```java
import java.io.*;
import java.util.*;


class BTNode{

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

    public BTNode(){}

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

    public  int getNodeId(){
        return nodeid;
    }

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

    public int getData(){
        return data;
    }

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

    public  int getLevelNum(){
        return levelNum;
    }

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

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

    public  BTNode getLeftChildPtr(){
        return leftChildPtr;
    }

    public  BTNode getRightChildPtr(){
        return rightChildPtr;
    }

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

        return leftChildPtr.getNodeId();
    }

    public  int getRightChildID(){
        if (rightChildPtr == null)
            return -1;
```

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

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

```
191
192         }
193
194
195
196     }
197
198
199
200     class BSTSearch{
201
202       public static void selectionSort(int array[], int arraySize){
203
204         for (int iterationNum = 0; iterationNum < arraySize-1; iterationNum++){
205
206             int minIndex = iterationNum;
207
208             for (int j = iterationNum+1; j < arraySize; j++){
209
210                 if (array[j] < array[minIndex])
211                     minIndex = j;
212
213             }
214
215             // swap array[minIndex] with array[iterationNum]
216             int temp = array[minIndex];
217             array[minIndex] = array[iterationNum];
218             array[iterationNum] = temp;
219
220         }
221
222     }
223
224
225       public static void main(String[] args){
226
227         Scanner input = new Scanner(System.in);
228
229         int numElements;
230         System.out.print("Enter the number of elements: ");
231         numElements = input.nextInt();
232
233         int array[] = new int[numElements];
234
235         int maxValue;
236         System.out.print("Enter the maximum value for an element: ");
237         maxValue = input.nextInt();
238
239         Random randGen = new Random(System.currentTimeMillis());
240
241         System.out.print("Array Generated: ");
242         for (int index = 0; index < numElements; index++){
243             array[index] = randGen.nextInt(maxValue);
244             System.out.print(array[index] +" ");
245         }
246         System.out.println();
247
248         selectionSort(array, numElements);
249
250         BinarySearchTree bsTree = new BinarySearchTree(numElements);
251         bsTree.constructBSTree(array);
252
253         int searchKey;
254         System.out.print("Enter a search key: ");
```

```
255         searchKey = input.nextInt();
256 
257         int keyIndex = bsTree.getKeyIndex(searchKey);
258 
259         if (keyIndex != -1)
260             System.out.println(searchKey +" is present ");
261         else
262             System.out.println(searchKey + " is not present ");
263 
264         }
265 
266     }
```