Eric Blasko

CSE 330 Data Structures

Winter 2018

HW 3 – problem 3.4 and 3.5

* **Source Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* hw3.cpp

\* 02/28/2018

\* This program finds the union and intersect of two sorted list. The list used are from the

\* standard library while the functions to find the union and intersect are provided above main.

\* Program will print the contents of both list, then print their union and intersect after

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <list>

using namespace std;

//This function uses const\_iterators to go through both list. Each value is compared and pushed onto

//a temp list. If one list reaches its end, a second loop will finish pushing remaining values to temp

template <typename T>

list<T> getUnion(const list<T> & a, const list<T> & b)

{

list<T> temp;

typename list<T>::const\_iterator i = a.begin();

typename list<T>::const\_iterator j = b.begin();

//done until one iterator reaches the end. Continue makes sure condition is checked after each

//if statement is completed

while(i != a.end() && j != b.end())

{

if(\*i == \*j)

{

temp.push\_back(\*i);

i++;

j++;

continue;

}

if(\*i < \*j)

{

temp.push\_back(\*i);

i++;

continue;

}

if(\*j < \*i)

{

temp.push\_back(\*j);

j++;

continue;

}

}

//if one iterator didnt reach end, finsh getting its values. Other will not loop as its at the end

while(i != a.end())

{

temp.push\_back(\*i);

i++;

}

while(j != b.end())

{

temp.push\_back(\*j);

j++;

}

return temp;

};

//The function loops using interators to find equal values in two list. Value that are equal are pushed

//onto a temp and returned at the end

template <typename T>

list<T> my\_intersect(const list<T> & a, const list<T> & b)

{

list<T> temp;

typename list<T>::const\_iterator i = a.begin();

typename list<T>::const\_iterator j = b.begin();

while(i != a.end())

{

for(j = b.begin(); j != b.end(); j++)

{

if(\*i == \*j)

{

temp.push\_back(\*i);

break;

}

}

i++;

}

return temp;

};

//Main function. Will push values onto two list and print there values. my\_union and my\_intersect will

//run after and print there values

int main()

{

list<int> L1;

list<int> L2;

list<int> myUnion;

list<int> myIntersect;

L1.push\_back(15);

L1.push\_back(20);

L1.push\_back(25);

L1.push\_back(30);

L1.push\_back(35);

L1.push\_back(40);

L2.push\_back(5);

L2.push\_back(10);

L2.push\_back(11);

L2.push\_back(25);

L2.push\_back(33);

L2.push\_back(40);

cout << "L1" << endl;

for(list<int>::iterator it = L1.begin(); it != L1.end(); it++)

cout << \*it << " ";

cout << endl;

cout << "L2" << endl;

for(list<int>::iterator it2 = L2.begin(); it2 != L2.end(); it2++)

cout << \*it2 << " ";

cout << endl;

myUnion = getUnion(L1,L2);

cout << "Union" << endl;

for(list<int>::iterator it3 = myUnion.begin(); it3 != myUnion.end(); it3++)

cout << \*it3 << " ";

cout << endl;

myIntersect = my\_intersect(L1,L2);

cout << "Intersect" << endl;

for(list<int>::iterator it4 = myIntersect.begin(); it4 != myIntersect.end(); it4++)

cout << \*it4 << " ";

cout << endl;

}

* **Sample Run**

Script started on 2018-03-03 15:53:14-0800

]0;005670557@csusb.edu@csevnc:~/cse330/hw3[005670557@csusb.edu@csevnc hw3]$ g++ -c hw2[K3.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/hw3[005670557@csusb.edu@csevnc hw3]$ g++ hw3.o

]0;005670557@csusb.edu@csevnc:~/cse330/hw3[005670557@csusb.edu@csevnc hw3]$ ./a.out

L1

15 20 25 30 35 40

L2

5 10 11 25 33 40

Union

5 10 11 15 20 25 30 33 35 40

Intersect

25 40

]0;005670557@csusb.edu@csevnc:~/cse330/hw3[005670557@csusb.edu@csevnc hw3]$ exit

Script done on 2018-03-03 15:53:33-0800

* **Time Complexity**

my\_union() runs through two list of size n. At worst case there are no value that are equal, thus the algorithm runs through 2 lists of size n. O(2n) which simplifies to O(n).

my\_intersect() runs through two list of size n. Each element of the first list is compared to every element in the second list. Thus, each iteration of n is compared n times, which gives O(n²)