Q1:

#include <iostream>  
  
using namespace std;  
  
struct Date  
{  
 int month, day, year;  
 Date\* left;  
 Date\* right;  
};  
class Tree  
{  
 Date\* root;  
 Date\* insert(int m, int d, int y, Date\* t);  
 Date\* findMin(Date\* t);  
 Date\* remove(int m, int d, int y, Date\* t);  
 void display(Date\* t);  
 bool find(int m, int d, int y, Date\* t);  
  
public:  
 Tree();  
 void display();  
 void insert();  
 void remove();  
 void search();  
};  
  
  
Date\* Tree:: insert(int m, int d, int y, Date\* t)  
{  
 if(t == NULL)  
 {  
 t = new Date;  
 t->month = m;  
 t->day = d;  
 t->year = y;  
 t->left = t->right = NULL;  
 }  
 else if(y < t->year)  
 t->left = insert(m,d,y, t->left);  
 else if (y == t->year)  
 {  
 if(m < t->month)  
 t->left = insert(m,d,y, t->left);  
 else if (m > t->month)  
 t->right = insert(m,d,y, t->right);  
 else if (m == t->month)  
 {  
 if(d < t->day)  
 t->left = insert(m,d,y, t->left);  
 else if (d > t->day)  
 t->right = insert(m,d,y, t->right);  
 }  
 }  
 else if(y > t->year)  
 t->right = insert(m,d,y, t->right);  
  
 return t;  
}  
  
Date\* Tree:: findMin(Date\* t)  
{  
 if(t == NULL)  
 return NULL;  
 else if(t->left == NULL)  
 return t;  
 else  
 return findMin(t->left);  
}  
  
Date\* Tree:: remove(int m, int d, int y, Date\* t)  
{  
  
 Date\* temp;  
  
 if(t == NULL)  
 return NULL;  
 else if(y < t->year)  
 t->left = remove(m,d,y, t->left);  
 else if (y == t->year)  
 {  
 if(m < t->month)  
 t->left = remove(m,d,y, t->left);  
 else if (m > t->month)  
 t->right = remove(m,d,y, t->right);  
 else if (m == t->month)  
 {  
 if(d < t->day)  
 t->left = remove(m,d,y, t->left);  
 else if (d > t->day)  
 t->right = remove(m,d,y, t->right);  
 else  
 {  
 if(t->left && t->right)  
 {  
 temp = findMin(t->right);  
 t->month = temp->month;  
 t->day = temp->day;  
 t->year = temp->year;  
 t->right = remove(t->month,t->day,t->year, t->right);  
 }  
 else  
 {  
 temp = t;  
 if(t->left == NULL)  
 t = t->right;  
 else if(t->right == NULL)  
 t = t->left;  
 delete temp;  
  
 cout << " The Date entered has been deleted." << endl;  
 }  
 }  
 }  
 }  
 else if(y > t->year)  
 t->right = insert(m,d,y, t->right);  
  
 return t;  
}  
  
void Tree:: display(Date\* t)  
{  
 if(t == NULL)  
 return;  
 display(t->left);  
  
 cout << t->month << "/"<< t->day << "/" << t->year << endl;  
  
 display(t->right);  
}  
  
bool Tree:: find(int m, int d, int y, Date\* t)  
{  
  
 if(t == NULL)  
 return false;  
 else if(y < t->year)  
 return find(m,d,y, t->left);  
 else if (y == t->year)  
 {  
  
 if(m < t->month)  
 return find(m,d,y, t->left);  
 else if (m > t->month)  
 return find(m,d,y, t->right);  
 else if (m == t->month)  
 {  
 if(d < t->day)  
 return find(m,d,y, t->left);  
 else if (d > t->day)  
 return find(m,d,y, t->right);  
 else  
 return true;  
 }  
 }  
 else if(y > t->year)  
 return find(m,d,y, t->right);  
 else  
 return false;  
}  
  
Tree:: Tree()  
{  
 root = NULL;  
}  
  
void Tree::display()  
{  
 cout <<"Full Binary Tree: " << endl;  
 display(root);  
 cout << endl << endl;  
}  
  
void Tree:: insert()  
{  
 int m, d, y;  
 cout <<"Enter the month: ";  
 cin >> m;  
 cout <<"Enter the day: ";  
 cin >> d;  
 cout <<"Enter the year: ";  
 cin >> y;  
  
 cout << endl;  
  
 root = insert(m, d, y, root);  
 cout << "The date has been inserted." << endl << endl;  
}  
  
void Tree::remove()  
{  
 int m, d, y;  
 cout <<"Enter the month: ";  
 cin >> m;  
 cout <<"Enter the day: ";  
 cin >> d;  
 cout <<"Enter the year: ";  
 cin >> y;  
  
 cout << endl;  
  
