Status	Finished
Started	Tuesday, 5 November 2024, 8:03 AM
Completed	Monday, 18 November 2024, 7:34 PM
Duration	13 days 11 hours
Marks	7.90/8.00
Grade	**9.88** out of 10.00 (**98.75**%)

Question 1

Partially correct

Mark 0.90 out of 1.00

In this question, you have to perform add **and delete on binary search tree**. Note that:

- When deleting a node which still have 2 children, **take the inorder successor** (smallest node of the right sub tree of that node) to replace it.

- When adding a node which has the same value as parent node, add it in the **left sub tree**.

Your task is to implement two functions: add and deleteNode. You could define one or more functions to achieve this task.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"
template<class T>
class BinarySearchTree
{
public:
    class Node;
private:
    Node* root;
public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    //Helping function

    void add(T value){
        //TODO
    }

    void deleteNode(T value){
        //TODO
    }
    string inOrderRec(Node* root) {
        stringstream ss;
        if (root != nullptr) {
            ss << inOrderRec(root->pLeft);
            ss << root->value << " ";
            ss << inOrderRec(root->pRight);
        }
        return ss.str();
    }

    string inOrder(){
        return inOrderRec(this->root);
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;
    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
};
```

For example:

Test	Result

BinarySearchTree<int> bst;
bst.add(9);
bst.add(2);
bst.add(10);
bst.deleteNode(9);
cout << bst.inOrder();
	2 10

BinarySearchTree<int> bst;
bst.add(9);
bst.add(2);
bst.add(10);
bst.add(8);
cout << bst.inOrder()<<endl;
bst.add(11);
bst.deleteNode(9);
cout << bst.inOrder();
	2 8 9 10
	2 8 10 11

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
//Helping functions
Node* helpAdd(Node* root,T val)
{

    if(!root)
    {
        root = new Node(val);
        return root;
    }

    (root->value < val) ? root->pRight = helpAdd(root->pRight,val) : root->pLeft = helpAdd(root->pLe
    return root;
}


void add(T value){
    //TODO

    root = helpAdd(root,value);

}

Node *helpDel(Node* root, T val)
{
    if(!root) return nullptr;

    if(root->value == val)
    {
        if(!root->pRight)
        {
            Node* tmp = root->pLeft;
            delete root;
            return tmp;
        }

        else if(!root->pLeft)
        {
            Node* tmp = root->pRight;
            delete root;
            return tmp;
        }

        else
        {
            Node* tr = root->pRight;
            while(tr->pLeft) tr = tr->pLeft;

            root->value = tr->value;
            root->pRight = helpDel(root->pRight,tr->value);
        }
    }

    (root->value < val) ? root->pRight = helpDel(root->pRight,val) : root->pLeft = helpDel(root->pLe
    return root;
```

```
55 }
56
57 void deleteNode(T value){
58     //TODO
59     root = helpDel(root,value);
60 }
```

	Test	Expected	Got	

✓	BinarySearchTree<int> bst;
	bst.add(9);
	bst.add(2);
	bst.add(10);
	bst.deleteNode(9);
	cout << bst.inOrder();
		2 10	2 10	✓

✓	BinarySearchTree<int> bst;
	bst.add(9);
	bst.add(2);
	bst.add(10);
	bst.add(8);
	cout << bst.inOrder()<<endl;
	bst.add(11);
	bst.deleteNode(9);
	cout << bst.inOrder();
		2 8 9 10
		2 8 10 11
			2 8 9 10
			2 8 10 11
				✓

Your code failed one or more hidden tests.

Partially correct

Marks for this submission: 0.90/1.00.

Question **2**

Correct

Mark 1.00 out of 1.00

Given class **BinarySearchTree**, you need to finish method getMin() and getMax() in this question.

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

template<class T>
class BinarySearchTree
{
public:
    class Node;

private:
    Node* root;

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;

    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    Node* addRec(Node* root, T value);
    void add(T value) ;
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```

For example:

Test	Result

`BinarySearchTree<int> bst;`
`for (int i = 0; i < 10; ++i) {`
`  bst.add(i);`
`}`
`cout << bst.getMin() << endl;`
`cout << bst.getMax() << endl;`
	`0`
	`9`

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
1  // STUDENT ANSWER BEGIN
2  // You can define other functions here to help you.
3
4  T getMin() {
```

```
    //TODO: return the minimum values of nodes in the tree.
    if(!root) return 0;
    Node* tr = root;
    while(tr->pLeft)
    {
        tr = tr->pLeft;
    }
    return tr->value;
}

T getMax() {
    //TODO: return the maximum values of nodes in the tree.
    if(!root) return 0;
    Node* tr = root;
    while(tr->pRight)
    {
        tr = tr->pRight;
    }
    return tr->value;
}

