Hw7:

Q1:

BST.h:

```
//BST
#ifndef H_BST
#define H_BST
#include <iostream>
#include "dateType.cpp"
using namespace std;
struct nodeType
{
dateType info;
nodeType *lLink;
nodeType *rLink;
};
//Definition of the class
class BST
{
public:
BST();
~BST();
bool search(const dateType searchItem) const;
void insert(const dateType insertItem);
//void deleteNode(const int deleteItem);
bool isEmpty() const;
void inorderTraversal() const;
void preorderTraversal() const;
void postorderTraversal() const;
void destroy(nodeType* &p);

private:
void inorder(nodeType *p) const;
void preorder(nodeType *p) const;
void postorder(nodeType *p) const;
nodeType *root;
};
#endif
```

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;
}

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)
root = newNode;
else
{
current = root;
while (current != NULL)
{
trailCurrent = current;
if (insertItem == current->info)
{
cout << "Duplicates are not "
<< "allowed. Value: " << insertItem << endl;
```

```
return;
}
else
                        if (insertItem < current->info)
current = current->lLink;
    else
current = current->rLink;
}//end while
if (insertItem < trailCurrent->info)
trailCurrent->lLink = newNode;
else
trailCurrent->rLink = newNode;
}
}//end insert
bool BST::isEmpty() const
{
return (root == NULL);
}
void BST::preorderTraversal() const
{
preorder(root);
cout << endl;
}
void BST::inorderTraversal() const
{
inorder(root);
cout << endl;
}
void BST::postorderTraversal() const
{
postorder(root);
cout << endl;
}
void BST::preorder(nodeType* p) const
{
if (p != NULL)
{
cout << p->info << " ";
preorder(p->lLink);
preorder(p->rLink);
}
}
void BST::postorder(nodeType* p) const
{
```

```
if (p != NULL)
{
postorder(p->lLink);
postorder(p->rLink);
cout << p->info << " ";
}
}
void BST::inorder(nodeType* p) const
{
if (p != NULL)
{
inorder(p->lLink);
cout << p->info << " ";
inorder(p->rLink);
}
}
void BST::destroy(nodeType* &p)
{
if (p != NULL)
{
destroy(p->lLink);
destroy(p->rLink);
delete p;
p = NULL;
}
}
BST::~BST()
{
destroy(root);
}
```

Main.cpp:

```
#include <iostream>
#include <fstream>
#include <string>
#include "BST.cpp"
using namespace std;
int main()
{
    dateType date1(12, 31, 2010);
    dateType date2(2, 29, 2009);
    dateType date3(1, 31, 2003);
    dateType date4(8, 15, 2005);

    dateType date5(6, 1, 2000);
```

```
    dateType date6(1, 1, 2004);
    dateType date7(7, 1, 2000);
    dateType date8(4, 1, 2000);
    BST Btree;
    Btree.insert(date1);
    Btree.insert(date2);
    Btree.insert(date4);
    Btree.insert(date3);
    Btree.insert(date8);
    cout << "inorder" << endl;
    Btree.inorderTraversal();
    cout << "preorder" << endl;
    Btree.preorderTraversal();
    cout << "postorder" << endl;
    Btree.postorderTraversal();

    return 0;
}
```

Result:

```
inorder
04-01-2000 01-31-2003 08-15-2005 02-01-2009 12-31-2010
preorder
12-31-2010 02-01-2009 08-15-2005 01-31-2003 04-01-2000
postorder
04-01-2000 01-31-2003 08-15-2005 02-01-2009 12-31-2010
```

Q2:

quicksort:

```
quickSort(arr[], low, high) {
    if (low < high) {
        /* pi is partitioning index, arr[pi] is now at right place */
        pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
partition (arr[], low, high)
{
    // pivot (Element to be placed at right position)
    pivot = arr[high];

    i = (low - 1)  // Index of smaller element and indicates the
    // right position of pivot found so far
```

```
    for (j = low; j <= high- 1; j++){

        // If current element is smaller than the pivot
        if (arr[j] < pivot){
            i++;    // increment index of smaller element
            swap arr[i] and arr[j]
        }
    }
    swap arr[i + 1] and arr[high])
    return (i + 1)
}
```

mergeSort:

```
mergeSort(array)
    If len(array) > 1 Then
      # This is the point where the array is divided into two subarrays
      halfArray = len(array) / 2

      FirstHalf = array[:halfArray]
      # The first half of the data set

      SecondHalf = array[halfArray:]
      # The second half of the data set

      # Sort the two halves
      mergeSort(FirstHalf)
      mergeSort(SecondHalf)

      k = 0

      # Begin swapping values
      While i < len(FirstHalf) and j < len(SecondHalf)
        If FirstHalf[i] < SecondHalf[j] Then
          array[k] = FirstHalf[i]
          i += 1
        Else
          array[k] = SecondHalf[j]
          j += 1
          k += 1
        EndIf
      EndWhile
    EndIf
```

Q3:

BS.cpp:

```cpp
#include <iostream>
using namespace std;
int BinarySearch(int arr[], int num, int beg, int end);

int main() {

   int arr[100], num, i, n, beg, end;

   cout <<"Enter the size of an array (Max 100) \n";
   cin >> n;

   cout <<"Enter the sorted values \n";

   for(i=0; i<n; i++) {

      cin >> arr[i];
   }

   cout <<"Enter a value to be search \n";
   cin >> num;

   beg = 0;
   end = n-1;

   BinarySearch (arr, num, beg, end);

   return 0;
}
int BinarySearch(int arr[], int num, int left, int right)
   {
   int mid;

   if (left > right){

      cout << "Number is not found";
      return 0;

   }
   else {

   mid = (left + right) / 2;
```

```
   if(arr[mid] == num){

      cout << "Number is found at " << mid << " index \n";
      return 0;

   } else if (num > arr[mid]) {

      BinarySearch (arr, num, mid+1, right);

   } else if (num < arr[mid]) {

      BinarySearch (arr, num, left , mid-1);
   }
   }

   }
```

Result:

```
Enter the size of an array (Max 100)
10
Enter the sorted values
1
2
3
4
5
6
7
8
9
11
Enter a value to be search
5
Number is found at 4 index
```