CSCE 221 Cover Page

First Name Dilanka Last Name Weerasinghe UIN 126007816

User Name dweerasinghe E-mail address dweerasinghe@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources	
People	
Web pages (provide URL)	https://www.youtube.com/watch?v=GfRQvf7MB3k
Printed material	
Other Sources	https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/analysi

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

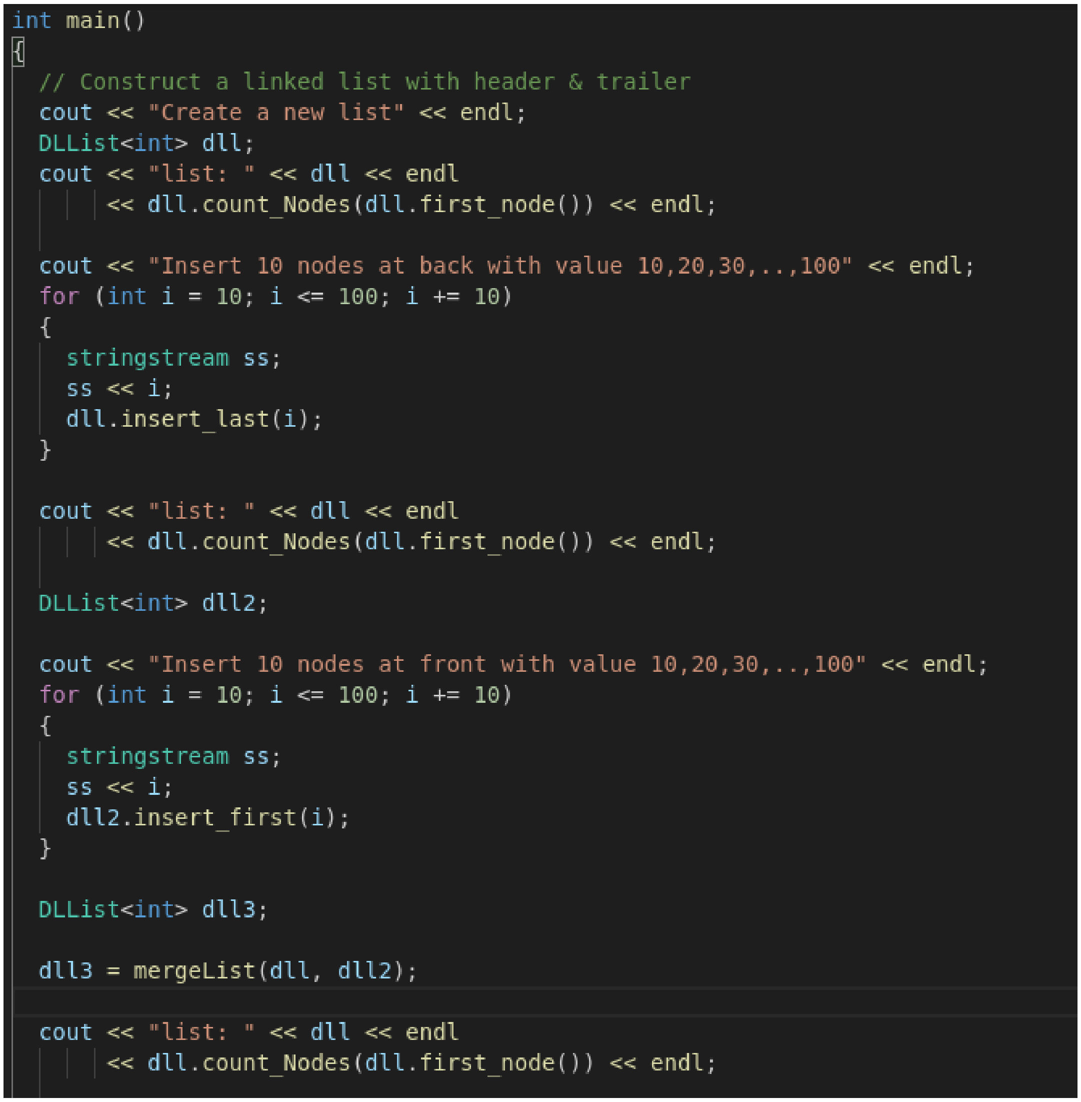
Your Name Dilanka Weerasinghe Date 16/3/20