 bool result = find(m, d, y, root);  
 if (result == false)  
 cout << "The Date has not been found in Binary Tree. Therefore no deletion available.";  
 else  
 root = remove(m, d, y, root);  
  
 cout << endl << endl;  
}  
  
void Tree::search()  
{  
 int m, d, y;  
 cout <<"Enter the month: ";  
 cin >> m;  
 cout << "Enter the day: ";  
 cin >> d;  
 cout <<"Enter the year: ";  
 cin >> y;  
  
 cout << endl;  
  
 bool result = find(m, d, y, root);  
 if (result == true)  
 cout << "The Date searched was found in the Binary Tree. " << endl;  
 else  
 cout << "The Date was not found in the Binary Tree. " << endl << endl;  
}  
  
int main()  
{  
 Tree t;  
 int selection;  
 do  
 {  
 cout << endl << "-------------------------------------------------" << endl;  
 cout << " Menu for Binary Tree " << endl << endl;  
 cout << "1. Display all Dates in Binary Tree " << endl;  
 cout << "2. Insert a Date " << endl;  
 cout << "3. Remove a Date " << endl;  
 cout << "4. Search for a Date " << endl;  
 cout << "5. Quit " << endl;  
 cout << "Enter your selection: ";  
 cin >> selection;  
  
 cout << endl;  
  
 if(selection == 1)  
 {  
 t.display();  
 }  
 else if (selection == 2)  
 {  
 t.insert();  
 }  
 else if (selection == 3)  
 {  
 t.remove();  
 }  
 else if (selection == 4)  
 {  
 t.search();  
 }  
 else if (selection == 5)  
 {  
 cout << "Program is exiting... ";  
 }  
 else  
 cout << "Error: Invalid menu selection.";  
  
 }while(selection != 5);  
  
 return 0;  
}

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 2

Enter the month: 11

Enter the day: 8

Enter the year: 1998

The date has been inserted.

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 2

Enter the month: 3

Enter the day: 8

Enter the year: 1971

The date has been inserted.

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 1

Full Binary Tree:

3/8/1971

11/8/1998

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 3

Enter the month: 11

Enter the day: 8

Enter the year: 1998

The Date entered has been deleted.

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 1

Full Binary Tree:

3/8/1971

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 4

Enter the month: 3

Enter the day: 8

Enter the year: 1971

The Date searched was found in the Binary Tree.

-------------------------------------------------

Menu for Binary Tree

1. Display all Dates in Binary Tree

2. Insert a Date

3. Remove a Date

4. Search for a Date

5. Quit

Enter your selection: 5

Program is exiting...

Process finished with exit code 0

Q2：

Merge Sort:

Procedure mergesort(array,N)

Array – list of elements to be sorted

N – number of elements in the list

Begin

If (N == 1) return array

Var array1 as array = a[0] … a[N/2]

Var array2 as array = a[N/2+1] … a[N]

Array1 = mergesort(array1)

Array2 = mergesort(array2)

Return merge( array1, array2)

End procedure

Procedure merge(array1, array2)

Array1 – first array

Array2 – second array

Begin

Var c as array

While ( a and b have elements )

If ( array[0] > array2[0] )

Add array2[0] to the end of c

Remove array2[0] from array2

Else

Add array1[0] to the end of c

Remove array1[0] from array1

End if

End while

While (a has elements)

Add a[0] to the end of c

Remove a[0] from a

End while

While (b has elements)

Add b[0] to the end of c

Remove b[0] from b

End while

Return c

End procedure

Quicksort:

Procedure quicksort(arr[], low, high)

Arr = list to be sorted

Low – first element of array

High – last element of array

Begin

If (low < high)

{

pivot = partition(arr, low, high);

quickSort(arr, low, pivot – 1);

quicksort(arr, pivot + 1, high);

}

End procedure

Than pivot are at the higher side of the array

Procedure partition (arr[], low, high)

Begin

Pivot = arr[high];

I = (low – 1)

For j = low to high

{

If (arr[i] <= pivot)

{

I++;

Swap arr[i] and arr[j]

}

}

Swap arr[I + 1] and arr[high]

Return (I + 1)

End procedure

Q3:

#include <iostream>

using namespace std;

int binarySearch(int array[], int x, int low, int high) {

if (high >= low) {

int mid = low + (high - low) / 2;

// If found at mid, then return it

if (array[mid] == x)

return mid;

// Search the left half

if (array[mid] > x)

return binarySearch(array, x, low, mid - 1);

// Search the right half

return binarySearch(array, x, mid + 1, high);

}

return -1;

}

int main(void) {

int array[] = {3, 4, 5, 6, 7, 8, 9};

int x = 4;

int n = sizeof(array) / sizeof(array[0]);

int result = binarySearch(array, x, 0, n - 1);

if (result == -1)

printf("Not found");

else

printf("Element is found at index %d", result);

}

-----

Element is found at index 1%