// STUDENT ANSWER END
```

	Test	Expected	Got	

✓	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(i);`
	`}`
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		0
		9
			0
			9
				✓

✓	`int values[] = { 66,60,84,67,21,45,62,1,80,35 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		1
		84
			1
			84
				✓

✓	`int values[] = { 38,0,98,38,99,67,19,70,55,6 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		0
		99
			0
			99
				✓

✓	`int values[] = { 34,81,73,48,66,91,19,84,78,79 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		19
		91
			19
			91
				✓

✓	`int values[] = { 94,61,75,36,34,58,62,74,54,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		34
		94
			34
			94
				✓

	Test	Expected	Got	

✓	`int values[] = { 32,0,2,84,34,78,70,60,95,71,26,62,0,22,95 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		0
		95
			0
			95
				✓

✓	`int values[] = { 53,24,32,40,80,47,81,88,42,29,31,91,77,73,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		24
		91
			24
			91
				✓

✓	`int values[] = { 32,19,23,33,76,1,37,53,18,89,28,1,77,52,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		1
		89
			1
			89
				✓

✓	`int values[] = { 25,29,57,30,62,56,60,55,88,56,70,83,56,75,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		17
		88
			17
			88
				✓

✓	`int values[] = { 75,13,83,83,30,40,10,86,17,21,45,22,22,72,63 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.getMin() << endl;`
	`cout << bst.getMax() << endl;`
		10
		86
			10
			86
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 3

Correct

Mark 1.00 out of 1.00

Given class **BinarySearchTree**, you need to finish method **find(i)** to check whether value i is in the tree or not; method **sum(l,r)** to calculate sum of all all elements v in the tree that has value greater than or equal to l and less than or equal to r.

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

template<class T>
class BinarySearchTree
{
public:
    class Node;

private:
    Node* root;

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;

    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    Node* addRec(Node* root, T value);
    void add(T value) ;
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```

For example:

Test	Result

`BinarySearchTree<int> bst;`
`for (int i = 0; i < 10; ++i) {`
`  bst.add(i);`
`}`
`cout << bst.find(7) << endl;`
`cout << bst.sum(0, 4) << endl`
	1
	10

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.

bool helpFind(Node* head,T i)
{
    if(!head) return false;
    if(head->value == i) return true;
    if(head->value > i) return helpFind(head->pLeft,i);
    return helpFind(head->pRight,i);
```

```
10
11  }
12
13  bool find(T i) {
14      // TODO: return true if value i is in the tree; otherwise, return false.
15      return helpFind(root,i);
16  }
17
18  T helpSum(Node* tr, T l, T r)
19  {
20      if(!tr) return 0;
21      if(tr->value < l) return helpSum(tr->pRight,l,r);
22      if(tr->value > r) return helpSum(tr->pLeft,l,r);
23
24      return tr->value + helpSum(tr->pLeft,l,r) + helpSum(tr->pRight,l,r);
25  }
26
27  T sum(T l, T r) {
28      // TODO: return the sum of all element in the tree has value in range [l,r].
29      return helpSum(root,l,r);
30  }
31
32  // STUDENT ANSWER END
```

	Test	Expected	Got	

✓	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(i);`
	`}`
	`cout << bst.find(7) << endl;`
	`cout << bst.sum(0, 4) << endl`
		1
		10
			1
			10
				✓

✓	`int values[] = { 66,60,84,67,21,45,62,1,80,35 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(5) << endl;`
	`cout << bst.sum(10, 40);`
		0
		56
			0
			56
				✓

✓	`int values[] = { 38,0,98,38,99,67,19,70,55,6 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(5) << endl;`
	`cout << bst.sum(10, 40);`
		0
		95
			0
			95
				✓

✓	`int values[] = { 34,81,73,48,66,91,19,84,78,79 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(5) << endl;`
	`cout << bst.sum(10, 40);`
		0
		53
			0
			53
				✓

✓	`int values[] = { 94,61,75,36,34,58,62,74,54,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		1
		70
			1
			70
				✓

	Test	Expected	Got	

✓	`int values[] = { 32,0,2,84,34,78,70,60,95,71,26,62,0,22,95 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		1
		114
			1
			114
				✓

✓	`int values[] = { 53,24,32,40,80,47,81,88,42,29,31,91,77,73,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		156
			0
			156
				✓

✓	`int values[] = { 32,19,23,33,76,1,37,53,18,89,28,1,77,52,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		207
			0
			207
				✓

✓	`int values[] = { 25,29,57,30,62,56,60,55,88,56,70,83,56,75,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		101
			0
			101
				✓

✓	`int values[] = { 75,13,83,83,30,40,10,86,17,21,45,22,22,72,63 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		175
			0
			175
				✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 4

Correct

Mark 1.00 out of 1.00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

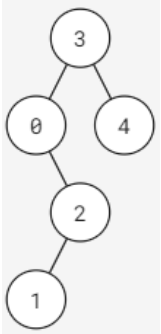
Request: Implement function:

```
vector<int> levelAlterTraverse(BSTNode* root);
```

Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the values of the nodes in each level, alternating from going left-to-right and right-to-left..