Homework 2

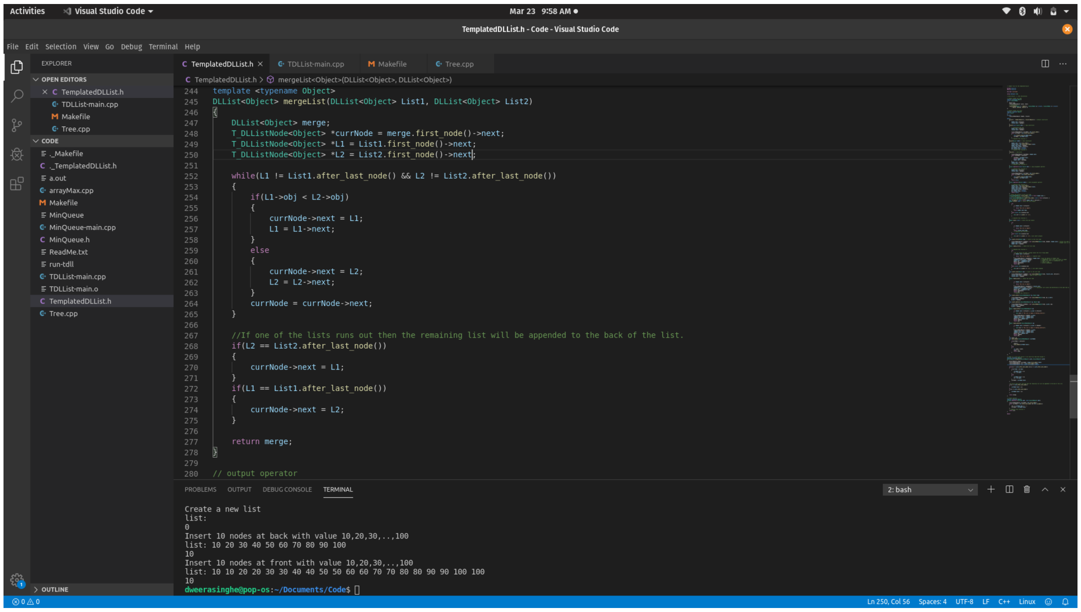
due March 16 at 11:59 pm to eCampus

1. (20 points) Given two sorted lists, L1 and L2, write an efficient C++ code to compute L1 $\bigcap$ L2 using only the basic STL list operations.

   (a) Provide evidence of testing: submit your code

      i. 

```
int main()
{
  // Construct a linked list with header & trailer
  cout << "Create a new list" << endl;
  DLList<int> dll;
  cout << "list: " << dll << endl
       << dll.count_Nodes(dll.first_node()) << endl;

  cout << "Insert 10 nodes at back with value 10,20,30,..,100" << endl;
  for (int i = 10; i <= 100; i += 10)
  {
    stringstream ss;
    ss << i;
    dll.insert_last(i);
  }

  cout << "list: " << dll << endl
       << dll.count_Nodes(dll.first_node()) << endl;

  DLList<int> dll2;

  cout << "Insert 10 nodes at front with value 10,20,30,..,100" << endl;
  for (int i = 10; i <= 100; i += 10)
  {
    stringstream ss;
    ss << i;
    dll2.insert_first(i);
  }

  DLList<int> dll3;

  dll3 = mergeList(dll, dll2);

  cout << "list: " << dll << endl
       << dll.count_Nodes(dll.first_node()) << endl;
```

