Problem 1

Code:

Main:

problem1 (Global Scope)

```cpp
#include "BST.h"
#include <iostream>

using namespace std;

dateType getDate();
void addDate(BST &Btree);
void removeDate(BST &Btree);
void searchDate(BST &Btree);

int main() {
    int option = 0;
    BST Btree;

    while (option != -1) {
        cout << "Enter 1 to add a date, 2 to remove a date, 3 to search for a date, 4 to display all dates in ascending order, or -1 to quit." << endl;
        cin >> option;
        switch (option) {
        case 1:
            addDate(Btree);
            break;
        case 2:
            removeDate(Btree);
            break;
        case 3:
            searchDate(Btree);
            break;
        case 4:
            cout << "In order traversal of the tree: " << endl;
            Btree.inorderTraversal();
            break;
        case -1:
            break;
        default:
            cout << "Enter a valid option!" << endl;
        }
    }
    return 0;
}

void addDate(BST& Btree) {
    dateType dateToAdd = getDate();
    Btree.insert(dateToAdd);
}
```

```cpp

void removeDate(BST& Btree) {
    dateType dateToRemove = getDate();
    Btree.deleteNode(Btree.getRoot(), dateToRemove);
}

void searchDate(BST& Btree) {
    dateType dateToSearch = getDate();
    bool ans;
    ans = Btree.search(dateToSearch);
    if (ans) {
        cout << "Element found" << endl;
    }
    else {
        cout << "Element not found" << endl;
    }
}

dateType getDate() {
    int d;
    int m;
    int y;
    cout << "Enter the day of the date: ";
    cin >> d;
    cout << "Enter the month of the date: ";
    cin >> m;
    cout << "Enter the year of the date: ";
    cin >> y;
    dateType date(d, m, y);
    return date;
}
```

BST.cpp

```
problem1                                                    BST
```

```cpp
#include <iostream>
#include "BST.h"

 using namespace std;

BST::BST()
{
    root = NULL;
}
bool BST::search(const dateType searchItem) const
{
    nodeType* current;
    bool found = false;
    if (root == NULL)
        cout << "Cannot search an empty tree." << endl;
    else
    {
        current = root;
        while (current != NULL && !found)
        {
            if (searchItem == current->info)
                found = true;
            else if (searchItem < current->info)
                current = current->lLink;
            else
                current = current->rLink;
        }
    }
    return found;
}

nodeType* BST::getRoot() {
    return root;
}

void BST::insert(const dateType insertItem)
{
    nodeType* current;
    nodeType* trailCurrent = NULL;
    nodeType* newNode;
    newNode = new nodeType;
    newNode->info = insertItem;
    newNode->lLink = NULL;
    newNode->rLink = NULL;
    if (root == NULL)
```

```
46          root = newNode;
47      else
48      {
49          current = root;
50          while (current != NULL)
51          {
52              trailCurrent = current;
53              if (insertItem == current->info)
54              {
55                  cout << "Duplicates are not "
56                      << "allowed. Value: " << insertItem << endl;
57                  return;
58              }
59              else
60                  if (insertItem < current->info)
61                      current = current->lLink;
62                  else
63                      current = current->rLink;
64          }
65          if (insertItem < trailCurrent->info)
66              trailCurrent->lLink = newNode;
67          else
68              trailCurrent->rLink = newNode;
69      }
70  }
71
72  nodeType* BST::deleteNode(nodeType* curr, const dateType deleteItem) {
73      if (curr == nullptr) {
74          return curr;
75      }
76      if (deleteItem < root->info) {
77          root->lLink = deleteNode(root->lLink, deleteItem);
78      }
79      else if (deleteItem > root->info) {
80          root->rLink = deleteNode(root->rLink, deleteItem);
81      }
82      else {
83          if (root->lLink == nullptr) {
84              nodeType* temp = root->rLink;
85              delete root;
86              return temp;
87          } else if (root->rLink == nullptr) {
```

