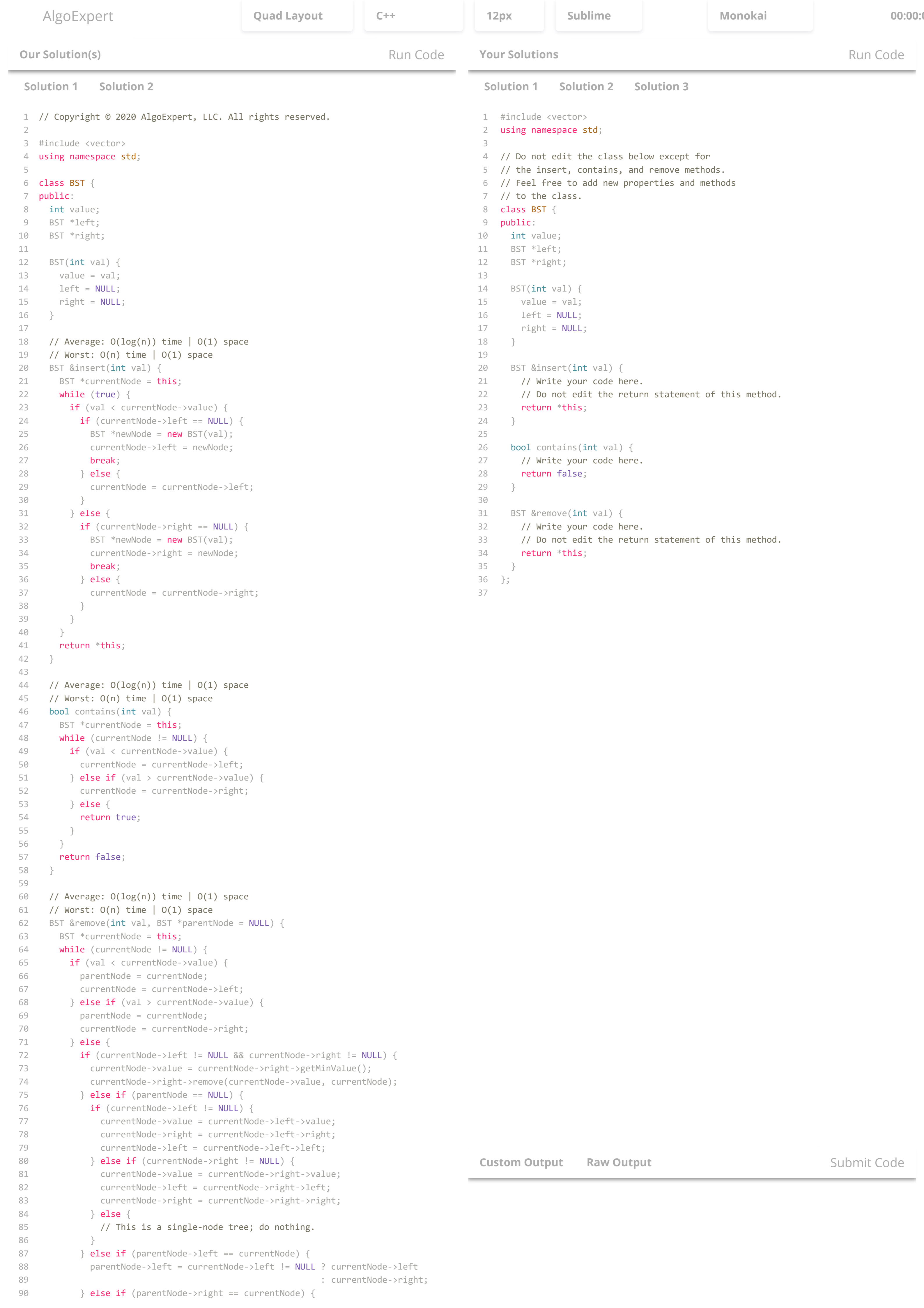

```cpp
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

#include <vector>
using namespace std;

class BST {
public:
  int value;
  BST *left;
  BST *right;

  BST(int val) {
    value = val;
    left = NULL;
    right = NULL;
  }

  // Average: O(log(n)) time | O(1) space
  // Worst: O(n) time | O(1) space
  BST &insert(int val) {
    BST *currentNode = this;
    while (true) {
      if (val < currentNode->value) {
        if (currentNode->left == NULL) {
          BST *newNode = new BST(val);
          currentNode->left = newNode;
          break;
        } else {
          currentNode = currentNode->left;
        }
      } else {
        if (currentNode->right == NULL) {
          BST *newNode = new BST(val);
          currentNode->right = newNode;
          break;
        } else {
          currentNode = currentNode->right;
        }
      }
    }
    return *this;
  }

  // Average: O(log(n)) time | O(1) space
  // Worst: O(n) time | O(1) space
  bool contains(int val) {
    BST *currentNode = this;
    while (currentNode != NULL) {
      if (val < currentNode->value) {
        currentNode = currentNode->left;
      } else if (val > currentNode->value) {
        currentNode = currentNode->right;
      } else {
        return true;
      }
    }
    return false;
  }

  // Average: O(log(n)) time | O(1) space
  // Worst: O(n) time | O(1) space
  BST &remove(int val, BST *parentNode = NULL) {
    BST *currentNode = this;
    while (currentNode != NULL) {
      if (val < currentNode->value) {
        parentNode = currentNode;
        currentNode = currentNode->left;
      } else if (val > currentNode->value) {
        parentNode = currentNode;
        currentNode = currentNode->right;
      } else {
        if (currentNode->left != NULL && currentNode->right != NULL) {
          currentNode->value = currentNode->right->getMinValue();
          currentNode->right->remove(currentNode->value, currentNode);
        } else if (parentNode == NULL) {
          if (currentNode->left != NULL) {
            currentNode->value = currentNode->left->value;
            currentNode->right = currentNode->left->right;
            currentNode->left = currentNode->left->left;
          } else if (currentNode->right != NULL) {
            currentNode->value = currentNode->right->value;
            currentNode->left = currentNode->right->left;
            currentNode->right = currentNode->right->right;
          } else {
            // This is a single-node tree; do nothing.
          }
        } else if (parentNode->left == currentNode) {
          parentNode->left = currentNode->left != NULL ? currentNode->left
                                                       : currentNode->right;
        } else if (parentNode->right == currentNode) {
```

```cpp
#include <vector>
using namespace std;

// Do not edit the class below except for
// the insert, contains, and remove methods.
// Feel free to add new properties and methods
// to the class.
class BST {
public:
  int value;
  BST *left;
  BST *right;

  BST(int val) {
    value = val;
    left = NULL;
    right = NULL;
  }

  BST &insert(int val) {
    // Write your code here.
    // Do not edit the return statement of this method.
    return *this;
  }

  bool contains(int val) {
    // Write your code here.
    return false;
  }

  BST &remove(int val) {
    // Write your code here.
    // Do not edit the return statement of this method.
    return *this;
  }
};
```

```
          parentNode->right = currentNode->left != NULL ? currentNode->left
                                                        : currentNode->right;
        }
        break;
      }
    }
    return *this;
  }

  int getMinValue() {
    if (left == NULL) {
      return value;
    } else {
      return left->getMinValue();
    }
  }
};
```

Run or submit code when you're ready.