ii. 

```cpp
template <typename Object>
DLList<Object> mergeList(DLList<Object> List1, DLList<Object> List2)
{
    DLList<Object> merge;
    T_DLListNode<Object> *currNode = merge.first_node()->next;
    T_DLListNode<Object> *L1 = List1.first_node()->next;
    T_DLListNode<Object> *L2 = List2.first_node()->next;

    while(L1 != List1.after_last_node() && L2 != List2.after_last_node())
    {
        if(L1->obj < L2->obj)
        {
            currNode->next = L1;
            L1 = L1->next;
        }
        else
        {
            currNode->next = L2;
            L2 = L2->next;
        }
        currNode = currNode->next;
    }

    //If one of the lists runs out then the remaining list will be appended to the back of the list.
    if(L2 == List2.after_last_node())
    {
        currNode->next = L1;
    }
    if(L1 == List1.after_last_node())
    {
        currNode->next = L2;
    }

    return merge;
}

// output operator
```

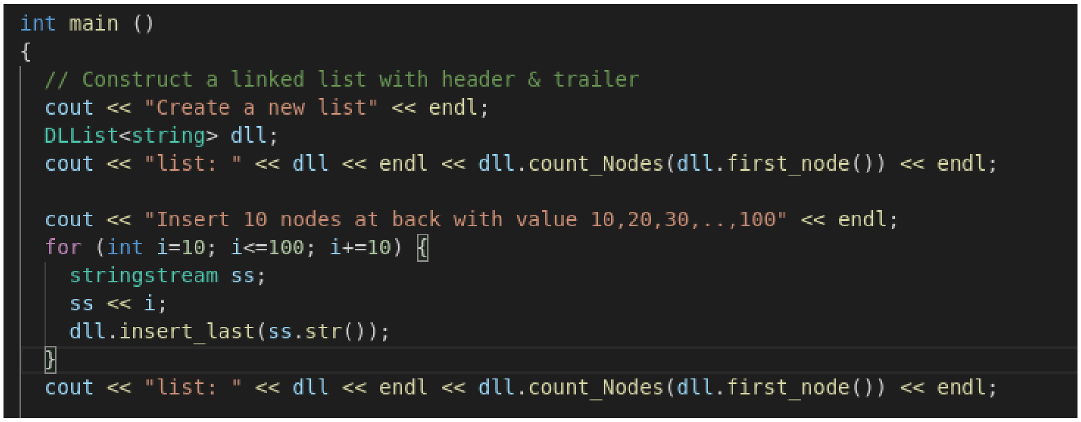
```
Create a new list
list:
0
Insert 10 nodes at back with value 10,20,30,..,100
list: 10 20 30 40 50 60 70 80 90 100
10
Insert 10 nodes at front with value 10,20,30,..,100
list: 10 10 20 20 30 30 40 40 50 50 60 60 70 70 80 80 90 90 100 100
10
dweerasinghe@pop-os:~/Documents/Code$
```

(b) What is the running time of your algorithm?

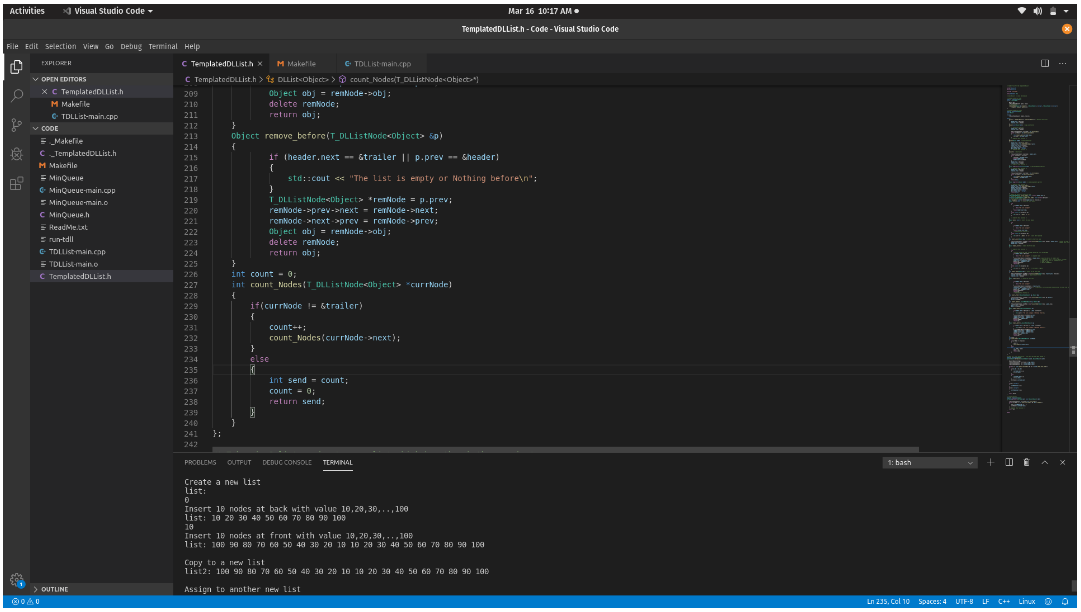
i. The running time is O(n + k) becuase there is only one move per node and there are two sets of nodes. n for the first list and k for the second list.

