```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 constructBSTree(int *array){
111
112             int leftIndex = 0;
113             int rightIndex = numNodes-1;
114             int middleIndex = (leftIndex + rightIndex)/2;
115
116             rootNodeID = middleIndex;
117             arrayOfBTNodes[middleIndex].setData(array[middleIndex]);
118
119             ChainNodes(array, middleIndex, leftIndex, rightIndex);
120
121         }
122
123
124         void ChainNodes(int* array, int middleIndex, int leftIndex, int rightIndex){
125
126
```

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

```
                    minIndex = j;

            }

            // swap array[minIndex] with array[iterationNum]
            int temp = array[minIndex];
            array[minIndex] = array[iterationNum];
            array[iterationNum] = temp;

        }

    }

int main(){

    int numElements;
    cout << "Enter the number of elements: ";
    cin >> numElements;

    int *array = new int[numElements];

    int maxValue;
    cout << "Enter the maximum value for an element: ";
    cin >> maxValue;

    srand(time(NULL));

    cout << "array generated: ";

    for (int index = 0; index < numElements; index++){
        array[index] = rand() % maxValue;
        cout << array[index] << " ";
    }

    cout << endl;

    selectionSort(array, numElements);


    BinarySearchTree bsTree(numElements);
    bsTree.constructBSTree(array);

    cout << "Inorder traversal: ";
    bsTree.PrintInOrderTraversal();

    cout << "Leaf nodes: ";
    bsTree.printLeafNodes();

    return 0;
}
```

```
Enter the number of elements: 10
Enter the maximum value for an element: 25
array generated: 1 19 13 1 19 5 12 11 3 10
Inorder traversal: 1 1 3 5 10 11 12 13 19 19
Leaf nodes: 1 5 12 19
```