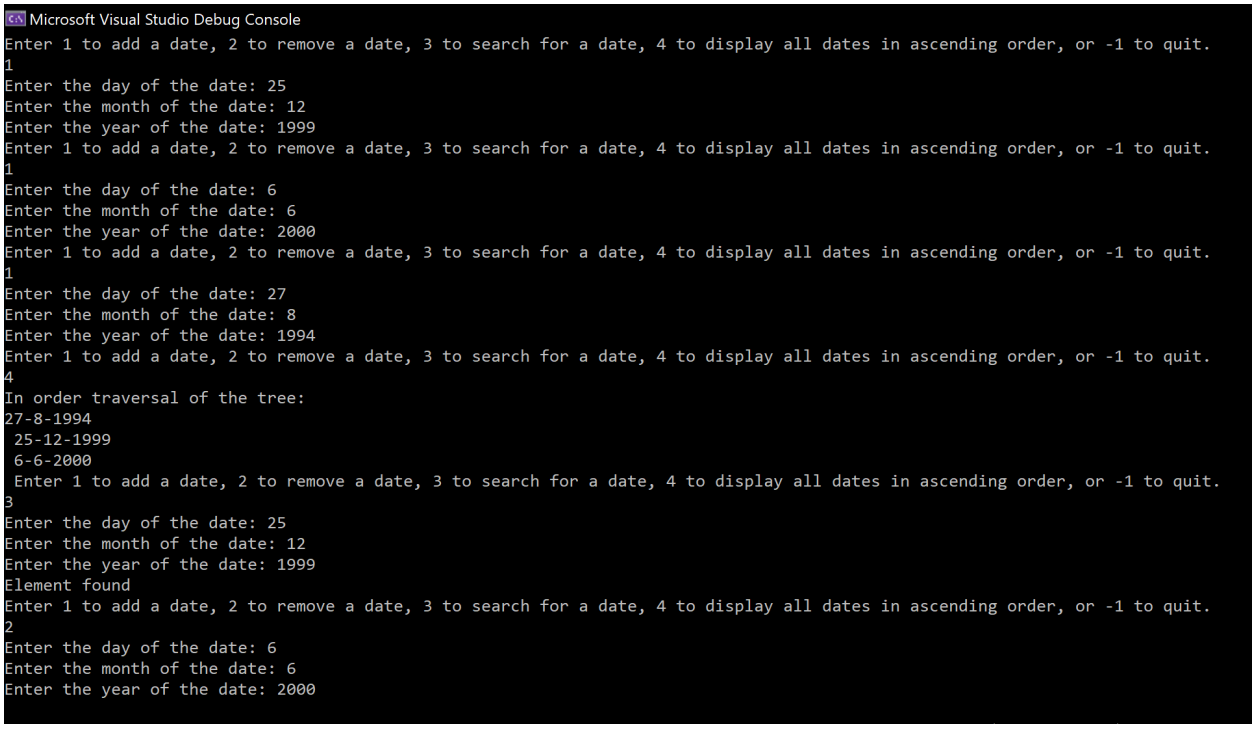
```
88                  nodeType* temp = root->lLink;
89                  delete root;
90                  return temp;
91              }
92              nodeType* temp = minValueNode(root->rLink);
93              root->info = temp->info;
94              root->rLink = deleteNode(root->rLink, temp->info);
95          }
96          return root;
97      }
98
99      nodeType* BST::minValueNode(nodeType* node) {
100         nodeType* curr = node;
101         while (curr && curr->lLink != nullptr) {
102             curr = curr->lLink;
103         }
104         return curr;
105     }
106
107     bool BST::isEmpty() const
108     {
109         return (root == NULL);
110     }
111     void BST::preorderTraversal() const
112     {
113         preorder(root);
114     }
115     void BST::inorderTraversal() const
116     {
117         inorder(root);
118     }
119     void BST::postorderTraversal() const
120     {
121         postorder(root);
122     }
123     void BST::preorder(nodeType* p) const
124     {
125         if (p != NULL)
126         {
127             cout << p->info << " ";
128             preorder(p->lLink);
129             preorder(p->rLink);
```

```
130         }
131     }
132     void BST::postorder(nodeType* p) const
133     {
134         if (p != NULL)
135         {
136             postorder(p->lLink);
137             postorder(p->rLink);
138             cout << p->info << " ";
139         }
140     }
141     void BST::inorder(nodeType* p) const
142     {
143         if (p != NULL)
144         {
145             inorder(p->lLink);
146             cout << p->info << " ";
147             inorder(p->rLink);
148         }
149     }
150     void BST::destroy(nodeType*& p)
151     {
152         if (p != NULL)
153         {
154             destroy(p->lLink);
155             destroy(p->rLink);
156             delete p;
157             p = NULL;
158         }
159     }
160     BST::~BST()
161     {
162         destroy(root);
163     }
164
```

BST.h

```
oblem1                                                    BST
1   #include "dateType.h"
2   #include <iostream>
3    using namespace std;
4   struct nodeType
5   {
6       dateType info;
7       nodeType* lLink;
8       nodeType* rLink;
9   };
10  class BST
11  {
12  public:
13      BST();
14      ~BST();
15      nodeType* getRoot();
16      bool search(const dateType searchItem) const;
17      void insert(const dateType insertItem);
18      nodeType* deleteNode(nodeType* curr, const dateType deleteItem);
19      nodeType* minValueNode(nodeType* node);
20      bool isEmpty() const;
21      void inorderTraversal() const;
22      void preorderTraversal() const;
23      void postorderTraversal() const;
24      void destroy(nodeType*& p);
25  private:
26      void inorder(nodeType* p) const;
27      void preorder(nodeType* p) const;
28      void postorder(nodeType* p) const;
29      nodeType* root;
30  };
31
```