2. (20 points) Write a C++ recursive function that counts the number of nodes in a singly linked list.

(a) Test your function using different singly linked lists. Include your code.

i. 

```cpp
int main ()
{
  // Construct a linked list with header & trailer
  cout << "Create a new list" << endl;
  DLList<string> dll;
  cout << "list: " << dll << endl << dll.count_Nodes(dll.first_node()) << endl;

  cout << "Insert 10 nodes at back with value 10,20,30,..,100" << endl;
  for (int i=10; i<=100; i+=10) {
    stringstream ss;
    ss << i;
    dll.insert_last(ss.str());
  }
  cout << "list: " << dll << endl << dll.count_Nodes(dll.first_node()) << endl;
```

ii. 

```cpp
            Object obj = remNode->obj;
            delete remNode;
            return obj;
    }
    Object remove_before(T_DLListNode<Object> &p)
    {
            if (header.next == &trailer || p.prev == &header)
            {
                std::cout << "The list is empty or Nothing before\n";
            }
            T_DLListNode<Object> *remNode = p.prev;
            remNode->prev->next = remNode->next;
            remNode->next->prev = remNode->prev;
            Object obj = remNode->obj;
            delete remNode;
            return obj;
    }
    int count = 0;
    int count_Nodes(T_DLListNode<Object> *currNode)
    {
        if(currNode != &trailer)
        {
            count++;
            count_Nodes(currNode->next);
        }
        else
        {
            int send = count;
            count = 0;
            return send;
        }
    }
};
```

```
Create a new list
list:
0
Insert 10 nodes at back with value 10,20,30,..,100
list: 10 20 30 40 50 60 70 80 90 100
10
Insert 10 nodes at front with value 10,20,30,..,100
list: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Copy to a new list
list2: 100 90 80 70 60 50 40 30 20 10 10 20 30 40 50 60 70 80 90 100

Assign to another new list
```

(b) Write a recurrence relation that represents your algorithm.

