySize-1; iterationNum++){
227
228             int minIndex = iterationNum;
229
230             for (int j = iterationNum+1; j < arraySize; j++){
231
232                 if (array[j] < array[minIndex])
233                     minIndex = j;
234
235             }
236
237             // swap array[minIndex] with array[iterationNum]
238             int temp = array[minIndex];
239             array[minIndex] = array[iterationNum];
240             array[iterationNum] = temp;
241
242         }
243
244     }
245
246
247 int main(){
248
249     int numElements;
250     cout << "Enter the number of elements: ";
251     cin >> numElements;
252
253     int *array = new int[numElements];
```

```
254
255         int maxValue;
256         cout << "Enter the maximum value for an element: ";
257         cin >> maxValue;
258
259         srand(time(NULL));
260
261         cout << "array generated: ";
262
263         for (int index = 0; index < numElements; index++){
264             array[index] = rand() % maxValue;
265             cout << array[index] << " ";
266         }
267
268         cout << endl;
269
270         selectionSort(array, numElements);
271
272         BinarySearchTree bsTree(numElements);
273         bsTree.constructBSTree(array);
274
275         int searchKey;
276         cout << "Enter a search key: ";
277         cin >> searchKey;
278
279         int keyIndex = bsTree.getKeyIndex(searchKey);
280
281         if (keyIndex != -1)
282             cout << searchKey << " is present " <<  endl;
283         else
284             cout << searchKey << " is not present " << endl;
285
286
287         return 0;
288     }
```