Example:

Given a binary search tree in the following:



In the first level, we should traverse from left to right (order: `3`) and in the second level, we traverse from right to left (order: `4, 0`). After traversing all the nodes, the result should be `[3, 4, 0, 2, 1]`.

Note: In this exercise, the libraries `iostream`, `vector`, `stack`, `queue`, `algorithm` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result
int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); printVector(levelAlterTraverse(root)); BSTNode::deleteTree(root);	[0, 3, 1, 5, 4, 2]

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
vector<int> levelAlterTraverse(BSTNode* root) {
    // STUDENT ANSWER

    vector<int> ans;
    if(!root) return ans;

    queue<BSTNode*> q1;
    queue<BSTNode*> q2;
    stack<int>s;
    q1.push(root);
    bool role = 0;

    while(!q1.empty() || !q2.empty())
    {
        while(!q1.empty() && !role)
        {
            BSTNode *tr = q1.front(); q1.pop();

            if(tr->left) q2.push(tr->left);
            if(tr->right) q2.push(tr->right);

            ans.push_back(tr->val);
        }

        while(!q2.empty() && role)
        {
            BSTNode *tr = q2.front(); q2.pop();

            if(tr->left) q1.push(tr->left);
            if(tr->right) q1.push(tr->right);

            s.push(tr->val);
        }

        while(!s.empty())
        {
            ans.push_back(s.top());
            s.pop();
        }

        role = !role;

    }


    return ans;
}
```

	Test	Expected	Got	
✓	int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); printVector(levelAlterTraverse(root)); BSTNode::deleteTree(root);	[0, 3, 1, 5, 4, 2]	[0, 3, 1, 5, 4, 2]	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question **5**

Correct

Mark 1.00 out of 1.00

Class **BTNode** is used to store a node in binary search tree, described on the following:

```
class BTNode {
    public:
        int val;
        BTNode *left;
        BTNode *right;
        BTNode() {
            this->left = this->right = NULL;
        }
        BTNode(int val) {
            this->val = val;
            this->left = this->right = NULL;
        }
        BTNode(int val, BTNode*& left, BTNode*& right) {
            this->val = val;
            this->left = left;
            this->right = right;
        }
};
```

Where `val` is the value of node (non-negative integer), `left` and `right` are the pointers to the left node and right node of it, respectively.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

Request: Implement function:

```
int rangeCount(BTNode* root, int lo, int hi);
```

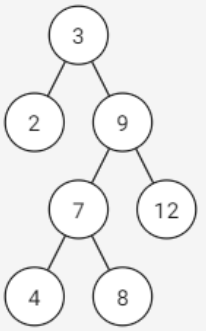
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements), `lo` and `hi` are 2 positives integer and `lo ≤ hi`. This function returns the number of all nodes whose values are between `[lo, hi]` in this binary search tree.

More information:

- If a node has `val` which is equal to its ancestor's, it is in the right subtree of its ancestor.

Example:

Given a binary search tree in the following:



With `lo=5, hi=10`, all the nodes satisfied are node `9, 7, 8`; there fore, the result is `3`.

Note: In this exercise, the libraries `iostream`, `stack`, `queue`, `utility` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int value[] = {3,2,9,7,12,4,8};`
`int lo = 5, hi = 10;`
`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
`cout << rangeCount(root, lo, hi);`
	3

`int value[] = {1167,2381,577,2568,124,1519,234,1679,2696,2359};`
`int lo = 500, hi = 2000;`
`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
`cout << rangeCount(root, lo, hi);`
	4

Answer: (penalty regime: 0 %)

Reset answer

```
int rangeCount(BTNode* root, int lo, int hi)
{

    if(!root) return 0;
    if(root->val < lo) return rangeCount(root->right,lo,hi);
    if(root->val > hi) return rangeCount(root->left,lo,hi);


    return 1 + rangeCount(root->left,lo,hi) + rangeCount(root->right,lo,hi);

}
```

	Test	Expected	Got	

✓	`int value[] = {3,2,9,7,12,4,8};`
	`int lo = 5, hi = 10;`
	`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
	`cout << rangeCount(root, lo, hi);`
		3	3	✓

✓	`int value[] = {1167,2381,577,2568,124,1519,234,1679,2696,2359};`
	`int lo = 500, hi = 2000;`
	`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
	`cout << rangeCount(root, lo, hi);`
		4	4	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 6

Correct

Mark 1.00 out of 1.00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

Request: Implement function:

```
int singleChild(BSTNode* root);
```

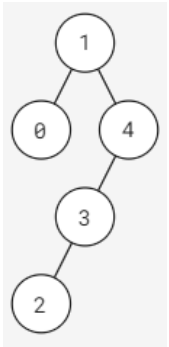
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the number of single children in the tree.

More information:

- A node is called a **single child** if its parent has only one child.

Example:

Given a binary search tree in the following:



There are `2` single children: node `2` and node `3`.

Note: In this exercise, the libraries `iostream` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result
int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); cout << singleChild(root); BSTNode::deleteTree(root);	3

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
int singleChild(BSTNode* root) {

    if(!root) return 0;

    if(
        (root->right && !root->left)
        ||
        (root->left && !root->right)
    )
        return 1 + singleChild(root->left) + singleChild(root->right);

    return singleChild(root->left) + singleChild(root->right);
}
```

	Test	Expected	Got	
✓	int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); cout << singleChild(root); BSTNode::deleteTree(root);	3	3	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 7

Correct

Mark 1.00 out of 1.00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

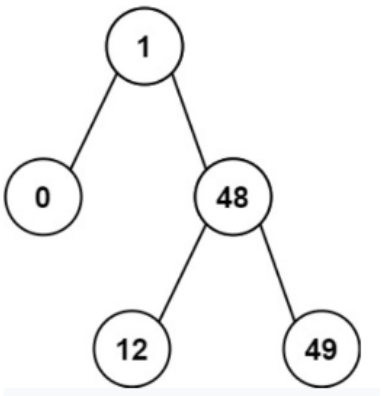
Request: Implement function:

```
int kthSmallest(BSTNode* root, int k);
```

Where `root` is the root node of given binary search tree (this tree has `n` elements) and `k` satisfy: `1 <= k <= n <= 100000`. This function returns the `k`-th smallest value in the tree.

Example:

Given a binary search tree in the following:



With `k = 2`, the result should be `1`.

Note: In this exercise, the libraries `iostream`, `vector`, `stack`, `queue`, `algorithm`, `climits` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

int arr[] = {6, 9, 2, 13, 0, 20};
int k = 2;
BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
cout << kthSmallest(root, k);
BSTNode::deleteTree(root);
	2

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
int help(BSTNode * tr, int& count, int k)
{
    if(!tr) return -1;
    int left = help(tr->left,count,k);

    if(left != -1) return left;

    if(count == k) return tr->val;
    count++;

    int right = help(tr->right,count,k);
    return right;
}

int kthSmallest(BSTNode* root, int k)
{
    int ans = 1;
    return help(root,ans,k);
}
```

	Test	Expected	Got	

✓	int arr[] = {6, 9, 2, 13, 0, 20};
	int k = 2;
	BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
	cout << kthSmallest(root, k);
	BSTNode::deleteTree(root);
		2	2	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Question 8

Correct

Mark 1.00 out of 1.00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

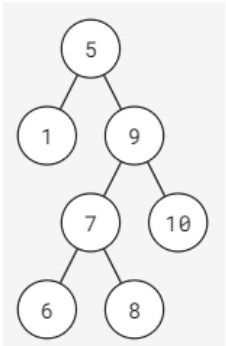
Request: Implement function:

```
BSTNode* subtreeWithRange(BSTNode* root, int lo, int hi);
```

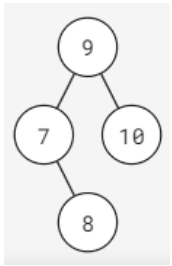
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the binary search tree after deleting all nodes whose values are outside the range `[lo, hi]` (inclusive).

Example:

Given a binary search tree in the following:



With `lo = 7` and `hi = 10`, the result should be:



Note: In this exercise, the libraries `iostream` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {0, 3, 5, 1, 2, 4};`
`int lo = 1, hi = 3;`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`root = subtreeWithRange(root, lo, hi);`
`BSTNode::printPreorder(root);`
`BSTNode::deleteTree(root);`
	`3 1 2`

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
BSTNode* subtreeWithRange(BSTNode* root, int lo, int hi) {
    // STUDENT ANSWER
    if(!root) return root;
    if(lo > root->val && hi > root->val) return subtreeWithRange(root->right,lo,hi);
    if(lo < root->val && hi < root->val) return subtreeWithRange(root->left,lo,hi);

    BSTNode* left = subtreeWithRange(root->left,lo,hi);
    BSTNode* right = subtreeWithRange(root->right,lo,hi);
    BSTNode *phe = new BSTNode(root->val,left,right);

    return phe;
}
```

	Test	Expected	Got	

✓	`int arr[] = {0, 3, 5, 1, 2, 4};`
	`int lo = 1, hi = 3;`
	`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
	`root = subtreeWithRange(root, lo, hi);`
	`BSTNode::printPreorder(root);`
	`BSTNode::deleteTree(root);`
		`3 1 2`	`3 1 2`	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.