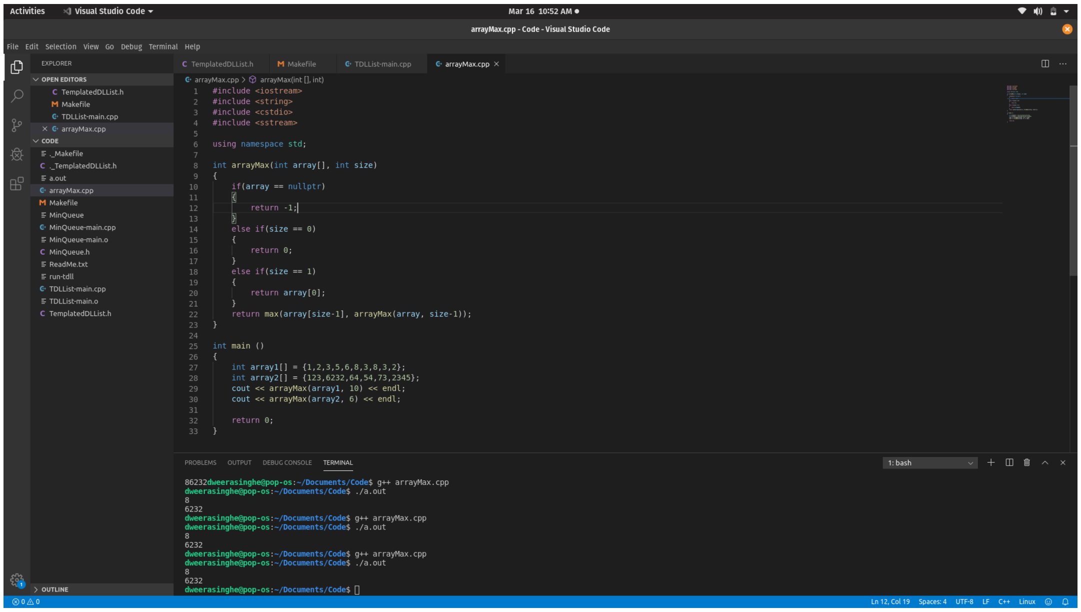
i. T(n) = T(n-1) + C

(c) Solve the recurrence relation using the iterating or recursive tree method to obtain the running time of the algorithm in Big-O notation.

i. T(n-1)
ii. T(n) = T(n-1) + C
iii. T(n-1) = T(n-2) + C
iv. T(n) = T(n-2) + C + C
v. T(n) = T(n-3) + 3C
vi. T(n) = T(n-x) + xC
vii. T(n) = T(n-n) + nC
viii. T(n) = O(n)

3. (20 points) Write a C++ recursive function that finds the maximum value in an array (or vector) of integers *without* using any loops.

(a) Test your function using different input arrays. Include the code.

i. 

```cpp
#include <iostream>
#include <string>
#include <cstdio>
#include <sstream>

using namespace std;

int arrayMax(int array[], int size)
{
    if(array == nullptr)
    {
        return -1;
    }
    else if(size == 0)
    {
        return 0;
    }
    else if(size == 1)
    {
        return array[0];
    }
    return max(array[size-1], arrayMax(array, size-1));
}

int main ()
{
    int array1[] = {1,2,3,5,6,8,3,8,3,2};
    int array2[] = {123,6232,64,54,73,2345};
    cout << arrayMax(array1, 10) << endl;
    cout << arrayMax(array2, 6) << endl;

    return 0;
}
```

```
86232dweerasinghe@pop-os:~/Documents/Code$ g++ arrayMax.cpp
dweerasinghe@pop-os:~/Documents/Code$ ./a.out
8
6232
dweerasinghe@pop-os:~/Documents/Code$ g++ arrayMax.cpp
dweerasinghe@pop-os:~/Documents/Code$ ./a.out
8
6232
dweerasinghe@pop-os:~/Documents/Code$ g++ arrayMax.cpp
dweerasinghe@pop-os:~/Documents/Code$ ./a.out
8
6232
dweerasinghe@pop-os:~/Documents/Code$ 
```

(b) Write a recurrence relation that represents your algorithm.

   i. T(n) = T(n-1) + C

(c) Solve the recurrence relation and obtain the running time of the algorithm in Big-O notation.

   i. T(n) = T(n-1) + C
   ii. T(n) = T(n-2) + 2C
   iii. O(n)

4. (20 points) What is the best, worst and average running time of quick sort algorithm?