Output:



Problem 2: Pseudocode

1. Merge sort:

```
void mergeSort(vector<int>& arr, int l, int r) {
    if (l < r) {
        int mid = l + (r - l) / 2;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }

void merge(vector<int>& arr, int l, int mid, int r) {
    int n1 = mid - l + 1;
    int n2 = r - mid;
    vector<int> L(n1), R(n2);
    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
```

```
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

2. Quick sort:

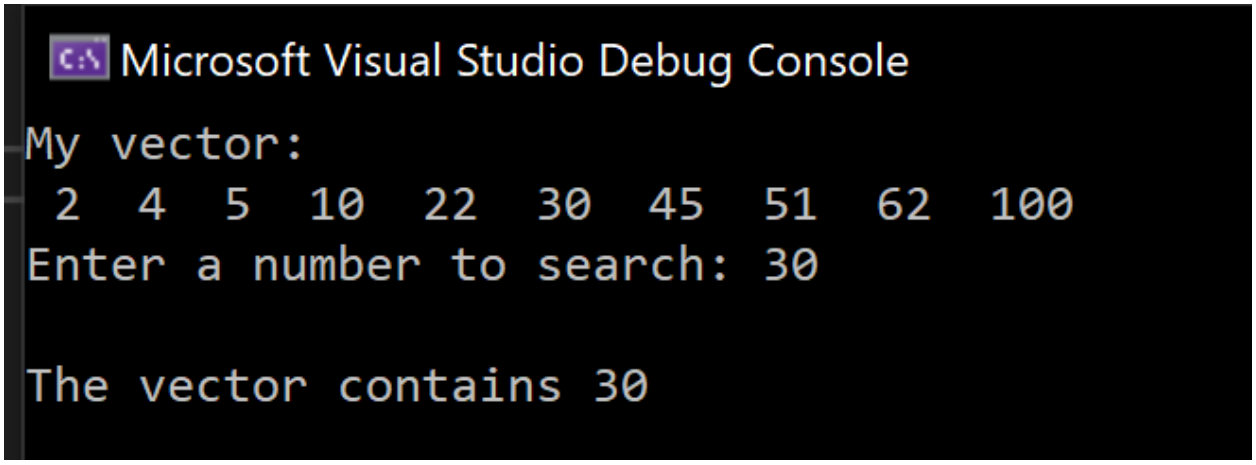
```
Quicksort(arr[], low, high) {
    if (low < high) {
        pivotIndex = Partition(arr, low, high)
        Quicksort(arr, low, pivotIndex - 1)
        Quicksort(arr, pivotIndex + 1, high)
    }
}

Partition(arr[], low, high) {
    pivot = arr[high]
    i = low - 1
    for (j = low; j < high - 1; j++) {
        if (arr[j] <= pivot) {
            i++
            swap arr[i] and arr[j]
         }
    }
    swap arr[i + 1] and arr[high]
    return (i + 1)
}
```

Problem 3:

Code:

```
blem3                                                    (Global Scope)
1   #include <iostream>
2   #include <fstream>
3   #include <string>
4   #include <vector>
5
6   using namespace std;
7
8   bool binarySearch(const vector <int>& nums, int num, int left, int right);
9
10  int main()
11  {
12      vector <int> nums = {2, 4, 5, 10, 22, 30, 45, 51, 62, 100};
13      cout << "My vector: " << endl;
14      vector<int>::iterator itr;
15      for (itr = nums.begin(); itr != nums.end(); ++itr) {
16          cout << " " << *itr << " ";
17      }
18      cout << endl;
19      cout << "Enter a number to search: ";
20      int num;
21      cin >> num;
22      cout << endl;
23      bool contains = binarySearch(nums, num, 0, nums.size() - 1);
24      if (contains) {
25          cout << "The vector contains " << num << endl;
26      }
27      else {
28          cout << "The vector does not contain " << num << endl;
29      }
30      return 0;
31  }
32
33  bool binarySearch(const vector <int> &nums, int num, int left, int right) {
34
35      if (left > right)
36          return false;;
37      int mid = left + (right - left) / 2;
38      if (nums[mid] == num)
39          return true;
40      if (nums[mid] > num) {
41          return binarySearch(nums, num, left, mid - 1);
42      }
43      else {
44          return binarySearch(nums, num, mid + 1, right);
45      }
46  }
    No issues found
```

Output:


Microsoft Visual Studio Debug Console
My vector:
2 4 5 10 22 30 45 51 62 100
Enter a number to search: 30
The vector contains 30