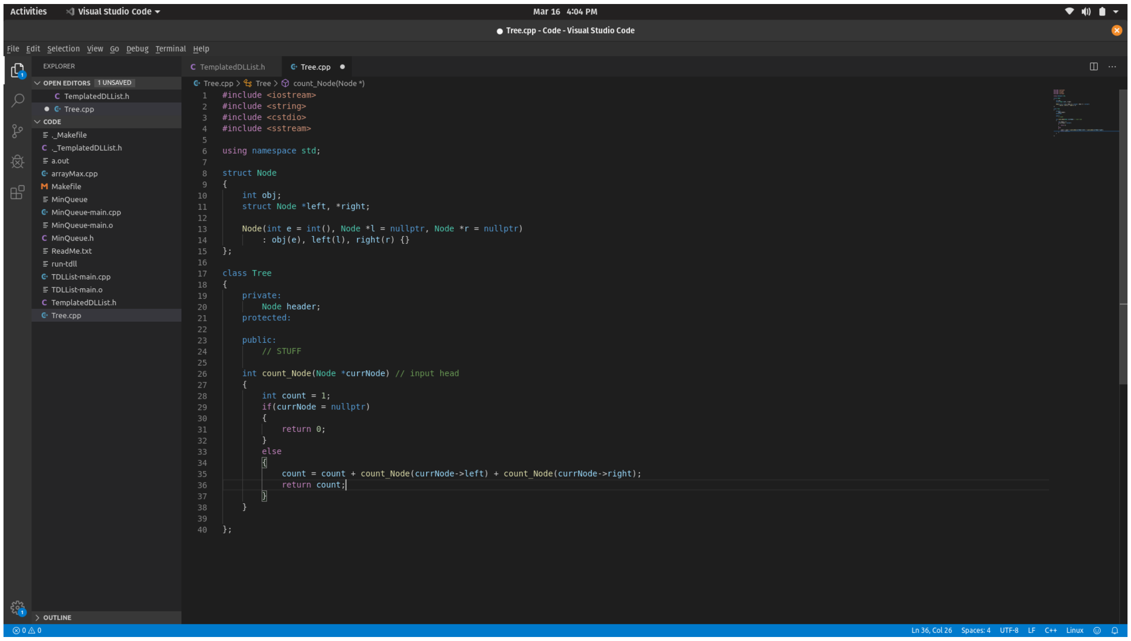
   (a) Provide recurrence relations and their solutions.

      i. Normal / Best
         A. T(n) = 2T(n/2) + n - 1
         B. T(n) = 2(2T(n/4) + n/2 - 1) + n - 1
         C. T(n) = 4(2T(n/8) + n/4 - 1) + 2n - 3
         D. T(n) = 8(2T(n/8) + n + n + n - 1 - 2 - 4
         E. T(1) = 0
         F. $2^k = n$
         G. $k = log_2 n$
         H. $T(n) = nlog_2 n - n + 1$
         I. $O(n) = nlog_2 n$
      ii. Worst
         A. $T(n) = T(n-1) + n - 1$
         B. $T(n) = T(n-2) + n - 1 + n - 2$
         C. $T(n) = T(n-3) + 3n - 1 - 2 - 3$
         D. $T(n) = T(1) + \sum_{i=0}^{n-1} n - i$
         E. $O(n) = n^2$

   (b) Provide arrangement of the input and the selection of the pivot point for each case.

      i. Selecting the pivot element as the max or the min every time for the worst time complexity, this time complexity is O(n)*constant. Therefore time complexity becomes O(n^2)
      ii. For best and worst case the pivot element is either random or split perfectly on either side. This will make sure that the split is good enough that the end result will be half and the time complexity will remain O(nlogn)

5. (20 points) Write a C++ function that counts the total number of nodes with two children in a binary tree (do not count nodes with one or none child). You can use a STL container if you need to use an additional data structure to solve this problem. Use the big-O notation to classify your algorithm. Include your code.

   (a) 

```cpp
#include <iostream>
#include <string>
#include <cstdio>
#include <sstream>

using namespace std;

struct Node
{
    int obj;
    struct Node *left, *right;

    Node(int e = int(), Node *l = nullptr, Node *r = nullptr)
        : obj(e), left(l), right(r) {}
};

class Tree
{
    private:
        Node header;
    protected:

    public:
        // STUFF

    int count_Node(Node *currNode) // input head
    {
        int count = 1;
        if(currNode = nullptr)
        {
            return 0;
        }
        else
        {
            count = count + count_Node(currNode->left) + count_Node(currNode->right);
            return count;
        }
    }

};
```

   (b) O(n) because it will count once for every node.